Máté József Csorba

Cost-Efficient Deployment of Distributed Software Services

Thesis for the degree of Philosophiae Doctor

Trondheim, May 2011

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Telematics

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

© Máté J. Csorba

ISBN 978-82-471-2781-0 (printed ver.)
ISBN 978-82-471-2782-7 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2011:122

Printed by NTNU-trykk

Abstract

New architectures and current technologies in software engineering enable the construction of systems with ever-increasing complexity. Distributed software systems today are most often composed of a number of building-blocks, operating in a networked environment and working together to achieve a variety of tasks. Software systems can cover several application domains, from smaller-scale dedicated or embedded systems (e.g. a security and video-surveillance system or a scientific simulator) to multi-user large-scale applications (e.g. web-based stores, or enterprize class data management systems). These systems are designed to be functionally correct according to some specification and at the same time to satisfy requirements related to the Quality of Service (QoS) they provide. This provided QoS depends on many parameters, such as properties of the network, the underlying execution hardware and system configuration. Mainly, two categories of properties can be considered: (i) capabilities, such as available bandwidth, processing power, storage and memory capacities, etc., and (ii) QoS requirements, such as volume of transactions, availability of hosts, amount of processing power, among others.

The topic of this thesis is *how* to decide *where* to allocate instances of software in a network of compute resources, under a variety of resource constraints and QoS requirements. The allocation of components in a distributed system, in other words its deployment mapping, can have a significant impact on the QoS provided by the system. Often, there are numerous deployment mappings that offer the same functionality, but nevertheless have very different QoS. In addition, some of the requirements the service has may be conflicting, so that improving the deployment with respect to one requirement might degrade the solution from another perspective. Hence, the problem of finding an optimal deployment mapping has many challenges and can be viewed from many angles. Obtaining a solution is further complicated when there is extensive dynamism and scale involved.

With the appearance of complex software systems the need for methods and algorithms that enable reconfiguration and adaptation has arisen. Self-adaptation has been an inherent property of several complex systems in nature that have been described theoretically and explored extensively. Thus, this thesis investigates how a bio-inspired method can be applied to the problem of obtaining deployment mappings for software services, and what are the benefits and tradeoffs of its application. The thesis presents a novel heuristic method for decentralized optimization aimed at finding near optimal mappings within reasonable time and for large scale.

Different incarnations of the deployment problem are explored throughout the papers included and several representative scenarios are investigated using simulations. For one of the scenario types means for obtaining global optimum solutions are provided and the results are used for cross-validating the simulations and to show that the heuristic algorithm presented in this thesis is able to provide effective means for solving deployment problems.

Furthermore, a formal approach is introduced to model services and deployment scenarios with different sets of requirements and optimization algorithms are given for the various scenarios. The approximative, computationally efficient, decentralized approach presented is believed to be adequate for on-line execution and can be tailored to given sets of requirements.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of philosophiae doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work was performed at the Department of Telematics, during the period 2007–2011, and has been supervised by Assoc. Prof. Poul E. Heegaard and Prof. Peter Herrmann.

First of all, I would like to thank my supervisors Poul E. Heegaard and Peter Herrmann for the fruitful discussions and their invaluable support during the ups and downs of my research. Furthermore, I would like to thank Assoc. Prof. Hein Meling at the University of Stavanger for the collaboration we had and his contribution to a significant amount of my work and publications. For initially helping kick start my research in the direction, which resulted in this thesis, and for his further constructive comments thanks to Prof. Bjarne E. Helvik at the Q2S Centre at NTNU. Moreover, for helping me with linear programming in the context of my research topic thanks to Prof. Di Yuan at Linköping University.

It has been a pleasure to teach at the department and work with Prof. Steinar H. Andresen and later with Prof. Lill Kristiansen. I appreciate the advice and the kind help in settling in, which I got from Adj. Assoc. Prof. Mazen M. Shiaa. Besides, I also would like to thank my colleagues for providing me with a professional and inspiring environment. It has been very efficient, yet fun to work at ITEM with you Andrés, Astrid, Elissar, Frank, Hien, Humberto, Jing, Laurent, Linda, Martin, Mona, Nor, Pål, and Randi, among others. Special thanks to Vidar, whom I shared office with at ITEM, for teaching me *nynorsk* on a daily basis. Furthermore, thanks to my former colleagues Árpád, Dániel, Gergely, Kristóf, and Vilmos, whom I worked and studied together with at Ericsson Hungary and at the Budapest University of Technology and Economics. Especially, thanks to Sándor Palugyai for the cooperation on the articles we co-authored. Also, I would like to acknowledge the help I got from János Miskolczi and Assoc. Prof. Gyula Csopaki in Budapest, who persuaded me to start working with research.

Last but not least, I would like to thank my family, my wonderful wife, Linda, and my parents (*Anyu* & *Apu*) for their love, patience and enormous support during all these years I spent completing this thesis.

Contents

List of Papers

Publications Included in the Thesis

A list of the papers included in Part II of this thesis. Some of the papers have undergone minor changes in order to fit the formatting of this book. The last paper, included in the Appendix, is a summary paper accepted for publication at the time of writing.

- PAPER A:
  Máté J. Csorba, Poul E. Heegaard, Peter Herrmann. *Cost Efficient Deployment of Collaborating Components*. 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'08). June 4–6, 2008, Oslo, Norway.

- PAPER B:
  Máté J. Csorba, Poul E. Heegaard, Peter Herrmann. *Adaptable Model-based Component Deployment Guided by Artificial Ants*. 2nd International Conference on Autonomic Computing and Communication Systems (Autonomics 2008). September 23–25, 2008, Turin, Italy.

- PAPER C:
  Máté J. Csorba, Poul E. Heegaard, Peter Herrmann. *Component Deployment Using Parallel Ant-nests*. International Journal on Autonomous and Adaptive Communications Systems (IJAACS). ISSN (Online): 1754-8640. ISSN (Print): 1754-8632. InderScience publishers.

- PAPER D:
  Máté J. Csorba, Hein Meling, Poul E. Heegaard, Peter Herrmann. *Foraging for better deployment of replicated service components*. 9th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'09). June 9–12, 2009, Lisbon, Portugal.

- PAPER E:
  Máté J. Csorba, Hein Meling, Poul E. Heegaard. *Laying Pheromone Trails for Balanced and Dependable Component Mappings*. 4th IFIP TC 6 International Workshop on Self-Organizing Systems (IWSOS'09). December 9–11, 2009, Zurich, Switzerland.

- PAPER F:
  Máté J. Csorba, Hein Meling, Poul E. Heegaard. *Ant system for service deploy-*

ment in private and public clouds. 2nd Workshop on Bio-Inspired Algorithms for Distributed Systems (BADS'10). June 7–11, 2010, Washington, DC, USA.

- PAPER G:
  Máté J. Csorba, Poul E. Heegaard. *Swarm Intelligence Heuristics for Component Deployment.* 16th EUNICE/IFIP WG 6.6 Workshop (EUNICE'10). June 28–30, 2010, Trondheim, Norway.

- APPENDIX:
  Máté J. Csorba, Hein Meling, Poul E. Heegaard. *A Bio-inspired Method for Distributed Deployment of Services.* New Generation Computing. Computing Paradigms and Computational Intelligence. Ohmsha Ltd. and Springer-Verlag.

Other Papers by the Author

A list of authored or co-authored papers that presented research outside the scope of this thesis.

- Máté J. Csorba, Prajwalan Karanjit, Steinar H. Andresen. *"The SIP Pod" - a VoIP Student Lab/Playground.* Remote Instrumentation Services on the e-Infrastructure. ISBN: 978-1-4419-5574-6. Springer, 2011.

- Máté J. Csorba, Poul E. Heegaard. *Optimization and Service Deployment in Private and Public Clouds.* ERCIM News. No. 83, 2010.

- Máté J. Csorba, Dániel Eöttevényi and Sándor Palugyai. *Experimenting with Dynamic Test Component Deployment in TTCN-3.* 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities (TridentCom'07). May 21–23, 2007, Lake Buena Vista, USA.

- Sándor Palugyai, Máté J. Csorba. *State-space reduction in the model of Access Control Lists in IP routers.* International Journal of Internet Protocol Technology. Vol. 2, No. 2, 2007.

- Máté J. Csorba, Sándor Palugyai. *A Performance Model for Load Test Components.* 5th Conference of PhD Students in Computer Science (CSCS'06). June 27–30, 2006, Szeged, Hungary. *Excellent talk award*

- Máté J. Csorba, Sándor Palugyai. *Performance Analysis of Concurrent PCOs in TTCN-3.* 18th International Conference on Testing Communicating Systems (TESTCOM'06). May 16–18, 2006, New York City, NY, USA.

- Sándor Palugyai, Máté J. Csorba, Sarolta Dibuz, Gyula Csopaki. *Measurement and Optimization of Access Control Lists.* Acta Cybernetica. Vol. 17, No. 2, 2006.

- Máté J. Csorba, Sándor Palugyai. *Early Performance Assessment of Software Architectures in Telecommunication.* 3rd International Conference on Computer as a Tool (EUROCON'05). November 21–24, 2005, Belgrade, Serbia and Montenegro.

- Sándor Palugyai, Máté J. Csorba. *Modeling Access Control Lists with Discrete-Time Quasi Birth-Death Processes*. 20th International Symposium on Computer and Information Sciences (ISCIS'05). October 26–28, 2005, Istanbul, Turkey.
- Sándor Palugyai, Máté J. Csorba. *Performance Modeling of Rule-Based Architectures as Discrete-Time Quasi Birth-Death Processes*. 14th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN'05). September 18–21, 2005, Chania, Greece.
- Sándor Palugyai, Máté J. Csorba. *Modeling List Topologies with Markovian Theory*. 11th Open European Summer School, IFIP WG 6.6, WG 6.4 and WG 6.9 Workshop (EUNICE'05). July 6–8, 2005, Madrid, Spain.
- Máté J. Csorba, Sándor Palugyai. *Performance Evaluation of Telecommunications Systems Specified with Formal Description Techniques*. 3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurement (IPS-MoMe'05). March 14–15, 2005, Warsaw, Poland.
- Sándor Palugyai, Máté J. Csorba. *Real-time Optimization of Access Control Lists*. 4th Conference of PhD Students in Computer Science (CSCS'04). July 1–4, 2004, Szeged, Hungary.
- Máté J. Csorba, Sándor Palugyai. *Testing Network Traffic Analyzers with TTCN-3*. 10th Eunice Summer School and IFIP WG 6.3 Workshop (EUNICE'04). June 14–16, 2004, Tampere, Finland.
- Máté J. Csorba, Sándor Palugyai, János Miskolczi, Sarolta Dibuz. *Performance Measurement of Routers using Conformance Testing Methods*. 2nd International Workshop on Inter-Domain Performance and Simulation (IPS'04). March 22–23, 2004, Budapest, Hungary.
- Máté J. Csorba, Sándor Palugyai, János Miskolczi. *Testing Network Traffic Analyzers using Conformance Testing Tools*. Híradástechnika (in Hungarian). Vol. 59, No. 3, 2004.
- Máté J. Csorba, Sándor Palugyai, János Miskolczi. *Conformance Testing of the OSPF routing protocol*. Híradástechnika (in Hungarian). Vol. 57, No. 8, 2002.

Nomenclature

The table below summarizes the notational elements used in Part I. The sets denoted in **boldface** are constants, whereas the sets in *italic* are running variables. Note that this notation is adjusted to the overall introduction and might differ from the notations applied in the included papers.

Table 1: Notational shorthand

Notation	Usage	Size	Description
A_i			availability of an instance i
a_r, b_r			parameters for the Taylor expansion of the temperature
A_{mn}, B_{mn}, C_{mn}			parameters for the Taylor expansion of the pheromones
β			the memory factor
$\mathbf{C}$	$c_i \in \mathbf{C}$	$\lvert\mathbf{C}\rvert$	set of components in a service S_l
$C_{\{0\|1\}}()$			a quadratic function to evaluate node-local costs
$\mathbf{D}$	$d_i \in \mathbf{D}$	$\lvert\mathbf{D}\rvert$	set of all existing network clusters
D	$d_i \in D$	$\lvert D\rvert \subseteq \lvert\mathbf{D}\rvert$	list of clusters used in mapping M
$F_{\{1\|2\|3\|4\}}()$			a function to evaluate the cost of a given deployment
$f_{c_i}^{(e)}$			execution cost of an instance
$f_{k_j}^{(c)}$			communication cost of a collaboration
$f_{n_i}^{(f)}$			financial cost of using a node
$F_F(M)$			financial cost of using nodes covered in M
$F_K(M)$			communication cost of mapping M
γ_r			temperature in iteration r
H	$n_i \in H$	$\lvert H\rvert \subseteq \lvert\mathbf{N}\rvert$	hop-list, i.e. nodes visited in iteration r
$H()$			an (exponential) performance function in CEAS
$h_r()$			autoregressive formulation of $H()$
$I()$			an indicator function
$\mathbf{K}$	$k_j \in \mathbf{K}$	$\lvert\mathbf{K}\rvert$	set of collaborations in a service S_l
L	$\hat{l}_{n,r} \in L$	$\lvert L\rvert \subseteq \lvert\mathbf{N}\rvert$	set of load samples
M_r	$m_{n,r} \in M_r$	$\lvert M\rvert = \lvert\mathbf{C}\rvert$	mapping $\mathbf{C} \to \mathbf{N}$ for a service in iteration r
$\mathbf{N}$	$n_i \in \mathbf{N}$	$\lvert\mathbf{N}\rvert$	set of all existing nodes
ω			scaling between F_K and execution costs
$p_{mn,r}$			random proportional rule for mapping m at node n
Φ	$\phi_1 \in \Phi$		set of dependability rules
$q_{\{0\|1\}}()$			helper functions returning a node n or a cluster d
r			iteration counter
ρ			the search focus
$\mathbf{S}$	$S_l \in \mathbf{S}$	$\lvert\mathbf{S}\rvert$	set of services to deploy
T			global load-balance estimate
$\tau_{mn,r}$			pheromone value for mapping m at node n in iteration r
$\vartheta()$			processing resource demand
w			replica execution cost

Part I

THESIS INTRODUCTION

Introduction

Distributed and pervasive software systems can typically be characterized by dynamic configurations and operational conditions that are unknown prior to their deployment. Traditional desktop software systems are relatively stable and static during their operation. Popular multi-user web applications and similar services are currently being deployed in large-scale data-center environments due to their inherent scaling needs. Data-center environments may consist of hundreds of thousands of nodes, and at this scale, the topology is typically in constant change. Changes in execution context might be due to scheduled maintenance, failures, or varying usage patterns of the different applications deployed. Accounting for all the parameters involved in these types of complex systems is a challenging undertaking. Flexible methods and models are necessary to maintain the desired QoS at acceptable costs, where the term desired QoS is interpreted as *the delivery of a service in accordance with its specification* [PJE09]. As it is often difficult to derive accurate estimations regarding the execution context of distributed software services at design-time, support for run-time adaptation is needed. The main contributions of this work can be outlined as follows.

Finding computationally effective **cost functions**, that is functions that express the utility of deployment mappings of a service. In order to capture the cost of a deployment, first a high-level functional modeling approach is found. The models consist of distributed, collaborating components and can be embellished with non-functional information, which serve as input for the evaluation of deployments. Different functions are presented for different requirements and deployment scenarios.

Using the cost functions derived, **a bio-inspired, decentralized optimization approach is applied** (previously applied to distributed path-finding in networks). The heuristic method is aimed at finding (close to) optimal mappings even under instable network conditions. Several incarnations of the deployment problem are investigated through simulations. Algorithms based on the bio-inspired method, called the Cross Entropy Ant System (CEAS), are devised and fine-tuned for solving the deployment problem.

The deployment approach is **cross-validated** against a traditional centralized technique that obtains the global optimum in one of the deployment scenarios studied. It is shown that the solutions obtained by simulating the proposed deployment logic are close to optimal, with low variance. Traditional centralized optimization methods fall short as the problem size grows, whereas the heuristic algorithms proposed can scale more conveniently to larger problem sizes. Moreover, due to the inherent dynamism that exist in the systems considered, a solution might quickly become suboptimal. The suggested heuristic approach, however, is able to rapidly adapt to changes in the environment.

This thesis consists of two main parts. The first part is intended to give the reader a short but comprehensive introduction to the research work conducted by the author in connection with this thesis. Guidelines for reading and a short summary of the publications included in the collection in Part II is given below.

The organization of the *Thesis Introduction* is as follows. Section 1 presents the background of the problem setting. The problem of obtaining deployment mappings between various building-blocks of software services and execution hosts in a network is introduced, followed by a discussion on the complexity, constraints and other aspects of the problem at hand.

The research methodology is presented in Section 2, which is a section discussing the key points of this work, including the importance of cost functions and how deployment costs are quantified, the swarm intelligence based optimization framework that is applied extensively in the thesis. Fine-tuning and extensions of the optimization approach and deployment strategies are discussed. Issues of scalability, convergence, overhead and validation of the results are also summarized in this section.

A summary of the contributions is given in Section 3 presenting how the included articles are interrelated and giving guidelines for reading.

Section 4 reviews related work, before a discussion in Section 5. The first part of the book ends with Section 6, where some of the work ongoing during the time of writing and possible future directions are outlined.

1. Research focus and background

Current software applications and services predominantly exist in a highly distributed and dynamic environment, where the underlying hardware systems might change their topology as well as their capacity and availability frequently. Another dimension of dynamicity arises from the varying usage patterns of a high volume of users software systems are designed to provide services for. Maintaining the desired QoS at reasonable costs requires efficient and flexible methods. The major focus of research in this thesis is on the deployment of software services. The work explores the possible advantages achievable via effecting changes in the software architecture and the related trade-offs that have to be taken into consideration. The relevant constraints and the complexity of the deployment problem are touched upon in this section, followed by a short discussion on the target platforms that can be considered as execution framework for the services this thesis is relevant for.

Furthermore, a set of requirements for a deployment logic is defined below that will be elaborated and will be referred to in the appropriate sections of this introduction. A deployment logic shall satisfy the requirements of:

- REQ-1. – *Efficiency*

  Being efficient means that a solution to the deployment problem is found that satisfies all the requirements of the services at hand. The obtained solution might not be the absolute optimum (one of the optima in case there are many), but it is obtained fast, i.e. within a reasonable amount of iterations. Also, the amount of iterations and the additional overhead required by the method to converge to a new solution after a change in the execution context has to be as low as possible.

- REQ-2. – *Robustness*

  Furthermore, the logic has to be able to operate in a dependable way if executed in an on-line environment. By developing a decentralized mechanism the need for centralized storage and decision making is avoided. Thus, the approach is free from a possible bottleneck and single point of failure.

- REQ-3. – *Autonomicity*

  Autonomicity is required to realize self-management, i.e. to collect information, to analyze it, and then to act based upon it autonomously.

- REQ-4. – *Adaptivity*

  Moreover, the dynamic environment in which software services are typically deployed into necessitate the capability of adaptation to be able to adapt deployments to run-time changes in the execution context.

- REQ-5. – *Scalability*

  Scalability of the deployment logic has to be considered with respect to overhead, resource usage and convergence time of the method. Also, it is important that as the problem size grows the additional burden on the method grows in a controlled manner. In many cases, if the search space for a deployment is very large, obtaining the optimum might require prohibitively long time. By that time, the execution context might even change due to the dynamic nature of the physical environment of the applications. To allow scaling to relevant problem sizes, the logic requires efficient data representation for the search method, and fast execution of the algorithms crawling through the data.

- REQ-6. – *Generality & Extensibility*

  Lastly, the method has to be general enough to allow for extensions with respect to problem constraints that can be considered. In other words, it has to be relatively easy to extend the logic to cater for new QoS dimensions for the services deployed.

Next, the general deployment problem is introduced.

1.1 The deployment problem

This thesis focuses on the problem of mapping building blocks of services to physical resources, i.e. execution hosts, so that the requirements the services have are satisfied. The deployment problem can be characterized as a multifaceted optimization problem, with multiple layers of complexity as illustrated by Figure 1. The overall goal of the deployment is to obtain an efficient mapping, $M : \mathbf{C} \rightarrow \mathbf{N}$ between the building-blocks (the set $\mathbf{C}$) of services ($\{S_1, S_2, \ldots\} \in \mathbf{S}$) and the available nodes ($\mathbf{N}$). A service S_l is defined as being provided by a set of components $\{c_1, c_2, \ldots\} \in \mathbf{C}$ that communicate with each other via a set of collaborations, $\{k_1, k_2, \ldots\} \in \mathbf{K}$. Hence, a collaboration $k_j = (k_{j,1}, k_{j,2})$ may exist between two components c_a and c_b, such that $k_{j,1} = c_a$ and $k_{j,2} = c_b$. This description assumes that all of the services can be mapped to the set of available resources. Generally, this is not the case due to compatibility problems and security restrictions.

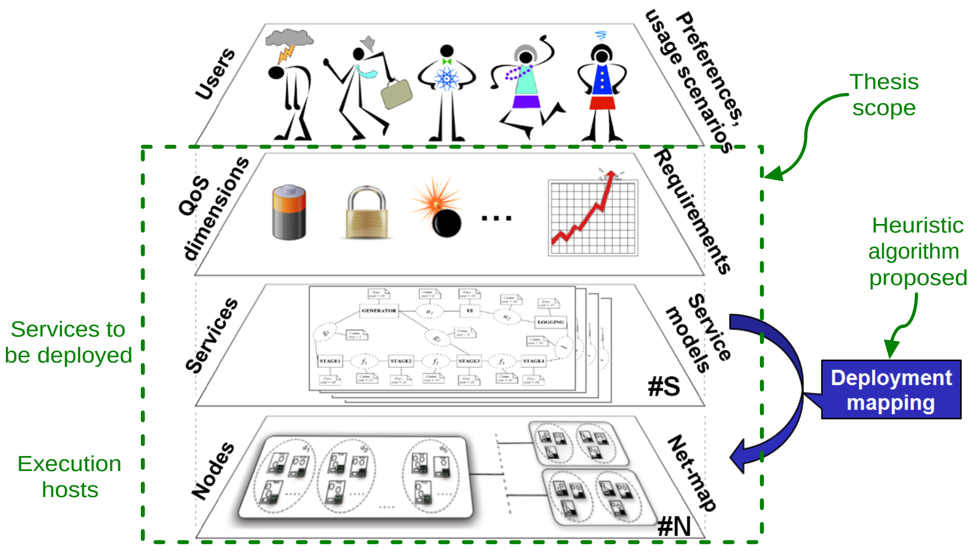


Figure 1: Multiple dimensions of the deployment problem (adopted from [Mal06])

The multiple layers show the execution hosts in the bottom and the services executed and deployed simultaneously on the second layer. QoS-related, or in other words non-functional (NF), requirements constitute the third layer of the problem, including aspects such as dependability, security, performance, energy-saving, etc. This layer contributes to an increased problem complexity as well, as the number of NF-requirements taken into account increases. The top layer includes the possible varieties of usage scenarios for the services. These usage scenarios can be captured in the models by enriching them with additional usage related information, such as for example arrival rates for the components that handle user requests. This fourth layer, however, is out of the scope of this thesis.

Relevant QoS requirements can be captured and incorporated into the service models at design time (REQ-6). In particular, to model services, collaboration oriented models are used, similar to UML 2.0 collaboration diagrams. The deployment logic is used to optimize the mappings and, at the same time, enable compatibility with existing frameworks and interfaces for deployment. A sample collaboration between two components is shown in layer two in Figure 2, which is a basic example used throughout the included papers to introduce service models. This example shows two software components (that have to be deployed) and a collaboration between them, each of which has a corresponding cost value. Components have costs related to their execution, e.g. memory or CPU share needed at the host, factored into a single value, whereas collaborations have costs that inform about the communication need between the components. Communication costs are composite values incorporating the volume of interaction between components, i.e. they are characterized by the amount of messages interchanged and the average message length.

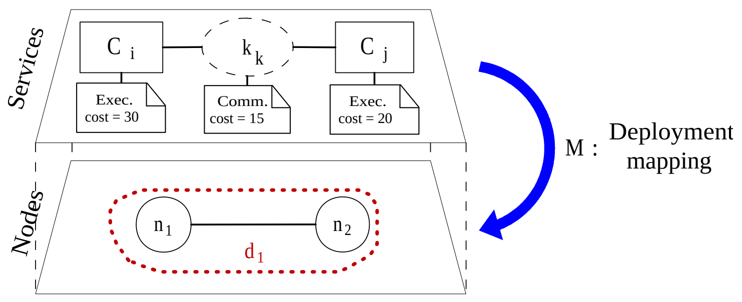


Figure 2: Example network and collaboration

The main notational elements used throughout Part I are summarized in Table 1. To demonstrate the notation with a very small example, consider the network and the example collaboration shown in Figure 2. In this case the target network consists of two nodes, $\mathbf{N} = \{n_1, n_2\}$, and there is only one cluster in the network, $\mathbf{D} = \{d_1\}$, which contains two nodes, $d_1 = \{n_1, n_2\}$. The set of services that have to be deployed consists of only one service $\mathbf{S} = \{S_1\}$, where S_1 is defined by the simple collaboration in the second layer in Figure 2. Hence, there are two components to be deployed $\mathbf{C} = \{c_i, c_j\}$ and one collaboration $\mathbf{K} = \{k_k\}$ between them. The variables, for example after 1 iteration of the deployment logic, can hold the following values. Clusters used $D_1 = \{d_1\}$, nodes visited $H_1 = \{n_1, n_2\}$, load samples taken from the nodes $L_1 = \{\hat{l}_{n_1,1}, \hat{l}_{n_2,1}\} = \{30 \rightarrow n_1, 20 \rightarrow n_2\}$, and the mapping M resulting from iteration 1 $M_1 = \{m_{n_1,1}, m_{n_2,1}\} = \{\{c_i \rightarrow n_1\}, \{c_j \rightarrow n_2\}\}$. $\Box$

1.2 Problem complexity and trade-offs

The problem of determining a new mapping, M, of instances, $\mathbf{C}$, to physical nodes, $\mathbf{N}$, while avoiding the violation of hard constraints was shown to be NP-hard [WSVY07]. Complexity can be verified by reducing the multidimensional bin packing problem [KLMS84] to this mapping problem by letting $\mathbf{N}$ represent the bins and $\mathbf{C}$ the objects to be packed. Objects may have specific needs for the available resource types and each constraint will span a dimension in this case. Even to determine if a valid packing exists is NP-hard in itself [WSVY07]. The problem of apportioning multiple finite resources to satisfy the QoS requirements of multiple applications along multiple QoS dimensions, labelled the MRMD problem has been investigated in [LLS$^+$99]. Complexity of the MRMD problem has been proven to be NP-hard by reducing the binary knapsack problem to one of its special cases. A more general version of the deployment problem, the general module allocation problem, with the exception of some special communication configurations, has been shown to be NP-complete in [FB89]. Introducing more requirement dimensions (REQ-6) provides additional complexity, for example when consistency protocols are added to improve dependability of the services. These results and considerations suggest that the problem formulation applied in this thesis, and presented in Section 1.1, results in an at least NP-hard problem. An additional cost factor is related to the overhead of actual physical placement of components (deployment) and thereafter their migration (re-deployment). This cost, however, is not considered in the following.

In addition to the complexity, the size of the deployment problem is often very large. Finding efficient mappings in realistic scenarios, such as in large-scale data center infrastructures the problem size becomes difficult to handle and finding exact solutions might be impossible. One of the major factors increasing the solution space is that a multitude of services have to be deployed simultaneously. Also, the physical network usually consists of a large number of nodes, often divided in many different administrative sites, where all the requirements have to be fulfilled even in deployments stretching across many sites. A large number of parameters has influence on the quality of the deployment. A number of services can be executed in parallel and hence, can compete for the same set of resources.

Further complicating the task of deployment is the dynamic and constantly changing context of software services, including user mobility, node churn, network clusters splitting and merging, incremental scaling needs, variations in accepted workloads, and more. To efficiently deal with these issues the capability for adaptation has to be inherent in a deployment method (REQ-4) as well as making decisions autonomously (REQ-3). Hence, the complexity of the problem and the size of the solution space in realistic scenarios suggests the application of heuristic algorithms instead of exhaustive search methods (considering REQ-1 and REQ-5). Self-organizing systems are known for exhibiting the required properties and are a good candidate for developing intelligent methods aiding service deployment.

In some cases the solution space can be reduced due to specific requirements. The simplest factor, which decreases the solution space is fixing the mapping of some instances, parts of a service to a given physical resource. Consider for example a service component that is responsible for operating a surveillance camera and thus it is fixed to the physical device itself. A fixed mapping cannot be changed during the search for an appropriate deployment, thus the deployment problem is cut back to the remaining components in the service, this is referred to as *binding* throughout the rest of this thesis. Binding generally reduces the search space. However, bound instances of the services still have to be taken into account when a given deployment mapping is evaluated and checked for compliance with the requirements. Service examples that contain bindings are presented in PAPER A, B, and C.

Runtime adaptation might enforce changing the placement of a set of instances of a service from the initial deployment mapping, conditioned by specific threshold values, characteristic to the cost of the required migration event. Several authors estimate the durations required for migrating for example operational VMs by conducting experiments. Durations vary, as expected, depending on the hardware context, e.g. bandwidth, and naturally on VM package size. However, for realistic sizes estimates lie typically around 60 to 90 seconds, see [CFH$^+$05, HON$^+$09, JHJ09]. Relocation of VM instances can be further sped up by compression techniques for complete machine states. An efficient compression approach, for example in [SPYH04], can provide a package size as low as 10% of the original memory consumption of a VM. Migration costs can be factored in as threshold values to allow changes in the deployment mappings only if the benefit is higher than the costs of migration. Runtime reconfiguration costs are further investigated in [HJS$^+$09, JJH$^+$09]. Besides, migration of service data is an additional challenge complicated by issues of shared data, inter-dependencies and user mobility. In case of many web-based applications with a very high number of users, standard commercial optimization tools simply do not scale to the problem size [ADJ$^+$10].

1.3 Target systems

The deployment logic is concerned with elementary building blocks of services. These elementary building blocks, referred to as components, are considered to be standalone, executable packages of software that have well-defined interfaces and can communicate via message exchange. Distributed components then collaborate and together provide a given service. Services in the deployment logic are looked upon from the service provider's perspective. However, a deployment that is deemed efficient (REQ-1) considers the general QoS perceived by the users of the services as that is mostly what generates revenue for the providers. There are many ways of improving QoS by influencing the software architecture, and in particular via changing the deployment of components, which is the focus of this work. A simple example is considering the latency of a service, which can be decreased effectively by identifying and deploying components that require voluminous interactions to the same host, which we refer to as collocation, or, alternatively deploy them to nodes that are connected by capacious links. If, however, the deployment logic has to consider general load-balancing at the same time the problem becomes significantly more difficult to solve.

The execution platform of services is possibly also a highly distributed hardware environment consisting of nodes heterogeneous in capabilities, amounts of resources and in access rights. This network of possible execution hosts is considered to be a hybrid environment, in which services can be deployed in various clusters or clouds, depending on the present conditions, and usage patterns. The utility of having several network clusters lies for example in the possibility of tackling peak load scenarios, or meeting dependability and performance requirements. Handling load-overshoots, for example, might dictate addition or removal of service instances. Execution in such dynamic and hybrid environments might be influenced by a plethora of various parameters making the search for an efficient deployment difficult. To model the dynamic environment a target network consists of nodes $n_i \in \mathbf{N}$, and the network itself can in addition be partitioned into clusters $\{d_1, d_2, \ldots\} \in \mathbf{D}$ as shown in Figure 3.

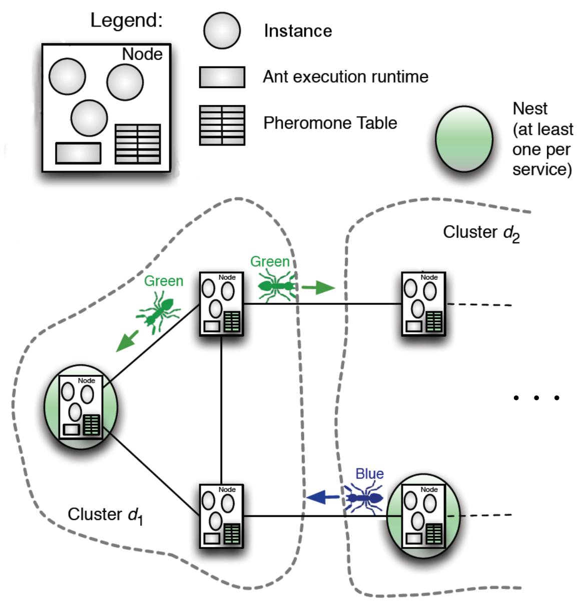


Figure 3: Example target network

Regarding the network it is assumed that all nodes are identical with respect to capacity and the network is fully interconnected. Furthermore, each node participating contains an *execution runtime* that encapsulates the functionalities of installation and execution of components. For every service that is deployed the deployment logic has to be run separately. This, however, can be done in parallel. Autonomous agents searching for a useful deployment for different services are depicted in different colors. Each separate type of agents, corresponding to separate services, has to have a designated node, a home-location called the *nest*. Each node has, beside the execution runtime, a dedicated array of memory available to the deployment logic only, called the *pheromone table*, and capacity for installing one or more components of arbitrary services. The *pheromone tables* contain information about component to node mappings for each service and each node (similarly to routing tables). The inner workings of the deployment logic are addressed in more detail in Section 2.

The concept of autonomic managers has gained much attention in autonomic computing as an entity that can automate management functionalities and provide standard interfaces externally. An autonomic manager implements an intelligent control loop introduced in the seminal paper [KC03]. The control loop, shown in Figure 4, is often referred to as MAPE for it's four main functions, **M**onitor, **A**nalyze, **P**lan, and

Execute. To realize self-management in a system some degrees of autonomicity has to be in place (REQ-3) to collect information about the system, analyze it, create a plan of necessary changes and finally to perform the required actions. Information used to facilitate autonomic management can be represented as knowledge that can be shared among distributed mangers working together for a common goal.

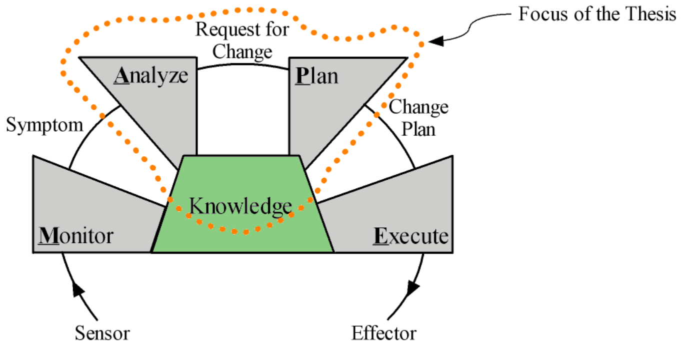


Figure 4: Focus of this thesis in the context of autonomic management (adopted from [KC03])

An autonomic manager featuring the MAPE loop and applied for efficient deployment of services would benefit from the contributions of this thesis in the analyzes, planning and knowledge part as shown in Figure 4 indicated by the dashed line. The deployment logic presented herein requires input from a monitoring unit describing the environment, and another unit, for execution, that can effect the placement suggested by the logic. These two features of the MAPE autonomic manager are not discussed in this thesis and are assumed to be in place. When a proper description of the environment and the current deployment is given, running the deployment logic will analyze the situation and find a suggested mapping that serves as a plan for change. This change can then be effected, in the form of a new placement, by the executor part of the cycle. Knowledge gathered by an autonomic manager, i.e. one instance of CEAS, can be shared with others via the pheromone tables.

A large variety of software execution frameworks and middleware are possible candidates for providing an underlying service to the deployment logic depending on what kind of services we consider. Various examples for different environments are presented in the papers included in this thesis. For collaborating software components one such candidate framework is the MUSIC middleware platform [RBL^{+}08] that promises support for self-$*$ properties and component based software. To consider dependability aspects of a deployment fault tolerant group communication systems are suitable platforms, such as the DARM platform [MG08]. In a cloud computing scenario, where the deployment logic is used to optimize placement of VM instances a typical Infrastructure-as-a-Service (IaaS) setting is used. Candidate cloud computing platforms include for example the Amazon EC2 system [LLC] or VMware. Importantly, however, the intention with the deployment logic is that it is agnostic to the underlying (monitoring/execution) platform, i.e. optimization of the deployment can be done based on service models, regardless of the physical framework underneath.

2. Methodology

The research in this thesis has been conducted in a step-wise manner, starting from discovering the gains that can be achieved in software performance by distributing components of an application in an efficient manner and noticing how changes in the software architecture, in particular in the deployment, affect the QoS even in relatively smaller scale systems. The deployment logic proposed in this thesis is aimed to satisfy the requirements specified earlier, REQs 1 through 6. To obtain a logic adhering to these requirements, tangible scenarios were selected with increasing complexity and size. These scenarios were used to test new developments in the logic. At every stage, an example that is tractable and which has verifiable solution(s) has been devised. New examples were used to test every increment of additional complexity. These examples are described in detail in the included papers in Part II.

For each example scenario extensive simulations were conducted. The algorithms, the logic consists of, were implemented in a process-oriented, discrete event simulator environment called Simula/DEMOS [Bir03]. Repeated simulation runs were executed for each scenario, e.g. 100 runs with one set of parameters, to gain some statistical insight into how the logic behaved. In particular, two methods are investigated extensively that are crucial for deployment decision making: (i) deriving cost functions and (ii) a bio-inspired optimization method.

Cost functions are devised to characterize the utility of a given deployment configuration. The appropriate functions contribute to finding efficient deployments, while allowing the method itself to operate with a large variety of QoS properties (REQ-6). Regarding optimality of the solutions, the target is not necessarily the absolute global optimum. Instead, mappings that have acceptable QoS and can be obtained fast are targeted (REQ-1).

Second, a bio-inspired, decentralized method is adopted and tailored to the deployment problem. With the decentralized logic the need for centralized information storage and decision making is avoided, thus eliminating a possible bottleneck and single point of failure in the system (REQ-2). Obtaining and recalculating deployment configurations due to changes in the execution context are necessary and useful in scenarios such as large-scale, geographically distributed data-centers provisioning reactive services or high-availability virtualized server environments. Here, the inherent adaptation capability and autonomicity of self-$*$ systems are utilized (REQs 3 and 4). To cope with large-scale problems, algorithms especially require an efficient data representation that is read and updated frequently. A suitable data representation not only saves storage space, but also contributes to faster execution (REQ-5).

Lastly, cross-validation of parts of the results was achieved via a centralized method, integer linear programming (ILP) that provides global optimum solutions for given deployment scenarios, based on a global overview. The integer programs developed as part of the thesis are implemented in the AMPL language [FGK02], and the Gurobi solver [Opt] was used for executing them.

Accordingly, the next sections first introduce the bio-inspired heuristic search method, the CEAS, applied in this thesis for optimization. Hence, the next section is a background section. Sections 2.2, 2.3 and 2.4 introduce the three main areas where the contributions of this thesis are concentrated at. The structure of pheromone databases are crucial for CEAS and have to be tailored to the specific problem targeted, this is discussed in Section 2.2. Section 2.3 introduces the concrete algorithms built upon CEAS. Lastly, Section 2.4 discusses the details of cost function design for various flavors of the deployment problem.

2.1 The Cross Entropy Ant System

This section introduces the bio-inspired heuristic optimization method that is at the core of the deployment logic. CEAS is used to obtain mappings M, i.e. solutions to a given deployment problem. The method originates from Helvik and Wittner who introduced it first in [HW01], as a subclass of Ant Colony Optimization (ACO) systems (for an introduction on ACO see [DMC96]). The CEAS is an agent-based optimization framework, in which the agents' behavior is inspired by the foraging patterns of ants. The key idea in CEAS is to let many agents, denoted *ants*, search iteratively for the solution of a problem taking into account the constraints and a cost function predefined. Every iteration consists of two phases. First, during the phase called *forward search* ants search for a possible solution, resembling the search for food in real world ants. The second phase is called *backtracking*, in which ants – after evaluating the solution found during the first phase – leave markings, denoted *pheromones*, that are in proportion to the quality of the solution. Pheromones are then distributed at different locations and can be used by forward ants in their search for improved solutions. Therefore, the best solution is approached gradually. Generally, there is a trade-off between convergence times and solution quality. Nevertheless, in case of the main topic of this thesis, deploying services in a dynamic environment, a pre-mature solution that satisfies both functional and non-functional requirements often suffices. ACO systems have been proven to find the optimum at least once with a probability close to one, and after that convergence to the optimum is secured in a finite number of iterations [SD02]. Since CEAS can be considered as a subclass of ACO the optimal deployment mapping will eventually emerge.

There is significant difference, however, between the various existing ant-based systems and the approach taken in CEAS in evaluating the solution and in pheromone updates. Namely that CEAS uses the *Cross Entropy (CE) method* originally introduced for stochastic optimization by Rubinstein in [Rub99]. The centralized CE method has been shown to be able to provide near-optimal solutions to NP-hard problems in polynomial time. The method can be reformulated to govern the independent, asynchronous behavior of agents in CEAS. Compared to similar bio-inspired systems the application of the CE method makes CEAS unique in that: (i) a formal foundation is provided, and (ii) it provides two stages, where the second stage adjusts a parameter regulating the search focus of the algorithm efficiently. The CE method is applied as described below.

CEAS has been successfully applied to a variety of problems, such as different path management strategies, shared backup path protection, p-cycles, adaptive paths with stochastic routing, and search for resources under QoS constraints, see a more elaborated list in [HHW08]. Additional details of the method have been worked out thoroughly and are summarized in [HW10]. Also, detailed studies are reported in [HW10] in particular with focus on implementation issues and trade-offs, reducing the management overhead imposed by the additional traffic of management packets, short recovery times and the mechanisms required to be in place for these improved features. These mechanisms include elitism, self-tuned packet rate control, and pheromone sharing. Using pheromone sharing, ants with overlapping search criteria can cooperate in finding solutions by (partially) sharing information.

In this thesis, CEAS is applied to obtain efficient deployment mappings in the form $M : \mathbf{C} \rightarrow \mathbf{N}$. In order to find mappings, ants move between interconnected nodes in the network in search for hosting capacities. During every iteration, when the *forward search*, described above, ends a cost function is used to evaluate the mapping found. The appropriate cost function, $F()$ is selected depending on the requirements (see Section 2.4). As discussed, the *backtracking* phase has to update the pheromone values proportionally to the cost found by $F(M)$. These pheromone values, denoted $\tau_{mn,r}$, will then be used in the iterations that follow to gradually find improved mappings. The value of a given $\tau_{mn,r}$ is directly associated with a set of instances m to be deployed at node n. Besides, a pheromone value can be updated in every iteration, thus the index for an iteration r. Pheromone values in general can represent the set m in different ways, this is described in Section 2.2 where techniques to encode pheromone values are discussed.

Using the pheromones stored in a node, a *random proportional rule*, shown in (1) is used to select deployment mappings during the *forward search* phase. That is, during the first phase of each iteration in CEAS, sets of instances can be selected for mapping to a node according to the random proportional rule matrix $p_{mn,r}$

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in M_{n,r}} \tau_{ln,r}} \tag{1}$$

The probability matrix $\mathbf{p}_r$ is then changed by applying the CE method and using the cost of the solution found. The objective of the method is to minimize the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. How optimization is done by minimizing the cross entropy between samples is elaborated in [Rub99], and a distributed, auto-regressive variant is presented in [HW01]. The main difference is that in [Rub99] all the global elements (e.g. nodes) are updated in each iteration based on a batch of samples, while in CEAS pheromones are updated after each sample, as described later in this section.

CEAS was originally developed to find efficient paths – between source and destination nodes – for network management purposes, in particular routing. In that case, cost functions were applied to evaluate the paths found and to influence routing decisions based on path quality. Using CEAS for routing the pheromones $\tau_{ij,r}$ were defined as an assignment between interface i and a node j at iteration r. Selection of the next hop to visit for each ant, in this case, is based on the random proportional rule. On the contrary, in this thesis mappings are selected by the random proportional rule and next hop selection is completely independent from the pheromones. Selection of the next hop is done solely based on a guided random walk, which optionally may use taboo-lists.

The regular updates of the pheromone values are controlled by a parameter called the *temperature*, denoted γ_r. This is chosen to minimize a *performance function*, denoted $H(F(M_r))$, defined as follows

$$H(F(M_r),\gamma_r) = e^{-F(M_r)/\gamma_r} \tag{2}$$

This function is applied to all r samples and the expected overall performance satisfies

$$h(p_{mn,r},\gamma_r) = E_{\mathbf{p}_{r-1}}(H(F(M_r),\gamma_r)) \geq \rho. \tag{3}$$

$E_{\mathbf{p}_{r-1}}(X)$ is the expected value of X subject to the probability matrix $\mathbf{p}_{r-1}$, and ρ is the *search focus* parameter close to 0 (typically 0.05 or less). Finally, a new updated set of rules, $\mathbf{p}_r$, is determined by minimizing the cross entropy between $\mathbf{p}_{r-1}$ and $\mathbf{p}_r$ with respect to γ_r and $H(F(M_r),\gamma_r)$.

To allow decentralized execution and to avoid having to use synchronized batch-oriented iterations the cost of mappings, $F(M_r)$ is calculated *immediately* after each sample, i.e. at the end of the *forward search* phase, and an auto-regressive formulation of the performance function, $h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1-\beta)H(F(M_r),\gamma_r)$ is applied. Moreover, the function $h_r(\gamma_r)$ is approximated by

$$h_r(\gamma_r) \approx \frac{1-\beta}{1-\beta^r}\sum_{i=1}^{r}\beta^{r-i}H(F(M_r),\gamma_r) \tag{4}$$

where $\beta \in \langle 0,1 \rangle$ is called the *memory factor*, used for weighting (geometrically) the output of the performance function. The temperature γ_r in turn is determined by minimizing it subject to $h(\gamma) \geq \rho$. The temperature furthermore is equal to

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r}\sum_{i=1}^{r}\beta^{r-i}H(F(M_i),\gamma) = \rho\} \tag{5}$$

which is a complicated (transcendental) function that is both storage and processing intensive since all observations up to the current sample, in other words the entire mapping cost history $\{F(M_1),\ldots,F(M_r)\}$ must be stored, and weights for all observations have to be recalculated [HW01]. This can be a prohibitively large resource demand, especially in an on-line application of CEAS. As a resolution it is assumed, given a $\beta \approx 1$, that the changes in γ_r are typically small from one iteration to the next, which enables a first order Taylor expansion of (5) as follows

$$\gamma_r = \frac{b_{r-1}+F(M_r)e^{-F(M_r)/\gamma_{r-1}}}{(1+\frac{F(M_r)}{\gamma_{r-1}})e^{-F(M_r)/\gamma_{r-1}}+a_{r-1}-\rho\frac{1-\beta^r}{1-\beta}} \tag{6}$$

where the parameters $a_0 = b_0 = 0$ and $\gamma_0 = -F(M_0)/\ln\rho$. Furthermore,

$$\begin{aligned} a_r &\leftarrow \beta(a_{r-1} + (1+\tfrac{F(M_r)}{\gamma_r})e^{-\frac{F(M_r)}{\gamma_r}}) \\ b_r &\leftarrow \beta(b_{r-1} + F(M_r)e^{-\frac{F(M_r)}{\gamma_r}}) \end{aligned} \tag{7}$$

where the performance function, (2), is adopted. Further details on the application of the Taylor expansion and its necessity can be found in Appendix A of [Wit03]. Recent work on the theoretical foundations of CEAS, [PH09] and [PH10], revisits the auto-regressive functions. This parallel work substitutes the hyperbolic approximations presented in this section with linear approximations. It is shown that linear approximations are more accurate and robust to radical changes than hyperbolic ones, while having similar computational complexity. These results have not yet been incorporated in the CEAS used in this thesis and remain to be tested.

Since the pheromone values have to be stored and updated regularly by the algorithms used in the deployment logic, it is worthwhile to look at the equations describing these operations. Note that these operations are independent of the actual pheromone encodings applied. Generally, pheromone values in CEAS are a function of the *entire history* of the mapping cost values, given by the cost function. Hence CEAS has what is denoted a search history dependent quality function [Z$^+$04]. Updates to the pheromone values are made by applying the performance function, (2), combining the last cost value $F(M_r)$ and the temperature γ_r, calculated by (6). Thus, updates are according to the following equation.

$$\tau_{mn,r} = \sum_{k=1}^{r} I((m,n) \in M_k)\beta^{\sum_{x=k+1}^{r} I((m,\cdot)\in M_x)} H(F(M_k),\gamma_r) \tag{8}$$

where $I()$ is a general indicator function, hence $I(x) = 1$ if x is true, 0 otherwise. The *memory factor*, β, supplies geometrically decreasing weights to the output of the performance function, enabling *evaporation* of pheromones. The exponent of β is somewhat complex since ants during *backtracking* do not update all nodes in the network, only those nodes that were visited during the preceding forward phase, i.e. $\forall n \in H$. This exponent in (8) represents the number of ants that have updated node n between time-step r and k when a mapping M_k was found, while $r-k$ is the total number of updates in the system, i.e. total number of ants that returned between time-step r and k. Hence $r-k \geq \sum_{x=k+1}^{r} I((m,\cdot) \in M_x)$. However, as for the temperature, in (5), excessive processing and storage requirements also apply for (8). To circumvent this problem (this time a second order) Taylor expansion can be applied to (8), resulting in

$$\tau_{mn,r} \approx I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}} + A_{mn} + \begin{cases} -\frac{B_{mn}}{\gamma_r} + \frac{C_{mn}}{\gamma_r^2} & \frac{1}{\gamma_r} < \frac{B_{mn}}{2C_{mn}} \\ -\frac{B_{mn}^2}{4C_{mn}} & \text{otherwise} \end{cases} \tag{9}$$

where

$$\begin{aligned} A_{mn} &\leftarrow \beta(A_{mn} + I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}}(1 + \tfrac{F(M_r)}{\gamma_r}(1 + \tfrac{F(M_r)}{2\gamma_r}))) \\ B_{mn} &\leftarrow \beta(B_{mn} + I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}}(F(M_r) + \tfrac{F(M_r)^2}{\gamma_r})) \\ C_{mn} &\leftarrow \beta(C_{mn} + I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}}(\tfrac{F(M_r)^2}{2})) \end{aligned} \tag{10}$$

The initial values for (10) are $A_{mn} = B_{mn} = C_{mn} = 0$ for all (m,n). Again, a stepwise explanation on how the Taylor expansion is applied is given in Appendix A of [Wit03].

Improvements in and evaluation of the scalability of the CEAS core method is discussed in details in [HW10].

This thesis does not deal with evaluating or improving the inner workings of the CEAS method. Instead, CEAS is applied as a heuristic optimization method to solve a problem it has never been applied to before. When the method was applied for finding deployment mappings, the theoretical background of CEAS, in particular the application of the CE method introduced above, was not modified. The main contributions of this thesis related to CEAS are concerning: (i) how to encode solution variables as pheromone values (Section 2.2); (ii) design of an algorithm that uses CEAS, in which the agents are guided in such a way which will result in finding efficient mappings (Section 2.3). Pheromone value encodings and the role of CEAS in the deployment mapping algorithm are presented next.

2.2 Pheromone value encodings

Optimization governed by the cost function starts with aligning pheromone values, τ, with the decision variables. During optimization values of the decision variables characterize a given solution. A solution to the deployment problem is a list of mappings M, which contains sets of instances (e.g. components or VMs) $C \subseteq \mathbf{C}$ per node $n \in \mathbf{N}$. Accordingly, at each node n the pheromone database has to be able to describe possible combinations of C, which will then be suggested for deployment for visiting ants, each entry being proportional to the quality of deploying the given set of instances to n. Importantly, however, the structure of this database has to be such that it is efficient both with respect to size and complexity to lower the algorithm's demand for storage and processing at the nodes.

Table 2: Pheromone encodings for a service S_l with $|C_l|$ instances

Encoding	DB size in a node	Encoding example with $\lvert C_l\rvert = 4$
bitstring	$2^{\lvert C_l\rvert}$	$[0000]b \ldots [1111]b$
per comp.	$2 \cdot \lvert C_l\rvert$	[0/1]; [0/1]; [0/1]; [0/1]
# replicas	$\lvert C_l\rvert + 1$	$[0] \ldots [4]$

Appropriate solutions can be found using different encodings, however, there are differences in terms of convergence times and solution quality. As the distributed pheromone database consumes memory in every node scalability dictates keeping the database as compact as possible. The two main parameters influencing the size of the database in each node are the amount of instances within each service ($|\mathbf{C}|$) and, also, the amount of services that can use the particular node for deployment ($|\mathbf{S}|$). The amount of instances within a service might not grow above a certain extent, the amount of services that the deployment logic shall be able to handle might in turn be very high. The memory needed to cater for the required amount of services can directly be influenced by the pheromone encoding. Beside the storage needs, an individual ant has to browse through the pheromone entries during its visit to a node. Clearly, a more compact pheromone database helps speeding up execution of the tasks it has to perform. These considerations about the pheromone database are not only relevant to the simulations conducted, but to an on-line implementation of the logic as well.

The most comprehensive encoding would be to implement addressing of the pheromone tables along three dimensions:

- all available nodes in the network;
- all instances to be deployed;
- yes/no flag to represent deployment.

Using a table this large in every node, however, would result in enormous memory consumption. Instead, different encodings were evaluated during this work, the three encodings that were published are summarized in Table 2 and are described below.

(i) A straightforward encoding (*bitstring*) is to implement a flag, i.e. a bit, assigned to each instance. This way producing a bitstring that can describe any combination of instances within a service. This encoding results in a database of size $2^{|\mathbf{C}|}$ for each service, within each node in the network. *bitstring* results in the largest database as it holds a single value for all possible combinations of instance mappings in every node, which can result in a prohibitively large memory need. It is important to note here that $|\mathbf{C}|$ represents the total number of components and possible replicas as well in a given service. Hence, $|\mathbf{C}|$ is not to be confused with the replication level of a single component. For example, in case of $|\mathbf{C}| = 15$ instances per service this encoding leads to 2^{15} pheromone values per service. Let the number of services that are to be deployed be $|\mathbf{S}| = 25$, in a network of $|\mathbf{N}| = 100$ nodes. This results in a total of $2^{15} \cdot 25 \cdot 100 = 81.92$ million individual pheromone values in that given network. This high amount of pheromones is difficult to handle computationally, even if agents in CEAS do not have to crawl through all values in every iteration. Furthermore, let the pheromone values be stored using 4 byte long floating point numbers. In this case, the total distributed pheromone database would consume 312.5 MBytes of memory in the network. Hence, the *bitstring* encoding is inefficient regarding memory consumption and the amount of required computations.

(ii) To reduce the size of the database more simple bookkeeping can be applied, taking into account solely the number of instances mapped to a given node, in the encoding denoted *# replicas*. This is the most compact way to encode the pheromone values. The tradeoff is that it cannot distinguish between instances in a service specification, thus it can only be applied if this limitation does not hinder the deployment, e.g. in case of a service model consisting of equally sized and equivalent replicas only. Considering the same example of $|\mathbf{C}| = 15$ instances in each of the $|\mathbf{S}| = 25$ services, in a network of $|\mathbf{N}| = 100$ nodes, the resulting database size is $(15+1) \cdot 25 \cdot 100 = 40000$ values, or approximately 0.15 MBytes.

(iii) As a good compromise between these two options, the encoding denoted *per comp* was developed. Using *per comp* does not result in information loss, while still being linear in size. This encoding uses one distinct pheromone entry for every instance indicating whether or not to deploy them at a given node. The slight disadvantage is that ants arriving at a node have to decide on the deployment mapping of each instance, one-by-one reading the multiple pheromone entries corresponding to the elements of the service (one separate table element for each). Nevertheless, this encoding provides the necessary reduction in the size of the database structure and allows scaling it up to handle larger amounts of services and larger service sizes. For example, if $|\mathbf{C}| = 15$, $|\mathbf{S}| = 25$, and $|\mathbf{N}| = 100$, the database would use $(2 \cdot 15) \cdot 25 \cdot 100 = 75000$ pheromone values, or 0.29 MBytes of memory.

Examples for the different pheromone table structures, given that 4 instances have to be deployed are shown in Figure 5.

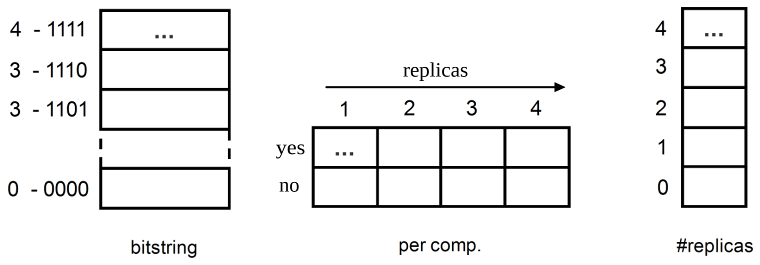


Figure 5: Three different pheromone table structures for 4 instances

Use of the pheromone tables and the CEAS method in the deployment algorithms developed is discussed in the next section.

2.3 Deployment algorithms

This section gives an introduction to the general parts of the deployment algorithm that was devised. The task of obtaining deployment mappings for a service is given to one instance of the logic. Ants belonging to this instance are then denoted as one *species*. Species have access to one service model each. The service model contains the set of instances, **C**, to be mapped. Ants are emitted continuously and select nodes to visit. There are two types of ants, *explorer* and *normal* ants.

During *forward search*, ants visit nodes and form $|H| \leq |\mathbf{N}|$ discrete sets from the set of available instances, **C**. The set that is formed may as well include the empty set for some nodes. Ants then evaluate this resulting mapping using the cost function. According to the cost, the pheromone database will be updated (using (8)) during *backtracking* in the nodes in H. The pheromone values corresponding to the mappings in M will be incremented, while the rest of the elements will be decremented slightly by the mechanism called *evaporation*. Pheromone values can then be normalized to provide a probability distribution indicating the likelihood of mapping a given set of instances to the particular node, as shown in (1). Eventually, after convergence the suggested solution emerges in the distributed pheromone database with probability near to one [SD02], unless there are two solutions, which have identical best cost. To avoid getting stuck in premature and sub-optimal solutions, explorer ants explore the search space during *forward search* by ignoring the pheromones and selecting mappings uniformly random instead.

It is possible to use explorer ants only in the initial phase of optimization. This technique can be used to explore and cover a significant portion of the problem space by random sampling. The length of this initial exploration depends on the problem size, in terms of network size and number of services. After initial exploration, the majority of ants are normal ants, while a smaller fraction are explorers, typically $5-10$ percent. This continued exploration is meant to capture fluctuations in the network, e.g. to detect new nodes connecting. Thereby responsiveness to dynamism in the environment can be improved. In any case, normal ants aim to find an optimal mapping. *Forward search* followed by *backtracking* completes an iteration, i.e. the lifetime of a single ant ends, and a new ant can be emitted starting a new iteration of the algorithm, unless a stopping criteria is met. For adaptation however, the algorithm can continue (at a lower rate) and maintain the deployment mappings until a new, more preferable mapping is found depending on the changes in the execution context. The same continuous ant behavior can be used:

(i) for obtaining an initial mapping to start execution of a service (in this case convergence to an applicable solution can serve as stopping criteria);

(ii) for an online (re-)deployment mechanism when the service is already running.

A simplified diagram illustrating the basic classes of the deployment logic and their relations is shown in Figure 6.

The two main classes are the *Nest* and the *Ant*. When a species is started at least one nest is placed on top of a node in the network. It is allowed for one species to have more than one nest for redundancy. Parallel nests will emit ants corresponding to the same set of services that can operate using the same pheromone tables [HW10]. If nests can be coordinated, this will not result in flooding the network with ants as the rate of emission in a stable network can be divided equally between the nests (bullet 7) in Section 6). Furthermore, ants emitted from different nests but optimizing mappings for the same service will update the same pheromone tables in the nodes they visit [HW10]. During execution, synchronization between nests is not necessary, but only a primary nest will execute deployment decisions and trigger the physical placement of components.

The functionalities required from each node in the network to support the deployment logic are summarized in the class *Node*. Every node must have an allocation table to support load sampling and allocation of processing resources, and a pheromone table for CEAS to operate with. Where appropriate nodes can have additional, higher level categorization, e.g. clusters, or clouds. The class *PheromoneTable* is used as a container for the distributed information needed by the logic, i.e. values of $\tau_{mn,r}$, and implements updating and selection mechanisms of (9) and (1). The actual implementation and format of how information is stored in *PheromoneTable* is depending on

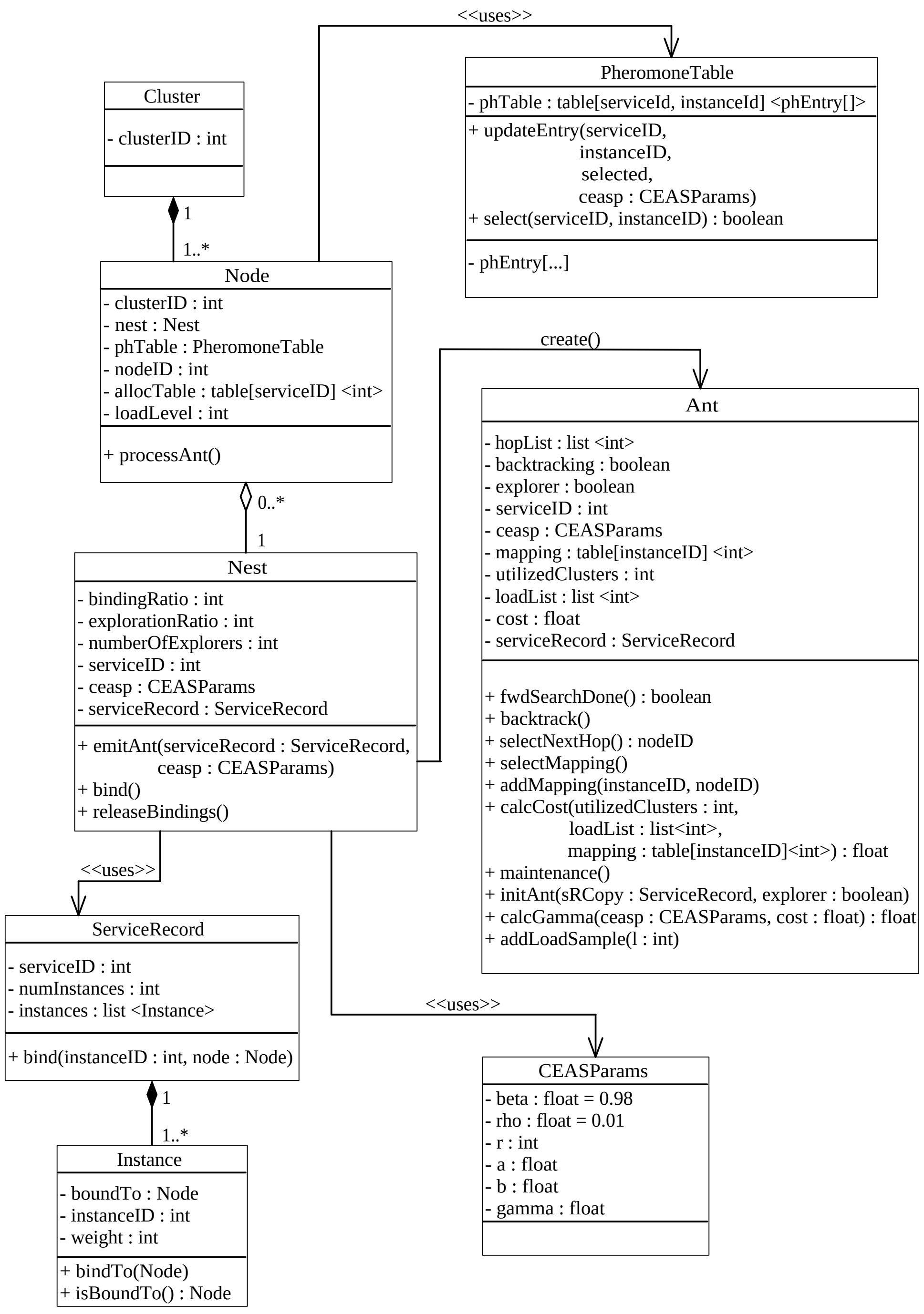


Figure 6: Generalized class diagram of the deployment algorithm

the encoding, which is discussed in Section 2.2.

A *Nest* uses in addition a set of CEAS related parameters and variables represented by the class *CEASParams*, which information is shared by all the nests of the same species, e.g. the memory parameter β, or the search focus ρ. The current values of CEAS related parameters are also passed on to the the ants, which in turn use them when accessing the pheromone tables. Moreover, a *Nest* must be able to access information about the service that the species has to deploy, this information is available in the class called *ServiceRecord*, corresponding to S_k. Also this class implements binding (fixing the deployment mapping) of instances within the service model. A single building block of a service, e.g. a component, a replica, or a VM is described by the class *Instance*. This is the class where costs associated with an instance can be stored, e.g. the processing cost of the instance in $weight$.

The second class of the logic, *Ant*, realizes the ants emitted iteratively by the $Nest$. Most of the intelligence is carried by the *Ant*, represented by methods of the class. The variables used by the ant during an iteration of the optimization process are part of the class, e.g. the mapping set *mapping* corresponding to M, and *hopList* for H, $utilizedClusters$ for D, $loadList$ for L, or *serviceRecord* for S_l.

In contrast to CEAS-based routing, where the temperature can be stored in the destination node of a path to be set up, for deployment mapping – as no distinct destination node exists – the temperature has to be passed on to subsequent ants via the nest. The simplified behavior of an ant is illustrated by the activity diagram in Figure 7.

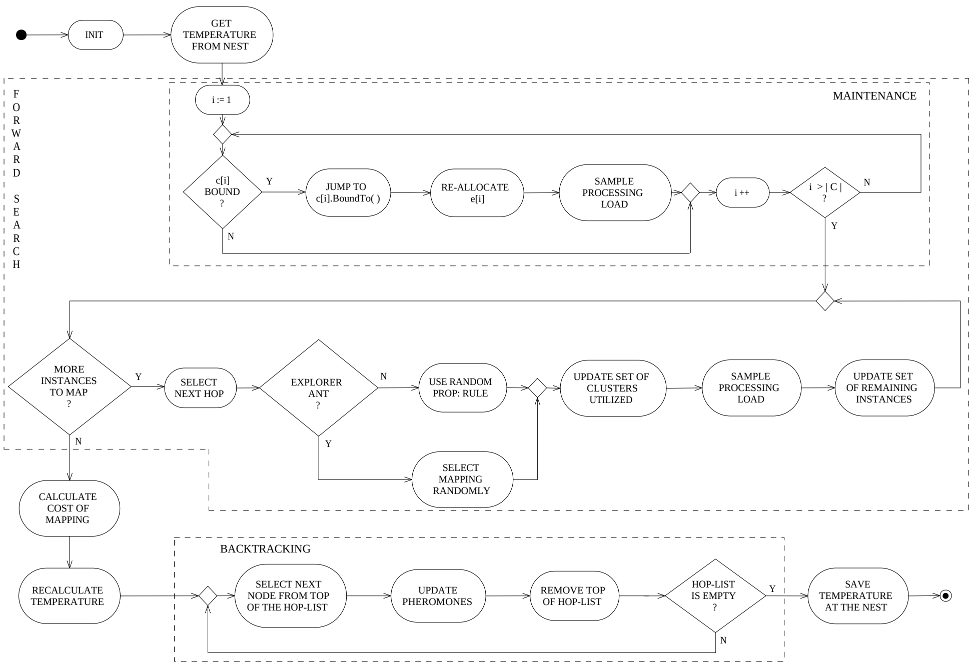


Figure 7: Generalized activity diagram for an *Ant*

After an ant is initialized, the current temperature, i.e. γ_r is retrieved from the nest. Then, the first phase of an iteration, *forward search* begins. First, an ant visits the nodes, if any, that already have a bound instance mapped to maintain these mappings, which will also be taken into account when the cost of the total mapping is evaluated. This *maintenance* phase includes re-allocating processing load for the bound instances and sampling the overall load at the nodes visited during this process. Moreover, the pheromones corresponding to these bound mappings will be included in the updates during *backtracking*. When finished with the bound instances, ants start visiting nodes by selecting their next hop randomly. The selection of the next node to visit is independent from the pheromone markings laid by previous ants. Different strategies for guiding the next hop selection, instead of using a pure random walk, can increase the frequency of finding efficient mappings, thus shortening convergence times. Guidance can be used to different extents, i.e. in case of simple settings without replication management where the network is not partitioned into clusters random drawing is applicable. Whereas when replication is required and dependability rules Φ have to be satisfied, the guidance and taboo-listing of nodes can be turned on. Guided random hop selection is introduced in more detail in PAPER E.

Using next hop selection an ant iteratively visits nodes until all instances are mapped. In a node n a mapping set $m_{n,r}$ is selected either randomly (explorer ant) or based on the pheromones using (1). Additional bookkeeping is done before the ant leaves a node, required auxiliary variables are updated, for example D, L, C. When the ant is ready with a complete mapping for the service the cost of the mapping is calculated (using the cost function that is applicable) and the temperature (carried along by the ant) is updated according to (6). The second phase in the lifetime of an ant, i.e. *backtracking* commences after the updated cost and temperature are available. During this phase, nodes in the hop-list are re-visited in a reversed order and the pheromone tables are updated according to (9). Lastly, before the ant is recycled in the nest, the current temperature for the species is updated.

The deployment logic uses the nests to emit ants into the network continuously, or until some stopping criteria is met. The simplified behavior of a *Nest* is illustrated in Figure 8. Improving convergence is the concept of *binding*, which allows nests to *fix* the mapping of one instance in M at a time, based on the most recent mapping of that instance. Binding parts of the mapping for a service can be conditioned by, for example rules in the set of requirements, example to which is the set Φ introduced in Section 2.4.3. After a *bind* ants looking for a suitable deployment for the same service do not change the fixed mapping in subsequent iterations and new searches will be conducted for the remaining instances only. Importantly however, bound parts are also taken into account when the cost of the total mapping is evaluated. Should a network disruption occur, such as splitting of clusters, these bindings are erased (*release bindings*) in the ant nest and the whole service will be taken into consideration in the following searches.

It has to be distinguished between the three notions of *mapping*, *binding* and *deployment* of instances of a service. The *mapping* M is a variable list constantly optimized, iteration by iteration by the logic, visible internally to the algorithm only. When an

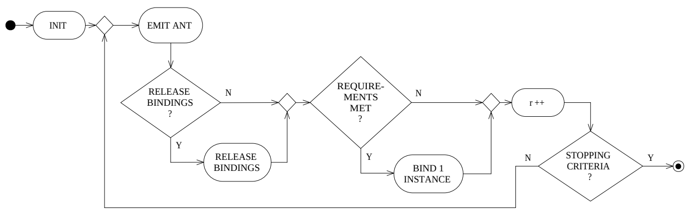


Figure 8: Generalized activity diagram for a *Nest*

instance is *bound* to a node the particular mapping for that instance is not changed anymore by the ants, until that binding is erased again. Lastly, *deployment* is the physical placement and instantiation of building blocks of a service on a node, which is triggered after the mapping M for the given service has converged to a satisfactory solution. This step corresponds to the Execute function in the MAPE loop, in Figure 4. Hence, the deployment mappings provided by the algorithms presented in this thesis are logical variables, which can be effected at desired times, and as such do not cause undesirable fluctuations by, for example, migrating parts of a service unnecessarily.

The algorithms within the deployment logic were refined continually throughout the work presented in papers A – F. The first version of the deployment algorithm was devised in PAPER A and was extended with the a basic version of load sampling in PAPER B. The nests received more attention in PAPER C. Additional variables required for handling dependability requirements were added in PAPER D, and guided random hop selection was introduced in PAPER E. The concept of taboo-lists to aid deployment in private and hybrid clouds, among others, was added in PAPER F. More precise definitions of the algorithms are included in the papers in Part II.

2.4 Quantifying deployment costs

The deployment logic presented in this thesis is built to be general enough to cater for practically any kind of requirements, as long as a corresponding function quantifying the cost of the specific property can be constructed (REQ-6). These functions, denoted $F()$, are specially designed to evaluate the utility of given configurations of the software architecture during the optimization process, which results in finding a preferred deployment. Accordingly, it is crucial for the method that the proper functions are available to rank the different deployments, guide the search iteration by iteration to a deployment mapping that satisfies the requirements. This deployment mapping can then be effected in the execution framework. Moreover, the construction of these functions has an influence not only on the quality of the solutions, but on the convergence rate as well. The importance of cost functions has gained attention recently in the autonomic computing and self-adaptive systems community as well [KD07, CLG$^+$09]. But, increasing the complexity of constructing such functions is the fact that they have to be applied in a decentralized method without any global knowledge. In the papers costs of a deployment have to be minimized, i.e. the objective is always *min* $F()$.

2.4.1 Load-balancing and communication costs

The first cost function is designed to reflect the goal of balancing the load and minimizing the communication cost. Balancing the load is practically the task of minimizing the difference between the load of each node and an ideal average load. Regarding the service, execution and communication costs are considered. The deployment logic has access to the service model. Services are defined in a collaboration-oriented style, as illustrated in Figure 2. Accessing the service model, agents in CEAS are aware of the execution costs, $f_{c_i}^{(e)}$ for each component, and the communication costs, $f_{k_j}^{(c)}$ for each collaboration. Thus, the total offered execution load for the given service can be calculated as $\sum_{i=1}^{|\mathbf{C}|} f_{c_i}^{(e)}$. Hence, the average load T over the available nodes in a network, $\mathbf{N}$, where the service is deployed becomes

$$T = \frac{\sum_{i=1}^{|\mathbf{C}|} f_{c_i}^{(e)}}{|\mathbf{N}|} \tag{11}$$

To cater for the communication costs $f_{k_j}^{(c)}$, of the collaborations k_j in the service, the function $q_0(M,c)$ is defined first.

$$q_0(M,c) = \{n \in \mathbf{N} | \exists (c \rightarrow n) \in M\} \tag{12}$$

This means that $q_0(M,c)$ returns the node n that hosts component c in the list of mappings M. Let collaboration $k_j = (c_1, c_2)$. The communication cost of k_j is 0 if components c_1 and c_2 are collocated, i.e. $q_0(M,c_1) = q_0(M,c_2)$, and the cost is $f_{k_j}^{(c)}$ otherwise (i.e. the collaboration is remote). Using an indicator function $I(x)$, which is 1 if x is true and 0 otherwise, this is expressed as $I(q_0(M,c_1) \neq q_0(M,c_2)) = 1$ if the collaboration is remote and 0 otherwise. To determine which collaboration k_j is remote, the set of mappings M is used. Given the indicator function, the overall communication cost of the service, $F_K(M)$, is the sum

$$F_K(M) = \sum_{j=1}^{|\mathbf{K}|} I(q_0(M,k_{j,1}) \neq q_0(M,k_{j,2})) \cdot f_{k_j}^{(c)} \tag{13}$$

Then, by simply adding up the two types of deployment costs, i.e. load-balancing and remote communication costs, the cost function is defined as

$$F_1(M) = \sum_{n=1}^{|\mathbf{N}|} |\hat{l}_n - T| + F_K(M) \tag{14}$$

where $\hat{l}_n$, $n = 1 \ldots |\mathbf{N}|$ denotes CPU-usage samples from all nodes. New samples are taken in every iteration of the method and describe the amount of execution load of the components mapped to each node n, i.e. $\sum_i f_{c_i}^{(e)}$, for $\forall c_i$, where $q_0(M,c_i) = n$. The absolute value $|\hat{l}_n - T|$ then is used to penalize deviation from the desired average load per node.

To illustrate evaluation of deployment mappings using the cost function, consider

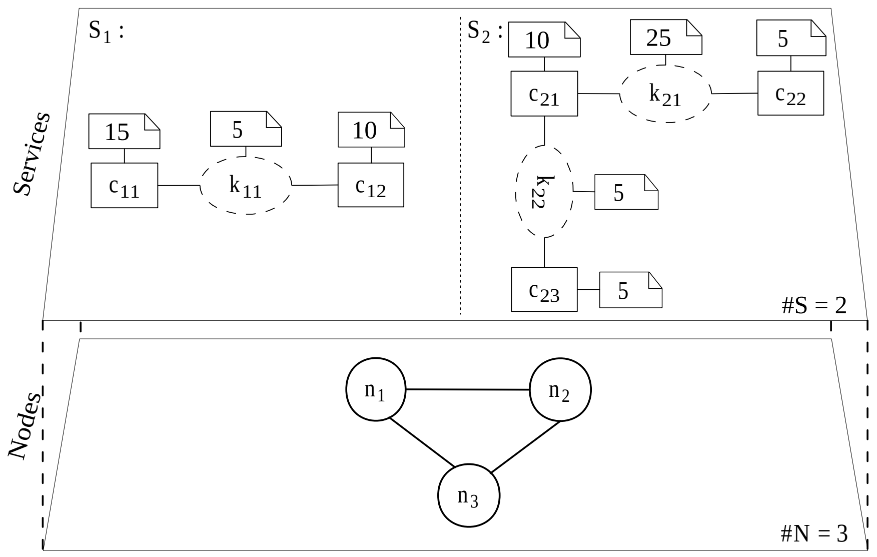


Figure 9: Example network and services

the simple network in the bottom layer of Figure 9 consisting of 3 identical nodes n_1, n_2, n_3 in a full mesh.

The deployment task is then to place service components into the example network. Let the two services $S_1, S_2 \in \mathbf{S}$ to be deployed be defined by the models in the *Services* layer of Figure 9, including the corresponding costs in the annotations. The two services have a total of 5 instances and 3 collaborations altogether.

Two deployment mapping examples, $M^{(1)} = \{\{c_{11} \to n_1\}, \{c_{12} \to n_2\}, \{c_{21} \to n_3\}, \{c_{22} \to n_3\}, \{c_{23} \to n_2\}\}$ and $M^{(2)} = \{\{c_{11} \to n_3\}, \{c_{12} \to n_2\}, \{c_{21} \to n_1\}, \{c_{22} \to n_2\}, \{c_{23} \to n_3\}\}$ are shown in Figure 10. A cost calculation example is presented below, and will be followed by more examples in the following sections, to illustrate the usage of the formulas only.

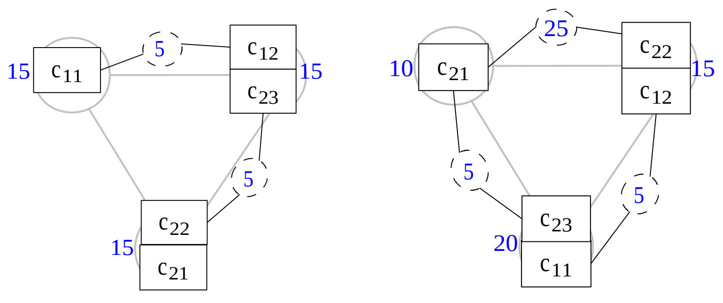
- *Cost calculation example 1.*

  The parameter T in (11) for this scenario is $T = (\sum_{i=1}^{5} f_{c_i}^{(e)})/3 = 15$, giving the targeted average load. The communication costs, (13) are calculated as follows

$$F_K(M) = \sum_{j=1}^{3} I(q_0(M, k_{j,1}) \neq q_0(M, k_{j,2})) \cdot f_{k_j}^{(c)},$$

  which gives $F_K(M^{(1)}) = 10$ with 2 remote collaborations and $F_K(M^{(2)}) = 35$ with all collaborations executed remotely.

  Figure 10 also shows the load levels at each of the nodes. These are the load levels that the deployment logic can sample from. The samples for



(a) Example deployment $M^{(1)}$ (b) Example deployment $M^{(2)}$

Figure 10: Deployment mapping examples for S_1 and S_2

the two mappings are $\{\hat{l}_1, \hat{l}_2, \hat{l}_3\} = \{15, 15, 15\}$ and $\{10, 15, 20\}$ respectively. Using these calculations and data the cost function can now be evaluated. The deployment costs according to (14) are

$$F_1(M) = \sum_{n=1}^{3} |\hat{l}_n - 15| + F_K(M).$$

Resulting in $F_1(M^{(1)}) = 10$, indicating absolute load balance, and $F_1(M^{(2)}) = 5 + 5 + 35 = 45$. Clearly, $M^{(2)}$ has much higher deployment cost due to the worse load distribution and the high remote communication costs. Note that these are the cost values obtained by all of the ant species (one for S_1 and another for S_2), since all of them have the same global overview. $\square$

The function (14) is applied for calculating deployment costs in PAPER A and PAPER G, together with related example service models.

2.4.2 Deployment costs for multiple services

Using (14), deployment costs of a single service can be quantified easily. When multiple services are to be deployed in the same network the average workload on the (physical) nodes T, presented in (11), has to be calculated taking into account the execution costs of all components from all services. Furthermore, to be able to determine a global load average given the amount of nodes in the network (note that all nodes are considered identical) the sum has to be over all nodes. Hence, *global* knowledge is required in this case. For improved scalability one instance of the deployment logic (and CEAS) is run for each service that has to be deployed. Through cooperation between the parallel instances of the logic the problem size is split into smaller pieces as a single instances has to deal with a single service only and thus, it does not need an overview over all the services and their deployment. Each CEAS instance stores information at participating nodes to facilitate cooperation and to obtain a more holistic view of the environment. Having a global parameter, like T would require synchronization between the instances of the deployment logic searching for a solution simultaneously.

To eliminate the need for global knowledge, the average workload T is replaced with a set of load samples $\hat{l}_n \in L$. To enable this sampling mechanism a reservation mechanism is introduced in the optimization process. When components c_i of a service are mapped to a node n_i, a slice of resources equivalent to the weights $f_{c_i}^{(e)}$ of the corresponding components is reserved. Reservations are logical variables visible to the deployment logic only, similarly to mappings or bindings, and are stored and maintained in the actual nodes. This mechanism is used by the logic to estimate resource usage on nodes, and to facilitate interaction between instances of CEAS. Agents in CEAS can then sample the current reservations on a node, and use these samples to evaluate the cost of a mapping involving that node. Excessive reservation of resources in a node suggests the agents to avoid mapping even more instances, similarly to the *detestation* approach applied for finding primary/backup paths in [WH02]. An important generalization is that nodes are modelled having no explicit capacity limits. Instead, *soft*-limits are used, giving high (non-linear) penalties. Samples that would exceed the capacity of a node are quickly outranked by better solutions as a high penalty is assigned to these infeasible mappings. Resource reservations also have to be maintained and kept current. This can be achieved using timestamping to prevent stale reservations.

Beside splitting the problem size into smaller pieces by launching separate instances of CEAS for every service and introducing sampling of load levels to remove the need for global knowledge, additional improvement in scalability can be achieved by combining load-sampling with a hop-list, similarly to taboo-listing presented in [HW10]. The hop-list H, $|H| \leq |\mathbf{N}|$, accounts for a list of nodes sampled, which is normally a significantly smaller set than the set of all nodes, $\mathbf{N}$. This way, the extent of the first sum in (14) can be decreased from $|\mathbf{N}|$ to $|H|$. Practically, this means that near-optimal solutions can be found without exploring the total sample space of $|\mathbf{N}|$ nodes, i.e. the logic does not need to know the size of the sample space. This combined approach is introduced and its effectiveness is discussed in PAPER B and PAPER C, and the concept is used in PAPER D, PAPER E and PAPER F as well.

The form of a cost function partly depends on the parameters used. To cater for multiple services and the additional mechanisms described above, (15) is formed that uses the deployment mappings in M and the set L. Experiments with various combinations of multiplicative and additive functions are presented in PAPER D included in Part II. According to the experiments, the multiplicative combination presented in (15) was chosen, because this combination showed the best performance. Where the $+1$ in the second term is used to avoid evaluating the function to zero in case there are no communication costs. Hence, load-balancing costs are preserved in any case.

$$F_2(M,H,L) = \left[\sum_{\forall n \in H} C_0(n) \right] \cdot (1 + \omega \cdot F_K(M)) \tag{15}$$

In (15), ω is a scaling parameter for $F_K(M)$, identical to (13), incorporating the communication costs individually for each service. ω in practice can be set intuitively according to the costs in the service models. $C_0(n)$ quantifies the node local costs for node n.

$$C_0(n) = \left(\sum_{i=0}^{\hat{l}_n} \frac{1}{[\sum_{\forall n \in H} \hat{l}_n] + 1 - i} \right)^2 \tag{16}$$

Function (16) targets load-balancing among nodes. Samples $\hat{l}_n \in L$ are obtained over the subset $H \subseteq \mathbf{N}$, in every iteration of CEAS. With the quadratic form, (16) gives exponential weighting to these costs. In principle, higher exponents than 2 can also be applied to emphasize load-balancing if needed. The sum inside (16) provides increasing penalties as the load sampled in a node, $\hat{l}_{n_i}$, gets closer to the total amount of load observed by the ant, i.e. $\sum_{\forall n \in H} \hat{l}_n$. Hence, this function gives an incentive to distribute the observed (integer valued) load. See Figure 11 for illustration, where the curve reaches its maximum when $\hat{l}_{n_i} = \sum_{\forall n \in H} \hat{l}_n$. The $+1$ in the denominator is used to avoid division by zero.

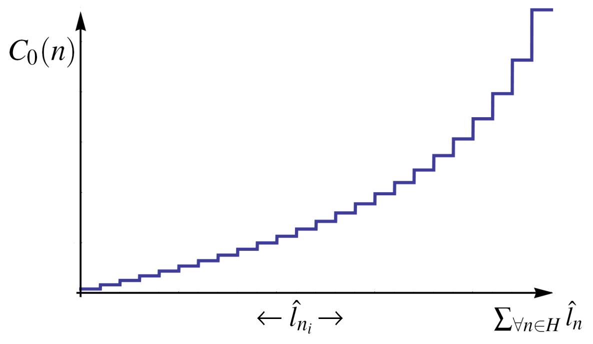


Figure 11: Graphical plot of the sum in $C_0(n)$

In the cost function (15), the first term counteracts the term for communication costs, $F_K(M)$. The effects of these two counteracting terms can be balanced using the scaling parameter ω. The quadratic nature of C_0 penalizes deviations from a balanced deployment mapping exponentially. Furthermore, Equation (15) can be used to evaluate the mappings presented in Figure 10 without using global knowledge.

- *Cost calculation example 2.*

  Assume that the ants sampled only does nodes that they actually used for mapping instances and that $\omega = 0.1$. For the ideal first mapping $M^{(1)}$ ants for S_1 sampled $\{\hat{l}_1, \hat{l}_2, \hat{l}_3\} = \{15, 15, -\}$ and ants for S_2 sampled $\{-, 15, 15\}$. Hence, for the first species

  $$C_0(n_1) = (\sum_{i=0}^{15} \frac{1}{30 + 1 - i})^2 = 0.5027,$$

  $C_0(n_2) = 0.5027$ and for the second species $C_0(n_2) = C_0(n_3) = 0.5027$. Communication costs are $F_K(M^{(1)}) = 5$ for both species individually. The overall cost for $M^{(1)}$ thus becomes

$$F_2(M^{(1)}, \{n_1, n_2\}, \{15, 15\}) =$$

$$= [C_0(n_1) + C_0(n_2)] \cdot (1 + 0.1 \cdot 5) = 1.5081$$

for the first species and $F_2(M^{(1)}, \{n_3, n_2\}, \{15, 15\}) = 1.5081$ for the second one.

Similarly, for mapping $M^{(2)}$ the samples by species are $\{-, 15, 20\}$ and $\{10, 15, 20\}$. Thus, the node local costs per species are

$$C_0(n_2) = (\sum_{i=0}^{15} \frac{1}{35 + 1 - i})^2 = 0.3327,$$

$$C_0(n_3) = (\sum_{i=0}^{20} \frac{1}{35 + 1 - i})^2 = 0.7333,$$

and

$$C_0(n_1) = (\sum_{i=0}^{10} \frac{1}{45 + 1 - i})^2 = 0.0728,$$

$C_0(n_2) = 0.1778$, $C_0(n_3) = 0.3608$. Furthermore, $F_K(M^{(2)}) = 5$ for species 1 and $F_K(M^{(2)}) = 30$ for species 2. Hence, the deployment costs in $M^{(2)}$ become $F_2(\ldots) = (0.3327 + 0.7333) \cdot 1.5 = 1.599$ and $F_2(\ldots) = 2.4456$ for the two individual species respectively.

This example demonstrates how the less efficient deployment mappings get outranked by the more optimal solutions, without the need for global overview in the logic. Specifically, S_1 sees only part of the load imbalance in the second configuration, hence it has a slightly higher cost $1.599 > 1.5081$. Whereas S_2 sees more of the imbalance and has significantly higher communication costs $2.4456 > 1.5081$. $\square$

The next section introduces how another type of requirements can be incorporated into the cost functions.

2.4.3 Replication management and load-balancing

The next type of requirements the deployment logic was extended with deals with increasing dependability through redundancy, and management of replicas in particular. When replication management is considered each component of a service may be replicated for fault tolerance and/or load-balancing. The previous network model is refined with the introduction of *clusters* of nodes, captured in the set **D** of clusters, for example d_1 and d_2 in the *Nodes* layer of Figure 12. The purpose of extending the network model is to be able to take into account dependability goals like cluster-disjoint placement of replicas. Each cluster may represent separate sites distinguished geographically or otherwise administratively. In this context, the aim of the deployment logic is to satisfy the dependability requirements and obtain an efficient mapping in the network. Regarding the services, models are defined according to the active replication approach, i.e. each component may have one or more replicas, which are operational as well, hence they have approximately equal execution costs (which cost will be referred to as w). Replicas are ready to take over the role of the a component should it fail. Thus, this replication model requires all replicas of a component to have their states updated and synchronized, using some consistency protocol. For example, consider S_3 and S_4 in Figure 12, where the replication level is 1 and 2 respectively. The role of the collaborations is simply to provide a consistent state for all the replicas.

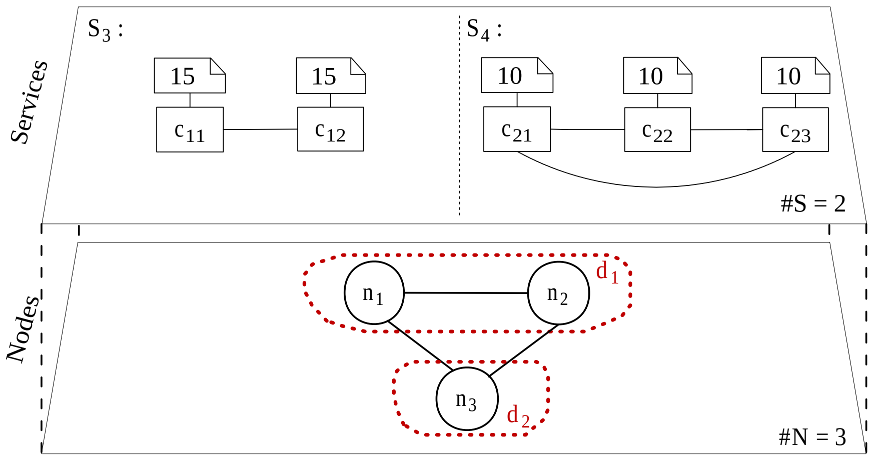


Figure 12: 2 simple replication examples

To cater for the modified set of requirements and the different network model (introduction of clusters), the cost function presented so far is modified. The new set of requirements is given as a set of dependability rules, denoted Φ, that serve as constrains to the minimization problem $min\ F()$. Φ is then defined with the aid of two helper functions that apply to a given service. The first one is $q_0(M,c)$ defined in (12) and the second one is $q_1(\mathbf{D},n)$ that returns the cluster d in which node n is located.

$$q_1(\mathbf{D},n) = \{d \in \mathbf{D} | n \in d\} \tag{17}$$

Let $\Phi = \phi_1 \wedge \phi_2$ be the set of dependability rules considered. The first rule, ϕ_1 requires replicas to be dispersed over as many clusters as possible. This rule aims to improve service availability despite potential network partitions that are assumed to be more likely to occur between cluster boundaries. In case the replication degree exceeds the number of available clusters, i.e. there are mode replicas than clusters available, at least one replica should be placed in each cluster. The second rule, ϕ_2, prohibits collocation of two replicas on the same node. It is important to note that in all the optimization problems discussed *soft* rather than *hard* constraints are used. The two rules formally,

RULE 1 $\phi_1 : \forall d \in \mathbf{D}, \forall c_i, c_u \in \mathbf{C}, |\mathbf{C}| < |\mathbf{D}| : q_1(\mathbf{D}, q_0(M,c_i)) \neq q_1(\mathbf{D}, q_0(M,c_u))$

RULE 2 $\phi_2 : \forall c_i, c_u \in \mathbf{C} : q_0(M,c_i) \neq q_0(M,c_u)$

Noticing that prohibiting collocations by ϕ_2 is contradictory to minimizing remote communication, communication costs from the service model are omitted in the cost calculation for replication management.

A new parameter, the number of utilized clusters, referred to as $|D|$, is added as a parameter to the cost function. To construct a new cost function several combinations were evaluated and results are published in PAPER D. The best results were obtained using a reciprocal term targeting ϕ_1, and reusing the load-balancing function in (16). First, (16) – previously used solely for load-balancing – is redefined and is applied for each node n to cater for ϕ_2. The redefined function, (18) is applied in two different ways, depending on the parameter $x \in \{0,1\}$. Specifically, $C_1()$ is a list of values, containing one element for each node covered in an iteration of CEAS. The list of nodes covered is available in the hop-list H. As noted for Equation (16), $C_1()$ as well applies a quadratic power, but higher exponents can also be used to enforce load-balancing even more.

$$C_1(M,L,n,x) = \Big(\sum_{i=0}^{\vartheta(n,x)} \frac{1}{[\sum_{\forall n \in H} \vartheta(n,x)] + 1 - i} \Big)^2 \tag{18}$$

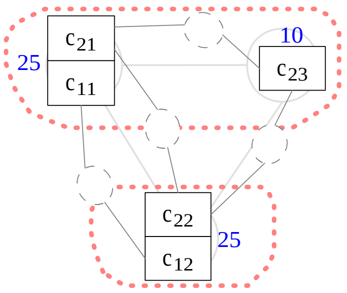
The two different usages are as follows. For $x = 1$, load samples in L are used, accounting for all concurrently executing services on the nodes sampled. $x = 0$ in turn represents solely the mappings, M, of replicas of the single service being deployed by the given instance of the logic. The difference is in the upper-bound, $\vartheta()$, of the summation and the sum in the denominator.

$$\vartheta(n,x) = (1-x) \cdot |m_n| \cdot w + x \cdot \hat{l}_n \quad \text{for } x \in \{0,1\} \tag{19}$$

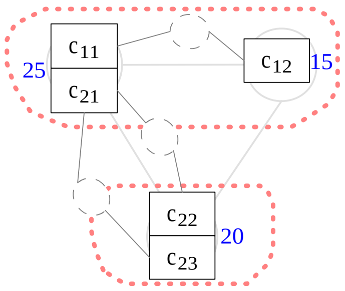
The upper-bound, as defined in (19), assumes that all replicas in a given service have the same weight, denoted w, which is derived from the service model, i.e. $\forall f_{c_i}^{(e)} = w$. Hence, their load can be assessed by multiplying by the number of replicas mapped to a given node. The number of instances mapped is found by $|m_n| = \sum_{\forall c_i \in \mathbf{C}} I(q_0(M,c_i) = n)$. This definition can easily be changed to support individual replica weights (instead of w) within a service model. Furthermore, the sum $\sum_{\forall n \in H} \vartheta(n,x)$ represents the overall execution load of one service ($x = 0$) or all services ($x = 1$) as observed by a given species. When $x = 1$ the sum is an approximation based on the samples from the set of nodes $H \subseteq \mathbf{N}$. In other words, $x = 0$ is the total processing resource demand of one service, whereas $x = 1$ accounts for the additional load of replicas from *other* concurrent services as well. This way, the list $C_1()$ provides a quadratic approximation of the share of load associated with each node as experienced by CEAS. An approximation only, as the logic does not have an exact global overview over the total offered load. Thus, the overall cost function used for replication management and load-balancing becomes

$$F_3(|D|,M,H,L) = \frac{1}{|D|} \cdot \sum_{\forall n \in H} C_1(M,-,n,0) \cdot \sum_{\forall n \in H} C_1(-,L,n,1) \tag{20}$$

On one hand, $C_1(\ldots,0)$ applies solely to replicas of one service, this way penalizing the violation of ϕ_2, or in other words favoring mappings where replicas are not collocated. On the other hand, $C_1(\ldots,1)$, is used for general load-balancing and, as such, it takes into account load imposed on nodes by the other services in the network. Using these separate terms the output of the cost function used in each iteration can be smoother, purposefully easing convergence, i.e. simplifying differentiation between very similar deployment mappings with nearly the same cost. To illustrate the cost evaluation Figure 13 presents two deployment mappings in the network of Figure 12, a valid one $M^{(3)}$ and an invalid one $M^{(4)}$. From a dependability perspective $M^{(4)}$ is invalid, since the deployment of S_3 is not cluster-disjoint and S_4's deployment is not node-disjoint (replicas c_{22} and c_{23} are collocated).



(a) Example deployment $M^{(3)}$



(b) Example deployment $M^{(4)}$

Figure 13: Deployment mapping examples for S_3 and S_4

Using (20) the cost of $M^{(3)}$ and $M^{(4)}$ is evaluated as follows (again, assuming the ants for the corresponding services only visited those nodes that they actually used for mapping).

- *Cost calculation example 3.*

For S_3 in $M^{(3)}$ the cost is

$$F_3(\{d_1,d_2\},M^{(3)},\{n_1,n_3\},\{25,25\}) =$$

$$= \frac{1}{2} \cdot \sum_{\forall n \in \{n_1,n_3\}} \Big(\sum_{i=0}^{|m_n|\cdot 15} \frac{1}{30+1-i}\Big)^2 \cdot \sum_{\forall n \in \{n_1,n_3\}} \Big(\sum_{i=0}^{\hat{l}_n} \frac{1}{50+1-i}\Big)^2$$

$$= \frac{1}{2} \cdot (0.4180 + 0.4180) \cdot (0.4414 + 0.4414) = 0.369,$$

and similarly for S_4,

$$F_3(\{d_1, d_2\}, M^{(3)}, \{n_1, n_3, n_2\}, \{25, 25, 10\}) =$$

$$= \frac{1}{2} \cdot \sum_{\forall n \in \{n_1, n_3, n_2\}} (\sum_{i=0}^{|m_n| \cdot 10} \frac{1}{30 + 1 - i})^2 \cdot \sum_{\forall n \in \{n_1, n_3, n_2\}} (\sum_{i=0}^{\hat{l}_n} \frac{1}{60 + 1 - i})^2$$

$$= \frac{1}{2} \cdot (0.1458 + 0.1458 + 0.1458) \cdot (0.2722 + 0.2722 + 0.0315) = 0.1259.$$

Now, considering the invalid configuration $M^{(4)}$, for S_3 the cost becomes

$$F_3(\{d_1\}, M^{(4)}, \{n_1, n_2\}, \{25, 15\}) =$$

$$= \frac{1}{1} \cdot \sum_{\forall n \in \{n_1, n_2\}} (\sum_{i=0}^{|m_n| \cdot 15} \frac{1}{30 + 1 - i})^2 \cdot \sum_{\forall n \in \{n_1, n_2\}} (\sum_{i=0}^{\hat{l}_n} \frac{1}{40 + 1 - i})^2$$

$$= (0.418 + 0.418) \cdot (0.8505 + 0.2012) = 0.8792,$$

and for S_4 we have

$$F_3(\{d_1, d_2\}, M^{(4)}, \{n_1, n_3\}, \{25, 20\}) =$$

$$= \frac{1}{2} \cdot (0.1458 + 1.0148) \cdot (0.5949 + 0.3161) = 0.5287.$$

In summary, the valid configuration gives the following costs 0.369 and 0.1259, while the invalid configuration with load-imbalance gives the significantly higher values of 0.8792 (penalizing lack of domain-disjointness) and 0.5287 (penalizing collocation) for service S_3 and S_4 respectively. □

Scenarios that involve dependability are presented in PAPER D, PAPER E and PAPER F.

2.4.4 Cluster-usage costs

Cluster-usage represents a type of cost different from the previously introduced properties. The model of the network is slightly extended with financial costs associated with using different clusters, and a deployment configuration is targeted that imposes minimal aggregated financial cost. At the same time, requirements from before, i.e. load balancing and dependability are to be adhered to. To facilitate this extended set of requirements, (20) is modified and a new term, denoted financial cost, is added to the function. First, let the total financial cost of using the nodes mapped in M be defined in F_F.

$$F_F(M) = \sum_{n_i \in q_1(M, \mathbf{C})} f_{n_i}^{(f)}, \tag{21}$$

where $f_{n_i}^{(f)}$ is the financial cost associated with using node $n_i \in N$. The helper function $q_1(M, \mathbf{C})$, when applied on the set $\mathbf{C}$, returns the set of nodes used for the mapping M. The new overall cost function becomes

$$F_4(|D|, M, H, L, z) = F_3(|D|, M, H, L) \cdot g(M, z), \tag{22}$$

in which $g(M,z)$ comes in two variants representing different weightings. The function is defined using a scaling parameter z and the sum of financial costs $F_F(M)$ as

$$g(M,z) = \begin{cases} g^{(l)}(M,z) = 1 + z \cdot F_F(M) & \text{linear weighting} \\ g^{(e)}(M,z) = 2 - e^{-(z \cdot F_F(M))^2} & \text{exponential weighting} \end{cases} \tag{23}$$

Setting $z = 0$ eliminates the financial costs from the evaluation of deployment mappings, returning to the original function in (20). Choosing $z > 0$, the influence of financial costs on the deployment can be scaled. The two alternatives in (23) represent a linear and an exponential increment in costs. With the more fine-grained exponential weighting mappings can be more balanced, avoiding under-utilization or overloading clusters, at the cost of introducing non-linearity in the cost function evaluation. Additionally, the exponential $g^{(e)}(M,z)$ is more complex, in that it has an upper limit, which is depending on the scaling parameter z, after which additional cluster usage ($f_{n_i}^{(f)}$) does not increase the financial cost part. That means that scaling, i.e. z, can be tailored to the service, so that financial costs increase rapidly where they have to, and the curve flattens out where additional resource usage is insignificant. Additional evaluation of how to take into account financial costs can be found in [Gul10].

Table 3: Deployment costs in $M^{(3)}$ and $M^{(4)}$ including financial costs

$F_4(\ldots) =$	Service	$F_F(M^{(3)})$	$F_F(M^{(4)})$	Cost of $M^{(3)}$	Cost of $M^{(4)}$
$F_3(\ldots) \cdot g^{(l)}(M, 0.1)$	S_3	6	10	0.5904	1.7584
$F_3(\ldots) \cdot g^{(l)}(M, 0.1)$	S_4	11	7	0.2644	0.8988
$F_3(\ldots) \cdot g^{(e)}(M, 0.1)$	S_3	6	10	0.4806	1.4350
$F_3(\ldots) \cdot g^{(e)}(M, 0.1)$	S_4	11	7	0.2143	0.7335

- *Cost calculation example 4.*

  As an example, consider the simple network again from Figure 12 and let cluster d_1 be financially expensive and cluster d_2 be a cheaper one and let $z = 0.1$. With concrete financial costs $f_{n_1}^{(f)} = f_{n_2}^{(f)} = 5$ and $f_{n_3}^{(f)} = 1$. Introducing financial costs will affect the overall deployment costs in the two scenarios $M^{(3)}$ and $M^{(4)}$ (Figure 13) as illustrated by numerical values in Table 3. $\square$

More tangible results for a cloud computing scenario are presented in Paper F. Figure 14 gives an intuitive presentation on how the four different terms – corresponding to domain-disjointness, node-disjointness, overall load-balancing and cluster costs (from left to right) respectively – are combined in $F_4(|D|, M, H, L, z)$. The terms in

$F_3(|D|, M, H, L)$ – ϕ_1, ϕ_2 and load-balancing – are illustrated by the red dashed lines. Whereas $F_2(M, H, L)$ is illustrated by the green dotted section, in which the quadratic load-balancing term is combined with a linear term representing communication costs. Each term (depicted as a separate simplified function, e.g. x^2 for a quadratic term) corresponds to one requirement, or in other words, constraint dimension. This contributes to an efficient step-by-step design to tackle the multifaceted problem at hand. An appropriate granularity for each dimension had to be found when designing each term. On the contrary, ensuring fast convergence dictates keeping the cost function as simple as possible, yet the output of the terms must be fine grained enough to rank the cost values of different solutions correctly. The cost functions are implemented in the simulator for the experiments included in this thesis. In an on-line implementation of the logic, however, evaluation of the costs using these functions is done by the agents, once in every iteration. Hence, the functions' computational complexity can affect overall efficiency.

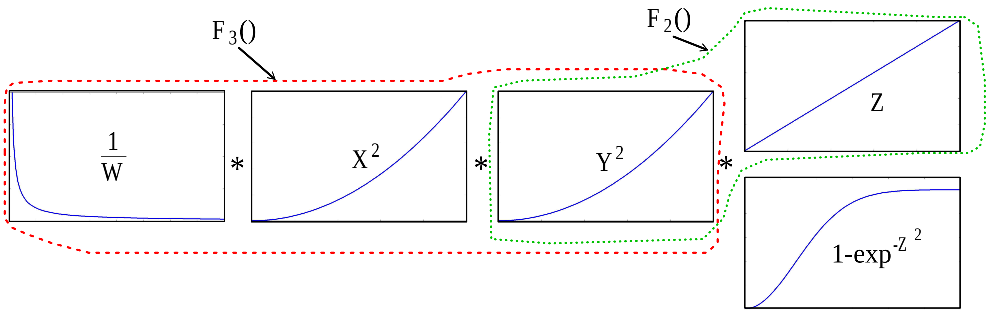


Figure 14: Multiplicative combination of the cost function factors in $F_4()$

In some cases, an abundance of possible mappings exists with different but very similar qualities, which may necessitate the application of non-linear cost function. An example to this is the version of (22), where exponential weighting is applied in (23), as illustrated by the lower version of the fourth term in Figure 14. The computationally more expensive exponential term can be used to obtain better influence on the costs of mappings by the cluster costs. Non-linearities can cause a slow down in the execution of the logic. Particular care has to be taken to weld components of the cost function together yielding a single function that is computationally effective and, at the same time, represents all the QoS requirements weighted properly by their importance. The various terms and combinations of functions are put into use in the stochastic optimization algorithms developed in this thesis. Their application is discussed in PAPER A – F.

3. Summary of papers

The main contributions of this thesis have been published in international peer-reviewed conference proceedings, except for PAPER C that is published in an international journal. As a guideline for reading, this *Thesis Introduction* part is recommended for a start giving a review on the background and the motivation for the work. For the papers in the *Included Papers* part a chronological order is a good choice as it also reflects the gradual developments in the research work. An additional paper in the appendix summarizes the research and is accepted for publication to an international journal at the time of writing. The papers are included as originally published, except that formatting has been adapted to this thesis. Moreover, the occasional inconsistencies in the notations applied in the different papers are due to the gradual developments in the algorithms and the method.

Generally, the papers can be categorized into four sub-topics:

- Category I. – Validation of mappings
- Category II. – Load-balancing and communication costs
- Category III. – Load-balancing and replication costs
- Category IV. – Cluster costs

The order and relations between the papers are shown in Figure 15.

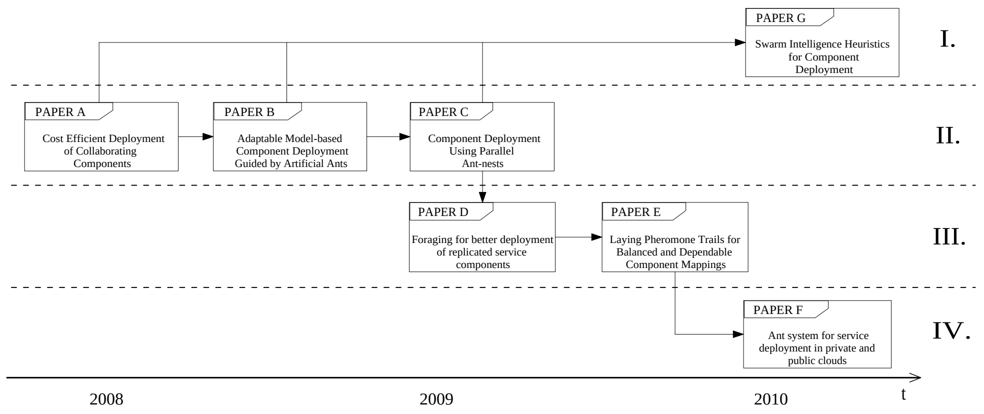

PAPER G
Swarm Intelligence Heuristics for Component Deployment
I.
PAPER A
Cost Efficient Deployment of Collaborating Components
PAPER B
Adaptable Model-based Component Deployment Guided by Artificial Ants
PAPER C
Component Deployment Using Parallel Ant-nests
II.
PAPER D
Foraging for better deployment of replicated service components
PAPER E
Laying Pheromone Trails for Balanced and Dependable Component Mappings
III.
PAPER F
Ant system for service deployment in private and public clouds
IV.
2008
2009
2010
t

Figure 15: Relationships and order between the included papers

In papers A – C focus is on services composed of software components and their adaptive deployment taking into account load-balancing and communication costs (category II.). These papers present research conducted together with the two advisers of the author Poul E. Heegaard and Peter Herrmann. Papers D and E follow with the different objectives of component replica management and considering dependability of their deployment (category III.), in which case Hein Meling also contributed to the research conducted by the author. A move from software components to VM instance deployment is achieved in paper F (category IV.), where certain financial costs are also taken into account. Lastly, paper G presents a centralized approach (category I.) with the main purpose of establishing a framework for validating the results obtained with the CEAS-based method introduced in this thesis. However, the results published so far in category I. are limited to problems from category II. as indicated. The journal paper in the appendix touches upon all four categories and summarizes the work. A brief summary of the included papers follows.

PAPER A

Cost Efficient Deployment of Collaborating Components

The problem of efficient deployment of software components is introduced in the first paper. In other words, finding the optimal mapping of components to physically available resources, while satisfying non-functional requirements is set as the main goal. A traditional task assignment problem is adapted to the service engineering context. The original problem used as an example in the paper has been solved using various methods by previous authors, referenced in the paper. One particularly relevant solution has been obtained using the centralized Cross Entropy method, which is used for comparison as well. The paper takes a new approach, which can be viewed as an intersection between software engineering and network management. A new distributed stochastic optimization method based on CEAS is presented and its place within the software development cycle is identified. Simulation results using the example model are presented and verified using the known, global optimal solution to the original problem.

PAPER B

Adaptable Model-based Component Deployment Guided by Artificial Ants

A robust and adaptive service execution platform is targeted by placing the deployment logic inside the software development cycle. Support for run-time redeployment of components for keeping the services within the allowed region of parameters is investigated. Examples of tangible service models embellished with non-functional requirements are used for the investigation. Practically, the paper demonstrates that the deployment logic is capable of handling multiple services simultaneously when the suggested extensions are applied to the algorithms presented in the preceding paper. More precisely, it is shown how the need for global knowledge can be eliminated from the method mainly via load sampling and applying an improved cost function. Also, adaptation capabilities are evaluated through scenarios where simple changes are injected to the network, such as node failures. Simulation results are shown for demonstration.

PAPER C

Component Deployment Using Parallel Ant-nests

Previously, global knowledge remaining within the decentralized algorithm was eliminated. In this extended journal paper version of the preceding paper the potential of the deployment logic for providing adaptation support in changing environments is further elaborated. In particular, cooperation between species executed in parallel and working for the same objective is highlighted. A discussion is included on how pheromones can be shared by ants from the same species but emitted by different nests. Furthermore, the place for triggering placement of services is identified as the nest, emitting ants for each species corresponding to the service. To free the deployment logic of single points of failure, replication of the nests is suggested, this way increasing reliability of the approach. Simulations show that parallel nests can interoperate efficiently in obtaining deployment mappings.

PAPER D

Foraging for better deployment of replicated service components

The new dimension, dependability is introduced in addition to performance requirements. First, the network model is extended with the notion of domains that represent groups of nodes based on administrative, geographical or other considerations. Second, a set of rules is defined related to dependability, and in particular, concerning management of replicas. Thus, service models are introduced containing replicas of components for the purpose of increasing dependability. A discussion is included on the efficiency of various combinations of cost function elements that are used to guide the heuristic optimization method to provide better deployment mappings. As an experimental setting a scenario is introduced including 15 services each with different redundancy levels, and deployment is simulated over a network comprising 10 nodes partitioned into 5 domains. Satisfaction of dependability rules is evaluated while load-balancing is to be maintained in the example network.

PAPER E

Laying Pheromone Trails for Balanced and Dependable Component Mappings

Replication management in large scale, clustered networks requires the deployment logic to be scalable to handle a significantly higher amount of replicas and nodes than in the preceding examples. Previously, various cost function designs were evaluated that take into account replication management in addition to load-balancing. A further refinement of the best cost function chosen earlier is included. Another important aspect of the deployment logic and CEAS at its core is elaborated, the pheromone tables. In particular, how to efficiently represent sets of component replicas in the distributed pheromone database to achieve better scaling is one of the major contributions of this paper. Accordingly, three different encodings are evaluated. Results are demonstrated on a larger example scenario than previously, and robustness of the approach to network partition failures is examined using the example.

PAPER F

Ant system for service deployment in private and public clouds

Virtualization has become a key enabler in large-scale computing platforms that may provide services for thousands or even millions of users through private networks or the Internet. This technology promises significantly better utilization and so-called elastic scaling to better meet and adapt to changing demands. However, existing technologies to enable this kind of scaling are based on hierarchically managed approaches that do not necessarily scale equally well. A significant challenge is to equip management systems with the capability to handle load-overshoots at times when increased demand cannot be provided for locally. This requires interoperation between networks of nodes provisioned by separate organizations, a concept referred to as federation of clouds, or hybrid, i.e. private and public clouds.

This paper conjectures that self-organizing techniques can very well be applied to (re)configure virtualized systems in hybrid cloud environments. Specifically, the paper suggests that the component deployment approach presented in previous work can be applied and tailored to the problem at hand to obtain efficient deployments of virtual machine images. In addition to replication and load-balancing, the objective of the optimization process is extended with financial costs that model usage of resources in public clouds over the Internet. The network model is extended to represent the notions of public and private clusters, which have different access rights depending on the perspective of a service. An example scenario is presented and simulation results are obtained for the method.

PAPER G

Swarm Intelligence Heuristics for Component Deployment

There are different options for validation of the deployment logic that was presented in the preceding papers. Many suggest a comparison between different kinds of bio-inspired methods applied to the deployment problem. This, however, is particularly difficult to implement and conduct quantitatively due to the very few available related work on bio-inspired methods for deployment decision making. Existing bio-inspired methods (and other types of heuristics as well) need to be tailored to the specific deployment problem to be able to execute them using example models and compare results with the results obtained using CEAS. Often, other heuristics require fine tuning of a plethora of parameters to execute efficiently, and to obtain a representative comparison between methods. Instead, a centralized, offline and exact solution method was chosen to obtain optimum solutions for example scenarios.

Optima in deployment scenarios are difficult to obtain, especially in large-scale environments and are intended to be used as lower bounds of possible deployment mappings. The ILP model presented in this paper is centralized, offline, exact and uses global knowledge to find optimum mappings, as opposed to the CEAS-based logic, which is a decentralized heuristic algorithm with no global knowledge. Using the optima obtained with ILP the output of the heuristic algorithm can be validated by evaluating how close the cost of the deployment mappings can get to the lowest cost possible. In this paper, 3 examples are presented for comparison, where deployment mappings are obtained by both the ILP and executing the deployment logic in a simulation. The study concludes that the heuristic deployment algorithm finds mappings with optimal or near-optimal costs with low variation for the included examples.

THESIS APPENDIX

A Bio-inspired Method for Distributed Deployment of Services

The last paper, included in Part III is summarizing the results of this thesis and touches upon each of the application scenarios of the deployment logic that has been investigated. The paper starts with the general problem setting and its complexity. Then the deployment logic is introduced and some of the examples from the previous papers are discussed, together with the validation approach.

4. Related work

The intention of this section is to review some of the related research directions that can either be considered surrounding the deployment mapping problem or contributing to some crucial parts of an anticipated solution to the problem. Works on methods, different approaches and related scenarios are reviewed in the categories below. Each section starts with a short paragraph introducing the relations of the topic to the research presented in this thesis.

The list of categories starts with a novel service development method, which could be extended with the deployment logic presented in this thesis. This would produce a complete service development and deployment cycle. In the first step of this cycle, service models have to be enriched with QoS-related information. This can be achieved by requirement capturing, discussed after the SPACE method. When changes in the execution context necessitate adaptation, the service and its components have to be reconfigured. To allow for adaptation various frameworks have been introduced that are aware of the execution context to a certain extent. Component-based service models and their reconfiguration, as well as relevant frameworks and middleware are touched upon below.

The QoS might be impacted by the deployment configuration of a service. Different aspects of QoS might be considered, such as balancing of the load imposed on the hosting environment or replication management, as discussed in this thesis. Power-usage modeling and management is another interesting aspect to consider. Relevant work with respect to these QoS dimensions is discussed in two subsequent sections. One of the relevant target environments for the deployment logic, where these aspects are important is virtualized data-centers, also touched upon in this section.

Approaching the deployment problem from a theoretical perspective, a group of similar problems, labelled task assignment problems are also discussed below. Migration of some or all parts of the services is required for adaptation, hence, different migration techniques are reviewed subsequently. Lastly, a set of relevant bio-inspired methods are discussed, and, in the last part, various deployment decision methods are discussed and compared qualitatively to the work presented in this thesis.

Arctis and the SPACE method

First a service development method is introduced, which describes a complete software development cycle. The deployment logic presented in this thesis is a candidate for a natural extension to the cycle, supporting autonomous (re-)deployment of services developed using this framework.

A method for specification by activities, collaborations and executable state machines (SPACE), devoted to the rapid and (by design) correct engineering of distributed services is introduced in [KH10]. A tool-chain supporting the concepts of the SPACE method, called *Arctis*, is also available and supports the overall stepwise development process. The development cycle starts from a functional model of the service, specified by UML collaborations and activities. Thus, the specification encapsulates – efficiently – distributed sub-functionalities that together interact and implement the service behavior. This specification style and *Arctis* facilitates reuse of building blocks to a significantly higher degree than it would be possible with component-based descriptions [KH09]. Building blocks (e.g. automated creation and transfer of SMS, user authentication mechanisms, etc.) are stored in collections of domain specific model libraries and can often be reused in various services while single components tend to have specific layouts for a particular application. Next in the development cycle, correctness checks are performed on the functional models, followed by a transformation to a component-oriented design model [KSH09]. An approach for integrating security aspects into the system design at the functional level has been proposed in [GHK09]. The component-oriented models, specified as UML state machines, are used to generate the executables by code-generation. This completes the automated development cycle from collaboration-oriented service models to executable implementations [Kra08].

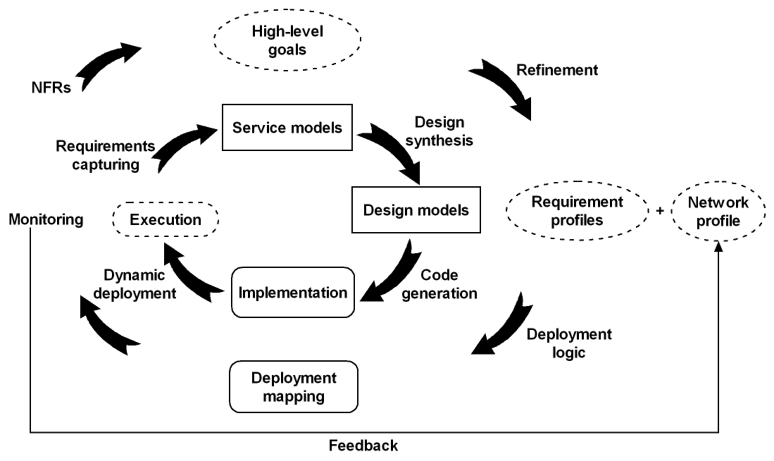


Figure 16: Development and deployment with SPACE

The implementations generated by the rapid development cycle of SPACE need to be deployed into a – possibly dynamic – execution environment. This is the step where the software life-cycle can benefit from autonomous deployment decision support. The suggested proceeding to integrate the deployment logic to the service engineering cycle is shown in Figure 16, where the inner cycle corresponds to the original SPACE cycle and the outer cycle is the suggested extension to it.

To enable deployment support service models can be amended by *high-level NF goals* encapsulating non-functional requirements (NFRs) of the service in a rather abstract manner. Refinement of NF goals can be done in parallel with the transformation of service models to design models. *Requirement profiles* obtained after the refinement step specify NFRs of the service components. Additionally, a *network profile* is used, representing the properties of the target environment, in which the service will be executed. The two profiles serve as input for the deployment logic, *requirement profiles* specifying the objective and the *network profile* specifying the search space. Beside aiding initial deployment of an implementation, another aspect is the dynamic re-deployment of a generated implementation. Deployment support can extend the regular service engineering cycle to support re-deployment, depending on changes in the execution context, as well. To support adaption, according to the MAPE control loop discussed in Section 1.3, feedback is required that can be provided by observing the execution environment via monitoring.

Requirement capturing

The step of capturing requirements is an important phase in the development cycle. NFRs have to be captured early at design time for deployment decision making as well. Hence, requirement capturing methods are relevant and are discussed in this section.

High-level goals were used in AI-research to guide agents already from the early '70s [Nil71]. A comprehensive review of goal-oriented engineering of requirements was discussed in [Lam04]. Using high quality NFRs is a factor contributing to the success any software project. A well elaborated requirement capturing approach is the so-called User Requirements Notation (URN) language, which claims to be the first standardization effort towards explicitly capturing NFRs throughout the design process [AM02]. Beside graphical notations, contract-based descriptions of QoS goals are used to specify NFRs in a machine-processable format in ***QML*** [FK98] and in ***CQML*** [Aag01]. Concerning a description for the execution environment, the same object paradigm can be used that UML employs to reduce complexity. In [TM05] a language compliant with the object paradigm is proposed intended to specify QoS in networked environments, which can be a candidate method to represent the ***net-map*** in a formal way. In most cases, annotations are used to establish an intermediate model that can be transformed to traditional performance models such as queuing networks, quasi birth-death processes, or colored petri nets. For example, in [FW04] three views of the system are combined to analyze performance of a distributed system: scenarios, the software and hardware structure, and available resources. Recurring patterns in resource architectures are identified and layered queuing models are used to analyze them. Furthermore, [FW04] suggests the use of annotated UML diagrams in the following manner: deployment diagrams to represent the allocation of resources, and activity diagrams or sequence and collaboration diagrams for the scenarios. Deployment diagrams with fault tolerance patterns are used extensively in [TSG04] to specify deployment restrictions. These restriction can then be reused in architectural design and a constraint solver is suggested to produce a deployment that satisfies the restrictions. Activity diagrams are suggested in [MEM$^+$09] for expressing system requirements by domain experts. These experts also define a cost function governing service selection in the suggested framework.

Component-based models and reconfiguration

The simplest level of adaptation can be achieved on the level of components. Many approaches have been suggested for enabling runtime reconfiguration of software considering components as the appropriate building blocks and level of granularity to start with. The deployment logic presented in this thesis addresses component level adaptation in the earlier papers, PAPER A, B and C. Relevant works for component-based reconfiguration are discussed in this section.

Dynamic reconfiguration of the system architecture is an example of adaptation at the design level. Decisions can be made statically (exhaustively defined configurations) or dynamically (during execution, based on high-level goals) [CLG$^+$09]. Both static (e.g. model checking) and dynamic analysis (e.g. simulation) and evaluation of components are essential for software modeling. Considerable effort has to be spent on understanding the environment a component may find itself in [Egy04]. An extensible environment, RAJE allows monitoring of resources, e.g. memory and CPU time consumed, in Java threads at runtime. Additionally, components can use contracts to describe the context they desire to be run in [Som04]. The concept of QoS-aware components, Q-components that dynamically compose Q-applications was introduced in [MRG04]. Q-components find and negotiate their corresponding service pairs based on a central directory service and some queuing network analysis. Minimizing the burden on the developer of components is targeted specifically in [CS02], by introducing a framework offering mechanisms to analyze and handle the interactions between components during reconfiguration. Moreover, support for dynamic reconfiguration of Java-based applications and components has received considerable attention in a variety of works, such as [Hal04, ATK05, PKF05]. Implementation design using UML models, in particular deployment diagrams, was discussed in [FSJB01]. The role of implementation design is to express the decisions made regarding hardware and software for an abstract system. Adaptation policies are generated systematically using Markov Decision Processes in [JHS$^+$04]. In general systems with multiple components, however, non-linearities make the formulation of system properties difficult, e.g. component failures are rare events. Hence, this deterministic approach suffers from state-space explosion when performance and dependability metrics are combined.

Context-aware and adaptive frameworks and middleware

Component-based reconfiguration is an attractive target for adaptation support. To enable this type of reconfiguration, however, some type of system-level support is needed. This support can be provided by a layer underlying the executed software components, in other words a middleware. Various middleware approaches and context-aware frameworks are candidate techniques that can effect the changes suggested and provide information about the environment needed by the deployment logic. Relevant middleware and frameworks are discussed in this section.

Extensive research has been conducted in recent years focused around platforms supporting various forms of adaptation and increased dependability. Autonomous replication management is targeted in a framework based on group communication systems, using which, a self-managed fault treatment mechanism that can adapt to network dynamics and changing requirements is devised in [Mel06]. Another distributed and dynamic middleware, ***QuAMo-***

bile [LSO+07] introduces independent application variants, where the appropriate instance is selected to provide context-awareness and adaptation. Service level agreements (SLAs) are commonly used as starting points for planning. A planning-based adaptation middleware was devised in the MUSIC project [REF+08], where, similarly to our approach, QoS related metadata was used in the service models. SLAs are applied as means for building policies in [ATZ07] as well. Based on the policies resource allocation is enabled in an autonomic environment. A middleware, *CARISMA*, employing the peer-to-peer concept is described in [CEM03]. *CARISMA* utilizes an auctioning-like mechanism for conflict resolution and adaptation that is triggered automatically by changes in the context. In [MRKE09] reliability of mobile software systems is improved by considering information from a variety of sources. These sources include not only execution logs, but software architectural models, execution profiles, information regarding the context and domain expert knowledge. Then, reliability predictions are made by learning algorithms, such as Hidden Markov Models and Dynamic Bayesian Networks. According to the predicted reliability configuration of the software is adapted using a middleware approach.

The *SmartFrog* framework, developed by HP Labs [Sab06], targets the description, deployment and management of distributed service components. This framework describes services as collections of components and applies a distributed engine comprised of daemons running on each node. A dedicated description language is used for specifying configurations, using which deployment information can be described and instantiated across a network. After instantiation, components of the application are hosted by the framework itself. The different configurations that can be instantiated are based on predefined and complex templates. The decision making mechanism that develops the best configuration to be instantiated, however, is open for research. The different kinds of middleware discussed are possible candidates for serving as a means of instrumenting deployment guided by the algorithms presented in this thesis.

Load-balancing and replication

As discussed in Section 3, PAPERs A – C focus on the QoS goal of balancing load imposed on the network by the deployed services, from the perspective of the deployment logic. In addition, deployment of replicas for increased dependability is investigated in PAPERs D – F. Hence, it is relevant to look into related techniques.

Load-balancing and replication are two important concepts in provisioning networked services. With load-balancing, often by applying dedicated load-balancers, workload can be distributed evenly across particular resources, e.g. network links, nodes, CPUs, etc. More efficient distribution of workload contributes to better resource utilization, throughput and response times, while also playing a key part in avoiding load-peaks. Different types of algorithms have been devised for general load-balancing problems. Classical solutions, such as Round-Robin and weighted Round-Robin or shortest job first scheduling were developed decades ago [Kle76]. More recent approaches include gradient models [LK87], economical and game theoretic approaches [GC02] and evolutionary algorithms based on ant colonies [HE07, MMB03] or intelligent bees [SCK08].

Another aspect that demands the use of multiple components is reliability, which can be increased through additional redundancy, or in other words, replication. There are several approaches to manage replicas, i.e. passive, semi-passive, active, semi-active replication [Df00]. The service models discussed in this thesis follow an active replication approach, which is also often referred to as the state machine approach [Sch90]. In the state machine approach all replicas perform exactly the same actions in the same order, this way remaining identical. This, however, requires that all replicas receive the requests in the same order. Thus, the state machine approach implies that a consistency protocol must be run among the replicas, and hence introduces larger communication and processing costs. Regarding the additional communication overhead of executing the required protocols, the notion of message complexity can be used, which is typically expressed by the number of messages generated by a failure-free run of the algorithm. Given a desired replication-level p, message complexity of the algorithms applied in common applications with replication is $o(p)$ in point-to-point and typically better than $o(p)$ (can be constant) in broadcast networks [Df00]. These considerations lead to the basic service models with replicas, introduced in Section 2.4.3.

Regeneration of crashed replicas appeared in the early Eden system, presented in [PNP88]. More recent distributed group communication systems, such as [MG08, MMHB08], can provide automatic reconfiguration and regeneration of replicas. The DARM framework [MG08] is one promising candidate for providing efficient deployments based on the algorithms in this thesis. Further results related to regeneration to remedy failures using replication are discussed in the context of peer-to-peer wide-area storage systems in [YV05]. Moreover, in [YG09] it is shown theoretically that inappropriate placement of replicas of inter-correlated objects can significantly impact system availability. Importantly, consistency, or replication protocols with arbitrary communication and computation costs can be taken into account in the system models used in this thesis as initial input for deployment mapping.

Power saving

Beside the QoS dimensions considered in the included papers, a natural extension to the set of requirements is to include power-management limitations of service deployment.

Recently, powerful economic factors appeared that are pushing information technologies in general towards lower energy usage. Consolidation of resources and better utilization are targeted, especially in large-scale data centers, to conserve power. Currently, energy consumption and related carbon emission of IT services are described as being equivalent or above the corresponding values for the airline industry. However, adopting new technologies might help making systems more efficient with respect to power saving. Energy costs, including the costs of extensive cooling, are among the main financial expenditures of operational server farms. The advantage of current data-centers hosting virtualized applications is that these applications can be adapted relatively easy, taking into account energy-saving aspects. This, however, requires adaptation of each individual component of the application [BGG$^+$10]. Hence, fine-grained models including the interactions between the components are also required.

Dynamic voltage scaling (DVS) is a technique often addressed in various works trying to minimize power usage. A thorough example for optimization, in particular minimizing global power usage of a data-center constrained by end-to-end delays is the topic of [HASL07]. The DVS technique was also applied targeting the trade-off between application completion time (task scheduling) and energy consumption. An algorithm taking into account both goals, and balancing between them effectively, has been proposed in [LZ09]. The authors of [SZL10] have the same targets of effective scheduling and minimizing energy usage in a grid scenario, but taking into account that energy usage has to be minimized simultaneously in parts of the grid having different ownerships. Effective scheduling can additionally be used to minimize energy consumption of memory banks as well [MB08].

One approach for minimizing total power consumption in wireless networks by adjusting transmission power according to a distributed protocol, applying centralized techniques locally only, was presented in [MG05]. A framework capable of estimating the energy consumption of Java-based applications, both during design and at runtime, was developed with a margin of estimation error below 5 per cent [SMM07]. Simplifications are justified in this approach by assumptions related to the Java Virtual Machine.

A useful optimization target in data-centers is the distribution of available power among the servers so as to get maximum performance. Generally, assigning more power to a host allows higher CPU frequencies, hence, results in faster execution in that host. On the contrary, running servers at their lowest possible power levels, allowing for more servers to be turned on within a given power budget, might also be an objective. Accordingly, the optimal power allocation depends on, among many other factors, the application scenario. [GHBDL09] gives a theoretical model, based on queuing networks, to analyze different power-to-frequency relationships and power allocation to nodes.

Virtualization and cloud computing

From a component-level deployment approach taken in PAPERs A – E focus has been shifted to services provided in a virtualized environment in PAPER F. Hence, an introduction to some of the key techniques in virtualization and cloud computing is provided in this section.

With the increasing demand for platforms supporting large-scale distributed systems data centers have undergone significant changes recently. Virtualized infrastructures have been introduced with the aim of achieving better utilization of resources while hosting services and with the introduction of on-demand scaling. Multiple, geographically dispersed data center sites have become a typical scenario. Enhanced availability despite the ever-present outages and quicker response to local demand are among the promises. Mayor players, such as Amazon, Google, Salesforce or Yahoo!, offer public access to large amounts of resources previously unavailable for smaller enterprizes. The concept of providing infrastructure level access to resources for virtual servers is popularly called cloud computing. Large data centers providing the infrastructure as a service are typically assembled of a high amount of cheap and less reliable blade servers, racks, hard disks, routers, etc., which might lead to increased failure rates [Dea]. Increased failure rates in turn, demand replication and repair mechanisms.

The complex requirement of reliable provisioning of services with efficient utilization of resources is one key selling point of cloud based Infrastructure-as-a-service type offerings. It is even more challenging, however, to feature the so-called elasticity property, in other words the capability of dynamically scaling the amount of available resources up or down according to the changes in demand. Dynamic service delivery is facilitated by the use of Virtual Machine (VM) Images deployed on demand in data centers. Multiple data centers under a single administrative entity are referred to as a computing cloud. VM images can be packaged in standardized ways by using for example the Amazon Machine Image format [LLC]. Also, standardized formats for packaging enable interoperation of different types of clouds covered under the terminology of hybrid-clouds, where typically private computing infrastructures are connected on demand to public cloud offerings. Organizations implementing the private cloud concept may be running different flavors of operating systems equipped with cloud support, such as Ubuntu's Enterprise Cloud solution [WGB09], which is compatible with the Amazon EC2 packaging format, thus allowing for migration of VMs into public clouds.

The intention of the Eucalyptus project, for example, is to support multiple cloud computing interfaces while preserving the back-end infrastructure [NWG+09b, NWG+09a]. Lack of service management facilities and interoperability between cloud providers have been identified as major obstacles limiting scalability of federated cloud computing environments [RBL+09]. Federated cloud environments need a unified interface for dynamically managing VMs, forming services. Moreover, a heterogeneous cloud computing architecture must also tackle the placement, migration, and monitoring of VMs across interoperability boundaries. A recent initiative [EL09] proposes mechanisms for placement, migration and monitoring of components. Virtualized environments and server consolidation generally achieved significant advances recently in the autonomic computing community. Effective deployments, however, require some kind of optimization approach, in addition to methods easing the actual execution of placement. Systems discussed above can provide the underlying mechanisms that are necessary to support service deployment in cloud environments. Focus in these approaches is mostly on the mechanics of VM deployment and not on the process of optimizing the deployment configurations. Few frameworks have been developed that target optimal placements for VMs under a variety of constraints, such as [VAN08, JHJ09]. The problem of hosting virtualized multi-tier applications is addressed from the provider's perspective in a self-adaptive capacity management framework in [CAAS07]. The system is modelled as a tandem queuing network of simple M/M/1 queues, however, without considering interdependencies.

Another aspect is virtualization of the networking resources for better utilization, which has been hailed as a key enabler of the evolution to future Internet technologies. The major building blocks of network virtualization have been identified as: i) application specific routing overlays, ii) consolidation of resources (OS virtualization) on a generic infrastructure and iii) efficient exploitation of network diversity [TZNTG09]. The problem of finding the optimal mapping between specific virtual network elements and a physical substrate has been briefly analyzed within the extent of the 4WARD project, results were summarized e.g. in [CJ09]. Future research directions were identified including optimization mechanisms, dynamic reallocation of network resources, managing virtual networks spanning across multiple infrastructure domains and interoperability of different virtual networks, which are all problems remaining to be solved. The applicability of virtual machine technology in routers is investigated in [BFdM09], targeting special features in virtualized networks. Moreover, future home environments are used as a proving ground for network virtualization in [BWS$^+$09]. State of the art in network virtualization and challenges have been sketched in [CB09].

Task assignment problems

To approach the deployment problem from an optimization perspective the first set of similar problems to look into is denoted task assignment problems, discussed in this section.

General task assignment problems are fundamental in combinatorial optimization and have several slightly different formulations, such as different transportation problems, finding the minimum cost flows in a graph or finding the maximum weight match in a bipartite graph with weights. In a simple task assignment problem there are agents and given tasks to be performed by them. Assignment of a task to an agent incurs some cost, which usually depends on the agent. Then, a given solution is an assignment between agents and tasks so that all tasks are performed and the total cost is minimal. Several algorithms have been devised to solve the basic version of the task assignment problem in polynomial time. During an early stage of research, conducted within the extent of this thesis, task assignment problems were studied with focus on scheduling software processes or modules on execution resources. In module assignment, however, the problem quickly becomes more difficult when communication between the modules is also allowed. Complexity of the general module assignment problem is NP-complete, as shown in [FB89].

A seminal paper, [Efe82], dealing with the problem of clustering software modules and assigning them to processors had large influence on the research and results of this thesis. The module assignment scenario presented therein – to introduce a suggested heuristic algorithm for the module assignment problem – was transformed to a collaboration-oriented service model with execution and communication costs in PAPER A. This example, even though it does not have a tangible meaning as a service, served as a reasonable test scenario, while the problem complexity remained NP-hard. Also, the transformed scenario was useful for validating deployment algorithms, as the solution of the example is well known. This scenario has been investigated in [WN04] too, and a comparison of results with some other approaches, such as a solution with *minmax* node costs in [WM93], was conducted. Besides, the example was reused as one of the test models to build the ILP in PAPER G.

Task, process and workload migration

When the deployment logic indicates that a more favorable configuration exists for the service, most probably parts of the service have to be moved to new locations. This is the point when migration techniques are required. A brief review of relevant methods follows.

Given a method that finds efficient and at least near-optimal deployment mappings for services, one of the biggest issues that needs consideration is how to migrate arbitrary parts of the service to allow redeployment suggested by the logic. This is a question discussed in many places in software migration and mobile agent scenarios. It is especially interesting to consider moving instances that have open connections, which have to be preserved even after migration. Mobile agent technologies have been investigated in the literature intensively, most of them focusing on agents implemented in Java. These agents are defined as programs that can move across nodes carrying their own code mainly for the purpose of avoiding data transfer over the network. Furthermore, these techniques allow having, for example, instead of a fixed client that retrieves massive amount of data from the server side and processes it, a client component that is moved to the server for the processing period for the single purpose of conserving network bandwidth. In other solutions the client-code is available everywhere, only state and data are moved and execution can continue. The majority of available approaches considers moving runtime state of instances only and does not handle open connections. Runtime state consists of state information and data. Furthermore, two types of migration are considered, strong and weak, where strong means continued operation at the new node after being transferred, and weak means data is transferred but the state machine (SM) of the agent is restarted from initial state. However, restart from an initial state is an option only if such a SM is meaningful.

Approaches focusing on moving components with state and data but not considering connections use different ways to achieve the same goal, to capture state during execution at regular intervals. Saving the state regularly can be done automatically or explicitly (function call in the code). The two main categories are modifying the Java Virtual Machine (JVM), or pre-processing the source code to insert state-capturing statements. Serialization might miss information from the JVM, such as method call stack, variables local to the method and the program counter. If we look at the migration of connections of a component most of the approaches deal with TCP connections. One way, followed in some methods, is using OS kernel tricks to access data not yet delivered. Others extend the TCP protocol itself. New states e.g. SUSPENDED, RESUME_REQ, etc., are introduced. This is a middleware approach, in which case the middleware takes over and closes/creates connections. More importantly, a new buffer/queue is added, which collects undelivered data. This queue is migrated together with the rest of the data, and is read first after migration. However, only one side of the connection can move at a time, since the migrated part resumes the connection and has to know the address of the original party. In addition, this involves exchange of messages during the migration period for the extended TCP states [ZXS04]. Migration of Java-based mobile agents has been achieved on the language level by the pre-processing approach. In this case, automatically inserted code saves runtime information and state and re-establishes them on restart [Fn98].

Wrapping in the TCP layer by two layers, one above and one under, and logging the TCP connection state is an alternative approach [ZMAB03]. No modifications to the TCP protocol itself are required and the application above TCP is not affected. Another TCP extension allows migration of the server endpoint of a client-server connection, and also has a specific API with functions that need to be called from the application such as *export_state*, *import_state*. Nevertheless, the originating server must stay alive for connection state delivery to the destination server and only a single process per connection is allowed, otherwise the connection state cannot be collected. For example, an HTTP server and multiple CGI worker processes are not allowed [SSII01].

SockMi allows moving both sides of a connection and any connection state is allowed, i.e. an unconnected socket is interesting especially for server-side use (e.g. migrating a socket in *listening* state). An API and a loadable kernel module is implemented in the approach and a daemon is used that basically moves the TCP connection state and data together with the receive/transmit queue. The daemons communicate and allow moving of the socket among them. Importantly, the approach uses Network Address Translation (NAT) at both nodes involved in the migration, thus the exporting node redirects packets and the importing node communicates using the IP address of the original exporting node during the connection. This mechanism leaves the question of long-lasting connections, during which several migrations can happen, unanswered [BCT07]. Mobile IP, even though well developed, is not suitable because it relies on a home agent (proxy) that does not change location and relays all packets to the migrating endpoint practically continuously [ZXS02].

Migrating workloads is equally important for data center applications as component migration. Therefore, near optimal workload placement based on evaluating traces is suggested in [GRC+08]. Authors suggest migrating workloads from overloaded servers and shutting down lightly-loaded servers. Several trade-offs are identified, such as between the required capacity and power usage, access QoS for CPU and memory, or the number of migrations. Furthermore, a genetic algorithm is used to analyze alternative workload placements and fuzzy logic is utilized to control migrations. The difficulties of inter-site migration (between different providers) are highlighted with a suggested solution for storage migration in [HON+09]. Complexity of placement of application data is highlighted in [ADJ+10], where the authors address the issues of shared data, inter-dependencies between data chunks, application changes and user mobility. The suggested solution is based on analyzing large, month-long traces and applying an iterative optimization algorithm. The trade-offs that have to be considered include minimizing inter-data-center traffic by co-location while minimizing latency by placing data close to the anticipated users. Sizes of these types of data placement problems are exploding with the millions of clients in clouds, and standard exact optimization methods simply do not scale to that extent.

Bio-inspired methods

The theoretical basis of the deployment logic presented in this thesis is provided by a bio-inspired AI approach, introduced in Section 2.1. In recent years, there has been plenty of research effort put into various approaches with theoretical background in processes observed in nature. This section gives a brief overview over the relevant bio-inspired approaches.

There are undoubtedly trade-offs in applying bio-inspired techniques compared to conventional, centralized alternatives. Hence, the applied methods have to exhibit improved efficiency and responsiveness to unpredictable fluctuations on the long-term to compensate for the possible increase in delays and waste of resources. Decentralization, however, is inevitable in autonomic systems, where any centralization of resources or control can hinder the systems's ability to adapt. Bio-inspired methods play an increasingly important role, because the concept of autonomic computing itself has its roots in biological models, such as the nervous system, swarm intelligence and artificial immune systems [DDF+06].

The heuristic optimization algorithms presented in this thesis employ a variant of ACO systems, the CEAS. The concept of ACO was introduced in [DMC96] as a multi-agent system for solving a wide variety of combinatorial optimization problems. ACO, being a bio-inspired method, defines agents' behavior based on a theoretical foundation resembling the foraging behavior of ants. The method has been applied to a variety of problems in communication systems, including load-balancing [S^+97, MMB03], routing in wired [CD98], and wireless networks [CDG05]. In [BMM02], a framework is presented to support the design, implementation and evaluation of peer-to-peer applications that are employing evolutionary programming techniques resembling complex adaptive systems.

The random proportional rules, similar to the one presented in Section 2.1, have been common already in early ACO systems and are present in other swarm intelligence systems, such as AntNet [CD98], Termite [RW03], or Beehive [WFZ04], as well. These rules have been shown to resemble swarm behavior closely through empirical studies, and have been recognized as fundamental for model based search [Z^+04].

The particular ACO type method applied, the CEAS, however, differs in its formal foundations from traditional ACO methods, in that it uses the CE method, first introduced in [Rub99]. Successful applications of CEAS include path management strategies, e.g. shared backup path protection [WH04], p-cycles [WHN05], search for available resources restricted by QoS constraints [WHH03], and finding adaptive paths with stochastic routing [H^+05]. Implementation issues and related trade-offs, such as management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism [H^+04] and self-tuned packet rate control [HW06a, HW06b]. Additional reduction in the overhead is accomplished by pheromone sharing [KWH08], where ants with overlapping requirements cooperate in finding solutions. The objective, when applying CEAS is often to minimize some cost determined by a given cost function. The control variable, denoted the temperature in Section 2.1, decreases gradually in these cases, in a similar way to the well-known approaches labelled simulated annealing [KGV83]. The gradual decrement of the temperature reduces the probability that the heuristic gets stuck in a premature and far from optimal solution.

Instead of applying evaporation to the pheromones a population-based approach is suggested by the authors in [GM02b], resembling mechanisms from genetic algorithms. The corresponding population update strategies were described in [GM02a], and were applied to multi-criteria optimization problems [GM03], employing one pheromone matrix for each optimization criterion. Furthermore, a multi-colony approach, where every colony has one part of the objective as a task, was suggested in [MM99]. The concept of spy ants that can look into the pheromone databases of other colonies as well was defined and experimented with in a study [DHR01]. Instead of a population of ants for each criterion weights are used by ants before accessing the pheromone database and solutions are constructed according to a weight vector also in [DGH$^+$04]. A different approach, an artificial hormone system is explored for the problem of task allocation in grids by the authors of [BR09]. An artificial hormone system works also decentralized similarly to ACO and can be characterized by processing cells that communicate by exchange of hormone messages (accelerators and suppressors). In the hormone-based control loop, however, messages have to be exchanged between all participating processing cells, which implies that the volume of required communication might hinder scalability.

General deployment decision making

The main focus of this thesis is on methods to aid the deployment of distributed software services. This section reviews some relevant approaches that target finding, optimizing, or planing deployments in various application scenarios.

The general objective in deployment decision making is to maximize the utility of the services at hand via influencing their deployment. Available approaches differ in their main concepts and in the view taken on the services, i.e. what types of services are considered and how they are modelled. Generally, the different approaches can be categorized based on (i) the time of execution, i.e. online or offline; (ii) the place of execution, centralized or decentralized; (iii) the type of method, approximative or exact search; (iv) the more concrete type of the algorithm applied, e.g. heuristic search, linear programming, policy-based, ontology-based, voting, auctioning, etc.

Placement of components is mostly introduced as an optimization task that results in a specific deployment architecture. This resulting architecture has large influence on the QoS that can be provided by the distributed system. In [KHD08], an algorithm has been devised based on calculating the usefulness of alternative configurations as weighted sums. The resulting approach is not computationally effective and serves as a trial to show that deployment decision making is important and necessary to apply. Most known approaches rely on a centralized optimizer that often has to crawl through the entire state-space of decision alternatives. Deployment decision making requires an optimization method to function properly and autonomicity has to be built in as a basic functionality into it.

Authors of [WS08] have reviewed service placement approaches in mobile ad-hoc networks. Observations suggest that a single centralized service can be deployed using passive monitoring alone. Deployment of instances of a distributed service, however, requires iterative message exchange. According to this survey, for deployment in mobile ad-hoc networks typically some middleware is used. Alternatively, it is tackled with an application of facility location theory (e.g. [KSW07]). The middleware approaches investigated in the review, however, fail in supporting truly distributed services, and applications of facility location theory require collecting information at a central node. None of the approaches reviewed consider deployment of services competing for available resources.

Centralized solutions requiring a global view, in particular ILPs, have previously been applied to clustering problems, for example in grid file systems [SME09]. In [BSDS98], an abstract model for applications is proposed and a novel binary integer programming model is introduced to focus on the deployment aspect. However, high computational complexity is a mayor drawback in solving this model. Other, more specific approaches that target minimizing execution delays, such as automated distributed partitioning, decomposition, or graph cutting algorithms lead to NP-hard complexity even in cases where the network consists of only 2-3 nodes [HS99]. The so-called Component Placement Problem (CPP) is defined first in [KIK03]. Authors devise a method that provides a single valid deployment for components by restricting the solution space to tractable sizes by capturing important constraints. Afterwards, the CPP can be solved, but without considering the quality of the solution. Generally, the exact solution algorithms applied for deployment mapping become computationally expensive for realistic problem sizes, even if they only consider a single QoS dimension.

Group-finding algorithms to find mappings in generic wide-area resource discovery is presented in [AOVP08]. In some way similarly to the foraging behavior of artificial ants, some approaches rely on extensive measurement data collection and storage. Optimal placement under a variety of constraints is then calculated based on the central storage, e.g. in [VAN08] and [JHJ09]. The latter proposes a centralized approach, in which an optimizer and model solver component is used to find optimal mappings for VMs. Traditional SLAs were used with planning-based decision making in the MUSIC project, with focus on a middleware [REF$^+$08].

Self-configuration to aid re-deployment was introduced in [MRM04]. A middleware design was presented targeted at disconnected operation via architectural self-reconfiguration, and also to increase availability. The regular MAPE cycle is adhered to in this middleware approach based on system monitoring, estimating (re-)deployment and effecting a new deployment. Similarly, deployment on mobile devices in wireless ad-hoc networks is targeted with a middleware design in [RG05]. Components are stored in local repositories and neighboring devices can exchange components in a peer-to-peer manner. The user of a device can request new components by contacting a centralized deployment manager, however, no QoS aspects are considered. Just-in-time component loading is used as a technique, while future deployment needs are predicted using a Markov model in the task-driven deployment framework of [RCBC07].

Heuristic algorithms for the deployment problem were devised first in [Mal06], and approximative solutions are suggested to overcome scaling problems. In [Mal06] utility is maximized solely from the users' perspective, not considering the preferences of the providers of the services. Moreover, deployment of a single service is targeted at a time, without taking into account the effects of services executed in parallel. The authors first formulate the problem by first defining a set of n components with properties and k nodes with corresponding parameters, a set of constraints that have to be satisfied by valid deployment architecture and an initial deployment mapping. Second, diverse functions are defined for various QoS aspects, such as availability as the number of successful inter-component interactions, latency as the time between request and response pairs plus computational delay, security as levels of encryption or key-length, or energy consumption as a weighted sum of computation and communication costs (in terms of energy) for each software component. Based on the well defined objectives, first mixed-integer linear and nonlinear programming are adopted to find optimal deployments with respect to the multiple QoS dimensions. However, after realizing that the solutions found by state-of-the-art MIP and MINLP solvers do not always improve the cost of the deployment, while being very complex at the same time, approximative algorithms are targeted. These algorithms, however, are application-domain specific. The first attempt is a greedy algorithm, which generates an improved solution every iteration and runs in polynomial time [MRMM05a]. Using a greedy algorithm, however, finding deployments still rely on a centralized logic. To allow distributed implementation of a deployment logic one solution is a genetic algorithm, presented in [MM06]. A multi-objective genetic algorithm was applied in [CW98]. Genetic algorithms can continually improve the solution. The drawback is that they generally have to be tailored very specifically to every problem.

Various other heuristics could be investigated for supporting deployment decision making, such as voting [HKU01], or token-ring based algorithms [NOK93]. In a voting-based method, processing of a task is done by several distributed processors simultaneously and independently. The chosen deployment mapping could then be selected from their output as the majority result of a voting. The distributed version of voting implies that each host independently calculates the desirable deployment based on its limited overview of the system. A token-ring based algorithm mimics the classical solution already applied in networking technologies since the '70s to handle distributed mutual exclusion problems. Designated messages, i.e. tokens can be circulated around all the nodes organized into logical ring structures. For a host to be able to use a resource, acquiring a token is prescribed. The classical method can be used in connection with (re-)deployment to control component migration in a network.

Similar to voting, a market-based approach, an auctioning algorithm was tailored to the deployment problem in [MRMM05b]. The peer-to-peer middleware, CARISMA employs auctioning-like mechanisms for conflict resolution and adaptation as well [CEM03]. Auctioning, however, is unable to detect potentially high performing nodes in the network if they do not have any components deployed initially [MRMM05b]. The QoS-brokering approach in [MD07] allows consumers to request services and then the broker can predict QoS under a variety of workloads using analytic queuing models and optimizing for maximized utility. Layered Queuing Networks in particular are employed in [JJH+08] to build an off-line framework for generating optimal configurations and policies. Moreover, an integrated software mobility framework addressing performance (with Layered Queuing Networks), reliability (using Hidden Markov Models) and energy consumption was proposed in [MEB+10].

Table 4: Classification of some related decision making methods

Approach	On-/Off-line [●/○]	De-/Centralized [●/○]	Approx./Exact [●/○]	Remarks, methods
[KHD08]	○	○	○	alternative configurations
[BSDS98]	○	○	○	ILP
[CW98]	○	○	○	dependence-graphs
[JJH+08]	○	○	○	QNs
[WN04]	○	○	●	centralized CE-method
[SME09]	●	○	○	grid file clustering
[HS99]	●	○	○	graph-cutting
[KIK03]	●	○	○	AI planning
[VAN08]	●	○	○	bin-packing
[RG05]	●	○	○	ad-hoc networks
[RCBC07]	●	○	○	QNs-based prediction
[MD07]	●	○	○	QNs
[JHJ09]	●	○	●	QNs & bin-packing
[REF+08]	●	○	●	planning-based adaptation
[MRM04]	●	○	●	stochastic & greedy alg.
[MRMM05a]	●	○	●	greedy alg.
[CEM03]	●	○	●	policies & auctioning
[MEB+10]	●	○	●	general framework
[XZF+07]	●	○	●	fuzzy learning
[AOVP08]	●	●	●	WAN resource discovery
[Mal06]	●	●	●	market-based alg.
[MM06]	●	●	●	genetic alg.
[MRMM05b]	●	●	●	auctioning
[HJ06]	●	●	●	remote comm. minimization
[MH07]	●	●	●	learning & ACO
This work	●	●	●	CEAS

Biologically-inspired resource allocation algorithms for service distribution problems have been targeted earlier in [HJ06], and stochastic optimization, in particular the CE method was suggested as a solution for some assignment problems in [WN04]. Besides, fuzzy learning is applied for configuration of server environments in [XZF+07], targeting efficient resource utilization by applying a two-level control mechanism. Another variant of ACO-based heuristics combined with artificial learning is applied to map applications in grid environments in [MH07]. This approach, however, considers only one instance – similar to the concept of a component in this thesis – in one species of ants and searches for a suitable assignment to a host.

The various methods relevant for deployment decision making are summarized in Table 4 for comparison with the CEAS-based logic presented in this thesis. This comparison between the approaches is solely qualitative, based on three properties, whether: (i) the execution is on-line or off-line, (ii) the algorithms are distributed or centralized, and (iii) the solutions are approximative or exact.

5. Discussion

This section summarizes the highlights of this work and the approaches taken. It also touches upon some of the advantages and disadvantages of the selected methods, and motivates some of the research left for future work.

Research presented in this thesis started with realizing the importance of efficient deployment of software services. After noticing that choosing different deployment configurations impact the QoS, work started on building methods and establishing strategies for obtaining deployment mappings that satisfy given, QoS-related requirements. Utility of the research results presented in the included papers is summarized in the following list.

1) Establishment of **cost functions** that sufficiently describe the utility of a deployment, with respect to given requirements.

2) Development of decentralized, **heuristic algorithms**, auxiliary techniques and strategies to optimize logical deployment mappings based on bio-inspired methods.

3) **Modeling** and simulation of realistic example **scenarios**, such as a cloud computing setting, to highlight the benefits of the suggested algorithms and the promising application areas.

4) **Cross-validation** of results either by traditional centralized optimization techniques or by choosing and adapting a theoretical problem with known solution.

PAPERS A – G contribute to the given 4 areas as shown in Figure 17.

The first paper, PAPER A was the first attempt to formulate the deployment problem and to develop an initial, yet centralized algorithm based on CEAS for solving it. The first cost function was derived to capture communication and execution costs and simulations were conducted. The scenario included in the paper was a transformed version of a well-known task assignment problem, with known optimum. Hence, the presented algorithm was validated by checking the solution it provided. Thus, PAPER A has contribution in all 4 areas presented above.

PAPER B removed the need for global knowledge in the deployment logic,

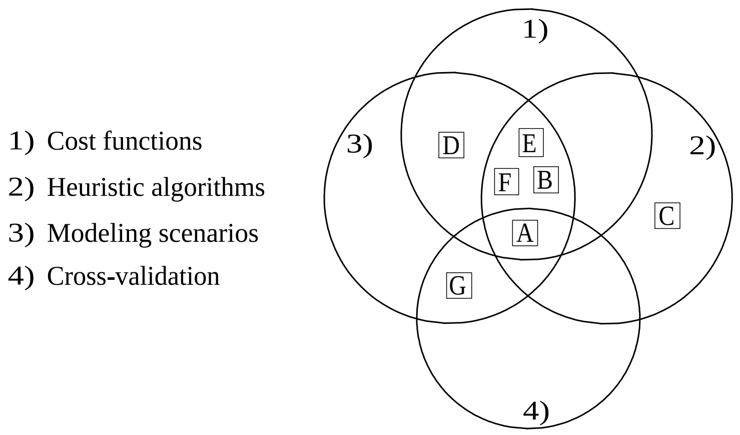


Figure 17: Contributions of PAPERs A – G

by reformulating the cost function and improving the CEAS-based algorithm as well. Also, a new set of service examples were introduced and simulations were conducted within a new scenario. This scenario was used to investigate how the logic adapts deployment mappings in the presence of a node failure and when a new node appears in the network. PAPER C, building on PAPER B, uses the same scenario and investigates what mechanisms are needed to enable deployment of several services in parallel using the deployment logic. Hence, its contribution falls into category (2).

A new set of dependability-related requirements was introduced in PAPER D. This paper mainly focused on experimenting with new cost functions to satisfy dependability rules. Simulation results were included based on a new scenario focusing on replication management. The subsequent paper, PAPER E refines the cost function targeting load-balancing and dependability, and mainly contributes to auxiliary aspects of the deployment mapping algorithm, such as the pheromone database. To experiment with different variants of pheromone encodings a new example scenario is included. Simulation results support the decision, which encoding to choose.

A different application scenario, service deployment in a cloud computing context is discussed in PAPER F. Introducing financial costs also necessitates improvements in the cost functions. Additionally, to handle new requirements of the cloud computing setting parts of the optimization algorithm had to be refined as well. Simulation results show that the deployment logic is flexible and capable of handling new sets of requirements efficiently.

The last paper, PAPER G presents a validation approach to examine the quality of the deployment mappings provided by the logic. Two new service models are used, together with the model already presented in PAPER A. A centralized, exact optimization approach is developed that requires global overview over the system and finds the global optimum. The deployment logic is run in a simulation using the same scenarios and the solutions found are compared to the global optimum.

In particular, the deployment optimization algorithms presented herein build on a swarm intelligence method, the CEAS. Using CEAS, the deployment logic can be tailored to the requirements and scenarios at hand with relatively few mandatory parameters in comparison with other swarm intelligence methods. There are some parameters required by the method, however, that need tuning and have an impact on the performance of the algorithm.

To improve the results of the thesis, a general target is to avoid over-engineering of the method and develop formal techniques to adjust parameters that previously had to be altered manually. For example, it would be beneficial to enable formal derivation of the parameters that fine-tune CEAS and the deployment logic based on scenario and service related information. Additionally, the constructed cost functions could be organized into libraries and combined for given scenarios. Besides, the core functions that describe the inner workings of the CEAS method, such as the pheromone or the temperature updates could also be revisited. The new, improved core functions in CEAS could possibly enhance the performance of the deployment logic.

During the development of the algorithms administrative domain boundaries, e.g. networks of competing companies, were not considered. The logic has to be equipped with additional techniques to tackle this possible lack of transparency between groups of hosts (e.g. in a large public cloud setting it might be difficult to achieve node-level insight). The eventual porting of the algorithms and simulations to more recent simulators, for example a Java-based environment, can also be considered, and would make the method more accessible to the general computer science and networking community.

In the included papers, generally 3 types of scenarios have been investigated, categories II – IV in Figure 15, with corresponding service models. The deployment mappings obtained by the logic for these services were evaluated. Nevertheless, new, more detailed and realistic service models could be developed. Especially, the cost values used in connection with model elements and scenarios, such as collaborations, replicas, or clusters need special consideration and could be determined more precisely. Furthermore, validation of deployment mappings can be extended to categories III and IV. Lastly, more extensive simulations could be conducted to gain more insight into the scaling and convergence properties of the algorithms, and also, into the adaptation capabilities of the logic. These issues are touched upon in Section 6.

In summary, the included papers present a deployment logic that was designed in accordance with the requirements specified in REQs 1 – 6, in Section 1. The service and network models that serve as input for the logic are general enough to describe a wide variety of interesting application scenarios. Simulation results together with validation of the solutions justify the efficiency of the algorithms devised. However, plenty of possible future research questions, worth investigating, can be identified by the author. These are summarized in the next section.

6. Future directions

There are many interesting paths of research, identified in this section, that can be taken based on and following up the work presented in this thesis. The first three items in the following list present work that is ongoing at the time of writing, the rest of the list is considered as future work. The items are not in a priority ordered list.

1) *Translation of the simulations to the highly scalable and Java-based, event-driven simulation environment PeerSim.*

   Translating the algorithms, currently implemented in the legacy Simula language and its DEMOS extension, to a more recent Java-based simulation environment can also be beneficial for example to facilitate experiments with existing middleware. The current simulation environment lacks support for more generic and efficient data-structures, such as *hash-maps*. Using PeerSim [JMJV] extensive simulations shall be conducted to test scalability and convergence of the algorithm. Another opportunity with a more scalable simulator is to evaluate the deployment logic's behavior compared to other relevant optimization methods that support distributed execution. A thorough comparison between the presented logic and other known bio-inspired methods, however, might imply having to deal with fine tuning of multiple parameters in case of many available methods. Porting the code to the Java-based environment makes the presented algorithms more accessible for the general computer science audience and can serve as an important step towards a deployable implementation.

2) *Developing additional ILP models.*

   A recurring comment on the work presented has been the lack of comparison with other similar methods. To address this issue a different type of optimization model, a centralized ILP is developed to gain insight into the performance and effectiveness of the heuristic approach. The ILP models published and presented in this thesis (Category I. in Figure 15) sufficiently describe some of the deployment scenarios discussed. They can be used to validate results in the first set of problems, i.e. Category II. in Figure 15, scenarios with load-balancing and remote communication costs. However, solving problems introduced in Categories III. and especially in IV. of the contributions require extended models that are being worked out at the time of writing. Challenges include validating the implementation of dependability requirements that are represented as soft-constraints in CEAS but can be specified as hard constraints in an ILP. Moreover, it is particularly challenging to describe the non-linear objective introduced in Category IV. of the contributions as an ILP, for which case a piecewise-linear approximation seems to be applicable [Gul10].

3) *The power saving dimension.*

One particularly interesting aspect – which received increased attention recently – is the power-usage characteristics of different deployments and how configurations could be optimized with respect to overall power saving. To be able to consider the additional dimension of requirements, i.e. power conservation, for more efficient deployments, power-usage and its costs have to be quantified and built into the cost functions. At the time of writing, the state-of-the-art in data-center power usage modeling is explored and work is being done on establishing node models and new objectives considering power conservation. One challenge, for example, is to consider dependability aspects of the deployment while power capacity constraints are present. Increased dependability is often achieved at the cost of increased power-usage, hence it is not obvious how to obtain a deployment mapping considering both requirements.

4) *Migration costs.*

Migrating workloads, state-less or state-full software components, and VM instances are all challenging tasks and require particular deep insight into the inner workings of the corresponding systems. VM migration is especially challenging in an inter-site (wide-area) context. The deployment logic presented in this thesis provides new configurations whenever the environment changes to an extent where the new deployment would be significantly better. Changing the current deployment, however, requires at least partial migration of a service, which in turn implies additional migration costs that should be accounted for in the cost functions. Quantifying costs of migration in software services is an interesting and broad research topic in itself. One possible solution for the deployment logic presented herein is to use threshold values proportional to migration costs that allow changing deployment configurations only if the benefit from the new configuration is high enough. The question about what are the costs that matter remains to be answered. Costs, for example, can be related to bandwidth required for transmission of state, or to downtime caused by switching over between replicas. There are many tradeoffs as well that need to be considered here, e.g. if bandwidth is not the main concern downtime of switching over can be very low.

5) *Scaling and larger problem sizes.*

The results obtained in this thesis are promising and some of the focus has been on increasing the method's scalability and adaptability. There is room, however, for implementing more extensive simulations and exploring even larger problem sizes. The two main directions considered where the scenarios shall be extended are larger networks (more nodes and clusters) and, especially, larger amounts of services deployed simultaneously. The former extension mainly affects the length of the iterations the method has to take, whereas the former influences the number of instances of CEAS that are executed (and have to cooperate) in parallel.

6) *Incremental scaling.*

To investigate the concept of incremental scaling, we may add another dimension of dynamicity to the system models and the deployment logic. Run-time component replication can be applied to increase/decrease the replication level at run-time to meet requirements in terms of performance and/or availability. Implementing this feature most probably requires redesign of the algorithms of the logic (to some extent). A sketch of the anticipated changes and a preliminary description of the logic follows.

Given a minimum replication level (e.g. $R = 3$), increase the number of replicas until requirements for availability are met (e.g. n-of-k system). With $A = \sum_{\kappa_j \in \kappa} \prod_{i \in \kappa_j} A_i$, where A_i is the availability of an instance i, and κ represents all $\binom{n}{k}$ combinations.

A preliminary algorithm can be given as:

```
1) – Start with $R = 3$ replicas
2) – Map $\{c_1, c_2, c_3\}$ during forward search, sample $\{A_1, A_2, A_3\}$
3) – **If** $(A > A_{req} \bigvee R = R_{max})$ **then** goto 5) **else** $R{+}{+}$
4) – Map $c_R$; goto 3)
5) – Evaluate the mapping using $F()$
6) – Backtrack
```

Additionally anticipated changes to the method:

a) – one additional table in the *nest* for pre-selection of R

b) – larger and more dynamic pheromone tables (in every node)

c) – A_i values to be sampled in every node

7) *Coordination and designated nest selection.*

   Execution of the deployment logic using multiple ant-nests to allow continuous operation in the presence of network partitions is discussed in PAPER C. Physical placement of components is designed to be triggered by a nest corresponding to the service to be placed. When multiple nests are used for the same service some mechanism has to be in place to elect a designated nest that can provide the trigger after convergence of the deployment algorithm. A coordination and designated nest selection protocol would be desirable that would ensure consistency for placement triggers. For practical purposes it should be ensured that the deployment logic is resilient to multiple leaders or no leaders at times.

8) *Better and broader service models.*

   The service models presented in the examples in Part II are representative for the deployment scenarios of Category II. – IV. Better and more complex models of services can be developed that are more realistic and describe aspects of services not included in this thesis. One such aspect is for example the inclusion of costs related to maintaining consistency between component replicas, i.e. the costs of executing certain consistency protocols.

9) *Introducing clients.*

   Service models discussed in Part II do not encompass too much information regarding the users of a given service. An additional option to refine the models used by the deployment logic is to incorporate more information regarding the client-side. Related information ranges from specifying user demand levels for selected components, as well as defining specific access points where services are accessed from inside the network. One related design question is whether to include client-specific information in the service model or to describe clients via a separate model overlapping with the service. Alternatively, adaptive design models can be targeted, where the number of components is regulated/scaled in accordance with load levels.

10) *Costs derived from service models.*

Costs, such as execution or communication costs, are defined intuitively in the examples presented and are used for evaluation purposes only. It could be beneficial, however, to derive these and similar relevant cost values automatically. Methods that are expected to be applicable to aid enriching the service models include: code analysis, constant transition costs, various offline measurements, or other prediction methods on expected demand.

11) *Deployment diagrams.*

Little work has been done on visualizing the output of the deployment logic for human readability. Standard UML Deployment Diagrams are one possible option that can be considered for depicting and editing deployment mappings. Another possible application would be the specification of desired deployment rules, constraints and parameters at design time, which could serve as input for the logic.

12) *Feedback to functional design.*

Initial deployment mappings obtained by the logic might be used preceding placement of a service to predict its utility and QoS. Methods and their place in the software development cycle are to be investigated to provide useful feedback to the functional design phase. Evaluation of the suggested deployment mapping can indicate prohibitively high costs, which problem might be tackled easier by choosing some other functional pattern at design time.

13) *Experiments using a middleware platform.*

As a long term goal, experiments with connecting the *Analyze* and *Plan* building blocks of the MAPE cycle (cf. Figure 4) implemented by the deployment logic to an *Effector* carrying through commands of the *Execute* building block might be worthwhile to conduct. In the replica management scenario the DARM framework [MG08] appears as a potential candidate, other middleware or various frameworks might possibly be used for this purpose that most likely requires extensive programming. This is a possible direction after more insight has been gained into the scalability properties of the method and it can be generalized into a library for integration.

Part II

INCLUDED PAPERS

PAPER A

Cost Efficient Deployment of Collaborating Components

Máté J. Csorba and Poul E. Heegaard and Peter Herrmann

In proceedings of 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'08)

Oslo, Norway, June 4-6, 2008

COST-EFFICIENT DEPLOYMENT OF COLLABORATING COMPONENTS

Máté J. Csorba, Poul E. Heegaard, Peter Herrmann
Department of Telematics,
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard, Peter.Herrmann}@item.ntnu.no

Abstract We study the problem of efficient deployment of software components in a service engineering context. Run-time manipulation, adaptation and composition of entities forming a distributed service is a multi-faceted problem challenged by a number of requirements. The methodology applied and presented can be viewed as an intersection between systems development and novel network management solutions. Application of heuristics, in particular artificial intelligence in the service development cycle allows for optimization and should eventually grant the same benefits as those existing in distributed management architectures such as increased dependability, better resource utilization, etc. The aim is finding the optimal deployment mapping of components to physically available resources, while satisfying all the non-functional requirements of the system design. Accordingly, a new component deployment approach is introduced utilizing distributed stochastic optimization.

1. Introduction

Today, computer applications tend to be highly distributed and dynamic. In addition, they are executed on hardware systems that change their topology and performance dynamically. This calls for flexible methods to deploy the software components realizing a networked application on the available hosts to achieve preferably high performance and low cost levels.

By such a software component we mean an executable stand-alone package of software that has a well-defined interface and can communicate with other components via message exchange. Furthermore, we define a service as a collaboration of distributed components running in a (possibly also highly distributed) hardware environment on different hosts, using distinct network elements for interconnection. A specific service can be observed from different views. We investigate the problem of efficient component deployment from the view of the service creator who is in most cases the provider of the service as well. We do so based on the starting point we use for our investigation, i.e. we start from a service specification, from a model that is a product of the service designer. Usually the parameters we are interested in are performance and cost effectiveness, which are both substantial from the provider's perspective if it comes to the deployment of a new service.

The problem of cost-efficient component deployment is challenged by multiple dimensions of Quality of Service (QoS), or in other words, non-functional requirements that need to be taken into account. To name a few there might be a fluctuation in the number of users of the service deployed who might also have arbitrary utility functions for the service as well as different usage scenarios. Additionally, the QoS requirements identified might change over time, the system designed might provide several services. This complicated combination of factors forms the basis of the problem we aim to solve. Namely, finding the optimal deployment mapping of components to physically available resources, while satisfying all the non-functional requirements of the system design.

The resulting deployment mapping has a large influence on the QoS that can and will be provided by the system. The most basic example of improving QoS by choosing a better deployment architecture is to consider only the latency of the service. The easiest way to satisfy latency requirements is to identify and deploy the components that require the highest volume of interactions onto the same resource, or to choose resources that are at least connected by links with sufficiently high capacity.

Several approaches have been followed to solve this problem, e.g. binary integer programming [BSDS98] or graph cutting [HS99]. Usually, complexity becomes NP-hard using these methods with more than 2-3 hosts. Others try to capture constraints and restrict the solution space [KIK03]. However, due to the exact solution algorithms computational complexity is still an issue. What is even more restrictive in these approaches is that they do not attempt to work with more than one QoS dimension at a time, while our objective is to deal with vectors of QoS properties in one run. Furthermore, we aim to be able to aid the deployment of several different services at the same time using the same framework.

Approximative solutions are devised by Malek et al., such as greedy algorithms, genetic programming for example in [Mal06]. Malek et al. however approaches the deployment problem from the user's perspective by maximizing an overall utility function. On the contrary, we aim to investigate the deployment problem from the service provider's perspective. Besides, autonomous replication management is targeted by Meling in a framework based on group communication systems [Mel06]. Widell et al. discuss an alternative solution based on a stochastic optimization method called the Cross Entropy (CE) Method [WN04].

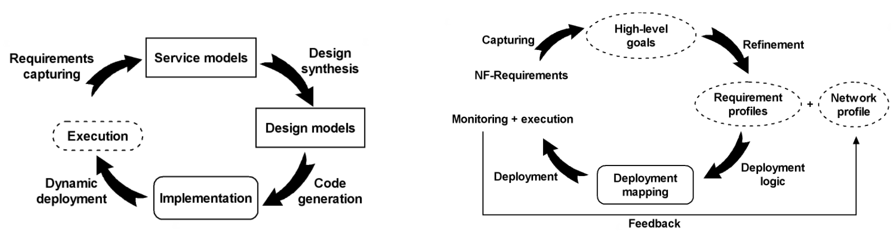
Generally, we require a method that is capable to adapt to changes in the environment in a highly efficient way. Also, as module allocation problems are proven to be NP-complete (cf. [FB89]), except in some special cases, heuristics are needed for providing an efficient solution. Accordingly, we chose a bio-inspired system, swarm intelligence as a basis for our method to solve the deployment problem in a fully distributed manner. As we omit any centralized database or building block and propose to use the analogy of pheromones for storing information in a distributed way the logic presented is robust and highly adaptive with respect to changing QoS provided by the service execution platform. Eventually, our aim is to develop a method for run-time component (re-)deployment support that allows execution of services within the allowed region of external parameters defined by the service requirements.

The remainder of this paper is organized as follows. The next section will introduce our system model and position our work. Sect. 3 briefly presents the Cross-Entropy Ant System (CEAS) that is used throughout the paper as the basis of our heuristic optimization method. Sect. 4 provides our solution to the target scenario and a summary of our algorithm. Sect. 5 comes with a more tangible example and compares our results to previous solutions. In the last section we conclude and touch upon our future work.

2. Support for Deployment Mapping

Our deployment approach fits to the engineering method SPACE which is devoted to the rapid and correct engineering of distributed services [KH06]. As depicted in Fig. 1a, in the process of developing a service, one creates first a purely functional service model. This specification is collaboration-oriented, i.e., the overall service specification is not composed from descriptions of the physical software components realizing the service but from models of distributed sub-functionalities which — in interaction — fulfill the complete service behavior. This specification style enables the development of service models by reusing building blocks from domain specific model libraries to a much higher degree than it would be possible when applying component-based descriptions (e.g., [HK07]). As modelling language, we use UML collaborations and activities.

After performing correctness checks on the service model (see [KSH07]), it is transformed to a component-oriented design model by a model transformation tool [KH07] which is specified by UML state machines. In the next step, code generators create executable Java code from the design model enabling a fully automated transformation of collaboration-oriented service models to executable programs. This process is well described in [KHB06].



(a) Development cycle (b) Deployment support

Figure 1: Development with SPACE and the deployment support

For the efficient deployment of the implementation, we extend the development cycle as shown in Fig. 1b. The service models are amended by *high-level non-functional (NF) goals* defining the non-functional requirements (NFR) of a service in a rather abstract manner. In parallel with the transformation from the service to the design models, these NF goals are refined into *requirement profiles* specifying the non-functional requirements of the service components. Moreover, a *network profile* is added, thus required and provided properties are collected describing the system and its target environment. Based on these inputs our deployment logic can be launched with the profiles specifying the goals and the *net-map* specifying the search space.

For capturing QoS requirements that are relevant to our system, we follow the collaboration-oriented style and capture NFRs in design time. NFRs usually represent qualities such as security, performance, availability, portability, etc. In fact, in our view the deployment logic should be able to handle any properties of the service, as long as we can provide a cost function for the specific property. In that matter we will exploit the advanced scalability of CEAS and the method of pheromone sharing.

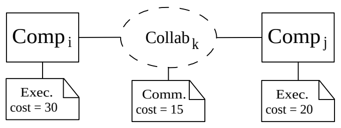


Figure 2: Collaboration with NFRs

In Fig. 2 a simple example of a collaboration between two components is depicted enriched with NFRs for both the components and for the collaboration binding them. This basic collection of requirements contains two types of cost values, an execution and a communication cost. The execution cost is added to the local cost of a node that contains the particular component after deployment. The communication cost is imposed on the connection between the two components participating in the collaboration. This simple example of collaboration-oriented specification and capturing of requirements will be illustrated in the example in Sect. 5.

Existing component deployment strategies and solutions use various centralized databases and decision logics. Relying on a fully centralized logic requires the burden of keeping the central database constantly updated and at the same time introduces a single point of failure in the system. Moreover, a performance bottleneck may arise at the node storing the central database and accommodating the decision logic both communication wise and storage wise.

In a distributed cooperative algorithm (semi-)autonomous agents cooperate to achieve certain common goals. Since in a distributed environment autonomous agents do not have an overview of the system as a whole, their decisions have to be based on information that is available locally to the place where they reside. To enable cooperation between agents, some sort of shared memory is required at each place an agent can visit. In our deployment logic, the information is distributed across all the nodes participating in the deployment. In this way, we achieve a completely robust, scalable and fault tolerant mechanism. Furthermore, to achieve a complete solution, our aims are twofold. First, the logic shall be able to obtain an initial deployment mapping based on the service model. Second, once the service is running, the logic shall be capable of monitoring online and execute the necessary changes to satisfy the requirements it is launched with.

The objective is to find the optimal, or at least a satisfactory, mapping in reasonable time between a number of component instances c, onto nodes n. A component, $c_i \in \mathbf{C}$ ($\mathbf{C}$ is the set of components available for (re-)deployment) can be a *client process*, or a *service process*, while a node, $n \in \mathbf{N}$ ($\mathbf{N}$ is the set of nodes) can be a *transit node*, e.g. a traditional IP router, a *server node*, which is capable of accommodating a service component, a *client node*, which is an aggregation point for client components, or a *mixed node* that can accommodate both client and service components.

The cost function $F(M)$ of the mapping $M : \mathbf{C} \to \mathbf{N}$ should be minimized under the constraints given by the *mapping scopes* $\mathbf{R}_i \subseteq \mathbf{N}$ for each component instance i. $\mathbf{R}_i$ is determined by the intersection of access restrictions, service provider policies (e.g. service level agreements of ISPs), provided and requested capabilities (soft costs) and provided and requested capacity requirements (hard costs, e.g. bandwidth limitations). Attached components, i.e. components restricted to a specific node will have an $\mathbf{R}_i$ set consisting of a single node, thus reducing the search space.

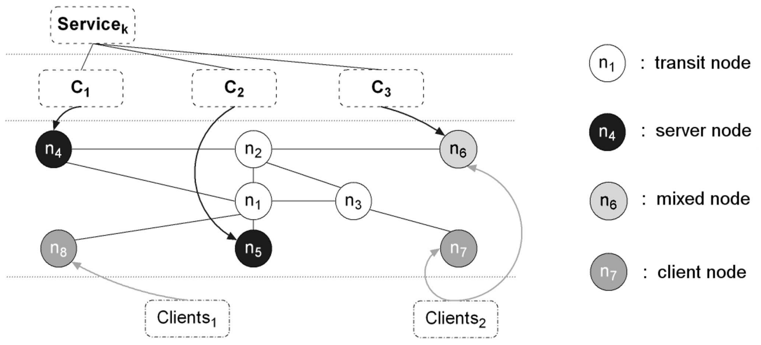


Figure 3: Component mapping example

An illustration of the model can be found in Fig. 3. Suppose we develop a service, *Service$_k$*, which is implemented by three service components $\mathbf{C} = \{c_1, c_2, c_3\}$ and the service is expected to be accessed by two distinguishable set of clients. Besides the *requirement profiles*, the service provider must provide the *net-map* for the decision logic as well, specifying the available nodes and links. Thus, the set of nodes becomes $\mathbf{N} = \{n_1, n_2, \ldots, n_8\}$. Client nodes in this case are considered to be aggregation nodes, i.e., they represent a single point of access to the network for the clients of the service, with a different meaning from the traditional notion of node. So, the designer can specify where in the provided *net-map* the clients are located and can insert additional parameters describing the clients of the service, such as the expected amount of clients, the expected service demand, etc. as NFRs. Constraints that will influence the optimal deployment can be assigned to nodes and links. For links, constraints appear as the costs of using the particular link for connection between two components that need to interact. Constraints assigned to nodes, for instance, can represent memory sizes restricting placement of component instances to a place. Besides, node properties can be interrelated, i.e., for example if a mixed type node (n_6) accommodates a service component it can influence the rest of the properties, e.g. lower the amount of allowed clients at the node by modifying the memory constraint.

Next, we introduce the stochastic optimization background, which we use for providing solutions to the component deployment and redeployment problem.

3. Cross Entropy Ant System

The deployment problem in this paper is approached by use of a distributed, robust and adaptive routing system called the Cross Entropy Ant System (CEAS) [HW01]. The CEAS is an Ant Colony Optimization (ACO) system as introduced by Dorigo et al. [DMC96], which is a multi-agent system for solving a wide variety of combinatorial optimization problems where the agents' behavior are inspired by the foraging behaviour of ants. Examples of successful application in communication system are load-balancing (Schoonderwoerd et al. [S^+97]), routing in wired networks by AntNet [CD98], and routing in wireless networks by AntHocNet [CDG05]. The key idea is to let many agents, denoted *ants*, iteratively search for the best solution according to the problem constraints and cost function defined. Each iteration consists of two phases; the *forward* ants search for a solution, which resembles the ants searching for food, and the *backward* ants that evaluate the solution and leave markings, denoted *pheromones*, that are in proportion to the quality of the solution. These pheromones are distributed at different locations in the search space and can be used by forward ants in their search for good solutions; therefore, the best solution will be approached gradually. To avoid getting stuck in premature and sub-optimal solutions, some of the forward ants will explore the state space freely ignoring the pheromone values.

The main difference between the ant based systems is the approach taken to evaluate the solution and update the pheromones. For example, AntNet uses reinforcement learning while CEAS uses the *Cross Entropy (CE) method for stochastic optimization* introduced by Rubinstein [Rub99]. The CE method is applied in the pheromone updating process by gradually changing the probability matrix $\mathbf{p}_r$ according to the cost of the paths. The objective is to minimize the cross entropy between two consecutive samples $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. For a tutorial on the method, [dBKMR05] is recommended.

The CEAS has demonstrated its applicability through a variety of studies of different path management strategies, such as shared backup path protection (SBPP) [WH04], p-cycles [WHN05], resource search under QoS constraints [WHH03], and adaptive paths with stochastic routing [H^+05]. Implementation issues and trade-offs, such as management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism [H^+04] and self-tuned packet rate control [HW06a], [HW06b]. Additional reduction in the overhead is accomplished by pheromone sharing [KWH08] where ants with overlapping requirements cooperate in finding solutions by (partly) sharing information.

In this paper, the CEAS is applied to obtain the best deployment of a set of components, $\mathbf{C}$, onto a set of nodes, $\mathbf{N}$. The nodes are physically connected by links used by the ants to move from node to node in search for available capacities. A given deployment at iteration r is a set $\mathbf{M}_r = \{\mathbf{m}_{n,r}\}_{n \in \mathrm{N}}$, where $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ is the set of components at node n at iteration r. In CEAS applied for routing the path is defined as a set of nodes from the source to the destination, while now we define the path as the deployment set $\mathbf{M}_r$. The cost of a deployment set is denoted $F(\mathbf{M}_r)$. Furthermore, in the original CEAS we assign the pheromone values $\tau_{ij,r}$ to interface i of node j at iteration r, while now we assign $\tau_{mn,r}$ to the component set m deployed at node n at iteration r. In Sect. 4 we describe the search and update algorithm in details.

In traditional CEAS applied for routing and network management, selection of the next hop is based on the *random proportional rule* presented below. In our case however, the *random proportional rule* is applied for deployment mapping. Accordingly, during the initial exploration phase, the ants randomly select the next *set of components* with uniform probability $1/E$, where E is the number of components to be deployed, i.e. the size of $\mathbf{C}$, while in the normal phase the next hop is selected according to the *random proportional rule* matrix $\mathbf{p}_r = [p_{mn,r}]$, where

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in \mathbf{M}_{n,r}} \tau_{ln,r}} \tag{1}$$

The pheromone values in (1) are determined considering the entire history of cost values $\mathbf{F}_r = \{F(\mathbf{M}_1), \ldots, F(\mathbf{M}_r)\}$ up to iteration r. The backward ants update the pheromone values at the nodes where one or more components in $\mathbf{M}_r$ are deployed. The pheromones are updated according to

$$\tau_{mn,r} = \sum_{k=1}^{r} I(l \in \mathbf{M}_{n,r}) \beta^{\sum_{x=k+1}^{r} I(x \in \mathbf{M}_k)} H(F(\mathbf{M}_k), \gamma_r) \tag{2}$$

where $I(x) = 1$ when x is true and 0 otherwise. $H(f,\gamma) = e^{-f/\gamma}$ is the performance function and $\beta \in (0,1)$ is the weight parameter, or in other words the memory factor in the auto-regressive formulation of the performance function. The auto-regressive formulation $h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1-\beta)H(F(\mathbf{M}_r), \gamma_r)$ is the key in CEAS for avoiding any centralized control and synchronized iterations. This reformulation allows the cost value $F(\mathbf{M}_r)$ to be calculated immediately after a single ant ends its forward movement, i.e. the ant manages to find a mapping for all the components originally assigned to it. Now, iteration r represents the total number of updates, in other words, the total number of backward ants returned. The reformulated performance function, $h_r(\gamma_r)$ can be approximated by

$$h_r(\gamma_r) \approx \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} e^{-\frac{F(\mathbf{M}_i)}{\gamma_r}} \tag{3}$$

see [HW01]. Thus, a digest of the search history is applied, where older cost values gradually disappear, i.e. *evaporate*. This evaporation is achieved using the memory factor β that provides geometrically decreasing weights for the output of the performance function. The control parameter, γ_r can be determined by minimizing γ subject to $h(\gamma) \geq \rho$, where ρ is the search focus parameter (typically 0.05 or less). For more details about the parameters and solutions to (2) and (3) see [Wit03].

4. Application of Ant-based deployment mapping

The deployment logic can be considered as an optimization task continuously executed by independent ant-like agents in the target network hosting the service we model. The continuous ant behavior contributes to the advantage of our approach, namely that the same logic can be used for an initial static mapping and for an online redeployment mechanism.

At first, every ant is assigned a task of deployment of $\mathbf{C}$ components. Thereafter, ants are started continuously and proceed with a random-walk on the provided *net-map* randomly selecting each next node to visit. Behavior at a visited node depends on if the ant is an *explorer* or a *normal* ant. A *normal* ant selects a subset of $\mathbf{C}$ governed by the pheromone levels at the node it currently resides in and stores its selection $\mathbf{m}_{n,r}$ in a mapping list $\mathbf{M}_r$, which is carried along by the ant. Similarly, an *explorer* ant selects a subset $\mathbf{m}_{n,r}$ based on a random decision instead of the distributed pheromone database. *Explorer* ants are used for exploring the available *net-map*, both initially and later as well for covering up fluctuations in the network, e.g. new nodes appearing. More precisely, the effects of *exploration* are twofold. First, as optimization starts *explorer ants* are used to cover up a significant amount of the problem space via random sampling. The required number of initial *exploration* iterations depend on the problem size, but it can be estimated by sampling the pheromone database size. After that, the *normal* phase starts, in which case only a fraction of the ants generated are flagged as *explorers*, thus allowing for the required responsiveness to changes in the environment, while *normal* ants are focusing on finding the optimum.

Once an ant has deployed all its assigned components the resulting mapping $\mathbf{M}_r$ can be evaluated by applying the cost function $F(\mathbf{M}_r)$ derived from the service specification. A more concrete example on $F(\mathbf{M}_r)$ can be found in Sect. 5. Once the mapping is evaluated, the ant goes back along the nodes in its path that has been stored in the hop-list $\mathbf{H}_r$ and updates pheromone values according to Equation (2) corresponding to the pairs of component sets and nodes it has selected during its journey. After that, a new iteration starts as a new ant is emitted, unless a stopping criteria is met. A stopping criteria can be constructed by observing the moving average of the evolving cost value, i.e. detecting convergence to a suggested solution. Another option is sampling the size of the distributed pheromone database during an iteration. After convergence a very strong pheromone value will emerge in the database, while inferior solutions will evaporate. The described process is summarized in Algorithm 1.

Generally, we have a trade-off between convergence speed and solution quality. Nevertheless, while deploying a service in a dynamic environment, which is our goal, a pre-mature solution that satisfies both functional and non-functional requirements often suffices. Thus the optimality requirement can be relaxed while taking restoration time requirements into consideration. Besides, it has been proven that ACO systems do in fact find the optimum at least once with probability close to one and when this has happened they converge to the optimum in a finite number of iterations. Since CEAS can be considered as a subclass of ACO the optimal deployment mapping will eventually emerge.

Algorithm 1 Deployment mapping of $\mathbf{C} = \{c_1, \ldots, c_E\}$ component instances

```
1 Select the initial node $n \in \mathbf{N}$ where the search will start randomly.
2 Select a set of components $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ which satisfies $n \in \mathbf{R}$ for every $c_i \in \mathbf{m}_{n,r}$
  according to the random proportional rule (normal ant), Equation (1), or in a
  totally random manner (explorer ant). If such a set cannot be found, goto step 5.
3 Update the ant's deployment mapping set, $\mathbf{M}_r = \mathbf{M}_r + \{\mathbf{m}_{n,r}\}$.
4 Update the set of components to be deployed, $\mathbf{C} = \mathbf{C} - \mathbf{m}_{n,r}$.
5 Select next node, $n$ randomly and add $n$ to the hop-list $\mathbf{H}_r = \mathbf{H}_r + \{n\}$.
6 If $\mathbf{C} \neq \emptyset$ then goto 2., otherwise evaluate $F(\mathbf{M}_r)$ and update the pheromone val-
  ues, Equation (2) corresponding to the $\{\mathbf{m}_{n,r}\} \in \mathbf{M}_r$ mappings going backwards
  along $\mathbf{H}_r$.
7 If stopping criteria is not met then increment $r$, initialize and emit new ant and
  goto 1.
```

5. Analysis of a Problem

As a representative example, we consider the scenario originally from Efe dealing with heuristical clustering of modules and assignment of clusters to nodes [Efe82]. This scenario has also been investigated by Widell et al., and a comparison to results of several other authors can be found in [WN04]. This scenario, even though artificial and may not be tangible from a designer's point of view, is sufficiently complex to test our deployment logic. The problem is defined in our approach as a collaboration of $E = 10$ components (labelled $c_1 \ldots c_{10}$) to be deployed and $K = 14$ collaborations between them k_j, $j = 1 \ldots K$, as depicted in Fig. 4. We consider three types of requirements in this specification. Besides the execution and communication costs, we have a restriction on components c_2, c_7, c_9, regarding their location. They must be bound to nodes n_2, n_1, n_3, respectively.

Furthermore, to be able to use similar mechanisms for specifying the *net-map* for the deployment logic, we propose to use the same object paradigm UML employs to reduce complexity. Thus, we specify the underlying physical map of hosts as a diagram, depicted in Fig. 5.

In this example, the target environment consists only of $N = 3$ identical, interconnected nodes with a single provided property, namely processing power and with infinite communication capacities. Accordingly, we only observe the total load ($\hat{l}_{n,r}$, $n = 1 \ldots N$) of a given deployment mapping at each node. The communication cost between two components is considered significant only if it appears between two separate nodes, and we will strive for a global optimal solution of equally distributed load among the processing nodes and the lowest cost possible, while taking into ac-

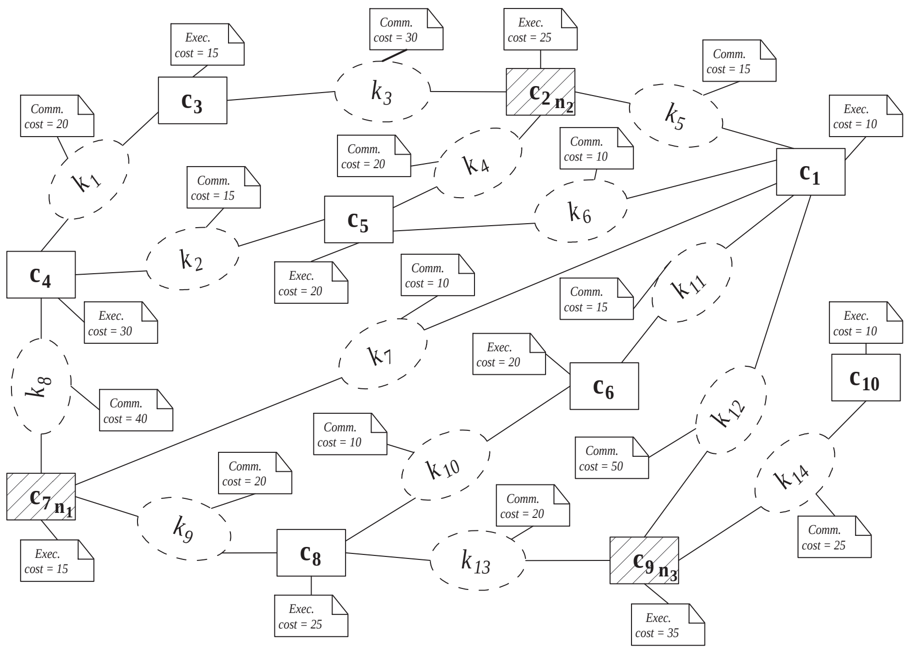


Figure 4: Collaborations and components in the example scenario

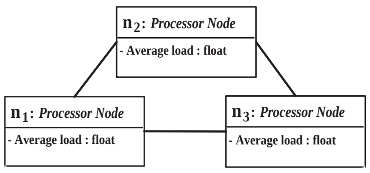


Figure 5: The target network of hosts in the example scenario

count the NFRs, execution cost f_{c_i}, $i = 1 \dots E$ and communication cost f_{k_j}, $j = 1 \dots K$. f_{c_i} and f_{k_j} are derived from the service specification, thus, the total offered execution load can be calculated before optimization starts as $\sum_{i=1}^{E} f_{c_i}$. This way, the logic can be aware of the target load

$$T = \frac{\sum_{i=1}^{E} f_{c_i}}{N} \tag{4}$$

By looking at the example in Fig. 4 and Fig. 5 for this service we have $T \cong 68$. Given a mapping $\mathbf{M}_r = \{\mathbf{m}_{n,r}\}$, the total load can be obtained as $\hat{l}_{n,r} = \sum_{c_i \in \mathbf{m}_{n,r}} f_{c_i}$. Furthermore, the overall cost function $F(\mathbf{M}_r)$ becomes

$$F(\mathbf{M}_r) = \sum_{n=1}^{N} |\hat{l}_{n,r} - T| + \sum_{j=1}^{K} I_j \, f_{k_j} \tag{5}$$

for mapping $\mathbf{M}_r$ suggested by ant r, where

$$I_j = \begin{cases} 1, & \text{if } k_j \text{ external} \\ 0, & \text{if } k_j \text{ internal to a node} \end{cases} \tag{6}$$

Optimization governed by the cost function $F(\mathbf{M}_r)$ starts with aligning pheromone values with the sets of deployed components. With the underlying set of nodes ($\mathbf{N}$) each ant will form N discrete sets from the set of available components ($\mathbf{C}$) that need to be deployed and evaluate the outcome of that deployment mapping ($\mathbf{M}_r$) at the end of its run. However, the ants only need to carry a list of the unrestricted components, i.e. with the exception of components c_2, c_7, c_9 that are bound to a node by a constraint, leaving the rest of 7 components for mapping. A flag is assigned to each of the remaining components giving 2^7 as the number of possible combinations for a set at a node. Thus, the pheromone database at each node has to accommodate 2^7 floating point numbers in this case. After normalizing the pheromones in a node we can observe the probability distribution of component sets mapped to that particular node by the ant system. Eventually the optimal solution(s) will emerge with probability one after convergence.

The pheromone database is indexed by a component set identifier. For example, Id. 36, which is equivalent to $'0100100'B$, indicates that the free components c_4 and c_8 are deployed on that node. In Fig. 6, pheromone levels (normalized as probabilities) for two sets of components at node n_1 are depicted. After the initial phase of 10000 explorer ants doing random search the emergence of the solution deemed optimal can be seen in Fig. 6a for the set of components c_4, c_8 in addition to c_7 attached in advance. Also, in Fig. 6b evolution of the pheromone corresponding to a suboptimal set of components, c_4, c_6, c_8 and c_7 deployed at n_1, is shown (observe the different scales on the Y-axis).

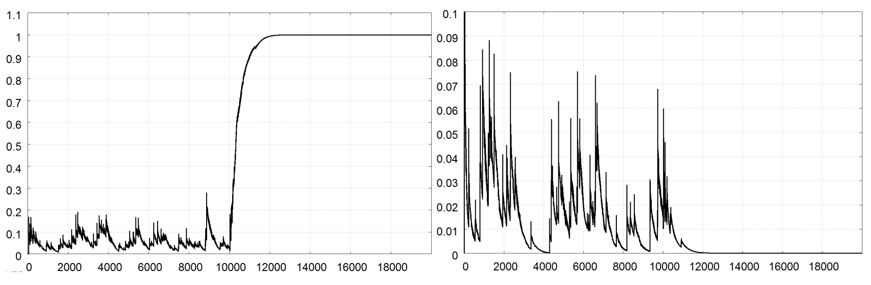

(a) Id. 36. (b) Id. 52.

Figure 6: Pheromones at node n_1

The optimal deployment mapping can be observed in Table 1. The lowest possible deployment cost, according to (5) is $17 + (200 - 100) = 117$.

The rare event of finding the optimal deployment with the lowest cost during a

Table 1: Optimal deployment mapping in the example scenario

node	components	$l_{n,opt}$	$\lvert l_{n,opt} - T \rvert$	internal collaborations
n_1	c_4,c_7,c_8	70	2	k_8, k_9
n_2	c_2,c_3,c_5	60	8	k_3, k_4
n_3	c_1,c_6,c_9,c_{10}	75	7	k_{11}, k_{12}, k_{14}
$\sum$cost			17	100

random search can be observed in Fig. 7. The exploration phase consists of the first 2000 ants, conducting a random search and resulting in a random cost figure. However, after exploration ends, from ant number 2001, the real optimization phase starts and the overall deployment cost is converging to the optimal value of 117. At the same time, we propose usage of a pheromone database that is allocated dynamically in the memory for storing pheromone values based on a threshold level that evaporates all the pheromone entries under a certain significance level. In Fig. 7, 1% threshold is applied, i.e. pheromones smaller than 1% of the highest value are deemed insignificant and are eliminated from the database.

The database size tops at 2^7 as the solution space is starting to be covered by exploration ants and thus it can be used as an indicator to switch to the optimization phase. Likewise, when the overall cost converges to the optimal value (117) the size of the database approaches one (if there is a single solution like in the example) as the single optimal solution prevails, allowing for convergence detection.

We can compare our results to the results obtained using the centralized CE method. A comparison between different solutions to the original problem from Efe can be found in [WN04]. Widell et al., in accordance with the original CE method, uses a selected distribution to generate a sample iteration, which is in case of the component deployment problem a particular deployment mapping. The generated samples are then used for updates in the parameter of the selected distribution. The updates are based on an assessment of the quality of the sample iteration. Sampling and updating is repeated until convergence is detected, which, due to stochasticity though might not be the optimal mapping of components. In fact, the number of ant runs in distributed CEAS can be compared to full iterations in the centralized CE method, as a single ant's lifetime (from leaving the nest until its return) is equivalent to the number of samples taken multiplied with the number of iterations.

For example, in [WN04] using 100 samples the mean number of iterations required for finding the optimal solution with 80% confidence is 41, which in turn is approximately equivalent to $100 \cdot 41 = 4100$ ant runs. We can see that using the same CE focus parameter, i.e. $\rho = 0.01$, and a memory factor of $\beta = 0.998$ (cf. Sect. 3), we can expect convergence times to average at 1200 ant runs for arbitrary number of explorations (Fig. 8) using our distributed CEAS approach. Here, we only compared our results to the most efficient solution by Widell et al. However, it is difficult to compare the two approaches in terms of number of iterations because they differ in the methodology, i.e. multiple samples in one iteration in Widell's work versus one iteration as a sample in CEAS. Nonetheless, we have found that our approach is capable of finding the optimal solution (cf. Table 1) with at least the same confidence,

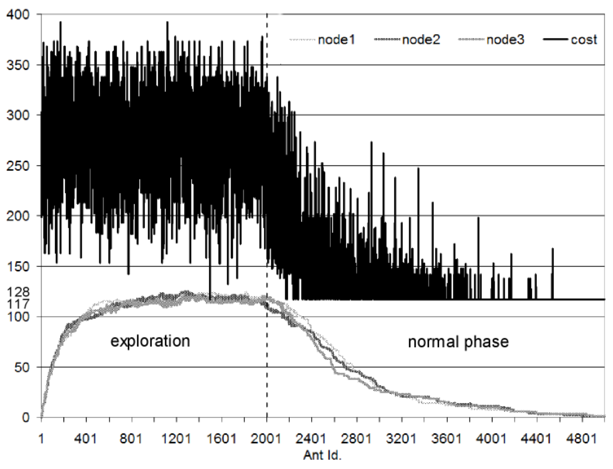


Figure 7: Observed cost and pheromone database sizes

requires less iterations, thus it is resource conserving and last but not least it is a completely distributed logic compared to the original CE-based method and the other strictly centralized solutions, e.g. clustering, bin-packing, etc.

In Fig. 8, results of running the deployment logic with different amounts (shown on the x-axis) of explorer ants are depicted. The mean values of 200 subsequent executions in each setting can be observed with the standard deviation of the results included as error bars. The deployment logic is currently implemented in a simulator written in the Simula/DEMOS language [Bir03] for evaluation purposes.

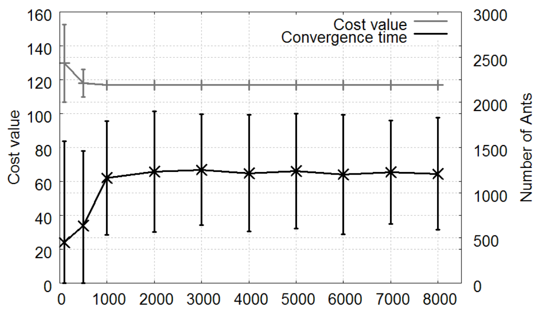


Figure 8: The observed cost and the number of ants required for convergence as a function of the number of explorer ants

It can be noted that above a sufficient amount of initial exploration of the problem the logic is quite robust in finding the optimal solution and stable in convergence time as well. However, in our algorithm we do not set the number of explorers to a constant number, instead we propose to use the dynamic database size as an indication for sufficient exploratory runs. Also, an advantage of our approach is that it can provide alternative solutions weighted by their cost and corresponding pheromone values will indicate the deployment mapping for those solutions. So, in a system where convergence time is very critical, even premature results can be used for near optimal deployment.

6. Closing Remarks

We presented a novel approach for the efficient deployment of software components taking into account QoS requirements captured during the modelling phase in the service engineering approach, SPACE. The procedure starts from high-level QoS goals and through requirement profiles utilizes swarm intelligence to provide solutions and to aid dynamic deployment. The logic itself can be executed in a fully distributed manner, thus it is not prone to deficiencies of existing centralized algorithms, such as performance bottlenecks and single point of failures. Our approach does not require a centralized database, instead it uses the analogy of pheromones distributed across the network of hosts. Furthermore, the logic, as it is presented here, is applied to provide the optimal, initial mapping of components to hosts, i.e. the network is considered rather static. However, our eventual goal is to develop support for run-time redeployment of components, this way keeping the service within an allowed region of parameters defined by the requirements. As the results with CEAS show our logic will be a prominent candidate for a robust and adaptive service execution platform.

Our work is conducted in cooperation with the ISIS (Infrastructure for Integrated Services) project funded by the Research Council of Norway comprising of multiple participants both from industry and academia. The methodology and algorithms presented are in-line with the objectives of ISIS that are to create a well-established service engineering platform for collaboration-oriented models, covering a development cycle from the requirements to seamless execution in a heterogenous and dynamic environment.

In our future work we will investigate applicability and utility of different deployment strategies based on the existing logic. Also, we plan to experiment with stochastic optimization methods other than the CE method. Another issue is database size management locally to the nodes hosting the service. The first step to address this issue was the introduction of dynamically allocated databases, which will be investigated further. Especially, in case of deployment of multiple services at the same time, which is one of the topics in our future research.

References

[Bir03] G. Birtwistle. *Demos - a system for Discrete Event Modelling on Simula.* Springer-Verlag New York, Inc., 2003.

[BSDS98] M. C. Bastarrica, A. A. Shvartsman, S. A. Demurjian, and Sr. A binary integer programming model for optimal object distribution. In *Proc. of the 2nd Int'l. Conf. on Principles of Distributed Systems*, 1998.

[CD98] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, 1998.

[CDG05] G. Di Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Trans. on Telecomm.*

(ETT) - Special Issue on Self Organization in Mobile Networking, 16(5), 2005.

[dBKMR05] P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134, 2005.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[Efe82] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15(6):50–56, 1982.

[FB89] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. on Software Engineering*, 15(11):1427–1436, 1989.

[H^+04] P. E. Heegaard et al. Distributed asynchronous algorithm for cross-entropy-based combinatorial optimization. In *Rare Event Simulation and Combinatorial Optimization, Budapest*, 2004.

[H^+05] P. E. Heegaard et al. Self-managed virtual path management in dynamic networks. In *Self-* Properties in Complex Information Systems, LNCS 3460*, 2005.

[HK07] P. Herrmann and F. A. Kraemer. Design of trusted systems with reusable collaboration models. In *Proc. of the Joint IFIP iTrust and PST Conferences on Privacy, Trust Management and Security, Moncton*, 2007.

[HS99] G. C. Hunt and M. L. Scott. The coign automatic distributed partitioning system. In *Proc. of the 3rd Symp. on Operating systems design and implementation*, pages 187–200, Berkeley, CA, USA, 1999. USENIX Association.

[HW01] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[HW06a] P. E. Heegaard and O. Wittner. Restoration performance vs. overhead in a swarm intelligence path management system. In *Proc. of the Fifth Int'l. Workshop on Ant Colony Optimization and Swarm Intelligence, Brussels*, 2006.

[HW06b] P. E. Heegaard and O. Wittner. Self-tuned refresh rate in a swarm intelligence path management system. In *Proc. of the EuroNGI Int'l. Workshop on Self-Organizing Systems, LNCS 4124*, 2006.

[KH06] F. A. Kraemer and P. Herrmann. Service specification by composition of collaborations - an example. In *Proc. of the 2006 Int'l Conf. on Web Intelligence and Intelligent Agent Technology, Hong Kong*. IEEE/WIC/ACM, 2006.

[KH07] F. A. Kraemer and P. Herrmann. Transforming collaborative service specifications into efficiently executable state machines. *Electronic Communications of the EASST*, 6, 2007.

[KHB06] F. A. Kraemer, P. Herrmann, and R. Bræk. Aligning uml 2.0 state machines and temporal logic for the efficient execution of services. In *Proc. of the 8th Int'l Symp. on Distributed Objects and Applications (DOA), LNCS 4276, Montpellier*, 2006.

[KIK03] T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using ai planning techniques. In *Proc. of the Int. Parallel and Distributed Processing Symposium*, 2003.

[KSH07] F. A. Kraemer, V. Slåtten, and P. Herrmann. Engineering support for uml activities by automated model-checking - an example. In *Proc. of the 4th Int'l Workshop on Rapid Integration of Software Engineering Techniques (RISE), University of Luxembourg*, 2007.

[KWH08] V. Kjeldsen, O. Wittner, and P. E. Heegaard. Distributed and scalable path management by a system of cooperating ants. In *Proc. of the Int'l. Conf. on Communications in Computing (CIC)*, 2008.

[Mal06] S. Malek. A user-centric framework for improving a distributed software system's deployment architecture. In *Proc. of the doctoral track at the 14th ACM SIGSOFT Symp. on Foundation of Software Engineering, Portland*, 2006.

[Mel06] H. Meling. *Adaptive Middleware Support and Autonomous Fault Treatment: Architectural Design, Prototyping and Experimental Evaluation*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2006.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[S$^+$97] R. Schoonderwoerd et al. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2), 1997.

[WH04] O. Wittner and B. E. Helvik. Distributed soft policy enforcement by swarm intelligence; application to load sharing and protection. *Annals of Telecommunications*, 59, 2004.

[WHH03] O. Wittner, P. E. Heegaard, and B. E. Helvik. Scalable distributed discovery of resource paths in telecommunication networks using cooperative ant-like agents. In *Proc. of the Congress on Evolutionary Computation, Canberra*, 2003.

[WHN05] O. Wittner, B. E. Helvik, and V. F. Nicola. Internet failure protection using hamiltonian p-cycles found by ant-like agents. *Journal of Network and System Management, Special issue on Self-Managing Systems and Networks*, 2005.

[Wit03] O. Wittner. *Emergent Behavior Based Implements for Distributed Network Management*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2003.

[WN04] N. Widell and C. Nyberg. Cross entropy based module allocation for distributed systems. In *Proc. of the 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, pages 187–200, Berkeley, CA, USA, 2004. USENIX Association.

PAPER B

Adaptable Model-based Component Deployment Guided by Artificial Ants

Máté J. Csorba and Poul E. Heegaard and Peter Herrmann

In proceedings of Second International Conference on Autonomic Computing and Communication Systems (Autonomics 2008)

Turin, Italy, Sep 23-25, 2008

ADAPTABLE MODEL-BASED COMPONENT DEPLOYMENT GUIDED BY ARTIFICIAL ANTS

Máté J. Csorba, Poul E. Heegaard, Peter Herrmann
Department of Telematics,
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard, Peter.Herrmann}@item.ntnu.no

Abstract We investigate a means for efficient deployment of distributed services comprising of software components. Our work can be viewed as an intersection between model-based service development and novel network management architectures. In a service engineering context, models of services embellished with non-functional requirements are used as input to our swarm intelligence based deployment logic. Mappings between resources provided by the execution environment and components are the results of our heuristic optimization procedure that takes into account requirements of the services. Deployment mappings will be used as feedback towards the designer and the provider of the service. Moreover, our heuristic algorithm possesses significant potential in adaptation of services to changes in the environment.

1. Introduction

In the process of realizing a service system several important decisions have to be made that will affect performance of the system as well as the quality of service (QoS) perceived by its user. A state-of-the-art method to develop distributed services implemented as software systems is starting from a platform independent model and realizing the service following a top-down step-wise refinement approach. A significant factor influencing the perceived QoS from the user's perspective is the deployment model of the particular service, in other words the configuration of the building-blocks of the service and their mapping to run-time processing elements and resources required for execution.

Nodes hosting a service may consist of heterogeneous hardware and may provide a dynamic environment for the services being executed, i.e. nodes can join and leave the network in an unpredictable manner. The evolving nature of the context of distributed services and mobility of clients requires the capability of adaptation to satisfy QoS requirements while also considering costs on the service provider's side. The wide range of possible requirements against the service makes the deployment and adaptation problem a multi-faceted challenge demanding multi-dimensional optimization.

The methodology we apply to solve the deployment problem can be viewed as an intersection between systems development and novel network management solutions.

There have been a couple of promising developments providing platforms for adaptivity and dependability. Autonomous replication management mainly for dependability is targeted by Meling in a framework based on group communication systems [Mel06]. A distributed dynamic middleware, QuAMobile is presented in [LSO$^+$07] that introduces independent application variants and selects between them for context-awareness and adaptation. Planning is based on service level agreements (SLAs) and QoS-aware metadata in the service model is used in the planning-based adaptation middleware of the MUSIC project (cf. [REF$^+$08]). A peer-to-peer middleware, CARISMA is utilizing an auctioning-like mechanism for conflict resolution and adaptation automatically triggered by context changes [CEM03]. In the QoS brokering approach from Menasce and Dubey consumers can request services, after which the broker uses analytic queuing models to predict QoS of the services under various workloads, thus looking for maximized utility [MD07].

Traditional techniques were applied for configuration of server environments such as fuzzy learning, e.g. Xu et al. applied a two-level control mechanism targeting efficient resource utilization in [XZF$^+$07]. Layered queuing networks are employed for generating optimal configurations and policies by Jung et al. in an offline framework [JJH$^+$08]. Some existing approaches have addressed the improvement of perceived QoS through changing the deployment of applications, however due to the exact solution algorithms, complexity becomes NP-hard already with more than 2-3 hosts or several QoS dimensions restricting applicability of these methods. A review of approximative solutions trying to overcome scaling problems, such as greedy algorithms, genetic programming, can be found in [Mal06]. These approaches try to maximize utility of a service purely from the users' perspective, whereas we aim to formulate and solve the deployment problem from the providers perspective while also considering the users perception of QoS. Besides, we aim to handle the deployment of multiple services at the same time.

We are building a logic that brings QoS-awareness into the development cycle, and that can manage the deployment of services and adapt to the context once these services are executed in a real environment. The application of our approach should grant the same benefits that exist in distributed management architectures, such as increased dependability, better resource utilization, etc. Moreover, the output of the logic presented is platform independent, thus it can drive a suitable middleware platform and will eventually allow adaptation to the changing environment of the modelled services that are executed.

It is an important design criteria that the deployment logic should allow execution in a fully distributed manner, thus it shall not be prone to deficiencies of existing centralized algorithms, such as performance bottlenecks and single point of failures. Besides, it is desirable to omit the burden of keeping a centralized decision logic updated and synchronized and this way to achieve better reaction times for context-awareness. Consequently, our aim is to develop a method supporting run-time component (re-)deployment that allows execution of services within the allowed region of external parameters defined by the service requirements. Considering all these aspects we approach the problem using a distributed, robust and adaptive routing system called the Cross Entropy Ant System (CEAS) [HW01, HHW08]. The CEAS is an Ant Colony Optimization (ACO) system as introduced by Dorigo et al. [DMC96], which is a multi-agent system for solving a wide variety of combinatorial optimization problems where the agents' behavior are inspired by the foraging behavior of ants. Examples of successful application in communication systems are load-balancing (Schoonderwoerd et al. [$S^{+}97$]), routing in wired networks by AntNet [CD98], and routing in wireless networks by AntHocNet [CDG05].

In [CHH08] we presented our novel approach for the efficient deployment of software components taking into account QoS requirements captured during the modelling phase. The procedure starts from high-level QoS goals and, through requirement profiles, utilizes swarm intelligence to provide solutions and to aid dynamic deployment. In [CHH08] this distributed approach was tested on component deployment in a static topology and compared with centralized deployment approaches. In this paper, we extend the approach to allow deployment of multiple service components simultaneously that adapt to changing topologies.

The remainder of this paper is organized as follows. The next section will present how the deployment logic fits into the development cycle. In Sect. 3 an introduction to CEAS, which is used throughout the paper as the basis of our heuristic optimization method, will be given. Next, in Sect. 4 we present our proposed solution giving the algorithm. After that, Sect. 5 sets the example scenario of three concurrent services. The results related to the examples are evaluated in Sect. 6. Finally, in Sect. 7 we conclude and touch upon our future work.

2. Deployment cycle

The deployment approach we are proposing will extend the development cycle SPACE. SPACE is devoted to the rapid and correct engineering of distributed services [KH06]. The stepwise modelling and refinement of the models is depicted in Fig. 1. First, a purely functional service model is created, which is collaboration-oriented meaning that the service specification is not a composition of descriptions of physical software components realizing the service. Instead, the collaboration-oriented specification is built from models of distributed sub-functionalities fulfilling — in interaction — the complete service behavior. The functional service model is specified by UML collaborations and activities. One of the advantages of this specification style is that it enables service modelling by reusing building blocks from collections of domain specific model libraries to a significantly higher degree than it would be possible with component-based descriptions [HK07].

Service models undergo correctness checks, as described in [KSH07], before they are transformed to a component-oriented design model by model transformation [KH07]. Next, using the component-oriented model, specified as UML state machines, code generators are used to create executable Java code enabling automated transformation of collaboration-oriented service models to executable implementations [KHB06].

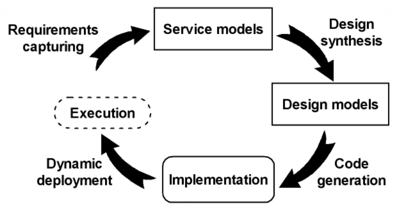


Figure 1: Development with SPACE

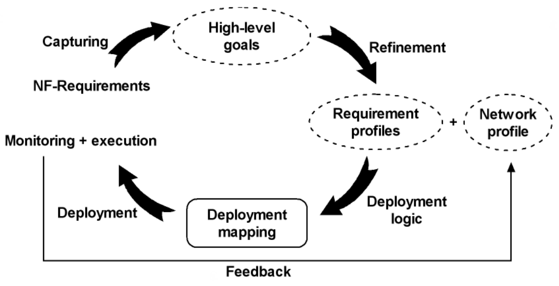


Figure 2: Deployment Support for SPACE

The dynamic deployment of the generated implementation is the step where the logic we propose can interact with the development cycle. The additional steps to support efficient deployment of the components that build up the service is shown in Fig. 2. In our deployment support cycle service models are amended by *high-level non-functional (NF) goals* that define non-functional requirements (NFRs) of the service being modelled in a rather abstract manner. Refinement of NF goals can be done in parallel with the transformation of service models to design models. *Requirement profiles* obtained in this step specify NFRs of the service components. In addition, a *network profile* is added representing provided properties describing the target environment the service will be executed in. Our deployment logic will be launched using these two profiles as input with requirements specifying the search goals and the *network profile* specifying the search space.

QoS requirements relevant to the service model are captured in a collaboration-oriented style design time, as a translation of traditional service level agreements. In NFRs, usually properties related to security, performance, availability, portability, etc. are addressed. More importantly, our view is that the deployment logic proposed will be able to handle any non-functional property of the service, as long as a suitable cost function is provided for the specific properties at hand. This feature will be provided by exploiting the advanced scalability of CEAS and the method of pheromone sharing.

Fig. 3 depicts a simple example of a collaboration between two components. This collaboration is enriched with NFRs for both the components and for the collaboration binding them. This basic collection of requirements contains two types of cost values, execution costs (f_{c_i}) and communication costs (f_{k_j}). The total number of cost values is equivalent to the total number of components in the service (E), plus the number of collaborations between them (K), i.e. $f_{c_i}, i = 1 \ldots E$ and $f_{k_j}, j = 1 \ldots K$. The execution cost is a local cost imposed on the host node or resource executing the particular component after deployment, whereas the communication cost loads the communication link between the two components involved in the collaboration. This simple example of collaboration-oriented specification and requirement capturing will be illustrated in the examples in Sect. 5.

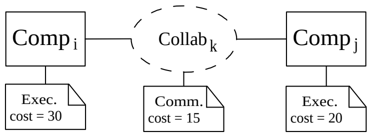


Figure 3: Collaboration with NFRs

Currently, existing deployment strategies and various approaches to aid deployment of software systems, e.g. ontology-based and reasoning engines, apply centralized decision logics based on centrally maintained databases. The disadvantages of approaching the problem this way are the burden of keeping a central database constantly updated and synchronized and the single point of failure introduced to the system. Accommodating the decision logic together with the central database on a single node may introduce bottlenecks both communication wise and storage wise.

In contrast to centralized approaches a distributed cooperative algorithm employs (semi-)autonomous agents, which cooperate to achieve certain common goals. To avoid the need for any type of global knowledge in deployment mapping, we employ autonomous agents operating in a distributed environment with their decisions based solely on information that is available locally to the place where they reside. At every node under the provision of the deployment logic some sort of shared memory is required that will be the vehicle for cooperation between the agents. Accordingly, the information required for optimization in our logic is distributed across all participating nodes. This property of the deployment mapping system contributes to robustness, scalability and fault tolerance. Furthermore, we intend to use the same logic first to obtain initial, optimal mapping of service components to hosts or resources, and second to guide necessary changes during execution of a service to satisfy the requirements it was launched with.

The objective of each ant species is to find either the optimal deployment mapping of component instances c_i onto nodes n_j or at least to find a mapping that satisfies the requirements within reasonable time. A component, $c_i \in \mathbf{C}$ ($\mathbf{C}$ is the set of components that together provide the service the species is responsible for) can have various properties and restrictions can apply regarding it's prospective hosts. For example, deployment of c_i can be restricted at node $n \in \mathbf{N}$ ($\mathbf{N}$ is the set of available nodes) as well as prescribed binding is allowed. A component can be bound to a node explicitly, this results in being excluded from the set of components that are available for the logic to be freely mapped to any available node. However, bound components are also taken into account by the ants during calculation of the resulting cost of a particular deployment mapping at a given iteration.

The basis of the heuristics, guiding the ants towards an optimized mapping, is the cost function $F(M)$ that is used to evaluate the resulting suggestion, $M : \mathbf{C} \rightarrow \mathbf{N}$, for deployment mapping. The ants target to minimize the cost calculated using $F(M)$ at every iteration, while the constraints given by the *mapping scopes* also have to be taken into consideration, i.e. $\mathbf{R}_i \subseteq \mathbf{N}$ for each component instance i. $\mathbf{R}_i$ is characterized by the policies given by the service provider (e.g. service level agreements of ISPs), as well as the access restrictions, the provided and requested capabilities (soft costs) and provided and requested capacity requirements (hard costs, e.g. bandwidth limitations). Using $\mathbf{R}_i$ component binding can easily be expressed assigning a single node to the set, thus restricting the search space.

Besides the *requirement profiles*, the service provider must provide the *net-map*, $\mathbf{N}$ for the decision logic as well, specifying the available nodes and links. Two types of constraints that can influence the optimal mapping are distinguished in the model. Constraints assigned to nodes and to links. For the latter type, constraints generally represent the cost of using the link for connecting two components, which have a functionality in the model requiring interaction between them. Constraints assigned to nodes or other resources related to execution of a component can, for instance, represent memory size limitations. Also, these constraints can be interrelated in a way that, for example, placement of a component on a node can lower the available amount of different types of resources at once with an amount depending on the available resources at the time of the mapping.

In the subsequent section the stochastic optimization background is introduced that is used throughout our logic to specify an algorithmic solution for the deployment problem.

3. Cross Entropy Ant System

The key idea is to let many agents, denoted *ants*, iteratively search for the best solution according to the problem constraints and cost function defined. Each iteration consists of two phases; the *forward* ants search for a solution, which resembles the ants searching for food, and the *backward* ants that evaluate the solution and leave markings, denoted *pheromones*, that are in proportion to the quality of the solution. These pheromones are distributed at different locations in the search space and can be used by forward ants in their search for good solutions; therefore, the best solution will be approached gradually. To avoid getting stuck in premature and sub-optimal solutions, some of the forward ants will explore the state space freely ignoring the pheromone values.

The main difference between various ant-based systems is the approach taken to evaluate the solution and update the pheromones. For example, AntNet [CD98] uses reinforcement learning while CEAS uses the *Cross Entropy (CE) method for stochastic optimization* introduced by Rubinstein [Rub99]. The CE method is applied in the pheromone updating process by gradually changing the probability matrix $\mathbf{p}_r$ according to the cost of the paths. The objective is to minimize the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. For a tutorial on the method, [Rub99] is recommended.

The CEAS has demonstrated its applicability through a variety of studies of different path management strategies [HHW08], such as shared backup path protection, p-cycles, adaptive paths with stochastic routing, and resource search under QoS constraints. Implementation issues and trade-offs, such as management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism [H$^+$04] and self-tuned packet rate control [HW06]. Additional reduction in the overhead is accomplished by pheromone sharing [KWH08] where ants with overlapping requirements cooperate in finding solutions by (partly) sharing information.

In this paper, the CEAS is applied to obtain the best deployment mapping $M : \mathbf{C} \rightarrow \mathbf{N}$ of a set of components, $\mathbf{C}$, onto a set of nodes, $\mathbf{N}$. The nodes are physically connected by links used by the ants to move from node to node in search for available capacities. A given deployment at iteration r is a set $\mathbf{M}_r = \{\mathbf{m}_{n,r}\}_{n\in\mathrm{N}}$, where $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ is the set of components at node n at iteration r. In CEAS applied for routing the path is defined as a set of nodes from the source to the destination, while now we define the path as the deployment set $\mathbf{M}_r$. The cost of a deployment set is denoted $F(\mathbf{M}_r)$. Furthermore, in the original CEAS we assign the pheromone values $\tau_{ij,r}$ to interface i of node j at iteration r, while now we assign $\tau_{mn,r}$ to the component set m deployed at node n at iteration r. In Sect. 4 we describe the search and update algorithm in details.

In CEAS applied for routing and network management, selection of the next hop is based on the *random proportional rule* presented below. In our case however, the *random proportional rule* is applied for deployment mapping. Accordingly, during the initial exploration phase, the ants randomly select the next *set of components* with uniform probability $1/E$, where E is the number of components to be deployed, i.e. the size of $\mathbf{C}$, while in the normal phase the next set is selected according to the *random proportional rule* matrix $\mathbf{p}_r = \{p_{mn,r}\}$, where

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l\in\mathbf{M}_{n,r}} \tau_{ln,r}} \tag{1}$$

A parameter γ_r denoted the *temperature*, controls the update of the pheromone values and is chosen to minimize the performance function

$$H(F(\mathbf{M}_r), \gamma_r) = e^{-F(\mathbf{M}_r)/\gamma_r} \tag{2}$$

which is applied to all r samples and the expected overall performance satisfies

$$h(p_{mn,r}, \gamma_r) = E_{\mathbf{p}_{r-1}}(H(F(\mathbf{M}_r), \gamma_r)) \geq \rho \tag{3}$$

$E_{\mathbf{p}_{r-1}}(X)$ is the expected value of X s.t. the rules in $\mathbf{p}_{r-1}$, and ρ is a parameter (denoted search focus) close to 0 (typically 0.05 or less). Finally, a new updated set of rules, $\mathbf{p}_r$, is determined by minimizing the cross entropy between $\mathbf{p}_{r-1}$ and $\mathbf{p}_r$ with respect to

γ_r and $H(F(\mathbf{M}_r),\gamma_t)$. Minimized cross entropy is achieved by applying the random proportional rule in (1) for $\forall_{mn}$ with

$$\tau_{mn,r} = \sum_{k=1}^{r} I(l \in \mathbf{M}_{n,r})\beta^{\sum_{j=k+1}^{r} I(j\in\mathbf{M}_k)} H(F(\mathbf{M}_k),\gamma_r) \tag{4}$$

where $I(x) = 1$ if x is true, 0 otherwise. See [Rub99] for further details and proof.

To avoid centralized control and synchronized batch oriented iterations, in CEAS the cost value $F(\mathrm{M}_r)$ is calculated *immediately* after each sample, i.e., when all components are mapped, and an auto-regressive performance function, $h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1-\beta)H(F(\mathrm{M}_r),\gamma_r)$ is applied approximated by

$$h_r(\gamma_r) \approx \frac{1-\beta}{1-\beta^r}\sum_{i=1}^{r}\beta^{r-i}H(F(\mathrm{M}_r),\gamma_r) \tag{5}$$

where $\beta \in < 0,1 >$ is a memory factor weighting (geometrically) the output of the performance function. The performance function will smoothen variations in the cost function, hence rapid changes in the deployment mapping and undesirable fluctuations will be avoided. This mechanism helps cooperation between the species.

As for the CE method, the temperature γ_r is determined by minimizing it subject to $h(\gamma) \geq \rho$. In [HW01] it is shown that the temperature equals

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r}\sum_{i=1}^{r}\beta^{r-i}H(F(\mathbf{M}_i),\gamma) = \rho\} \tag{6}$$

However (6) is a complicated (transcendental) function that is both storage and processing intensive since all observations up to the current path sample, i.e. the entire path cost history $F(\mathbf{M}_r) = \{F(\mathbf{M}_1),\cdots,F(\mathbf{M}_r)\}$ must be stored, and weights for all observations have to be recalculated. In an on-line operation of a network node, such resource requirements are impractical. Instead it is assumed, given a β close to 1, that the changes in γ_r are typically small from one iteration to the next. This enables a first order Taylor expansion of (6), and a second order Taylor expansion of (4), see [HW01, Wit03] for more details.

4. Distributed deployment logic

Our deployment logic can be considered as a swarm of independent ant-like agents executing an optimization task continuously in the target network hosting the service we model. This continuous behavior contributes to the advantage of our approach, i.e. that the same logic provides an initial static mapping and can be used for online redeployment. In this paper we extend the deployment approach to handle multiple services deployed simultaneously by allowing interoperation of artificial ant species, each of them representing a particular service realized by distributed software components. More importantly, beyond achieving a scalable extension for multiple services we target other scalability issues as well. By using a new cost function for evaluating solutions during the heuristical optimization process we eliminate the need for any global knowledge, i.e. species can now be launched and operate independently from each other without having a global view on the requirements set for all the services in the system. Another step towards scalability is to limit the ants to visit only those nodes within the network-map that are effectively used for deploying components of the service represented by their species. This way, we address significantly larger problem sizes, consisting of a higher number of nodes and simultaneous services.

Initially, each ant is assigned a task of deployment of $\mathbf{C}$ components stemming from the set of components of the service represented by its species. After initialization the ants start a random-walk in the network of nodes, described by the *net-map*, selecting every next hop randomly. After an ant arrives at a node its behavior depends on if it is a *explorer* or a *normal* ant. The latter type of ant uses the corresponding instance of the distributed pheromone database at the node to select a subset, $\mathbf{m}_{n,r}$, of $\mathbf{C}$ for mapping and stores this selection in a mapping list. The mapping list $\mathbf{M}_r$ is carried along by the ant during its search. On the contrary, an *explorer* ant selects a subset $\mathbf{m}_{n,r}$ based on a random decision without using the pheromone values available at the node. The ratio of *explorer* ants can be regulated and this type of ants are used to initially explore the *net-map* and also to cover up fluctuations, e.g. new nodes appearing, in the network later on in the optimization process. This type of exploration can be considered as random sampling from the problem space and results in a random cost figure. The number of iterations in the initial *exploration* phase depends on the problem size, however, the end of this phase can be detected by monitoring the pheromone database size as it extends with the growing number of possibilities covered up by the ants doing a random-walk in the problem space. After the *normal* phase starts only a fraction of the ants, e.g. 5-10%, are flagged as *explorers*, allowing for the required responsiveness to changes in the environment, while *normal* ants are focusing on finding the optimum.

To support finding the optimal deployment mapping for multiple concurrent services a means of interoperation is needed among the species responsible for the different services being deployed. As we consider optimal deployment of services from the provider's perspective we target balancing of execution costs imposed on the nodes that are used for execution of the services, or in other words load-balancing, which generally requires global overview of the system's operating conditions. Nevertheless, we want to avoid any centralized structure, and use a completely distributed optimization method. For this reason, we have introduced a processing power reservation mechanism that has to be implemented in every node in addition to the pheromone database. The different ant species use this allocation mechanism to indicate their latest resource usage in a node at iteration r. As ants from every species use the allocation in every node they actually use for deployment mapping, this mechanism will provide interaction between the components. Sampling the current sum of allocations in every visited node can give a general overview for the ants, thus load-levels in participating nodes can be incorporated into the cost calculations at the end of each iteration. We refer to load-level samples taken during an ant run with the set $\mathbf{NL}_r$. Samples that suggest exceeding the capacity of a node are quickly outranked by better solutions as a high penalty is assigned to infeasible solutions. The actual implementation of sampling is left to the middleware. Allocation entries that are outdated are invalidated to preserve consistency.

After the forward search is over, i.e. an ant managed to come up with a mapping for all the components it was assigned with, the resulting mapping can be found in the set $\mathbf{M}_r$ and will be evaluated using the cost function of the service. How to formulate the cost function $F()$ depends on the NFRs that the service model is extended with. Currently, we use two parameters for the cost function, the deployment mapping set $\mathbf{M}_r$ and the load-level samples taken during an iteration $\mathbf{NL}_r$ and we consider execution and communication costs derived from the service model as introduced in Sect. 2. Thus, our cost function consists of two components, node related costs (**NC**) and link, i.e. collaboration related costs (**LC**). The aim is to minimize the overall value of (7).

$$F(\mathbf{M}_r, \mathbf{NL}_r) = [\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NC}(n_j)] \cdot (1 + x \cdot \mathbf{LC}) \tag{7}$$

where x is a parameter. $F(\mathbf{M}_r, \mathbf{NL}_r)$ is used by all species the same way, and has a component strictly local to the species, **LC**, which incorporates the collaboration costs

$$\mathbf{LC} = \sum_{j=1}^{K} I_j \, f_{k_j} \tag{8}$$

where I_j is an indicator function to sum all the communication costs of the collaborations that happen between different nodes

$$I_j = \begin{cases} 1, & \text{if } k_j \text{ external} \\ 0, & \text{if } k_j \text{ internal to a node} \end{cases} \tag{9}$$

The first component, $\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NC}(n_j)$, of the overall cost function is related to node local costs and aims to incorporate load-balancing among the nodes providing the services being executed. Furthermore, it is important to note that only those nodes that are visited during the search phase in iteration r are included in the hop-list, $\mathbf{H}_r$. The node related cost is calculated individually for every visited node according to

$$\mathbf{NC}(n_j) = [\sum_{i=0}^{\mathbf{NL}_{n,r}(n_j)} \frac{1}{\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NL}_{n,r} + 1 - i}]^y \tag{10}$$

Equation (10) calculates the execution costs for node n_j based on the load-levels sampled and it is the basis for counteracting the cost component **LC**. On one hand, **LC** tries to put weight on component mappings that have as much as possible of the collaborations within the same node(s) by favoring mappings that use less nodes for deployment with a low cost value. This way minimizing external communication. On the other hand, Equation (10) has an effect of distributing components, thus equalizing execution load among the available hosts to the highest extent possible. This way two counteracting requirement types are tackled in the same cost function. The exponent **y** (in Equation (10)) allows to focus more on load-balancing instead of minimization of collaboration costs by selecting a larger value, while the multiplier **x** (in Equation

(7)) can be used to scale collaboration costs if needed. We generally use $\mathbf{x} = 0.1$ and $\mathbf{y} = 2$, as well as with the cost values in the example scenario presented in Sect. 5.

The cost function (7) is used at the end of an iteration to evaluate the mapping found by the ant. Thereafter, the ant travels backward along the path stored in the hop-list $\mathbf{H}_r$. This mechanism is called *backtracking*. During *backtracking* the pheromone values are updated according to Equation (4). This ends the behavior of a single ant and unless a stopping criteria is met a new ant can be initiated and emitted. There are different options for constructing a stopping criteria. One can be for example the observation of the moving average of the evolving cost value and detecting convergence to a suggested solution. Another option is sampling the size of the distributed pheromone database during an iteration. Convergence can be detected by observing a very strong pheromone value that will emerge in the database, while inferior solutions will evaporate.

It is important to note that the same ant behavior can be used for all the species, i.e. for all the services being deployed simultaneously. The described process is summarized in Algorithm 1.

Algorithm 1 Deployment mapping of **C**

```
1  Select the initial node $n \in \mathbf{N}$ where the search will start randomly.
2  Select a set of components $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ which satisfies $n \in \mathbf{R}$ for every $c_i \in \mathbf{m}_{n,r}$ according to the random proportional rule (normal ant), Equation (1), or in a totally random manner (explorer ant). If such a set cannot be found, goto step 7.
3  Update the ant's deployment mapping set, $\mathbf{M}_r = \mathbf{M}_r + \{\mathbf{m}_{n,r}\}$.
4  Update the set of components to be deployed, $\mathbf{C} = \mathbf{C} - \mathbf{m}_{n,r}$.
5  (Re-)allocate processing power at the current node, $n$ according to $f_{c_i}, \forall c_i \in \mathbf{m}_{n,r}$.
6  Sample the estimated load-level, $\mathbf{nl}_{n,r}$ at the current node $n$, and $\mathbf{NL}_r = \mathbf{NL}_r + \{\mathbf{nl}_{n,r}\}$.
7  Select next node, $n$ randomly and add $n$ to the hop-list $\mathbf{H}_r = \mathbf{H}_r + \{n\}$.
8  If $\mathbf{C} \neq \emptyset$ then goto 2., otherwise evaluate $F(\mathbf{M}_r, \mathbf{NL}_r)$ using the mapping set $\mathbf{M}_r$ and the samples taken ($\mathbf{NL}$).
9  Update the pheromone values, Equation (4), corresponding to the $\{\mathbf{m}_{n,r}\} \in \mathbf{M}_r$ mappings going backwards along $\mathbf{H}_r$.
10 If stopping criteria is not met then start new iteration (increment $r$), initialize and emit new ant and goto 1.
```

For optimization to be successful the pheromone values have to be aligned with the sets of deployed components. During an iteration each ant visits $\mathbf{n} \subseteq \mathbf{N}$ nodes and will form n discrete sets from the available components ($\mathbf{C}$) carried along. At the end of an iteration the suggested deployment mapping, $\mathbf{M}_r$ is evaluated. The pheromone database for each species is built by assigning a flag to every component that is free for deployment mapping, i.e. which is not bound to a specific node by requirements. Thus, the number of components available for the species will be $E^* \subseteq E$, and the size of the pheromone database becomes 2^{E^*}, equal to the number of possible combinations for a set at a node, which is specific for each service. Accordingly, the physical requirement for an execution platform supporting our approach is to accommodate 2^{E^*} floating point numbers at every node. If the pheromone database in a node is normalized between $0 \ldots 1$ it can be observed as a probability distribution of component sets mapped to that node by the artificial ants. Once a converged state is reached the optimal solution(s) emerge with probability one.

Indexing of the pheromone database can be done using component set identifiers. For example, consider a basic set of 5 components in a service, $\mathbf{C} = \{c_1, c_2, c_3, c_4, c_5\}, E = 5$. Then indexing is done using an E long binary bitstring. In this case, e.g. element 17 of the pheromone database, which is equivalent to $'10001'B$, refers to the deployment of components c_1, c_5 at the current node. Besides, we propose to use a dynamically allocated pheromone database based on thresholds that can be used for evaporating pheromone entries under a given significance level to achieve better scalability. Currently, we apply a threshold of 1%, i.e. pheromone values lower than 1% of the highest value are considered insignificant and are eliminated from the database.

5. Design examples

In this section we introduce 3 different service models for demonstrating the deployment logic. The first example has been introduced originally in [KH06]. $S1$ has a component that operates a security door and a card reader with a keycode entry panel. The two latter components are bound to n_1 by requirements. Besides, a central component administers access rights by using an authentication and a authorization server with corresponding databases as separate components (Fig. 4).

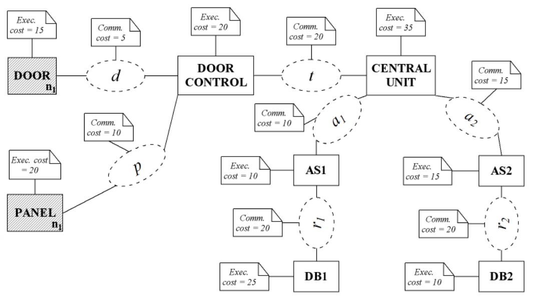


Figure 4: S1 - The Access Control System

The second example, Fig. 5 models a video surveillance system that has one surveillance camera component bound to each of the five nodes by default. A central control and a recording unit manages the system and uses a main and a backup storage device for storing surveillance information in a replicated database.

*S*3, the third service is a model of a process controller that consists of 4 main stages of processing. In addition, S3 has a main generator component that produces the input impulses for the processing stages and a logging module monitoring the output of the four stages. On top of that, a user interface component can be used for direct human interaction with the system (Fig. 6). In *S*3 all the components can freely be mapped to any of the nodes in *N*, depending on current availability of resources.

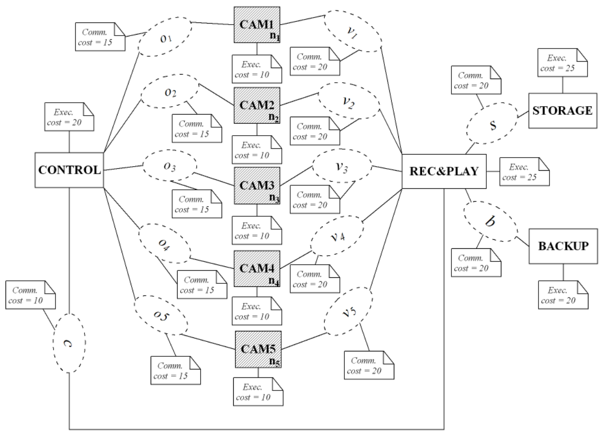


Figure 5: S2 - The Video Surveillance System

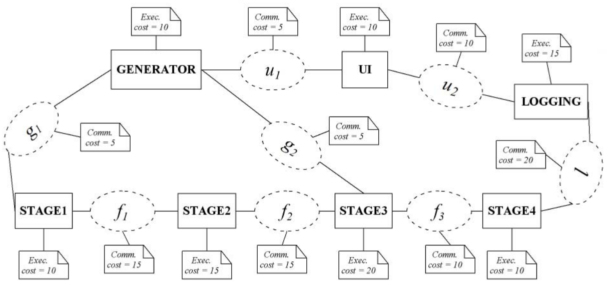


Figure 6: S3 - The Process Controller System

An ant species is assigned to each of the services and deployment mapping is conducted on the underlying network of hosts, which consists of 5 nodes with equivalent capabilities in the example setting. The execution and collaboration costs assigned to each element of the models are summarized in Table 1.

The behavior and the output of our artificial intelligence approach will further be evaluated in the next section.

Table 1: Components and costs in the examples

S1 - Access Control		S2 - Surveilliance		S3 - Process Control.	
ci / kj	fc / fk	ci / kj	fc / fk	ci / kj	fc / fk
DOOR	15	CAM1..5	10	GENERATOR	5
PANEL	20	CONTROL	20	STAGE1	10
DOOR CTRL	20	REC./PLAY	25	STAGE2	15
CENTRAL UNIT	35	STORAGE	25	STAGE3	20
AS1	10	BACKUP	20	STAGE4	10
AS2	15			LOGGING	15
DB1	25			UI	10
DB2	10				
d	5	o1..5	15	g1..2	5
p	10	v1..5	20	u1	5
t	20	c	10	u2	10
a1	10	s	20	f1	15
a2	15	b	20	f2	15
r1	20			f3	10
r2	20			l	20

6. Evaluating the scenario

The service deployment mapping problem with execution and communication costs can be NP-hard even in case of a single service (c.f. [CHH08]). The example provided here has multiple optimal and near-optimal solutions with different sets of components deployed on various nodes. However, it is important to recall that we are interested in providing solutions satisfying the requirements in reasonable time and not necessarily in always finding the optimum. For demonstration a solution taken from the output of the logic is shown in Table 2.

Table 2: Example deployment mapping

	S1	S2	S3
n1	DOOR, PANEL	CAM1	STAGE4, LOGGING
n2	AS2, DB2	CAM2, REC/PLAY, STORAGE, BACKUP	
n3	DOOR CTRL.	CAM3	STAGE1, STAGE2, STAGE3
n4	AS1, DB1	CAM4	UI
n5	CENTRAL UNIT	CAM5, CONTROL	GENERATOR

To evaluate the algorithm we propose, first we compare it to two different approaches. In the first one (denoted *globT*, *allnodes*) ants use a simpler cost function, Equation (11) that is easier to calculate, but requires the shared knowledge of the sum of all offered execution costs, T.

$$F(\mathbf{M}_r) = \sum_{\forall n \in N} |\mathbf{nl}_{n,r} - T| + \sum_{j=1}^{K} I_j \, f_{k_j} \tag{11}$$

That means that to apply (11) all of the species associated to the multiple services being deployed simultaneously have to be aware of each others total processing power demand and incorporate it into T. Knowing the global constant, T, ants can calculate the deviation of the execution load from a global average, i.e. share the load among the participating nodes. This is included in the first part of (11). The second part of this cost function includes collaboration costs the same way as in Equation (8). For details see [CHH08].

The second approach used for comparison (denoted *globT*) uses the same cost function, i.e. Equation (11), with the exception that ants are not required to visit and sample all the nodes available in the *net-map*, only those that are actually used for deployment by their species, i.e. we use $n \in \mathbf{H}_r$ instead of $\forall n \in N$. This is a significant difference with respect to scaling as the set of available nodes, N can be high thus putting a heavy burden on the ants that have to visit and sample all of the nodes. Our approach, Algorithm 1, aims to provide deployment mapping without requiring any *global* prerequisite and also without driving ants to cover $\forall n \in N$, which contributes to increased scalability. Algorithm 1, that uses Equation (7) as cost function, is denoted *distF* throughout this section.

Table 3: Number of iterations until convergence

	avg	stdev
globT, all nodes	23679	14795
globT	18487	5469
distF, 5%	8234	12800

First, in Table 3 we compare the amount of iterations, i.e. ant runs executed before reaching a converged state. The results are derived from the output of 100 runs of each approach and we conclude that *distF* requires significantly less iterations to converge compared to the semi-*global* approaches. The 5% indicates the percentage of explorer ants used during the *normal phase* that is standard procedure to achieve context-awareness by constantly allowing some of the ants randomly explore the *net-map*.

It can be noted that if adaptability to events such as nodes appearing/disappearing or link fluctuations is not necessary, hence an initial deployment mapping satisfying the NF-requirements is sufficient, the search for a static mapping can even more be accelerated.

In Table 4 we compare the average and the deviation of the converged cost values produced by the three different approaches solving the simultaneous deployment mapping of the example services $S1$, $S2$ and $S3$. For this comparison the converged deployment mapping suggested by the different approaches is observed and evaluated by applying the cost function Equation 7.

Table 4: Average cost of mapping

	S1	S2	S3
globT, all nodes, avg	2.0246	4.1725	1.7251
globT, all nodes, stdev	0.0513	0.1754	0.297
globT, avg	1.87	4.7977	1.7033
globT, stdev	0.1272	0.2275	0.1818
distF, 5%, avg	1.7945	4.3021	1.5932
distF, 5%, stdev	0.0943	0.1773	0.0588

We can see that the deviation of the costs values in *distF* is at least as low as in case of the least scalable approach, *globT* with *all nodes* sampled. Besides, the actual cost values for *distF* are the best for services *S*1 and *S*3, and for *S*2 it is between the least scalable and the scalable *globT* versions. Accordingly, our algorithm works without the requirement for any *global* knowledge and using a more scalable sampling of nodes while producing equivalent or better results for the deployment problem.

To evaluate the capability of the logic to adapt to changes in the context we have investigated two simple scenarios with the example services. In the first scenario a single node failure occurs and sometime later on the node is repaired and operational again. However, we experiment with *soft-errors* meaning that deployment to the selected node will be disallowed for the species after the error event occurs, but the node will still be operational for the components that are bound to it by requirements. We allow this exception for those services that would otherwise go down and hence the deployment mapping would be meaningless for them (e.g. *S*2, which has a component bound to each of the five nodes). The second scenario introduces an additional new node to the network after the species have converged to a solution with 5 nodes.

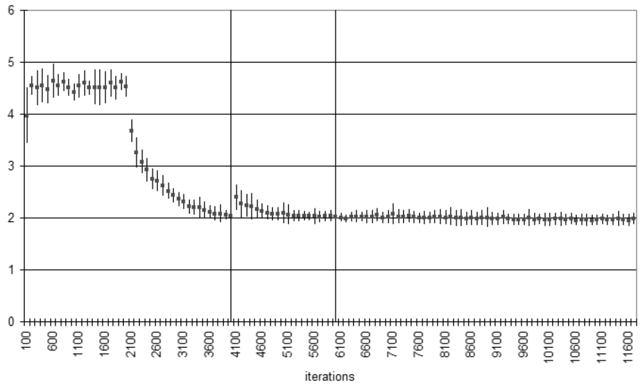


Figure 7: Costs for S1 with node error and repair

The results of injecting a node error followed by a repair of the same node later on can be seen in Figures 7, 8 and 9. These figures show how the cost values found by the three species evolve. The figures display the average (as dots) and the deviation (as error bars) of results from 100 runs of the same scenario. The first 2000 iterations represent the initial *exploration* phase with considerably high costs, but as the *normal* phase starts from iteration 2001 the costs of component mappings start to decrease as species start to cooperate and better and better solutions are found. Vertical markers show the events of node n_3 going down and then coming back to operation. We can see that the cost values increase for every service after a node goes down, but interestingly species for *S*1 and *S*3 are able to find an almost equally low cost level by redeploying their components on the remaining nodes. Species corresponding to *S*2 in turn remains at a slightly higher cost level because it has a component bound to the erroneous node thus it has to face degraded load-sharing among the nodes. After n_3 is repaired components of $S2$ can be re-mapped to achieve a lower cost level again with 5 nodes, which happens within a few iterations.

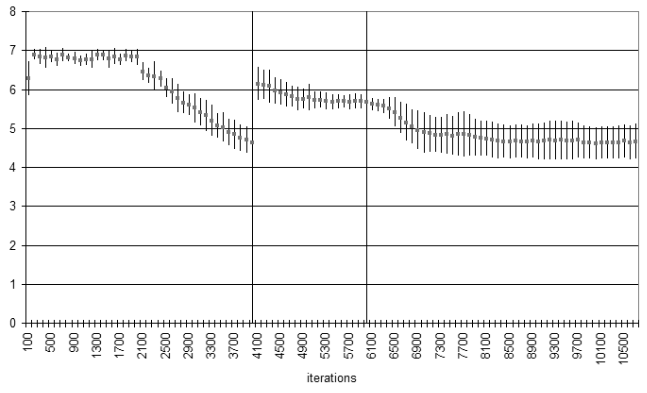


Figure 8: Costs for S2 with node error and repair

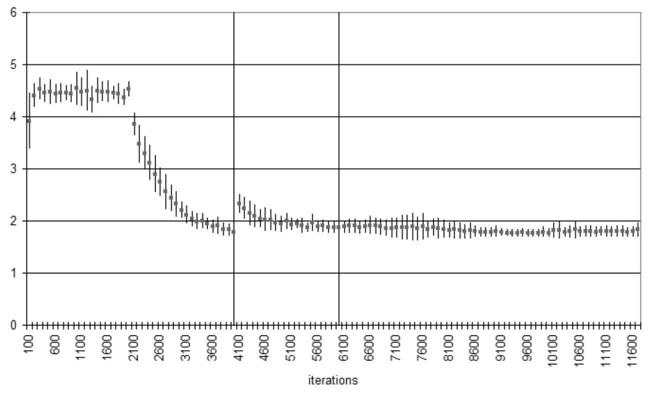


Figure 9: Costs for S3 with node error and repair

It is also interesting to observe the control parameter γ_r, i.e. the temperature that governs the performance function during a run with error and repair events. In Figure 10 we can observe the changes in the temperature and see how species react to changes in the environment. First, the node failure is detected very quickly and the temperature increases as lower cost solutions disappear from the solution space. The temperature increase is significant in $S2$ that is mostly affected by the change but it is also noticeable for $S3$, whereas $S1$ seems to be slightly influenced. After the repair has been made the swarm reacts slower only after some iterations can we observe decreasing temperatures. $S3$ notices the better conditions first and $S2$ follows. More iterations later all species converge to a stabilized state with lowest temperatures.

Our second test scenario introduces a 6th node to the environment of the services. The 3 species are already in a converged state when the new node appears, indicated by a vertical line in Figure 11. Some iterations later explorer ants, which represent 5%

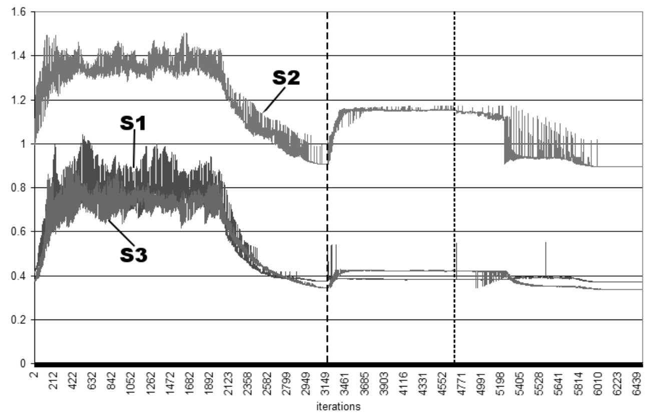


Figure 10: Temperature within the 3 species

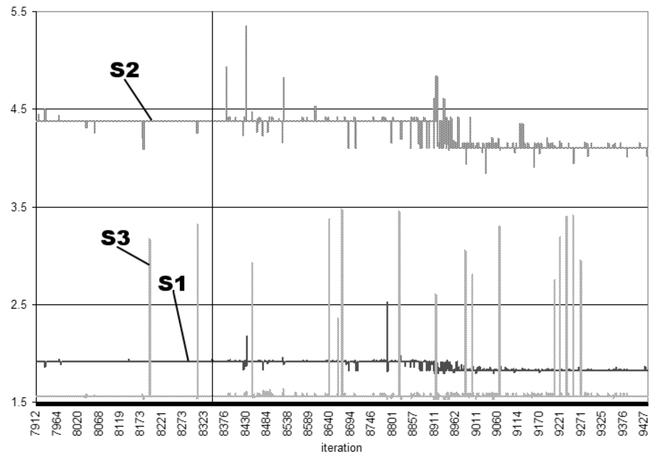


Figure 11: Costs after a new node appears

of all ants in every species, start to indicate that a better, lower cost solution exists in the changed environment. Deployment mapping of services $S1$ and $S2$ is adapted to the changes giving a somewhat lower cots value, while the cost level for $S3$ does not change with an additional node.

Moreover, in Figure 12, the number of pheromone entries in nodes $n_1 \dots n_6$ corresponding to $S1$ can be seen over time. As the problem space is explored initially the database size tops somewhat below 2^6 as $E^* = 6$ for $S1$ (cf. Section 4). After exploration the swarm quickly converges to a low cost solution so there is little variation in the pheromones. The additional node is inserted at the iteration indicated by a vertical line, before that node n_6 has an empty pheromone table. Explorer ants discover the new node quickly, thus new entries appear in the database pointing to mappings with a lower cost. After a short period of fluctuation caused by the re-reservation of processing power by the 3 species the database contains a few pheromone entries until convergence is completed and only the optimal mapping remains.

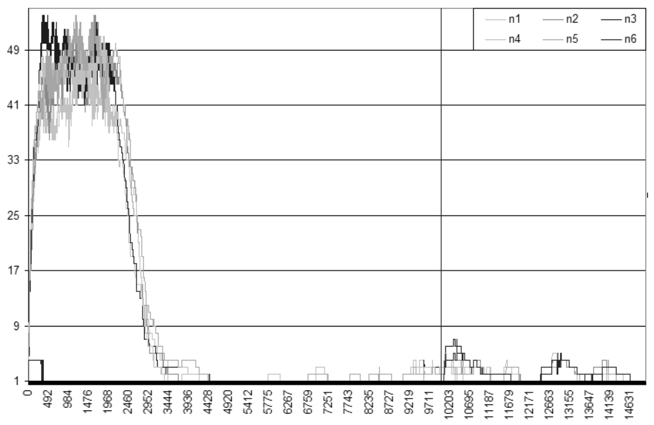


Figure 12: Database size for $S1$, with a 6th node inserted

7. Conclusions

We presented a new swarm intelligence logic for the efficient deployment mapping of software components to execution resources. A model-driven approach was presented that drives optimization of the component mapping using non-functional requirements incorporated into the model during the modelling phase. The deployment logic is executed in a fully distributed manner, thus it is free of deficiencies of most of the existing centralized approaches, such as performance bottlenecks and single points of failure. The presence of a central database or decision entity is not required to run the logic, instead we use the analogy of pheromones distributed across the network of execution hosts to store information. All the intelligence is carried along by ant-like agents.

Besides, we have showed that using CEAS our deployment logic is capable of handling multiple services simultaneously and does not require global knowledge to achieve better load-balancing among the nodes, while striving to minimize remote communication at the same time. Our goal is to develop support for run-time redeployment of components to keep the services within an allowed region of parameters defined by the requirements. With methods in CEAS an the development cycle SPACE we target a robust and adaptive service execution platform. Furthermore, we intend to address scalability issues and consider larger network domains within the deployment problem.

Considering convergence time we have a trade-off between convergence speed and solution quality. Nevertheless, while deploying services in a dynamic environment pre-mature solutions satisfying both functional and non-functional requirements often suffice. More importantly, ACO systems have been proven to be able to find the optimum at least once with probability close to one and as this happens convergence to the optimum is secured in a finite number of iterations. The optimal deployment mapping can be obtained with high confidence since CEAS can be considered as a subclass of ACO algorithms. Another advantage of our approach is the capability to provide alternative solutions weighted by their cost values, which can be selected for deployment easily as the corresponding pheromones indicate their proposed mapping. Currently, the deployment logic is implemented in a simulator written in the Simula/DEMOS language [Bir03] for evaluation purposes.

Our work is conducted in cooperation with the ISIS (Infrastructure for Integrated Services) project funded by the Research Council of Norway. The algorithm and approach presented are in-line with the objectives of ISIS that are to create an established service engineering platform for collaboration-oriented models, covering the development cycle from the requirements to seamless execution in a heterogenous and dynamic environment.

Future work on the topic will investigate inclusion of a wider range of QoS requirements and develop necessary improvements on the cost functions. It is an interesting topic to look into how larger networks of nodes influence scalability and convergence times. Similarly, introduction of a new type of species corresponding to user demands towards services targeting better resource utilization possesses challenges. Besides, we plan to experiment with distributed optimization methods other than the CE method guiding the ant-based deployment logic and also to do more extent comparison between state-of-the-art optimization methods and our work.

References

[Bir03] G. Birtwistle. *Demos - a system for Discrete Event Modelling on Simula.* Springer-Verlag New York, Inc., 2003.

[CD98] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, 1998.

[CDG05] G. Di Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Trans. on Telecomm. (ETT) - Special Issue on Self Organization in Mobile Networking*, 16(5), 2005.

[CEM03] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. on Software Engineering*, 29(10):929–945, 2003.

[CHH08] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), Oslo*, LNCS 5053, pages 253–268. IFIP, June 2008.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[H^+04] P. E. Heegaard et al. Distributed asynchronous algorithm for cross-entropy-based combinatorial optimization. In *Rare Event Simulation and Combinatorial Optimization, Budapest*, 2004.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HK07] P. Herrmann and F. A. Kraemer. Design of trusted systems with reusable collaboration models. In *Proc. of the Joint IFIP iTrust and PST Conferences on Privacy, Trust Management and Security, Moncton*, 2007.

[HW01] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[HW06] P. E. Heegaard and O. Wittner. Self-tuned refresh rate in a swarm intelligence path management system. In *Proc. of the EuroNGI Int'l. Workshop on Self-Organizing Systems, LNCS 4124*, 2006.

[JJH+08] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2008.

[KH06] F. A. Kraemer and P. Herrmann. Service specification by composition of collaborations - an example. In *Proc. of the 2006 Int'l Conf. on Web Intelligence and Intelligent Agent Technology, Hong Kong*. IEEE/WIC/ACM, 2006.

[KH07] F. A. Kraemer and P. Herrmann. Transforming collaborative service specifications into efficiently executable state machines. *Electronic Communications of the EASST*, 6, 2007.

[KHB06] F. A. Kraemer, P. Herrmann, and R. Bræk. Aligning uml 2.0 state machines and temporal logic for the efficient execution of services. In *Proc. of the 8th Int'l Symp. on Distributed Objects and Applications (DOA), LNCS 4276, Montpellier*, 2006.

[KSH07] F. A. Kraemer, V. Slåtten, and P. Herrmann. Engineering support for uml activities by automated model-checking - an example. In *Proc. of the 4th Int'l Workshop on Rapid Integration of Software Engineering Techniques (RISE), University of Luxembourg*, 2007.

[KWH08] V. Kjeldsen, O. Wittner, and P. E. Heegaard. Distributed and scalable path management by a system of cooperating ants. In *Proc. of the Int'l. Conf. on Communications in Computing (CIC)*, 2008.

[LSO+07] S. A. Lundesgaard, A. Solberg, J. Oldevik, R. France, J. Ø. Aagedal, and F. Eliassen. Construction and execution of adaptable applications using an aspect-oriented and model driven approach. In *Proc. of DAIS, LNCS 4531*, pages 76–89. IFIP, 2007.

[Mal06] S. Malek. A user-centric framework for improving a distributed software system's deployment architecture. In *Proc. of the doctoral track at the 14th ACM SIGSOFT Symp. on Foundation of Software Engineering, Portland*, 2006.

[MD07] D. Menasce and V. Dubey. Utility-based QoS brokering in service oriented architectures. In *Proc. of the Int'l Conf. on Web Services (ICWS), Salt Lake City, Utah*, July 2007.

[Mel06] H. Meling. *Adaptive Middleware Support and Autonomous Fault Treatment: Architectural Design, Prototyping and Experimental Evaluation*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2006.

[REF+08] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E. Stav. Composing components and services using a planning-based adaptation middleware. In *Proc. of SC, LNCS4954*, pages 52–67. Springer-Verlag, 2008.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[S+97] R. Schoonderwoerd et al. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2), 1997.

[Wit03] O. Wittner. *Emergent Behavior Based Implements for Distributed Network Management*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2003.

[XZF+07] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2007.

PAPER C

Component Deployment Using Parallel Ant-nests

Máté J. Csorba and Poul E. Heegaard and Peter Herrmann

To appear in International Journal on Autonomous and Adaptive Communications Systems (IJAACS)

ISSN (Online): 1754-8640. ISSN (Print): 1754-8632.

InderScience publishers

COMPONENT DEPLOYMENT USING PARALLEL ANT-NESTS

Máté J. Csorba, Poul E. Heegaard, Peter Herrmann
Department of Telematics,
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard, Peter.Herrmann}@item.ntnu.no

Abstract Our paper targets the problem of efficient software component deployment in distributed networked services. We approach the problem from a model-based service development aspect and propose a solution based on novel methods applied in network management. The swarm intelligence based deployment logic we have developed uses service models that are defined in UML 2.0. These service models are embellished to contain the non-functional requirements against the implementation of the service, which are necessary to solve the component deployment problem and to obtain a deployment mapping. The result of the heuristic optimization procedure are mappings between components and the resources provided by the execution environment the service is deployed into. In this work the deployment is viewed from the perspective of the service provider and our algorithm possesses significant potential in providing adaptation support for services in various changing environments.

1. Introduction

Realizing distributed service systems requires important design decisions that will affect both the performance of the implementation and possibly the quality of service (QoS) perceived by the user of the service. In the process of developing distributed services we start from a platform independent model and realize the service following a step-wise refinement approach. A significant factor that influences the QoS perceived is the deployment of the particular software instances, in other words configuration of building-blocks of the service and their mapping to run-time processing elements and resources available during execution.

Besides, the environment of a distributed service might be changing in an unpredictable manner, e.g. hosts can join and leave the network anytime. Heterogeneous and swiftly reconfigurable hardware introduces additional dynamism that has to be dealt with by an adaptable service. This dynamics together with the mobility and cardinality of users provide a significant impact on the QoS. Increasing the provided QoS will in turn inevitably increase the costs on the providers' side. Thus, we have to consider adaptability issues carefully in the engineering process addressing the non-functional requirements of a service. In particular, diligent deployment planning and support are necessary.

The great deal of factors influencing the optimal mapping of service components makes the deployment problem a multi-faceted challenge and demands multi-dimensional optimization techniques. In order to deal with the deployment problem we unify novel system development methods and network management solutions. In particular, we use the model-based service engineering technique SPACE as a starting point for the optimization. Here, the program code of a service is not created directly but derived from abstract system models. One model describes the functionality of the service while non-functional aspects like security, dependability or performance issues are specified in more detailed models. In the functional model, the system topology is not expressed by physical components but by abstract roles which collaborate with each other to perform certain subservices. The role identifiers and the collaboration descriptions can be embellished with QoS parameters expressing the performance demands of the specified system. When deploying the service, the QoS parameters can then be used to decide about the assignment of the roles to the available physical devices. Further, the approach enables us to provide common solutions, which are not restricted to any particular middleware platform.

We consider distributed execution of the deployment logic an important design criteria to avoid the deficiencies of existing centralized algorithms, such as performance bottlenecks and single points of failure. Besides, we aim to save resources by omitting the burden of keeping a centralized decision logic updated and synchronized constantly, this way achieving faster reaction times, which is desirable for context-awareness. By supporting run-time component (re-)deployment our goal is to allow execution of services within the allowed region of non-functional parameters defined by the service requirements. Taking into account the aspects introduced we apply principles from a distributed, robust and adaptive routing system called the Cross Entropy Ant System (CEAS) [HW01], [HHW08].

In [CHH08b] we described how to take into account QoS requirements that are captured while a service model is built and how to incorporate them into high-level goals that can be used for obtaining efficient deployment mapping of software components. Our distributed approach was tested on a static topology and compared to centralized approaches. We apply swarm intelligence to provide dynamic deployment and adaptation support for multiple services executed simultaneously in [CHH08a]. In this paper we present an elaborated version of our deployment support algorithm and show some examples and experiments with them.

The remainder of this paper is organized as follows. Next we present how our deployment logic fits into the service development cycle we target. An introduction to CEAS follows in Section 3. In Section 4 our algorithm and the construction of the main cost function that drives the logic are presented. Section 5 introduces example scenarios of three concurrent services. Some results related to the examples are evaluated in Section 6. In Section 7 we discuss work related to ours and highlight some of the advantages our approach has. Finally, in Section 8 we conclude and touch upon future work.

2. Deployment cycle

The deployment approach we are proposing is an extension to the model-based engineering method SPACE which is devoted to the rapid and correct engineering of distributed services [KH06]. In contrast to other model-based engineering techniques, SPACE uses collaboration-oriented models. Such a model does not describe single physical components but a distributed sub-functionality which is realized by the cooperation of two or more components. The overall service behavior is provided by the interaction of the sub-functionalities such that the service model is the composition of the corresponding collaboration-oriented models. An advantage of this specification style is that it facilitates service design by the reuse of model building blocks from collections of domain specific model libraries to a significantly higher degree than it would be possible with component-based descriptions [HK07]. The reason for this is that distributed sub-functionalities (e.g., automated creation and transfer of SMS, user authentication mechanisms) can often be used in various services while single components tend to have quite specific layouts for each particular application.

The stepwise modelling and refinement approach of SPACE is depicted by the inner cycle in Figure 1. The service models are expressed by UML collaboration and activity diagrams. They are mostly specified by simply selecting and instantiating model building blocks from a library and by describing the interaction between the models. Afterwards, the system description undergoes various correctness checks, as described in [KSH07], before it is automatically transformed to a component-oriented design model specified by a UML state machine for each physical component. Automatic code generators are used to create executable Java code from the state machines such that the collaboration-oriented service models can be fully-automatically transformed to executable implementations [KHB06].

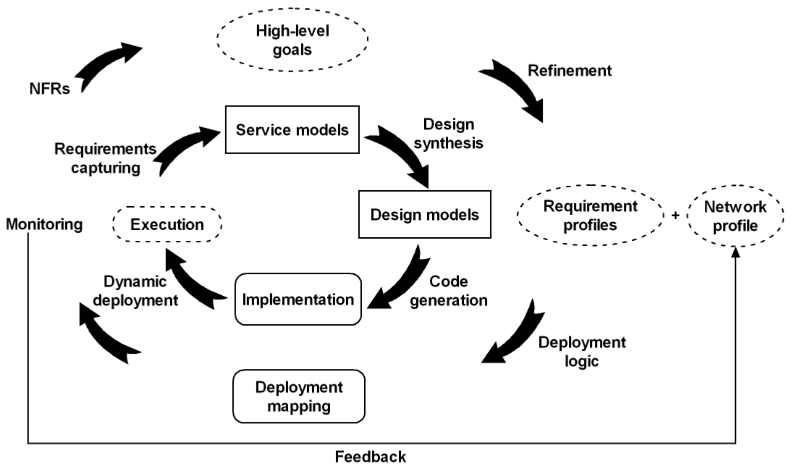


Figure 1: Development and deployment with SPACE

The dynamic deployment of the generated implementation is the step where the logic we propose can interact with the development cycle. The additional steps to support efficient deployment of the components realizing the service are shown in the outer cycle in Figure 1. The collaboration-based service models are amended by *high-level non-functional (NF) goals* defining non-functional requirements (NFRs) of a service in a rather abstract manner. Typical goals expressed by NFRs are security, performance, availability and portability. Refinement of NF goals can be done in parallel with the transformation of the service models to component-oriented design models. *Requirement profiles* obtained in this step specify NFRs of the service components. In addition, a *resource profile* is added representing properties which describe the target environment, the service will be executed in.

Link related constraints represent the cost of using the link connecting two interacting components. Constraints assigned to nodes or other resources related to execution of a component can, for instance, represent memory size limitations. These constraints can be interrelated in a way that, for example, placement of a component on a node can lower the available amount of different types of resources at once with an amount depending on the available resources at the time of the mapping.

Our deployment logic will be launched using both the requirement and the resource profiles as input with requirements specifying the search goals and the *resource profile* specifying the search space. The QoS requirements of the service model are captured in a collaboration-oriented style at design time, instead of setting up traditional service level agreements. Our view is that the deployment logic proposed will be able to handle any non-functional property of the service, as long as a suitable cost function is provided for the specific properties at hand. This feature will be provided by exploiting the advanced scalability of CEAS and the method of pheromone sharing as explained later in this paper.

Figure 2 depicts a simple example of a collaboration between two components. This collaboration is enriched with NFRs for both the components and for the collaboration binding them and thus it is part of the *requirement profile*. This basic collection of requirements contains two types of cost values, execution costs (f_{c_i}) and communication costs (f_{k_j}). The total number of cost values is equivalent to the total number E of components in the service, plus the number K of collaborations between them (i.e., $i \in \{1 \ldots E\}$ for f_{c_i} and $j \in \{1 \ldots K\}$ for f_{k_j}). The execution cost is a local cost imposed on the host node or resource executing the particular component after deployment, whereas the communication cost loads the communication link between the two components involved in the collaboration. As of now, we use constant costs in the model, thus their values are not dependent on the current utilization or other properties of the underlying hardware, which is described in the *resource profile*. Furthermore, we benefit from using collaborations as design elements as they incorporate local behavior of all participants and all interactions between them. That is, a single cost value can describe communication between component instances, without having to care about the number of messages sent, individual message sizes, etc. Besides, we believe that NFRs other than execution costs and communication costs can also be added similarly to our models as long as proper scaling is applied.

The collaboration-oriented specification and requirement capturing will be illustrated in the examples in Section 5.

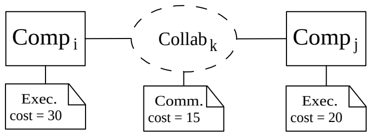


Figure 2: Collaboration with NFRs

Most existing deployment strategies and various approaches to aid deployment of software systems like ontology-based and reasoning engines, apply centralized decision logics based on centrally maintained databases, e.g. [Mal06], [REF$^+$08]. Disadvantages of this proceeding are the burden to keep a central database constantly updated and synchronized as well as the single point of failure introduced to the system. Accommodating the decision logic together with the central database on a single node may introduce both communication-wise and storage-wise bottlenecks.

In contrast to centralized approaches, a distributed cooperative algorithm employs (semi-)autonomous agents, which cooperate to achieve certain common goals. To avoid the need for any type of global knowledge in deployment mapping, we employ autonomous agents operating in a distributed environment with their decisions based solely on information that is available locally to the place where they reside. At every node under the provision of the deployment logic some sort of local information shared by several passing agents is required that will be the vehicle for cooperation between the agents. Accordingly, in our logic the information required for optimization is distributed across all participating nodes. This property of the deployment mapping system contributes to robustness, scalability and fault tolerance. Furthermore, we intend to use the same logic first to obtain an initial, optimal mapping of service components to hosts or resources, and second to guide necessary changes during execution of a service to satisfy the requirements it was launched with.

The objective of each agent is to find either the optimal deployment mapping of component instances $c_i \in \mathbf{C}$ onto nodes $n_j \in \mathbf{N}$ or at least to find a mapping that satisfies the requirements within reasonable time. It is important to note that the actual placement of components constituting a service does not change with the mappings found in each iteration. Instead placement is done on a significantly larger timescale compared to a single iteration, as it is triggered only when an ant species converged to a given mapping. An actual re-deployment of a service involves migrating components, which in turn incurs additional cost. Migration costs can be taken into account as a threshold that prohibits re-deployment if the benefit from the new component mapping is not high enough. In our current work however we do not take into account component migration costs.

The basis of the heuristics, guiding the agents to find an optimized mapping, is the cost function $F(\mathbf{M})$ that is used to evaluate the current suggestion in several iterations. Often, however, the components cannot be freely assigned to nodes but due to certain system constraints are restricted to particular ones. This can be based on policies given by the service provider (e.g. service level agreements of ISPs). Limitations can be further based on access restrictions as well as on the provided and requested capabilities (soft costs) and on capacity requirements (hard costs, e.g. bandwidth limitations). The cost function $F(\mathbf{M})$ has to consider these limitations which on the other hand restrict the search space and, in consequence, support the performance of our deployment algorithm.

3. Cross Entropy Ant System

The Cross Entropy Ant System (CEAS) as introduced by Helvik and Wittner [HW01] is an Ant Colony Optimization (ACO) system, which was originally introduced by Dorigo et al. [DMC96]. In other words, it is an agent-based optimization system, where the agents' behavior are inspired by the foraging behavior of ants. ACO has been successfully applied in a variety of communication systems applications, see for example [S$^+$97], [CD98], and [CDG05].

The key idea is to let many agents, denoted *ants*, iteratively search for the best solution according to the problem constraints and cost function defined. Each iteration consists of two phases; the *forward* ants search for a solution, which resembles the ants searching for food, and the *backward* ants that evaluate the solution and leave markings, denoted *pheromones*, that are in proportion to the quality of the solution. These pheromones are distributed at different locations in the search space and can be used by forward ants in their search for good solutions; therefore, the best solution will be approached gradually. To avoid getting stuck in premature and sub-optimal solutions, some of the forward ants will explore the state space freely ignoring the pheromone values.

The main difference between various ant-based systems is the approach taken to evaluate the solution and update the pheromones. For example, AntNet [CD98] uses reinforcement learning while CEAS uses the *Cross Entropy (CE) method for stochastic optimization* introduced by Rubinstein [Rub99]. The CE method is applied in the pheromone updating process by gradually changing the probability matrix $\mathbf{p}_r$ according to the cost of the paths. The objective is to minimize the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. For a tutorial on the method, [Rub99] is recommended.

The CEAS has demonstrated its applicability through a variety of studies of different path management strategies [HHW08], such as shared backup path protection, p-cycles, adaptive paths with stochastic routing, and resource search under QoS constraints. Implementation issues and trade-offs, such as management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism [H$^+$04] and self-tuned packet rate control [HW06]. Additional reduction in the overhead is accomplished by pheromone sharing [KWH08] where ants with overlapping requirements cooperate in finding solutions by (partly) sharing information.

In this paper, the CEAS is applied to obtain the best deployment mapping $\mathbf{M} : \mathbf{C} \to \mathbf{N}$ of a set of components, $\mathbf{C}$, onto a set of nodes, $\mathbf{N}$. The nodes are physically connected by links used by the ants to move from node to node in search for available capacities. A given deployment at iteration r is a set $\mathbf{M}_r = \{\mathbf{m}_{n,r}\}_{n \in \mathrm{N}}$, where $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ is the set of components at node n at iteration r. In CEAS applied for routing the path is defined as a set of nodes from the source to the destination, while now we define the path as the deployment set $\mathbf{M}_r$. The cost of a deployment set is denoted $F(\mathbf{M}_r)$. Furthermore, in the original CEAS we assign the pheromone values $\tau_{mn,r}$ to interface i of node j at iteration r, while now we assign $\tau_{mn,r}$ to the component set m deployed at node n at iteration r. In Section 4 we describe the search and update algorithm in details.

In CEAS applied for routing and network management, selection of the next hop is based on the *random proportional rule* presented below. In our case however, the *random proportional rule* is applied for deployment mapping. Accordingly, during the initial exploration phase, the ants randomly select the next *set of components* with uniform probability $1/E$, where E is the number of components to be deployed, i.e. the size of $\mathbf{C}$, while in the normal phase the next set is selected according to the *random proportional rule* matrix $\mathbf{p}_r = \{p_{mn,r}\}$, where

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in \mathbf{M}_{n,r}} \tau_{ln,r}} \tag{1}$$

A parameter γ_r denoted the *temperature*, controls the update of the pheromone values and is chosen to minimize the performance function

$$H(F(\mathbf{M}_r), \gamma_r) = e^{-F(\mathbf{M}_r)/\gamma_r} \tag{2}$$

which is applied to all r samples and the expected overall performance satisfies

$$h(p_{mn,r}, \gamma_r) = E_{\mathbf{p}_{r-1}}(H(F(\mathbf{M}_r), \gamma_r)) \geq \rho \tag{3}$$

$E_{\mathbf{p}_{r-1}}(X)$ is the expected value of X s.t. the rules in $\mathbf{p}_{r-1}$, and ρ is a parameter (denoted search focus) close to 0 (typically 0.05 or less). Finally, a new updated set of rules, $\mathbf{p}_r$, is determined by minimizing the cross entropy between $\mathbf{p}_{r-1}$ and $\mathbf{p}_r$ with respect to γ_r and $H(F(\mathbf{M}_r), \gamma_r)$.

To avoid centralized control and synchronized batch oriented iterations, in CEAS the cost value $F(\mathbf{M}_r)$ is calculated *immediately* after each sample, i.e., when all components are mapped, and an auto-regressive performance function, $h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1-\beta)H(F(\mathbf{M}_r), \gamma_r)$ is applied approximated by

$$h_r(\gamma_r) \approx \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(\mathbf{M}_r), \gamma_r) \tag{4}$$

where $\beta \in <0, 1>$ is a memory factor weighting (geometrically) the output of the performance function.

As for the CE method, the temperature γ_r is determined by minimizing it subject to $h(\gamma) \geq \rho$. In [HW01] it is shown that the temperature equals

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(\mathbf{M}_i), \gamma) = \rho\} \tag{5}$$

However Equation (5) is a complicated (transcendental) function that is both storage and processing intensive since all observations up to the current path sample, i.e. the entire path cost history $\{F(\mathbf{M}_1),\cdots,F(\mathbf{M}_r)\}$ must be stored, and weights for all observations have to be recalculated. In an on-line operation of a network node, such resource requirements are impractical. Instead it is assumed, given a β close to 1, that the changes in γ_r are typically small from one iteration to the next. This enables a first order Taylor expansion of Equation (5) providing:

$$\gamma_r = \frac{b_{r-1}+F(\mathbf{M}_r)e^{-F(\mathbf{M}_r)/\gamma_{r-1}}}{(1+\frac{F(\mathbf{M}_r)}{\gamma_{r-1}})e^{-F(\mathbf{M}_r)/\gamma_{r-1}}+a_{r-1}-\rho\frac{1-\beta r}{1-\beta}} \tag{6}$$

where $a_0 = b_0 = 0$ and $\gamma_0 = -F(\mathbf{M}_0)/\ln\rho$, and

$$\begin{aligned} a_r &\leftarrow \beta(a_{r-1} + (1+\tfrac{F(\mathbf{M}_r)}{\gamma_r})e^{-\frac{F(\mathbf{M}_r)}{\gamma_r}}) \\ b_r &\leftarrow \beta(b_{r-1} + F(\mathbf{M}_r)e^{-\frac{F(\mathbf{M}_r)}{\gamma_r}}) \end{aligned} \tag{7}$$

where the performance function in Equation (2) is adopted. See Appendix A in [Wit03] for a stepwise explanation of how the Taylor expansion is applied.

The pheromone values in CEAS are a function of the *entire history* of path cost values and hence CEAS has what is denoted a search history dependent quality function in [Z$^+$04]. Pheromone values, and the corresponding random proportional rules, are updated for every path sample made. This implies that updates are made by applying the performance function in Equation (2) combining the last cost value $F(\mathbf{M}_r)$ and the temperature γ_r calculated by Equation (6). Pheromones are updated by

$$\tau_{mn,r} = \sum_{k=1}^{r} I((m,n)\in\mathbf{M}_k)\beta^{\sum_{x=k+1}^{r} I((m,\cdot)\in\mathbf{M}_x)}H(F(\mathbf{M}_k),\gamma_r) \tag{8}$$

The memory factor β gives geometrically decreasing weights to the output of the performance function, which means that CEAS implements evaporation of pheromones. The exponent of β is somewhat complex since backward ants do not update all nodes in the network but only those nodes that were visited by the corresponding forward ant. The exponent in Equation (8) is the number of ants that have updated node n at “time” r since “time” k when a path $\mathbf{M}_k$ was found, while $r-k$ is the total number of updates in the system, i.e. total number of returned backward ants, at ”time” r since “time” k. Hence $r-k \geq \sum_{x=k+1}^{r} I((m,\cdot)\in\mathbf{M}_x)$.

However, as for Equation (5), excessive processing and storage requirements also apply for Equation (8). Hence a (second order) Taylor expansion of Equation (8) is appropriate, giving

$$\tau_{mn,t} \approx I((m,n)\in\mathbf{M}_r)e^{-\frac{F(\mathbf{M}_r)}{\gamma_r}} + A_{mn} + \begin{cases} -\frac{B_{mn}}{\gamma_r}+\frac{C_{mn}}{\gamma_r^2} & \frac{1}{\gamma_r}<\frac{B_{mn}}{2C_{mn}} \\ -\frac{B_{mn}^2}{4C_{mn}} & \text{otherwise} \end{cases} \tag{9}$$

where

$$\begin{aligned}
A_{mn} &\leftarrow \beta(A_{mn} + I((m,n) \in \mathbf{M}_r)e^{-\frac{F(\mathbf{M}_r)}{\gamma_r}}(1 + \tfrac{F(\mathbf{M}_r)}{\gamma_r}(1 + \tfrac{F(\mathbf{M}_r)}{2\gamma_r}))) \\
B_{mn} &\leftarrow \beta(B_{mn} + I((m,n) \in \mathbf{M}_r)e^{-\frac{F(\mathbf{M}_r)}{\gamma_r}}(F(\mathbf{M}_r) + \tfrac{F(\mathbf{M}_r)^2}{\gamma_r})) \\
C_{mn} &\leftarrow \beta(C_{mn} + I((m,n) \in \mathbf{M}_r)e^{-\frac{F(\mathbf{M}_r)}{\gamma_r}}(\tfrac{F(\mathbf{M}_r)^2}{2}))
\end{aligned} \tag{10}$$

The initial values of Equation (10) are $A_{mn} = B_{mn} = C_{mn} = 0$ for all (m,n). A stepwise explanation of the Taylor expansion is given in Appendix A in [Wit03].

4. Distributed deployment logic

The deployment logic we have developed can be considered as a swarm of independent agents with ant-like behavior that execute an optimization task in the network of possible target hosts. They proceed with their behavior continuously and target the deployment of a service based on information from the service model. Due to this continuous behavior the same logic can provide an initial mapping for the system components and can also support run-time redeployment. By allowing interoperation of artificial ant species via indirect communication between them multiple services can be deployed using our approach. We assign the deployment of one set of service components to each ant-species that way making a species responsible for the deployment mapping of a single service realized by distributed software components.

By applying an appropriate cost function that is used for evaluating the solutions found during the heuristical optimization process we omit the need for any global knowledge within the species. In other words, a set of species can be launched and can operate independently without any of them requiring a global view on the total set of requirements for all services being deployed in the system at a time. Another important property supporting scalability is that the ants are limited to visit only those nodes in the network that are effectively used in the deployment of service components. This property allows to address significantly larger problem sizes that consist of a high number of nodes and a large number of services executed simultaneously.

After an ant is created it is assigned the task of deployment of a set of components, **C**, that constitute the service represented by the species the ant is originated from. After initialization at the ant nest, the node where all the ants from a species are originated, the ant samples the current temperature. Next, the ant selects the first hop in the network where it starts a random-walk. The layout of the network is described by the *net-map*. Every next hop is selected randomly, and after its arrival to a node the ants behavior depends on if it is a *explorer* or a *normal* ant. *Normal* ants inspect the local instance of the distributed pheromone database at the node and selects a subset, $\mathbf{m}_{n,r}$, of **C** for mapping, which selection is stored in a list. The mapping list, $\mathbf{M}_r$ is carried along by the ant during its search. An *explorer* ant in turn selects a subset $\mathbf{m}_{n,r}$ randomly, without using the pheromone database available at the node. It is important to note that the ratio of *explorer* ants can be regulated to an arbitrary extent. The purpose of this ant-type is twofold, initial exploration of the *net-map* and discovery of fluctuations in the network during the continuous optimization process, e.g. new nodes appearing. Random exploration can be considered as sampling from the problem space and results in a random cost figure. During the initial deployment mapping phase a number of *exploration* iterations are used. The length of this period depends on the problem size, the end of this period however can be detected by monitoring the pheromone database size as it extends while a growing number of possible solutions are covered up by the ants doing a random-walk in the problem space. During the *normal* phase only a fraction of the ants, e.g. 5-10%, are flagged as *explorers*. Nevertheless, continuously emitted *explorers* are needed for enhanced responsiveness and detection of changes in the network environment, while *normal* ants focus on finding the optimal deployment mapping.

As we target the deployment of multiple services using the same mechanisms, species have to have a means of interoperation to obtain the optimal mapping for the components taking into account resource usage of the services being deployed simultaneously. We consider the optimal deployment of services from the perspective of the provider, thus our logic aims at balancing the execution costs that are imposed on the nodes used for execution of the services, or in other words load-balancing, which generally requires a global overview on the system's operating conditions. However, we want to avoid having any centralized structure in our algorithm and use a distributed optimization method instead. This requirement lead us to introduce a mechanism through which species can reserve processing power at arbitrary nodes and that will serve as a means of indirect communication between different species. This reservation mechanism needs to be available at every participating node in addition to the capability of storing elements of the pheromone database. Ant species will indicate their latest requested resource usage, corresponding to the execution costs in the service model they are responsible for, by allocation at the visited nodes at every iteration r. Sampling the current set of allocations in every visited node can give a general overview for the ants, thus load-levels in participating nodes can be incorporated into the cost calculations at the end of each iteration. We refer to load-level samples taken during an ant run with the set $\mathbf{NL}_r$. The actual implementation of load-level sampling is left to the middleware. More important is the fact that, allocation entries that are outdated are invalidated to preserve consistency. Invalidation of outdated allocations can easily be done by the ants before they allocate a new value at the nodes they visit. This means, however, that outdated allocations of a species can still remain present in some nodes as an ant does not necessarily visit the same set of nodes in each iteration. Accordingly, an iteration identifier, i.e. r, for each species is stored together with the allocations in each node. During cost evaluation, when load-levels for each node are calculated, an ant takes into account only those allocation samples that have the highest r identifier for each service.

At first an ant conducts a forward search until it manages to find a mapping to all the components it is assigned with. This resulting mapping is stored in the set $\mathbf{M}_r$ by the ant and will be evaluated by the cost function of the service. The key point in the optimization process is the formulation of the cost function, $F()$, which takes into account the NFRs that the service model is extended with. Our cost function uses two parameters updated in every iteration, the component mapping set $\mathbf{M}_r$ and the load-level samples taken during forward search $\mathbf{NL}_r$. Moreover, the cost calculation is based on the execution and communication costs derived from the service model as introduced in Section 2. Correspondingly, the cost function is built up by two components, node related costs (**NC**) and link, i.e. collaboration related costs (**LC**). The aim of each species is to minimize the overall value of its own function F, Equation (11).

$$F(\mathbf{M}_r, \mathbf{NL}_r) = [\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NC}(n_j)] \cdot (1 + x \cdot \mathbf{LC}) \tag{11}$$

where **x** is a parameter. Equation (11) is used by all species the same way, and has a component strictly local to the species, **LC**, which incorporates the collaboration costs

$$\mathbf{LC} = \sum_{j=1}^{K} I_j \, f_{k_j} \tag{12}$$

where I_j is an indicator function applied to take into account only those communication costs of the collaborations that happen between different nodes, i.e. remotely

$$I_j = \begin{cases} 1, & \text{if } Collab_k \; is \text{ external} \\ 0, & \text{if } Collab_k \; is \text{ internal to a node} \end{cases} \tag{13}$$

Node related costs are incorporated into the component $\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NC}(n_j)$. This section of the overall cost function aims at achieving load-balancing among the nodes providing the services being executed. More important, in this part only the subset of nodes the ant has visited is taken into account not the total amount of nodes. References to nodes visited in iteration r are contained in the hop-list, $\mathbf{H}_r$. This node related cost is calculated individually for every visited node according to

$$\mathbf{NC}(n_j) = [\sum_{i=0}^{\mathbf{NL}_{n,r}(n_j)} \frac{1}{\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NL}_{n,r} + 1 - i}]^y \tag{14}$$

Execution costs from one ant's perspective for node n_j are calculated according to Equation (14) based on the load-levels sampled. The cost component **NC** tries to counteract component **LC**. On one hand, **LC** tries to put weight on component mappings that have as much as possible of the collaborations within the same node(s) by favoring mappings that use less nodes for deployment with a low cost value. This way minimizing external communication. On the other hand, Equation (14) has an effect of distributing components, thus equalizing execution load among the available hosts to the highest extent possible. This way two counteracting requirement types are tackled in the same cost function. The exponent **y** in Equation (14) can be tuned to focus more on load-balancing instead of the collaboration costs by selecting a larger value. The multiplier **x** in Equation (11) in turn can be used to scale collaboration costs if needed. The values we generally used in the example scenario presented in Section 5 were $\mathbf{x} = 0.1$ and $\mathbf{y} = 2$.

At the end of each iteration the mapping found by the ant is evaluated using Equation (11). Likewise, the temperature is recalculated according to Equation (6). After the forward search an ant travels back to its nest along the path stored in the hop-list $\mathbf{H}_r$ during the phase called *backtracking*. On its way back to its nest the ant updates the pheromone values according to Equation (9). Arriving back to its nest node the ant updates the temperature with the new, recalculated value that will be used by the next ant in the subsequent iteration. Passing the temperature value to the next iteration at the nest is a necessary step as in contrast to routing and network path management this information cannot be stored at a particular destination node making it available to subsequent ants, because no designated destination node exists during deployment mapping. In other words, a given deployment mapping, $\mathbf{M}: \mathbf{C} \rightarrow \mathbf{N}$ can stretch over a set of nodes $\mathbf{N}$, thus the same mapping $\mathbf{M}$ can be found by the ants visiting the same set of nodes not necessarily in the same order, but making the same mapping decisions. After the *backtracking* phase the ant's behavior is ended, a new ant will be initiated and emitted unless a stopping criteria is met.

There are different options for constructing a stopping criteria. For example, the moving average of the evolving cost value can be observed and this way convergence to a suggested solution can be detected. Another option is sampling the size of the distributed pheromone database during an iteration, thus convergence can be detected by observing a very strong pheromone value that emerges in the database, while inferior solutions evaporate. All the ant species, i.e. representing arbitrary simultaneous services, will execute the described process summarized in Algorithm 1.

The distributed pheromone database that guides the ants' decisions has to be aligned to the sets of deployed components. During an iteration each ant visits $\mathbf{n} \subseteq \mathbf{N}$ nodes and will form n discrete sets from the available components ($\mathbf{C}$) carried along. At the end of every iteration r a deployment mapping $\mathbf{M}_r$ will be evaluated. For every species a pheromone database is built by assigning a flag to every component that is free for deployment mapping, i.e. which is not bound to a specific node by requirements. This leaves a species with a number of components $E^* \subseteq E$, correspondingly the pheromone database size becomes 2^{E^*}, equal to the number of possible combinations for a set at a node, a number specific for each service. Thus, the memory requirement for an execution platform accommodating our logic is to store 2^{E^*} floating point numbers at every node. We can also normalize the pheromone entries in a node to be between $0 \ldots 1$, in this way the entries can be viewed as a probability distribution of component sets mapped to that node by the artificial ants. Once a converged state is reached the optimal solution(s) emerge with probability one.

The pheromone database constructed as described above can be accessed and indexed using identifiers pointing at sets of components. For example, consider a basic set of 5 components constituting a service, $\mathbf{C} = \{c_1, c_2, c_3, c_4, c_5\}, E^* = E = 5$. Indexing is done using an E^* long binary bitstring, e.g. element 17 of the pheromone database, which is equivalent to $'10001'B$, refers to the deployment of a set of two components, $\{c_1, c_5\}$. In addition, the pheromone database we use can be allocated dynamically based on thresholds used for evaporating pheromone entries under a given significance level to achieve better scalability. In the examples we apply a threshold of 1%, i.e. pheromone values lower than 1% of the highest value are considered insignificant and are eliminated from the database.

Algorithm 1 Deployment mapping of service components $\mathbf{C} \in S_j$ from $Nest_k$

```
1  Read the current temperature, $\gamma_0$ from $Nest_k$.
2  Select the first node $n \in \mathbf{N}$ randomly where the search from $Nest_k$ will start, and add $n$ to the hop-list $\mathbf{H}_r = \mathbf{H}_r + \{n\}$.
3  Select a set of components $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ so that $n \in \mathbf{R}$ is satisfied for every $c_i \in \mathbf{m}_{n,r}$ according to the random proportional rule (*normal* ant), Equation (1), or in a totally random manner (*explorer* ant). If such a set cannot be found, goto step 8.
4  Update the ant's deployment mapping set, $\mathbf{M}_r = \mathbf{M}_r + \{\mathbf{m}_{n,r}\}$.
5  Update the set of components to be deployed, $\mathbf{C} = \mathbf{C} - \mathbf{m}_{n,r}$.
6  (Re-)allocate processing power at the current node, $n$ according to $f_{c_i}, \forall c_i \in \mathbf{m}_{n,r}$.
7  Sample the estimated processing load level at $n$, $\mathbf{nl}_{n,r}$, and let $\mathbf{NL}_r = \mathbf{NL}_r + \{\mathbf{nl}_{n,r}\}$.
8  Select next node, $n$ randomly and add $n$ to the hop-list $\mathbf{H}_r = \mathbf{H}_r + \{n\}$.
9  If $\mathbf{C} \neq \emptyset$ then goto 3., otherwise evaluate $F(\mathbf{M}_r, \mathbf{NL}_r)$ using the mapping set $\mathbf{M}_r$ and the samples taken ($\mathbf{NL}$).
10 Recalculate $\gamma_0$ based on the new cost value, Equation (6).
11 Update the pheromone values, Equation (9), corresponding to the $\{\mathbf{m}_{n,r}\} \in \mathbf{M}_r$ mappings going backwards along $\mathbf{H}_r$.
12 Update $\gamma_0$ at $Nest_k$.
13 If stopping criteria is not met then start new iteration (increment $r$), initialize and emit new ant and goto 1.
```

5. Design examples

In this section we present 3 service models demonstrating the modelling method. The example models will be used to evaluate the deployment logic subsequently. The first example service, $S1$ has been introduced originally in [KH06]. $S1$ has a component that operates a security door and a card reader with a keycode entry panel. These two components are bound to node n_1 by their requirements. In addition, a central component administers access rights using an authentication and an authorization server with corresponding databases implemented as separate components (Figure 3).

The second example, Figure 4 models a video surveillance system. The service has one surveillance camera component bound to each of the five nodes by default. A central control and a recording unit is used for managing the system and a main and a backup storage device for storing surveillance information in a replicated database.

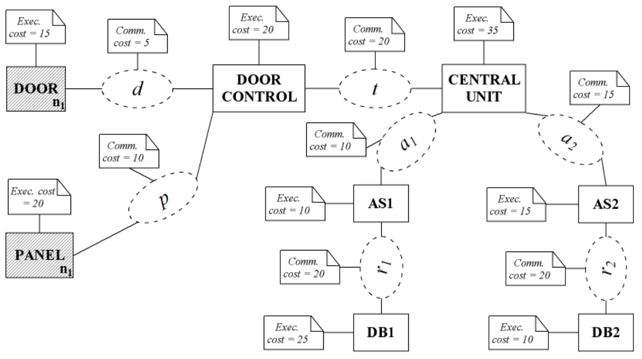


Figure 3: S1 - The Access Control System

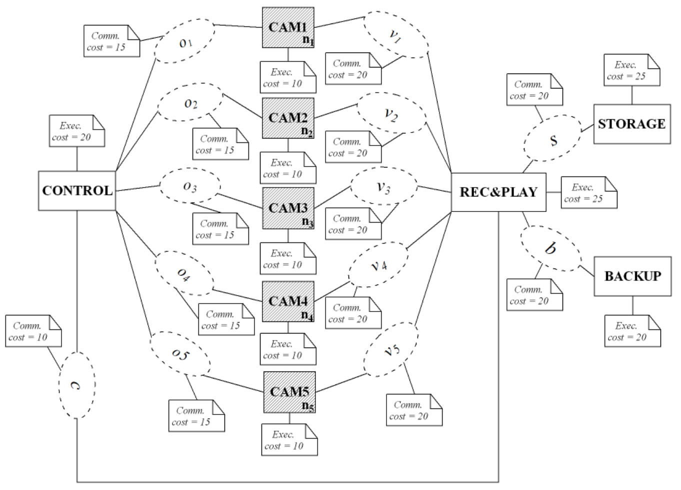


Figure 4: S2 - The Video Surveillance System

The third service is a model of a process controller that consists of 4 main stages of processing. Besides, S3 has a main generator component that produces the input impulses for the processing stages and a logging module is monitoring the output of the four stages. On top of that, a user interface component can be used for direct human interaction with the system (Figure 5). In $S3$ none of the components are bound directly, thus every component can freely be mapped to any of the nodes in N, depending the on current availability of resources.

Ant species are assigned to the services and a deployment mapping is conducted on the underlying network of hosts, which consists of 5 nodes with equivalent capabilities in the example setting. That is, the *resource profile* in this case contains 5 hosts, which are capable of executing components and are interconnected in a full mesh with equally capacious links. The exact values of execution costs (f_c) assigned to

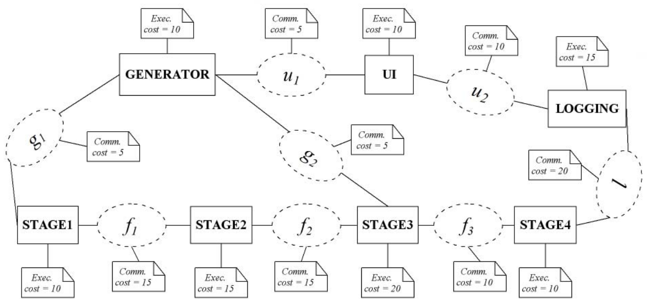
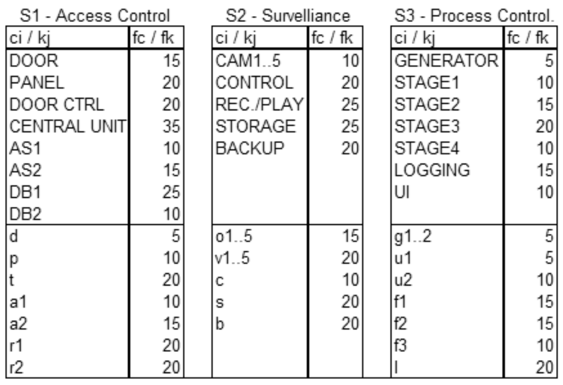


Figure 5: S3 - The Process Controller System

components (c_i) and communication costs (f_k) assigned to collaborations (k_j) in the example scenario are summarized in Table 1.

Table 1: Components and costs in the examples



S1 - Access Control

ci / kj	fc / fk
DOOR	15
PANEL	20
DOOR CTRL	20
CENTRAL UNIT	35
AS1	10
AS2	15
DB1	25
DB2	10
d	5
p	10
t	20
a1	10
a2	15
r1	20
r2	20

S2 - Surveliance

ci / kj	fc / fk
CAM1..5	10
CONTROL	20
REC./PLAY	25
STORAGE	25
BACKUP	20
o1..5	15
v1..5	20
c	10
s	20
b	20

S3 - Process Control.

ci / kj	fc / fk
GENERATOR	5
STAGE1	10
STAGE2	15
STAGE3	20
STAGE4	10
LOGGING	15
UI	10
g1..2	5
u1	5
u2	10
f1	15
f2	15
f3	10
l	20

How the swarm intelligence logic handles the example services and scenario will be evaluated in the next section.

6. Evaluating the scenario

We will use the example scenario and services introduced in Section 5 to observe some of the properties of our deployment logic. This example setting has multiple optimal and near-optimal solutions based on the wide variety of possible groupings of components on the available nodes. What makes the distributed optimization task even more challenging is that the service deployment problem using execution and communication costs can be NP-hard even if we have to deal with a single service at a time (c.f. [CHH08b]).

Based on the example setting of 3 services and 5 nodes initially the number of required iterations to obtain a solution is significantly smaller compared to the size of the problem if we would used conventional exhaustive search, as in this case, components of a each service can take $\mathbf{N}^{E^*_{Sk}}$ mapping combinations, where E^*_{Sk} is the number of unbound components in service k. If we are deploying all 3 services at the same time the size of this example becomes $\mathbf{N}^{\sum_k E^*_{Sk}}$, which would then require the exhaustive exploration of 5^{15} possible configurations.

Thus, it is important to recall that we are interested in finding mappings satisfying the requirements in reasonable time and not necessarily in always finding the optimal solution. Furthermore, we split the problem into 3 as we have one type of species for each of the services and these species are executed simultaneously. The guided heuristic search for a mapping can even more be accelerated if there is no need for supporting adaptability to events such as nodes appearing/disappearing or link fluctuations, hence an initial deployment mapping satisfying the NF-requirements is sufficient. However, we are interested in a complete solution paving the way for model-based adaptability. For demonstration a solution, i.e. a mapping of components to nodes ($\mathbf{M} : \mathbf{C} \rightarrow \mathbf{N}$) is shown in Table 2. This mapping is an example output of the logic.

Table 2: Example deployment mapping using 5 nodes

	S1	S2	S3
n1	DOOR, PANEL	CAM1	STAGE4, LOGGING
n2	AS2, DB2	CAM2, REC/PLAY, STORAGE, BACKUP	
n3	DOOR CTRL.	CAM3	STAGE1, STAGE2, STAGE3
n4	AS1, DB1	CAM4	UI
n5	CENTRAL UNIT	CAM5, CONTROL	GENERATOR

Two simple scenarios were evaluated to investigate the capability of the method to adapt to changes in the execution context of the services. First a single node failure occurs in the network, the node goes down and becomes repaired and operational some time later. In this case we experiment with *soft-errors*, meaning that deployment to the selected node will be disallowed for the species after the error event occurs, but the node will still be operational for the components that are bound to it by requirements. An additional node appears in the network in the second scenario. The new node is added after all the species have converged to a solution with 5 nodes.

The evolution of deployment mapping costs as calculated by the corresponding cost functions are shown for two of the example services, $S2$ and $S3$, in Figure 6. The costs for each of the services are calculated based on the costs f_c and f_k (cf. Table 1) of the given service and the processing costs of other service components sharing the same set of hosts, which can be predicted using the allocation samples NL_r. In the figure results of injecting a node error and a corresponding repair 4000 iterations later can be seen. The plot shows the average cost value as dots and the minimum/maximum values as error bars obtained by running 100 times the same scenario. The first 2000 iterations represent the initial *exploration* phase. As the *normal* phase starts from iteration 2001 the costs of component mappings start to decrease as species start to cooperate and better and better solutions are found.

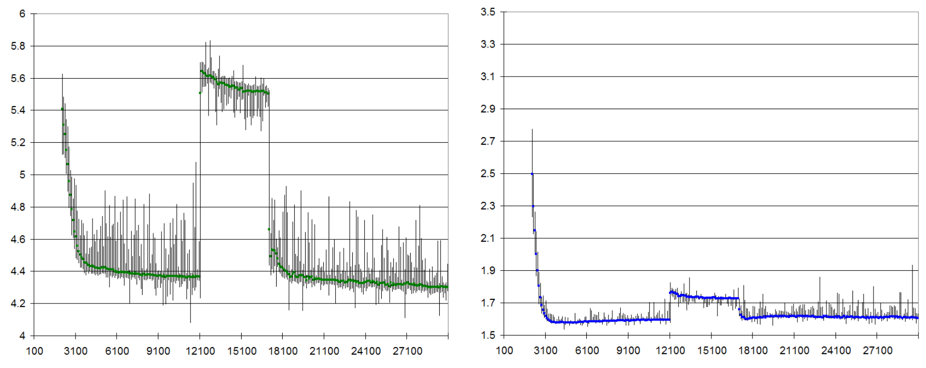


Figure 6: Costs for S2 and S3 with node error and repair

As the node error appears, node n_3 goes down, the cost values increase for all of the services, although not to the same extent. The cost levels for $S2$ are generally higher, probably as a result of a high amount of communication demand, which is also making the redeployment of components constituting the service more difficult and dependent on the resource usage of parallel services. Service $S3$ is able to find a new deployment mapping on the remaining nodes for its components that comes with an almost equally low cost as before the node error. After n_3 is repaired components can be re-mapped again to achieve lower cost levels with 5 nodes. Adaptation to the repair event in the environment happens within a few iterations.

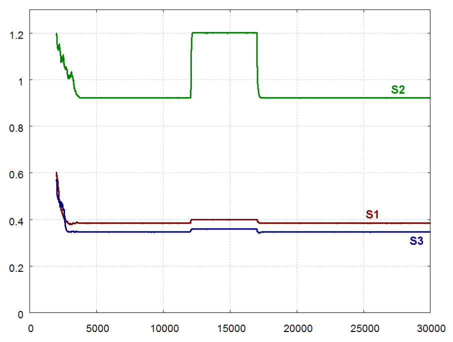
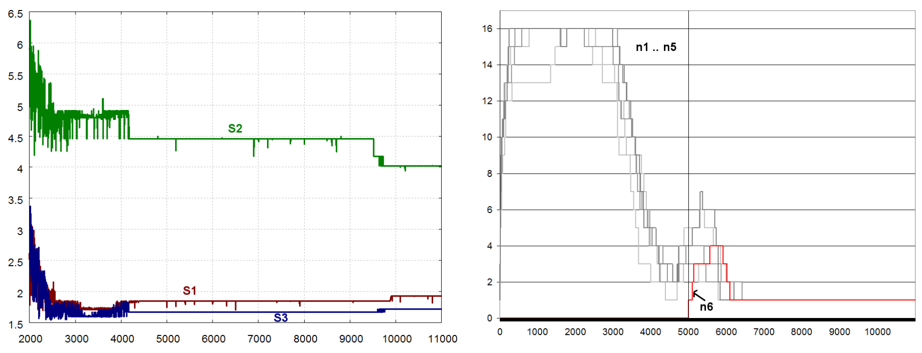


Figure 7: Temperature within the 3 species

It is also interesting to observe the control parameter γ_r, i.e. the temperature that governs the performance function during a run with error and repair events. In Figure 7 changes in the temperature and how species react to changes in the environment are shown. The node failure is detected very quickly and the temperature increases suddenly as solutions with a lower cost disappear from the solution space. The increase in temperature is most significant in $S2$, while $S1$ and $S3$ are slightly affected. After the repair the swarm reacts as ants find better solutions. More iterations later all species converge to a stabilized state with low temperatures again.

In our second test scenario we introduce a new, 6th node to the environment of the services. The species are already in a converged state when the new node appears around iteration 5000. At this point the swarm is using only 5% of the ants emitted for *exploration*. After some iterations explorers start to indicate that a better, lower cost solution exists in the changed environment. Soon the better solutions with a lower cost reach majority in the pheromone database eventually resulting in adaptation, i.e. redeployment of components. The new overall deployment mapping of the 3 services imposes slightly higher costs for $S1$ and $S3$, but a significantly lower cost for $S2$, which indicates that the swarm managed to achieve consensus for a globally better solution, even though the individual species did not have a global overview of the system themselves. The cost values for the species are depicted on the left in Figure 8.

New, valuable mappings are appearing in the distributed pheromone database as new entries. On the right in Figure 8 the number of pheromone entries corresponding to $S2$, i.e. the database size is depicted for nodes $n_1 \ldots n_6$. As $S2$ has only 4 components that are available for redeployment, $E^*_{S2} = 4$, the maximum database size is $2^{E^*_{S2}} = 16$. During initial exploration the various solutions are explored and the database size tops. After exploration the swarm quickly converges to a low cost solution so there is little variation in the pheromones. The additional node is inserted at the iteration indicated by a vertical line, before that node n_6 has an empty pheromone table. Explorer ants discover the new node quickly, thus new entries appear in the database pointing to mappings with a lower cost, and the database in n_6 is building up too. After a short period of fluctuation caused by the re-reservation of processing power by the species suboptimal solutions are evaporated from the database and after convergence an optimal mapping remains only.



(a) (b)

Figure 8: Costs and the pheromone database after a new node appears

7. Related Work

Today, a fair number of middleware approaches aiming at adaptability and dependability have become available. An autonomous replication management approach for dependability is suggested by Meling in a framework based on group communication systems [Mel06]. A dynamic middleware, QuAMobile, is presented in [LSO$^+$07], which introduces independent application variants and selects between them for context-awareness and adaptation. Service level agreements (SLAs) and QoS-aware metadata is used in the planning-based adaptation middleware of the MUSIC project (cf. [REF$^+$08]). A peer-to-peer middleware, CARISMA is utilizing an auctioning-like mechanism for conflict resolution and adaptation automatically triggered by context changes [CEM03]. In the QoS brokering approach from Menasce and Dubey consumers can request services, after which a broker uses analytic queuing models to predict QoS of the services under various workloads, and to look for maximized utility [MD07]. A framework, SmartFrog [Sab06], for describing, deploying and managing distributed service components has been developed in HP Labs. Similarly to our concept, this framework describes services as collections of components and applies a fully distributed engine comprised of daemons running on each node. The same engine can be used to deploy many services simultaneously, which is a property of our deployment logic too. A description language is used for specifying configurations, using which deployment information can be described and instantiated across a network. After instantiation components that form the application are hosted by the framework itself.

The various kinds of middleware discussed above are all very good candidates for serving as a means of instrumenting deployment guided by our logic. In our work we do not deal with implementation and support of component instantiation and migration. The algorithms we are developing serve as an optimization and decision logic triggering component placement, while the actual decisions made by the logic are autonomous and are based on the service model.

Effective component placement requires some kind of optimization approach in addition to methods easing the execution of it. Virtualized environments and server consolidation achieved significant advances recently in the autonomic computing community. Xu et al. applied fuzzy learning for configuration management in server environments via a two-level control mechanism that targeted efficient resource utilization [XZF$^+$07]. Queuing models, namely layered queuing networks are employed by Jung et al. for generating optimal configurations and policies in an offline framework [JJH$^+$08]. Others also suggest improving the perceived QoS through changing the deployment of applications, however due to the exact solution algorithms, complexity becomes NP-hard already with more than 2-3 hosts or several QoS dimensions restricting applicability of these methods. Approximative solutions that can overcome some of the scaling problems, such as greedy algorithms and genetic programming are discussed in [Mal06]. The algorithms presented therein are aiming to maximize utility of a service from the users' perspective, whereas we formulate and solve the deployment problem in a way that takes into account the providers' goals while considering user requirements as well. Moreover, our deployment logic should grant the same benefits that exist in distributed management architectures, such as increased dependability, better resource utilization, etc. while still preserving an output format that is platform independent.

8. Conclusions

In our work we discussed modelling of distributed networked services, incorporating non-functional requirements into service models for the purpose of obtaining optimized deployment of components constituting the services. A swarm intelligence logic was built for solving the optimization task of mapping software components to execution resources. The deployment logic can be executed in a fully distributed manner, making it free of the deficiencies most of the existing centralized approaches suffer from, such as performance bottlenecks or single points of failure. Instead of a central database or control and decision making entity we use the analogy of pheromones distributed across the network of execution hosts to store information. Pheromones that will be used by ant-like agents possessing all the intelligence required to solve the deployment problem. Using CEAS we have showed that our method can handle the deployment of multiple services executed in parallel without requiring knowledge available globally to all the species, i.e. all of the species rely only on the view they obtain on the execution environment and make independent decision based on that. Species assigned with the task of deploying a particular set of service components are able obtain load-balancing among the execution nodes, while striving to minimize remote communication between components at the same time. Nevertheless, the goal of optimization is not necessarily obtaining the global optimum, but instead to keep the services within an allowed region of QoS parameters defined by the requirements. With methods in CEAS and the development cycle SPACE we target a robust and adaptive service execution platform. The deployment logic is being developed in a way to address scalability issues and to consider larger domains of networks within the deployment problem.

Convergence time is often cardinal in stochastic optimization frameworks, here we have a traded-off between convergence speed and solution quality. However, during service deployment in a dynamic environment pre-mature solutions satisfying both functional and non-functional requirements often suffice. More important, ACO systems are proven to be able to find an optimal solution at least once with a probability close to one and as this happens convergence to this solution is assured within a finite number of steps. As CEAS can be considered a subclass of ACO algorithms, optimality of deployment mappings obtained with CEAS can be trusted with high confidence. In addition, our approach can provide multiple alternative solutions weighted by their cost values, which can easily be selected and used for deployment based on the corresponding pheromone values indicating the proposed mapping.

Our methodology has been developed in cooperation with the project ISIS (Infrastructure for Integrated Services) funded by the Research Council of Norway. Both the algorithm and the collaboration-oriented modelling approach are in-line with the main objectives of ISIS, i.e. to create a service engineering platform that covers the service development cycle from the requirements capturing phase to seamless execution in heterogeneous and dynamic environments. The deployment logic is continuously under development and exists currently as a preliminary implementation in a simulator environment [Bir03] for evaluation purposes.

There are many objectives and directions that will be considered to further improve our method. Different types of QoS requirements will be considered for inclusion, which will also influence developments and refinements of the cost functions we use. Larger networks and their influence on convergence and scalability will be investigated. The user perspective has not taken a role so far in the optimization process. We plan to address this aspect with constructing separate species corresponding to user demands. This aspect will however introduce new challenges as it will increase the dimensions of the deployment problem. Besides, we also plan to experiment with a variety of stochastic optimization methods for guiding the artificial ants we employ and compare our results. Another interesting topic is to use the deployment mapping logic as feedback towards the service designer for influencing the functionality in the model before implementation. Nevertheless, the next issue that will be tackled is making nest nodes that emit ants more dependable by developing strategies based on, among others, replication.

References

[Bir03] G. Birtwistle. *Demos - a system for Discrete Event Modelling on Simula.* Springer-Verlag New York, Inc., 2003.

[CD98] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, 1998.

[CDG05] G. Di Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Trans. on Telecomm. (ETT) - Special Issue on Self Organization in Mobile Networking*, 16(5), 2005.

[CEM03] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. on Software Engineering*, 29(10):929–945, 2003.

[CHH08a] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In *Proc. of the 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), Turin*. ICST/ACM, September 2008.

[CHH08b] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), Oslo*, LNCS 5053, pages 253–268. IFIP, June 2008.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[H^+04] P. E. Heegaard et al. Distributed asynchronous algorithm for cross-entropy-based combinatorial optimization. In *Rare Event Simulation and Combinatorial Optimization, Budapest*, 2004.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HK07] P. Herrmann and F. A. Kraemer. Design of trusted systems with reusable collaboration models. In *Proc. of the Joint IFIP iTrust and PST Conferences on Privacy, Trust Management and Security, Moncton*, 2007.

[HW01] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[HW06] P. E. Heegaard and O. Wittner. Self-tuned refresh rate in a swarm intelligence path management system. In *Proc. of the EuroNGI Int'l. Workshop on Self-Organizing Systems, LNCS 4124*, 2006.

[JJH$^+$08] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2008.

[KH06] F. A. Kraemer and P. Herrmann. Service specification by composition of collaborations - an example. In *Proc. of the 2006 Int'l Conf. on Web Intelligence and Intelligent Agent Technology, Hong Kong*. IEEE/WIC/ACM, 2006.

[KHB06] F. A. Kraemer, P. Herrmann, and R. Bræk. Aligning uml 2.0 state machines and temporal logic for the efficient execution of services. In *Proc. of the 8th Int'l Symp. on Distributed Objects and Applications (DOA), LNCS 4276, Montpellier*, 2006.

[KSH07] F. A. Kraemer, V. Slåtten, and P. Herrmann. Engineering support for uml activities by automated model-checking - an example. In *Proc. of the 4th Int'l Workshop on Rapid Integration of Software Engineering Techniques (RISE), University of Luxembourg*, 2007.

[KWH08] V. Kjeldsen, O. Wittner, and P. E. Heegaard. Distributed and scalable path management by a system of cooperating ants. In *Proc. of the Int'l. Conf. on Communications in Computing (CIC)*, 2008.

[LSO$^+$07] S. A. Lundesgaard, A. Solberg, J. Oldevik, R. France, J. Ø. Aagedal, and F. Eliassen. Construction and execution of adaptable applications using an aspect-oriented and model driven approach. In *Proc. of DAIS, LNCS 4531*, pages 76–89. IFIP, 2007.

[Mal06] S. Malek. A user-centric framework for improving a distributed software system's deployment architecture. In *Proc. of the doctoral track at the 14th ACM SIGSOFT Symp. on Foundation of Software Engineering, Portland*, 2006.

[MD07] D. Menasce and V. Dubey. Utility-based QoS brokering in service oriented architectures. In *Proc. of the Int'l Conf. on Web Services (ICWS), Salt Lake City, Utah*, July 2007.

[Mel06] H. Meling. *Adaptive Middleware Support and Autonomous Fault Treatment: Architectural Design, Prototyping and Experimental Evaluation*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2006.

[REF$^+$08] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E. Stav. Composing components and services using a planning-based adaptation middleware. In *Proc. of SC, LNCS4954*, pages 52–67. Springer-Verlag, 2008.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[S$^+$97] R. Schoonderwoerd et al. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2), 1997.

[Sab06] R. Sabharwal. Grid infrastructure deployment using smartfrog technology. In *Proc. of the Int'l Conf. on Networking and Services (ICNS), Santa Clara, USA*, pages 73–79, July 2006.

[Wit03] O. Wittner. *Emergent Behavior Based Implements for Distributed Network Management*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2003.

[XZF$^+$07] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2007.

[Z$^+$04] M. Zlochin et al. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131:373–395, 2004.

PAPER D

Foraging for better deployment of replicated service components

Máté J. Csorba and Hein Meling and Poul E. Heegaard and Peter Herrmann

In proceedings of 9th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'09)

Lisbon, Portugal, June 9-12, 2009

FORAGING FOR BETTER DEPLOYMENT OF REPLICATED SERVICE COMPONENTS

Máté J. Csorba[1], Hein Meling[2], Poul E. Heegaard[1], Peter Herrmann[1]

[1]*Department of Telematics,*
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard, Peter.Herrmann}@item.ntnu.no

[2]*Department of Electrical Engineering and Computer Science,*
University of Stavanger,
Stavanger, Norway
Hein.Meling@uis.no

Abstract Our work focuses on distributed software services and their requirements in terms of system performance and dependability. We target the problem of finding optimal deployment mappings involving multiple services, i.e. mapping service components in the software architecture to the underlying platforms for best possible execution. We capture important non-functional requirements of distributed services, regarding performance and dependability. These models are then used to construct appropriate cost functions that will guide our heuristic optimization method to provide better deployment mappings for service components. This paper mainly focuses on dependability. In particular, a logic enabling replication management and deployment for increased dependability is presented. To demonstrate the feasibility of our approach, we model a scenario with 15 services each with different redundancy levels deployed over a 10-node network. We show by simulation how the deployment logic proposed is capable to satisfy replica deployment requirements.

1. Introduction

Distributed applications and services are increasingly being hosted by infrastructure providers over virtualized architectures, enabling on-demand resource scaling, such as in the Amazon EC2 platform [LLC]. An important concern in such platforms is the problem of *finding optimal deployment mappings* involving multiple services spread across multiple sites. During service execution a plethora of parameters influence the optimal deployment mapping, and more so in a distributed environment where concurrent services influence each other as well. Furthermore, some applications have non-functional requirements related to dependability, such as fault tolerance and high availability. Upholding such requirements demands replication protocols to ensure consistency, but also adds additional complexity to the optimization problem. Ideally, the deployment mappings should minimize the resource consumption, yet provide enough resources to satisfy the dependability requirements of services.

This paper presents a novel modeling and optimization methodology for deployment of replicated service components. We model services in a platform independent manner using the SPACE [KH06] methodology. As previously shown by Fernandez-Baca [FB89], the general module allocation problem is NP-complete except for certain communication configurations, thus heuristics are required to obtain solutions efficiently. Based on our service models, we apply an heuristic optimization method called the Cross-Entropy Ant System (CEAS) [HHW08], which is able to take multiple parameters into account when making a decision on the deployment mapping. The approach also enables us to perform optimizations in a decentralized manner, where replicated services can be deployed from anywhere within the system, avoiding the need for a centralized control for maintaining information about services and their deployment.

There are a number of reasons to develop replicated services, including fault tolerance, high availability and load balancing. This work focuses on fault tolerance and availability, and in this context, the objective is to improve the availability characteristics of the service by appropriate allocation of service replicas to nodes, such that the impact of replica failures and network failures is reduced. And at the same time minimizing the resource consumption.

Generally, to support replicated services the underlying architecture needs to provide *replication protocols* to ensure consistency between replicas, e.g., active or passive replication protocols [Sch93, BMST93]. Such protocols have different implicit communication and computation costs and can be taken into account in our model. In addition, a *replication management* infrastructure, e.g. [OMG00, MMHB08, MG08], is necessary to support deployment of replicas and managing reconfigurations when failures occur. One example is the distributed autonomous replication management framework (DARM) [MG08]. DARM focuses on the deployment and operational aspects of the system, where the gain in terms of improved dependability is likely to be the greatest. DARM is equipped with mechanisms for localizing failures and system reconfiguration. Reconfiguration is handled without any human intervention, and according to application-specific dependability requirements. The benefits of DARM are twofold: (i) the cost of deploying and managing highly available applications can be significantly reduced, and (ii) its dependability characteristics can be improved as shown in [HMM05]. The approach presented in this paper can be combined with frameworks such as DARM in order to improve the deployment mapping operation; such an implementation has been left as future work.

There are at least three cases where finding suitable deployment mappings are of significance to replication management: (i) initial deployment of replicas according to some policy; (ii) reconfiguration of deployed replicas that have failed or become unavailable due to a network partition according to some maintenance policy; (iii) migration of replicas to re-balance the system load. The deployment mapping policy used in this paper, is formulated as a cost function to the optimization problem, essentially stating that replicas should be placed on nodes and domains (sites) so as to improve the dependability of the service being deployed.

The paper is organized as follows. The next section presents how replicas are modeled in the SPACE modeling framework. Sec. 3 introduces CEAS and provides a description of the deployment algorithm. Subsequently, we formulate the optimization problem and present cost functions used to solve it. Simulation results using our logic are presented in Sec. 5. Finally, in Sec. 6 we conclude and touch upon future work.

2. Replica services in SPACE

To account for dependability requirements while deploying replicated service components, collaboration-oriented models can be used. To this end, the SPACE [KH06] methodology provides a modeling technique for automated engineering of distributed applications. In contrast to other UML-based methods, it enables the composition of system descriptions from collaboration-oriented sub-models that does not specify the behavior of a single physical component, but rather describes the sub-functionality encompassing various system entities. Such sub-models are typically easier to reuse than component-oriented building blocks, since different systems in a particular domain often have similar sub-functions, which can be coupled in various ways. Each sub-function can then easily be specified as a collaborative building block once, and thus the creation of a new system can be reduced to the design of a new combination of these pre-defined blocks.

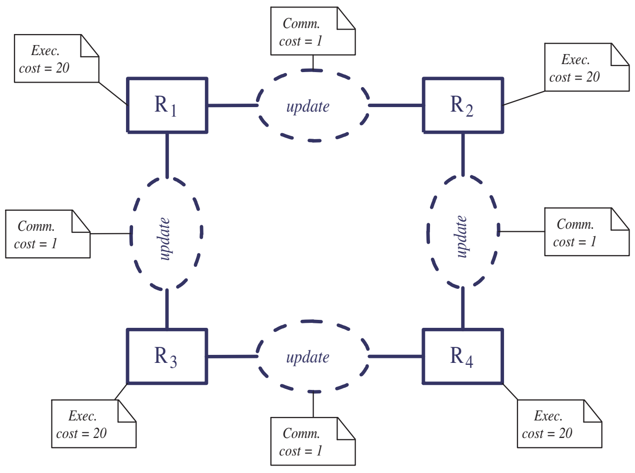


Figure 1: Example service, $S1$ with 4 replicas and corresponding costs

In SPACE, the topology of a system is modeled with UML collaborations, while behavior is described using UML activities. SPACE is accompanied by the Arctis [KBH09] tool, which enables composition of models, various model checker-based correctness proofs and automated transformation to executable Java code.

In this paper, only UML collaborations are used. Fig. 1 depicts a simple example model. It describes the pair-wise replication of data between the physical components

R_1 to R_4. The updating function is modeled by four collaboration uses called *update*, each specifying the alignment of data between the two linked components. Although SPACE offers specification of multiple instances of a collaboration [KBH07], for clarity only four instances of *update* are used here. SPACE models can be embellished with additional non-functional requirements that can be exploited by our deployment logic, i.e., the execution costs assigned to components or costs that are specific to each collaboration between replicas. Within a given service specification, some (or all) of the service components might require replication to improve their dependability. We propose to model and specify component replication using the same methodology applied for designing the services themselves. In other words, we specify a set of replicas related to a specific component separately, i.e. as a collaboration of replicas of a single component.

To test our deployment logic, we assume an active replication approach, where each replica of the service performs according to the client requests. Thus, replicas have the same execution cost. Each replica is also assigned a communication (collaboration) cost to account for the cost of ensuring consistency (state updates) between replicas. The example scenario illustrated in Fig. 1 is used as a basis for the simulations presented in Sec. 5.

3. Replica Deployment using the Cross Entropy Ant System

To find suitable replica placements a collection of ant-like agents, denoted *ants*, search iteratively for the best solution according to a cost function, restricted by the problem constraints. To find a solution ants are guided using the analogy of *pheromones*, which are proportional to the quality of the solution. CEAS uses the *Cross Entropy method* for stochastic optimization introduced by Rubinstein [Rub99], and has demonstrated its capabilities and relevance through a variety of studies of different path management strategies. For an intuitive explanation and introduction to CEAS, see [HHW08].

Table 1 gives the notation for sets and variables used throughout our description.

Table 1: Notational shorthand

Shorthand	Usage	Description
$\mathbf{S}$	$S_k \in \mathbf{S}$	set of service instances
$\mathbf{C}_k$	$c_i \in \mathbf{C}_k$	set of all replicas in S_k
$\mathbf{D}$	$d \in \mathbf{D}$	set of all existing domains
$\mathbf{N}$	$n \in \mathbf{N}$	set of all existing nodes
$\lvert\mathbf{C}_k\rvert$	$\lvert\mathbf{C}_k\rvert$	number of replicas to be deployed
D_r	$d \in D_r$	list of domains used in deployment of S_k
N_r	$n \in N_r$	list of nodes used in deployment of S_k
NL_r	$nl_{n,r} \in NL_r$	load-level samples for S_k
M_r	$m_{n,r} \in M_r$	mapping list for S_k
H_r	$n \in H_r$	hop-list for S_k

In this paper, we apply CEAS to obtain the best mapping of a set of replicas onto a set of nodes, $M : \mathbf{C} \rightarrow \mathbf{N}$. The pheromone values used by the ants, denoted $\tau_{mn,r}$, correspond to a set of replicas, m mapped to node n at iteration r. Ants use a random proportional rule for selecting the individual mappings.

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in M_{n,r}} \tau_{ln,r}} \tag{1}$$

The pheromone values $\tau_{mn,r}$ in (1) are updated continuously by the ants as follows:

$$\tau_{mn,r} = \sum_{k=1}^{r} I(l \in M_{n,r}) \beta^{\sum_{j=k+1}^{r} I(j \in M_k)} H(F(M_k), \gamma_r) \tag{2}$$

where $I(x) = 1$ if x is true, 0 otherwise. See [HHW08] for further details.

A parameter γ_r denoted the *temperature*, controls the update of the pheromone values and is chosen to minimize the performance function

$$H(F(M_r), \gamma_r) = e^{-F(M_r)/\gamma_r} \tag{3}$$

which is applied to all r samples.

To enable a distributed optimization process the cost of a mapping, $F(M_r)$ is calculated *immediately* after each sample i.e., when all replicas are mapped, and an auto-regressive performance function, $h_r(\gamma_r)$ is applied, Eq. 4.

$$h_r(\gamma_r) \approx \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(M_r), \gamma_r) \tag{4}$$

where $\beta \in \langle 0, 1 \rangle$ is a *memory factor* weighting (geometrically) the output of the performance function. This mechanism smooths variations in the cost function, hence rapid changes in the deployment mapping and undesirable fluctuations can be avoided. The temperature, γ_r is determined by minimizing it subject to $h(\gamma) \geq \rho$, thus

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(M_i), \gamma) = \rho\} \tag{5}$$

where ρ is a parameter (denoted *search focus*) close to 0 (typically 0.05 or less).

Eq. (5) is a transcendental function that is storage and processing intensive since all observations up to the current sample, i.e., the entire mapping cost history $F(M_r), \forall r$ should be stored, and weights for all observations would have to be recalculated, thus putting an impractical burden on the on-line operation of the logic. Accordingly, we assume that, given a β close to 1, changes in γ_r are typically small from one iteration to the next, enabling a first order Taylor expansion of (5), and a second order Taylor expansion of (2), see [HHW08] for more details. More importantly, we are able to obtain an optimal deployment mapping with high confidence, since CEAS can be considered as a subclass of Ant Colony Optimization (ACO) algorithms [DMC96], which have been proven to be able to find the optimum at least once with probability close to one. Once the optimum has been found, convergence is secured in a finite number of iterations.

Algorithm 1 Code for $Nest_k$

```
1: Initialization:
2:     $r \leftarrow 0$                                {Number of iterations}
3:     $\gamma_r \leftarrow 0$                               {Temperature}
4: while $r < R$                               {Stopping criteria}
5:     $antAlgo(r, \gamma_r)$                          {Emit new ant}
6:     $r \leftarrow r + 1$
```

We now present the steps executed by the deployment logic to obtain a mapping of replicas. Behavior of the logic is separated into Algorithm 1, which describes the simple functionality of a *Nest*, i.e. basic additional intelligence in one of the nodes, and Algorithm 2, which describes the behavior of the ants that are subsequently emitted from the *Nest*. The role of a *Nest* can be played by an arbitrary node. The steps are executed independently by ants of each species, where a species is directly involved in the deployment of a specific service. Each ant initiated from the nest node of a species is assigned a set of replicas, **C**; in this case the replica instances to deploy. The ant then starts a random-walk in the network, selecting the next hop at random. Behavior at a node depends on if the ant is an *explorer* or a *normal* ant. *Normal* ants select a subset of **C** for mapping to the current node according to the pheromone database and store this selection in the mapping list, M_r. An *explorer* ant, however does the selection without using the pheromone values in a completely random manner.

The benefits of applying *explorer* ants are twofold, first they initially explore the solution space and second, they are used for faster discovery of changes in the network during optimization. In both cases, *explorers* do not use the pheromone tables, instead they build up an initial database. Besides, they are used to detect alternative solutions while the system undergoes short- or long-term changes. The amount of *explorer* vs. *normal* ants is a configurable ratio parameter to the logic. Initial exploration is essentially a random sampling of the problem space and the number of iterations depends on the problem size. However, the end of this phase can be detected by monitoring the pheromone database size. Optimizing the deployment mappings based on the available cost functions should be performed using a distributed method, avoiding a centralized structure. To do so, each node provides a processing power reservation mechanism. Ant species use this mechanism to indicate their resource usage in every node they utilize for their replicas. Processing power reservation can be updated by a given percentage of ants, which is again a parameter to the logic, i.e., only a certain fraction (e.g. 10%) of iterations result in re-allocation at the nodes, see Lines $18-21$ in Algorithm 2. Outdated allocations get invalidated in the nodes to preserve consistency. In addition, the allocation mechanism can serve as a means of interaction between the species. Thus, the current sum of allocations in a node can be sampled providing a general overview for the ants. These load-level samples are denoted NL_r. The decreased ratio of reservations by the ants (e.g. only 10% of them) contributes to obtaining a smoother series of NL_r samples. The actual implementation of sampling is left to the middleware.

The *forward search* phase of an ant is over when all component replicas are mapped and the resulting mapping is stored in M_r. The algorithm proceeds with evaluating the resulting mapping using the appropriate cost function $F_i()$. After evaluating the cost of the mapping, the ant *backtracks* to its nest using the hop-list, H_r. During *backtracking*, pheromone values distributed across the network of nodes are updated according to Eq. (2). After the ant finds its way back to the nest node or times out a new ant can be initiated and emitted. The same behavior can be used for all ants, even though they are of different species.

The main purpose of the pheromone database is its usage in Algorithm 2, Line 13. In every iteration, an ant will form $|N_r|$ discrete subsets of $\mathbf{C}$ as it visits $n \subseteq N_r$ nodes. In order to be able to describe replica mappings to nodes, values of the pheromone database have to be aligned with replica sets. Accordingly, the pheromone database is built by assigning a flag to every replica available for deployment in a service, $\forall c_i \in \mathbf{C}$, with the exception of replicas that are bound to specific nodes explicitly by requirements and thus, they cannot be moved.

Algorithm 2 Ant code for deployment mapping of component replicas $\mathbf{C} \in S_k \subset \mathbf{S}$ from $Nest_k$

```
1: Initialization:
2:    $H_r \leftarrow \emptyset$                                   {Hop-list; insertion-ordered set}
3:    $M_r \leftarrow \emptyset$                                   {Deployment mapping set}
4:    $D_r \leftarrow \emptyset$                                   {Set of utilized domains}
5:    $NL_r \leftarrow \emptyset$                                  {Set of load samples}
6: function $antAlgo(r,k)$
7:    $\gamma_r \leftarrow Nest_k.getTemperature()$               {Read the current temperature}
8:    while $\mathbf{C} \neq \emptyset$                            {More replicas to deploy}
9:       $n \leftarrow selectNextNode()$                          {Select first node}
10:      if explorer ant
11:         $m_{n,r} \leftarrow random(\subseteq \mathbf{C})$     {Explorer ant; randomly select a set of replicas}
12:      else
13:         $m_{n,r} \leftarrow rndProp(\subseteq \mathbf{C})$    {Normal ant; select replicas according to Eq. (1)}
14:      if $\{m_{n,r}\} \neq \emptyset, n \in d_k$               {At least one replica mapped to this domain}
15:         $D_r \leftarrow D_r \cup d_k$                         {Update the set of domains utilized}
16:      $M_r \leftarrow M_r \cup \{m_{n,r}\}$                    {Update the ant's deployment mapping set}
17:      $\mathbf{C} \leftarrow \mathbf{C} - \{m_{n,r}\}$         {Update the set of replicas to be deployed}
18:      if $r$ mod 10 = 0                                         {Only every 10th ant modifies allocations}
19:         foreach $c_i \in m_{n,r}$
20:            $sumpp \leftarrow sumpp + f_{c_i}$                 {Sum the exec. costs imposed by $S_k$}
21:         $n.reallocProcLoad(S_k, sumpp)$                       {(re-)allocate processing power needed by $S_k$}
22:      $nl_{n,r} \leftarrow n.getEstProcLoad()$                 {Get the estimated processing load at node n}
23:      $NL_r \leftarrow NL_r \cup \{nl_{n,r}\}$                 {Add to the list of samples}
24:   $cost \leftarrow F(M_r, D_r, NL_r)$                         {Parameters depending on the cost function}
25:   $\gamma_r \leftarrow updateTemp(cost)$                      {Given cost, recalculate temperature according to Eq. (5)}
26:   foreach $n \in H_r.reverse()$                               {Backtrack along the hop-list}
27:      $n.updatePheromone(m_{n,r}, \gamma_r)$                   {Update pheromone value at n, Eq. (2)}
28:   $Nest_k.setTemperature(\gamma_r)$                           {Update $\gamma_r$ at $Nest_k$}
29: function $selectNextNode()$                                   {SELECT UNIQUE RANDOM NODE}
30:   $\mathbf{R} \leftarrow \mathbf{N} - currentNode$            {Set of candidate nodes for ant traversal}
31:   $n \leftarrow random(\mathbf{R})$                           {Select candidate node at random}
32:   $H_r \leftarrow H_r \cup \{n\}$                             {Add node to the hop-list}
33:   return $n$
```

The pheromone database will contain $2^{|\mathbf{C}|}$ elements, equal to the number of possible combinations for a set c_i at a node, which is specific for each service. This determines a physical requirement for the execution platform that supports our logic, namely to be able to accommodate $2^{|\mathbf{C}|}$ floating point numbers for each of the services in every node. If the pheromone database in a node is normalized between $\{0 \ldots 1\}$ it can be observed as a probability distribution of replica sets mapped to that node. In a converged state the optimal solution(s) will emerge with probability one.

4. Construction of the Cost Function

When applying the optimization method presented in Sec. 3 it is essential to formulate a proper cost function aimed at guiding the optimization process towards an *appropriate solution*. An *appropriate solution* is a solution to the deployment mapping problem satisfying the system requirements, F_{req} derived from the service specification, while accounting for the costs of the mapping, $F_i()$. Trying to find a global optimal solution does not make much sense in the systems considered here, as the solution would most likely be suboptimal by the time, the optimal mapping could be applied. However, the algorithm can continue optimization even after a feasible mapping is found, that can trigger (re-)deployment of replicas. By optimal mapping we mean mappings with the lowest possible cost, while for a feasible mapping $F_i()$ $< F_{req}$ is enough. Note that the formulation of the deployment problem below is independent of the methods we apply to obtain a solution.

$$\boxed{\begin{array}{l}\min F_i() \quad \{< F_{req}\} \\ \text{subject to } \mathbf{\Phi}\end{array}}$$

In each iteration of our deployment logic, the cost function is evaluated for every suggested mapping, $m_{n,r}$, (cf. Algorithm 2, Line 24). Properties of this function impact the quality of the solutions obtained as well as the convergence time, or in other words, the number of iterations required to reach a stable solution. In order to develop a logic that can aid replica deployment and increase dependability by influencing the mapping of software architecture the cost function has to be carefully selected. However, what is the proper function to use depends on the requirements and goals of the service. Here, we target efficient placement of component replicas in an active replication scheme aimed at improving the dependability.

We define the mapping functions f_k and $g_{k,d}$ as follows.

Definition 1. *Let f_k: $r_k \rightarrow d$ be the mapping of replica r_k to domain $d \in \mathbf{D}$*

Definition 2. *Let $g_{k,d}$: $r_k \rightarrow n_d$ be the mapping of replica r_k to node $n_d \in \mathbf{N}$ in domain $d \in \mathbf{D}$*

We then define two distinct rules that the deployment logic targets. The first one states that replicas shall be distributed across as many domains as possible for increased dependability, i.e. two replicas of the same service shall not be placed in the same domain preferably, or if there are more replicas than domains available there shall be at least one replica in all domains (ϕ_1).

Rule 1 $\phi_1 : f_k \neq f_l \iff k \neq l \wedge |S_k| < |\mathbf{D}|$

Whereas the other rule declares that two replicas of the same service should not be co-located on the same node (ϕ_2).

Rule 2 $\phi_2 : g_{k,d} \neq g_{l,d} \iff k \neq l, \forall d$

Deployment mappings of component replicas can be evaluated by the deployment cost function $F_i()$. Accordingly, we formulate the replica deployment problem as the task of minimizing $F_i()$ subject to $\Phi = \phi_1 \wedge \phi_2$.

The problem of producing deployment mappings that conform to the rules introduced above is approached step-wise by introducing different types of cost functions. We start by considering ϕ_1 only and use information collected by the ant species during *forward search* by counting the number of domains that have been used to map replicas at an iteration, this variable will be denoted D_r. Using D_r we will experiment with a reciprocal (6) and a linear function (7) too. The latter case uses the number of replicas, $|\mathbf{C}|$, a constant derived from the service model and thus known to each species.

$$F_1(D_r) = \frac{1}{|D_r|} \tag{6}$$

$$F_2(D_r, \mathbf{C}) = |\mathbf{C}| - |D_r| + 1 \tag{7}$$

Similarly, we include ϕ_2 into the cost function by a reciprocal and a linear function and combine it with (6) and (7) as follows.

$$F_3(D_r, N_r) = \frac{1}{|D_r|} \cdot \frac{1}{|N_r|} \tag{8}$$

$$F_4(D_r, N_r, \mathbf{C}) = (|\mathbf{C}| - |D_r| + 1) \cdot (|\mathbf{C}| - |N_r| + 1) \tag{9}$$

$$F_5(D_r, N_r, \mathbf{C}) = \frac{1}{|D_r|} \cdot (|\mathbf{C}| - |N_r| + 1) \tag{10}$$

$$F_6(D_r, N_r, \mathbf{C}) = (|\mathbf{C}| - |D_r| + 1) \cdot \frac{1}{|N_r|} \tag{11}$$

In (8)-(11) we utilize a variable, N_r, which denotes the number of nodes that have been used by a specific species for deploying replicas at iteration r, this is also reported by each ant during the *forward search* phase. We evaluate all four possible combinations of the reciprocal and linear functions targeting ϕ_1 and ϕ_2.

The last combination of cost functions, in (12), is a combination of the simple reciprocal function in (6) targeting ϕ_1 combined with a more complex function used successfully in service component deployment [CHH08].

$$F_7(D_r, M_r, NL_r) = \frac{1}{|D_r|} \cdot F_{lb}(M_r, NL_r) \tag{12}$$

F_{lb} uses two parameters that are updated in every iteration, the replica mapping set M_r and the load-level samples taken in the nodes visited by the ant ($n_j \in H_r$), denoted NL_r. This function accounts for the execution and communication costs derived from the service specification as introduced in Sec. 2. Correspondingly, the function consists of two main parts, node (*NC*) and collaboration related costs (*LC*).

$$F_{lb}(M_r, NL_r) = [\sum_{\forall n_j \in H_r} NC(n_j)] \cdot (1 + x \cdot LC) \tag{13}$$

where x is a parameter used to balance the effect of the *LC* term, as needed. The component *LC* is strictly local to each species and incorporates the collaboration costs

$$LC(M_r) = \sum_{j=1}^{K} I_j\, f_{k_j}; \text{ where } I_j = \begin{cases} 1, & \text{if } Collab_k \text{ external} \\ 0, & \text{if } Collab_k \text{ internal to a node} \end{cases} \tag{14}$$

Thus, the term *LC* will take into account communication costs (f_{k_j}) assigned to those collaborations that happen between different nodes only, in other words aiming at minimizing remote communication.

Costs related to execution of replicas, i.e., node local costs are incorporated into the first term in (13). Node local costs aim at achieving load-balancing among the nodes hosting replicas. Importantly, in this term only the subset of nodes an ant has actually visited (H_r) is taken into account, not the total amount of nodes. The term that is calculated individually for each of the nodes in H_r is shown in (15).

$$NC_{n_j}(NL_{n,r}) = [\sum_{i=0}^{NL_{n,r}(n_j)} \frac{1}{\sum_{\forall n_j \in H_r} NL_{n,r} + 1 - i}]^y \tag{15}$$

The term *NC* counteracts the other term in (13), *LC*, which puts weight on replica mappings that have as much as possible of the collaborations within the same node(s). *NC* has an effect of distributing replicas, thus equalizing execution load among the available nodes to the highest extent possible. This way two counteracting requirement types are tackled in the same function. The exponent y in (15) can shift the focus towards load-balancing against minimization of remote communication in collaborations. In the experiments in this paper we use $x = 10^{-5}$ and $y = 2$, which are adjusted to the cost values derived from the models, e.g. see the example service in Fig. 1.

Using F_{lb} we are able to smoothen the output of the cost evaluation executed for each iteration of the deployment logic. Its purpose is to ease convergence of the logic by making the solution space more fine grained, i.e. simplifying differentiation between very similar deployment mappings with nearly the same cost value.

The next section presents simulation results evaluating all the cost functions presented here using an example setup.

5. Simulation Results

To evaluate our approach and the proposed cost functions, we developed a test scenario. The scenario consists of a network of 10 identical nodes clustered into 5 domains (cf. Fig. 2). The 5 domains have $3, 2, 1, 1, 3$ nodes. Using this network of nodes, each ant species executing Algorithm 2 is assigned a replica service for deployment. A set of 15 actively replicated services with redundancy levels shown in Table 2 is used for the evaluation.

Table 2: Service instances in the example

Service	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
#replicas	4	6	4	4	4	5	5	6	6	6	6	7	8	9	10

For example, see $S1$ in Fig. 1. Each replica within a service has identical execution cost, and all replicas have the same cost in all services. Similarly, the same is true for the communication costs, i.e. $f_{c_i} = 20, \forall i$ and $f_{k_j} = 1, \forall j$.

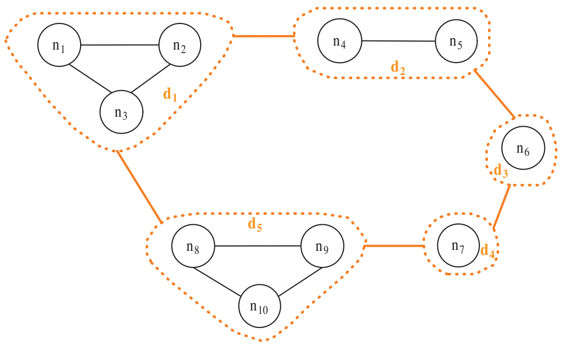


Figure 2: Test network of hosts clustered into 5 domains

For the evaluation scenario with $S_1 \ldots S_{15}$, the deployment logic (Algorithm 2) is executed 50 times using the cost functions discussed in Sec. 4 and we compare their behavior. The deployment logic was described by a process-oriented simulation model implemented in Simula/DEMOS [Bir03].

For the problem at hand, deploying replicas of each service yield $\mathbf{N}^{C_k}$ mapping combinations; deploying all 15 services simultaneously would account for an exhaustive search of $\mathbf{N}^{\Sigma C_k} = 10^{90}$ possible configurations. For the evaluation, the execution of Algorithm 2 was limited to $r_{max} = 30000$ iterations (significantly smaller than exhaustive search), unless convergence is obtained earlier. All 15 species, one for each service, were executed simultaneously. This is in accordance with our goal to find an appropriate solution within reasonable time, even though it may not be the optimal mapping. After each run, the obtained deployment mapping was checked against ϕ_1 and ϕ_2. Results for selected functions are presented in Table 3.

In case of $F_1(D_r)$, which is based on observing the number of domains (D_r) utilized for deployment mapping of replicas, ϕ_2 (cf. Sec. 4) is not checked because the cost function does not consider this rule. From the 50 independent runs we see that in some cases ϕ_1 is not satisfied; some of the 15 services fail to utilize as many domains as they could. That is due to the limited number of iterations we allowed for the species to achieve convergence and because this cost function is very simple, i.e. lacking a more smooth, more fine grained evaluation of the deployment mappings for the ant species.

In the second branch of cost functions, Eq. (8)–(11), we apply two very simple functions together to take into account ϕ_1 and ϕ_2 at the same time. The experiments show that the combination of two functions of the same kind, i.e. two linear or two reciprocal functions, gives inferior results to applying a combination of one reciprocal and a linear. This might be caused by smoother cost output in case of the latter, which results in better convergence and better solution quality, i.e. a deployment mapping that satisfies the requirements with a higher probability. Nevertheless, there were 2 violations of ϕ_2 within the 50 runs, that means that one replica was co-located with another in one of the services. This is possible for services that have a high number of replicas, e.g. 9 or 10, which easily occupy 5 domains, thus obtain the lowest cost possible considering the first part of the cost function resulting in a mapping that violates ϕ_2 after convergence. These services, with these simple cost functions are able to decrease their mapping costs only marginally by spreading their replicas further among the available hosts, which results in sub-optimal solutions, thus violations of ϕ_1 or ϕ_2.

Now, if we look at the last combination of functions in Table 3, we can see how our load-balancing function performed with the extension of taking into account the number of domains utilized (D_r). From the 50 independent runs the deployment logic converged to a stable solution in all of the cases. ϕ_1 was successfully taken into account by the first reciprocal term and resulted in no violations. In one case however, one of the services failed to satisfy ϕ_2, i.e. a replica was co-located with another one. After a closer look we can see that this involved service S_{15} comprising 10 replicas. The reason for this violation is that the load-balancing function, $F_{lb}(M_r, NL_r)$, has enforced a deployment mapping, which was better for global load-balancing in this particular case by taking into account this global goal to a greater extent and thus, violating the rule prohibiting co-location of replicas. However, as in S_{15} the number of replicas is equal to the number of available nodes there is not much space left for the logic to place replicas so that load-balancing is also achieved, which is the main goal for this part of the cost function. Clearly, applying the cost function we propose implies taking a broader view on the deployment problem. The tradeoff might be that under certain circumstances the mapping of replicas might violate one of the rules formulated, but the gain is that we can obtain a globally better and more effective

Table 3: Replication rules satisfied, 50 trials each

Cost function	ϕ_1	ϕ_2	Comments
$F_1(D_r)$	88%	n/a	all due to no convergence
$F_5(D_r, N_r, \mathbf{C})$	100%	96%	all due to no convergence
$F_7(D_r, M_r, NL_r)$	100%	98%	all converged

mapping, still using a fully distributed logic and doing so faster, i.e. within reasonable time.

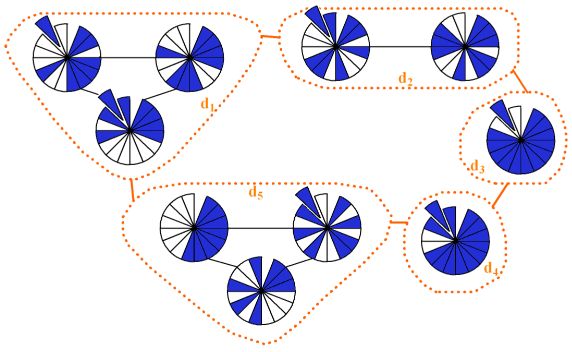


Figure 3: Example mapping of replicas in $S_1 \ldots SS_{15}$, with S_2 exploded

To get a picture of how replicas are mapped to the underlying nodes clustered into domains one of the possible mappings is depicted in Fig. 3, in which each slice of the pie diagram corresponds to a specific service $S_k \subset \mathbf{S}$. As in this optimal mapping there is no co-location of replicas, a slice being shaded means that there is a single replica placed on the particular node. It is easy to notice that the two domains consisting of a single node (d_3, d_4) are heavily packed with replicas due to the fact that there are many services, which can exploit 5 domains or more. This makes overall load-balancing among the available nodes more difficult.

Furthermore, to illustrate the behavior and convergence of our logic, in Fig. 4 we look at the cost output of some species that guides the mapping of replicas as a function of number of iterations.

The three services presented have 6 (S_{11}), 8 (S_{13}) and 10 (S_{15}) replicas to deploy. The first 2000 iterations, i.e. the *exploration phase* is not shown in the figure. After 2000 initial iterations optimization continues and the cost values decrease, thus indicating increasingly improved mapping of replica components. We stop the simulation where the costs do not improve anymore, in this particular case after approximately 10000 additional iterations. By checking ϕ_1 and ϕ_2 we can see that the mapping obtained in this run satisfies both. In case where the number of replicas is high, e.g. 10, values do not deteriorate too much from a consensus level between the parallel species (considering $N = 10$) as ϕ_1 restricts the solution space. That means that for service S_{15} all of the nodes have to host one replica according to the rule. Whereas services with less replicas to deploy have a significantly larger valid solution space, i.e. service S_{11} and S_{13} find, in some cases, a solution satisfying ϕ_1 and ϕ_2 but with a higher overall cost, thus we can see some deviations in the cost output in the figure before obtaining convergence. After consensus is reached among the species the actual deployment of component replicas can be triggered. Practically, when a species detects that the cost values obtained by its ants are stable over a period of time, replicas corresponding to this species can be (re-)deployed. The technical solution as well as the protocol of replica placement/re-deployment is, however, left to the middleware (e.g., DARM).

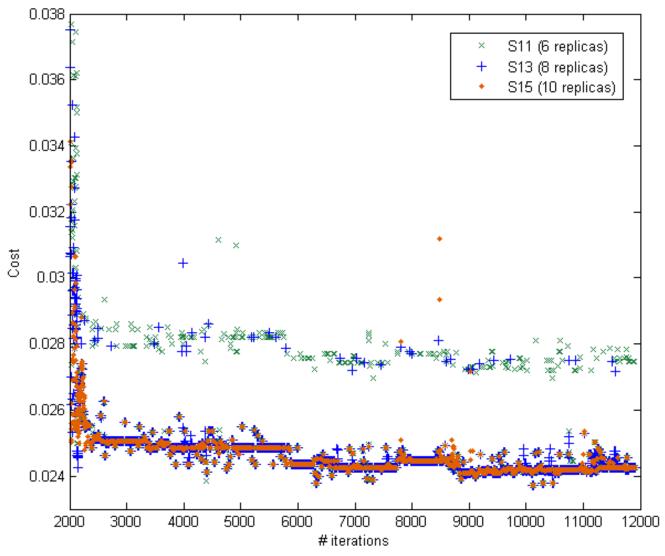


Figure 4: Costs with 6, 8 and 10 replicas

6. Closing Remarks

Our focus has been on a heuristic optimization technique aided by swarm intelligence that can manage deployment of software components, in particular component replicas for increased dependability. To obtain an efficient mapping of replicas we utilize service models specified as UML 2.0 collaborations. These models are enriched with non-functional requirements that are used in the cost evaluation of the mappings made by the deployment logic. Importantly, our method is a fully distributed approach, thus it is free of discrepancies most of the existing centralized solutions suffer from, e.g. performance bottlenecks and single points of failure. Instead of having a centralized database we use the analogy of pheromones used by foraging ants as a distributed database across the network of hosts. This database can be quite compact as all the intelligence is carried along by the ant-like agents.

We have showed that, using CEAS for optimization and SPACE for modeling, the deployment logic is capable of handling various non-functional requirements present in service specifications. Extending on our previous work, in this paper focus has been on how the logic can deal with basic dependability requirements concerning replication management. Eventually, our goal is to aid run-time (re-)deployment and replication of software components while considering the execution environment and satisfying the requirements of the service.

For future work we plan to introduce new types of species corresponding to user demands towards the services being deployed. However, this will introduce new challenges as it will increase the dimensions of the deployment problem significantly. More fine grained cost modeling, e.g. passive replication, with costs dependent on replica attributes will be part of future investigations. This will also involve more extensive simulations with new experimental settings. Larger network sizes will also be investigated together with their impact on convergence and scalability. Another interesting aspect we will experiment with is how splitting/merging of domains influences the output of our logic, besides assessing what level of node churn can be tolerated by our method. Generally, context-aware adaptation is considered one of the main tracks we follow in our future work.

References

[Bir03] G. Birtwistle. *Demos - a system for Discrete Event Modelling on Simula.* Springer-Verlag New York, Inc., 2003.

[BMST93] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. pages 199–216, 1993.

[CHH08] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In *Proc. of the 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), Turin*. ICST/ACM, September 2008.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[FB89] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. on Software Engineering*, 15(11):1427–1436, 1989.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HMM05] B. E. Helvik, H. Meling, and A. Montresor. An approach to experimentally obtain service dependability characteristics of the jgroup/arm system. In *Proc. of the 5th European Dependable Computing Conference (EDCC), LNCS 3463*, pages 179–198, 2005.

[KBH07] F. A. Kraemer, R. Bræk, and P. Herrmann. Synthesizing components with sessions from collaboration-oriented service specifications. In *Proc. of the 13th Int. SDL Forum conference on Design for dependable systems, LNCS 4745*, pages 166–185, Berlin, Heidelberg, 2007. Springer-Verlag.

[KBH09] F. A. Kraemer, R. Bræk, and P. Herrmann. Compositional service engineering with arctis. *Telektronikk*, (01), 2009.

[KH06] F. A. Kraemer and P. Herrmann. Service specification by composition of collaborations - an example. In *Proc. of the 2006 Int'l Conf. on Web Intelligence and Intelligent Agent Technology, Hong Kong*. IEEE/WIC/ACM, 2006.

[LLC] Amazon Web Services LLC. Amazon elastic compute cloud. `http://aws.amazon.com/ec2`.

[MG08] H. Meling and J. L. Gilje. A distributed approach to autonomous fault treatment in spread. In *Proc. of the 7th European Dependable Computing Conference (EDCC)*, pages 46–55, 2008.

[MMHB08] H. Meling, A. Montresor, B. E. Helvik, and Ö. Babaoglu. Jgroup/arm: a distributed object group platform with autonomous replication management. *Softw., Pract. Exper.*, 38(9):885–923, 2008.

[OMG00] OMG. Fault tolerant CORBA specification. Technical Report ptc/00-04-04, OMG Document, April 2000.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[Sch93] F. B. Schneider. Replicated management using the state-machine approach. *Distributed systems (2nd Ed.)*, pages 169–197, 1993.

PAPER E

Laying Pheromone Trails for Balanced and Dependable Component Mappings

Máté J. Csorba and Hein Meling and Poul E. Heegaard

In proceedings of 4th IFIP TC 6 International Workshop on Self-Organizing Systems (IWSOS'09)

Zurich, Switzerland, December 9-11, 2009

LAYING PHEROMONE TRAILS FOR BALANCED AND DEPENDABLE COMPONENT MAPPINGS

Máté J. Csorba[1], Hein Meling[2], Poul E. Heegaard[1]

[1]*Department of Telematics,*
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard}@item.ntnu.no

[2]*Department of Electrical Engineering and Computer Science,*
University of Stavanger,
Stavanger, Norway
Hein.Meling@uis.no

Abstract This paper presents an optimization framework for finding efficient deployment mappings of replicated service components (to nodes), while accounting for multiple services simultaneously and adhering to non-functional requirements. Currently, we consider load-balancing and dependability requirements. Our approach is based on a variant of Ant Colony Optimization and is completely decentralized, where ants communicate indirectly through pheromone tables in nodes. In this paper, we target scalability; however, existing encoding schemes for the pheromone tables did not scale. Hence, we propose and evaluate three different pheromone encodings. Using the most scalable encoding, we evaluate our approach in a significantly larger system than our previous work. We also evaluate the approach in terms of robustness to network partition failures.

1. Introduction

Data centers are increasingly used to host services over a virtualized infrastructure that permits on-demand resource scaling. Such systems are often comprised of multiple geographically dispersed data center sites to accommodate local demand with appropriate resources, and to ensure availability in case of outages. Major service providers, e.g. Amazon [LLC], Google, Yahoo! and others all use such infrastructures to power their world wide web offerings, popularly called cloud computing infrastructures. These systems are typically built using large numbers of cheap and less reliable blade servers, racks, hard disks, routers, etc., thus leading to higher failure rate [Dea]. To cope with increased failure rates, replication and repair mechanisms are absolutely crucial.

Another related and important concern in such data center infrastructures is the problem of *finding optimal deployment mappings for a multitude of services*, while ensuring proper balance between load characteristics and service availability in every infrastructure site. During execution a plethora of parameters can impact the deployment mapping, e.g due to the influence of concurrent services. Another set of parameters in the mix is the dependability requirements of services. Upholding such requirements not only demands replication protocols to ensure consistency, but also adds additional complexity to the optimization problem. Ideally, the deployment mappings should minimize resource consumption, yet provide enough resources to satisfy the dependability requirements of services. However, Fernandez-Baca [FB89] showed that the general module allocation problem is NP-complete except for certain communication configurations, thus heuristics are required to obtain solutions efficiently.

This paper extends our previous work to find optimal deployment mappings [CMHH09], [CHH08] based on a heuristic optimization method called the Cross-Entropy Ant System (CEAS). The strengths of the CEAS method is its capability to account for multiple parameters during the search for optimal deployment mappings [HHW08]. The approach also enables us to perform optimizations in a decentralized manner, where replicated services can be deployed from anywhere within the system, avoiding the need for a centralized control for maintaining information about services and their deployment.

The main goal of this paper is twofold; to provide additional simulation results (i) involving scaling up the problem size, both in terms of number of nodes and replicas deployed, and (ii) evaluating its ability to tolerate network partition failures (split/merge). Scaling up the problem size turned out to be more challenging than first anticipated, and thus certain enhancements were necessary in the algorithm and the data representation. To tackle the challenges we met, we have introduced a new cost function, run-time binding of replicas, a new method for selecting next-hops and new pheromone encodings. In addition, we have used more simple service models in the current study. There are generally two branches of works where finding optimal replica deployment mappings are necessary and useful. On the one hand, virtual machine technology is increasingly being used in data centers for providing high availability and thus needs to consider the placement of replicas in the data centers to ensure efficient utilization of the system resources. The advantage of this approach is that (server) applications running on virtual machines can be repaired simply by regenerating them in another physical machine. This is the approach taken by the Amazon EC2 system [LLC] and in VMware, among others. The general drawback with virtualization for fault tolerance and high availability is that the storage system used to maintain application state must be independently replicated as it would otherwise constitute a single point of failure. On the other hand, server applications written specifically for fault tolerance typically replicate their application state to all replica processes, avoiding any single points of failure. These systems are typically built using a middleware based on a group communication system with support for repair mechanisms, e.g. DARM [MG08].

The importance and utility of deployment decision making and optimization has been identified previously, e.g. in [KHD08]. Recently, Joshi et al [JHJ09] proposed a centralized approach in which an optimizer and model solver component is used to find optimal mappings specifically in the field of virtual machine technology. We however, intend to pursue a fully distributed solution that is based on optimization techniques and can support context-awareness and adaptation.

The paper is organized as follows. The next section presents our view on component replicas, their deployment, corresponding costs and requirements. In Sec. 3 the fundamentals of the CEAS are described. Sec. 4 proposes our algorithm for solving the deployment mapping problem and subsequently we demonstrate its operation in Sec. 5. Finally, we conclude and touch upon future work.

2. System Model, Assumptions and Notation

We consider a large-scale distributed system consisting of a collection of *nodes*, $\mathscr{N}$, connected through a network. Nodes are organized into a set of *domains*, $\mathscr{D}$, as illustrated by d_1 and d_2 in Figure 1. All nodes within a domain are located at the same geographic site, whereas different domains are in separate sites. The objective of the distributed system is to provide an environment for hosting a set of *services*, $\mathscr{S} = \{S_1, S_2, \ldots\}$, to be provided to external clients. Let C_i^k be the i^{th} *component* of service k, and let $S_k = \{C_1^k, \ldots, C_q^k\}$ denote the set of components for service k, where $q = |S_k|$. Each component may be replicated for fault tolerance and/or load-balancing purposes. Thus, let R_{ij}^k denote the j^{th} replica of C_i^k. Hence, $C_i^k = \{R_{i1}^k, \ldots, R_{ip_i}^k\}$, where $p_i \geq 1$ is the redundancy level of C_i^k. Moreover, $S_k = \{R_{11}^k, \ldots, R_{1p_1}^k, \ldots, R_{i1}^k, \ldots, R_{ip_i}^k\}$ is the expansion of the component sets into replicas for service k.

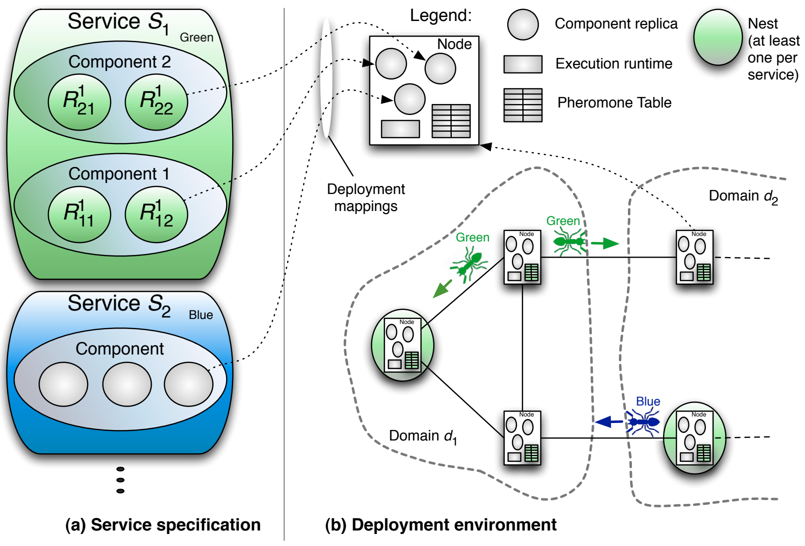


Figure 1: Overview of the deployment environment and service specification.

The objective of the algorithms herein is to discover suitable *deployment mappings* between *component replicas* (*replicas* for brevity) and nodes in the network, such that the dependability requirements of all services are preserved with minimal resource consumption. To accomplish this, the CEAS optimization method is used, which works by evaluating a *cost function*, $F()$, for different deployment mappings. The CEAS method is implemented in the form of *ants* moving around in the network to identify potential locations where replicas might be placed. An ant is simply an agent with associated state; as such it is simply a message on which the ant algorithm is executed at each visited node. We say that different *ant species* are responsible for different services, e.g. the green and blue ants in Figure 1 represent the green and blue services, respectively.

As Figure 1 shows, each node contains an *execution runtime* whose tasks are to install, run and migrate replicas. A node also has a *pheromone table* which is manipulated by ants visiting the node to reflect their knowledge of existing mappings. Moreover, the pheromone table is used by ants for selecting suitable deployment mappings; it is not used for ant routing as in the Ant Colony Optimization (ACO) approach [DMC96]. See Sec. 4.2 for details.

To deploy a service, at least one node must be running a *nest* for that service. The tasks of a nest are twofold: (i) to emit ants for its associated service, and (ii) trigger installation of replicas at nodes, once a predefined *convergence criteria* is satisfied, e.g. after a certain number of iterations of the algorithm. An iteration is defined as one round-trip trajectory of the ant, during which it builds a *hop list*, H_r, of visited nodes. A nest may be replicated for fault tolerance, and emit ants independently for the same service. During execution of the CEAS method, synchronization between nests is not necessary, but only a primary nest will execute deployment decisions. Figure 1 shows a two-way replicated nest for the green service; nests for the blue service are not shown.

Initially, the composition of services to be deployed is specified as UML collaborations embellished with non-functional requirements that are used as input to the cost function of our algorithm to evaluate deployment mappings (cf. [CHH10]). Our aim is not to find the globally optimal solution. The rationale for this is simple; by the time the optimal deployment mapping could be applied, it is likely to be suboptimal due to dynamics of the system. Rather, we aim to find a *feasible mapping*, meaning that it satisfies the requirements for the deployment of the service, e.g. in terms of redundancy and load-balancing. These requirements are specified as a set of rules, denoted Φ. Thus, our objective function becomes *min* $F()$ subject to Φ. Moreover, our algorithm can continue to optimize even though an appropriate mapping has been found and deployed into the network. Once a (significantly) better mapping is found, reconfiguration can take place.

Next we define the dependability rules, Φ, that constrain the minimization problem, but first we define two mapping functions. These rules and functions apply to service k.

DEFINITION 1 *Let* $f_{j,d} : R^k_{ij} \rightarrow d$ *be the mapping of replica* R^k_{ij} *to domain* $d \in \mathscr{D}$.

DEFINITION 2 *Let $g_j : R^k_{ij} \rightarrow n$ be the mapping of replica R^k_{ij} to node $n \in \mathscr{N}$.*

Rule ϕ_1 requires replicas to be dispersed over as many domains as possible, aimed to improve service availability despite potential network partitions[1]. Specifically, replicas of component C^k_i shall be placed in different domains, until all domains are used. If there are more replicas than domains, i.e. $|C^k_i| > |\mathscr{D}|$, at least one replica shall be placed in each domain. The second rule, ϕ_2, prohibits two replicas of C^k_i to be placed on the same node, n.

RULE 1 $\phi_1 : \forall d \in \mathscr{D}, \forall R^k_{ij} \in C^k_i : f_{j,d} \neq f_{u,d} \;\Leftrightarrow (j \neq u) \wedge |C^k_i| < |\mathscr{D}|$

RULE 2 $\phi_2 : \forall R^k_{ij} \in C^k_i : g_j \neq g_u \;\Leftrightarrow (j \neq u)$

Combining these rules gives us the desired set of dependability rules, $\Phi = \phi_1 \wedge \phi_2$. In order to adhere to ϕ_1, the ant gathers data about domains utilized for mapping replicas; hence, let D_r denote the set of domains used in iteration r. The ant also collects information about replicas mapped to various nodes. Thus, we introduce $m_{n,r} \subseteq S_k$ as the set of service k replicas mapped to node n in iteration r. Moreover, let $M_r = \{m_{n,r}\}_{\forall n \in H_r}$ be the deployment mapping set at iteration r for all visited nodes. Finally, ants also collect load-level samples, $l_{n,r}$, from every node $n \in H_r$ visited in iteration r; these samples are added to the *load list*, L_r, indexed by the node identifier, n.

The load-levels observed by an ant are a result of many concurrently executing ant species reserving resources for their respective services. For simplicity, all replicas have the same node-local execution cost, w, whereas communication costs are ignored. An ant during its visit to node n reserves processing resources for the replicas, if any, that it has chosen to map to n. Mappings made at n during iteration r are stored in $m_{n,r}$, thus, resources of size $|m_{n,r}| \cdot w$ are reserved during a visit, assuming identical cost for all replicas. With this notational framework in place, we are now ready to introduce the cost function used by the deployment logic.

First, we define a list, NC_x, that can carry an element for each node visited by the ant and which elements account for specific execution costs imposed on those nodes. Elements of the list are calculated two different ways ($x = 1$ or 2), using the observations on the services executed in parallel (L_r), and the mappings of replicas made by the ant itself (M_r).

$$NC_x[n] = (\sum_{i=0}^{\vartheta_x} \frac{1}{\Theta_x + 1 - i})^2 \tag{1}$$

Parametrization of the list is done by changing the upper bound of the summation, ϑ_x and the constant in the denominator, Θ_x. Accordingly, ϑ_x and Θ_x are defined as follows.

$$\vartheta_x = \begin{cases} |m_{n,r}| \cdot w, & x = 1 \\ |m_{n,r}| \cdot w + L_r(n), & x = 2 \end{cases} \tag{2}$$

[1]We assume network partitions are more likely to occur between domain boundaries.

The constant, Θ_x represents the overall execution load of one service or all services. In other words, Θ_1 is the total processing resource demand of the service deployed by the ant, whereas Θ_2 represents the overall joint load of the service being deployed and the load of replicas executed in parallel.

$$\Theta_x = \begin{cases} \sum_{\forall n \in H_r} |m_{n,r}| \cdot w, & x = 1 \\ \sum_{\forall n \in H_r} (|m_{n,r}| \cdot w + L_r(n)), & x = 2 \end{cases} \tag{3}$$

Importantly, the equations only have to be applied on the subset of nodes an ant has actually visited (H_r), which is beneficial for scalability as there is no need for exploring the total amount of available nodes. Finally, to build a cost function that satisfies our requirements with regard to Φ, while maintaining load-balancing, we formulate $F()$ using a combination of terms, as shown in (4).

$$F(D_r, M_r, L_r) = \frac{1}{|D_r|} \cdot \sum_{\forall n \in H_r} NC_1(n) \cdot \sum_{\forall n \in H_r} NC_2(n) \tag{4}$$

Thus, we use (1) for load-balancing, i.e. to distribute replicas to the largest extent possible. The three terms correspond to our goals in the optimization process. The first reciprocal term caters for ϕ_1. Applying (1) solely on the replicas of the service the ant species is responsible for ($x = 1$) penalizes violation of ϕ_2, i.e. favors a mapping where replicas are not collocated, but distributed evenly. Lastly, the standard application of (1), $x = 2$, balances the load taking into account the presence of other services during the deployment mapping. A more detailed introduction to the application of the load-balancing term can be found in [CHH08] and [CHH10]. The next section describes how the cost function plays a role in driving the optimization using the CEAS.

3. The Cross-Entropy Ant System

We build our algorithm around the CEAS to obtain optimal deployment mappings with high confidence. CEAS can be considered as a subclass of ACO algorithms [DMC96], which have been proven to be able to find the optimum at least once with probability close to one; once the optimum has been found, convergence is assured within a finite number of iterations. The key idea is to have many ants, search iteratively for a solution according to a cost function defined according to problem constraints. Each iteration is divided into two phases. Ants conduct *forward search* until all the replicas are mapped successfully. After that, the solution is evaluated using the cost function, the ants continue with *backtracking* leaving *pheromone* markings at nodes. This resembles real-world ants foraging for food. The pheromone values are proportional to the solution quality determined by the cost function. These pheromone markings are distributed to nodes in the network, and are used during forward search to select replica sets for deployment mapping, gradually approaching the lowest cost solution. In forward search, a certain proportion of ants do a random *exploration* of the state space, ignoring the pheromone trails. Exploration reduces the occurrence of premature convergence leading to sub-optimal solutions. The CEAS uses the *Cross-Entropy (CE) method* introduced by Rubinstein [Rub99] to evaluate solutions and update the pheromones. The CE method is applied to gradually change a probability matrix $\mathbf{p}_r$ according to the cost of the mappings with the objective of minimizing the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. The method itself has been successfully applied in different fields of network and path management, for examples and an intuitive introduction we refer to [HHW08].

In our algorithm, the CEAS is applied to obtain an appropriate deployment mapping, $\mathscr{M} : C_i^k \to \mathscr{N}$, of the replicas ($C_i^k$) of service S_k onto a set of nodes. A deployment mapping is evaluated by applying the cost function as $F(M_r)$. In the following, let $\tau_{mn,r}$ be the pheromone value corresponding to, $m_{n,r}$, the set of replicas mapped to node n in iteration r. Various pheromone encoding schemes are discussed in Sec. 4.2.

To select a set of replicas to map to a given node, ants use the so-called *random proportional rule* matrix, $\mathbf{p}_r = \{p_{mn,r}\}$ presented below. Similarly, *explorer ants* select a set of replicas with uniform probability $1/|C_i^k|$, where $|C_i^k|$ is the number of replicas to be deployed.

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in M_{n,r}} \tau_{ln,r}} \tag{5}$$

A parameter γ_r denoted the *temperature*, controls the update of the pheromone values and is chosen to minimize the performance function, which has the following form

$$H(F(M_r), \gamma_r) = e^{-F(M_r)/\gamma_r} \tag{6}$$

and is applied to all r samples. The expected overall performance satisfies the equation

$$h(p_{mn,r}, \gamma_r) = E_{\mathbf{p}_{r-1}}(H(F(M_r), \gamma_r)) \geq \rho \tag{7}$$

$E_{\mathbf{p}_{r-1}}(X)$ is the expected value of X s.t. the rules in $\mathbf{p}_{r-1}$, and ρ is a parameter (denoted *search focus*) close to 0 (in our examples 0.01). Finally, a new updated set of rules, $\mathbf{p}_r$, is determined by minimizing the cross entropy between $\mathbf{p}_{r-1}$ and $\mathbf{p}_r$ with respect to γ_r and $H(F(M_r), \gamma_t)$. Minimized cross entropy is achieved by applying the random proportional rule in (5) for $\forall_{mn}$ with

$$\tau_{mn,r} = \sum_{k=1}^{r} I(l \in M_{n,r}) \beta^{\sum_{j=k+1}^{r} I(j \in M_k)} H(F(M_k), \gamma_r) \tag{8}$$

where $I(x) = 1$ if x is true, 0 otherwise. See [Rub99] for further details and proof.

As we target a distributed algorithm that does not rely on centralized tables or control, neither on batches of synchronized iterations, the cost values obtained by applying Eq. (4) are calculated *immediately* after each sample, i.e. in each iteration r. Then, an auto-regressive performance function, $h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1-\beta) H(F(\mathrm{M}_r), \gamma_r)$ is applied, where $\beta \in <0, 1>$ is a *memory factor* that gives weights to the output of the performance function. The performance function smoothes variations in the cost function and helps avoiding undesirable rapid changes in the deployment mappings.

The *temperature* required for the CEAS, e.g. in Eq. 6, is determined by minimizing it subject to $h(\gamma) \geq \rho$ (cf. [HW01])

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(M_i), \gamma) = \rho\} \tag{9}$$

However, (9) is a complicated function that is storage and processing intensive since all observations up to the current sample, i.e. the entire mapping cost history $F(M_r) = \{F(M_1), \ldots, F(M_r)\}$ must be stored, and weights for all observations have to be recalculated. This would be an impractical burden to on-line execution of the logic. Instead, given that β is close to 1, it is assumed that changes in γ_r are relatively small in subsequent iterations, which enables a first order Taylor expansion of (9), and a second order Taylor expansion of (8), see [HW01], thus saving memory and processing power.

4. Ant Species Mapping Replicas

In this section we present our deployment algorithm, how we apply the CEAS method, and three different ways of encoding replica mappings into pheromone values.

4.1 Swarm-based Component Deployment

Our algorithm has successfully been applied for obtaining component mappings that satisfy non-functional requirements. In addition, the algorithm's capability to adapt to changing network conditions, for example caused by node-failures, has been investigated, cf. [CHH08]. However, from a dependability point of view it is interesting to equip the logic with the capability to adapt to dynamicity of domains, i.e. splitting/merging of domains. To also cater for domain splits and merges we propose to initiate an ant-nest in multiple nodes belonging to separate domains. These ant-nests will emit ants corresponding to the same set of services, this however, will not result in flooding the network with ants as the rate of emission in a stable network can be divided equally between the nests. Besides, ants emitted from different nests but optimizing mappings for the same service will update the same pheromone tables in the nodes they visit during their search for a solution. The concept of multiple nests has been introduced in [CHH10].

Algorithm 1 shows the code of a single ant-nest that sends out an ant in every iteration. The idea is that when a coherent network suffers a split, there shall be at least one nest in each region after the split event that will maintain a pheromone database in each region. (By a region we denote a set of nodes partitioned into one or more domains.)

To ease convergence of the mappings made by the ants the nests are allowed to bind one replica at a time if some condition applies. Here we check rules ϕ_1 and ϕ_2 against the mapping obtained in the current iteration, M_r. Replica bindings are indicated in the service specification that is derived from the model of the service.
After a replica has been bound to a specific host ants in subsequent iterations will not try to find a new mapping for it, instead these bound mappings are maintained and the search is conducted for the remaining replicas only. Importantly however, bound

Algorithm 1 Code for $Nest_k$ corresponding to service S_l at any node $n \in \mathcal{N}$

```
1: Initialization:
2:     $r \leftarrow 0$                                          {Number of iterations}
3:     $\gamma_r \leftarrow 0$                                          {Temperature}
4: while $r < R$                                         {Stopping criteria}
5:     $M_r \leftarrow antAlgo(r,k)$                           {Emit new ant, obtain $M_r$}
6:     $update(availableDomains)$            {Check the number of available domains}
7:     if $splitDetected() \vee mergeDetected()$
8:         $release(S_l)$          {Delete existing bindings for all replicas $c_i \in C_i^l$}
9:     if $\phi_1(M_r, availableDomains) \wedge \phi_2(M_r)$
10:        $bind1(M_r)$             {Bind one of the still unbound replicas in $C_i^l$}
11:    $r \leftarrow r+1$                                   {Increment iteration counter}
```

replicas are also taken into account when the cost of the total mapping is evaluated by the ant. When a split or a merge event occurs these soft-bindings are flushed by the ant nest and, for example in case of a merge, two nests being in the same region can start to cooperate and share bindings and pheromone tables again.

Here it is important to clearly distinguish between the notions of replica mapping, binding and deployment. We use the term mapping during the optimization process, where our algorithm is constantly optimizing an ordering of replicas of a service to underlying execution hosts, but only internally to the algorithm itself. When a replica is bound to a host it means that from that point the algorithm does not change the mapping between that replica and a host. By deployment however, we refer to the actual physical placement of a software component replica to a node, which is triggered after the mappings obtained by our algorithm have converged to a satisfactory solution. The latter property ensures that there is no undesirable fluctuation in the migration of replicas using our method. In Algorithm 2 we present the steps executed by the ants emitted from a nest.

Each species of ants retrieves and updates the temperature used in the CEAS method from the nest where they are emitted from. First, an ant visits the nodes, if any, that already have a bound replica mapped to maintain these mappings, which will be taken into account when the cost of the total mapping is evaluated. The pheromones corresponding to these bound mappings will also be updated during *backtracking*. Besides, ants allocate processing power corresponding to the execution costs of the bound replicas, derived from the service specification. After maintenance the ants jump over to nodes selected in a guided random manner and attempt to map some replicas to the node they reside in. This selection of the next node to visit, in contrast to e.g. ant-based routing algorithms, is independent from the pheromone markings laid by the ants. The selection of replica mappings in each node, however, is influenced by the pheromones.

Here, we distinguish between *explorer* and *normal* ants, where the former selects a set of replicas to map randomly and the latter uses the pheromone table at the current node. In case of a *normal* ant the selection process varies depending on the form of the pheromone tables (cf. Sec. 4.2). After some variables carried along by the ant (M_r, D_r, C_i^k) are updated a sample of the sum of execution load on the current node is taken by the ant. This replica load reservation mechanism is intended to function as an indirect way of communication between species executed in parallel. At the end of the *forward search* phase, when the ant has managed to map all the replicas of the service, the mapping is evaluated using the cost function and the temperature is recalculated using the obtained cost value. The last part in the lifetime of a single ant is the *backtracking* phase, during which the ant revisits the nodes that have been used for the mapping of the service and updates the pheromone database.

Algorithm 2 Ant code for mapping of replicas $C_i^l \in S_l \subset \mathscr{S}$ from $Nest_k$

```
1:  Initialization:
2:      $H_r \leftarrow \emptyset$                                  {Hop-list; insertion-ordered set}
3:      $M_r \leftarrow \emptyset$                                  {Deployment mapping set}
4:      $D_r \leftarrow \emptyset$                                  {Set of utilized domains}
5:      $L_r \leftarrow \emptyset$                                  {Set of load samples}
6:  function $antAlgo(r,k)$
7:      $\gamma_r \leftarrow Nest_k.getTemperature()$              {Read the current temperature}
8:      foreach $c_i \in C_i^l$                                    {Maintain bound replica mappings}
9:          if $c_i.bound()$
10:             $n \leftarrow c_i.boundTo()$                       {Jump to the node where this comp. is bound}
11:             $n.reallocProcLoad(S_k, w)$                        {Allocate processing power needed by comp.}
12:             $l_{n,r} \leftarrow n.getEstProcLoad()$            {Get the estimated processing load at node n}
13:             $L_r \leftarrow L_r \cup \{l_{n,r}\}$              {Add to the list of samples}
14:     while $C_i^l \neq \emptyset$                               {More replicas to map}
15:         $n \leftarrow selectNextNode()$                        {Select next node to visit}
16:         if explorerAnt
17:             $m_{n,r} \leftarrow random(\subseteq C_i^l)$       {Explorer ant; randomly select a set of replicas}
18:         else
19:             $m_{n,r} \leftarrow rndProp(\subseteq C_i^l)$      {Normal ant; select replicas according to Eq. (5)}
20:         if $\{m_{n,r}\} \neq \emptyset, n \in d_k$             {At least one replica mapped to this domain}
21:             $D_r \leftarrow D_r \cup d_k$                      {Update the set of domains utilized}
22:         $M_r \leftarrow M_r \cup \{m_{n,r}\}$                  {Update the ant's deployment mapping set}
23:         $C_i^l \leftarrow C_i^l - \{m_{n,r}\}$                 {Update the set of replicas to be deployed}
24:         $l_{n,r} \leftarrow n.getEstProcLoad()$                {Get the estimated processing load at node n}
25:         $L_r \leftarrow L_r \cup \{l_{n,r}\}$                  {Add to the list of samples}
26:     $cost \leftarrow F(M_r, D_r, L_r)$                         {Calculate the cost of this given mapping, using Eq. (4)}
27:     $\gamma_r \leftarrow updateTemp(cost)$                     {Given cost, recalculate temperature according to Eq. (9)}
28:     foreach $n \in H_r.reverse()$                              {Backtrack along the hop-list}
29:         $n.updatePheromone(m_{n,r}, \gamma_r)$                 {Update pheromone table in n, Eq. (8)}
30:     $Nest_k.setTemperature(\gamma_r)$                          {Update the temperature at $Nest_k$}
```

The gain in using a guided but random hop-selection instead of a pure random walk lies in that with the proper guidance the frequency of finding an efficient mapping is greater. The idea is that at first the next node is selected from a domain that has not yet been utilized until all visible domains are covered, leading to better satisfaction of ϕ_1. Then the next hop selection continues with drawing destinations from the set of nodes not yet used in the mapping by checking with the variable M_r, before reverting to totally random drawing. The guided hopping strategy for the selection of a next node to visit is summarized in Algorithm 3.

Algorithm 3 Procedure to select the next hop for an ant

```
 1: function $selectNextNode()$                      {Guided random hop}
 2:   if $H_r = \mathbf{N}$                           {All nodes visited}
 3:     $n \leftarrow random(\mathbf{N})$             {Select candidate node at random}
 4:   else
 5:     if $D_r = \mathbf{D}$                         {All available domains utilized}
 6:       $n \leftarrow random(N \setminus M_r)$      {Select a node that has not been used yet}
 7:     else
 8:       $d_i \leftarrow random(\mathbf{D} \setminus D_r)$   {Select a domain not yet used}
 9:       $n \leftarrow random(d_i)$                  {Select a node within this domain}
10:   $H_r \leftarrow H_r \cup \{n\}$                 {Add node to the hop-list}
11:   return $n$
```

4.2 Encoding Sets of Replicas into Pheromone Entries

Generally, pheromone entries can be viewed as a distributed database located in the nodes available in the network considered for deployment. This distributed database has to be built so that it is able to describe arbitrary combinations of replicas of a given service component. At the same time the size of this database is crucial for obtaining better scalability for our approach. The reasons are twofold. The first reason is related to memory consumption as each participating node has to cater for a pheromone database for each service being deployed. Thus, memory consumption grows with the database size (depending on the encoding) and with the number of parallel services, where we can influence the former. Second, as we can see in the algorithm description in Sec. 4.1, an individual ant agent has to browse through the pheromone entries during its visit at a node, so clearly, a more compact encoding helps speeding up execution of the tasks an ant has to perform. The different encodings we proposed are shown in Table 1.

Table 1: Three pheromone encodings for a service with $|C_i^k|$ replicas

Encoding	DB size in a node	Encoding example w/ $|C_i^k| = 4$
bitstring	$2^{|C_i^k|}$	$[0000]b \ldots [1111]b$
per comp.	$2 \cdot |C_i^k|$	[0/1]; [0/1]; [0/1]; [0/1]
# replicas	$|C_i^k| + 1$	$[0] \ldots [4]$

The *bitstring* encoding is the largest as it has a single value for all possible combinations of replica mappings in every node, which results in prohibitively large memory need. For example, in case of 20 replicas per service this encoding leads to 2^{20} pheromone values, which by using 4 byte long floating point numbers would require 4 MB of memory for each of such services at every node. To tackle this problem we might reduce the pheromone table size by applying more simple bookkeeping taking into account solely the number of replicas mapped to a given node (# *replicas*). This results in the most compact pheromone database, however it comes with a drawback that it can only be applied if there is no need to distinguish between replicas in the service specification (for example considering replication and dependability aspects only). As a trade-off we developed a third encoding (*per comp.*) that results in no information loss and still linear growth of the pheromone database. *per comp.* uses one distinct pheromone entry for every replica instance indicating whether or not to deploy a replica at a given node. The drawback is that an ant arriving at a node has to decide on the deployment mapping of each replica, one-by-one reading the multiple pheromone entries. Nevertheless, a reduction in the database structure size is necessary for scaling the algorithm up to larger amounts of nodes and replicas. How the various encodings perform will be demonstrated with an example in Sec. 5.

5. Simulation Results

To evaluate the deployment mapping logic proposed above we start with an example where 10 services ($S_1 \ldots S_{10}$) are being deployed simultaneously, that means 10 independent species are released. Besides, we apply 20 ant nests to look at a simple split/merge scenario where 1 nest for every service remains in each region after the split. Each service has a redundancy level as shown in Table 2. The simulation of the logic's behavior is conducted in a custom built discrete event simulator.

Table 2: Service instances in the example

Service	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
# replicas	2	3	4	5	6	7	8	9	10	11

Mapping of the services is conducted in a network of 11 interconnected hosts, where we assume full mesh connectivity and do not consider the underlying network layer. The 11 nodes are partitioned into 5 domains as depicted in Fig. 2.

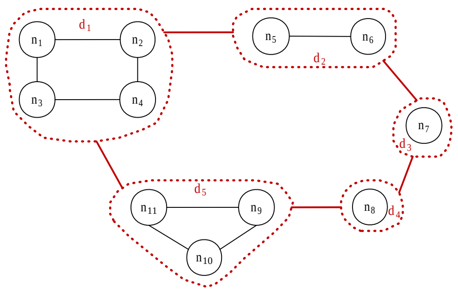
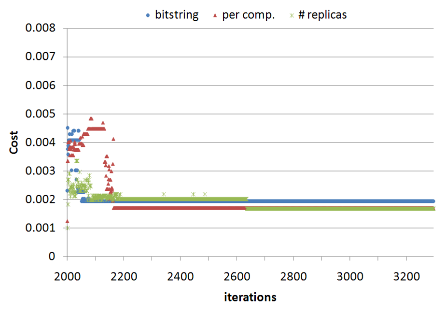


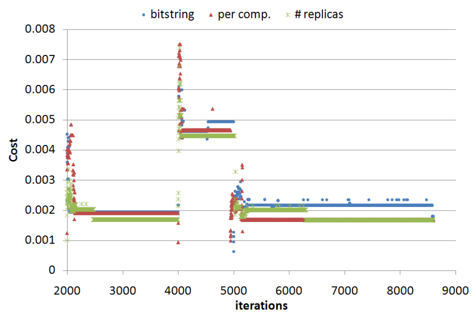
Figure 2: Test network of hosts clustered into 5 domains

In this setting we conducted simulations with all three pheromone encodings. To test our concept of tackling domain splitting we have used a basic setting where domain d_1 containing 4 nodes has been split from the rest of the domains and later the two regions merged again. We then compared the resulting deployment mappings with the mappings obtained by executing our logic with no splitting. To demonstrate how the cost evaluation works in the optimization process the evolution of the cost output is displayed in case of service S_{10} in Fig. 3 with the three pheromone encodings introduced in Sec. 4.2. Fig. 3a shows how the optimal mappings are found and kept maintained iteration by iteration. The experiment is repeated with the introduction of the splitting of d_1 after 4000 iterations, the evolution of mapping costs is shown in Fig. 3b.

An appropriate solution is found almost identically with the three different encodings. However, the *bitstring* encoding converges to a solution with slightly higher overall cost, whereas the lowest cost is obtained first by *per comp.* and somewhat later by *# replicas* too. In Fig. 3 the first 2000 iterations are not shown as the simulations start with 2000 explorer iterations for the sake of comparability. Initially, a random cost figure appears corresponding to exploration that is omitted here. The amount of initial exploration was constrained by the *bitstring* encoding. The more compact encodings would require significantly less iterations, e.g. one tenth of that. In Fig. 3b, where a domain splits at iteration 4000 we can observe how the swarm adapts the mappings to a more expensive configuration after the event has happened. Similarly, as the domains merge the deployment mappings are adapted to utilize a more optimal configuration. The *bitstring* encoding in this test case is unable to find exactly the same mapping and converges to a somewhat more costly solution. *per comp.* is the fastest to obtain the lowest cost mapping followed by the third encoding about 1000 iterations later.



(a) Without split



(b) With split

Figure 3: Mapping costs of S_{10}

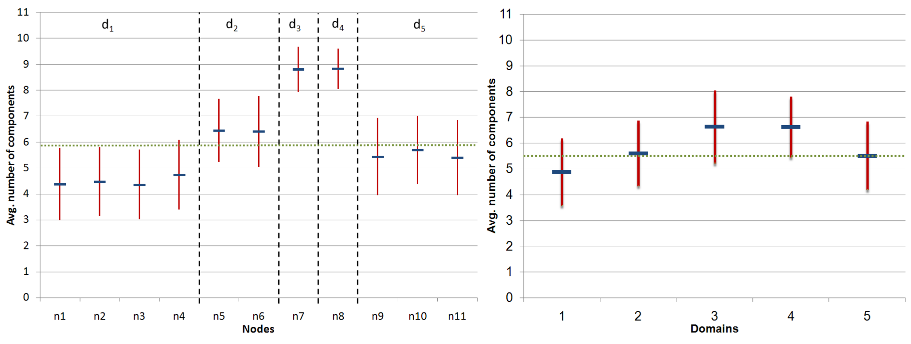
Considering the rules that we formulated regarding the dependability of the deployment mapping (cf. Sec. 2) Table 3 shows the three different pheromone encodings and the percentage of test cases, which succeeded in satisfying the two rules. The results are obtained by executing the algorithm 100 times with different input seeds.

Table 3: Success rate of the three encodings

wo/ split	ϕ_1	ϕ_2	w/ split	ϕ_1	ϕ_2
bitstring	100%	88%	bitstring	100%	87%
per comp.	100%	100%	per comp.	100%	100%
# replicas	100%	100%	# replicas	100%	99%

Our first objective was load-balancing among the nodes participating in the execution of the services, while basic dependability rules are satisfied too. To investigate that aspect we can look at the average number of replicas placed onto the nodes after convergence. Here, we chose the best encoding (cf. Table 3), i.e. *per comp.*. In Fig. 4a the average load placed on the 11 nodes ($n_1 \dots n_{11}$) partitioned into the 5 domains is depicted. A total of 65 replicas constituted the ten service instances giving an average of 5.91 replicas per node; shown as a dotted horizontal line. We observed that the smaller domains, e.g. d_3, d_4, were overloaded compared to the rest due to ϕ_1, but generally replicas were placed quite evenly, showing that cooperation between the species worked.

As a next step towards developing our logic further for larger scales we repeated our experiment with a setting consisting of 50 nodes in 5 domains (containing 20-10-5-5-10 nodes respectively). Naturally, an increased amount of available resources for placement would make the deployment mapping problem actually easier, so we have used larger service specifications too to scale up the problem. Accordingly, the 10 services assigned to ant species were sized as $|C_i^k| = i \cdot 5$ replicas for S_i, where $i = 1 \dots 10$, thus giving a total amount of 275 replicas.



(a) Over 11 nodes

(b) Over 5 domains (50 nodes)

Figure 4: Load-balancing (average number of replicas and deviation per node)

We repeated the experiment 50 times using the selected encoding, *per comp.*, and allowing a maximum amount of 10000 iterations in each run. The resulting average execution load in the 5 available domains is depicted in Fig. 4b, where the average number of replicas per node, using identical domains, would be 5.5 that is shown with a dotted horizontal line. Regarding load-balancing similar effects are observed as in the previous example. We can look at the dependability aspects of the solutions obtained in the 50 runs of the simulation too. As the problem size was significantly larger and the number of allowed iterations was constrained too, collocation is observed in some cases (in Table 4), while rule ϕ_1 is never violated. Violations of ϕ_2 are more frequent in case the number of replicas was close to the number of available nodes (e.g. S_{10}), which makes satisfying ϕ_2 harder when load-balancing has to be performed simultaneously.

Table 4: Collocation within the 10 large services

service	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
collocation (ϕ_2)	0	1	0	3	1	1	0	3	1	13

6. Closing Remarks

Our focus has been on applying swarm intelligence, in particular the CEAS method to manage the deployment of collaborating software components. While developing our distributed approach we targeted a logic that shall not be over-engineered and uses only a few parameters that do not depend on the problem at hand (e.g. to avoid having to adjust parameters and cost functions manually). It is also required to be able to handle certain degrees of dynamics and adaptation to changes in the execution context. These are the reasons that lead us to nature inspired methods and systems that do not have to be altered significantly for every new target system. We have tested the ability of the logic to handle domain splitting and dealing with dependability requirements as well as load-balancing using two example settings and a custom built simulator. We believe that applying CEAS will not only result in a tailored optimization method but, at least on the long run, it will allow the implementation of a prototype of a truly distributed system that will support run-time deployment within software architectures.

Our results are promising and are inline with our efforts to further develop the deployment logic and increase its scalability and adaptability. Furthermore, we plan to experiment with another dimension of dynamicity by introducing run-time component replication that means that the amount of replicas in a service might change at run-time. Moreover, extensive simulations will be conducted to test scalability and convergence of the algorithm and also to evaluate its behavior compared to other relevant optimization methods that support distributed execution. This is a possible direction for future work, however, we advocate that a thorough comparison could in fact be a separate paper in itself as it would require fine tuning of multiple parameters in case of many available methods to be able to look into the scenario at hand with confidence.

References

[CHH08] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In *Proc. of the 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), Turin*. ICST/ACM, September 2008.

[CHH10] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Component deployment using parallel ant-nests. *To appear in Int'l Journal on Autonomous and Adaptive Communications Systems (IJAACS)*, 2010.

[CMHH09] M. J. Csorba, H. Meling, P. E. Heegaard, and P. Herrmann. Foraging for better deployment of replicated service components. In *Proc. of the 9th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), LNCS 5523*, pages 87–101, 2009.

[Dea] J. Dean. Software engineering advice from building large-scale distributed systems. `http://research.google.com/people/jeff/stanford-295-talk.pdf`.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[FB89] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. on Software Engineering*, 15(11):1427–1436, 1989.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HW01] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[JHJ09] K. Joshi, M. Hiltunen, and G. Jung. Performance aware regeneration in virtualized multitier applications. In *Proc. of the Workshop on proactive failure avoidance recovery and maintenance (PFARM)*, 2009.

[KHD08] R. Kusber, S. Haseloff, and K. David. An approach to autonomic deployment decision making. In *Proc. of the 3rd Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5343*, pages 121–132, 2008.

[LLC] Amazon Web Services LLC. Amazon elastic compute cloud. `http://aws.amazon.com/ec2`.

[MG08] H. Meling and J. L. Gilje. A distributed approach to autonomous fault treatment in spread. In *Proc. of the 7th European Dependable Computing Conference (EDCC)*, pages 46–55, 2008.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

PAPER F

Ant system for service deployment in private and public clouds

Mátě J. Csorba and Hein Meling and Poul E. Heegaard

In proceedings of 2nd Workshop on Bio-Inspired Algorithms for Distributed Systems (BADS'10)

Washington, DC, USA, Jun 7-11, 2010

ANT SYSTEM FOR SERVICE DEPLOYMENT IN PRIVATE AND PUBLIC CLOUDS

Máté J. Csorba[1], Hein Meling[2], Poul E. Heegaard[1]

[1]*Department of Telematics,*
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard}@item.ntnu.no

[2]*Department of Electrical Engineering and Computer Science,*
University of Stavanger,
Stavanger, Norway
Hein.Meling@uis.no

Abstract Large-scale computing platforms that serve thousands or even millions of users through the Internet are on a path to become a pervasive technology available to companies of all sizes. However, existing technologies to enable this kind of scaling are based on a hierarchically managed approach that does not scale equally well. Moreover, existing systems are also not equipped to handle the dynamism that may emerge as a result of severe failures or load surges.

In this paper, we conjecture that using self-organizing techniques for system (re)configuration can improve both the scalability properties of such systems as well as their ability to tolerate churn. Specifically, the paper focuses on deployment of virtual machine images onto physical machines that reside in different parts of the network. The objective is to construct balanced and dependable deployment configurations that are resilient. To accomplish this, a method based on a variant of Ant Colony Optimization is used to find efficient deployment mappings for a large number of virtual machine image replicas that are deployed concurrently. The method is completely decentralized; ants communicate indirectly through pheromone tables located in the nodes.

An example scenario is presented and simulation results are obtained for the method.

1. Introduction

Cloud computing infrastructures have in recent years become increasingly important for provisioning services that demand reliability and performance, yet are capable to utilize the computing resources efficiently. A major benefit of cloud infrastructures is their ability to dynamically scale up or down as the demand curve changes. One approach in which such dynamic service delivery can be accomplished is through the use of Virtual Machine (VM) images that can be deployed on demand within the cloud. Such VM images are packaged in a standardized way that allows for dynamic deployment, e.g. the Amazon Machine Image format [LLC]. Moreover, use of a common VM packaging format also enable a computing model where both public and private cloud providers can interoperate. In this context, a *public cloud provider* offers a large infrastructure of compute resources that are provided to (paying) users over the Internet. This is sometimes called Infrastructure-as-a-Service, and Amazon EC2 [LLC] is an example of a public cloud provider. On the other end, we have *private clouds* that offer a more limited scale of resources, typically accessible only to users directly affiliated with the private cloud owner, e.g. a single organization. These organizations may be running Ubuntu's Enterprise Cloud solution [WGB09], which is compatible with Amazon EC2 in packaging format. The intention of the Eucalyptus project, for example, is to support multiple cloud computing interfaces while preserving the back-end infrastructure [NWG+09].

Lack of service management facilities and interoperability between cloud providers have been identified as major obstacles limiting scalability of federated cloud computing environments [RBL+09]. Such environments need a unified interface for dynamically managing VMs forming cloud services. Moreover, a heterogeneous cloud computing architecture must also tackle the placement, migration, and monitoring of VMs across interoperability boundaries [EL09]. In this work however, we rely on the presumed existence of such interoperability and service management facilities, and focus our attention on the *service placement problem*. As such, our approach is independent of the specific flavors of the underlying interoperability and management facilities provided.

In this paper, we examine the effects of a hybrid environment in which services are deployed in either the private cloud, public clouds, or both depending on the present usage pattern. Such a scenario is especially interesting with respect to handling load overshoots that may be caused by dependability and/or performance requirements. For example, as the service usage pattern change, VM instances may be added or removed from the public cloud, while retaining the same number of VM instances within the private cloud. During execution in such a hybrid cloud environment, a plethora of highly dynamic parameters influence the optimal deployment configurations, e.g. due to the influence of concurrent services and varying client load. Ideally, the deployment mappings should minimize and balance resource consumption, yet provide sufficient resources to satisfy the dependability requirements of services. However, Fernandez-Baca [FB89] showed that the general module allocation problem is NP-complete except for certain communication configurations, thus heuristics are required to obtain solutions efficiently.

Our approach is based on a heuristic and decentralized optimization method aimed at *finding suitable mappings* between VM replicas and nodes, in the various clusters of the network, capable of hosting them. The set of mappings selected are constrained in three dimensions ensuring: cloud internal load balancing, cloud global load balancing, and availability of VM replicas in multiple clusters for improved dependability. To accomplish this we use the Cross-Entropy Ant System (CEAS) [HHW08], which is based on Ant Colony Optimization (ACO) [DMC96]. CEAS uses ant-like agents, denoted *ants*, that can move around in the network, identifying potential locations where replicas might be placed.

1.1 Related Work

The notation of regenerating replicas to replace crashed ones was first proposed by Pu [PNP88] in the context of the Eden system. More recent systems [MG08, MMHB08] provide automatic reconfiguration and regeneration of replicas in the context of group communication systems, and Om [YV05] focus on regeneration in a peer-to-peer wide-area storage system. Another recent initiative [EL09] propose similar mechanisms for placement, migration and monitoring of components in the cloud. Such systems provide the underlying mechanisms that are necessary to support service deployment in cloud environments. However, focus is mostly on failure recovery by regeneration of new replicas to improve availability and reliability, and do not try optimize the replica-to-node mappings. Yu and Gibbons [YG09] show theoretically that replica placements of inter-correlated objects can significantly impact system availability if not placed appropriately. Our work is focused on finding suitable (near optimal) replica-to-node mappings that improve both availability and load balancing properties. Albrecht et al. [AOVP08] describe a generic wide-area resource discovery system taking a database-like approach to enable querying for available resources; they use a *centralized* group-finding optimization algorithm to find mappings. Their approach rely on extensive measurement data collection, in some sense not unlike our ants. However in our approach, measurements are not stored centrally in a database or in a DHT. Instead, ants encode such measurement data using a *decentralized* mathematical framework that enable us to heuristically find near optimal solutions rapidly. Many other frameworks have focused on finding optimal placements for virtual machines under a variety of constraints [VAN08, JHJ09]. Maximizing the utility of services via deployment decision making has been investigated in [KHD08]. An algorithm has been devised by the authors that is based on calculating the usefulness of the alternative configurations as weighted sums. Nevertheless, the resulting approach is not computationally effective and serves as a trial to show that deployment decision making is important and necessary to apply. However, these approaches rely on a centralized optimizer that often has to crawl through the entire state-space of decision alternatives. The SmartFrog [Sab06] deployment and management framework from HP Labs describes services as collections of components and applies a distributed engine comprised of daemons running on every node in a network. Collections of components together with their configuration parameters can be activated and managed to deliver the desired services even in large-scale systems. The scale of these systems and the execution framework is close to the environment we envisage for the successful execution of autonomic component-based software services and which we target with our deployment logic. Xu et al., in [XZF$^+$07], applies a novel approach in similar server environments for configuration management based on fuzzy learning and targeting efficient resource utilization. Others, e.g. the authors of [HJ06], have turned to biologically-inspired resource allocation algorithms to solve service distribution problems. We as well have chosen to follow the path of bio-inspired algorithms.

Our approach is self-organizing and uses a fully decentralized optimization technique based on the CEAS system [HHW08] which is adaptive to network dynamics and is particularly suited for multi-constrained optimization problems. Our previous work [CMHH09] has focused on finding efficient mappings within relatively small scale clusters; and we experimented with different pheromone encodings for improving scalability in [CMH09]. We have targeted a decentralized solution to avoid the burden of maintaining centralized databases and to eliminate performance and dependability bottlenecks. The trade-off of decentralization and heuristic search is that our approach might not always find the global optimum, especially when dealing with large problems, but it may converge to a near optimal solution. We advocate, however, that satisfactory deployment, e.g. in terms of redundancy and load-balancing, can be achieved using a mapping close to optimal as in dynamic environments the global optimum can be invalidated by the time it would be found and installed.

To further study the efficiency of our approach and cross-validate our results against centralized solutions we are also working on integer programs capable of calculating optimal replica mappings based on a global view of the system. Centralized solutions requiring a global view, in particular integer linear programs (ILPs), have previously been applied to clustering problems, for example in grid file systems [SME09]. Similar techniques can be used to build an ILP for checking how close mappings obtained by our algorithm are to the optimum solution. Preliminary results of this evaluation can be found in [CH10]. In this paper we discuss an extension to our bio-inspired algorithm to consider resource allocation in federated public and private clouds as well as to obtain optimal utilization of resources in case of overshoot scenarios.

In the next section we introduce the system model and notation we use, how the deployment rules and the cost function are defined. The basics of the CEAS are presented in Sec. 3, followed by a description of the algorithm we propose in Sec. 4. An example scenario and corresponding simulation results are shown in Sec. 5. Finally, in Sec. 6 we conclude and touch upon future work.

2. System Model

In this section we introduce the system model and the notation that we use. We also clarify our assumptions, and define dependability constraints and rules related to deployment mapping. Finally, we present a cost function aimed to guide our heuristic search algorithm.

2.1 Model and Notation

We model the system as a large collection, $\mathscr{N}$, of interconnected nodes. $\mathscr{N}$ is partitioned into a set $\mathscr{D}$ of *clusters*, as illustrated by d_1 and d_2 in Figure 1. Clusters are usually formed according to geographical location or otherwise distinct administrative region. We assume that compute clouds are provided by a single administrative entity that can consist of several clusters. Our objective is to find deployment mappings for this environment for a set of *services*, $\mathscr{S} = \{S_1, S_2, \ldots\}$. Each service may contain replicated VMs to provide the service with fault tolerance and load-balancing. The deployment mapping for S_k is defined as a set of mappings $\mathscr{M} : S_k \rightarrow \mathscr{N}$. Let V_i^k be the i^{th} VM of S_k, where $S_k = \{V_1^k, \ldots, V_q^k\}$ is the set of VMs constituting service S_k, q being the number of VMs in the service, $|S_k|$. Accordingly, let R_{ij}^k denote the j^{th} replica of V_i^k so that $V_i^k = \{R_{i1}^k, \ldots, R_{ip_i}^k\}$, where $p_i \geq 1$ is the redundancy level of VM V_i^k. Then, for service S_k, the set of VM instance replicas becomes $S_k = \{R_{11}^k, \ldots, R_{1p_1}^k, \ldots, R_{i1}^k, \ldots, R_{ip_i}^k\}$. In the remainder of the paper, we use the terms VM replica and VM instance interchangeably.

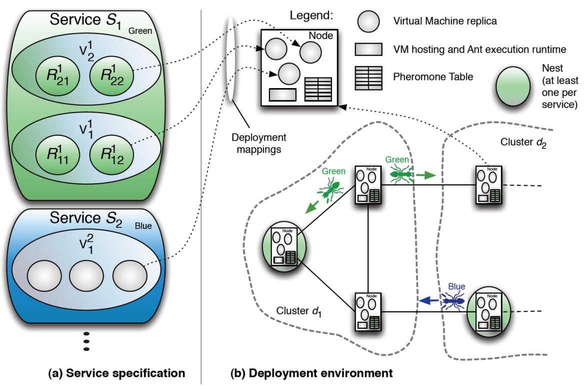


Figure 1: Overview of the deployment environment and service specification.

The objective of our deployment logic is to find suitable *mappings* between VM replicas and nodes, capable of hosting them in the various clusters of the network. Using CEAS, ants move around in the network, trying to identify potential locations where replicas might be placed. Ants have associated state; as such they can be implemented as messages on which our algorithm is executed in every node they visit. For each service, there is one *ant species* responsible for finding the deployment mapping for its associated service. This is illustrated in Figure 1 by the green and blue ant species representing the green and blue service, respectively. It is important to notice that a species of ants corresponds to a service, i.e. a set of VM replicas $S_k = \{R_{11}^k, \ldots\}$. Thus, an increase in the number of VM instances within a service in itself does not lead to impairment of scalability.

As shown in Figure 1, every node contains an *execution runtime*, that supports installing, running and migrating replicas. Furthermore, each node also has a *pheromone table* that will be updated and read regularly by the ants. The purpose of the pheromone table is to assist ants in selecting suitable deployment mappings; this is in contrast to the original ant system proposed by Dorigo et al. [DMC96], in which pheromones are used for ant routing. For every service that has to be deployed, at least one node must also host an ant *nest*. The tasks of a nest are twofold:

1 – to emit ants for its associated service, and

2 – trigger installation of VM replicas onto nodes according to the deployment mappings found.

Installation is triggered once a predefined *convergence criteria* is reached, e.g. after a certain number of iterations of the algorithm, or when a sufficiently low deployment cost level is reached. The actual installation of the VMs of a service is taken care of by the *execution runtime*, details of which are not discussed here, instead we refer to related work (in Sec. 1). Our goal is to build a core logic for optimizing the deployment mappings and, at the same time, enable compatibility with existing frameworks and interfaces for deployment in the clouds.

An iteration, r, of the algorithm is defined as one round-trip trajectory of the ant. During an iteration r, the ant builds and carries along a *hop list*, H_r, keeping track of the visited nodes. The nest can also be replicated for fault tolerance, thereby emitting ants for the same service from multiple nodes. Synchronizing these nests is not necessary, however, only one designated nest is allowed to trigger physical placement of VMs. In Figure 1, the green service has two nest replicas.

2.2 Mapping Rules

The CEAS approach is a heuristic optimization method, and as such our target is not to find the globally optimal solution. This is simply because by the time the optimal mapping configuration could be found and installed, it might be suboptimal due to dynamics of the system. Instead, we aim to find a *feasible mapping*, meaning that it satisfies the requirements for the deployment of the service, e.g. in terms of redundancy and load-balancing. Below we will define a set of rules, denoted Φ, to encapsulate these requirements. One of the key functions in CEAS is the use of a *cost function*, denoted $F()$, that evaluates the quality of a mapping M_r found in iteration r of the algorithm. Thus, the objective of the algorithm is to minimize the cost of the mapping $F(M_r)$ subject to Φ. It should be noted that the algorithm continues to optimize the mapping after an appropriate mapping has been found and applied in the network, once a (significantly) better mapping is found, reconfiguration can take place. The dependability rules that constrain the minimization are defined using the two mapping functions, $f_{j,d}$ and g_j, below that apply to service k.

DEFINITION 1 *Let $f_{R,d} : R \rightarrow d$ be the mapping of replica R to cluster $d \in \mathscr{D}$.*

DEFINITION 2 *Let $g_{R,n} : R \rightarrow n$ be the mapping of replica R to node $n \in \mathscr{N}$.*

Using the two mapping functions defined above we now specify two dependability rules. The first rule, ϕ_1 below, requires replicas to be dispersed over as many clusters as possible, aimed to improve service availability despite potential network partitions between the clusters. Specifically, replicas of VM V_i^k are mapped to different clusters, until all clusters are present in the mapping or all replicas have been mapped to distinct clusters. If the redundancy level of the VM is greater than the number of available clusters in the network, i.e. $|V_i^k| > |\mathscr{D}|$, at least one VM replica is placed in each cluster. Hence, when $j = u$, replicas of V_i^k may be mapped to the same cluster. The second rule, ϕ_2, prohibits two replicas of V_i^k to be placed on the same node, n.

RULE 1 $\phi_1 : f_{R_j,d} \neq f_{R_u,d} \Leftrightarrow (R_j \neq R_u) : \forall d \in \mathscr{D}, \forall R \in V_i^k, \wedge |V_i^k| < |\mathscr{D}|$

RULE 2 $\phi_2 : g_{R_j,n} \neq g_{R_u,n} \Leftrightarrow (j \neq u) : \forall R \in V_i^k$

Combining the rules above we obtain a dependability constraint set for services $\Phi = \phi_1 \wedge \phi_2$. In order to adhere to ϕ_1, the ant gathers data about the clusters utilized for mapping VM replicas; hence, the set of clusters used in iteration r is denoted by D_r. Similarly, the set of replicas from service k mapped to node n in iteration r is denoted by $m_{n,r} \subseteq S_k$. Thus, the ant builds a deployment mapping set $M_r = \{m_{n,r}\}_{\forall n \in H_r}$ for all visited nodes. Finally, ants also collect load-level samples, denoted $l_{n,r}$, from every node $n \in H_r$ visited. The ant uses a *load list*, L_r to carry along all the samples. Load-levels observed by the ant, at the nodes that it visits, are a result of many concurrently executing ant species reserving resources for their respective VM instances. Two different possibilities of implementing this reservation mechanism that serves as a means of indirect communication between ant species have been explored in [CHH08a] and [CMH09], here we omit the description of them.

Each VM replica, R_{ij}^k, of a service k has a node-local execution cost (weight of the replica), denoted by $w^k = \{w_{ij}^k\}$, $i = 1 \dots q, j = 1 \dots p_i$. This cost is used when ants allocate resources for their corresponding services. Note that, to keep the model simple initially, we consider only identical VM replicas ($w^k = w, \forall i, j, k$). However, in future work we will extend the model to cater for more detailed service models that contain information on individual execution costs for the VM replicas and also communication costs for the communication links between VMs of a service. We have already experimented with these types of costs in previous work in the field of software component deployment [CHH08b, CHH08a].

2.3 Cost Function

In what follows, we will define some equations that will be used to define the cost function. Let C_x be a list of values, one for each node visited by an ant. Each value refers to the execution load of the corresponding node.

$$C_x[n] = (\sum_{i=0}^{\vartheta_x(n)} \frac{1}{\Theta_x + 1 - i})^2 \tag{1}$$

The algorithm uses two versions of Eq. (1), depending on the parameter $x \in \{0,1\}$. For $x = 1$, load-level observations, L_r, are used, accounting for all concurrently executing services on the respective nodes. When $x = 0$, the mappings, M_r, made by the ant itself are used, only taking into account the load of those VM instances that are part of the service. The two different usages differ in the upper-bound of the summation and the constant in the denominator, ϑ_x and Θ_x respectively. They are presented next in Eq. (2) and (3).

$$\vartheta_x(n) = |m_{n,r}| \cdot w + x \cdot L_r(n) \quad \text{for } x \in \{0,1\} \tag{2}$$

$$\Theta_x = \sum_{\forall n \in H_r} \vartheta_x(n) \quad \text{for } x \in \{0,1\} \tag{3}$$

Θ_x is a constant representing the overall execution load of one service or all services (depending on the parameter x). More specifically, Θ_0 is the total processing resource demand of the service deployed by the ant, whereas $\Theta_1 = \Theta_0 + \mathscr{L}$, where $\mathscr{L}$ represents the additional load of replicas of *other* concurrently executing services. For $\mathscr{L}$, we account only for those instances that are mapped to the nodes visited by the ant, and as such have reserved processing power for themselves. Note that in (3), Θ_x is applied only for the subset of nodes that the ant has visited, H_r. This is favorable for the scalability of the algorithm, since it does not have to explore the entire network $\mathscr{N}$ exhaustively.

With the notational framework in place we are ready to introduce the cost function used to evaluate deployment mappings obtained with CEAS. To take into account the requirements of load-balancing and dependability (according to Φ) when obtaining VM replica mappings the following cost function is defined.

$$F(D_r, M_r, L_r) = \frac{1}{|D_r|} \cdot \sum_{\forall n \in H_r} C_0(n) \cdot \sum_{\forall n \in H_r} C_1(n) \tag{4}$$

Note that, we use (1) to favor globally balanced mappings, i.e. to distribute VM instance load on the network as evenly as possible. In (4) the first term corresponds to enforcing ϕ_1. The second term, C_0 applies solely to the VM replicas of the service the ant is responsible for, thus penalizing the violation of ϕ_2. The last term, C_1, is used for load-balancing and, as such, it takes into account load imposed on nodes by the other services in the network.

Next, we discuss how the CEAS uses the cost function to guide the ants in finding an optimal deployment mapping and we present the definition of pheromone values.

3. Cross Entropy Ant System for Replica Deployment

This section describes the basics of the CEAS method necessary for presenting our deployment algorithm.

The core of our deployment logic is built around the CEAS method [HHW08], which can be considered a subclass of ACO algorithms [DMC96]. ACO systems have proven to be able to find the optimum solution to a problem at least once with a probability close to one. Once the optimum has been found, convergence is assured in a finite number of iterations. The logic employs ants searching iteratively for a solution. Solutions found by ants are evaluated using a predefined cost function ($F(M_r)$) that takes into account the constraints of the problem. Every iteration consists of a round-trip by the ant and has two distinct phases. In the first phase, the ant conducts a *forward search* and tries to find a mapping for all VMs in the service it is responsible for. Once a complete mapping has been found, the suggested solution is evaluated using the cost function. In the second phase of the lifetime of an ant, called *backtracking*, ants deposit *pheromone* markings at each node they have visited, much like it is done in the real world when ants forage for food. The key idea is that these pheromone values are proportional to the quality of the solution, which was determined by the cost function. The optimum is then approached gradually by using the pheromone tables during forward search for selecting VM instance mappings in nodes. Note that, ants have two modes of operation denoted *explorer ants* and *normal ants*. Normal ants behave as described above, using pheromone tables during forward search. On the other hand, explorer ants ignore pheromone markings during forward search; instead they do a random exploration of the search space. The ratio of normal vs explorer ants is configurable; typically 5-10 % are dedicated as explorer ants. The concept of explorers is used two ways, first to detect changes or better opportunities in the environment, and second, to reduce the occurrence of premature convergence leading to sub-optimal solutions.

A cornerstone in CEAS is the *Cross-Entropy* (CE) method proposed by Rubinstein [Rub99]. In CEAS, the CE method is used both to evaluate solutions and for updating pheromone values. Specifically a probability matrix, $\mathbf{p}_r$, is modified gradually according to the cost returned by the cost function $F()$. The objective of applying the CE method is to minimize the cross entropy between consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. For an introduction and other example applications, see [HHW08].

Let $\tau_{mn,r}$ denote the pheromone value. This value is essentially an encoding of the VM instance mapping $m_{n,r}$ at node n in iteration r. Hence, the pheromone database must be able to store pheromone values that encode the various deployment configurations for various services. Three possible pheromone encoding techniques are discussed and evaluated in [CMHH09]; herein the best encoding is used.

While visiting a node, explorer ants select a set of VM replicas to map to that node with the uniform probability $1/|V_i^k|$, where $|V_i^k|$ is the number of replicas to be deployed. On the other hand, normal ants select VM replicas to deploy based on a *random proportional rule*. This rule is encoded as a probability matrix, $\mathbf{p}_r = \{p_{mn,r}\}$.

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in M_r} \tau_{ln,r}} \tag{5}$$

Updates to the pheromone values are controlled by a *temperature* parameter, γ_r. The temperature is chosen so as to minimize the performance function, $H()$, below.

$$H(F(M_r), \gamma_r) = e^{\frac{-F(M_r)}{\gamma_r}} \tag{6}$$

$H()$ is applied consecutively to all mappings (*samples*) obtained in all iterations. The expectation of the overall performance then satisfies

$$E_{\mathbf{p}_{r-1}}(H(F(M_r), \gamma_r)) \geq \rho \tag{7}$$

$E_{\mathbf{p}_{r-1}}(X)$ is the expected value of X s.t. the rules in $\mathbf{p}_{r-1}$; ρ is a *search focus* parameter close to 0 (we use $\rho = 0.01$). Further, the CE method is used to obtain a new set of rules for the next iteration, $\mathbf{p}_r$, by minimizing the cross entropy between two consecutive rules with respect to γ_r and $H(F(M_r), \gamma_r)$. This is achieved by applying the random proportional rule, Eq. (5), for $\forall_{m_n}$ using the pheromone value

$$\tau_{mn,r} = \sum_{k=1}^{r} I(l \in M_r) \beta^{\sum_{j=k+1}^{r} I(j \in M_k)} H(F(M_k), \gamma_r) \tag{8}$$

where the indicator function $I(x) = 1$ if x is true, or 0 otherwise. For details and proofs of the CE method see [Rub99].

To avoid centralized tables or control, or batches of synchronized iterations, the cost values have to be calculated *immediately* when a new sample (M_r) has been obtained, i.e. in each iteration. To enable this, an auto-regressive version of the performance function is used, as follows

$$h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1-\beta)H(F(\mathrm{M}_r), \gamma_r) \tag{9}$$

where $\beta \in \langle 0, 1\rangle$ is a *memory factor*. β is used to give proper weights to the output of the performance function introduced above. To avoid rapid undesirable changes in the deployment mapping, the performance function will smooth variations in the cost function. The *temperature* parameter, γ_r, is determined by minimization s.t. $h(\gamma) \geq \rho$ (cf. [HHW08])

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r} \sum_{k=1}^{r} \beta^{r-k} H(F(M_k), \gamma) = \rho\} \tag{10}$$

To avoid having to store and process the entire mapping history $F(M_r) = \{F(M_1), \ldots, F(M_r)\}$ for each iteration r, which would make the system impractical, we instead assume that subsequent changes in γ_r are relatively small. Hence, we can apply a first order Taylor expansion on Eq. (10). Similarly, for Eq. (8) a second order Taylor expansion can be applied to save memory and processing power [HHW08].

After discussing the basic methods in CEAS we proceed describing its application to build the deployment mapping algorithm.

4. Virtual Machine Replication in Cloud Computing

We have developed an optimization algorithm using CEAS that aims to find suitable deployment configurations for VM replicas in a cloud computing setting. This algorithm is described in this section.

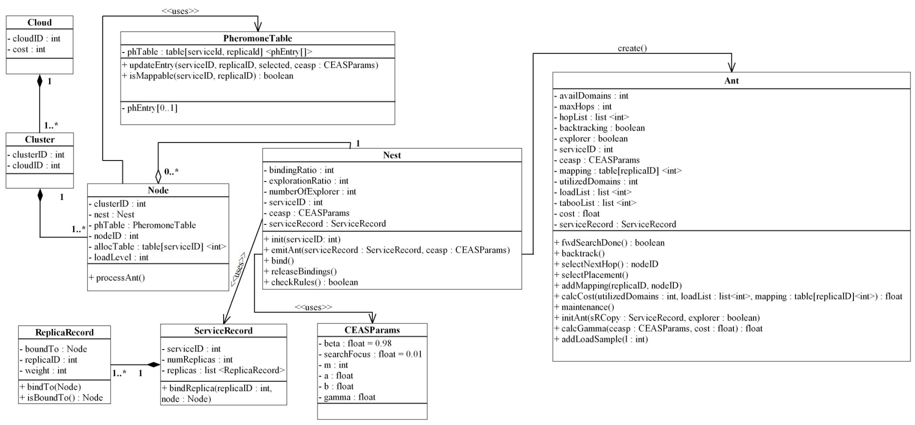


Figure 2: Class Diagram for the CEAS-based Deployment Algorithm

The components of the logic are summarized in the class-diagram in Fig. 2. The mayor component is the *Nest* that is placed on one of the nodes in the network. It is allowed for one species to have more than one nest for additional dependability. Every node must have some additional properties to support the deployment logic, including an instance of the *PheromoneTable* that is used as a container for the distributed information needed by the logic, i.e. $\tau_{mn,r}$. A *Nest* has in addition a set of CEAS related parameters and variables represented by the component *CEASParams*, which information is shared by all the nests of the same species, e.g. β, ρ. In addition, a *Nest* must be able to access information about the service that the species has to deploy, this component is called the *ServiceRecord*, corresponding to S_k. The second mayor building block, *Ant*, of the logic is realizing the ants emitted iteratively by the *Nest*. Most of the intelligence is carried by the *Ant*, represented by methods, as well as some of the variables used during one iteration of the optimization process, e.g. the VM instance mapping set *mapping* corresponding to M_r.

Algorithm 1 Summary of the behavior of $Nest_k$ at any node $n \in \mathcal{N}$

```
1: init();
2: while $r < R$                                   {Stopping criteria}
3:     emitAnt(serviceRecord, ... );
4:     $r \leftarrow r + 1$                         {Increment iteration counter}
```

In Algorithm 1 the behavior of a nest is summarized briefly. Omitting the details, a *Nest* emits (creates and resets) *Ants* sequentially during the optimization process and continues until a stopping criteria, such as a convergence criteria, is fulfilled. Alternatively, nests can continue emitting ants even after the system has stabilized and converged to a given solution, this way providing the capability of adaptation should changes occur in the execution context. Discovery of new, higher utility mappings resulting from context change is supported by explorer ants. Modifying the placement of the services once they are initially deployed can be conditioned by thresholds such as the cost of VM instance migration. Several authors estimate the durations required for migrating operational VMs by conducting experiments. Durations vary, as expected, depending on the hardware context, e.g. bandwidth, and naturally on the VM package size. However, for realistic VM sizes estimates lie typically around 60 to 90 seconds, see [CFH$^+$05], [HON$^+$09], [JHJ09]. These migration costs can be factored in as threshold values to allow changes in the deployment mappings only if the benefit is higher than the costs of migrating.

The number of iterations required for convergence to an initial stable solution depends on the problem size. For the example scenario, introduced in the next section, the maximum number of iterations allowed for the algorithm was 2000 *explorer* ants followed by an additional 3000 (10% *explorer* and 90% *normal*) ants for each species that were executed in parallel. The algorithm was able to find a solution in all simulation runs within this amount of iterations. Increasing the number of nodes in itself does not make the deployment problem more difficult. An increased network size actually allows to algorithm to find lower cost mappings easier due to the larger amount of available resources. The number of services and the amount replicas within the services is what impacts scalability more, as the number of species executed in parallel is proportional to the number of services and the complexity of one species' task increases as the number of replicas increases [CMH09]. The behavior of an *Ant* is briefly presented in Algorithm 2.

Algorithm 2 Summary of the behavior of a single ant

```
1:  Initialization:
2:      $H_r \leftarrow \emptyset$                                   {Hop-list; insertion-ordered set}
3:      $M_r \leftarrow \emptyset$                                   {Deployment mapping set}
4:      $D_r \leftarrow \emptyset$                                   {Set of utilized domains}
5:      $L_r \leftarrow \emptyset$                                   {Set of load samples}
6:  $\gamma_r \leftarrow Nest_k.getTemperature()$                   {Get nest temp.}
7:  while not fwdSearchDone()                                  {More replicas to map}
8:      $n \leftarrow selectNextNode()$                           {Select next node to visit}
9:      if explorerAnt
10:         $m_{n,r} \leftarrow random(\subseteq V_i^l)$             {Randomly select replicas}
11:     else
12:         $m_{n,r} \leftarrow rndProp(\subseteq V_i^l)$            {Select using Eq. (5)}
13:     if $\{m_{n,r}\} \neq \emptyset \wedge n \in d_k$          {Cluster used in mapping}
14:         $D_r \leftarrow D_r \cup d_k$                         {Update utilized clusters}
15:     $M_r \leftarrow M_r \cup \{m_{n,r}\}$
16:     $V_i^l \leftarrow V_i^l - \{m_{n,r}\}$
17:     $L_r \leftarrow L_r \cup \{l_{n,r}\}$                     {Estimated proc. load at node n}
18: $cost \leftarrow calcCost(M_r)$                             {Compute cost of mapping}
19: $\gamma_r \leftarrow calcGamma(cost)$                       {Compute temp., Eq. (10)}
20: foreach $n \in H_r.reverse()$                              {Backtrack along hop-list}
21:     $n.updatePhTable()$                                    {Update pheromones, Eq. (8)}
22: $Nest_k.setTemperature(\gamma_r)$                           {Update temp. at $Nest_k$}
```

Every ant that is emitted receives the appropriate parameters from the nest, such as the *explorer* flag, the description of the service to be deployed, CEAS-related parameters, etc. After initialization the *Ant* proceeds with visiting new nodes during *forward search* until the search is done, i.e. a mapping has been found for all the VMs in the service. In other words, the stopping criteria incorporated into the method $fwdSearchDone()$ checks whether the set V_i^l, which is listing the VM instances not yet mapped by the ant during the current iteration, has become empty. Next nodes are selected via the method *selectNextHop*() that takes into account $cluster$ taboo-lists and node taboo-lists. Taboo-lists are built by the ant during its *forward search* and are updated continuously adding references to clusters and nodes visited to the two lists respectively. The purposes of the two taboo-lists are to cover all available clusters first, to aid satisfying cluster-disjointness, and if the ant has to proceed even after visiting all the clusters then to avoid revisiting the same nodes. Beside the taboo-lists nodes are selected in a random manner. Mappings at each node are selected by the method $selectPlacement()$ that, depending on whether the ant is an *explorer* or not, uses the local $PheromoneTable$ via Eq. 5 or not. Before leaving the node the *Ant* also has to sample load-levels ($L_r(n)$) at the node to achieve load-balancing. When *forward search* is done the *Ant* calculates the cost of the mapping ($F(M_r)$), then recalculates the *temperature* (Eq. 10) and updates the pheromone tables going backward according to its hop-list, H_r, applying Eq. 8. When the ant successfully returns to its nest it is reset and emitted again in the following iteration.

Further improvements in the scalability of CEAS can be made by applying elitism, pheromone sharing and self-tuned packet rate control, additional mechanisms that are described in [HW10]. Next, we present an example scenario we experimented with and the results obtained using our algorithm.

5. Example Scenario and Results

In this section we present an example cloud computing scenario demonstrating the behavior of our deployment logic. The scenario, as in Figure 3, consists of 5 private clouds (Cloud $C,\dots,G$) that are connected to the public Internet, thereby enabling connections to public cloud providers (Cloud A,B). Capacities in the public area can thus be utilized on demand, but are subject to economic costs. Conversely usage of the private clusters is free for a service with a home location in that private cloud. Thus, deploying and hosting a VM instance in a node within one of the clouds implies additional costs $|n_i|$, $\forall n_i \in \mathscr{N}$; these costs are summarized in Table 1.

The tangible meaning of the above partitioning and cost assignment is the following concept. It is natural for any organization to execute all VM instances within their privately owned clusters as long as requirements allow, e.g. replication requirements can be satisfied with the available amount of private clusters, as hosting VMs in the private cloud can be considered free compared to costs of the public clouds. In the example setting, there is a trade-off between a large cloud provider with several clusters and plenty of nodes available for placement, which is more expensive to use than paying for hosting in the smaller cloud offering with less resources.

Table 1: Usage costs for the clouds in the example

	Cloud A	Cloud B	Cloud $C\dots G$
Cost $\lvert n_i \rvert$	10	1	0

The deployment task in the case studied in this paper is then to deploy 125 services in total. Administrators in each private cloud have the task of deploying 25 services using their own available resources and, if needed, using public resources as well. It is not allowed, however, to utilize nodes in a private cloud other than the home location for a service (∞ cost for the neighboring private clouds). In this example every service consists of 5 VM instances, among others for dependability reasons, that have to be deployed, thus giving the task of deploying a total number of 625 VMs to the deployment mapping algorithm.

Deployment of the set of VM instances is done in a network environment partitioned into the set of public and private clouds, which in turn are partitioned further into clusters. Every private cloud has two clusters possibly in a private network domain administered by a single authority. In addition the two public clouds Cloud A and Cloud B contain 5 and 3 clusters respectively, resulting in a total of d_i, $i = 1\dots18$ clusters. Furthermore, each cluster is a collection of nodes. Clusters in the private

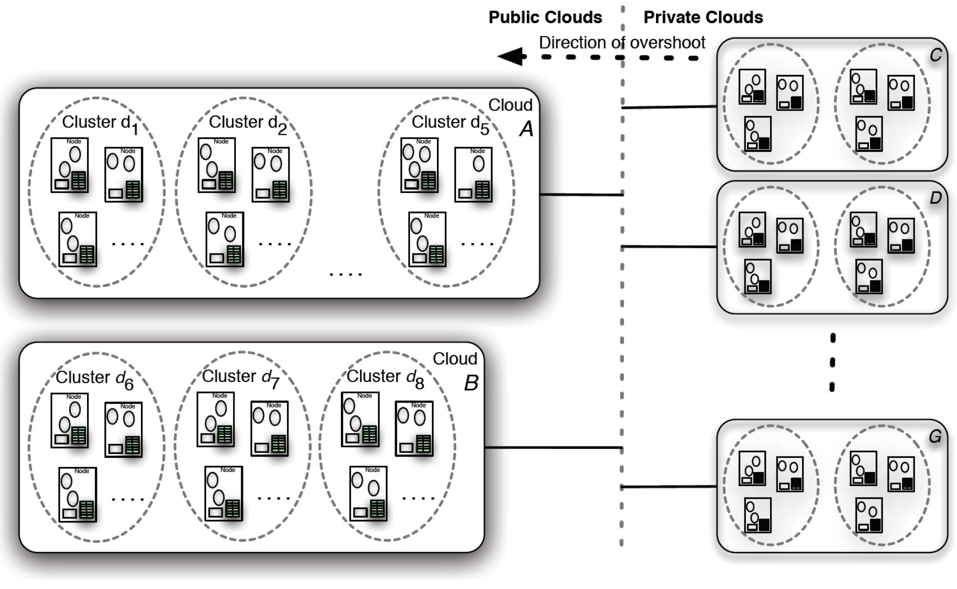


Figure 3: Example scenario

clouds consist of 5 nodes, whereas clusters in the public clouds contain 10 nodes each, which gives a total of 130 nodes available in this network.

We employ one ant species for each of the services, i.e. there will be exactly 25 ant species for each private cloud responsible for deployment mapping of the 25 services local to the corresponding clouds. In other words, 25 ant nests are placed within every private cloud that execute our algorithm and emit ants accordingly. We define a variable Λ accounting for the additionally incurring costs of using hosts in public clouds as a sum over all hosts that participate in mapping $\mathscr{M}$ obtained during the given iteration

$$\Lambda = \sum_{\forall n_i \in M} |n_i| \tag{11}$$

In our experiment, we executed our algorithm using two extended variants of the cost function. The extension to the original function Eq. (4) is shown in the following

$$F' = F(D_r, M_r, L_r) \cdot (1 + g(x)), \tag{12}$$

where function $g(x)$ is defined in two variants using parameter x and Eq. (11).

$$g(x) = \begin{cases} x \cdot \Lambda, & \text{if linear weighting} \\ 1 - e^{-(x \cdot \Lambda)^2}, & \text{if exponential weighting} \end{cases} \tag{13}$$

To see the resulting VM instance mapping when cloud-related costs are absent we set $x = 0$. On the contrary, to include cloud-related costs the scaling parameter is set to $x > 0$. The exact value of parameter x is dependent on the values we apply as costs of using public clouds, hence it is a scaling parameter. In the example scenario with cloud costs $\{10, 1\}$ the scaling parameter we applied was $x = 0.1$.

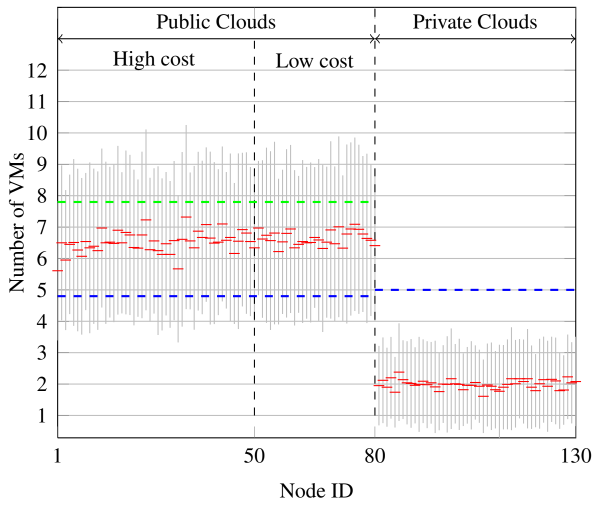


Figure 4: VM instances per node, $x = 0$, no weighting

The two different alternatives in Eq. (13) represent a linear increment (the former) and an exponential increment (the latter alternative) in cloud costs, when $x > 0$. By applying a more fine-grained exponential weighting to cloud-related costs VM mappings are expected to become more balanced, avoiding under-utilization or overload of clusters.

To investigate the three alternatives we executed simulations of the example setting, running the logic 100 times using each variant of the cost function presented above. In Figure 4, VM instance mapping is presented in case cloud-related costs are not taken into consideration, i.e. every node has zero cost for hosting a VM. Simulation results are averaged after the algorithm has converged to a solution and deviation from the average number of instances per node is shown as error bars. We can observe that in the first case 2 VMs are mapped on average to hosts within the private clusters, i.e. on nodes $n_{81..90}$, $n_{91..100}$, $n_{101..110}$, $n_{111..120}$, $n_{121..130}$. That means that for the 10 nodes within each private cloud approximately 20 VMs are mapped for hosting, leaving $(5 \cdot 25) - 20 = 105$ VMs for mapping into the public network. Then, as anticipated we have an average around the $(105 \cdot 5)/80 = 6.6$ VMs mapped to the public hosts in the network, which lies between the two extremes of mapping 3 VMs in public and 2 in private clusters $(3 \cdot 125)/80 = 4.7$ and mapping all 5 VMs of a service to public clusters $(5 \cdot 125)/80 = 7.8$, as shown by the horizontal dashed lines. Naturally, the algorithm does not distinguish between the two public cloud offerings in this case.

In the second experiment (Figure 5) we turned on cloud-related costs and executed our algorithm under the same circumstances otherwise as before. In this case results show that the logic manages to find a mapping that considers the financial penalties of using public clouds. The public cloud with plenty of resources and high cost (nodes $n_{1.50}$) is barely used for deployment, whereas the lower cost public offering is heavily loaded with VMs, while all the dependability requirements are fulfilled,

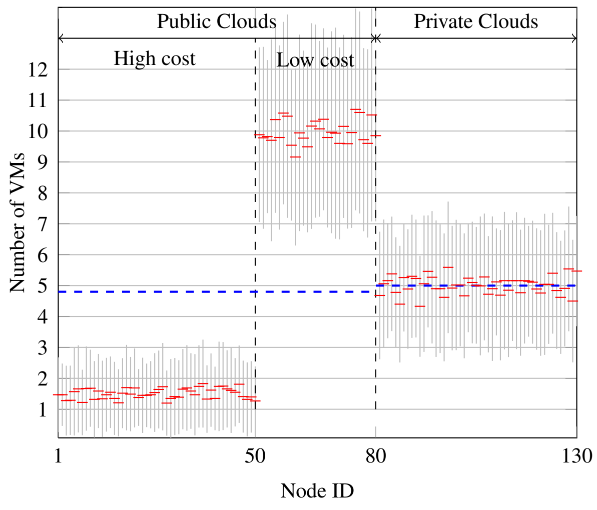


Figure 5: VM instances per node, $x > 0$, linear weighting

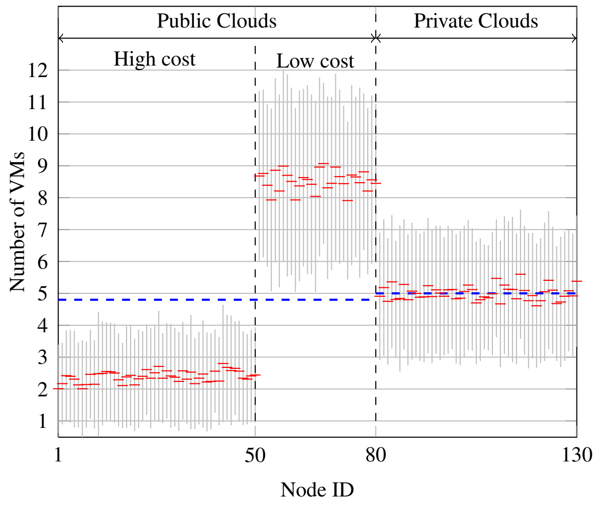


Figure 6: VM instances per node, $x > 0$, exponential weighting

i.e. cluster-disjointness and node-disjointness. At the same time in the private clouds containing 10 nodes, as expected, 5 VMs are mapped to each node on average. This means that each one of the 25 services that are executed within a given private cloud places 1 VM in each of the two local clusters available at 0 cost $((2 \cdot 25)/10 = 5)$. However, due to the cluster-disjointness criteria the 3rd, 4th and 5th VM replica has to be placed to a public cloud with the lowest increment in costs possible, nonetheless taking into account the rest of the requirements.

In the third set of simulations (Figure 6) we executed our deployment mapping algorithm applying the exponential cost function, the second alternative in Eq. (13).

Changes from the previous example can mainly be observed in mappings in the public clouds. Mappings in private clouds are not changed due to the application of the same requirements. Using a slightly more complicated cost evaluation in the algorithm, however, we obtained more balanced deployment mappings. Under the given cost values assigned to the different public offerings the cheaper public cloud gets less overloaded with VMs while the number of mappings in the larger public cloud increases to take over some of the execution load while the original requirements remain satisfied.

In the examples above we have shown that the deployment logic we are developing can be applied in a cloud computing scenario by adjusting the corresponding cost functions evaluating VM mappings, thus adapting to different costs related to usage of resources offered by public providers. Using the logic we are able to obtain VM instance mappings that satisfy dependability and performance requirements while minimizing financial penalties in the special case of handling overshoot scenarios in private clouds.

6. Conclusions

We have presented a swarm intelligence framework targeting the deployment of VM instances in a cloud computing environment. We have designed an algorithm that is fully distributed, scales well by decomposing the problem of deploying multiple services, and paves the way for a deployment logic capable of finding near optimal mappings.

Through the evaluation presented in this paper, we are convinced that our deployment logic is applicable in a cloud computing setting. Nevertheless, we are currently working on a more thorough validation of our approach with new examples. We are also developing a centralized solution using integer linear programming to obtain exact optima as a lower bound for our simulation results. As such, we emphasize the fact that our approach is a heuristic method and may not reach the lower bounds; however, our approach offers the significant advantages of decentralization.

References

[AOVP08] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson. Design and implementation trade-offs for wide-area resource discovery. *ACM Trans. on Internet Technology 8(4)*, 2008.

[CFH$^+$05] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of the 2nd ACM/USENIX Symp. on Networked Systems Design and Implementation (NSDI*, pages 273–286, 2005.

[CH10] M. J. Csorba and P. E. Heegaard. Swarm intelligence heuristics for component deployment. In *Proc. of the 16th Eunice Int'l Workshop and IFIP WG6.6 Workshop*, 2010.

[CHH08a] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In *Proc. of the 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), Turin*. ICST/ACM, September 2008.

[CHH08b] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), Oslo*, LNCS 5053, pages 253–268. IFIP, June 2008.

[CMH09] M. J. Csorba, H. Meling, and P. E. Heegaard. Laying pheromone trails for balanced and dependable component mappings. In *Proc. of the 4th Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5918*, 2009.

[CMHH09] M. J. Csorba, H. Meling, P. E. Heegaard, and P. Herrmann. Foraging for better deployment of replicated service components. In *Proc. of the 9th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), LNCS 5523*, pages 87–101, 2009.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[EL09] E. Elmroth and L. Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *Proc. of the 8th Int'l Conf. on Grid and Cooperative Computing (GCC)*, pages 253–260, 2009.

[FB89] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. on Software Engineering*, 15(11):1427–1436, 1989.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HJ06] T. Heimfarth and P. Janacik. Ant based heuristic for os service distribution on adhoc networks. In *Biologically Inspired Cooperative Computing (BICC)*, pages 75–84, 2006.

[HON$^+$09] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi. A live storage migration mechanism over wan for relocatable virtual machine services on clouds. In *Proc. of the 9th IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGRID)*, pages 460–465, 2009.

[HW10] P. E. Heegaard and O. Wittner. Overhead reduction in a distributed path management system. *Computer Networks*, 54(6):1019–1041, 2010.

[JHJ09] K. Joshi, M. Hiltunen, and G. Jung. Performance aware regeneration in virtualized multitier applications. In *Proc. of the Workshop on proactive failure avoidance recovery and maintenance (PFARM)*, 2009.

[KHD08] R. Kusber, S. Haseloff, and K. David. An approach to autonomic deployment decision making. In *Proc. of the 3rd Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5343*, pages 121–132, 2008.

[LLC] Amazon Web Services LLC. Amazon elastic compute cloud. `http://aws.amazon.com/ec2`.

[MG08] H. Meling and J. L. Gilje. A distributed approach to autonomous fault treatment in spread. In *Proc. of the 7th European Dependable Computing Conference (EDCC)*, pages 46–55, 2008.

[MMHB08] H. Meling, A. Montresor, B. E. Helvik, and Ö. Babaoglu. Jgroup/arm: a distributed object group platform with autonomous replication management. *Softw., Pract. Exper.*, 38(9):885–923, 2008.

[NWG$^+$09] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: an open-source cloud computing infrastructure. *Journal of Physics: Conference Series*, 180(9), 2009.

[PNP88] C. Pu, J. D. Noe, and A. Proudfoot. Regeneration of replicated objects: A technique and its eden implementation. *IEEE Trans. on Software Engineering*, 14:936–945, 1988.

[RBL$^+$09] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.*, 53(4):535–545, 2009.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[Sab06] R. Sabharwal. Grid infrastructure deployment using smartfrog technology. In *Proc. of the Int'l Conf. on Networking and Services (ICNS), Santa Clara, USA*, pages 73–79, July 2006.

[SME09] H. Sato, S. Matsuoka, and T. Endo. File clustering based replication algorithm in a grid environment. In *Proc. of the 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID)*, pages 204–211, Washington, DC, USA, 2009. IEEE Computer Society.

[VAN08] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[WGB09] S. Wardley, E. Goyer, and N. Barcet. Ubuntu enterprise cloud architecture. Technical report, Aug 2009.

[XZF$^+$07] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2007.

[YG09] H. Yu and P. B. Gibbons. Optimal inter-object correlation when replicating for availability. volume 21, pages 367–384, 2009.

[YV05] H. Yu and A. Vahdat. Consistent and automatic replica regeneration. *ACM Trans. on Storage*, 1(1):3–37, 2005.

PAPER G

Swarm Intelligence Heuristics for Component Deployment

Máté J. Csorba and Poul E. Heegaard

In proceedings of 16th EUNICE/IFIP WG 6.6 Workshop (EUNICE'10)

Trondheim, Norway, June 28-30, 2010

SWARM INTELLIGENCE HEURISTICS FOR COMPONENT DEPLOYMENT

Máté J. Csorba, Poul E. Heegaard
Department of Telematics,
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard}@item.ntnu.no

Abstract We address the problem of efficient deployment of software services into a networked environment. Services are considered that are provided by collaborating components. The problem of obtaining efficient mappings for components to host in a network is challenged by multiple dimensions of quality of service requirements. In this paper we consider execution costs for components and communication costs for the collaborations between them. Our proposed solution to the deployment problem is a nature inspired distributed heuristic algorithm that we apply from the service provider's perspective. We present simulation results for different example scenarios and present an integer linear program to validate the results obtained by simulation of our algorithm.

1. Introduction

Implementing distributed networked software systems requires many important design decisions to be made that have strong influence on the Quality of Service (QoS) perceived by the user of the service. A major factor in satisfying QoS requirements of a software service is the configuration of the elementary building-blocks of the service and their mapping to the suitable network elements and resources that are available during execution. Moreover, it is also important for next generation software systems to be capable to (self-)adapt to foreseen and unforseen changes that appear in the execution context. This dynamism in the context of services is even increased by enabling swiftly reconfigurable hardware, allowing mobility and changes in the cardinality of users. Simply improving the QoS metrics without planing is, however, inevitably increasing the costs on the providers' side leading to multifaceted optimization problems.

We address the issue of obtaining efficient and adaptable mappings for software components of networked services as an optimization problem in a distributed environment. We model services as being built by collaborating components with several dimensions of QoS requirements including but not restricted to dependability, performance or energy saving aspects. Correspondingly, our service models are extended with costs relevant for the various dimensions of requirements in a more detailed model. Based on the service models we apply heuristics and a nature-inspired optimization method, called the Cross Entropy Ant System (CEAS) [HW01], [HHW08], [HW10] to solve the problem of deploying service components into the network.

Distributed execution of our deployment mapping algorithm has been an important design criteria to avoid the deficiencies of existing centralized algorithms, e.g. performance bottlenecks and single points of failure. Moreover, we intend to conserve resources by eliminating the need for centralized decision-making and the required updates and synchronization mechanisms. In our earlier work [CHH08b] we selected a well-known example in the domain of task assignment problems and converted it to our context of collaborating components with execution and communication costs. In this paper we extend the initial example from [CHH08b] with two additional example system models, present an Integer Linear Program (ILP) able to solve component mapping problems with load-balancing and remote communication minimization criteria, and compare simulation results obtained by executing our algorithm on the examples presented with the optimum cost solutions given by the ILP.

Related to our work a fair number of approaches aim at improving dependability and adaptability through influencing the software architecture. QoS-aware metadata is utilized together with Service Level Agreements (SLAs) in the planning-based middleware in the MUSIC project [REF$^+$08]. SLAs are common means to target policy based research allocation, e.g. [ATZ07]. The SmartFrog deployment and management framework from HP Labs describes services as collections of components and applies a distributed engine comprised of daemons running on every node in a network [Sab06]. Fuzzy learning is applied for configuration management in server environments targeting efficient resource utilization by Xu et al. in [XZF$^+$07]. Biologically-inspired resource allocation algorithms in service distribution problems have been targeted earlier too, such as by the authors of [HJ06]. A different approach, namely layered queuing networks are employed by Jung et al. in an off-line framework for generating optimal configurations and policies [JJH$^+$08]. Changing the deployment mapping of applications is investigated by others as well, however due to the fact that complexity of exact solution algorithms becomes NP-hard already in case of 2-3 hosts or several QoS dimensions applicability of these methods is restricted. Heuristics, such as greedy algorithms and genetic programming are used by Malek to maximize utility of a service from the users' perspective in [Mal06], whereas we formulate the problem from the providers' view, while still considering user requirements. The various middleware approaches can be very good candidates serving our approach as a means of instrument for deployment that is guided by our logic.

The remainder of this paper is organized as follows. First, in Sect. 2 we discuss the component deployment problem in more detail. Next, an introduction to CEAS follows in Sect. 3. In Sect. 4 an ILP is formulated for the general component deployment problem discussed in this paper. Sect. 5 presents service examples and the corresponding simulation results are evaluated. Finally, in Sect. 6 we conclude and touch upon future work.

2. The Component Deployment Problem

We define the component deployment problem as an optimization problem, where a number $|\mathbf{C}|$ of components (labelled c_i; $i = 1,\ldots,|\mathbf{C}|$) have to be mapped to a set $\mathbf{N}$ of nodes. Components can communicate via a set $\mathbf{K}$ of collaborations (k_j; $j = 1,\ldots,|\mathbf{K}|$). We consider three types of requirements in the deployment problem. Components have execution costs e_i; $i = 1,\ldots,|\mathbf{C}|$, collaborations have communication costs f_j; $j = 1,\ldots,|\mathbf{K}|$ and some of the components can be restricted in deployment mapping. Components that are restricted to be deployed to specific nodes are called *bound* components. Accordingly, we distinguish between the three concepts of component *mappings*, meaning a set variable obtained and refreshed in every iteration of our algorithm; component *bindings*, which represent a requirement that fixes the mapping of components to a constant value; and component *deployment* that is the actual physical placement of components to nodes, including the bound components. Physical deployment of components is triggered after the mappings obtained and refreshed in every iteration by the deployment algorithm converges to a solution satisfying the requirements.

Furthermore, we consider identical nodes that are interconnected in a full-mesh and are capable of hosting components with unlimited processing demand. We observe the processing load components hosted at a node impose and target load-balancing between the nodes available in the network. By balancing the load we mean minimizing the deviation from the global average per node execution cost. Total offered execution load is obtained from the service specification by the sum $\sum_{i=1}^{|\mathbf{C}|} e_i$, thus the global average execution cost can be obtained as

$$T = \frac{\sum_{i=1}^{|\mathbf{C}|} e_i}{|\mathbf{N}|}. \qquad (1)$$

Communication costs are considered only if a collaboration between two components happens remotely, i.e. it happens between two nodes. In other words, if two components are *colocated* (are placed onto the same node) we do consider communication between them to be free (not consuming any network bandwidth).

Our deployment logic is launched with the service model enriched with the requirements specifying the search criteria and with a resource profile of the hosting environment specifying the search space. In our view, however, the logic we develop is capable of catering for any other types of non-functional requirements too, as long as a suitable cost function can be provided for the specific QoS dimension at hand. In this paper, costs in the model are constant, independent of the utilization of underlying hardware. This limitation has been removed and explored in [CHH08a] by the authors. Furthermore, we benefit from using collaborations as design elements as they incorporate local behavior of all participants and all interactions between them. That is, a single cost value can describe communication between component instances, without having to care about the number of messages sent, individual message sizes, etc. For more information on how we use collaborations for system modelling we refer to [CHH08b] and [CHH08a].

We, then define the objective of the deployment logic as obtaining an efficient (low-cost if possible optimum) mapping of components to nodes, $\mathbf{M} : \mathbf{C} \rightarrow \mathbf{N}$, one that satisfies the requirements in reasonable time. More importantly, the actual placement of components does not change with every iteration of the algorithm but is changed only on a larger timescale, once a mapping is converged to a solid solution to avoid churn. Re-deployment, for adaptation to changes in the execution context involves migrating components, which naturally incurs additional costs. In principle migration costs can be considered as thresholds prohibiting changing the deployment mapping if the benefit is not high enough. In this work, however, we do not consider adaptation and migration costs.

The heuristics we use in our deployment logic is guided by a cost function, $F(\mathbf{M})$ that is used to evaluate the suggested mappings iteration-by-iteration. The construction of the cost function is in accordance with the requirements of the service. How we build the corresponding cost function is discussed in Sect. 3.2.

3. Component Deployment using the Cross Entropy Ant System

3.1 The Cross Entropy Ant System

The key idea in the CEAS is to let many agents, denoted *ants*, iteratively search for the best solution according to the problem constraints and cost function defined. Each iteration consists of two phases; the *forward* ants search for a solution, which resembles the ants searching for food, and the *backward* ants that evaluate the solution and leave markings, denoted *pheromones*, that are in proportion to the quality of the solution. These pheromones are distributed at different locations in the search space and can be used by forward ants in their search for good solutions; therefore, the best solution will be approached gradually. To avoid getting stuck in premature and sub-optimal solutions, some of the forward ants will explore the state space freely ignoring the pheromone values.

The main difference between various ant-based systems is the approach taken to evaluate the solution and update the pheromones. For example, AntNet [CD98] uses reinforcement learning while CEAS uses the *Cross Entropy (CE) method for stochastic optimization* introduced by Rubinstein [Rub99]. The CE method is applied in the pheromone updating process by gradually changing the probability matrix $\mathbf{p}$ according to the cost of the solutions found. The objective is to minimize the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$ for iteration r and $r-1$ respectively. For a tutorial on the method, [Rub99] is recommended.

The CEAS has demonstrated its applicability through a variety of studies of different path management strategies [HHW08], such as shared backup path protection, p-cycles, adaptive paths with stochastic routing, and resource search under QoS constraints. Implementation issues and trade-offs, such as management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism and self-tuned packet rate control. Additional reduction in the overhead is accomplished by pheromone sharing where ants with overlapping requirements cooperate in finding solutions by (partly) sharing information, see [HW10] for details and examples on application.

In this paper, the CEAS is applied to obtain the best deployment mapping $\mathbf{M} : \mathbf{C} \rightarrow \mathbf{N}$ of a set of components, $\mathbf{C}$, onto a set of nodes, $\mathbf{N}$. The nodes are physically connected by links used by the ants to move from node to node in search for available capacities. A given deployment at iteration r is a set $\mathbf{M}_r = \{\mathbf{m}_{n,r}\}_{n\in\mathbf{N}}$, where $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ is the set of components at node n at iteration r. In the CEAS applied for routing the path is defined as a set of nodes from the source to the destination, while now we use the deployment set $\mathbf{M}_r$ instead. The cost of a deployment set is denoted $F(\mathbf{M}_r)$. Furthermore, in the original CEAS we assign the pheromone values $\tau_{ij,r}$ to interface i of node j at iteration r, while now we assign $\tau_{mn,r}$ to the component set $\{m\}$ deployed at node n at iteration r.

In CEAS applied for routing and network management, selection of the next hop is based on the *random proportional rule* presented below. In our case however, the *random proportional rule* is applied for deployment mapping. Accordingly, during the initial exploration phase, the ants randomly select the next *set of components* with uniform probability $1/|\mathbf{C}|$, where $|\mathbf{C}|$ is the number of components to be deployed, while in the normal phase the next set is selected according to the *random proportional rule* matrix $\mathbf{p} = \{p_{mn,r}\}$, where

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l\in\mathbf{M}_{n,r}} \tau_{ln,r}} \tag{2}$$

The pheromone values τ_r are updated by the backward ants as a function of the previous pheromone value τ_{r-1}, the cost of the deployment set $\mathbf{M}_r$ at iteration r, and a control variable γ_r (denoted the *temperature*) that captures the history of the cost values such that the total cost vector does not have to be stored. The CEAS function was first introduced in [HW01], for later extensions, details and examples see [HW10].

3.2 The Component Deployment Algorithm

To build a distributed cooperative algorithm, in contrast to a centralized approach, we employ the autonomous ant-like agents (denoted ants) of the CEAS method that cooperate in pursuing a common goal. Ants base their decisions solely on information available locally at a node they currently reside in. Accordingly, in our logic the information required for optimization is distributed across all participating nodes, this being a contributing factor to robustness, scalability and fault tolerance of the method.

In our deployment algorithm ants are emitted from one designated place in the network, from the so-called ant-nest. When an ant starts its random-walk over the available nodes it is assigned with the task of deploying the set of components, $\mathbf{C}$. One round-trip (not necessarily visiting every node, but arriving back at the nest eventually) of an ant is called an iteration of the algorithm, which is repeated continuously until convergence or until the algorithm is stopped. During its random-walk the ant selects the next hop to visit entirely randomly. When arriving at a node the ant's behavior depends on if the ant is an *explorer* or a *normal* ant. The difference between the two types of ants is in the method they use to select a mapping, $m_n \subset \mathbf{M}$, for a (maybe empty) subset of $\mathbf{C}$ at each node $n \in \mathbf{N}$ they visit. Normal ants decide on whether to deploy some components at a particular node based on the pheromone database at the given node, whereas explorer ants make a decision purely randomly, thus enforcing exploration of the state-space. Explorer ants can be used both initially to cover up the search space and later, after the system is in a stable state as well to discover fluctuations in the network and, thus to aid adaptation. The useful number of initial explorer iterations is depending on the given problem size (e.g. can be estimated by the number of components in a service). Normal ants, on the other hand focus entirely on optimizing the mapping ($\mathbf{M}$) iteration-by-iteration using and updating the pheromone tables.

An important building-block of the algorithm is the cost function applied to evaluate the deployment mapping obtained by one ant during its search. We denote the application of the cost function as $F(\mathbf{M})$. The function itself is built in accordance with the requirements. As discussed in Sect. 2, in this paper we consider load-balancing and remote cost minimization. Every ant doing its search for a mapping is capable of sampling load-levels at the nodes visited, thus the execution load imposed on the nodes in the network can be obtained by each ant as a list of sample values in the form of

$$\hat{l}_n = \sum_{c_i \in m_n} e_i, n \in \mathbf{N}, m \in \mathbf{M}. \tag{3}$$

The overall cost function evaluating the mapping obtained in an iteration using Eq. (1) and Eq. (3) then becomes

$$F(\mathbf{M}) = \sum_{n=1}^{\mathbf{N}} |\hat{l}_n - T| + \sum_{j=1}^{\mathbf{K}} I_j \, f_j \tag{4}$$

where

$$I_j = \begin{cases} 1, & \text{if } k_j \text{ remote} \\ 0, & \text{if } k_j \text{ internal to a node} \end{cases} \tag{5}$$

is an indicator function for evaluating collaborations between pairs of components.

Optimization of mappings is achieved by gradually modifying pheromone values aligned to sets of components. Pheromone values are organized into tables and are stored in every node participating in hosting the service being deployed. We use *bitstring* encoding to index the pheromone table, each entry representing a given combination/subset of components, i.e. a flag is assigned to every (unbound) component in the service model. Thus, the pheromone database has to accommodate $2^{|\mathbf{C}|}$ floating point entries using this encoding. Normalizing the entries in a node an ant can obtain a probability distribution of component sets to be mapped at the particular node. Using CEAS, which is a subclass of Ant Colony Optimization (ACO) algorithms, the optimal solution emerges in a finite number of iterations once the optimum has been observed, which does in fact happen with probability close to one in ACO systems.

Indexing of the database based on component set identifiers is illustrated in the following example. Let us start with a service provided by 4 components, i.e. $|\mathbf{C}| = 4$, $\mathbf{C} = \{c_1, c_2, c_3, c_4\}$, the database size is $2^4 = 16$. If an ant needs to collect or update information e.g. regarding the deployment of the component set $\{c_2, c_4\}$, the set labelled by the bitstring $'1010'B$ has to be addressed, which is equivalent to accessing the $'1010'B = 10^{th}$ element of the pheromone table at the node the ant resides in. The lifetime of an ant, which is equivalent to one iteration of the algorithm, contains two phases. First *forward search* is conducted during which the ant looks for a deployment mapping $\mathbf{M}$ for the set $\mathbf{C}$ visiting arbitrary nodes in the network. Once a complete mapping $\mathbf{M}$ is obtained by the ant the mapping has to be evaluated using the cost function $F(\mathbf{M})$. Using F the cost of the mapping is obtained using which the ant updates the pheromone databases in the nodes visited (the forward route has been stored in the hop-list, $\mathbf{H}$) during *forward search* during the second phase of its lifetime called *backtracking*. Once an ant is finished with backtracking and arrives back to its nest a new iteration can start and a new ant will be emitted. With convergence of the (distributed) pheromone database a strong value will emerge indicating the suggested deployment mapping, while inferior combinations will evaporate. The algorithmic steps of ants' behavior are summarized briefly in Algorithm 1.

Algorithm 1 Deployment mapping of the set of components $\mathbf{C} = \{c_1, \ldots, c_{|\mathbf{C}|}\}$

```
1 Select next node to visit, $n$ randomly and add $n$ to the hop-list $\mathbf{H} = \mathbf{H} + \{n\}$.
2 Select a set of components $m_n \subseteq \mathbf{C}$ according to the random proportional rule
  (normal ant), Eq. (2), or in a random manner (explorer ant). If such a set cannot
  be found, goto step 1.
3 Update the ant's deployment mapping set, $\mathbf{M} = \mathbf{M} + \{m_n\}$.
4 Update the set of remaining components to be deployed, $\mathbf{C} = \mathbf{C} - m_n$.
5 If $\mathbf{C} \neq \emptyset$ then goto 1., otherwise evaluate $F(\mathbf{M})$ and update the pheromone
  values corresponding to the list of mappings $\{m_n\} \in \mathbf{M}$ going backwards along
  $\mathbf{H}$.
6 If stopping criteria is not met then initialize and emit new ant and goto 1.
```

In the algorithm we presented we have a trade-off between convergence speed and the quality of the obtained solution. However, during the deployment of services in a dynamic environment a pre-mature solution, which does satisfy the functional and non-functional requirements often suffices. In the next section we establish a centralized, off-line model to evaluate our deployment logic and to form a basis for cross-validation of the results obtained by simulation of the algorithm.

4. An Integer Program to Validate the Algorithm

To validate the results we obtain via simulation and using various deployment scenarios we have developed an Integer Linear Program (ILP), which we solve using regular solver software. In this ILP we take into account the two counteracting objectives we presented in Sect. 2 and define a solution variable $m_{i,j}$ that will show the optimal, i.e. lowest cost mapping of components to nodes.

We start the definition of the ILP with two parameters. First $b_{i,j}$

$$b_{i,j} = \begin{cases} 1, & \text{if component } c_i \text{ is bound to node } n_j, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

which enables the model to fix some of the mappings to cater for bound components, if any, in the model of a given service. Second, T

$$T = \lfloor \frac{\sum_{i=1}^{|\mathbf{C}|} e_i}{|\mathbf{N}|} \rfloor \tag{7}$$

that is used to approximate the ideal load-balance among the available nodes in the network. Beside the binary solution variable showing the resulting mapping, $m_{i,j}$

$$m_{i,j} = \begin{cases} 1, & \text{if component } c_i \text{ is mapped to node } n_j, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

we utilize two additional variables. One variable for checking whether two components that communicate via a collaboration, k_i, are colocated or not, in variable col_i.

$$col_i = \begin{cases} 0, & \text{if } c_l, c_k \in k_i \text{ and } c_l \text{ is colocated with } c_k, \\ 1, & \text{otherwise.} \end{cases} \tag{9}$$

Moreover, another variable, $Delta_j$, for calculating the deviation from the ideal load-balance among the nodes participating in hosting the components.

$$\Delta_j \geq 0, \forall n_j \in \mathbf{N} \tag{10}$$

The objective function we use in the ILP is naturally very similar to Eq. (4), however to keep the model within the linearity requirement of the ILP here we use addition.

$$min \sum_{j=1}^{|\mathbf{N}|} \Delta_j + \sum_{i=1}^{|\mathbf{K}|} f_i \cdot col_i \tag{11}$$

Having established the objective function we have to define the constraints the solutions are subjected to to obtain feasible mappings. First we stipulate that there has to be one and only one mapping for all of the components.

$$\sum_{j=1}^{|\mathbf{N}|} m_{i,j} = 1, \forall c_i \in \mathbf{C} \tag{12}$$

In addition, the ILP has to take into account that some component mappings might be restricted in the case of components explicitly bound in the model. Thus, we restrict the mapping variable $m_{i,j}$ using $b_{i,j}$.

$$m_{i,j} \geq b_{i,j}, \forall c_i \in \mathbf{C}, \forall n_j \in \mathbf{N} \tag{13}$$

Next we introduce two constraints to implicitly define the values of the variable Δ_j that we apply in the objective function. We use two constraints instead of a single one to avoid having to use absolute vales (i.e. the $abs()$ function) and thus we avoid non-linear constraints.

$$\sum_{i=1}^{|\mathbf{C}|} e_i \cdot m_{i,j} - T \leq \Delta_j, \forall n_j \in \mathbf{N} \tag{14}$$

$$T - \sum_{i=1}^{|\mathbf{C}|} e_i \cdot m_{i,j} \leq \Delta_j, \forall n_j \in \mathbf{N} \tag{15}$$

Lastly, we introduce constraints, again for implicitly building the binary variable indicating colocation of components.

$$m_{i,j} + m_{k,j} \leq (2 - col_l), k_l = (c_i, c_k) \in \mathbf{K}, \forall c_i, c_k \in \mathbf{C}, \forall n_j \in \mathbf{N} \tag{16}$$

$$m_{i,j1} + m_{k,j2} \leq 1 + col_l, k_l = (c_i, c_k) \in \mathbf{K}, \forall c_i, c_k \in \mathbf{C}, \forall n_{j1}, n_{j2} \in \mathbf{N} \tag{17}$$

Based on this definition the ILP can be executed by a solver program and mapping costs can be obtained by submitting the appropriate data describing any given scenario corresponding to our general definition of the deployment problem in Sect. 2. The optimum mapping of components to nodes will be obtained according to the objective Eq. (11) subject to Eq. (12) – (17).

In the next section we present the example models we used in our simulations and also evaluate the mapping costs obtained by finding a lower bound using the ILP presented in this section.

5. Example Scenarios

In this section we present the three example models we simulated and validated the corresponding results for.

The first example we consider has been investigated, solved and the solutions were compared to other authors work by Widell et al. in [WN04]. This example originates from heuristical clustering of modules and assignment of clusters to nodes [Efe82], which problem is NP-hard. We translate the module clustering and assignment problem to execution costs and communication costs while the complexity remains NP-hard even if we only deal with a single service at a time. Fig. 5a shows the first example set of components and the collaborations between them, it comprises $|\mathbf{C}| = 10$ components interconnected by $|\mathbf{K}| = 14$ collaborations. Out of the ten components three (shaded) are bound to one of the nodes, c_2 to n_2, c_7 to n_1 and c_9 to n_3. The execution costs of components and communication costs of collaborations are shown as UML note labels.

The next figure (Fig. 2) shows the single optimal solution of mapping the components of the first example to three available nodes, $n_{1..3}$, i.e. it shows the optimum mapping $\mathbf{M} : \mathbf{C} \rightarrow \mathbf{N}$. In this optimum cost mapping, the deviation from the total load-balance ($T = 68$) among the nodes are $2, -8, 7$ respectively and the mapping incurs a communication cost of 100 for the remote collaborations. Thus, this mapping

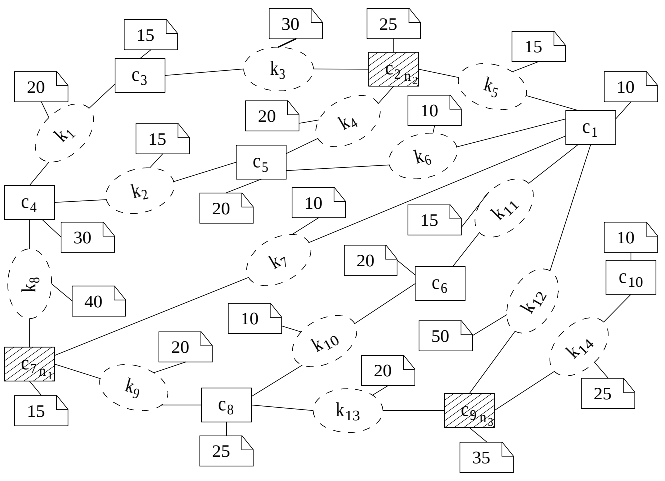


Figure 1: Example 1, costs

results in an overall cost value equal to 117. More details are discussed about this example in [CHH08b].

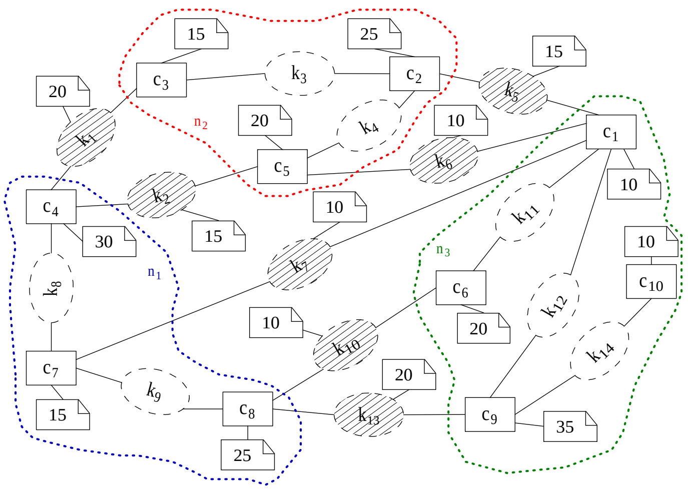


Figure 2: Example 1, optimum mapping

The second example we considered is obtained by extending the first setting into a larger service model and at the same time increasing the number of nodes available for deployment mapping (Fig. 3). The number of components has been increased to 15, out of which 5 components are bound, more collaborations have been introduced and one additional node is added to the deployment environment.

This extended example is introduced to examine how our deployment algorithm behaves with the same type of problem as the computational effort needed increases

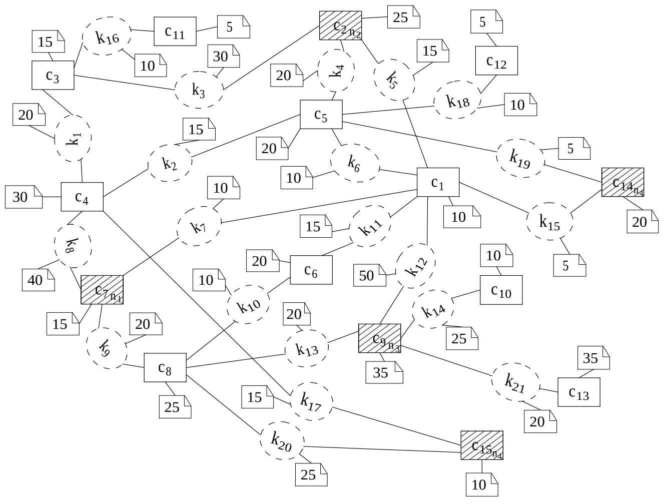


Figure 3: Example 2, costs

or, in other words, as the solution state-space is extended.

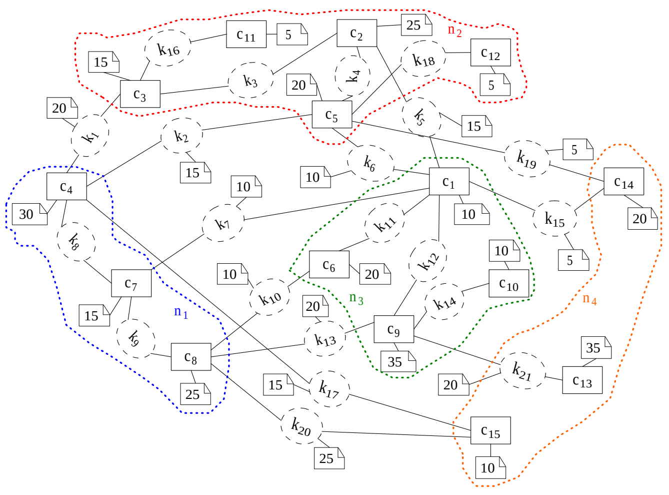


Figure 4: Example 2, optimum mapping

The next figure, Fig. 4 presents the resulting mapping of components of the second service model to four equivalent nodes after a solution with optimum cost has been found.

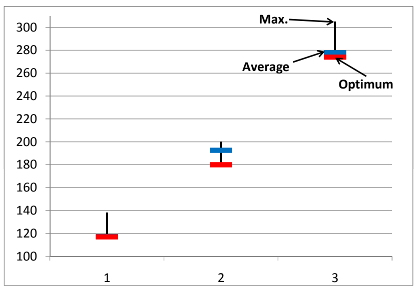
In the third example the cardinality of **C** remains the same but the configuration is changed as well as the number of collaborations. In addition, the number of available nodes is increased to 6, as shown in Table 1. Due to the even more complex collaboration pattern between the components in *Example* 3 we omit the figure showing the corresponding model and its optimal mapping to 6 nodes.

Table 1 also shows the absolute minimum cost values (*Optimum cost*) in the three example settings obtained by the ILP presented in Sect. 4. The presented values were generated by executing the algorithm in the simulation environment 100 times for each example model. We can see from the resulting solutions that in the first example our algorithm finds the optimum in 99 simulation runs. The somewhat larger scenario, *Example* 2 is more difficult to solve for the algorithm, thus we have a larger deviation in the mapping costs. The average cost found is slightly above the optimum as well in this case. However, it is to be noted that the adjective *slightly* is appropriate here as by changing the placement of a single component from the optimum configuration to a near-optimal one increases the costs not only by 1 but significantly more, this is also the reason for the increased deviation in this case. (In fact, the algorithm has generally found a variety of three different configurations, the optimum with cost 180 and two sub-optimal configurations with costs 195 and 200, this gave the average of 193 shown in Table 1.

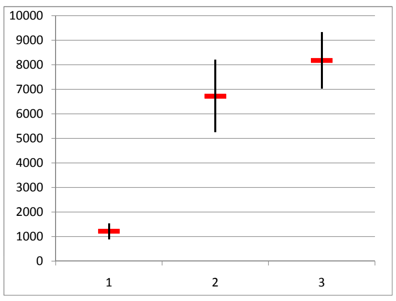
Table 1: Example scenarios

	|**C**| (bound)	|**K**|	|**N**|	Optimum cost	Sim. avg.	Sim. stdev.
Example 1	10(3)	14	3	117	117.21	2.1
Example 2	15(5)	21	4	180	193	9.39
Example 3	15(5)	28	6	274	277.9	5.981

In *Example* 3 the algorithm managed to obtain solutions with costs closer to the absolute optimum obtained by the ILP, with less deviation at the same time. The main cause of this lies in the fact that the collaboration costs in the example are more fine grained, thus, the algorithm managed to find near-optimum solutions with only slightly higher costs. The average mapping costs obtained in the 100 runs and the number of *normal-ant* iterations our CEAS-based algorithm had to perform are shown in Fig. 5.



(a) Mapping costs



(b) Number of iterations

Figure 5: Simulation results for the three examples

The most significant difference for the algorithm in complexity between the original example and the extended ones lies in the size of the pheromone database needed. As the algorithm uses a binary pheromone encoding, in the first example the number of pheromone entries required is 2^7 as the number of components free to map is 7. In the larger examples, however, the pheromone database size increases to 2^{10} in each node, which results in a theoretical state-space of $4 \cdot 2^{10}$ and $6 \cdot 2^{10}$ for CEAS. Correspondingly, in the experiment we applied 2000 *explorer-ants* for the initial example, but 5000 of them in the larger examples. One iteration of a centralized global-knowledge logic, such as the ILP in Sect. 4, is not really comparable with one iteration in the distributed CEAS, which is a tour made by the ant. Nevertheless, the iterations and cuts required by the ILP while solving the three example settings are shown in Table 2. The number of required (explorer and normal) iterations in CEAS is naturally higher than what is required for the ILP with a global overview. However, we advocate that we gain more by the possibility of a completely distributed execution of our algorithm and also because of the capability of adaptation to changes in the context, once the pheromone database is built up after the initial phase. For more on the adaptation capabilities of CEAS in component deployment problems we refer to [CHH08a] and [CMHH09].

Table 2: Computational effort needed by the ILP

	Example 1	Example 2	Example 3
Simplex iterations	86	495	1075
Branch and cut nodes	0	5	33

6. Closing Remarks

We presented how the deployment of distributed collaborating software components can be supported by swarm intelligence and we have introduced our ACO-based algorithm for obtaining optimal mappings of components to execution hosts. The software components we consider are described by a model enriched with relevant costs, in this particular paper with execution and communication costs. The logic we have developed can be executed in a fully distributed manner, thus, it is free of the deficiencies most existing centralized approaches suffer from, such as performance bottlenecks or single points of failure. We have showed that our algorithm is able to obtain load-balancing among the execution hosts and minimize remote communication at the same time that constitute two contradictory objectives in the deployment mapping problem.

The two main contributions of this paper are the ILP model applicable for component deployment scenarios and the cross-validation of the results obtained by our algorithm and the ILP using the three example scenarios presented. We have used the ILP to find the optimum mappings and a lower bound for the mapping costs obtained by heuristics. It is difficult, however, to compare execution times or required iterations of the two different approaches. During service deployment in a dynamic environment we are often satisfied with pre-mature solutions as long as they adhere to all the functional and non-functional requirements. ACO-based systems are proven to be able to find the optimum at least once with a probability near one, afterwards convergence to this solution is assured within a finite number of steps. CEAS, the main cornerstone in our algorithm can be considered as a subclass of ACO algorithms.

As future work we can identify several directions, out of which we are currently focusing on extending the ILP definition to cater for requirements other than the ones discussed in this paper, such as dependability aspects. Besides, we will continue validating the behavior of our deployment algorithm with extended service models and network scenarios.

References

[ATZ07] D. Ardagna, M. Trubian, and L. Zhang. Sla based resource allocation policies in autonomic environments. *J. of Parallel Distrib. Comput.*, 67(3):259–270, 2007.

[CD98] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, 1998.

[CHH08a] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In *Proc. of the 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), Turin*. ICST/ACM, September 2008.

[CHH08b] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), Oslo*, LNCS 5053, pages 253–268. IFIP, June 2008.

[CMHH09] M. J. Csorba, H. Meling, P. E. Heegaard, and P. Herrmann. Foraging for better deployment of replicated service components. In *Proc. of the 9th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), LNCS 5523*, pages 87–101, 2009.

[Efe82] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15(6):50–56, 1982.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HJ06] T. Heimfarth and P. Janacik. Ant based heuristic for os service distribution on adhoc networks. In *Biologically Inspired Cooperative Computing (BICC)*, pages 75–84, 2006.

[HW01] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[HW10] P. E. Heegaard and O. Wittner. Overhead reduction in a distributed path management system. *Computer Networks*, 54(6):1019–1041, 2010.

[JJH$^+$08] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2008.

[Mal06] S. Malek. A user-centric framework for improving a distributed software system's deployment architecture. In *Proc. of the doctoral track at the 14th ACM SIGSOFT Symp. on Foundation of Software Engineering, Portland*, 2006.

[REF$^+$08] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E. Stav. Composing components and services using a planning-based adaptation middleware. In *Proc. of SC, LNCS4954*, pages 52–67. Springer-Verlag, 2008.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[Sab06] R. Sabharwal. Grid infrastructure deployment using smartfrog technology. In *Proc. of the Int'l Conf. on Networking and Services (ICNS), Santa Clara, USA*, pages 73–79, July 2006.

[WN04] N. Widell and C. Nyberg. Cross entropy based module allocation for distributed systems. In *Proc. of the 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, pages 187–200, Berkeley, CA, USA, 2004. USENIX Association.

[XZF$^+$07] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2007.

Part III

THESIS APPENDIX

A Bio-inspired Method for Distributed Deployment of Services

Mátè J. Csorba and Hein Meling and Poul E. Heegaard

Accepted for publication in New Generation Computing

Ohmsha Ltd. and Springer

A BIO-INSPIRED METHOD FOR DISTRIBUTED DEPLOYMENT OF SERVICES

Máté J. Csorba[1], Hein Meling[2], Poul E. Heegaard[1]

[1]*Department of Telematics,*
Norwegian University of Science and Technology,
Trondheim, Norway
{Mate.Csorba, Poul.Heegaard}@item.ntnu.no

[2]*Department of Electrical Engineering and Computer Science,*
University of Stavanger,
Stavanger, Norway
Hein.Meling@uis.no

Abstract We look at the well-known problem of allocating software components to compute resources (nodes) in a network, given resource constraints on the infrastructure and the quality of service requirements of the components to be allocated to nodes. This problem has many twists and angles, and has been studied extensively in the literature. Solving it is particularly problematic when there is extensive dynamism and scale involved. Typically, heuristics are needed.

In this paper, we present a new breed of heuristics for solving this problem. The distinguishing feature of our approach is a decentralized optimization framework aimed at finding near optimal mappings within reasonable time and for large scale. Three different incarnations of the problem are explored through simulations. For one problem instance, we also provide exact solutions, and show that our technique is able to find near optimal solutions with low variance. In the largest example, a public-private cloud computing scenario is used, where different clouds are associated with financial costs, and we show that our approach is capable of balancing the load as expected for such a scenario.

Keywords: Service Deployment, Biologically-inspired Systems, Decentralized Optimization

1. Introduction

Many popular web applications and services are currently being deployed in large-scale data center environments due to the inherent scaling needs of such applications. Moreover, data center environments may consist of hundreds of thousands of nodes, and at this scale, there is bound to be a constant flux of topology changes due to scheduled maintenance and failures, and dynamism due to varying usage patterns and characteristics of the different applications deployed within the hosting data centers. Accounting for *all* the parameters involved in such a system is a challenging undertaking, and flexible methods are necessary to maintain the desired Quality of

Service (QoS) levels at acceptable costs. A fundamental problem in this scenario is the *mapping of components*[1] *to nodes* within the hosting data center(s), while accounting for the necessary tradeoffs that characterize the environment and the services to be deployed. This is what we refer to as *the deployment problem*.

The main contribution of this work is a bio-inspired, decentralized optimization technique for solving the deployment problem. The method is a search-based heuristic aimed at finding near optimal mappings even under harsh network conditions. We explore several incarnations of this problem through simulations; each at a different level of granularity and targeting different QoS requirements, as a mean to demonstrate the flexibility of our approach. For comparison, one instance of the problem is also solved with a traditional centralized optimization technique that finds the exact optimum. We show that for three problem sizes, the simulations of our decentralized approach can still achieve close to the optimal value, with low variance. Note however, techniques that find the exact optimum fall short as the problem size grows, whereas our heuristic is shown to scale conveniently to large problem sizes. Moreover, due to the inherent dynamism that exists in the systems we are targeting, a solution would quickly become suboptimal. However, our approach is able to rapidly adapt to changes in the environment, e.g. a network partition/merge event, by recalculating the deployment configurations, constrained by a threshold reflecting the migration costs. This ability of our approach is invaluable for deploying services in large-scale data center networks.

Our decentralized optimization framework is built around the Cross Entropy Ant System (CEAS) [HW01, HHW08], which is derived from Ant Colony Optimization (ACO) [DMC96]. CEAS uses ant-like agents, denoted *ants*, that can move around in the network, identifying potential locations where components might be placed, and leave pheromone trails as a means to facilitate indirect communication about suitable locations for placement. The CEAS framework requires the definition of a cost function for the specific optimization problem at hand. The purpose of the cost function is to evaluate the utility of a given deployment configuration during the optimization process, eventually leading to the preferred deployment configuration. The proper selection of these functions is crucial for guiding the search for a deployment mapping that satisfies the QoS requirements of the service, which can then be deployed using some execution framework. Essential are also the constraints of the problem; be it load balancing or a certain availability level.

The remainder of the paper is organized as follows. In the following, we further elaborate on the deployment problem and place it in the context of a software development and deployment cycle. Then in Sec. 1.2, we discuss the complexity and scalability issues that are facing us in solving large scale deployment problems, followed by a discussion on how to capture and consider costs and requirements of the services that are deployed. In Sec. 1.4 we present candidate target environments for our deployment framework. Cost function design is discussed in Sec. 2, while Sec. 3

[1]A component is assumed to be any software component that can be placed on a physical compute node, e.g. a building block of a service or application, or a virtual machine. Herein these concepts are used somewhat interchangeably.

gives a general introduction to the CEAS framework that we build upon and presents a general definition of our algorithm. In Sec. 4 three example scenarios are shown: (i) deployment of collaborating components with communication costs (Sec. 4.1), (ii) deployment of replicas constrained by dependability requirements (Sec. 4.3), followed by our approach to mapping virtual machines in a public-private cloud computing environment (Sec. 4.4). In Sec. 4.5, we present a centralized approach for finding optimal mappings under certain sets of requirements and validate our results obtained by the decentralized algorithm. Finally, we review some related research and conclude.

1.1 The Deployment Problem

As described initially, we target the problem of mapping subsets of services to physical resources, i.e. nodes capable of hosting such services, in an efficient manner such that the QoS requirements of the services are satisfied. This problem can be viewed in the context of the software development and deployment cycle as partially illustrated in Fig. 1, where each layer captures a part of the multifaceted deployment problem. The execution nodes are shown in the bottom part, whereas the services to be deployed are modeled at the second layer up. The goal then is to obtain the mapping $M : \mathbf{C} \rightarrow \mathbf{N}$ between the building blocks of the service (the set $\mathbf{C}$) and the available nodes ($\mathbf{N}$).

The non-functional requirements of services are captured in the QoS dimensions layer; these can be issues such as dependability, security, performance, or energy-saving, all of which contribute to increasing the problem size and complexity. Finally, on the top layer we have the possible varieties of usage scenarios that can be captured by enriching the service models with additional usage related information, e.g. arrival rates for a component that handles user requests.

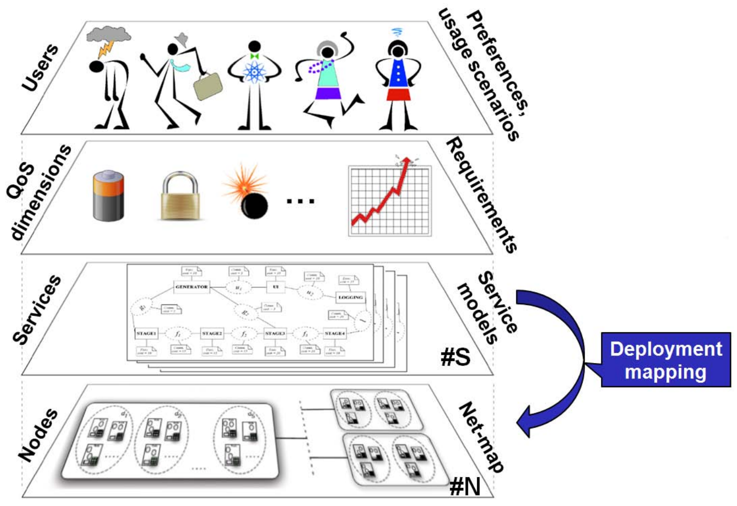


Figure 1: The multiple dimensions of the deployment problem.

1.2 Problem Complexity and Scalability Issues

The deployment mapping problem can be formulated as a multi-dimensional bin packing problem [KLMS84], where each physical node is a bin and each constraint spans a dimension. The components with their associated resource requirements are the objects that are to be packed in the bins (nodes) available in the system. Thus, since bin packing is NP-hard [KLMS84], the deployment mapping problem is also NP-hard. Even determining the existence of a valid packing is itself a NP-hard problem [WSVY07]. Moreover, the general module allocation problem has been shown to be NP-complete [FB89], except for some communication configurations.

Given the complexity of solving the deployment mapping problem, it is to be expected that it cannot be solved even for moderately large scenarios. And especially not in realistic scenarios, such as finding efficient mappings in large-scale data center infrastructures, as the problem size grows exponentially with the number of hosting nodes and the number of components to be deployed. And also because a multitude of services are deployed simultaneously, while ensuring proper balance between load characteristics and service availability at every data center site.

Moreover, mappings found by an algorithm can be affected by a plethora of parameters during execution, e.g. due to the influence of concurrent services. Also, introducing dependability requirements for the services results in additional complexity, e.g. due to the use of replication protocols and their need to ensure consistency. Thus given the problem complexity and for the relevant problem sizes, our only option is to look for efficient heuristic algorithms instead of seeking to find the exact optimum.

Furthermore, the heuristic algorithm should also be resilient to dynamics, or specifically it should exhibit some degree of autonomy and adapt to changes in the environment. Such changes might include mobility of users, node churn, network disconnection (split/merge), incremental scaling of services, among others. Thus, our approach to develop an autonomic process for tracking the best solutions in a dynamic environment is to build on the inherent self-organization properties of the CEAS framework. This framework facilitates decentralized optimization, avoiding the need for any centralized information storage and decision making. To cope with large scale problems however, an efficient data representation (storage) is necessary at each node participating in the decentralized algorithm. We discuss and evaluate three different data representations in Sec. 3.3.

1.3 Costs and Constraints

Given a specific deployment problem, our optimization technique, or deployment logic, must account for a range of parameters representing the QoS requirements and constraints of the services being deployed. In this section we describe how these requirements and constraints are captured and give a few examples.

The QoS requirements are captured at design time and specified in a collaboration-oriented design model. The requirements may represent qualities such as security, performance, availability, portability, etc. In fact, our deployment logic can capture any kind of system property, as long as a suitable cost function can be defined for it. Fig. 2 shows a sample collaboration diagram, enriched with two cost function attributes (QoS requirements), namely *execution* and *communication cost*. The execution cost capture the CPU requirement of a component to be deployed, whereas the communication cost reflect its network usage requirement.

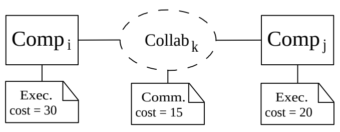


Figure 2: A collaboration diagram with non-functional requirements.

A service may constrain the placement of its components to specific nodes by means of a binding, e.g. a database server may have to be assigned to a specific node. Such a binding generally reduces the search space. However, bound components are still accounted for in the cost of a computed deployment mapping.

Our approach is designed as a continuous optimization process in order to facilitate rapid response to dynamism; hence reconfiguration of initial component placements are expected. However, to avoid migrating components for marginal cost savings, the cost of migration must be accounted for; herein we use a simple threshold scheme.

The three example scenarios presented in Sec. 4 use different requirements. The main overall aim is to load-balance execution cost across the available nodes, while accounting for all services running in the environment, remote communication costs, and dependability of the service being deployed. The second scenario introduces replica management rules to enforce cluster- and node disjointness. In the last example, we extend the set of requirements to include financial costs of using different clusters for placement in a public-private cloud computing setting. Without loss of generality, all cost functions are formalized as minimization problems, i.e. the less the cost the better the solution.

1.4 System Model and Target Systems

This section describes the system model and gives some notation, followed by a brief description of potential target systems for which our deployment logic will be of relevance.

We consider the elementary building blocks (components) of a service as the unit to be deployed. Each component has well-defined interfaces and communicates by means of message exchange. A *service* is defined as the collaboration among its constituent components, which may be distributed across a network of nodes. This network of nodes offers an execution environment for the services, and can be organized into different network topologies, e.g. multiple clusters and/or clouds, depending on the usage scenario. Our optimization technique does not facilitate any means to find the optimal topology, but instead will find near optimal placements of components within a given topology. It will however, leverage knowledge about the topology in terms of constraints and requirements, e.g. specifying peak load scenarios and dependability requirements. Changing the execution context might dictate addition or removal of service instances.

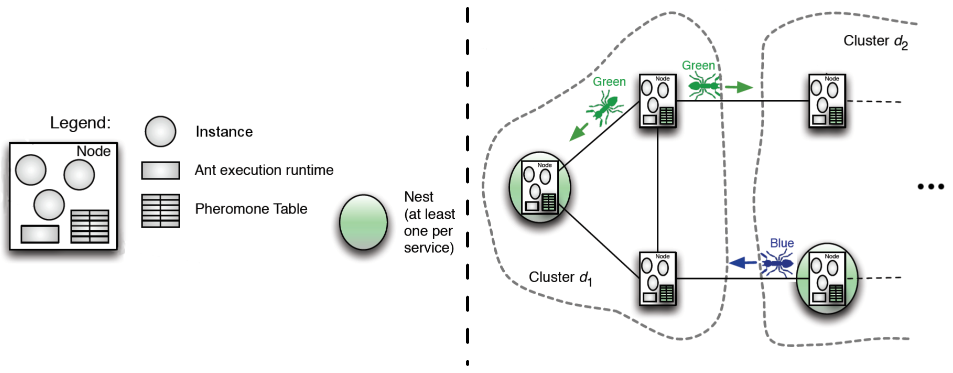


Figure 3: Example target network

We model the system as a collection, $n_i \in \mathbf{N}$, of interconnected nodes. $\mathbf{N}$ is partitioned into a set $\mathbf{D}$ of *clusters*, as illustrated by d_1 and d_2 in Fig. 3. Clusters are usually formed according to geographical location or otherwise distinct administrative region. Let $\mathbf{S}$ denote the set of services to be deployed. The objective from Sec. 1.1 is thus to deploy a set of components $\mathbf{C}$ providing service $S_l \in \mathbf{S}$, and likewise for all services.

As shown in Fig. 3, every node has an *execution runtime* used to support installation, optimization and execution. Furthermore, for every service at least one instance of the CEAS, referred to as *species*, is run. CEAS's autonomous agents searching for a deployment of a given service are shown in different colors in Fig. 3. For each of the species at least one designated node has to be present, serving as a home location for the agents, called the *nest*. A node also maintains information about the goodness of mappings in an information table (called the *pheromone table*, for details see Sec. 3.3), and capacity for installing one or more components of arbitrary services.

There exists a range of software execution frameworks and middleware that are potential candidates for integration with our deployment logic. For example, in case of collaborating software components one such candidate is the MUSIC middleware platform [RBL+08] that supports some self-* properties and component based software. When dependability aspects are of relevance, fault tolerant and self-repairing systems such as the DARM platform [MG08] can take advantage of our deployment logic. Finally, we mention a cloud computing scenario, where the deployment logic is used to optimize placement of VM instances. Candidate platforms include, e.g. Amazon EC2 [LLC] and VMware. Nevertheless, the aim of the deployment logic is to be agnostic to the execution environment, i.e. optimization of the mappings can be done based on service models, regardless of the underlying platform. Example scenarios relevant to these execution environments will be presented in Sec. 4, followed by an evaluation based on simulations. Having introduced our targeted architectures, we now shift our attention to the construction of cost functions and how they are used throughout the optimization process. But first, we summarize our nomenclature in Table 1.

Table 1: Nomenclature

Shorthand	Usage	Size	Description
$\mathbf{S}$	$S_l \in \mathbf{S}$	$\lvert\mathbf{S}\rvert$	Set of services to deploy
$\mathbf{C}$	$c_i \in \mathbf{C}$	$\lvert\mathbf{C}\rvert$	Set of components in S_l
$\mathbf{K}$	$k_j \in \mathbf{K}$	$\lvert\mathbf{K}\rvert$	Set of collaborations in S_l
$\mathbf{N}$	$n \in \mathbf{N}$	$\lvert\mathbf{N}\rvert$	Set of all existing nodes
$\mathbf{D}$	$d \in \mathbf{D}$	$\lvert\mathbf{D}\rvert$	Set of all existing clusters
M	$m_{n,r} \in M_r$	$\lvert M\rvert=\lvert\mathbf{C}\rvert$	Mapping $\mathbf{C} \rightarrow \mathbf{N}$ in iteration r
D	$d \in D_r$	$\lvert D\rvert \leq \lvert\mathbf{D}\rvert$	List of clusters used in M
H	$n \in H_r$	$\lvert H\rvert \leq \lvert\mathbf{N}\rvert$	Hop-list of nodes visited
L	$l_{n,r} \in L_r$	$\lvert L\rvert = \lvert\mathbf{H}\rvert$	Load samples taken in iteration r

2. The Cost Function

We now discuss the construction of cost functions for our optimization technique. A cost function, denoted $F()$, aims to evaluate the utility of a certain deployment configuration, and is used in each iteration of CEAS. Cost function design constitutes an important part of our work, and it is crucial to our technique to capture all aspects of the optimization problem. It has significant influence on the quality of the solutions and on the convergence rate. Recent work on autonomic computing also uses utility functions [KD07], however, our decentralized approach demands more sophisticated functions due to lack of global knowledge at the decision logic. Within the application scenarios considered the cost values that describe a given deployment are to be minimized, i.e. the objective becomes *min* $F()$.

The cost function used by our technique is configured as a combination of several functions, typically one for each requirement/constraint dimension. Thus, the difficulty of efficient design is multifaceted. One is to find the appropriate granularity for each requirement/constraint dimension of a service. Furthermore, ensuring fast convergence dictates keeping the functions as simple as possible, yet the output of the functions must be fine grained enough to distinguish between two significantly different mappings. In some cases, an abundance of possible mappings exists with different but very similar qualities, which may lead to a non-linear cost function. Non-linearities can render their evaluation computationally expensive, causing a significant slow down in execution of the logic. Particular care has to be taken to weld components of the cost function together yielding a single function that is computationally effective and, at the same time, represents all the QoS requirements weighted by their importance. To achieve this the most common combinations are multiplicative or additive combination of cost function components. We begin with a simple function that uses some global knowledge and considers deployment of a single service and then extend the function gradually.

2.1 Load-balancing and Communication Costs

In the first cost function, we consider only execution and communication costs, and we use this to build more complex functions. Assume the deployment logic has access to a service model specifying execution costs, e_i for each component $c_i \in \mathbf{C}$, and communication costs, f_j for each collaboration $k_j \in \mathbf{K}$. The total offered execution load for a given service is then $\sum_{i=1}^{|\mathbf{C}|} e_i$. Hence, the average load T in a network $\mathbf{N}$ becomes

$$T = \left\lfloor \frac{\sum_{i=1}^{|\mathbf{C}|} e_i}{|\mathbf{N}|} \right\rfloor \tag{1}$$

To quantify remote communication costs we first introduce the indicator function $I(j)$, where $I(j) = 1$ if collaboration k_j is remote and $I(j) = 0$ if k_j is internal to a node. That is, we assume the cost of node internal communication is negligible ($I(j) = 0$), and thus only consider the execution cost needed by these components, whereas for remote communication both costs are accounted for. To determine which collaboration k_j is remote the set of mappings, M is used. Given $I(j)$, the remote communication cost, $\Omega(M)$, covering all collaborations of the service is simply the sum

$$\Omega(M) = \sum_{j=1}^{|\mathbf{K}|} I(j) \cdot f_j \tag{2}$$

Thus, the combined cost function becomes

$$F_1(M) = \sum_{n=1}^{|\mathbf{N}|} |\hat{l}_n - T| + \Omega(M) \tag{3}$$

where $\hat{l}_n$, $n = 1 \ldots |\mathbf{N}|$ represent CPU load samples, and M is the mappings to evaluate. That is, load samples describe the execution load impact of the components mapped to a given node n, i.e. $\sum_i e_i$, for $\forall i$ where $c_i \rightarrow n$.

2.2 Multiple Services and Eliminating Global Knowledge

So far we covered QoS requirements captured in the modeling phase of a single service. Next, we extend the function to multiple services and eliminate the need for global knowledge.

Finding T in (3) requires global knowledge of all the services deployed simultaneously. To eliminate the need for this global knowledge, we simply replace T with a set of load samples $l_n \in L$. To capture this notion of load sampling, we introduce a reservation mechanism in the optimization process. When components of a service are mapped to a node n_i, resources equivalent to the weights of the corresponding components will be reserved; the reservations are maintained in the actual nodes. CEAS can use this mechanism to estimate the resource usage on nodes, and to facilitate interaction between the different species. Agents in CEAS can then sample the current reservations on a node, and use this to evaluate the cost of a mapping involving that node. Samples that would exceed the capacity of a node are quickly outranked by better solutions as a high penalty is assigned to infeasible mappings. To keep the resource reservations current, we also add a timestamp-based eviction mechanism to prevent stale reservations.

To further improve scalability, only a portion of the network is considered, represented by the hop-list H, such that $|H| \leq |\mathbf{N}|$; the hop-list correspond to the set of nodes visited to obtain load samples. Thus, the upper bound of the first summation in (3) is reduced to $|H|$. In practice, this means that we are able to achieve near-optimal results without having to sample all $|\mathbf{N}|$ nodes in the network. This sampling scheme is applied to our example scenarios in Sec. 4.2-4.4.

To enable simultaneous deployment of multiple services, one species is run for each service. These species interoperate to obtain a more holistic view of their environment. To facilitate this cooperation, each species store information at participating nodes. The objective of each species is to find a satisfactory mapping for the components of its service, while accounting for services being deployed concurrently.

Formulating a cost function partly depends on the parameters used; we use the deployment mappings in M and the load samples L. We reformulate the cost function as a multiplicative function instead of the additive version, presented in (3), as follows in (4). For more on experiments with additive and multiplicative functions we refer to [CMHH09].

$$F_2(M,H,L) = \Big[\sum_{\forall n \in H} C_0[n]\Big] \cdot (1 + \omega \cdot \Omega(M)) \tag{4}$$

where ω is a scaling parameter for $\Omega(M)$. $F_2(M,H,L)$ has a component, $\Omega(M)$ identical to (2), incorporating the communication costs individually for each service. Function $C_0[n]$, defined in (5), quantifies the node local costs for node n. Samples are obtained over a subset $H \subseteq \mathbf{N}$. This function targets load balancing among nodes.

$$C_0[n] = \Big(\sum_{i=0}^{L[n]} \frac{1}{\Theta_0 + 1 - i}\Big)^2 \tag{5}$$

where Θ_0 is the sum of load samples $\Theta_0 = \sum_{\forall n \in H} L[n]$. The first term in (4) counteracts the term for communication costs, $\Omega(M)$. The effects of these two counteracting terms can be balanced using the scaling parameter ω. The quadratic nature of C_0 allows shifting focus from minimizing the communication costs to load balancing.

2.3 Dependability Rules

We now turn our attention to dependability requirements; each component of a service may be replicated for fault tolerance and/or load-balancing. We also refine our view of the network by introducing *clusters* of nodes, $\mathbf{D}$. Each cluster may represent separate geographic sites. In this context, the aim of the deployment logic is to satisfy the dependability requirements specified in the service model and obtain an efficient mapping in the network, for an example see Sec. 4.3.

To support this scenario, the cost function must also accommodate the additional requirements. These requirements are specified as a set of dependability rules, denoted

Φ, that constrain the minimization problem as *min* $F()$ subject to Φ. We define Φ with the aid of two mapping functions (that apply to a given service k).

DEFINITION 1 *Let $f_{i,d} : c_i \to d$ be the mapping of replica c_i to cluster $d \in \mathbf{D}$.*

DEFINITION 2 *Let $g_i : c_i \to n$ be the mapping of replica c_i to node $n \in \mathbf{N}$.*

The first rule, ϕ_1 requires replicas to be dispersed over as many clusters as possible, aimed at improving service availability despite potential network partitions. We assume that network partitions are more likely to occur between cluster boundaries. More specifically, replicas of a component should be placed in different clusters. If the replication degree exceeds the number of available clusters, at least one replica should be placed in each cluster. The second rule, ϕ_2, simply prohibits collocation of two replicas on the same node. Formally,

RULE 1 $\phi_1 : \forall d \in \mathbf{D}, \forall c_i \in \mathbf{C} : f_{i,d} \neq f_{u,d} \Leftrightarrow (i \neq u) \wedge |\mathbf{C}| < |\mathbf{D}|$

RULE 2 $\phi_2 : \forall c_i \in \mathbf{C} : g_i \neq g_u \Leftrightarrow (i \neq u)$

Let $\Phi = \phi_1 \wedge \phi_2$ be the set of dependability rules considered. Note that, prohibiting collocations by ϕ_2 is contradictory to minimizing remote communication. Thus, when considering dependability we omit communication costs, e.g. in Sec. 4.3 and 4.4. To cater for these new rules, the number of utilized clusters, referred to as $|D|$, is added as a parameter to the cost function. Several combinations of reciprocal and linear functions were evaluated in [CMHH09]. The function that gave the best results is a combination of a reciprocal term targeting ϕ_1 – using $|D|$, and the load-balancing function in (5), applied in two different ways. First, we redefine (5) applicable for each node n – previously used solely for load-balancing – to cater for ϕ_2 as well. The redefined function, (6) is applied in two different ways depending on the parameter $x \in \{0, 1\}$. C_x is a list of values, containing one element for each node covered in an iteration of CEAS (listed in H).

$$C_x[n] = \Big(\sum_{i=0}^{\vartheta_x[n]} \frac{1}{\Theta_x + 1 - i} \Big)^2 \tag{6}$$

For $x = 1$, load samples L are used, accounting for all concurrently executing services on the nodes sampled in L. $x = 0$ in turn represents solely the mappings, M, taking into account the load imposed by the components that are part of a given service. The two usages differ in the upper-bound of the summation and the constant in the denominator, ϑ_x and Θ_x respectively, defined in (7) and (8).

$$\vartheta_x[n] = |m_n| \cdot w + x \cdot L[n] \quad \text{for } x \in \{0, 1\} \tag{7}$$

$$\Theta_x = \sum_{\forall n \in H} \vartheta_x[n] \quad \text{for } x \in \{0, 1\} \tag{8}$$

Θ_x represents the overall execution load of one service ($x = 0$) or all services ($x = 1$). Above, we assume that all replicas have the same weight, denoted w, hence their load can be assessed by multiplying with the number of replicas mapped to a given node, $|m_n|$. Accordingly, it is ϑ_x where parameter M of the cost function is used in this setting. This definition can easily be changed to support individual replica weights. In summary, Θ_0 is the total processing resource demand of one service, whereas Θ_1 accounts for the added load of replicas of *other* concurrent components. In L we account only for those instances that are mapped to the nodes covered in a given iteration (given in H), and as such have reserved processing power for themselves. This way, the list $C_x[n]$ provides a quadratic approximation of the share of load associated with each node as experienced by CEAS, an approximation only as CEAS does not have an exact global overview over the total offered load. Thus, the overall cost function used for dependability becomes

$$F_3(D,M,H,L) = \frac{1}{|D|} \cdot \sum_{\forall n \in H} C_0[n] \cdot \sum_{\forall n \in H} C_1[n] \tag{9}$$

On the one hand, C_0 applies solely to replicas of one service, this way penalizing the violation of ϕ_2, or in other words favoring mappings where replicas are not collocated. On the other hand, C_1, is used for general load-balancing and, as such, it takes into account load imposed on nodes by the other services in the network. Using these separate terms we are able to smoothen the output of the cost function used in each iteration, purposefully easing convergence by making the solution space more fine grained, i.e. simplifying differentiation between very similar deployment mappings with nearly the same cost. Scenarios involving dependability are presented in Sec. 4.3.

2.4 Cluster Costs

In the next scenario, we assume there are different financial costs associated with using different clusters, see for example Sec. 4.4. Hence, we now wish to find deployment configurations that can also minimize the financial cost, while maintaining load balancing and the dependability requirements. To facilitate this, we extend the function in (9) with another term. First, let Λ be the financial cost of using the nodes mapped in M, defined as $\Lambda = \sum_{\forall n_i \in M} |n_i|$, where $|n_i|$ is the financial cost of using node $n_i \in M$. Thus our new function becomes

$$F_4(D,M,H,L,z) = F_3(D,M,H,L) \cdot (1 + g(z)), \tag{10}$$

where $g(z)$ comes in two variants using a scaling parameter z and Λ

$$g(z) = \begin{cases} z \cdot \Lambda & \text{linear weighting} \\ 1 - e^{-(z \cdot \Lambda)^2} & \text{exponential weighting} \end{cases} \tag{11}$$

Setting $z = 0$ eliminates the financial costs, returning to the original function in (9). The choice of z depends on the cost of using the different clusters. The two alternatives in (11) represent a linear and an exponential increment in financial costs, when $z > 0$. When applying the more fine-grained exponential weighting to the cost function, we expect to observe more balanced mappings, avoiding under-utilization or overloading of clusters.

Next we present our proposed deployment logic, along with the CEAS optimization method and associated algorithms. The cost functions presented in this section are used in these algorithms to evaluate the deployment configurations.

3. The Deployment Logic

To solve the deployment mapping problem presented in Sec. 1.1 we use the CEAS method introduced by Helvik and Wittner [HW01]. CEAS is an agent-based optimization framework, in which the agents' behavior is inspired by the foraging patterns of ants. The key idea in CEAS is to let many agents, denoted *ants*, search iteratively for the solution to a problem taking into account the constraints and a predefined cost function. Every iteration consists of two phases. In the *forward search* phase, ants search for a possible solution, resembling the search for food in real-world ants. The second phase is called *backtracking*, in which ants – after evaluating the solution found during forward search – leave markings, called *pheromones*, that are in proportion to the quality of the solution. Pheromones are then distributed at different locations in the search space and can be used by forward ants in their search for improved solutions. Therefore, the best solution will be approached gradually. To avoid getting stuck in premature and sub-optimal solutions, some of the forward ants explore the search space, ignoring the pheromones. There is a principal difference, however, between the various existing ant-based systems and the approach taken in CEAS in evaluating the solution and in pheromone updates. CEAS uses the *Cross Entropy (CE) method* for stochastic optimization introduced by Rubinstein [Rub99]. The CE method is applied during the pheromone updating process, gradually changing the probability matrix $\mathbf{p}_r$ according to the cost of the solution found in iteration r. Then, the objective is to minimize the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. For a tutorial on the method, [Rub99] is recommended. Next we present the CEAS method in more detail, followed by a generalized version of our deployment algorithm, which was applied in the evaluations in Sec. 4. Minor algorithmic differences between the different scenarios are also discussed.

3.1 The Cross Entropy Ant System

We apply CEAS to obtain efficient deployment mappings in the form $M : \mathbf{C} \rightarrow \mathbf{N}$ between sets of components, $\mathbf{C}$, and sets of nodes, $\mathbf{N}$. Ants move between nodes across network links in search for nodes with hosting capacities. The cost of mappings is evaluated using the cost functions discussed in Sec. 2, i.e. applying them as $F(M)$ in every iteration of CEAS. The pheromone values, $\tau_{mn,r}$, in CEAS for deployment mapping are assigned to the component set m deployed at node n at iteration r. The *random proportional rule* (**rpr**) in (12) is used to select deployment mappings. That is, during normal forward search, a set of components is selected according to the **rpr** matrix $p_{mn,r}$

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in M_{n,r}} \tau_{ln,r}} \tag{12}$$

A temperature parameter γ_r, controls the update of the pheromone values and is chosen to minimize the performance function

$$H(F(M_r), \gamma_r) = e^{-F(M_r)/\gamma_r} \tag{13}$$

which is computed for all r iterations such that the expected overall performance satisfies

$$h(p_{mn,r}, \gamma_r) = E_{\mathbf{p}_{r-1}}(H(F(M_r), \gamma_r)) \geq \rho \tag{14}$$

$E_{\mathbf{p}_{r-1}}(X)$ is the expected value of X s.t. the rules in $\mathbf{p}_{r-1}$, and ρ is a *search focus* parameter close to 0 (typically 0.05 or less). Finally, a new updated set of rules, $\mathbf{p}_r$, is determined by minimizing the cross entropy between $\mathbf{p}_{r-1}$ and $\mathbf{p}_r$ with respect to γ_r and $H(F(M_r), \gamma_r)$.

To avoid centralized control and synchronized batch-oriented iterations the cost value $F(M_r)$ is calculated *immediately* after each sample, i.e. when all components are mapped, and an auto-regressive performance function, $h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1 - \beta)H(F(M_r), \gamma_r)$ is applied. This function is approximated by

$$h_r(\gamma_r) \approx \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(M_r), \gamma_r) \tag{15}$$

where $\beta \in \langle 0, 1 \rangle$ is a *memory factor*, used for weighting (geometrically) the output of the performance function. The temperature γ_r in turn is determined by minimizing it subject to $h(\gamma) \geq \rho$. The temperature furthermore is equal to

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(M_i), \gamma) = \rho\} \tag{16}$$

which is a complicated (transcendental) function that is both storage and processing intensive since all observations up to the current sample, i.e. the entire mapping cost history $\{F(M_1), \cdots, F(M_r)\}$ must be stored, and weights for all observations have to be recalculated [HW01]. This can be a prohibitively large resource demand, especially in online nodes. As a resolution we assume, given a $\beta \approx 1$, that the changes in γ_r are typically small from one iteration to the next, which enables a first order Taylor expansion of (16) as follows

$$\gamma_r = \frac{b_{r-1} + F(M_r)e^{-F(M_r)/\gamma_{r-1}}}{(1+\frac{F(M_r)}{\gamma_{r-1}})e^{-F(M_r)/\gamma_{r-1}} + a_{r-1} - \rho\,\frac{1-\beta r}{1-\beta}} \tag{17}$$

where $a_0 = b_0 = 0$ and $\gamma_0 = -F(M_0)/\ln\rho$. Furthermore,

$$\begin{aligned} a_r &\leftarrow \beta(a_{r-1} + (1 + \frac{F(M_r)}{\gamma_r})e^{-\frac{F(M_r)}{\gamma_r}}) \\ b_r &\leftarrow \beta(b_{r-1} + F(M_r)e^{-\frac{F(M_r)}{\gamma_r}}) \end{aligned} \tag{18}$$

where the performance function, (13), is adopted. The pheromone values in CEAS are a function of the *entire history* of mapping cost values, hence CEAS has what is denoted a search history dependent quality function [Z$^+$04]. Updates to the pheromone values are made by applying the performance function, (13), combining the last cost value $F(M_r)$ and the temperature γ_r, calculated by (17). Pheromones are updated as follows.

$$\tau_{mn,r} = \sum_{k=1}^{r} I((m,n) \in M_k)\beta^{\sum_{x=k+1}^{r} I((m,\cdot)\in M_x)} H(F(M_k), \gamma_r) \tag{19}$$

The *memory factor*, β, supplies geometrically decreasing weights to the output of the performance function, enabling evaporation of pheromones. The exponent of β is somewhat complex since ants during *backtracking* do not update all nodes in the network, only those nodes that were visited during the preceding forward phase. The exponent in (19) represents the number of ants that have updated node n between time-step r and k when a mapping M_k was found, while $r-k$ is the total number of updates in the system, i.e. total number of ants that returned between time-step r and k. Hence $r-k \geq \sum_{x=k+1}^{r} I((m,\cdot) \in M_x)$. However, as for (16), excessive processing and storage requirements also apply for (19). A (second order) Taylor expansion of (19) is appropriate, giving

$$\tau_{mn,r} \approx I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}} + A_{mn} + \begin{cases} -\frac{B_{mn}}{\gamma_r} + \frac{C_{mn}}{\gamma_r^2} & \frac{1}{\gamma_r} < \frac{B_{mn}}{2C_{mn}} \\ -\frac{B_{mn}^2}{4C_{mn}} & \text{otherwise} \end{cases} \tag{20}$$

where

$$\begin{aligned} A_{mn} &\leftarrow \beta(A_{mn} + I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}}(1 + \tfrac{F(M_r)}{\gamma_r}(1 + \tfrac{F(M_r)}{2\gamma_r}))) \\ B_{mn} &\leftarrow \beta(B_{mn} + I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}}(F(M_r) + \tfrac{F(M_r)^2}{\gamma_r})) \\ C_{mn} &\leftarrow \beta(C_{mn} + I((m,n) \in M_r)e^{-\frac{F(M_r)}{\gamma_r}}(\tfrac{F(M_r)^2}{2})) \end{aligned} \tag{21}$$

The initial values for (21) are $A_{mn} = B_{mn} = C_{mn} = 0$ for all (m,n). For a stepwise explanation of the Taylor expansion and how it is applied we refer to Appendix A in [Wit03]. Further improvements in the scalability of CEAS are described in [HW10].

3.2 The Deployment Algorithm

In this section, we present our general deployment algorithm and explain how it is executed. First, the task of obtaining deployment mappings for a given service is assigned a species – a given type – of ants via the service model that can be interpreted by the logic. The service model contains the set of components, **C**, to be mapped. Ants are emitted (cf. Algorithm 1) continuously and select nodes to visit depending its type. There are two types of ants, explorer and normal ants. *Normal ants* select a subset of **C** (can be $\emptyset$) at every node they visit based on the content of the local pheromone table at the node, whereas *explorer ants* select a subset based on a random decision, ignoring the pheromones. The selections made by the ant are stored in M, and carried along with the ant; they represent a deployment mapping made during one iteration of the algorithm.

Initially, only explorer ants are used to explore and cover a significant portion of the mapping problem space by random sampling. The length of initial exploration depends on the problem size, in terms of network size and number of services. After initial exploration, the majority of ants are normal ants, while a smaller fraction are explorers, typically 5-10 percent. This continued exploration is meant to capture fluctuations in the network, e.g. new nodes connecting, and thereby improve responsiveness to dynamism in the environment. The normal ants try to find an optimal mapping.

We also distinguish between two phases in one iteration of the algorithm. In every iteration the ant starts with *forward search* and completes the iteration with *backtracking*. During *forward search* the ant obtains a mapping M, i.e. a suggested deployment for the service, which can then be evaluated by $F(M)$. This ends the first phase and *backtracking* can start, which consists of revisiting the nodes visited in the first phase – according to the hop-list H – and updating the pheromone databases in those nodes. This ends an iteration and a new ant can be emitted, starting a new iteration of the algorithm, unless a stopping criteria is met. Usually however, the algorithm will continue to explore the network for improved mappings; enabling adaptation to changes (reconfiguration) in the execution context.

Generally, we have a trade-off between convergence speed and solution quality. Nevertheless, while deploying services in a dynamic environment, a premature solution that satisfies both functional and non-functional requirements often suffices. ACO systems have been proven to find the optimum at least once with a probability close to one, and after that convergence to the optimum is secured in a finite number of iterations [SD02]. Since CEAS can be considered as a subclass of ACO the optimal deployment mapping will eventually emerge.

Algorithm 1 Code for $Nest_k$ corresponding to service S_l at any node $n \in \mathbf{N}$

```
 1: Initialization:
 2:     $r \leftarrow 0$                                   {Number of iterations}
 3:     $\gamma_r \leftarrow 0$                                   {Temperature}
 4: while $\infty$                                         {Stopping criteria can be applied here}
 5:     $M \leftarrow antAlgo(l,k)$                        {Emit new ant for service l from Nest k, obtain M}
 6:     $update(availableClusters)$                        {Check the number of available clusters}
 7:     if $splitDetected() \vee mergeDetected()$
 8:         $release(S_l)$                                 {Delete existing bindings for all instances $c_i \in C_l$}
 9:     if $\Phi(M, availableClusters)$
10:         $bind1(M)$                                     {Bind one of the still unbound instances in $C_l$}
11:     $r \leftarrow r+1$                                 {Increment iteration counter}
```

Improved dependability of the approach can be obtained by means of replicated ant *nests*. The same ant species may be emitted from multiple nests, providing resilience to node failures affecting the node hosting a nest. Note that, this nest replication does not lead to flooding of the network with ants, as the rate of ant emission in a stable network can be divided equally among nests. Moreover, ants emitted from different nests, but associated with the *same service*, will operate on the *same pheromone table entries* in the nodes they visit. During execution of CEAS, synchronization between nests is not necessary. However, a primary nest must make the final deployment decision, triggering physical placement of the components.

In contrast to CEAS used for routing, where the temperature is stored in the destination node, our CEAS implementation has no notion of destination for the deployment mapping. Instead a mapping, M, is distributed over a set of nodes. Yet, ants are able to find the same mapping M, while visiting the same set of nodes, possibly in a different order, and making the same mapping decision. To provision for this capability, ants returning to their nest at the end of the *backtracking* phase, will pass on the temperature parameter to their immediate successor ants.

Algorithm 2 Ant code for mapping service S_l

```
1: Initialization:
2:     $H_r \leftarrow \emptyset$                                  {Hop-list; insertion-ordered set}
3:     $M_r \leftarrow \emptyset$                                  {Deployment mapping set}
4:     $D_r \leftarrow \emptyset$                                  {Set of utilized clusters}
5:     $L_r \leftarrow \emptyset$                                  {Set of load samples}
6: function $antAlgo(r,k)$
7:     $\gamma_r \leftarrow Nest_k.getTemperature()$               {Read the current temperature}
8:     foreach $c_i \in \mathbf{C}$                                {Maintain bound mappings}
9:         if $c_i.bound()$
10:            $n \leftarrow c_i.boundTo()$                        {Jump to the node where this comp. is bound}
11:            $n.reallocProcLoad(S_l, e_i)$                       {Allocate processing power needed by comp.}
12:            $l_{n,r} \leftarrow n.getEstProcLoad()$             {Get the estimated processing load at node n}
13:            $L_r \leftarrow L_r \cup \{l_{n,r}\}$               {Add to the list of samples}
14:    while $\mathbf{C} \neq \emptyset$                           {More instances to map}
15:        $n \leftarrow selectNextNode()$                         {Select next node to visit}
16:        if explorerAnt
17:            $m_{n,r} \leftarrow random(\subseteq \mathbf{C})$   {Explorer ant; randomly select a set of comps.}
18:        else
19:            $m_{n,r} \leftarrow rndProp(\subseteq \mathbf{C})$  {Normal ant; select comps. according to (12)}
20:        if $\{m_{n,r}\} \neq \emptyset, n \in d_k$              {At least one comp. mapped to this cluster}
21:            $D_r \leftarrow D_r \cup d_k$                       {Update the set of clusters utilized}
22:        $M_r \leftarrow M_r \cup \{m_{n,r}\}$                   {Update the ant's deployment mapping set}
23:        $\mathbf{C} \leftarrow \mathbf{C} - \{m_{n,r}\}$        {Update the set of replicas to be deployed}
24:        $l_{n,r} \leftarrow n.getEstProcLoad()$                 {Get the estimated processing load at node n}
25:        $L_r \leftarrow L_r \cup \{l_{n,r}\}$                   {Add to the list of samples}
26:    $cost \leftarrow F(D_r, M_r, H_r, L_r)$                     {Calculate the cost of this given mapping}
27:    $\gamma_r \leftarrow updateTemp(cost)$                      {Given cost, recalculate temperature according to (16)}
28:    foreach $n \in H_r.reverse()$                               {Backtrack along the hop-list}
29:        $n.updatePheromone(m_{n,r}, \gamma_r)$                  {Update pheromone table in n}
30:    $Nest_k.setTemperature(\gamma_r)$                           {Update the temperature at $Nest_k$}
```

We clearly distinguish between the notions of component *mapping*, *binding* and *deployment*. The *mapping* M is a variable list constantly optimized, iteration by iteration by the logic only visible internally to the algorithm. When an instance is *bound* to a node that means that the particular mapping for that instance is not changed anymore by the ants until that binding is erased again. Lastly, by *deployment* we refer to the physical placement and instantiation of an instance on a node, which is triggered after the mapping M for the given service has converged to a satisfactory solution. The latter property ensures that there is no undesirable fluctuation in the migration of replicas using our method.

Improving convergence is the concept of *binding* of components, which allows nests to *fix* one instance in the latest mapping M obtained by the ants, if some condition applies. For example, we have a condition that checks if the mapping M satisfies Φ as condition of a *bind* event. After a *bind* ants for the same service do not change the fixed mapping in subsequent iterations and new searches will be conducted for the remaining instances only. Importantly however, bound instances are also taken into account when the cost of the total mapping is evaluated. Should a split or a merge event occur in the network these bindings are erased in the ant nest and the total amount of instances will be taken into consideration in the following searches. In

Algorithm 1 and Algorithm 2 we present the version of the deployment algorithm that uses binding as well as guided random hopping (Algorithm 3), which we discuss next.

First, an ant visits the nodes, if any, that already have a bound instance mapped to, in order to maintain these mappings. These will also be taken into account when the cost of the total mapping is evaluated. The pheromones corresponding to bound mappings will also be updated during *backtracking*. Ants allocate processing power corresponding to the execution costs of the bound replicas, derived from the service specification. This first phase of an ant's tour is denoted *maintenance*. After this phase, ants turn to guided random hop-selection.

The selection of the next node to visit, in contrast to e.g. ant-based routing algorithms, is independent from the pheromone markings laid by the ants. Pheromone tables are used only for selecting components or replicas to map to nodes. The advantage of using the guided selection shown in Algorithm 3 as opposed to a pure random walk lies in that with the proper guidance, the frequency of finding an efficient mapping is higher. In case of replica management, for example, idea is that at first the next node is selected from a cluster that has not yet been utilized until all visible clusters are covered, leading to better and faster satisfaction of ϕ_1 (see Sec. 2). Then, next hop selection continues with drawing destinations from the set of nodes not yet used in the mapping. This can be done by checking with the variable M, before reverting to totally random drawing.

Algorithm 3 Next-hop selection procedure for an ant

```
1:  function $selectNextNode()$                      {Guided random jump}
2:    if $H = \mathbf{N}$                                {All nodes visited}
3:      $n \leftarrow random(\mathbf{N})$              {Select candidate node at random}
4:    else
5:      if $D = \mathbf{D}$                             {All available clusters utilized}
6:        $n \leftarrow random(N \setminus M)$          {Select a node that has not been used yet}
7:      else
8:        $d_i \leftarrow random(\mathbf{D} \setminus D)$   {Select a cluster not yet used}
9:        $n \leftarrow random(d_i)$                    {Select a node within this cluster}
10:   $H \leftarrow H \cup \{n\}$                        {Add node to the hop-list}
11:   return $n$
```

The guided hop-selection algorithm can be used to different extents, i.e. in case of simple settings without replication management where the network is not partitioned into clusters this function simply can be reverted to random drawing, e.g. the scenarios presented in Sec. 4.1 and 4.2. Whereas when replication is present and dependability rules, Φ, have to be satisfied, the guidance and taboo-listing can be turned on. Next, we discuss how the data in pheromone tables can be represented efficiently.

3.3 Encoding Pheromone Values

Optimization governed by the cost function starts with aligning pheromone values with the sets of deployed components and defining the structure of the pheromone database for the ants. With the underlying set of nodes each ant will form $|\mathbf{N}|$ discrete sets from the set of available components ($\mathbf{C}$) that need to be deployed and will evaluate the outcome of that deployment mapping at the end of each iteration. In the simplest encoding, we assign a flag to each of the component instances and build a bitstring for a service of size $2^{|\mathbf{C}|}$. This way a single pheromone database instance located at a node becomes equal in size to the number of possible combinations for forming a subset of **C**. After normalizing the pheromones in a node we can observe the probability distribution of component sets mapped to that particular node by the ant system. Eventually, after convergence the suggested solution emerges in the distributed pheromone database with probability near to one.

In case of a *normal* ant the selection process for selecting a set of instances to map to a node depends on the form of the pheromone tables, in particular on how the pheromone values are encoded. Appropriate solutions can be found using different encodings, however, there are differences in terms of convergence times and solution quality. Efficient encodings are required for scalability of the logic as well. In [CMH09] we proposed and evaluated three different pheromone encodings. Generally, pheromone tables can be viewed as a distributed database with elements located in each node available in the network considered for deployment. Entries in the database have to be able, on one hand to describe arbitrary combinations of components or replicas. On the other hand, as the distributed database consumes some memory in every node – and the required memory grows both with the amount of services deployed ($|\mathbf{S}|$) and with the sizes of the services ($\mathbf{C}$) –, the size of this database is crucial for scalability. We wish accommodate as many as possible services, thus we have to efficiently manage the memory need, which we can directly influence by choosing an appropriate pheromone encoding. Beside the storage needs an individual ant agent has to browse through the pheromone entries during its visit to a node. Clearly, a more compact pheromone database helps speeding up execution of the tasks it has to perform. The different encodings we proposed and their corresponding sizes are shown in Table 2.

Table 2: Three pheromone encodings for a service with $|C|$ instances

Encoding	DB size in a node	Encoding example w/ $\|C_l\| = 4$
bitstring	$2^{\|C_l\|}$	$[0000]b \ldots [1111]b$
per comp.	$2 \cdot \|C_l\|$	[0/1]; [0/1]; [0/1]; [0/1]
# replicas	$\|C_l\| + 1$	$[0] \ldots [4]$

The first encoding, called *bitstring*, is the largest as it holds a single value for all possible combinations of replica mappings in every node, which can result in prohibitively large memory need. For example, in case of 20 components per service this encoding leads to 2^{20} pheromone values, which by using 4 byte long floating point numbers would require 4 MB of memory for each of such services at every node. To reduce the table size we can apply simpler bookkeeping taking into account solely the number of replicas mapped to a given node, shown in # *replicas*. This is the most compact pheromone entry encoding, however, the tradeoff is that it cannot distinguish between replicas in a service specification, thus it can only be applied if there is no need to distinguish between component replicas, e.g. due to equally sized replicas. As a good compromise in between we have developed the *per comp* encoding that results in no information loss, while still being linear in size. The *per comp* encoding uses one distinct pheromone entry for every instance indicating whether or not to deploy them at a given node. The slight disadvantage is that ants arriving at a node have to decide on the deployment mapping of each replica, one-by-one reading the multiple pheromone entries corresponding to the elements of the service (one separate table element for each). Nevertheless, this encoding provides the necessary reduction in database structure size for allowing scaling up to larger amounts of services and larger service sizes.

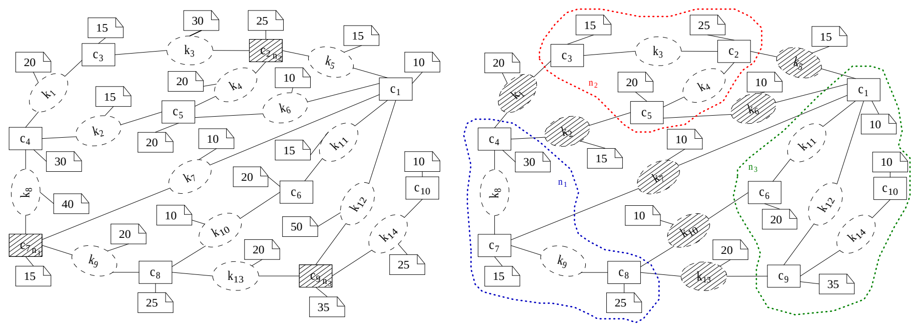
In the following, examples are presented where CEAS and the deployment logic is applied in different scenarios.

4. Example Scenarios

To demonstrate the deployment mappings that can be obtained using our deployment logic 4 different scenarios are presented in the subsequent subsections, followed by a subsection on cross-validation of some of the results by application of a centralized method.

4.1 Deployment of Collaborating Software Components

As a first example we consider a scenario that has been introduced by Efe, originally modeling clustering of modules and cluster assignment to nodes [Efe82]. The same scenario has been investigated by several authors, including Widell et al. who compared the related results in [WN04]. This artificial clustering problem is modeled as a collaboration of components and is used to test the deployment logic. We define Efe's example as a collaboration of $|\mathbf{C}| = 10$ components (labelled $c_1 \dots c_{10}$) to be deployed and $|\mathbf{K}| = 14$ collaborations between them ($k_1, \dots k_{14}$), as depicted in Fig. 4a. Besides, the execution and communication costs, we have a restriction on components c_2, c_7, c_9, regarding their location. They must be placed in nodes n_2, n_1, n_3, respectively.



(a) Collaborations and components in the example (b) Optimum with 3 nodes

Figure 4: Simple example service

In this example, the target environment consists of $|\mathbf{N}| = 3$ identical, interconnected nodes. To gather information continuously the ants employed by the logic sample the CPU load levels, $\hat{l}_n$. We target minimum remote communication, i.e. we take into account communication costs for collaborations between two components if they are not collocated in the same node. At the same time we look for a globally balanced CPU load over all the nodes available. Furthermore, for this first example we have $T \cong 68$ (cf. (1)) as average target load in the 3 nodes.

The optimum solution of the example is depicted in Fig. 4b, with the lowest possible cost value of $17 + 100 = 117$ (cf. (3)). Finding an efficient deployment with the lowest cost is illustrated in Fig. 5. After an initial 2000 *explorer* ants, optimization starts and the overall cost is converging to the optimum value of 117. The size of the dynamically allocated pheromone database – in which a threshold can be applied to regulate the amount of significant entries – can also be observed in the bottom half of the figure. The database size tops at 2^7 as the number of maximum available components is 7 out of the total of 10, with 3 bound components. In case of convergence to a single solution, like in the example at hand, solutions other than the optimum can be evaporated from the database, thus reducing its size if needed, as shown.

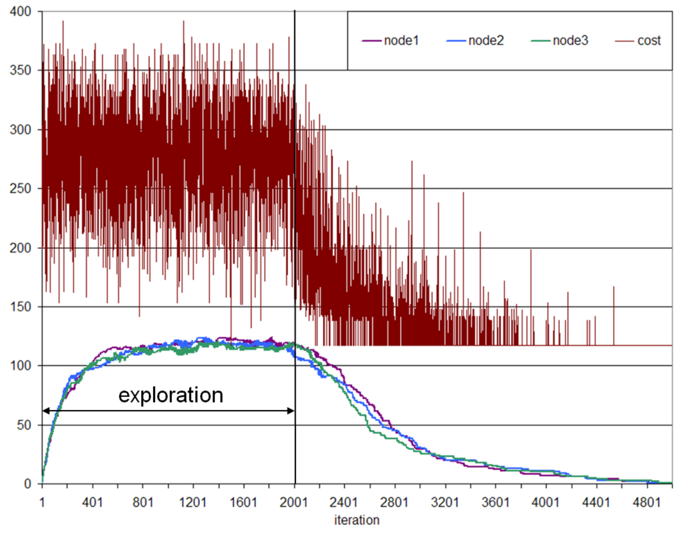


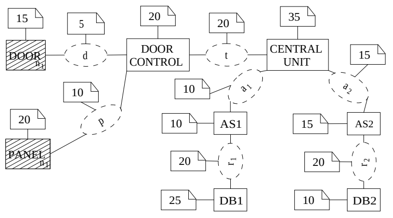
Figure 5: Observed cost and pheromone database sizes

For more details and comparison of the results obtained with the CEAS and other methods using this example we refer the reader to [CHH08b]. Accordingly, we find that our distributed approach is capable of obtaining efficient mappings in NP-hard deployment scenarios. Next, we continue with increasing complexity by considering more services simultaneously in another example setting.

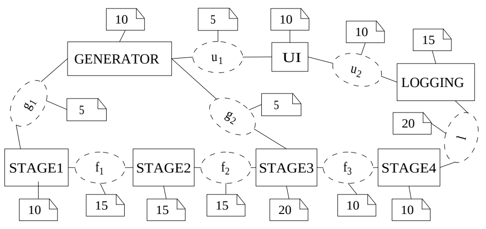
4.2 Multiple Species

In [CHH08a] we introduced three service models for experimental use. The first example has been introduced originally in [KH06]. S_1 operates a security door and a card reader with a keycode entry panel using authentication and authorization servers and databases. The second example, models a video surveillance system. A central control with a recording unit manages the system and uses a main and a backup storage device for storing surveillance information. S_3 is a model of a process controller that consists of 4 main stages of processing, logging, a user interface, and a generator component. The service models are presented in Fig. 6.

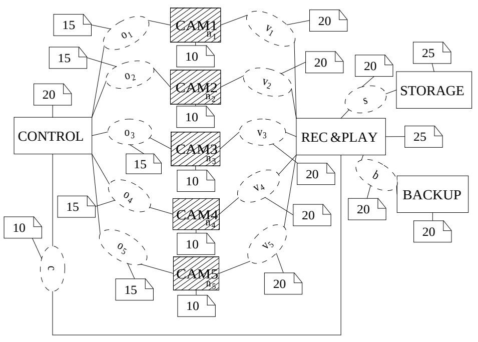
An ant species is assigned to each of the services and deployment mappings are obtained using a network of 5 nodes. This example has multiple optimal and near-optimal solutions with different sets of components deployed on various nodes. In [CHH08a] we tested the effectiveness (e.g. number of iterations required for convergence, average cost of mappings found) of the approach considering that in the multiplicative cost function, (4), we omitted any global knowledge, i.e. we do not use the global shared knowledge T, (1), anymore. Instead the parallel species are aware of each others processing power demand through sampling, i.e. L. We have found that the multiplicative function (4) allows the logic to deploy multiple parallel species equally effectively. Synchronization of the separate ant nests is not necessary, however, only one designated nest shall be allowed to trigger placement.



(a) S_1



(b) S_3



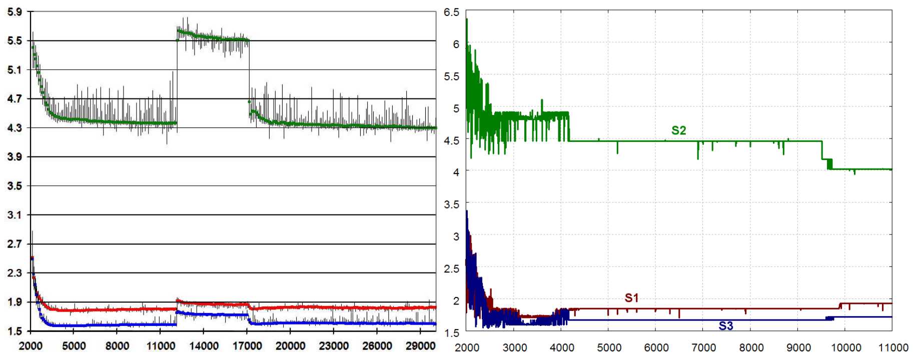
(c) S_2

Figure 6: 3 service examples

To evaluate adaptation capabilities of the logic we investigated two simple scenarios, a single node failure and reparation, and the appearance of a new node in the network. The new node is added after all the species have converged to a solution with 5 nodes.

The evolution of costs (Y-axis) – obtained by (4) – for the 3 services in the example is shown in Fig. 7 as a function of the number of iterations (X-axis). A node error is injected after iteration 12000 followed by a repair approximately 4000 iterations later (Fig. 7a shows average and min-max values). The first 2000 iterations represent the initial *exploration* phase. Due to the abrupt change in the context – a previously used node disappears – costs increase quickly after the failure. Service S_2 suffers most, as indicated by the highly increased costs, mostly due the large communication demand of that service. Deployment costs return to normal again somewhat slower after the node has been repaired and explorer ants discover the new, more useful configuration.

In the second test a new (6th) node is added to the network of 5 nodes after around 5000 iterations, at which point the 3 species are already converged to a stable solution. At this point, 5% of the emitted ants are *explorers*, which eventually (approx. after 9500 iterations) discover the new node and a new, more efficient deployment (Fig. 7b). It is also to be noted that the species agree on a new configuration where services S_1 and S_3 suffer some degradation, i.e. in terms of higher costs, but S_2 has a high gain, thus the overall utility of the new deployment is better. Regarding the number of iterations we note that using conventional exhaustive search methods we would have needed to explore $|\mathbf{N}|^{|\mathbf{C_{S_k}}|}$ possible configurations for a service S_k and we have to add up the numbers for $\forall S_k \in \mathbf{S}$ to represent the problem size. So, first we gain on splitting the problem of deploying multiple services using one type of species for each service simultaneously. Second, we increase computational effectiveness by not having to explore the search space – e.g. of size 5^{24} possible configurations in this example with 3 services – exhaustively.



(a) Node error/repair (b) New node appearing

Figure 7: Costs for the 3 services in two test scenarios

4.3 Deployment of Component Replicas

We focus next on dependability achieved by efficient replication management. Fig. 8 depicts a simple example service model. A component is actively replicated in 4 replicas, R_1 to R_4, with the continuous updating mechanism modeled as collaborations between the replicas. Each replica in the service performs according to the client requests, thus, replicas have the same execution cost.

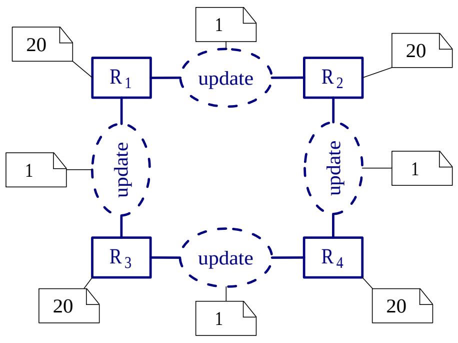


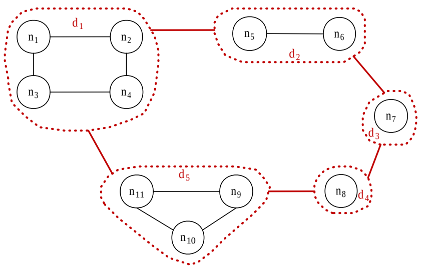
Figure 8: Replicated components in example service S_1

The example, furthermore, consists of 10 services ($S_1 \dots S_{10}$) that are being deployed, using 10 independent species of ants. Also, we use 20 ant nests to be able to look at a simple cluster splitting and merging scenario, with one nest remaining for each of the species in each of the regions formed after the split. Each service S_i, $i = 1 \dots 10$ has a redundancy level (amount of replicas) of $i+1$. The network of nodes for the experiment consists of 11 fully interconnected nodes, partitioned into 5 clusters (Fig. 9a). Simulations of the scenario were conducted in a discrete event simulator custom built and programmed in Simula/DEMOS language.

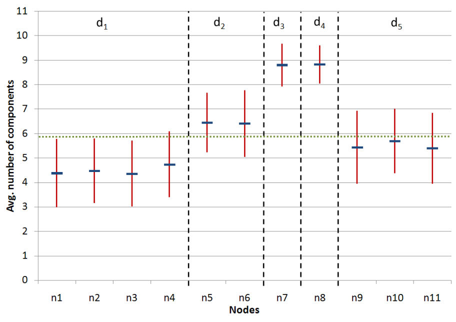
First, we look at the objective of obtaining a balanced deployment mapping with respect to execution load. This objective has to be followed while maintaining the dependability rules of Φ. In Fig. 9b, we look at the average number of replicas mapped to the 11 nodes in the test network. A total amount of 65 identically sized component replicas are mapped, which – in a homogenous, non-clustered network – would give an average of 5.91 replicas per node (shown as a dotted horizontal line). As an effect of cluster disjointness (ϕ_1) smaller clusters, such as d_3, d_4, suffer from overloading, but generally replicas are placed quite evenly across the available nodes, showing that cooperation between the species works. Furthermore, for a larger experiment with 50 nodes and 275 replica instances we refer to [CMH09].

In the same example setting we conducted simulations to test adaptation capabilities of the logic. Three different pheromone encodings are tested in this example, more about the various encodings and their effects in Sec. 3.3. To test capabilities to remedy cluster splitting and merging, cluster d_1 containing 4 nodes is split from the rest of the network at the cluster boundary and some time later it merges back, thus restoring the original network scenario. For example, the cost output in case of service S_{10} is displayed in Fig. 9c. Cluster d_1 splits from the rest after iteration 4000 and we can observe how the swarm adapts and obtains new mappings for a more expensive configuration (increased cost) due to the reduced network. Similarly, after the merge of the two regions, around approximately iteration 5000, mappings are adapted utilizing cluster d_1 again, thus resulting in a lower cost configuration again.

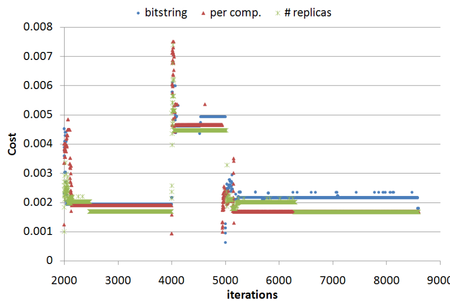
Regarding the number of iterations required for obtaining reasonable and stable



(a) Test network of nodes clustered into 5 clusters



(b) Load-balancing over 11 nodes



(c) Splitting/merging of cluster d_5

Figure 9: Example scenario with 10 services

mappings, e.g. shown on the X-axis in Fig. 9c, we conclude that they are reasonably low and do not increase the overhead significantly, especially compared to exhaustive search, which would require evaluation of 11^{65} possible configurations. Mapping of replicas considering the dependability rules becomes harder when the number of replicas in a service is close to the amount of available nodes. Instead of having hard-constraints that strictly cannot be violated, like e.g. in traditional integer programming, we utilize soft-constraints incorporated into the cost functions we use. The tradeoff might be that sometimes the algorithm prefers a globally lower cost mapping with better overall load-balancing that, however, violates some of the soft-constraints for one service, e.g. for a large service that has as many replicas as many nodes exist, there might be a single collocation (violation of ϕ_2) due to the better utilization of an otherwise under-utilized node.

Besides looking at adaptation, in Fig. 9c we also present how the costs evolve using the three different encodings introduced in Sec. 3.3. After a split and merge event the *bitstring* encoding converges to a solution with slightly higher overall cost than before, whereas the lowest cost is obtained first by *per comp.* and somewhat later by *# replicas*. The first 2000 iterations are not shown as, initially, a random cost figure appears corresponding to exploration that is omitted here. Every simulation starts with 2000 explorer iterations for the sake of comparability, even though the amount of initial exploration was constrained by the *bitstring* encoding. The more compact encodings require significantly less iterations, e.g. one tenth of the amount used. The *bitstring* encoding in this test case is unable to find exactly the same mapping and converges to a somewhat more costly solution. *per comp.* is the fastest to obtain the lowest cost mapping followed by the third encoding about 1000 iterations later.

Table 3: Success rates of the three encodings in the example setting

wo/ splitting	ϕ_1	ϕ_2	w/ splitting	ϕ_1	ϕ_2
bitstring	100%	88%	bitstring	100%	87%
per comp.	100%	100%	per comp.	100%	100%
# replicas	100%	100%	# replicas	100%	99%

Considering the dependability rules $\phi_1 \wedge \phi_2$, Table 3 shows the three different pheromone encodings and the percentage of test cases, which succeeded in satisfying the two rules. The results were obtained by executing the algorithm 100 times for each encoding, with different input seeds. The results indicate that choosing *per comp.* not only provides the best compromise between scaling and descriptiveness but gives more efficient results too.

4.4 VM Instance Placement in Private and Public Clusters

In this section we look at how self-organizing techniques applied for system (re)configuration can be used to improve scalability and dependability of virtualized service systems. Specifically, we discuss deployment of virtual machine (VM) images to physical machines in a large scale network. Simulation results with the decentralized deployment logic are presented for an example cloud computing scenario. For additional details about the deployment scenario see [CMH10].

To demonstrate the behavior of the logic consider the scenario depicted in Fig. 10. The network consists of 5 private clouds (Cloud $C,\ldots,G$) connected to public cloud providers (Cloud A,B) via the Internet. The publicly available capacities can be utilized on demand, but are subject to financial costs. Conversely usage of private clusters is free for a service with a home location in that private cloud. Thus, deploying and hosting a VM instance on a node within one of the clouds implies additional costs; namely a cost of 10 for a node in *Cloud A*, 1 for *Cloud B* and 0 cost for the private clouds $C \ldots G$. The scaling parameter, cf. (11), we applied in the example was $z = 0.1$.

Intuitively, this partitioning and cost assignment models a scenario where organizations naturally execute VM instances within their privately owned cluster as long as the requirements allow, i.e. satisfactory replication levels can be achieved with the available resources in the private clusters. Accordingly, hosting VMs in private clusters is considered free as opposed to hosting in public clouds. Moreover, in the example a trade-off exists between a large cloud provider with several clusters and plenty of nodes available for placement, which is more expensive to use than paying for hosting in the smaller cloud offering less resources.

Simulations execute the deployment task of mapping 125 services, i.e. 25 in each private cloud, using public clouds for deployment on demand. Without allowing usage of resources in private clusters other than the originating ones. This is practically achieved by assigning ∞ cost for neighboring private clouds. Each of the 125 services consist of 5 VM instances – among others for dependability reasons – that have to be deployed, thus resulting in the task of deploying a total number of 625 VMs.

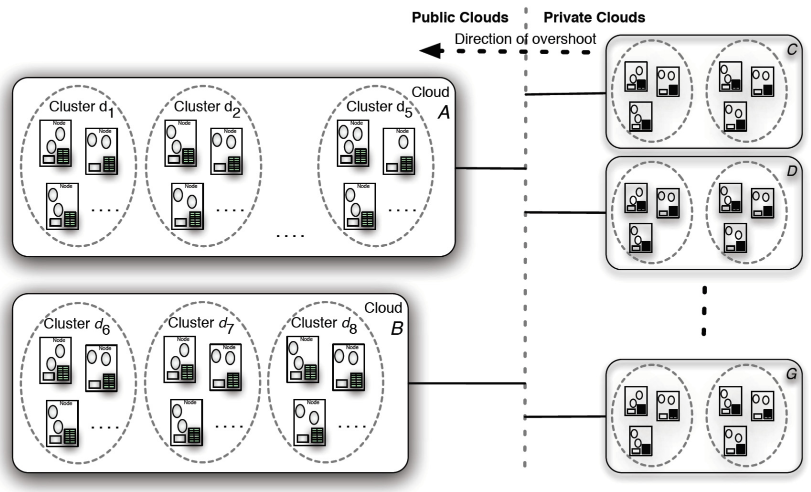


Figure 10: Cloud computing example scenario

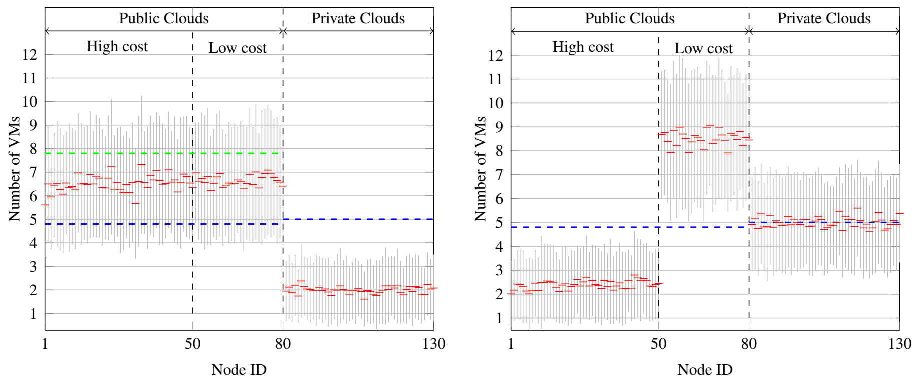
The target network of the private and public clouds offers 130 nodes in different clusters. 5 and 10 nodes are available in each private and public cluster respectively. Any single authority owning a private cloud administers two clusters, which together with the 5 (cloud *A*) plus 3 (cloud *B*) clusters result in a total of 18 clusters. Regarding the services one species is used for each, giving 25 nests in each private cloud emitting ants. We investigate 2 combinations of (10) and (11) with the above example setting:

1 no cloud costs, $z = 0$;

2 exponential weighting for cloud costs, $z > 0$.

We conducted simulations with the above variants of cost evaluation and checked the resulting deployment mappings. In Fig. 11a, mapping of VM instances – after convergence – is shown when $z = 0$, i.e. every node n_i has zero financial cost for hosting a VM, error bars show the deviation of the results. In case there are no financial costs of using a node we can observe that VMs are mapped evenly – while maintaining the dependability requirements – over all the nodes resulting in an average of 2 VMs per node in the private clouds, which leaves $625 - (5 \cdot 20) = 525$ VMs for the public clouds *A* and *B*. Further, the VMs in the public clouds are mapped quite evenly too over the available 80 nodes around the average of $525/80 = 6.6$. Moreover, the logic does not distinguish between the two different public cloud offerings in this case. The two extremes of mapping 3 VMs in public $((3 \cdot 125)/80 = 4.7)$ and 2 in private clusters; and mapping all 5 VMs in the services in the public clusters

$((5 \cdot 125)/80 = 7.8)$ are shown by the dashed lines.



(a) VMs per node, $z = 0$, no weighting (b) VMs per node, $z > 0$, exponential weighting

Figure 11: Example scenario with 10 services

When $z > 0$, we present the average amount of VM instances per node in case of the exponential cost function in Fig. 11b. In this case, the deployment logic finds mappings that successfully take into account the financial costs of hosting in public clouds. The public cloud with plenty of resources but, thus, higher costs (Cloud A, nodes $n_{1...50}$) receives a low amount of instances, whereas the lower cost public offering (Cloud B) is heavily loaded with VMs. Note that, the requirements of node and cluster-disjointness are still satisfied. In the private clusters 5 VMs are mapped to each node on average, i.e. each one of the 25 services places 1 VM in each of the two locally available clusters, which incur 0 additional cost. However, due to the cluster-disjointness criteria the 3rd, 4th and 5th VM instance has to be placed to a public cloud. To handle this overshoot the algorithm looks for the lowest possible increment in cost that gives the resulting deployment. Differences in using the two variants of (11) lay in that with a more complex cost evaluation (exponential instead of a simple linear) more balanced deployment mappings are obtained, i.e. given the cost values assigned to cloud A and B the cheaper public cloud gets less overloaded with VMs, while the number of mappings in the larger public cloud increases to take over some of the execution load [CMH10].

The number of iterations the algorithm requires to produce the results discussed above depends on the problem size. We used 2000 *explorer* ants followed by an additional 3000 (10% *explorer* and 90% *normal*) ants for each species. An increased amount of nodes in itself would not make the deployment problem be more difficult to solve. In fact, an increased network size actually allows the algorithm to find better mappings, with lower costs, easier due to the larger amount of available resources. Scalability is impacted to a larger extent by the amount of services and the amount of VMs within the services executed simultaneously, as the number of species executed in parallel is proportional to the number of services and the complexity of the tasks an ant has to perform increases as the number of instances grows [CMH09].

With the example presented in this section we have shown that the deployment logic can be applied in a cloud computing scenario by carefully looking at the cost functions driving the optimization and adjusting them to the perceived utility of the various configurations. This way concepts of financial costs in connection with resource usage can also be used in the evaluation of deployment mappings, in addition to the traditional non-functional requirements of performance and dependability.

4.5 Cross-validation of Deployment Mappings with ILP

Distributed execution of our deployment mapping algorithm has been an important design criteria to avoid the deficiencies of existing centralized algorithms, e.g. performance bottlenecks and single points of failure. In addition, we intend to conserve resources by eliminating the need for centralized decision-making and the required updates and synchronization mechanisms. In Sec. 4.1, we presented an example – well-known in task assignment problems – converted to our context of collaborating components (see Fig. 4 for the service and the optimum mapping). In this section, we extend on this initial example with two additional service models, present an Integer Linear Program (ILP) able to solve component deployment problems with load-balancing and remote communication minimization criteria. We then compare simulation results obtained by executing the deployment algorithm on the example models in this section with the optimum cost solutions given by the ILP. Complexity of the deployment examples remains NP-hard even if we only deal with a single service at a time.

Beside the first model, the second example has an extended solution space, obtained by extending the first example into a larger service model (15 components, 5 bound, additional collaborations) and increasing the number of nodes (4 nodes) available for deployment mapping. The third example leaves the cardinality of $\mathbf{C}$ unchanged, while changing the model – the configuration of components – and increasing the amount of collaborations, and the number of available nodes is increased to 6 as well. The concrete amount of components, $|\mathbf{C}|$, and collaborations, $|\mathbf{K}|$, in the service models and the amount of nodes, $|\mathbf{N}|$, are shown in Fig. 12. The ILP we have developed to validate our simulation results will be presented next. We take into account the two counteracting objectives of load-balancing and remote communication minimization and solve the ILP using regular solver software, for further details we refer to [CH10].

We start with defining the solution variable $m_{i,j}$ representing the set of mappings M.

$$m_{i,j} = \begin{cases} 1, & \text{if component } c_i \text{ is mapped to node } n_j, \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

that will indicate and efficient mapping of components to nodes; and we continue with two parameters. First, $b_{i,j}$

$$b_{i,j} = \begin{cases} 1, & \text{if component } c_i \text{ is bound to node } n_j, \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

which enables the model to fix some of the mappings, if any components are bound in the model. Second, T, defined in (1) to approximate the ideal load-balance among the available nodes in the network.

Beside the binary $m_{i,j}$ we utilize two additional variables. The first for checking collocations, in col_i.

$$col_i = \begin{cases} 0, & \text{if } c_l, c_k \in k_i \text{ and } c_l \text{ is collocated with } c_k, \\ 1, & \text{otherwise.} \end{cases} \tag{24}$$

Another variable, Δ_j, to indirectly calculate the deviation from the ideal load-balance among the nodes hosting the components.

$$\Delta_j \geq 0, \forall n_j \in \mathbf{N} \tag{25}$$

The objective function used in the ILP is naturally very similar to the linear cost function (3).

$$min \sum_{j=1}^{|\mathbf{N}|} \Delta_j + \sum_{i=1}^{|\mathbf{K}|} f_i \cdot col_i \tag{26}$$

After having established the objective function, the constraints the solutions are subjected to have to be defined. First we stipulate that there has to be one and only one mapping for all of the components.

$$\sum_{j=1}^{|\mathbf{N}|} m_{i,j} = 1, \forall c_i \in \mathbf{C} \tag{27}$$

The ILP has to take into account that some component mappings might be restricted (*binding*) to particular nodes. Thus, we restrict the variable $m_{i,j}$ using $b_{i,j}$.

$$m_{i,j} \geq b_{i,j}, \forall c_i \in \mathbf{C}, \forall n_j \in \mathbf{N} \tag{28}$$

We introduce two additional constraints to implicitly define the values of the variable Δ_j that we apply in the objective function. We use two constraints instead of a single one to avoid having to use absolute vales (i.e. the $abs()$ function) and thus we avoid non-linear constraints.

$$\sum_{i=1}^{|\mathbf{C}|} e_i \cdot m_{i,j} - T \leq \Delta_j, \forall n_j \in \mathbf{N} \tag{29}$$

$$T - \sum_{i=1}^{|\mathbf{C}|} e_i \cdot m_{i,j} \leq \Delta_j, \forall n_j \in \mathbf{N} \tag{30}$$

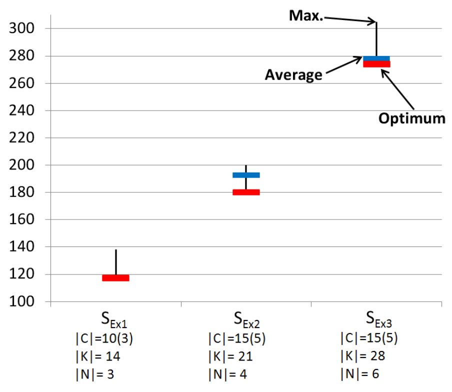
Similarly, we define two additional constraints for implicitly building the binary variable, col_i, indicating collocation of components.

$$m_{i,j} + m_{k,j} \leq (2 - col_l), k_l = (c_i, c_k) \in \mathbf{K}, \forall c_i, c_k \in \mathbf{C}, \forall n_j \in \mathbf{N} \tag{31}$$

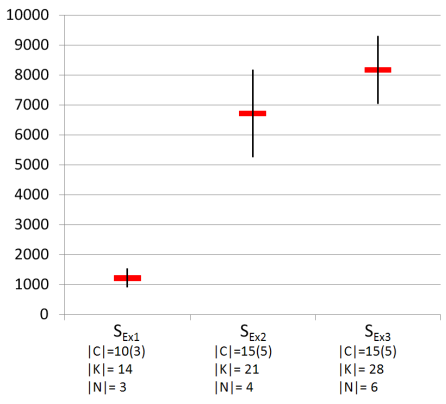
$$m_{i,j1} + m_{k,j2} \leq 1 + col_l, k_l = (c_i, c_k) \in \mathbf{K}, \forall c_i, c_k \in \mathbf{C}, \forall n_{j1}, n_{j2} \in \mathbf{N} \tag{32}$$

Using the above definitions the ILP can be executed using an appropriate solver. By submitting the appropriate data, defining the example services introduced above, we obtain the optimum mappings and their corresponding costs according to the objective/cost function (26). subject to the constraints in (27) – (32).

As a result of the cross-validation we obtain the absolute minimum cost values ($Optimum$) in the three example settings $S_{Ex1\dots3}$. Simulation results are generated by executing the algorithm 100 times for each example model. Average costs (*Average*) and the maximum deviation (*Max.*) are shown in Fig. 12a as well as the optimum obtained by ILP. Results show that our algorithm finds the optimum 99% of the time in case of the first example. In the somewhat larger scenario, S_{Ex2}, it is more difficult to find the absolute lowest cost mapping, thus we observe larger deviations in mapping costs. However, it is to be noted that by changing the mapping of a single component from the optimum configuration to a near-optimal one increases the costs by values larger than 1, which is also the reason for increased deviations in this case. In fact, the two most frequent sub-optimal solutions, beside the optimum cost of 180, were configurations with a cost of 195 and 200, giving an average of 193 in the end, shown in Fig. 12a. In case of the third example, S_{Ex3}, the algorithm managed to obtain solutions with costs closer to the absolute optimum – obtained by the ILP – with less deviation at the same time. The main reason for this is that communication costs in S_{Ex3} are relatively more fine grained, which resulted in finding near-optimum solutions with slightly higher costs only. The average number (and deviation) of *normal ant* iterations required by the algorithm to obtain the solutions are shown in Fig. 12b.



(a) Mapping costs



(b) Number of iterations

Figure 12: Cross-validation of the simulation results

It is to be noted that the difference in complexity is significant between the original example from Sec. 4.1 and the two extended models. Using the simplest binary pheromone encoding (cf. Sec. 3.3) the first example requires a pheromone database of size $3 \cdot 2^7$ in the network of 3 nodes, as the number of unbound components is 7. In the larger examples, the pheromone database size increases to $4 \cdot 2^{10}$ and $6 \cdot 2^{10}$ for

CEAS. It is difficult to precisely compare the computational effort required by the ILP and CEAS for solving the same problems. One iteration of a centralized logic with global knowledge, e.g. an ILP, can not really be compared to one iteration in the distributed CEAS, which is a tour made by the ant.

The solver software for the ILP provides some information regarding the iterations and cuts that were required during execution, i.e. 86, 495 and 1075 *simplex iterations*; and 0, 5 and 33 *branch and cut nodes* were reportedly required for the examples S_{Ex1}, S_{Ex2} and S_{Ex3} respectively. The number of required (*explorer* and *normal*) iterations in CEAS is naturally higher than what is required for the ILP with a global overview. However, we advocate that we gain more by the possibility of a completely distributed execution of our algorithm and also because of the capability of adaptation to changes in the context, once the pheromone database is built up after the initial phase.

5. Related work

Influencing the software architecture by changing the deployment configuration has been found to be an efficient way to improve utility of services. Deployment decision making requires an optimization method to function properly and autonomicity has to be built in as a basic functionality. An algorithm has been devised in [KHD08] that is based on calculating the usefulness of alternative configurations as weighted sums. Nevertheless, the resulting approach is not computationally effective and serves as a trial to show that deployment decision making is important and necessary to apply. Various other approaches have been followed to tackle the problem of efficiently mapping instances to resources, or hosts for adequate execution. Many authors start with centralized observation and control, utilizing for example binary integer programming [BSDS98], graph cutting [HS99], or some hybrid approach, e.g. proposed by the authors of [JHJ09], where an optimizer and a model solver work together to find optimal mappings specifically in the field of virtual machine technology. Computational complexity, which in most unrestricted cases is NP-hard, often prohibits application of centralized exhaustive methods above certain problem sizes, even as small as networks of only a handful of nodes. Other approaches try to restrict the solution space to tractable sizes by capturing important constraints [KIK03], but exhaustive search is still ineffective in practical problems, especially if we consider more than one QoS property of a configuration or more than one service at a time. Heuristics and approximative algorithms manage practical problem sizes more effectively. Malek et al. devised heuristic algorithms for software component deployment, based on greedy search and genetic programming in [Mal06], approaching the problem from the user's perspective instead that of the service providers'. Besides, stochastic optimization appeared as an alternative first in [WN04], suggesting the use of the Cross-Entropy method as well.

Regeneration of replicas to remedy crashed components was proposed first by Pu [PNP88] in the context of the Eden system. More recent systems, e.g. DARM [MMHB08], provide automatic reconfiguration and regeneration of replicas in the context of group communication systems, and Om [YV05] focus on regeneration in a peer-to-peer wide-area storage system. Recently, with the advent of cloud computing, standardized cloud interfaces propose similar mechanisms for placement, migration and monitoring of components in the cloud [EL09]. Recent studies [CFH$^+$05, HON$^+$09, JHJ09] show that the duration of virtual machine migration is in the order of 60-90 seconds. Service deployment support, that is intended to execute placement instructions given by our deployment logic are provided by such systems, however, focus is usually simply on failure recovery and improvement of availability without trying to optimize the new configurations and mappings. Authors in [YG09] show theoretically that replica placements of inter-correlated objects can impact system availability significantly if not chosen appropriately. Our work, on the other hand, focuses on improving both availability and overall performance.

Another centralized approach, namely group-finding algorithms to discover mappings in generic wide-area resource discovery is presented in [AOVP08]. In some way similarly to the foraging behavior of our artificial ants some approaches rely on extensive measurement data collection, however, our deployment logic does not store data centrally. Optimal placement of VMs under a variety of constraints has been focus of some research, e.g. in [VAN08] and [JHJ09]. The SmartFrog [Sab06] deployment and management framework from HP Labs describes services as collections of components and applies a distributed engine comprised of daemons running on every node in a network. Collections of components together with their configuration parameters can be activated and managed to deliver the desired services even in large-scale systems. The scale of these systems and the execution framework is close to the environment we envisage for the successful execution of autonomic component-based software services and which we target with our deployment logic. Configuration management in similar server environments based on fuzzy learning, targeting efficient resource utilization is investigated by Xu et al. in [XZF$^+$07]. Biologically-inspired resource allocation algorithms appear in [HJ06] to tackle service distribution problems. This is the path we too have chosen to follow while developing our deployment logic.

The basic method we built our deployment logic upon, the CEAS has been applied successfully to a variety of studies of different path management strategies, such as shared backup path protection, p-cycles, adaptive paths with stochastic routing, and resource search under QoS constraints [HHW08]. Implementation issues and trade-offs, such as the management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism and self-tuned packet rate control. Additional reduction in the overhead is accomplished by pheromone sharing, where ants with overlapping requirements cooperate in finding solutions by (partially) sharing information (see [HW10] for details). In CEAS applied to routing, a routing path, from source to destination, is the target of the search. Instead of the cost of mappings a routing path is evaluated in each iteration, i.e. a corresponding cost function is applied to the paths found. Furthermore, the pheromone values for routing CEAS is given by, $\tau_{ij,r}$ and represent an assignment between interface i and a node j at iteration r. Selection of the next hop for each ant, in this case, is based on the random proportional rule in the contrary to our algorithms.

6. Conclusions

In this paper, we summarize our recent research towards obtaining an intelligent solution, a decentralized logic that is capable of finding near-optimal deployments for building blocks of services in a dynamic network environment. We presented how our bio-inspired heuristic approach looks for an efficient mapping between software components and nodes iteratively. Example scenarios were discussed ranging from the deployment of collaborating software components and management of replicas to virtualization in hybrid cloud environments. Additionally, an ILP model was shown that can be used to cross-validate the solutions found by our algorithm in an offline, centralized manner.

Many interesting paths of future work are considered. The algorithms presented are to be re-implemented in a scalable, Java-based simulator in order to explore larger scale scenarios. Increasing problem sizes are anticipated with the introduction of larger networks and, especially, larger amounts of parallel services. Nevertheless, efficient pheromone encodings provide the necessary reduction in the size of database structures and allow controlled increase in complexity as problem sizes grow. Approximative methods, such as the heuristic algorithms presented in this paper, will continue to dominate on-line deployment decision making due to their flexibility and faster convergence. Besides, the core functions that describe the inner workings of the CEAS method, such as the pheromone or the temperature updates can also be revisited. The new, improved core functions in CEAS can possibly enhance the performance of the deployment logic.

The ILP model presented shall be extended to capture all the example scenarios. Regarding the scope of requirements, the next dimension to include is power-saving. Also, service models can be extended to capture additional aspects, such as consistency protocol costs. Besides, quantifying migration related costs is an interesting and difficult issue to look at in itself.

References

[AOVP08] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson. Design and implementation trade-offs for wide-area resource discovery. *ACM Trans. on Internet Technology 8(4)*, 2008.

[BSDS98] M. C. Bastarrica, A. A. Shvartsman, S. A. Demurjian, and Sr. A binary integer programming model for optimal object distribution. In *Proc. of the 2nd Int'l. Conf. on Principles of Distributed Systems*, 1998.

[CFH$^+$05] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of the 2nd ACM/USENIX Symp. on Networked Systems Design and Implementation (NSDI*, pages 273–286, 2005.

[CH10] M. J. Csorba and P. E. Heegaard. Swarm intelligence heuristics for component deployment. In *Proc. of the 16th Eunice Int'l Workshop and IFIP WG6.6 Workshop*, 2010.

[CHH08a] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In *Proc. of the 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), Turin.* ICST/ACM, September 2008.

[CHH08b] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), Oslo*, LNCS 5053, pages 253–268. IFIP, June 2008.

[CMH09] M. J. Csorba, H. Meling, and P. E. Heegaard. Laying pheromone trails for balanced and dependable component mappings. In *Proc. of the 4th Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5918*, 2009.

[CMH10] M. J. Csorba, H. Meling, and P. E. Heegaard. Ant system for service deployment in private and public clouds. In *Proc. of the 2nd workshop on Bio-inspired algorithms for distributed systems (BADS)*, pages 19–28, New York, NY, USA, 2010. ACM.

[CMHH09] M. J. Csorba, H. Meling, P. E. Heegaard, and P. Herrmann. Foraging for better deployment of replicated service components. In *Proc. of the 9th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), LNCS 5523*, pages 87–101, 2009.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[Efe82] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15(6):50–56, 1982.

[EL09] E. Elmroth and L. Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *Proc. of the 8th Int'l Conf. on Grid and Cooperative Computing (GCC)*, pages 253–260, 2009.

[FB89] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. on Software Engineering*, 15(11):1427–1436, 1989.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HJ06] T. Heimfarth and P. Janacik. Ant based heuristic for os service distribution on adhoc networks. In *Biologically Inspired Cooperative Computing (BICC)*, pages 75–84, 2006.

[HON$^+$09] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi. A live storage migration mechanism over wan for relocatable virtual machine services on clouds. In *Proc. of the 9th IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGRID)*, pages 460–465, 2009.

[HS99] G. C. Hunt and M. L. Scott. The coign automatic distributed partitioning system. In *Proc. of the 3rd Symp. on Operating systems design and implementation*, pages 187–200, Berkeley, CA, USA, 1999. USENIX Association.

[HW01] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[HW10] P. E. Heegaard and O. Wittner. Overhead reduction in a distributed path management system. *Computer Networks*, 54(6):1019–1041, 2010.

[JHJ09] K. Joshi, M. Hiltunen, and G. Jung. Performance aware regeneration in virtualized multitier applications. In *Proc. of the Workshop on proactive failure avoidance recovery and maintenance (PFARM)*, 2009.

[KD07] J. O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[KH06] F. A. Kraemer and P. Herrmann. Service specification by composition of collaborations - an example. In *Proc. of the 2006 Int'l Conf. on Web Intelligence and Intelligent Agent Technology, Hong Kong*. IEEE/WIC/ACM, 2006.

[KHD08] R. Kusber, S. Haseloff, and K. David. An approach to autonomic deployment decision making. In *Proc. of the 3rd Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5343*, pages 121–132, 2008.

[KIK03] T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using ai planning techniques. In *Proc. of the Int. Parallel and Distributed Processing Symposium*, 2003.

[KLMS84] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proc. of the 16th annual ACM symposium on Theory of computing*, 1984.

[LLC] Amazon Web Services LLC. Amazon elastic compute cloud. `http://aws.amazon.com/ec2`.

[Mal06] S. Malek. A user-centric framework for improving a distributed software system's deployment architecture. In *Proc. of the doctoral track at the 14th ACM SIGSOFT Symp. on Foundation of Software Engineering, Portland*, 2006.

[MG08] H. Meling and J. L. Gilje. A distributed approach to autonomous fault treatment in spread. In *Proc. of the 7th European Dependable Computing Conference (EDCC)*, pages 46–55, 2008.

[MMHB08] H. Meling, A. Montresor, B. E. Helvik, and Ö. Babaoglu. Jgroup/arm: a distributed object group platform with autonomous replication management. *Softw., Pract. Exper.*, 38(9):885–923, 2008.

[PNP88] C. Pu, J. D. Noe, and A. Proudfoot. Regeneration of replicated objects: A technique and its eden implementation. *IEEE Trans. on Software Engineering*, 14:936–945, 1988.

[RBL$^+$08] R. Rouvoy, M. Beauvois, L. Lozano, J. Lorenzo, and F. Eliassen. Music: an autonomous platform supporting self-adaptive mobile applications. In *Proc. of the 1st workshop on Mobile middleware: embracing the personal communication device*. ACM, Dec 2008.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[Sab06] R. Sabharwal. Grid infrastructure deployment using smartfrog technology. In *Proc. of the Int'l Conf. on Networking and Services (ICNS), Santa Clara, USA*, pages 73–79, July 2006.

[SD02] T. Stützle and M. Dorigo. A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans. on Evolutionary Computation*, 6(4):358–365, 2002.

[VAN08] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[Wit03] O. Wittner. *Emergent Behavior Based Implements for Distributed Network Management*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2003.

[WN04] N. Widell and C. Nyberg. Cross entropy based module allocation for distributed systems. In *Proc. of the 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, pages 187–200, Berkeley, CA, USA, 2004. USENIX Association.

[WSVY07] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. of the 4th USENIX Symp. on Networked Systems Design and Implementation*, 2007.

[XZF$^+$07] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2007.

[YG09] H. Yu and P. B. Gibbons. Optimal inter-object correlation when replicating for availability. volume 21, pages 367–384, 2009.

[YV05] H. Yu and A. Vahdat. Consistent and automatic replica regeneration. *ACM Trans. on Storage*, 1(1):3–37, 2005.

[Z+04] M. Zlochin et al. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131:373–395, 2004.

Abbreviations

ACO	Ant Colony Optimization
AI	Artificial Intelligence
API	Application Programming Interface
CE	Cross Entropy
CEAS	Cross Entropy Ant System
CPP	Component Placement Problem
CPU	Central Processing Unit
DARM	Distributed Autonomous Replication Management
DVS	Dynamic Voltage Scaling
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
ILP	Integer Linear Programming
IP	Internet Protocol
ISIS	Infrastructure for Integrated Services
JVM	Java Virtual Machine
MAPE	Monitor, Analyze, Plan and Execute
NAT	Network Address Translation
NF	Non-functional
NFR	Non-Functional Requirement
NP	non-deterministic polynomial-time
OS	Operating System

QNs	Queuing Networks
QoS	Quality of Service
rpr	random proportional rule
SBPP	Shared Backup Path Protection
SLA	Service Level Agreement
SM	State Machine
SPACE	SPecification by Activities, Collaborations and Executable state machines
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URN	User Requirements Notation
VM	Virtual Machine
WAN	Wide Area Network

Bibliography

[Aag01] J. Ø. Aagedal. *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo, Norway, 2001.

[ADJ$^+$10] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: automated data placement for geo-distributed cloud services. In *Proc. of the 7th USENIX Conf. on Networked systems design and implementation (NSDI)*, pages 2–, 2010.

[AM02] D. Amyot and G. Mussbacher. Urn: Towards a new standard for the visual description of requirements. 2002.

[AOVP08] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson. Design and implementation trade-offs for wide-area resource discovery. *ACM Trans. on Internet Technology 8(4)*, 2008.

[ATK05] A. Akkerman, A. Totok, and V. Karamcheti. Infrastructure for automatic dynamic deployment of j2ee applications in distributed environments. In *Proc. of the 3rd Int. Working Conference on Component Deployment (CD), LNCS 3798*, pages 17–32, 2005.

[ATZ07] D. Ardagna, M. Trubian, and L. Zhang. Sla based resource allocation policies in autonomic environments. *J. of Parallel Distrib. Comput.*, 67(3):259–270, 2007.

[BCT07] M. Bernaschi, F. Casadei, and P. Tassotti. Sockmi: a solution for migrating tcp/ip connections. In *Proc. of the 15th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing*, 2007.

[BFdM09] A. Berl, A. Fischer, and H. de Meer. Using system virtualization to create virtualized networks. *ECEASST*, 17, 2009.

[BGG$^+$10] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis. Energy-Efficient Cloud Computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[Bir03] G. Birtwistle. *Demos - a system for Discrete Event Modelling on Simula.* Springer-Verlag New York, Inc., 2003.

[BMM02] O. Babaoglu, H. Meling, and A. Montresor. Anthill: a framework for the development of agent-based peer-to-peer systems. In *Proc. of the 22nd Int. Conf. on Distributed Computing Systems*, pages 15–22, 2002.

[BMST93] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. pages 199–216, 1993.

[BR09] U. Brinkschulte and A. Renteln. Analyzing the behavior of an artificial hormone system for task allocation. In *Proc. of the 6th Int. Conf. on Autonomic and Trusted Computing (ATC)*, pages 47–61, 2009.

[BSDS98] M. C. Bastarrica, A. A. Shvartsman, S. A. Demurjian, and Sr. A binary integer programming model for optimal object distribution. In *Proc. of the 2nd Int'l. Conf. on Principles of Distributed Systems*, 1998.

[BWS$^+$09] A. Berl, R. Weidlich, M. Schrank, H. Hlavacs, and H. de Meer. Network virtualization in future home environments. In *Proc. of the 20th Integrated Management of Systems, Services, Processes and People in IT, IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, (DSOM)*, pages 177–190, 2009.

[CAAS07] I. Cunha, J. Almeida, V. Almeida, and M. Santos. Self-adaptive capacity management for multi-tier virtualized environments. In *Proc. of the 10th IFIP/IEEE Int. Symp. on Integrated Network Management (IM)*, pages 129 –138, 2007.

[CB09] N. M. Mosharaf Kabir Chowdhury and R. Boutaba. Network virtualization: state of the art and research challenges. *IEEE Comm. Mag.*, 47:20–26, 2009.

[CD98] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, 1998.

[CDG05] G. Di Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Trans. on Telecomm. (ETT) - Special Issue on Self Organization in Mobile Networking*, 16(5), 2005.

[CEM03] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. on Software Engineering*, 29(10):929–945, 2003.

[CFH$^+$05] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of the 2nd ACM/USENIX Symp. on Networked Systems Design and Implementation (NSDI*, pages 273–286, 2005.

[CH10] M. J. Csorba and P. E. Heegaard. Swarm intelligence heuristics for component deployment. In *Proc. of the 16th Eunice Int'l Workshop and IFIP WG6.6 Workshop*, 2010.

[CHH08a] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In *Proc. of the 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), Turin*. ICST/ACM, September 2008.

[CHH08b] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), Oslo*, LNCS 5053, pages 253–268. IFIP, June 2008.

[CHH10] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Component deployment using parallel ant-nests. *To appear in Int'l Journal on Autonomous and Adaptive Communications Systems (IJAACS)*, 2010.

[CJ09] J. Carapinha and J. Jiménez. Network virtualization: a view from the bottom. In *Proc. of the 1st ACM workshop on Virtualized Infrastructure Systems and Architectures (VISA)*, 2009.

[CLG$^+$09] B. H. Cheng, R. Lemos, H. Giese, et al. Software engineering for self-adaptive systems: A research roadmap. In B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer-Verlag, 2009.

[CMH09] M. J. Csorba, H. Meling, and P. E. Heegaard. Laying pheromone trails for balanced and dependable component mappings. In *Proc. of the 4th Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5918*, 2009.

[CMH10] M. J. Csorba, H. Meling, and P. E. Heegaard. Ant system for service deployment in private and public clouds. In *Proc. of the 2nd workshop on Bio-inspired algorithms for distributed systems (BADS)*, pages 19–28, New York, NY, USA, 2010. ACM.

[CMHH09] M. J. Csorba, H. Meling, P. E. Heegaard, and P. Herrmann. Foraging for better deployment of replicated service components. In *Proc. of the 9th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), LNCS 5523*, pages 87–101, 2009.

[CS02] X. Chen and M. Simons. A component framework for dynamic reconfiguration of distributed systems. In *Proc. of the 1st Int. Working Conference on Component Deployment (CD), LNCS 2370*, pages 82–96, 2002.

[CW98] S. Choi and C. Wu. Partitioning and allocation of objects in heterogeneous distributed environments using the niched pareto genetic-algorithm. In *Proc. of the 5th Asia Pacific Software Engineering Conference (APSEC)*, Washington, DC, USA, 1998. IEEE Computer Society.

[dBKMR05] P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134, 2005.

[DDF$^+$06] S. Dobson, S. G. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *TAAS*, 1(2):223–259, 2006.

[Dea] J. Dean. Software engineering advice from building large-scale distributed systems. `http://research.google.com/people/jeff/stanford-295-talk.pdf`.

[Déf00] X. Défago. *Agreement-Related Problems: From Semi-Passive Replication to Totally Ordered Broadcast. Number 2229.* PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2000.

[DGH$^+$04] K. Doerner, W. Gutjahr, R. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research*, 131:79–99, 2004.

[DHR01] K. Doerner, R. F. Hartl, and M. Reimann. Are competants more competent for problem solving? - the case of a multiple objective transportation problem. In *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO)*, page 802, 2001.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[Efe82] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15(6):50–56, 1982.

[Egy04] A. Egyed. Dynamic deployment of executing and simulating software components. In *Proc. of the 2nd Int. Working Conference on Component Deployment (CD), LNCS 3083*, pages 113–128, 2004.

[EL09] E. Elmroth and L. Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *Proc. of the 8th Int'l Conf. on Grid and Cooperative Computing (GCC)*, pages 253–260, 2009.

[FB89] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. on Software Engineering*, 15(11):1427–1436, 1989.

[FGK02] R. Fourer, D. M. Gay, and B. W. Kernighan. Ampl: A modeling language for mathematical programming. *ISBN:0-534-38809-4*, 2002.

[FK98] S. Frølund and J. Koistinen. QoS specification in distributed object systems. Technical Report HPL-98-159, HP Tech Report, 1998.

[FSJB01] J. Floch, R. T. Sanders, U. Johansen, and R. Bræk. Using uml for implementation design of sdl systems. In *SDL Forum*, pages 90–106, 2001.

[Fün98] S. Fünfrocken. Transparent migration of java-based mobile agents. In *Proc. of the 2nd Int. Workshop on Mobile Agents (MA), LNCS 1477*, pages 26–37, 1998.

[FW04] G. Franks and M. Woodside. Multiclass multiservers with deferred operations in layered queueing networks, with software system applications. In *Proc. of the 12th IEEE/ACM Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2004.

[GC02] D. Grosu and A. T. Chronopoulos. A game-theoretic model and algorithm for load balancing in distributed systems. In *Proc. of the 16th Int. Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2002.

[GHBDL09] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *Proc. of the 11th Int. Joint Conf. on Measurement and Modeling of Computer Systems, (SIGMETRICS/Performance)*, pages 157–168, 2009.

[GHK09] L. A. Gunawan, P. Herrmann, and F. A. Kraemer. Towards the integration of security aspects into system development using collaboration-oriented models. In Dominik Slezak, Tai-hoon Kim, Wai-Chi Fang, and Kirk P. Arnett, editors, *Security Technology*, volume 58 of *Communications in Computer and Information Science*, pages 72–85. 2009.

[GM02a] M. Guntsch and M. Middendorf. Applying population based aco to dynamic optimization problems. In *Ant Algorithms (ANTS), LNCS 2463*, pages 111–122, 2002.

[GM02b] M. Guntsch and M. Middendorf. A population based approach for aco. In *Proc. of Applications of Evolutionary Computing (EvoWorkshops)*, pages 72–81, 2002.

[GM03] M. Guntsch and M. Middendorf. Solving multi-criteria optimization problems with population-based aco. In *Proc. of the 2nd Int. Conf. on Evolutionary Multi-Criterion Optimization (EMO)*, pages 464–478, 2003.

[GRC$^+$08] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *Proc. of the IEEE Int. Conf. on Dependable Systems and Networks (DSN)*, pages 326 –335, 2008.

[Gul10] A. N. Gullhav. Service deployment problems with replicated and non-replicated components: a mixed integer linear programming approach. Technical report, Department of Industrial Economics and Technology Management, NTNU, Dec 2010.

[H$^+$04] P. E. Heegaard et al. Distributed asynchronous algorithm for cross-entropy-based combinatorial optimization. In *Rare Event Simulation and Combinatorial Optimization, Budapest*, 2004.

[H$^+$05] P. E. Heegaard et al. Self-managed virtual path management in dynamic networks. In *Self-* Properties in Complex Information Systems, LNCS 3460*, 2005.

[Hal04] R. S. Hall. A policy-driven class loader to support deployment in extensible frameworks. In *Proc. of the 2nd Int. Working Conference on Component Deployment (CD), LNCS 3083*, pages 81–96, 2004.

[HASL07] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Tran. on Computers*, 56(4):444–458, 2007.

[HE07] C. K. Ho and H. T. Ewe. Ant colony optimization approaches for the dynamic load-balanced clustering problem in ad hoc networks. In *Proc. of the IEEE Swarm Intelligence Symposium (SIS)*, pages 76–83, 2007.

[HHW08] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[HJ06] T. Heimfarth and P. Janacik. Ant based heuristic for os service distribution on adhoc networks. In *Biologically Inspired Cooperative Computing (BICC)*, pages 75–84, 2006.

[HJS$^+$09] M. Hiltunen, K. Joshi, R. Schlichting, G. Jung, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. Technical report, April 2009.

[HK07] P. Herrmann and F. A. Kraemer. Design of trusted systems with reusable collaboration models. In *Proc. of the Joint IFIP iTrust and PST Conferences on Privacy, Trust Management and Security, Moncton*, 2007.

[HKU01] B. Hardekopf, K. Kwiat, and S. Upadhyaya. A decentralized voting algorithm for increasing dependability in distributed systems. In *Proc. of the Joint Meeting of the 5th World Multiconf. on Systemics, Cybernetics and Informatics (SCI 2001) and the 7th Int. Conf. on Information Systems Analysis and Synthesis (ISAS 2001)*, 2001.

[HMM05] B. E. Helvik, H. Meling, and A. Montresor. An approach to experimentally obtain service dependability characteristics of the jgroup/arm system. In *Proc. of the 5th European Dependable Computing Conference (EDCC), LNCS 3463*, pages 179–198, 2005.

[HON$^+$09] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi. A live storage migration mechanism over wan for relocatable virtual machine services on clouds. In *Proc. of the 9th IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGRID)*, pages 460–465, 2009.

[HS99] G. C. Hunt and M. L. Scott. The coign automatic distributed partitioning system. In *Proc. of the 3rd Symp. on Operating systems design and implementation*, pages 187–200, Berkeley, CA, USA, 1999. USENIX Association.

[HW01] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[HW06a] P. E. Heegaard and O. Wittner. Restoration performance vs. overhead in a swarm intelligence path management system. In *Proc. of the Fifth Int'l. Workshop on Ant Colony Optimization and Swarm Intelligence, Brussels*, 2006.

[HW06b] P. E. Heegaard and O. Wittner. Self-tuned refresh rate in a swarm intelligence path management system. In *Proc. of the EuroNGI Int'l. Workshop on Self-Organizing Systems, LNCS 4124*, 2006.

[HW10] P. E. Heegaard and O. Wittner. Overhead reduction in a distributed path management system. *Computer Networks*, 54(6):1019–1041, 2010.

[JHJ09] K. Joshi, M. Hiltunen, and G. Jung. Performance aware regeneration in virtualized multitier applications. In *Proc. of the Workshop on proactive failure avoidance recovery and maintenance (PFARM)*, 2009.

[JHS$^+$04] K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, W. H. Sanders, and A. Agbaria. Online model-based adaptation for optimizing performance and dependability. In *Proc. of the 1st ACM SIGSOFT Workshop on Self-Managed Systems (WOSS)*, pages 85–89, 2004.

[JJH$^+$08] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2008.

[JJH$^+$09] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *Middleware*, pages 163–183, 2009.

[JMJV] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[KBH07] F. A. Kraemer, R. Bræk, and P. Herrmann. Synthesizing components with sessions from collaboration-oriented service specifications. In *Proc. of the 13th Int. SDL Forum conference on Design for dependable systems, LNCS 4745*, pages 166–185, Berlin, Heidelberg, 2007. Springer-Verlag.

[KBH09] F. A. Kraemer, R. Bræk, and P. Herrmann. Compositional service engineering with arctis. *Telektronikk*, (01), 2009.

[KC03] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, 2003.

[KD07] J. O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[KH06] F. A. Kraemer and P. Herrmann. Service specification by composition of collaborations - an example. In *Proc. of the 2006 Int'l Conf. on Web Intelligence and Intelligent Agent Technology, Hong Kong*. IEEE/WIC/ACM, 2006.

[KH07] F. A. Kraemer and P. Herrmann. Transforming collaborative service specifications into efficiently executable state machines. *Electronic Communications of the EASST*, 6, 2007.

[KH09] F. A. Kraemer and P. Herrmann. Automated Encapsulation of UML Activities for Incremental Development and Verification. In *Proc. of the 12th Int. Conf. on Model Driven Engineering, Languages and Systems (Models)*, pages 571–585, 2009.

[KH10] F. A. Kraemer and P. Herrmann. Reactive semantics for distributed uml activities. In *Proc. of Formal Techniques for Distributed Systems, Joint 12th IFIP WG 6.1 Int. Conf., FMOODS 2010 and 30th IFIP WG 6.1 Int. Conf., FORTE 2010*, pages 17–31, 2010.

[KHB06] F. A. Kraemer, P. Herrmann, and R. Bræk. Aligning uml 2.0 state machines and temporal logic for the efficient execution of services. In *Proc. of the 8th Int'l Symp. on Distributed Objects and Applications (DOA), LNCS 4276, Montpellier*, 2006.

[KHD08] R. Kusber, S. Haseloff, and K. David. An approach to autonomic deployment decision making. In *Proc. of the 3rd Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5343*, pages 121–132, 2008.

[KIK03] T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using ai planning techniques. In *Proc. of the Int. Parallel and Distributed Processing Symposium*, 2003.

[Kle76] L. Kleinrock. *Queueing Systems, Volume II: Computer Applications*. Wiley Interscience, 1976.

[KLMS84] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proc. of the 16th annual ACM symposium on Theory of computing*, 1984.

[Kra08] F. A. Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2008.

[KSH07] F. A. Kraemer, V. Slåtten, and P. Herrmann. Engineering support for uml activities by automated model-checking - an example. In *Proc. of the 4th Int'l Workshop on Rapid Integration of Software Engineering Techniques (RISE), University of Luxembourg*, 2007.

[KSH09] F. A. Kraemer, V. Slåtten, and P. Herrmann. Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[KSW07] D. Krivitski, A. Schuster, and R. Wolff. A local facility location algorithm for large-scale distributed systems. *Journal of Grid Computing*, 5:361–378, 2007.

[KWH08] V. Kjeldsen, O. Wittner, and P. E. Heegaard. Distributed and scalable path management by a system of cooperating ants. In *Proc. of the Int'l. Conf. on Communications in Computing (CIC)*, 2008.

[Lam04] A. Lamsweerde. Goal-oriented requirements enginering: A roundtrip from research to practice. In *Proc. of the 12th IEEE Int. Requirements Engineering Conference*, pages 4–7, Washington, DC, USA, 2004. IEEE Computer Society.

[LK87] F. C. H. Lin and R. M. Keller. The gradient model load balancing method. *IEEE Trans. Softw. Eng.*, 13:32–38, 1987.

[LLC] Amazon Web Services LLC. Amazon elastic compute cloud. `http://aws.amazon.com/ec2`.

[LLS+99] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource QoS problem. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, RTSS '99, pages 315–, Washington, DC, USA, 1999. IEEE Computer Society.

[LSO+07] S. A. Lundesgaard, A. Solberg, J. Oldevik, R. France, J. Ø. Aagedal, and F. Eliassen. Construction and execution of adaptable applications using an aspect-oriented and model driven approach. In *Proc. of DAIS, LNCS 4531*, pages 76–89. IFIP, 2007.

[LZ09] Y. C. Lee and A. Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Proc. of the 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID)*, pages 92–99, 2009.

[Mal06] S. Malek. A user-centric framework for improving a distributed software system's deployment architecture. In *Proc. of the doctoral track at the 14th ACM SIGSOFT Symp. on Foundation of Software Engineering, Portland*, 2006.

[MB08] A. Merkel and F. Bellosa. Memory-aware scheduling for energy efficiency on multi-core processors. In *Proc. of the 2008 Conf. on Power aware computing and systems (HotPower)*, pages 1–, 2008.

[MD07] D. Menasce and V. Dubey. Utility-based QoS brokering in service oriented architectures. In *Proc. of the Int'l Conf. on Web Services (ICWS), Salt Lake City, Utah*, July 2007.

[MEB+10] S. Malek, G. Edwards, Y. Brun, H. Tajalli, J. Garcia, I. Krka, N. Medvidovic, M. Mikic-Rakic, and G. S. Sukhatme. An architecture-driven software mobility framework. *J. Syst. Softw.*, 83:972–989, 2010.

[Mel06] H. Meling. *Adaptive Middleware Support and Autonomous Fault Treatment: Architectural Design, Prototyping and Experimental Evaluation*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2006.

[MEM+09] S. Malek, N. Esfahani, D. A. Menasce, J. P. Sousa, and H. Gomaa. Self-architecting software systems (SASSY) from QoS-annotated activity models. In *Proc. of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*, pages 62–69, Washington, DC, USA, 2009. IEEE Computer Society.

[MG05] R. Montemanni and L. M. Gambardella. Swarm approach for a connectivity problem in wireless networks. In *Proc. of the 2nd IEEE Swarm Intelligence Symposium (SIS)*, 2005.

[MG08] H. Meling and J. L. Gilje. A distributed approach to autonomous fault treatment in spread. In *Proc. of the 7th European Dependable Computing Conference (EDCC)*, pages 46–55, 2008.

[MH07] S. B. Musunoori and G. Horn. Co-ordination in intelligent ant-based application service mapping in grid environments. In *Proc. of the IEEE Swarm Intelligence Symposium (SIS)*, pages 303–309, 2007.

[MM99] C. E. Mariano and E. Morales. Moaq: An ant-q algorithm for multiple objective optimization problems. In *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO)*, pages 894–901, 1999.

[MM06] S. Malek and N. Medvidovic. A user-centric approach for improving a distributed software system's deployment architecture. Technical Report USCCSE-2006-602, University of Southern California, 2006.

[MMB03] A. Montresor, H. Meling, and O. Babaoğlu. Messor: load-balancing through a swarm of autonomous agents. In *Proc. of the 1st Int. Conf. on Agents and peer-to-peer computing*, pages 125–137, 2003.

[MMHB08] H. Meling, A. Montresor, B. E. Helvik, and Ö. Babaoglu. Jgroup/arm: a distributed object group platform with autonomous replication management. *Softw., Pract. Exper.*, 38(9):885–923, 2008.

[MRG04] D. A. Menascé, H. Ruan, and H. Gomaa. A framework for QoS-aware software components. In *WOSP*, pages 186–196, 2004.

[MRKE09] S. Malek, R. Roshandel, D. Kilgore, and I. Elhag. Improving the reliability of mobile software systems through continuous analysis and proactive reconfiguration. In *Proc. of the 31st Int. Conf. on Software Engineering, ICSE-Companion*, pages 275 –278, 2009.

[MRM04] M. Mikic-Rakic and N. Medvidovic. Support for disconnected operation via architectural self-reconfiguration. In *Proc. of the 1st Int. Conf. on Autonomic Computing (ICAC)*, 2004.

[MRMM05a] M. Mikic-Rakic, S. Malek, and N. Medvidovic. Improving availability in large, distributed component-based systems via redeployment. In *Proc. of the 3rd Int. Working Conference on Component Deployment (CD), LNCS 3798*, pages 83–98, 2005.

[MRMM05b] M. Mikic-Rakic, N. Medvidovic, and S. Malek. A decentralized redeployment algorithm for improving the availability of distributed systems. In *Proc. of the 3rd Int. Working Conference on Component Deployment (CD), LNCS 3798*, 2005.

[Nil71] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence, McGraw-Hill.* 1971.

[NOK93] M. Nakamura, K. Onaga, and S. Kyan. A mutual exclusion algorithm for a distributed network of autonomous nodes. In *Proc. of the IEEE Int. Symp. on Circuits and Systems*, 1993.

[NWG$^+$09a] D. Nurmi, R. Wolski, C. Grzegorczyk, et al. The eucalyptus open-source cloud-computing system. In *Proc. of the 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID)*, pages 124 –131, 2009.

[NWG$^+$09b] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: an open-source cloud computing infrastructure. *Journal of Physics: Conference Series*, 180(9), 2009.

[OMG00] OMG. Fault tolerant CORBA specification. Technical Report ptc/00-04-04, OMG Document, April 2000.

[Opt] Gurobi Optimization. The gurobi optimizer. `http://gurobi.com`.

[PH09] L. Paquereau and B. E. Helvik. Revisiting the auto-regressive functions of the cross-entropy ant system. In *Proc. of the 4th Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5918*. 2009.

[PH10] L. Paquereau and B. E. Helvik. Opportunistic ant-based path management for wireless mesh networks. In *Proc. of the 7th Int. Conf. on Swarm Intelligence (ANTS), LNCS 6234*, pages 480–487, 2010.

[PJE09] B. E. Helvik L. Paquereau P. J. Emstad, P. E. Heegaard. *Dependability and performance with discrete event simulation*. Tapir akademiske forlag, NTNU, 2009.

[PKF05] S. Paal, R. Kammüller, and B. Freisleben. Crosslets: Self-managing application deployment in a cross-platform operating environment. In *Proc. of the 3rd Int. Working Conference on Component Deployment (CD), LNCS 3798*, pages 52–66, 2005.

[PNP88] C. Pu, J. D. Noe, and A. Proudfoot. Regeneration of replicated objects: A technique and its eden implementation. *IEEE Trans. on Software Engineering*, 14:936–945, 1988.

[RBL$^+$08] R. Rouvoy, M. Beauvois, L. Lozano, J. Lorenzo, and F. Eliassen. Music: an autonomous platform supporting self-adaptive mobile applications. In *Proc. of the 1st workshop on Mobile middleware: embracing the personal communication device*. ACM, Dec 2008.

[RBL$^+$09] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.*, 53(4):535–545, 2009.

[RCBC07] P. Rigole, T. Clerckx, Y. Berbers, and K. Coninx. Task-driven automated component deployment for ambient intelligence environments. *Pervasive Mob. Comput.*, 3:276–299, 2007.

[REF$^+$08] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E. Stav. Composing components and services using a planning-based adaptation middleware. In *Proc. of SC, LNCS4954*, pages 52–67. Springer-Verlag, 2008.

[RG05] H. Roussain and F. Guidec. Cooperative component-based software deployment in wireless ad hoc networks. In *Proc. of the 3rd Int. Working Conference on Component Deployment (CD), LNCS 3798*, 2005.

[Rub99] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[RW03] M. Roth and S. Wicker. Termite: ad-hoc networking with stigmergy. In *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM)*, volume 5, pages 2937–2941, 2003.

[S$^+$97] R. Schoonderwoerd et al. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2), 1997.

[Sab06] R. Sabharwal. Grid infrastructure deployment using smartfrog technology. In *Proc. of the Int'l Conf. on Networking and Services (ICNS), Santa Clara, USA*, pages 73–79, July 2006.

[Sch90] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv. 22(4)*, 1990.

[Sch93] F. B. Schneider. Replicated management using the state-machine approach. *Distributed systems (2nd Ed.)*, pages 169–197, 1993.

[SCK08] V. Sesum-Cavic and E. Kühn. Instantiation of a generic model for load balancing with intelligent algorithms. In *Proc. of the 3rd Int'l Workshop on Self-Organizing Systems (IWSOS), LNCS 5343*, pages 311–317, 2008.

[SD02] T. Stützle and M. Dorigo. A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans. on Evolutionary Computation*, 6(4):358–365, 2002.

[SME09] H. Sato, S. Matsuoka, and T. Endo. File clustering based replication algorithm in a grid environment. In *Proc. of the 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID)*, pages 204–211, Washington, DC, USA, 2009. IEEE Computer Society.

[SMM07] C. Seo, S. Malek, and N. Medvidovic. An energy consumption framework for distributed java-based systems. In *Proc. of the 22nd IEEE/ACM Int. Conf. on Automated Software Engineering*, pages 421–424, New York, NY, USA, 2007. ACM.

[Som04] N. L. Sommer. Towards a dynamic resource contractualisation for software components. In *Proc. of the 2nd Int. Working Conference on Component Deployment (CD), LNCS 3083*, pages 129–143, 2004.

[SPYH04] O. Sato, R. Potter, M. Yamamoto, and M. Hagiya. Uml scrapbook and realization of snapshot programming environment. In Kokichi Futatsugi, Fumio Mizoguchi, and Naoki Yonezaki, editors, *Software Security - Theories and Systems*, volume 3233 of *Lecture Notes in Computer Science*, pages 281–295. Springer Berlin / Heidelberg, 2004.

[SSII01] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Highly available internet services using connection migration. Technical Report DCS-TR-462, Rutgers University, 2001.

[SZL10] R. Subrata, A. Y. Zomaya, and B. Landfeldt. Cooperative power-aware scheduling in grid computing environments. *J. Parallel Distrib. Comput.*, 70:84–91, 2010.

[TM05] C. Teyssie and Z. Mammeri. Uml-based approach for network QoS specification. *ICN (Vol. 1)*, 2005.

[TSG04] M. Tichy, D. Schilling, and H. Giese. Design of self-managing dependable systems with uml and fault tolerance patterns. In *Proc. of the 1st ACM SIGSOFT Workshop on Self-Managed Systems (WOSS)*, pages 105–109, 2004.

[TZNTG09] K. Tutschku, T. Zinner, A. Nakao, and P. Tran-Gia. Network virtualization: Implementation steps towards the future internet. *ECEASST*, 17, 2009.

[VAN08] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[WFZ04] H. Wedde, M. Farooq, and Y. Zhang. Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior. In *Proc. of the 4th Int. Workshop Ant Colony Optimization and Swarm Intelligence (ANTS)*, pages 83–94, 2004.

[WGB09] S. Wardley, E. Goyer, and N. Barcet. Ubuntu enterprise cloud architecture. Technical report, Aug 2009.

[WH02] O. Wittner and B. E. Helvik. Cross entropy guided ant-like agents finding dependable primary/backup path patterns in networks. In *Proc. of the Congress on Evolutionary Computation (CEC2002)*, pages 1528–1533, 2002.

[WH04] O. Wittner and B. E. Helvik. Distributed soft policy enforcement by swarm intelligence; application to load sharing and protection. *Annals of Telecommunications*, 59, 2004.

[WHH03] O. Wittner, P. E. Heegaard, and B. E. Helvik. Scalable distributed discovery of resource paths in telecommunication networks using cooperative ant-like agents. In *Proc. of the Congress on Evolutionary Computation, Canberra*, 2003.

[WHN05] O. Wittner, B. E. Helvik, and V. F. Nicola. Internet failure protection using hamiltonian p-cycles found by ant-like agents. *Journal of Network and System Management, Special issue on Self-Managing Systems and Networks*, 2005.

[Wit03] O. Wittner. *Emergent Behavior Based Implements for Distributed Network Management*. PhD thesis, NTNU, Dept. of Telematics, Norway, 2003.

[WM93] C. M. Woodside and G. G. Monforton. Fast allocation of processes in distributed and parallel systems. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):164–174, 1993.

[WN04] N. Widell and C. Nyberg. Cross entropy based module allocation for distributed systems. In *Proc. of the 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, pages 187–200, Berkeley, CA, USA, 2004. USENIX Association.

[WS08] G. Wittenburg and J. Schiller. A survey of current directions in service placement in mobile ad-hoc networks. In *Proc. of the 6th Annual IEEE Int. Conf. on Pervasive Computing and Communications*, pages 548–553, 2008.

[WSVY07] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. of the 4th USENIX Symp. on Networked Systems Design and Implementation*, 2007.

[XZF$^+$07] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2007.

[YG09] H. Yu and P. B. Gibbons. Optimal inter-object correlation when replicating for availability. volume 21, pages 367–384, 2009.

[YV05] H. Yu and A. Vahdat. Consistent and automatic replica regeneration. *ACM Trans. on Storage*, 1(1):3–37, 2005.

[Z$^+$04] M. Zlochin et al. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131:373–395, 2004.

[ZMAB03] D. Zagorodnov, K. Marzullo, L. Alvisi, and T. C. Bressoud. Engineering fault-tolerant tcp/ip servers using ft-tcp. In *Proc. of the 2003 Int. Conf. on Dependable Systems and Networks (DSN)*, 2003.

[ZXS02] X. Zhong, C.-Z. Xu, and H. Shen. Ip-based protocols for mobile internetworking. In *Proc. of ACM SIGCOMM*, 2002.

[ZXS04] X. Zhong, C.-Z. Xu, and H. Shen. A reliable and secure connection migration mechanism for mobile agents. In *Proc. of the 24th Int. Conf. on Distributed Computing Systems Workshops (ICDCSW)*, 2004.