A Branch and Price Approach for Deployment of Multi-tier Software Services in Clouds

Anders N. Gullhav*, Bjørn Nygreen

Department of Industrial Economics and Technology Management, The Norwegian University of Science and Technology, NO-7491, Trondheim, Norway

Abstract

This paper considers a service deployment problem that combines service placement and replication level decisions in a cloud computing context. The services are composed of multiple components that are to be placed on nodes in the private cloud of the service provider or, if the private cloud has limited capacity, partly in a public cloud. In the service delivery, the provider has to take into account the quality of service guarantees offered to his end-users. To solve the problem, we develop a branch and price algorithm, where the subproblems both are formulated as a linear mixed integer program and a shortest path problem with resource constraints (SPPRC) on a network with a special structure. The SPPRC can be solved by an exact label-setting algorithm, but to speed up the solution process, we develop a heuristic label-setting algorithm based on a reduced network and simplified dominance rule. Our results show that using the heuristic subproblem solver is efficient. Furthermore, the branch and price algorithm performs better than a previously developed pre-generation algorithm for the same problem. In addition, we analyze and discuss the differences in solutions that utilize resources in a public cloud to different degrees. By conducting this analysis we are able to identify some essential characteristics of good solutions.

Keywords: Branch and price, Shortest path problem with resource constraints, Multi-tier service, Replication, Cloud computing

1. Introduction

In this work, we are considering a service deployment problem of a software-as-a-service (SaaS) provider that offers a set of services to his end-users. In the provisioning, the SaaS provider (SP) must scale his services according to the performance and availability guarantees specified in the service level agreements (SLAs) contracts that define the services in terms of functionality and quality. Furthermore, the SP also has to decide where to run the services. A typical objective in this problem is to minimize the cost of provision, while fulfilling the service quality guarantees.

In principle, this decision problem can be solved statically or dynamically. Herein, we consider the demand to vary over time, but within certain periods, the demand is stationary in a stochastic sense. When these periods are sufficiently long and recurring, that is, they might reflect working

*Corresponding author

Email addresses: anders.gullhav@iot.ntnu.no (Anders N. Gullhav), bjorn.nygreen@iot.ntnu.no (Bjørn Nygreen)

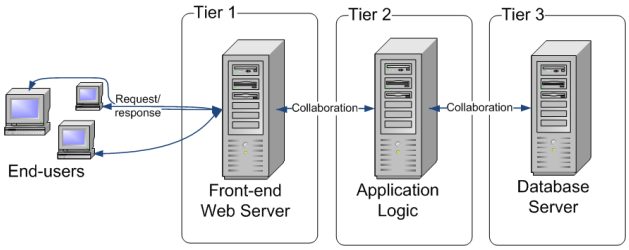

Figure 1: Illustration of a three-tier web service

hours, evenings, etc., it is possible to compute a stationary deployment solution for each period, and apply the appropriate solution when one enters a new period. If the infrastructure running the services is failure-prone, it is necessary to complement the stationary solution with a strategy to return from a failed state back to the stationary solution. This strategy can be based on migrating (by e.g., live migration [7]) or restarting service components on preallocated backup locations, as proposed in [5], or based on activating standby service components [8]. However, in cases where the demand is constantly fluctuating and not in a stationary state, it is necessary to solve the deployment problem dynamically, but such cases are not considered here.

The different SaaS services offered by the SP are represented by multi-tier services, which are services composed of several components collaborating to provide a service to the end-users. An example of a multi-tier service is a three-tier web service composed of a web server tier, an application tier and a database tier. Figure 1 illustrates the structure of a three-tier web service. Each tier corresponds to a software component, referred to as component throughout this paper, which runs on one or more virtual machines (VMs), which in turn run on physical servers. E.g. in periods of low service demand, the web server component of a service might run on only one VM, while with increased demand, the component is deployed by running multiple identical VMs in order to provide a service in accordance with the SLA. While a component might run on multiple VMs, a VM will only run one component. The considered SP owns and operates a limited set of servers, forming a private cloud, which are capable of running the VMs of the service components. An important operational cost component in a data center is the cost of energy, and a strategy used to minimize the cost of energy usage in the VM scheduling is to turn off servers that are not required with the current demand [16]. For the SP, there might be periods where the service demand is too high to be able to provide the services from the private cloud alone. In such cases the SP has the option to lease resources from a public infrastructure-as-a-service (IaaS) provider (e.g. Amazon [1]), denoted a public cloud provider. When the infrastructure used by the SP to provide his services is composed of both a private and a public cloud, this infrastructure is referred to as a hybrid cloud [22]. In cases where the SP maintains a private infrastructure, it is often desirable to fully utilize this infrastructure before leasing capacity from an IaaS provider.

The private and public clouds typically consist of a large amount of cheap, off-the-shelf hardware, which make the services prone to failures, and hence, make fault tolerance an important consideration in the deployment of cloud services. When a VM fails, one has to provide a new VM to maintain service. A technique to improve the fault tolerance is standby redundancy, that is, the principle of allocating standby resources that can be activated in case of a failure. The use of standby redundancy reduces the time from a failure until the moment the new VM is up and running. This time is commonly denoted failover time. Undheim et al. [28] present different ways to implement standby redundancy in a cloud context. Moreover, software systems and

conceptual frameworks that employ standby redundancy by running passive standby VMs on the infrastructure are proposed by Cully et al. [8] and Distler et al. [9].

In [12], we present a novel optimization model of the service deployment problem of the SP that includes decisions both related to the replication of the components of multi-tier services and related to the placement of the replicas of the components. In the model, each component of a service could be replicated into a number of load-balanced replicas, referred to as *active replicas*, and in addition, *passive replicas* are used to improve the fault tolerance of the component. However, since we are interested in the performance and fault tolerance of the whole service, not only the components, the selection of replication levels of the different components of a service is linked. In cases where different services interact through their placement (e.g., by running on the same servers), the most cost-efficient way to replicate the components of a given service is dependent of the replication of the components of other services. This is reflected in the model. The replication level decisions, i.e., deciding the number of active and passive replicas of each component of each service, are somewhat similar to the decisions of an optimization problem referred to as the redundancy allocation problem [20]. This problem consists of allocating parallel replicas to different subsystems in series so that the reliability is over a given threshold, while minimizing the cost.

The service deployment problem was modeled as a linear mixed-integer program (MIP), and solved using a commercial MIP solver in [12]. We also reformulated the problem and obtained a pattern-based formulation. The linear relaxation of the reformulation was shown to be much stronger than that of the former, *direct MIP formulation*. Nevertheless, the number of variables in the reformulation grew exponentially with the size of the problem, and we could only optimize over a small subset of the variables. Since we seek to find a stationary solution, we argue that one can spend some time searching for a near-optimal or optimal solution. If the solution quality is of more importance than the time to find a solution, we suggest using an exact solution method. Here, we propose a branch and price (B&P) algorithm, in which the master problem is based on the mentioned pattern-based formulation. The subproblem of the B&P is solved using a MIP solver and a label-setting algorithm. The latter seeks to find the shortest path in a network, which to our knowledge has a novel structure. While developing an exact label-setting algorithm, we observed that this algorithm has deficiencies related to its dominance rule.

A contribution of this paper is an efficient heuristic label-setting algorithm based on a reduced network and simplified dominance rule. Using this heuristic in conjunction with an exact MIP solver speeds up the B&P algorithm. However, in some nodes of the enumeration tree, no improving columns can be found, and hence, using the heuristic algorithm is ineffective. Another contribution of this paper is a simple rule to decide whether the heuristic algorithm should be used in a node, or the exact MIP solver should be called directly. A major question we seek to answer in our computational study is by which methods the subproblem should be solved. The paper also provides a discussion on how the size of the private cloud, relative to the service's resource requirements, affects the solution structure.

The outline of the paper is as follows. Next, in Section 2, we present some works related to the service deployment problem, and in Section 3, we describe the problem in more detail. Two variants of the problem are formulated in Section 4, before the B&P algorithm is explained in Section 5. More details of the algorithm are found in Appendix A. The numerical results of our experiments with the algorithm, along with a discussion of the results, are presented in Section 6. Finally, we conclude the paper in Section 7.

2. Related Work

As stated in the introduction, the part of the problem that regards the replication level decisions is related to the redundancy allocation problem, which has applications in many areas including in the design of computer systems. Ashrafi et al. [3] present optimization models with the goal to optimize the reliability of a software system. More recently, Meedeniya et al. [21] use multi-objective optimization to explore the trade-off between reliability and energy consumption when building redundancy into an embedded system. While the redundancy allocation problem is related to our work, this problem does not consider decisions associated with the placement of the redundant components as considered herein.

Regarding the part of the problem that relates to the placement of the VMs, there exists a lot of literature on scheduling and placement of VMs in clouds. A recent survey on resource management in clouds is found in Jennings and Stadler [17]. We also review some literature in [12]. Speitkamp and Bichler [27] present static optimization models for the problem of placing VMs on servers in enterprise data centers. In contrast to our work, they do not model services as consisting of multiple tiers, and neither do they treat replication of VMs. Dynamic optimization models that investigate the trade-off between energy usage and performance can be found in several works [2, 10, 19, 24]. Moreover, Google proposed another dynamic optimization model in the ROADEF/EURO challenge 2012 [25]. In relation to the B&P algorithm proposed in this paper, the solution approach of Kramer et al. [19] is interesting since it is one of the few approaches in the VM placement literature that is based on column generation. Another related solution approach is presented by Breitgand and Epstein [6], where column generation is used to solve a static service placement problem, both with and without considering the cost of migrating VMs. Like us, they require the provider to comply with requirements specified in SLAs, but while we consider the service deployment problem of a SaaS provider, they regard the decision problem of an IaaS provider. Both Kramer et al. [19] and Breitgand and Epstein [6] end the column generation after solving the linear relaxation of the restricted master problem to optimality. To obtain an integer solution, they solve this problem as an integer program, and thus, in contrast to the exact B&P algorithm proposed herein, their solution approaches remain heuristic algorithms. Shi et al. [26] consider a similar static placement problem that includes requirements specifying that a set of VMs should run on different nodes for fault tolerance reasons, or the a node can only run a set of VMs from the same user for security reasons. They develop both a linear integer programming formulation to obtain an optimal static placement solution, and faster heuristics for performing dynamic (online) placement of VMs. Furthermore, Goudarzi and Pedram [11] and Ardagna et al. [2] regard a dynamic placement problem of an IaaS provider, which considers placement of multi-tier services with decisions related to the load balancing and bounds on the performance. They model the relationship between resource allocation and performance by queuing theory, and develop non-linear integer programming formulations. While the latter works allow using several VMs for each tier, none of the reviewed placement literature so far, except our previous work, models the placement of passive backup replicas. However, Bin et al. [5] regard a related placement problem of an IaaS provider where the VMs are allocated a fixed number of backup locations to which they can be migrated in case of a failure. They develop a constraint programming model considering both the placement of VMs and the allocation of backup locations. While there are some similarities with our work, our work explicitly differs by modeling the VMs as part of multi-tier services with associated SLA requirements, and by regarding the number of passive replicas as variable.

3. Problem Description

We let $\mathcal{S}$ be the set of multi-tier services offered by the SP, and let $\mathcal{Q}_i$, $i \in \mathcal{S}$, be the set of components, i.e., tiers, of service i. A component $q \in \mathcal{Q}_i$ runs in one or more VMs (due to replication, discussed in Section 3.1), which in turn should run on a server, denoted a *node* in the following. A VM contains only one replica of one component, and hence there is a one-to-many relationship between components and VMs. A private cloud placement is an assignment of a VM running component q of service i to a node in the private cloud of the SP. The nodes provide resources like CPU power, memory and storage to the VMs, and we let the set $\mathcal{G}$ contain the different types of resources. We assume the nodes to be identical with equal resource capacities N_{Cg} of resource type $g \in \mathcal{G}$. The VMs placed in the private cloud are of given sizes, depending on the specific component running in the VM, and we let G_{Aiqg} be the amount of resource of type g assigned to a VM running component q of service i when placed on a node. For simplicity, we will denote component q of service i as the pair (i, q) in the following.

3.1. Quality of Service

The quality of service (QoS) levels of the services are stated in SLAs, and common QoS measures for web services include the response time, either the average or a percentile of the distribution, throughput and downtime [23]. Replication of the components of a service is used to manage the service's QoS, and the SP might choose to replicate a component into multiple active load-balanced replicas, i.e., VMs, to increase the service performance. To make the services fault-tolerant, the SP might additionally place passive back-up replicas on the nodes. In the treatment of replication in this paper, there is a one-to-one match between a replica and a VM, and the notions are used interchangeably. The replicas of the same component should not be placed on the same node, a requirement referred to as node-disjoint placement. This is to prevent that multiple replicas of a single component go down due to a single node failure. The passive replicas do not serve demand in a failure-free situation, and consume fewer resources than active replicas [8, 9]. Therefore, instead of being assigned G_{Aiqg} resources, a passive replica of the pair (i, q) is assigned $G_{Piqg} (< G_{Aiqg})$ resources of type g when placed on a node. Each node that runs at least one passive replica needs to maintain a pool of shared backup resources for activation of passive replicas. The size of this pool is a trade-off between fault tolerance and resource utilization since a small pool size will make it difficult to provide enough resources to a passive replica when it is activated; and a large pool means that a large amount of resources are unused in a failure-free situation. Herein, the pool size is set such that the passive replica requiring the largest increase in resource assignment when being activated can be activated. This means that a node can in the worst case only guarantee the activation of one passive replica at a time, and thus the number of passive replicas on a node is limited to N_P, which enforces a distribution of the passive replicas on the infrastructure.

In this paper, we will not consider a specific QoS measure, but instead assume that there exists a method to check if a service, with given replication levels of its components, satisfies the QoS guarantees. Gullhav et al. [13] present a method that takes the number of active and passive replicas of each component, the assigned resources (translated to service times of the requests) of the replicas and failure probabilities as input, and outputs an approximated response time distribution of the service. In [12], we introduced *replication patterns* in order to express the relationship between the number of replicas of each component and the QoS guarantees in a linear MIP. Thus, a replication pattern $r \in \mathcal{R}_i$ of service i is a combination of the number of active and passive replicas of each component in the service. We let these numbers be denoted by R_{Aiqr} and

R_{Piqr}, which means that if replication pattern r is chosen for service i, one has to deploy R_{Aiqr} active replicas and R_{Piqr} passive replicas of each component $q \in \mathcal{Q}_i$. We assume that the set of replication patterns $\mathcal{R}_i$ for each service i is given as input, and that all the replication patterns in these sets satisfy the QoS guarantees.

The performance of the underlying network is neglected in our models. Within a single data center, neglecting the network latency might be fair, but between clouds of different geographical locations, one might argue that the network latency plays a role. On the other hand, this negligence simplifies the models considerably. However, there is a bound on the number of different services that can be run from a single node, N_S. With this constraint, the model will implicitly drive different components from the same service to be run on the same nodes, and so, the amount of inter-node communication, stemming from the collaboration between components of the same service, is expected to decrease.

3.2. Placement in Public Clouds

So far, we have taken for granted that the private cloud has enough resources to run all the services, and only discussed the placement of the replicas in the private cloud of the SP. Now, we consider the case where the private cloud does not have enough nodes to run all services, and the SP gets the option, or is rather forced, to lease extra resources from a public cloud provider. We let N_N denote the number of nodes in the private cloud. In this case, leasing means to run a replica of a pair (i, q) in a VM of an appropriate size in the data centers of the public cloud provider at a cost. We do not assume that the public cloud provider supports the activation of passive replicas, so passive replicas are always run in the private cloud. Since we do not consider network effects explicitly in this work, we can consider the public cloud resources as a generic resource pool, possibly consisting of offers from several public cloud providers. We can also determine before the optimization the cheapest way (at which provider, and in what geographical location, etc.) to place an active replica of the pair (i, q) in the public cloud. The cost of this placement is denoted C_{Ciq}, and it is dependent on the resource requirement of the active replica. Generally, the public cloud providers offer predetermined, fixed *VM types* differing by resource capacity and cost, and typically the resource capacities of the VM types are coarse grained. For example, Amazon [1] offers seven general-purpose VM types (referred to as *Instances* by Amazon), where the resource capacities double from one VM type to the next. The amount of resource assigned to an active replica of pair (i, q), G_{Aiqg}, is set before considering the available VM types offered in the public cloud. When selecting the VM type that is going to be used for placement of an active replica in the public cloud, one has to select a VM type that provides enough resources for each $g \in \mathcal{G}$, and one will select the cheapest VM type providing this. This means that two pairs (i_1, q_1) and (i_2, q_2) with $G_{Ai_1q_1g} \neq G_{Ai_2q_2g}$ for all $g \in \mathcal{G}$, might have to run in the same type of VM in the public cloud at an identical cost.

3.3. Cost of Deployment

The cost of deployment can consist of different cost components related to the service provisioning. When only considering deployment in the private cloud of the SP, we consider the cost of energy usage as the sole cost component. Like, several other optimization models for VM placement, we minimize the number of nodes that is turned on. A reason for this choice is that a node that is turned on, but in an idle state, consumes as much as 70 percent of its peak power [4], and hence, running the services on as few nodes as possible will reduce the cost substantially. Another situation arises when a hybrid cloud is used for placement. The motivation behind this scenario

is that the private cloud does not have enough resources for running all services, and thus the SP has to use a public cloud. In this scenario, the important cost component to minimize is the cost of placing VMs in the public cloud, while fully utilizing the nodes in the private cloud.

4. Mathematical Formulations

Two formulations of the service deployment problem are presented next. The first formulates the *private cloud model*, where only placement in the private cloud is considered, while the second formulates the *hybrid cloud model* that also allows for placement in a public cloud. Both formulations were proposed [12], but are repeated here as they form the master problem of our B&P algorithm. To model the placement of replicas, they use *node patterns*, which represent a feasible assignment of replicas to a node, that is, the assignment must respect the resource capacities of the node, the requirement for shared backup resources, the requirement for node-disjoint placement of replicas, and upper bounds on the number of passive replicas and the number of services on a node. Since the nodes are considered identical, any node pattern can represent any node in the private cloud. In this section, we take the set of node patterns as granted, but in Section 5, we will formulate a subproblem for the B&P, which is used to generate node patterns dynamically.

4.1. Private Cloud Model

In the formulation below, the integer variables x_b denote the number of times each node pattern $b \in \mathcal{B}$ is used in the solution, where $\mathcal{B}$ represents the set of node patterns. The parameters $W_{biq} \in \{0,1\}$ and $V_{biq} \in \{0,1\}$ indicate the placement of an active replica ($W_{biq} = 1$) and a passive replica ($V_{biq} = 1$) of the pair (i,q) in node pattern b. Furthermore, the binary variables y_{ir} indicate the selection of a replication pattern $r \in \mathcal{R}_i$ for service i.

$$\min z_P = \sum_{b \in \mathcal{B}} x_b \tag{4.1}$$

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1 \quad \forall i \in \mathcal{S} \tag{4.2}$$

$$\sum_{b \in \mathcal{B}} W_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0 \; \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{4.3}$$

$$\sum_{b \in \mathcal{B}} V_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Piqr} y_{ir} = 0 \; \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{4.4}$$

$$x_b \in \mathrm{Z}_+ \quad \forall b \in \mathcal{B} \tag{4.5}$$

$$y_{ir} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall r \in \mathcal{R}_i \tag{4.6}$$

The objective function (4.1) minimizes the number of used node patterns, and thereby also nodes. The equalities (4.2) ensure that one replication pattern is selected for each service. Furthermore, the equalities (4.3) and (4.4) establish the relation between the node pattern variables and the replication pattern variables, and so provide that the correct number of active and passive replicas of each pair (i,q) are placed on the nodes, according the chosen replication pattern. Finally, (4.5) and (4.6) define the node pattern and replication pattern variables as non-negative integer and binary, respectively.

4.2. Hybrid Cloud Model

The motivational case of placing replicas in a public cloud is that there exist time periods where the SP does not have enough capacity to run all services in his private cloud. We let w_{Ciq} be integer variables denoting the number of active replicas of (i, q) placed in the public cloud. The mathematical formulation of the hybrid cloud model is given below.

$$\min z_H = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{Ciq} w_{Ciq} \tag{4.7}$$

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1 \quad \forall i \in \mathcal{S} \tag{4.8}$$

$$\sum_{b \in \mathcal{B}} W_{biq} x_b + w_{Ciq} - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{4.9}$$

$$\sum_{b \in \mathcal{B}} V_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Piqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{4.10}$$

$$\sum_{b \in \mathcal{B}} x_b \leq N_N \tag{4.11}$$

$$w_{Ciq} \in \mathrm{Z}_+ \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{4.12}$$

$$x_b \in \mathrm{Z}_+ \quad \forall b \in \mathcal{B} \tag{4.13}$$

$$y_{ir} \in \{0, 1\} \quad \forall i \in \mathcal{S}, \forall r \in \mathcal{R}_i \tag{4.14}$$

The objective (4.7) minimizes the cost of placing active replicas in the public cloud. The equalities (4.8) correspond to (4.2) in the private cloud model. The balance equalities (4.9) now include a term for the public cloud placement of active replicas, while (4.10) remains identical to (4.4). The inequality (4.11) reflects that the number of nodes is limited by setting an upper bound on the sum of node patterns. At optimum (4.11) will be satisfied as an equality. At last, (4.12) - (4.14) define the decision variables as integer or binary. It should be noted that the w_{Ciq} variables are naturally integer as long as all x_b and y_{ir} variables are non-fractional.

5. The Branch and Price Approach

In this section, we describe the algorithmic features of our B&P approach, and especially concentrate on the solution methods used for solving the subproblem. In short, B&P is a solution method that uses column generation in a branch and bound (B&B) framework. For the models of Section 4, it would be possible in small cases to include all feasible node patterns and solve the models as integer programs (IPs) using B&B. However, for realistic cases, this is not practical, and thus, we propose to solve the model using B&P. In B&P, one generates new node patterns in each B&B node until no further profitable node patterns exist. This is done by alternating between solving the master problem, i.e., the formulations in Section 4, as a linear program (LP), and a subproblem that generates new node patterns, until no node patterns with negative reduced cost exists. The subproblem minimizes the reduced cost of a new node pattern by using the dual variables of the master problem as coefficients in the objective function. When no node pattern with negative reduced cost is found, one finishes the current B&B node and uses the same criteria for branching or pruning as in the traditional B&B. However, the branching rules used in B&P

applications often differ from the typical branching rule in the B&B framework. The branching rule used herein is discussed in Section 5.3. Since the master problem LP only contains a subset of the feasible node patterns, this problem is referred to as the restricted master problem (RMP), and this is solved using a commercial LP solver. To obtain integer solutions by other means than branching, one can solve the RMP as an IP at different points in time. The results of the experiments presented in Section 6 show that we find all our best integer solutions this way. This strategy is also used in the literature, e.g., by Gunnerud et al. [14], which find all their integer solutions by solving an IP.

An overview of the B&P algorithm is shown in Algorithm 1. To keep the pseudocode compact we let z, $\boldsymbol{x}$, and $\boldsymbol{y}$, with vectors in bold font, denote the RMP's LP objective value, the node pattern solution vector, and the replication pattern solution vector; and $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and η be the dual variables of the RMP. $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ refer to dual variables of the balance equations (4.3) and (4.4) (resp. (4.9) and (4.10)), $\boldsymbol{\gamma}$ stems from the branching constraints introduced in Section 5.3, while η represents the dual variable of (4.11), and is only present in the hybrid cloud model. Moreover, ζ, $\boldsymbol{w}$, and $\boldsymbol{v}$ denote, respectively, the reduced cost of the new node pattern, the binary vector indicating the active replicas deployed in the new node pattern, and the binary vector indicating the passive replicas. The vectors $\boldsymbol{w}$ and $\boldsymbol{v}$ correspond to the node pattern coefficients W_{biq} and V_{biq}. In Algorithm 1, symbols with a superscript I refer to the current incumbent solution.

Algorithm 1 Pseudocode of the branch and price algorithm

Require: an initial set of node patterns, $\mathcal{B}$, such that feasibility is assured.
Initialize tree by creating a root node
$z^I \leftarrow \infty$ // *Initialize the objective value of the current incumbent to* ∞
while there exists unsolved nodes **do**
  Get next unsolved B&B node accoring to the best first strategy
  repeat
    $(z, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \eta) \leftarrow$ `solveRMP`$(\mathcal{B})$
    $(\zeta, \boldsymbol{w}, \boldsymbol{v}) \leftarrow$ `solveSubproblem`$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \eta)$
    if $\zeta < 0$ **then**
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{w}, \boldsymbol{v})\}$ //*add the new node pattern, represented by* $(\boldsymbol{w}, \boldsymbol{v})$*, to* $\mathcal{B}$
    end if
  until $\zeta \geq 0$ //*implemented as check against a small positive number*
  if $z < z^I$ **then**
    if solution $(\boldsymbol{x}, \boldsymbol{y})$ is fractional **then**
      Branch and add new nodes with parent objective value z to the set of unsolved nodes
    else
      $z^I \leftarrow z$, $(\boldsymbol{x}^I, \boldsymbol{y}^I) \leftarrow (\boldsymbol{x}, \boldsymbol{y})$ // *store the new best solution*
      Remove all unsolved nodes with parent objective value worse than z^I (prune)
    end if
  end if
  if /*any condition*/ **then** {// *specified in Section 6.1*}
    $(z, \boldsymbol{x}, \boldsymbol{y}) \leftarrow$ `solveMasterIP`$(\mathcal{B})$ //*Solve the RMP as an IP*
    if $z < z^I$ **then**
      $z^I \leftarrow z$, $(\boldsymbol{x}^I, \boldsymbol{y}^I) \leftarrow (\boldsymbol{x}, \boldsymbol{y})$ // *store the new best solution*
      Remove all unsolved nodes with parent objective value worse than z^I (prune)
    end if
  end if
end while
return $(\boldsymbol{x}^I, \boldsymbol{y}^I)$

In the B&P implementation, we have done two small changes to the master problem formula-

tions. The equalities (4.3)-(4.4) and (4.9)-(4.10) are changed to $\geq$ constraints. This has at least two advantages. Firstly, it makes the dual variables of these constraints non-negative instead of free, which might help to reduce the instability of the dual solutions [32]. Secondly, the feasible region is enlarged which should make it easier to find a feasible solution. However, this also means that one could obtain solutions that deploy more replicas than required. Especially, the passive replicas, which require relatively small amounts of resources, are susceptible to this.

The subproblem that is used to generate new node patterns can be formulated and solved in various ways. In general, subproblems can often be solved as MIPs, and in many applications, the subproblem can also be formulated as a shortest path problem with resource constraints (SPPRC), which is solved using a label-setting algorithm [15]. In addition, heuristic solution methods based on label-setting algorithms have been utilized successfully to solve different types of subproblems [15]. Solving the subproblem by a label-setting algorithm has an advantage over a MIP solver in that it is relatively easy to extract more than one node pattern in each column generating iteration. Herein, we have formulated the subproblem as a MIP and an SPPRC. For the SPPRC, we develop both an exact and a heuristic label-setting algorithm, outlined in Section 5.2. However, when using a heuristic subproblem solver, one must complement the heuristic solver with an exact subproblem solver to maintain an exact B&P algorithm.

5.1. Formulating the Subproblem as a MIP

The subproblem can be formulated as the following MIP, where the α_{iq} and β_{iq} are the dual variables of the constraints (4.3) and (4.4) (respectively (4.9) and (4.10)) of the private cloud (respectively hybrid cloud) model. Furthermore, the formulation uses binary placement variables w_{iq} and v_{iq}, corresponding to the node pattern coefficients W_{biq} and V_{biq}; and binary variables s_i to indicate whether at least one replica of service i is placed on the node, or not. The amount of shared backup resources for activation of passive replicas are represented by the variables m_g.

$$\min \zeta = 1 - \Big(\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \alpha_{iq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \beta_{iq} v_{iq} \Big) \tag{5.1}$$

$$w_{iq} + v_{iq} \leq s_i \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.2}$$

$$\sum_{i \in \mathcal{S}} s_i \leq N_S \tag{5.3}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} v_{iq} \leq N_P \tag{5.4}$$

$$m_g - (G_{Aiqg} - G_{Piqg}) v_{iq} \geq 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall g \in \mathcal{G} \tag{5.5}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G_{Aiqg} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G_{Piqg} v_{iq} + m_g \leq N_{Cg} \quad \forall g \in \mathcal{G} \tag{5.6}$$

$$m_g \geq 0 \quad \forall g \in \mathcal{G} \tag{5.7}$$

$$w_{iq} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.8}$$

$$v_{iq} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.9}$$

$$s_i \in \{0,1\} \quad \forall i \in \mathcal{S} \tag{5.10}$$

The objective function (5.1) to be minimized corresponds to the reduced cost of a node pattern in the private cloud model. The expression for the reduced cost in the hybrid cloud model is discussed in Section 5.4. The constraints (5.2) ensure node-disjoint placement, and at the same time force s_i to take value 1, as long as there is at least one replica from service i selected in the node pattern. Moreover, (5.3) and (5.4) set upper bounds on the number of different services, and the number of passive replicas that can run on a node, respectively. The resource capacities of the nodes are handled by (5.6), where the first and second term account for the resources assigned to the active and passive replicas, and the third term accounts for the shared backup resources. As discussed in Section 3.1, the shared backup resources should be greater than or equal to the increase in assigned resources when activating the passive replica with the largest resource increase. This is enforced by the inequalities (5.5). Lastly, (5.7) - (5.10) give the variable definitions.

5.2. Formulating the Subproblem as an SPPRC

The underlying idea of the SPPRC formulation is to let the service components represent the vertices of a graph (two vertices per component, corresponding to an active and a passive replica), and then appropriately connect the vertices with directed arcs, such that the resulting graph is acyclic. The resources of the SPPRC include the reduced cost, the amount of a node's resources assigned to replicas, the amount of a node's resources reserved to allow passive replicas to be activated, in addition to the number of passive replicas and the number of different services placed on the node. The goal is to find the minimum reduced cost path from a dummy source vertex to a dummy sink vertex, which is feasible with respect to the other resources. Such a feasible path corresponds to a feasible node pattern, and a label-setting algorithm is used to obtain these paths.

The label-setting algorithm runs on a directed acyclic graph $(\mathcal{V}, \mathcal{A})$ which can be viewed as a two-layered network. The upper layer network is illustrated in Figure 2, and is composed of the dummy source and sink vertices, σ_0 and τ_0, and several *service blocks*, one for each service in $\mathcal{S}$. From σ_0, there is an arc to every service block; and from each service block, there are arcs to all other service blocks of higher order, in addition to an arc to τ_0. The service blocks can be ordered in different ways, and the ordering can change between separate calls to the label-setting algorithm. Designing the network with the service blocks as central elements is beneficial because of the restricted number of different services that can be visited, cf. constraint (5.3). That is, no more than N_S service blocks can be visited by a path.

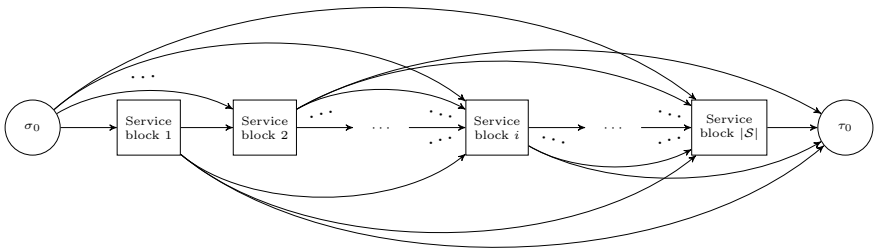

Figure 2: The upper layer network of the SPPRC formulation

Each service block is a network in itself, and the structure of these subnets is depicted in Figure 3. The service block is composed of four types of vertices: one σ-vertex, one τ-vertex, one or more a-vertices and one or more p-vertices. The σ-vertex, labelled σ_i in Figure 3, is the entrance of

the service block and every arc into the service block is directed to this vertex. Similarly, the τ-vertex, labelled τ_i in Figure 3, is the exit of the service block, and all arcs from a service block to another service block or to τ_0 are directed out from the τ-vertex. Each a-vertex in the service subnet represents an active replica, i.e. the vertex a_{i1} in Figure 3 represents an active replica of the first component of service i; and a path visiting this vertex is interpreted as a placement of such a replica in the node pattern corresponding to the path. Analogously, each p-vertex in the service subnet represents a passive replica. Again, the ordering of the components can change between separate calls to the label-setting algorithm. From the σ-vertex, there is an arc to all a-vertices and p-vertices; and from an a-vertex or p-vertex, there are arcs to all a-vertices and p-vertices of higher order, in addition to the τ-vertex. Note that, there are no arcs between a-vertices and p-vertices representing active and passive replicas of the same component, and no arc directly from the σ-vertex to the τ-vertex.

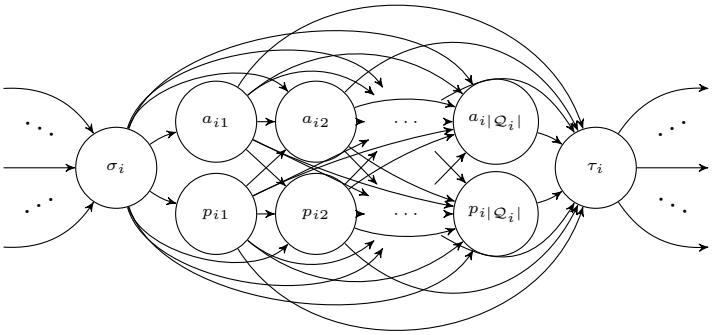

Figure 3: The subnet of a service block in the SPPRC formulation

Algorithm 2 outlines the basic structure of the label-setting algorithm. The pseudocode is kept compact and general, and thus does not show the special properties of the network. Each vertex ν maintains a set of labels, $\mathcal{L}_\nu$, and the algorithm is initialized by letting the label set of the dummy source vertex σ_0 have one label with reduced cost 1, representing a path containing only the source vertex σ_0. Then, all vertices are considered in topological order; that is, if $\mathcal{A}$ contain an arc from ν to ω, vertex ν is considered before ω. Before the labels of a vertex ν is extended, all dominated labels in $\mathcal{L}_\nu$ are removed. At the end, when the dummy sink vertex τ_0 is considered, all labels in $\mathcal{L}_{\tau_0}$ with negative reduced cost are returned. The extension and domination of labels is detailed in Appendix A.1 and Appendix A.2, respectively.

Since a label-setting algorithm without a dominance rule essentially is a full enumeration algorithm, a critical algorithmic element is the efficiency of this rule. When conducting experiments with the label-setting algorithm on the SPPRC formulation, it became clear that the interaction between the shared backup resources, corresponding to m_g in the subproblem MIP, and the total amount of resources assigned to replicas on the node, made the dominance quite weak. However, fixing m_g before calling the label-setting algorithm would lead to much faster, but heuristic, solution method. There are two implications of this fixing that will increase the speed of the algorithm: the dominance becomes more efficient, and one can reduce network size by disregarding the vertices representing passive replicas with $G_{Aiqg} - G_{Piqg} > m_g$ for at least one g. We consider two types of fixings: m_g fixed to zero for all g, and m_g fixed to positive values. When using this heuristic

Algorithm 2 Pseudocode of the label-setting algorithm

Require: a directed acyclic graph $(\mathcal{V}, \mathcal{A})$
$\mathcal{L}_{\sigma_0} \leftarrow \{\ell_0\}$
for all vertices $\nu \in \mathcal{V}$ in topological order **do**
  if domination should be conducted **then**
    Domination: find and remove all dominated labels in $\mathcal{L}_\nu$
  end if
  for all labels $\ell \in \mathcal{L}_\nu$ **do**
    for all feasible extensions of label ℓ **do**
      Extension: extend label ℓ to obtain label k
      $\omega \leftarrow$ vertex of k
      $\mathcal{L}_\omega \leftarrow \mathcal{L}_\omega \cup \{k\}$
    end for
  end for
end for
return all labels (paths) in $\mathcal{L}_{\tau_0}$ with negative reduced cost

label-setting algorithm in the B&P, the label-setting algorithm is run twice in each call to the subproblem solver (Line 7 in Algorithm 1): one run with a zero-valued m_g, and one run with at least one positive m_g. More details on this heuristic dominance and network reduction, in addition to details on how the positive values of the m_g variables are set, are found in Appendix A.4.

5.3. Branching

In B&P, it is well-known that branching directly on the variables generated by the subproblem results in an unbalanced tree. Such a branching rule would also be difficult to implement in the subproblem, and could destroy its structure [31]. While it is not efficient to branch directly on the x_b variables, one can use the traditional branching rules for the binary variables y_{ir}. However, in our B&P algorithm, we prioritize branching on the node patterns as long as there exist fractional x_b variables, and we only describe the branching rule for the node patterns here. Regarding the w_{Ciq} variables of the hybrid cloud model, we pointed out in Section 4.2 that these variables are naturally integer, and hence, we do not branch on these variables.

Several works, including [29, 31, 33], discuss branching rules which ideas can be applied in our problem. Generally, these rules select a subset of the generated columns, such that the sum of the corresponding column variables are fractional. Based on this subset, one adds a constraint to the master problem that either bounds this sum to be less than or equal to the fractional value rounded down, or conversely, bound this sum to be greater than or equal to the fractional value rounded up. However, the rule to select the subset of columns has to be designed such that it can be handled by the subproblem. For a cutting stock problem, Vance [29] uses a branching rule that selects a set of rows, denoted a *branching set*, in the master problem, and chooses the subset of columns to branch on as the columns with coefficients greater than a certain value in each selected row. We base our rule on this idea, but limit ourselves to consider pairs of rows.

First, we define $\mathcal{J}$ as the set of all branching sets of cardinality two, i.e., an element $j \in \mathcal{J}$ corresponds to a pair of rows in (4.3)-(4.4) (analogously (4.9)-(4.10)), which means that j can either represent two active replicas, two passive replicas, or one active and one passive replica. Because replicas of the same component are required to run on different nodes, it is only meaningful that the two rows of a branching set corresponds to two different service components. We use the symbols $H_{Ajiq} \in \{0, 1\}$ to indicate if the row (i, q) in (4.3) is included in the branching set j ($H_{Ajiq} = 1$); and likewise $H_{Pjiq} \in \{0, 1\}$ to indicate if the row (i, q) in (4.4) is included in the branching set

j ($H_{Pjiq} = 1$). Moreover, let us define the set $\mathcal{B}_j$ as the current set of node patterns that have coefficients (W_{biq} or V_{biq}) equal to one in the two rows represented by the branching set j, cf. (5.11).

$$\mathcal{B}_j = \{b \in \mathcal{B} \mid \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \ (W_{biq} \geq H_{Ajiq} \wedge V_{biq} \geq H_{Pjiq})\} \quad \forall j \in \mathcal{J} \tag{5.11}$$

Using the above definitions, branching is done by selecting a branching set $j \in \mathcal{J}$, such that

$$\sum_{b \in \mathcal{B}_j} x_b = \phi \notin \mathbb{Z}$$

is fractional in the current solution; and then use the following inequalities as branching constraints in the master problem:

$$\sum_{b \in \mathcal{B}_j} x_b \leq \lfloor \phi \rfloor \quad \text{and} \quad \sum_{b \in \mathcal{B}_j} x_b \geq \lceil \phi \rceil.$$

Therefore, at a given B&B node t in the tree, one would have a set $\bar{\mathcal{J}}_t$ of active branching sets representing down branches (upper bounds), and a set $\underline{\mathcal{J}}_t$ of active branching sets representing up branches (lower bounds). The branching constraints are written as (5.12) and (5.13), where U_{tj} and L_{tj} are set to the fractional value rounded down and up, respectively.

$$\sum_{b \in \mathcal{B}_j} x_b \leq U_{tj} \quad \forall j \in \bar{\mathcal{J}}_t \tag{5.12}$$

$$\sum_{b \in \mathcal{B}_j} x_b \geq L_{tj} \quad \forall j \in \underline{\mathcal{J}}_t \tag{5.13}$$

We need to point out that it could happen that an active branching set $\hat{\jmath} \in \bar{\mathcal{J}}_t$ (or $\underline{\mathcal{J}}_t$) is used again when a new branching decision is made. In such a situation, the first child of B&B node t, say t_1, will have its current bound $U_{t_1\hat{\jmath}} < U_{t\hat{\jmath}}$ (or $L_{t_1\hat{\jmath}} > L_{t\hat{\jmath}}$), i.e. tightened, while the other child, say t_2, will have $\hat{\jmath}$ as element in both sets of active branching sets, that is, $\hat{\jmath} \in \bar{\mathcal{J}}_{t_2} \cap \underline{\mathcal{J}}_{t_2}$.

Furthermore, the branching rule will not make a node pattern invalid in any B&B node, which means that we do not need to keep track of whether a node pattern is valid or not in each B&B node. In addition, when we solve the RMP as an IP, we disregard all branching constraints and optimize over all node patterns found so far.

A question that is not yet addressed is if this branching rule leads to a complete algorithm, that is, can we guarantee that one could always eliminate fractional x_b variables using this rule? Proposition 5 in [33] states that for the cutting stock problem with columns as binary vectors, one might have to use branching sets with cardinality greater than the maximum value of the right hand sides in the master problem. If we had fixed all the replication pattern variables y_{ir} in the enumeration tree to obtain fixed right hand sides in (4.3) - (4.4), the maximum right hand side which could possibly be observed in a B&B node corresponds to:

$$\bar{R} = \max\{\max_{i \in \mathcal{S}, q \in \mathcal{Q}_i, r \in \mathcal{R}_i} \{R_{Aiqr}\}, \max_{i \in \mathcal{S}, q \in \mathcal{Q}_i, r \in \mathcal{R}_i} \{R_{Piqr}\}\}.$$

In the test instances used in Section 6, this number would be much larger than two. Nevertheless, when setting upper bounds on the run time, as done in Section 6, we have not encountered situations where using branching sets with larger cardinality than two have been necessary to eliminate a fractional solution. Yet, we cannot be certain that if the maximum run time is increased, we will still be able to branch successfully on branching sets of cardinality two only.

5.3.1. Modifications of the Subproblem Formulations

The dual variables of the branching constraints (5.12) and (5.13), $\bar{\gamma}_j$ for all $j \in \bar{\mathcal{J}}_t$ and $\underline{\gamma}_j$ for all $j \in \underline{\mathcal{J}}_t$, need to be considered in the computation of the reduced cost of a new node pattern. In the MIP formulation of the subproblem this is done by introducing new binary variables, $\bar{p}_j$ for all $j \in \bar{\mathcal{J}}_t$, and $\underline{p}_j$ for all $j \in \underline{\mathcal{J}}_t$, which take value 1 if the branching set j is included in the new node pattern. With this definition the modified objective function is given in (5.14). Moreover, the inequalities (5.15)-(5.16) ensure that the $\bar{p}_j$ and $\underline{p}_j$ variables take the correct value. Note that since $\bar{\gamma}_j \leq 0$ and $\underline{\gamma}_j \geq 0$, the objective function will try to make $\bar{p}_j, j \in \bar{\mathcal{J}}_t$, equal to zero, and hence we only have to force it to take value one in case the new node pattern will include the branching set j. For $\underline{p}_j, j \in \bar{\mathcal{J}}_t$, we have the opposite situation as the objective will try to make the variable equal to one, and therefore we have to force the variable to zero as long as the new node pattern will not include the branching set. Hence, each new branching constraint in the master problem will lead to one new binary variable and one new constraint in the MIP formulation of the subproblem.

$$\min \zeta_t = 1 - \Big(\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \alpha_{iq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \beta_{iq} v_{iq} + \sum_{j \in \bar{\mathcal{J}}_t} \bar{\gamma}_j \bar{p}_j + \sum_{j \in \underline{\mathcal{J}}_t} \underline{\gamma}_j \underline{p}_j \Big) \tag{5.14}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Ajiq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Pjiq} v_{iq} - \bar{p}_j \leq 1 \quad \forall j \in \bar{\mathcal{J}}_t \tag{5.15}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Ajiq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Pjiq} v_{iq} - 2\underline{p}_j \geq 0 \quad \forall j \in \underline{\mathcal{J}}_t \tag{5.16}$$

It is also necessary to modify the SPPRC formulation and label-setting algorithm to account for the branching decisions. Details on this is found in Appendix A.3.

We emphasize that adding branching constraints to the master problem and implementing the branching decisions in the subproblem formulations make the problems harder to solve. Therefore, it might be desirable to select B&B nodes high up in the tree when selecting new nodes in the B&P algorithm (cf. line 4 of Algorithm 1). Hence, we have implemented best first as the node selection strategy of our algorithm.

5.4. Generating Node Patterns in the Hybrid Cloud Model

In order to generate new improving node patterns in the hybrid cloud model (4.7) - (4.14), we have to make some small modifications to the subproblem. When computing the reduced cost of a new node pattern, we now have to take into account the dual variable η of constraint (4.11). Moreover, the objective coefficient of a new node pattern in the master problem objective (4.7) is zero, not one as in the private cloud model. The modified objective function of the MIP formulation of the subproblem is given in (5.17) below. Note that $\eta \leq 0$, and that the first term of the modified objective function is constant.

$$\min \zeta_t = -\eta - \Big(\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \alpha_{iq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \beta_{iq} v_{iq} + \sum_{j \in \bar{\mathcal{J}}_t} \bar{\gamma}_j p_j + \sum_{j \in \underline{\mathcal{J}}_t} \underline{\gamma}_j p_j \Big) \tag{5.17}$$

Like the modifications in the MIP formulation, the modifications in the SPPRC formulation and the label-setting algorithm are only minor. Instead of initializing the label-setting algorithm with a label with reduced cost 1, one initializes the algorithm with a label with reduced cost $-\eta$. Otherwise, the algorithm remains the same.

6. Numerical Results and Discussion

The main goal of our experimental study is to evaluate the different solvers for the subproblem. In addition, we are comparing the proposed B&P algorithm with the solution method presented in our earlier work [12]. The comparisons are done on both the private cloud and hybrid cloud model. At the end of the section, we will also investigate and present some characteristics of the solutions of the hybrid cloud model. Before the results are presented and discussed, we will give a description of the setup of the experiments.

6.1. Experimental Setup

We have implemented the B&P and label-setting algorithms in C++, and compiled the code with GCC version 4.8.2 with option `-O3`. The experiments have been run on a CentOS 5.8 machine with a dual core 3.0GHz Intel E5472 Xeon processor and 16 GB of memory. The Xpress-Optimizer version 27.01.02 of the FICO Xpress Optimization Suite version 7.8 was used for solving the master problem and the MIP formulation of the subproblem. The MIP solver has been able to utilize eight threads in the B&B. All experiments have been given a maximum run time of five hours.

In the experiments, we have not used real data. Even so, we believe that the data we have used realistically represent real world cases. The test instances used are of 6 different sizes, ranging from 5 services to 50 services, and for each instance size, we have 5 cases. The cases are generated based on 10 different dummy services with between 3 and 5 components each and different resource requirement characteristics. On average, each service consists of 4 components, and hence, the total number of components in the largest case with 50 different services amounts to 200. Each of the dummy services specifies distributions from which the G_{Aiqg} and G_{Piqg} parameters are randomly generated, based on a seed, and the components of a service have different distributions to imitate real SaaS services. The replication pattern data are also different among the cases: the average number of replication patterns per service is 7.5, and the average number of active and passive replicas per component over all replication patterns is 3.67 and 2.49, respectively. In all cases, $N_S = 3$ and $N_P = 4$; and we consider the CPU as the only resource type in the experiments. The average value of G_{Aiqg} and G_{Piqg} are 23.0% and 1.60%, given in percent of the CPU capacity of the nodes. Furthermore, the median of the maximum (minimum) G_{Aiqg} and G_{Piqg} over all cases is 45.0% and 3.00% (8.00% and 0.333%).

Regarding the experiments on the hybrid cloud model, we also have to set a bound on the number of nodes in the private cloud, N_N. For each of the test cases described above, we can construct different cases for the hybrid cloud model by changing the N_N parameter, and we have chosen to give this parameter values based on the best bound obtained on the corresponding test case in the private cloud experiments. Specifically, we have set N_N to 75% or 90% of the best bound, rounded up to the nearest integer. For simplicity, we will refer to these percentages as the *private cloud coverage*. When placed in the public cloud, an active replica, say (i, q), is run in a VM which capacity is at least as large as G_{Aiqg}. Typically, the public cloud providers offer predetermined VM types of different cost and capacity, and most often the cost per resource unit are constant over all VM types. Herein, we have used the hypothetical VM types listed in Table 1, which cost and capacity structures resemble the ones offered by Amazon (see [1]). We assume that there exists in total 6 VM types offered by 2 different public cloud providers. Since our mathematical model does not distinguish between providers, we can compute before the optimization the cost of placing an active replica of each component (i, q) in the public cloud. That is, C_{Ciq} equals the cost of the cheapest VM that has larger capacity than G_{Aiqg}. Note that the cost values in the table are not

given units. Since we are not comparing the cost of running services in the private cloud with the cost of placement in the public cloud, the units are not important. However, the relative differences in cost and capacity of the VM types are reflecting the reality.

Table 1: Data of the public cloud VM types used in the hybrid cloud experiments. The capacity is given in percentage of the private cloud node capacity.

Provider 1		Provider 2	
Cost	Capacity	Cost	Capacity
10	10%	15	15%
20	20%	30	30%
40	40%	60	60%

The test cases are primarily labelled based on the number of services, but also given a suffix `a` to `e` distinguishing the five different cases with an equal number of services. E.g., the five private cloud test cases with 20 services are labelled from `P20-a` to `P20-e`. When all cases with 20 services are referred to as a whole, we ignore the suffix. Similarly, the hybrid cloud cases with 20 services are labelled `H20-a` to `H20-e`.

In the B&P algorithm, one has the option of calling an IP solver (the MIP solver of Xpress) on the RMP (cf. line 20 of Algorithm 1). In the implementation used in the experiments, we have chosen to call the IP solver directly after the root node is solved, and then after every 100th solved B&B node. In order not to spend too much time on solving the IPs, we have set the maximum run time of the IP solver to 600 seconds. Another parameter, which has to be set, is the maximum number of node patterns that is added to the RMP when the label-setting algorithms are used as solution method for the subproblem. For the exact label-setting algorithm (E-LSA) we have set the maximum number of node patterns to 20. Since the heuristic label-setting algorithm (H-LSA) solves two different networks in each iteration, one with $m_g = 0$ and one with $m_g > 0$, we will add at most 10 node patterns from each of the two runs to the RMP.

6.2. Evaluation of the Solution Methods

As we want to maintain an exact solution approach, we have to complement the heuristic solution method with an exact solution method for the subproblem. Therefore, we are interested in assessing the relative performance of the exact solution methods. To do this, we are using two different setups of the B&P algorithm in the tests: one where the subproblem is solved by using the Xpress MIP solver on the MIP formulation in Section 5.1; and one where the subproblem is solved using the E-LSA of Section 5.2. Table 2 displays the results of experiments on the private cloud cases with 5 and 10 services. The columns labelled *Best sol.* report the objective value, i.e., the number of nodes required in the placement of the best found solution. We can see that we are able to solve all of the ten smallest cases to optimality before we reach the maximum run time. Moreover, E-LSA uses fewer iterations but adds more node patterns to the master problem. Since we add up to 20 node patterns to the master problem every time the subproblem is solved, we should expect to solve the subproblem fewer times. While the solution times of the E-LSA are on par with the solution times of the MIP in the `P5` cases, we can see that the solution times increase dramatically on the `P10` cases. For these cases, the MIP is clearly faster; and, not shown in the table, the differences in solution time continue to increase for the larger cases. Thus, we conclude that we prefer to use the MIP as the exact solution method for the subproblem. Moreover, the results disclose that the LP relaxation of the node pattern formulation is very tight. In all of the

cases displayed in Table 2, the objective value of the optimal solution is equal to the rounded up value of the RMP after solving the root node. When using the E-LSA as the subproblem solver, we are able to find the optimal solution by solving the RMP as an IP after solving the root node. When the MIP is used as the subproblem solver, there are two cases, `P5-a` and `P10-a`, where the IP of the RMP does not give the optimal solution after the root node. Nevertheless, the next time the RMP is solved as an IP, after 100 more B&B nodes, the optimal solutions are found.

Table 2: Comparison of the exact solution methods for the subproblem: MIP vs. Exact label-setting algorithm (E-LSA). The column *np* lists the number of generated node patterns, while the column *iter.* lists the total number of times the subproblem is solved. Solution times are given in seconds.

	MIP						**E-LSA**					
	Best sol.	Best bound	Sol. time	B&B nodes	np	iter.	Best sol.	Best bound	Sol. time	B&B nodes	np	iter.
`P5-a`	14	14	121	101	299	400	14	14	47	1	500	34
`P5-b`	17	17	22	1	133	134	17	17	18	1	434	29
`P5-c`	15	15	30	1	169	170	15	15	29	1	525	35
`P5-d`	14	14	28	1	155	156	14	14	50	1	519	38
`P5-e`	18	18	28	1	146	147	18	18	53	1	507	41
`P10-a`	32	32	329	101	516	617	32	32	4582	1	955	55
`P10-b`	32	32	103	1	352	353	32	32	838	1	823	50
`P10-c`	35	35	134	1	363	364	35	35	2182	1	930	61
`P10-d`	32	32	112	1	363	364	32	32	752	1	854	50
`P10-e`	37	37	106	1	357	358	37	37	922	1	808	45

In the experiments on the larger cases, where many B&B nodes are explored, there are B&B nodes where neither the heuristic nor the exact subproblem solver manages to find new improving node patterns. Consequently, only one iteration with an exact solution method is necessary to solve many B&B nodes. On average in the cases with 20, 30, 40 and 50 services, a new improving node pattern is generated by the subproblem in only 25% of the B&B nodes. In a B&B node that does not produce a single node pattern, using the heuristic subproblem solver, which has to be followed by an exact solution method, is clearly a wasted iteration. Ideally, we would like to know in advance if it is possible to generate an improving node pattern in a B&B node; and if so, we want to use the heuristic subproblem solver; otherwise, we will call the exact subproblem solver directly. However, we do not have this type of information; but we try to make a prediction about the possibility of generating an improving node pattern in a B&B node by comparing the final LP objective value of the node's parent and the current LP objective value of the node itself. In a minimization problem, the objective value of a B&B node t, say z_t, can never be less than the final objective value of the parent node, say z_t^{PAR}, i.e., $z_t \geq z_t^{PAR}$; and when node patterns with negative reduced cost are added to the RMP, z_t decreases monotonically. Thus, if the relative difference between these values is small, we suggest that it is less probable to find an improving node pattern in this B&B node. Specifically, we say that if the relative difference is less than a threshold, denoted by Δ, we call the exact subproblem solver next. This condition is shown in inequality (6.1).

$$\frac{z_t - z_t^{PAR}}{z_t^{PAR}} \leq \Delta \tag{6.1}$$

Table 3 shows some results of the experiments with the B&P with the H-LSA as the primary

subproblem solver, where Δ is given different values between zero and one. In the extreme case where $\Delta = 1$, the B&P will never use the H-LSA in other B&B nodes than the root node, and if $\Delta = 0$, the B&P will use the H-LSA in every B&B node until it cannot find an improving node pattern, and then switch to the exact solution method. Moreover, for each of the instance sizes with 30, 40 and 50 services, the table lists the number of solved cases (out of 5 for each size), the average relative gap[1] between the objective value of the best found solution and the best bound after reaching the run time limit, the average percentage of subproblem iterations conducted with the H-LSA, and the average number of B&B nodes. At the bottom of the table, there are also averages over all cases with 30 services or more. Generally, we can see that when Δ is given a large value, the B&P algorithm explores a larger amount of B&B nodes compared to the cases when Δ is given a small value. This result might be explained by, as already discussed, the high amount of B&B nodes where no improving node patterns can be found, and thus smaller values of Δ imply more wasted calls to the H-LSA. Overall, the results show that using the H-LSA in the B&P tree is valuable, at least if we manage to control when to use the H-LSA and when to call the exact solution method directly. We see that the smallest average gap is achieved when $\Delta = 1 \cdot 10^{-5}$. For this value, we observe that on average the H-LSA is used as the solution method in about 20% of the subproblem iterations, as opposed to about 40% when $\Delta = 0$.

Table 3: H-LSA results with different thresholds, Δ. *Avg. H-LSA iter.* refers to the percentage of calls to the H-LSA subproblem solver out of the total number of times the subproblem is solved. Run time limit: 5 hours.

		Threshold on relative difference, Δ					
		1	$1 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$5 \cdot 10^{-6}$	0
P30	Solved cases	1	0	1	1	1	1
	Avg. gap	1.359%	1.347%	1.158%	0.9559%	1.157%	0.9580%
	Avg. H-LSA iter.	4.468%	5.004%	7.811%	18.17%	23.35%	42.26%
	Avg. B&B nodes	1522	1747	1572	1450	1335	1175
P40	Solved cases	0	0	0	0	0	0
	Avg. gap	3.225%	3.070%	3.216%	2.788%	2.936%	2.935%
	Avg. H-LSA iter.	5.107%	5.845%	9.188%	23.00%	26.46%	40.26%
	Avg. B&B nodes	1535	1507	1500	1230	1204	1099
P50	Solved cases	0	0	0	0	0	0
	Avg. gap	4.013%	3.901%	3.783%	3.554%	3.783%	4.267%
	Avg. H-LSA iter.	7.503%	9.376%	9.579%	18.62%	22.54%	40.02%
	Avg. B&B nodes	1192	1152	1109	955.8	912.0	832.0
P30, P40 & P50	Avg. gap	2.866%	2.772%	2.719%	2.433%	2.625%	2.720%
	Avg. H-LSA iter.	5.692%	6.742%	8.859%	19.93%	24.12%	40.85%
	Avg. B&B nodes	1416	1469	1394	1212	1150	1035

In Gullhav and Nygreen [12], we compared the performance and solution quality of the direct MIP formulation and a heuristic algorithm that pre-generates a subset of all maximal node patterns, gives this as input to the pattern-based models, and subsequently solves the formulations of Section 4 as an IP. The results showed that the pre-generation algorithm outperformed the direct MIP formulation, both in terms of speed and solution quality. Here, we will concentrate on comparing

[1] $\frac{\text{obj. val. of best solution - final best bound}}{\text{final best bound}}$

the B&P algorithm with the pre-generation algorithm. Specifically, we will consider two versions of the B&P algorithm: one which only solves the subproblem with the exact MIP model of Section 5.1; and one which uses the H-LSA as the primary subproblem solver whenever the relative difference between the parent objective and the current objective is less than $1 \cdot 10^{-5}$. If the H-LSA fails to find an improving node pattern, the exact MIP model is used as the solver. We denote these two alternative B&P algorithms for B&P MIP and B&P H-LSA in the following. Regarding the pre-generation algorithm, we have pre-generated node patterns as it was done in our earlier paper, and for each of the cases in the subsequently presented results we have included a subset of between 250 000 and 300 000 maximal node patterns (out of millions) in the optimization runs. For the `P50` cases, this corresponds to about 0.1% of the total number of maximal node patterns. Our experience is that if one includes more node patterns than this, it will not improve the solutions significantly, but instead slow down the MIP solver.

Table 4 compares the average gaps of the pre-generation and the B&P algorithms at different time steps for the cases with 20 or more services. The results of the pre-generation is reported without accounting for the time it takes to pre-generate the node patterns, and the gaps are calculated based on the final best bound of the B&P H-LSA, which in all cases equals the final best bound of the B&P MIP. One can see from the columns of Table 4 labelled *Final*, that after a maximum of five hours of run time, the B&P H-LSA has the smallest average gaps in all cases. Generally, at a given time step the B&P H-LSA has found better solutions than both the pre-generation and the B&P MIP. Also, the B&P MIP outperforms pre-generation in most cases. For the `P50` case we can see that the average gap after 3600 seconds for the B&P MIP is not given. The explanation is that the B&P MIP has not yet solved the root node, and subsequently the master problem as an IP, after 3600 seconds in any of the `P50` cases. Therefore, no integer solutions are found at this point. Not shown in the table, the magnitude of the final gaps of B&P H-LSA varies from 1 for the unsolved `P20` cases to between 4 and 7 for the `P50` cases. In the latter cases, the best bounds are in the interval $[162, 174]$. Furthermore, both B&P MIP and B&P H-LSA solve three of the five `P20` cases, and B&P H-LSA solves one of the five `P30` cases. The pre-generation cannot find the optimal solution in any case.

Table 4: Average relative gap (in %) at different time steps (seconds): comparison of the pre-generation algorithm and the branch and price algorithms. Run time limit: 5 hours.

	Pre-generation					**B&P MIP**					**B&P H-LSA**				
	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final
`P20`	3.895	3.895	3.592	3.002	3.002	1.824	1.553	1.553	1.553	1.241	1.553	0.924	0.620	0.620	0.620
`P30`	8.196	5.423	4.663	4.663	4.663	2.909	2.320	2.320	1.916	1.732	2.522	1.729	1.343	1.139	0.956
`P40`	7.903	6.566	5.552	5.251	4.674	5.098	3.804	3.509	3.070	3.070	3.958	3.081	2.938	2.788	2.788
`P50`	9.119	9.119	7.224	6.258	5.798	N/A	5.677	4.844	4.614	4.493	5.144	4.494	4.137	3.789	3.554

Now, we are concentrating on experiments with the hybrid cloud model of Section 4.2. The B&P H-LSA and pre-generation algorithms are set up as in the private cloud cases. That is, Δ is still $1 \cdot 10^{-5}$, and the pre-generation algorithm optimizes a hybrid cloud case over the same node patterns that were used for the corresponding private cloud case. As discussed in Section 6.1, the specified number of nodes in the hybrid cloud cases are set based on the best bound of the corresponding private cloud cases. We omit to give the best bound for all private cloud cases, but instead give the values for one case of each instance size. The best bounds for the cases `P20-a`, `P30-a`, `P40-a` and `P50-a` are, respectively, 64, 75, 101 and 124, and the values of the other cases

are quite similar to the value of the case of the same size.

Table 5 compares the gaps at different points in time of the pre-generation algorithm and the B&P H-LSA algorithm for the hybrid cloud cases with 90% private cloud coverage. Again, the gaps of the pre-generation algorithm are computed based on the final best bound of the B&P H-LSA algorithm. Our tests show that the B&P H-LSA produces better results than the B&P MIP, but for presentation purposes, we exclude the results of the latter from the tables regarding hybrid cloud cases. However, the table shows that the branch and price algorithm is clearly better than pre-generation for all cases. The gaps of the B&P H-LSA are smaller than those of the pre-generation algorithm. Table 5 also shows that the average gaps are larger compared to the private cloud results in all cases. Moreover, the time to solve the root node is longer for both solution methods, and the B&P H-LSA is able to find a solution within 3600 seconds in only one of the five `H50` cases. On the other hand, the pre-generation algorithm finds a solution within 3600 seconds in four of the five `H50` cases, but the gap in one of these cases is as much as 98.40%. When considering the hybrid cloud cases with 75% private cloud coverage, we obtain similar results. As seen in Table 6, the B&P H-LSA produces solutions with smaller gaps in all cases. However, the relative gaps are smaller than in the cases with 90% private cloud coverage, and so these cases are seemingly easier to solve. This observation is consistent with the results in [12], where we showed that the hybrid cloud model became harder to solve as the number of nodes in the private cloud approached the best found solution of the private cloud model.

Table 5: Average relative gap (in %) at different time steps (seconds): comparison of the pre-generation algorithm and the branch and price algorithm using H-LSA on the hybrid cloud cases with 90% private cloud coverage. Run time limit: 5 hours.

	Pre-generation					**B&P H-LSA**				
	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final
`H20`	28.99	27.20	25.63	23.58	23.41	8.044	6.775	5.146	5.146	5.146
`H30`	41.43	39.13	37.28	36.84	34.97	17.64	14.85	12.08	11.49	11.38
`H40`	107.9	45.14	44.30	43.71	42.45	23.49	18.39	17.64	17.26	16.88
`H50`	62.38	48.44	46.00	43.54	43.54	25.74	22.39	22.33	20.74	18.66

Table 6: Average relative gap (in %) at different time steps (seconds): comparison of the pre-generation algorithm and the branch and price algorithm using H-LSA on the hybrid cloud cases with 75% private cloud coverage. Run time limit: 5 hours.

	Pre-generation					**B&P H-LSA**				
	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final
`H20`	10.89	10.17	9.410	9.180	9.180	2.007	1.769	1.709	1.304	1.196
`H30`	14.34	13.05	12.51	12.47	12.47	4.126	3.933	3.576	3.497	3.497
`H40`	16.74	14.84	14.73	14.67	14.15	7.208	5.847	5.489	5.899	5.092
`H50`	37.53	16.13	15.99	17.24	17.24	N/A	7.641	6.738	5.606	5.454

Even though the B&P H-LSA finds better solutions than the other algorithms in the comparisons in Tables 4 - 6, the B&P H-LSA can only prove optimality in a very few number of cases. In addition, the gaps shown in the tables are relatively large. For the private cloud model, the gaps are in general smaller, but this model has also structural similarities with the cutting stock problem, for which column generation provides very tight bounds [30]. However, for the hybrid cloud model, we cannot be certain about the tightness of the bounds. Therefore, it is hard to

quantify the economic consequences of the magnitude of the bounds. For the private cloud model, a positive gap means that it might be possible to use fewer nodes (and less energy) than in the best found solution, and for the hybrid cloud model, the magnitude of the gap can be translated to the extra cost of the best solution compared to the minimum cost value that cannot be rejected as infeasible.

6.3. Comparison of the Private Cloud and Hybrid Cloud Solutions

The private cloud and hybrid cloud models are in many ways quite similar, but the different objective functions and the upper bound on the number of nodes in the private cloud of the hybrid cloud model give their respective solutions distinct features. In the private cloud model, the focus is on finding efficient ways to pack a node, while in the hybrid cloud model, one also has to decide which active replicas should be placed in the public cloud. At first, one could think that obtaining efficient packings of the nodes still was the primary success factor for obtaining good solutions in the hybrid cloud model, and that the active replicas that did not fit in the selected efficient packings were moved to the public cloud. However, the solutions of the hybrid cloud model show that other factors are important to obtain a good solution.

With the node capacity normalized to 100, Table 1 shows that the cost per unit of resource is equal 1 for all VM types. But since the public cloud VM types are offered in pre-defined discrete sizes which might not fit exactly with the resource requirement of an active replica of (i, q), G_{Aiqg}, the cost per unit of resource might be higher. E.g., if $G_{Aiqg} = 32$, this active replica would fit in the largest VM of provider 1 in Table 1 and cost 40. The cost per unit of resource is then $40/32 = 1.25$, which is considerable higher than 1. In the private cloud model, this unit cost is of no importance for the solution quality, but in the hybrid cloud model it is a meaningful quantity. To compare the solutions of the private cloud cases and the hybrid cloud cases we have computed a *cost-resource-ratio* (CRR) by dividing the total cost of placing a set of active replicas in the public cloud by the sum of resources (G_{Aiqg}) required by the same active replicas. The second column of Table 7 gives the CRR of the best solutions of the private cloud cases, i.e., the CRR is computed by dividing the potential cost of placing all the active replicas in the public cloud by the total amount of resource required by the active replicas. The columns labelled *tot* give the same numbers for the hybrid cloud cases with 90% and 75% private cloud coverage. Note that these numbers are not influenced directly by the placement, but computed based on the replication patterns selected in the solution. The table shows that the CRR computed over all active replicas are often identical. In the cases where they differ, it means that the solutions use different replication patterns. What is more interesting is the CRR computed over all active replicas placed in the public cloud (columns labelled *puc*). These ratios are much smaller than the ratio of the private cloud cases and the ratios over all active replicas in the hybrid cloud cases. This observation is important as it tells us that the optimization process selects the active replicas with relatively low cost per resource unit to be placed in the public cloud. Just for comparison, Table 7 also presents the CRR computed over all active replicas placed in the private cloud (cf. column *prc*) and it shows that these ratios are higher than the ratio of the private cloud cases. This is because the active replicas with low cost per resource unit are placed in the public cloud, and so the relatively "expensive" active replicas remain in the private cloud. Furthermore, if one compares the CRR computed over the active replicas in the public cloud one can see that the CRRs for the cases with 90% private cloud coverage are always smaller than the ones with 75% private cloud coverage. This finding supports the interpretation above as it means that if the private cloud coverage is increased, the optimization

process will move the active replicas in the public cloud with relatively high cost per resource unit back to the private cloud.

Table 7: The cost-resource-ratio (CRR) of the private cloud solution, and the hybrid cloud solutions with 90% and 75% private cloud coverage (pcc). For the latter, the CRR is computed over all active replicas (*tot*), over the active replicas placed in the private cloud (*prc*), and over the active replicas placed in the public cloud (*puc*).

	Private cloud	**Hybrid cloud: 90% pcc**			**Hybrid cloud: 75% pcc**		
		prc	*tot*	*puc*	*prc*	*tot*	*puc*
H20-a	1.2124	1.2338	1.2123	1.0491	1.2753	1.2123	1.0750
H20-b	1.2234	1.2459	1.2234	1.0680	1.2888	1.2234	1.0810
H20-c	1.2218	1.2444	1.2211	1.0355	1.2825	1.2211	1.0811
H20-d	1.1986	1.2177	1.1986	1.0489	1.2589	1.1983	1.0604
H20-e	1.2197	1.2393	1.2202	1.0676	1.2795	1.2202	1.0788
H30-a	1.2098	1.2336	1.2114	1.0516	1.2725	1.2113	1.0816
H30-b	1.2269	1.2448	1.2269	1.0758	1.2920	1.2269	1.0848
H30-c	1.2240	1.2451	1.2239	1.0647	1.2840	1.2237	1.0848
H30-d	1.1876	1.2059	1.1876	1.0531	1.2459	1.1876	1.0595
H30-e	1.2078	1.2293	1.2091	1.0603	1.2720	1.2091	1.0705
H40-a	1.2023	1.2251	1.2021	1.0603	1.2617	1.2020	1.0796
H40-b	1.2166	1.2403	1.2166	1.0709	1.2801	1.2165	1.0862
H40-c	1.2234	1.2487	1.2234	1.0577	1.2878	1.2234	1.0824
H40-d	1.1966	1.2172	1.1966	1.0535	1.2581	1.1961	1.0646
H40-e	1.2112	1.2322	1.2111	1.0651	1.2706	1.2110	1.0822
H50-a	1.2036	1.2272	1.2036	1.0596	1.2676	1.2034	1.0770
H50-b	1.2150	1.2388	1.2150	1.0668	1.2802	1.2150	1.0736
H50-c	1.2230	1.2494	1.2232	1.0571	1.2892	1.2231	1.0823
H50-d	1.1958	1.2173	1.1958	1.0597	1.2575	1.1954	1.0682
H50-e	1.2099	1.2318	1.2099	1.0727	1.2716	1.2098	1.0789
Average	1.2115	1.2334	1.2116	1.0599	1.2738	1.2115	1.0766

Lastly, Table 8 compares the objective values of the best found solutions of the hybrid cloud cases divided by the reduction in the number of nodes compared to the best found solution of the corresponding private cloud case, that is, $z_H/(z_P - N_N)$. We denote these numbers as the *public cloud node cost*. By regarding the costs of the public cloud VM types in Table 1, the minimum cost of moving all active replicas of a fully utilized node to the public cloud is 100. Since the CRR values of the active replicas in the public cloud are considerably higher than 1 (cf. Table 7), one should expect a higher cost than 100. However, the public cloud node costs presented in Table 8 take (average) values both above and below 100. The latter can be explained by the fact that in many cases the average node utilization in the private cloud cases is significantly less than 100%. Hence, when $z_P - N_N$ nodes are removed from the solution of a private cloud case, one does not have to compensate by leasing resources from a public cloud provider equivalent to $z_P - N_N$ fully utilized nodes. Furthermore, Table 8 also illustrates that the public cloud node cost is consistently less when the private cloud coverage is 90% compared to 75%, which is natural because the average utilization of the N least utilized nodes is reduced as N_N is increased, and thus fewer resources per node have to be compensated for by using the public cloud. Another feature of the numbers in the table is that the public cloud node costs seem to decrease with the size of the test cases, with the H30 cases being exceptions. This can be explained by the fact that the gaps between the best found solution and the best bound in the private cloud cases increase with the problem size, and

as a consequence, the average node utilization is lower in the larger cases.

Table 8: Public cloud node cost. Comparison between the best solutions of the hybrid cloud cases with 90% and 75% private cloud coverage (pcc).

	90% pcc	**75% pcc**
H20	96.24	104.1
H30	98.15	104.7
H40	90.73	100.1
H50	87.75	98.12

7. Conclusions

We have studied two versions of the service deployment problem proposed in [12]: one for periods where the private cloud of the service provider has enough resource capacity to run all services, and one for periods where a public cloud is used for placement as well, forming a hybrid cloud. A B&P algorithm was proposed to solve the problems. The subproblem of the B&P was solved by a MIP solver and a label-setting algorithm, which was run on a network designed to exploit the special problem structure. Firstly, we developed an exact label-setting algorithm, but due to a weak dominance rule, the run time of the algorithm did not scale well to larger problems. One contribution of this paper is a heuristic label-setting algorithm (H-LSA), which heuristically reduces the underlying network and simplifies the dominance rule based on the problem structure. The H-LSA was complemented with the exact MIP solver to obtain an exact and complete B&P algorithm. Our experiments showed that using the H-LSA sped up the solution process. However, we also observed that in many B&B nodes, no node patterns with negative reduced cost did exist, and calling the H-LSA in such a case results in a wasted iteration. Another contribution of this paper is a novel strategy to predict the likeliness of finding node patterns with negative reduced cost in a B&B node, and in nodes where this seems unlikely, we directly call the MIP solver.

We also compared the B&P algorithm with the pre-generation algorithm presented in [12], and the results showed that the B&P outperformed the other algorithm. Moreover, the B&P managed to solve all test cases with 10 services or less, but as the problem size grew the relative gap between the objective value of the best-found solution and the best bound increased when a maximum run time was set. For the cases with 50 services, the gap was over five percent on average after five hours of run time. The results of the experiments on the hybrid cloud model indicate that this model is more difficult to solve than the private cloud model; the relative gaps were larger and it took longer time to solve the root node. We also obtained results showing that the hybrid cloud model becomes harder to solve when the private cloud coverage approaches 100%. Since the hybrid cloud case with equally many services and service components as a private cloud case, but with fewer nodes, are more difficult to solve than the private cloud case, this must mean that it is not necessarily easy to select the active replicas for placement in the public cloud.

By analyzing the solutions of the hybrid cloud model, we found as expected that the active replicas placed in the public cloud fit better in the VM types offered in the public cloud, i.e., they have less cost per unit of resource required, than the active replicas placed in the private cloud. This is a critical factor in a good solution.

Acknowledgments

We want to thank the anonymous reviewers for their constructive comments and suggestions, which helped us to improve the quality of this paper.

Appendix A. SPPRC Details

Appendix A.1. Labels and Label Extension

Based on the network description in Section 5.2, we let $\mathcal{V}_A$ ($\mathcal{V}_P$) be the set of all a-vertices (p-vertices) in all service subnets; and $\mathcal{V}_\sigma$ and $\mathcal{V}_\tau$ be the sets of all σ-vertices and τ-vertices, respectively. Note that σ_0 and τ_0 of the upper layer network (Figure 2) are not contained in the latter sets. Moreover, since a visit to an a-vertex or p-vertex is analogous to a placement of an active or passive replica, we have to assign the dual variables, which are used in the mathematical formulation of the subproblem in Section 5.1, to the respective vertices. Therefore, a vertex $\nu \in \mathcal{V}_A$ is assigned a dual variable α_ν corresponding to the dual variable α_{iq} of the service-component pair (i, q) that the vertex represents. Likewise, each vertex $\nu \in \mathcal{V}_P$ is assigned a dual variable β_ν corresponding to the dual variable β_{iq}. Lastly, we also keep track of the resource usage of a node pattern by assigning each $\nu \in \mathcal{V}_A$ resource consumption values $G_{A\nu g}$ for all resource types $g \in \mathcal{G}$, resembling the resource consumption G_{Aiqg} of (i, q) which ν represents. Similarly, each $\nu \in \mathcal{V}_P$ is assigned the resource consumption values $G_{P\nu g}$ for all $g \in \mathcal{G}$.

To conduct the label-setting algorithm, each label stores data according to Table A.1. We use the notation $\psi(\ell)$ to refer to the vertex of label ν, and similarly $\lambda(\ell)$, $\zeta(\ell)$, $f_g(\ell)$, $m_g(\ell)$, $\pi(\ell)$ and $\xi(\ell)$ to refer to the rest of the data listed in the table.

Table A.1: The data stored for each label

Symbol	Description
ψ	pointer to the vertex of the label
λ	pointer the predecessor label
ζ	the reduced cost of the label
m_g	the amount of resource of type $g \in \mathcal{G}$ reserved for activation of passive replicas
f_g	the accumulated resource of type $g \in \mathcal{G}$ (i.e. CPU, memory, etc.), including m_g
π	the accumulated number of passive replicas
ξ	the accumulated number of different services

The path represented by a label ℓ is feasible if all of the following conditions hold:

$$f_g \leq N_{Cg}, \quad \forall g \in \mathcal{G} \tag{A.1}$$

$$\pi \leq N_P \tag{A.2}$$

$$\xi \leq N_S \tag{A.3}$$

These conditions are equivalent to the constraints (5.6), (5.4) and (5.3), respectively.

When a label ℓ is extended to a vertex ν to form a new label k, the data stored for the new label is constructed as shown below. (A.6) - (A.10) are denoted resource extension functions.

$$\psi(k) = \nu \tag{A.4}$$

$$\lambda(k) = \ell \tag{A.5}$$

$$\zeta(k) = \begin{cases} \zeta(\ell) - \alpha_\nu & \text{if } \nu \in \mathcal{V}_A \\ \zeta(\ell) - \beta_\nu & \text{if } \nu \in \mathcal{V}_P \\ \zeta(\ell) & \text{otherwise} \end{cases} \tag{A.6}$$

$$m_g(k) = \begin{cases} m_g(\ell) + \max\{G_{A\nu g} - G_{P\nu g} - m_g(\ell), 0\} & \text{if } \nu \in \mathcal{V}_P \\ m_g(\ell) & \text{otherwise} \end{cases} \quad \forall g \in \mathcal{G} \tag{A.7}$$

$$f_g(k) = \begin{cases} f_g(\ell) + G_{A\nu g} & \text{if } \nu \in \mathcal{V}_A \\ f_g(\ell) + G_{P\nu g} + \max\{G_{A\nu g} - G_{P\nu g} - m_g(\ell), 0\} & \text{if } \nu \in \mathcal{V}_P \\ f_g(\ell) & \text{otherwise} \end{cases} \quad \forall g \in \mathcal{G} \tag{A.8}$$

$$\pi(k) = \begin{cases} \pi(\ell) + 1 & \text{if } \nu \in \mathcal{V}_P \\ \pi(\ell) & \text{otherwise} \end{cases} \tag{A.9}$$

$$\xi(k) = \begin{cases} \xi(\ell) + 1 & \text{if } \nu \in \mathcal{V}_\sigma \\ \xi(\ell) & \text{otherwise} \end{cases} \tag{A.10}$$

Functions (A.4) and (A.5) set the vertex and predecessor, respectively, and (A.6) computes and stores the accumulated reduced cost, based on to which vertex set ν belongs. Note that the label constructed in the initialization in Algorithm 2 has an accumulated reduced cost of 1, corresponding to the constant term in the objective function (5.1) in the MIP formulation ($-\eta$ in the hybrid cloud model). Furthermore, (A.7) computes the amount of resources reserved for activation of passive replicas. If the vertex ν represents a passive replica and $G_{A\nu} - G_{P\nu} > m_g(\ell)$, $m_g(k)$ is increased from $m_g(\ell)$, otherwise $m_g(k)$ remains at the level of $m_g(\ell)$. (A.8) computes the accumulated resources. If vertex ν represents an active replica, the extension comprises adding the resource usage of ν to the accumulated resources of ℓ. On the other hand, if ν represents a passive replica, one also have to account from a possible increase in the resources reserved for activation of passive replicas, likewise done in (A.7). If ν neither represents an active nor passive replica, $f_g(k)$ takes the same value as $f_g(\ell)$. Lastly, (A.9) and (A.10) increment the number of passive replicas and different services if ν represents a passive replica or is an entrance of a service block, respectively.

Appendix A.2. Dominance Criteria

The goal of implementing the dominance step of Algorithm 2 is to reduce the number of extended labels, and thereby speed up the solution procedure. It should be noted that without the dominance step, Algorithm 2 would have been a procedure enumerating all feasible paths.

In order to describe our dominance criteria, we define the set of path extensions for ℓ, $\mathcal{E}(\ell)$, as the partial paths, starting in vertex $\psi(\ell)$ and ending in the sink vertex τ_0, that can be concatenated with the partial path represented by ℓ, and result in a resource-feasible path from σ_0 to τ_0. Then, domination is defined as follows; label ℓ_1 *dominates* label ℓ_2 if:

$$\psi(\ell_1) = \psi(\ell_2) \tag{A.11}$$

$$\zeta(\ell_1) \leq \zeta(\ell_2) \tag{A.12}$$

$$\mathcal{E}(\ell_1) \supseteq \mathcal{E}(\ell_2) \tag{A.13}$$

If conditions (A.11) - (A.13) hold, it is certain that any path being constructed by concatenating the partial path represented by ℓ_1 and any partial path $e \in \mathcal{E}(\ell_2)$ cannot have a worse reduced cost than the path constructed from the partial path represented by ℓ_2 and the same e. Therefore, ℓ_1 dominates ℓ_2, and the label ℓ_2 can be removed from the set of labels at vertex $\psi(\ell_2)$, i.e. $\mathcal{L}_{\psi(\ell_2)}$.

Unfortunately, in most cases, ours including, it is not practical to check condition (A.13) directly. Nevertheless, since the resource extension functions (A.7) - (A.10) of the SPPRC are non-decreasing, the conditions (A.14) - (A.17) below imply condition (A.13) [15].

$$m_g(\ell_1) \geq m_g(\ell_2) \quad \forall g \in \mathcal{G} \tag{A.14}$$

$$f_g(\ell_1) \leq f_g(\ell_2) \quad \forall g \in \mathcal{G} \tag{A.15}$$

$$\pi(\ell_1) \leq \pi(\ell_2) \tag{A.16}$$

$$\xi(\ell_1) \leq \xi(\ell_2) \tag{A.17}$$

The domination is performed in line 4 of Algorithm 2, and it consists of checking if labels dominate each other, and if so remove the dominated label. If the domination is implemented naïvely, one has to compare every label with all other labels, in total $|\mathcal{L}_\nu|^2$ comparisons at vertex ν. However, if one maintain the set of labels at a node ordered according to the reduced cost ζ, one only has to compare every label with other labels with worse (i.e., higher) reduced cost (cf. domination criterion (A.12)).

In line 3 of Algorithm 2, we have the option to decide whether the domination step should be conducted in a given vertex or not. We have conducted the domination step in every vertex, except the dummy source and sink, σ_0 and τ_0; and at the source vertex σ_1 of the first service.

Appendix A.3. Implementing the Branching in the SPPRC

The basic idea on how to deal with the branching decisions is to keep track of which branching sets a label has visited, and if visited for a second time, i.e., both replicas corresponding to a branching set are included in the label, we have to correct the reduced cost of the label with the dual variable of the branching set. First we define $\mathcal{V}_{Cj} \subset (\mathcal{V}_A \cup \mathcal{V}_P)$ for each $j \in \bar{\mathcal{J}}_t \cup \underline{\mathcal{J}}_t$ as the set of vertices representing the service component pairs (i, q) with $H_{Ajiq} = 1$ or $H_{Pjiq} = 1$. Note that the cardinality of $\mathcal{V}_{Cj}$ is always two. In addition to the data in Table A.1, every label ℓ has to keep track of the number of times the vertices related to branching set $j \in \bar{\mathcal{J}}_t$ ($j \in \underline{\mathcal{J}}_t$) is visited. We denote these counters as $\bar{\theta}_j$ ($\underline{\theta}_j$). When extending a label ℓ to label k at vertex ν, these counters are updated according to (A.18)-(A.19). At every extension, we also need to monitor if the label visits a branching set for the second time, and if so subtract the corresponding dual variable from the accumulated reduced cost. To do this, we build the sets $\bar{\mathcal{F}}$ and $\underline{\mathcal{F}}$ as the set of branching sets that are visited for the second time at the current vertex, ν. After updating $\bar{\theta}_j$ and $\underline{\theta}_j$, the sets are built as shown in (A.20) and (A.21) for each label k. By using these sets, the resource extension function for updating the reduced costs, (A.6), are updated to (A.22).

$$\bar{\theta}_j(k) = \begin{cases} \bar{\theta}_j(\ell) + 1 & \text{if } \nu \in \mathcal{V}_{Cj} \\ \bar{\theta}_j(\ell) & \text{otherwise} \end{cases} \quad \forall j \in \bar{\mathcal{J}}_t \tag{A.18}$$

$$\underline{\theta}_j(k) = \begin{cases} \underline{\theta}_j(\ell) + 1 & \text{if } \nu \in \mathcal{V}_{Cj} \\ \underline{\theta}_j(\ell) & \text{otherwise} \end{cases} \quad \forall j \in \underline{\mathcal{J}}_t \tag{A.19}$$

$$\bar{\mathcal{F}}(k) = \{j \in \bar{\mathcal{J}}_t \mid \nu \in \mathcal{V}_{Cj} \wedge \bar{\theta}_j(k) = 2\} \tag{A.20}$$

$$\underline{\mathcal{F}}(k) = \{j \in \underline{\mathcal{J}}_t \mid \nu \in \mathcal{V}_{Cj} \wedge \underline{\theta}_j(k) = 2\} \tag{A.21}$$

$$\zeta(k) = \begin{cases} \zeta(\ell) - \alpha_\nu - \sum_{j \in \bar{\mathcal{F}}(k)} \bar{\gamma}_j - \sum_{j \in \underline{\mathcal{F}}(k)} \underline{\gamma}_j & \text{if } \nu \in \mathcal{V}_A \\ \zeta(\ell) - \beta_\nu - \sum_{j \in \bar{\mathcal{F}}(k)} \bar{\gamma}_j - \sum_{j \in \underline{\mathcal{F}}(k)} \underline{\gamma}_j & \text{if } \nu \in \mathcal{V}_P \\ \zeta(\ell) & \text{otherwise} \end{cases} \tag{A.22}$$

The introduction of branching constraints in the master problem also influences how the domination can be conducted. When comparing the reduced costs of two labels to check if ℓ_1 dominates ℓ_2, one can have that label ℓ_1 has visited one vertex in a branching set $\bar{\jmath}$, while ℓ_2 has not. If $\bar{\jmath}$ is an element in $\bar{\mathcal{J}}_t$, another visit by an extension of ℓ_1 to this branching set will incur a *penalty* of $\bar{\gamma}_{\bar{\jmath}}$, while an extension of ℓ_2 to the same vertex will not incur a penalty. Moreover, if ℓ_2 has visited one vertex, not visited by ℓ_1, in a branching set $\underline{j} \in \underline{\mathcal{J}}_t$, the extension of label ℓ_2 will receive a *bonus* of $\underline{\gamma}_{\underline{j}}$ if the partial path, represented by ℓ_2, is extended to the second vertex of the branching set, while an extension of ℓ_1 to this vertex will not get this bonus. A way to overcome this issue is to compensate the reduced cost of ℓ_1 by the potential penalties which can be observed by an extension of ℓ_1, but not by an extension of ℓ_2; and compensate the reduced cost of ℓ_2 by the potential bonuses which can be observed by an extension of ℓ_2, but not by an extension of ℓ_1. Jepsen et al. [18] uses this idea in order to account for the dual variables stemming from subset-row (SR) inequalities in the master problem of a vehicle-routing problem. Since the SR inequalities are $\leq$-type of constraints, they only have to compensate the reduced cost with potential penalties in their dominance criteria. Now we define $\bar{\mathcal{P}}$ (and $\underline{\mathcal{P}}$) as the set of branching sets $j \in \bar{\mathcal{J}}_t$ (and $\underline{\mathcal{J}}_t$) that are already visited once and has the potential to be visited once more. After extension from label ℓ to label k at vertex ν, the sets are built according to (A.23) and (A.24), where $\omega \succ \nu$ reads as ω succeeds ν in the topological order of the acyclic network, i.e., ω can potentially be visited after the current vertex ν. Using these definitions, we rewrite the dominance criterion (A.12) to (A.25). That is, label ℓ_1 dominates label ℓ_2 if (A.11), (A.14)-(A.17) and (A.25) holds.

$$\bar{\mathcal{P}}(k) = \{j \in \bar{\mathcal{J}}_t \mid \bar{\theta}_j(k) = 1 \wedge \exists \omega \in \mathcal{V}_{Cj}\ (\omega \succ \nu)\} \tag{A.23}$$

$$\underline{\mathcal{P}}(k) = \{j \in \underline{\mathcal{J}}_t \mid \underline{\theta}_j(k) = 1 \wedge \exists \omega \in \mathcal{V}_{Cj}\ (\omega \succ \nu)\} \tag{A.24}$$

$$\zeta(\ell_1) - \sum_{j \in (\bar{\mathcal{P}}(\ell_1) \setminus \bar{\mathcal{P}}(\ell_2))} \bar{\gamma}_j \quad \leq \quad \zeta(\ell_2) - \sum_{j \in (\underline{\mathcal{P}}(\ell_2) \setminus \underline{\mathcal{P}}(\ell_1))} \underline{\gamma}_j \tag{A.25}$$

Appendix A.4. Details of the Heuristic Label-setting Algorithm

The idea of the heuristic label-setting algorithm is to fix the m_g's in advance of calling Algorithm 2. We consider two cases: $m_g = 0$ for all $g \in \mathcal{G}$, in which no passive replicas can be placed in the current node pattern; and $m_g > 0$ for at least one resource type g, which means that passive replicas with $G_{Aiqg} - G_{Piqg} > m_g$ cannot be placed in the node pattern. It is very easy to adapt the SPPRC formulation and the label-setting algorithm to the former case. All vertices $\nu \in \mathcal{V}_P$, i.e., vertices representing passive replicas, together with all arcs connected to ν are removed from the SPPRC graph, and the dominance criteria (A.14) and (A.16) are disregarded. Otherwise, the algorithm works as before. Certainly, this network reduction and simplification of the dominance rule will have considerable impact on the run time of one pass of the algorithm. In the latter case, with non-zero m_g, we choose to pre-select $|\mathcal{G}|$ passive replicas for placement, assuming that $|\mathcal{G}| \leq N_P$ holds in the following. Each of these replicas will function as a benchmark for the m_g

(for different resources g). That is, $m_g = (G_{A\hat{\imath}_g\hat{q}_g g} - G_{P\hat{\imath}_g\hat{q}_g g})$ for all $g \in \mathcal{G}$ where the service components $(\hat{\imath}_1, \hat{q}_1), \ldots, (\hat{\imath}_g, \hat{q}_g), \ldots, (\hat{\imath}_{|\mathcal{G}|}, \hat{q}_{|\mathcal{G}|})$ are the pre-selected passive replicas. To implement this pre-selection and fixing in the SPPRC formulation and label-setting algorithm, more changes has to be done, but the changes are still simple. Firstly, we remove the vertices representing the active replicas of service components $(\hat{\imath}_g, \hat{q}_g)$ for all $g \in \mathcal{G}$ in addition to the connected arcs. Furthermore, all vertices $\nu \in \mathcal{V}_P$ with $(G_{A\nu g} - G_{P\nu g}) > m_g$ for at least one g, and the arcs connected to these vertices, are removed. Then, the order of the services is slightly changed to facilitate the forced placement of the pre-selected passive replicas. Firstly, the service blocks of the pre-selected replicas, denoted *pre-selection blocks*, are ordered as the first service blocks (cf. Figure 2), and the arcs from σ_0 to all service blocks except the first are removed. From all the pre-selection blocks, except the last one in the ordering, there are only arcs from the τ-vertex to the σ-vertex of the next pre-selection block. From the last pre-selection block, there are arcs to all forward service blocks and the τ_0 vertex. Moreover, the subnet of the pre-selection blocks (cf. Figure 3) is rearranged by letting the $p_{\hat{\imath}_g\hat{q}_g}$ vertices be ordered as the first. All arcs out of the σ-vertices are removed except the one arc to the first of possibly $|\mathcal{G}|$ pre-selected passive replicas of the service. In case there are several pre-selected replicas in a subnet, each pre-selected replica is linked by an arc, while the last pre-selected replica has arcs to every other a-vertex and p-vertex of higher order. Lastly, the dominance rule will disregard criterion (A.14).

A fundamental observation about good solutions of the problem is that node patterns typically contain either zero or N_P passive replicas, and in the latter case, the passive replicas have quite similar resource requirements for activation $(G_{Aiqg} - G_{Piqg})$. We use this observation when deciding the pre-selected passive replicas, that is, we try to pick $|\mathcal{G}|$ passive replicas which together with $N_P - |\mathcal{G}|$ other passive replicas make up a set of passive replicas with large dual variables compared to their total resource usage if placed together on a node. For presentation purposes, we have omitted the details of the procedure used to pre-select passive replicas.

References

[1] Amazon Web Services, 2015. Amazon Web Services (AWS) - Cloud Computing Services. Last visited 2015/03/04. URL `http://aws.amazon.com/`

[2] Ardagna, D., Panicucci, B., Trubian, M., Zhang, L., 2012. Energy-aware autonomic resource allocation in multitier virtualized environments. IEEE Transactions on Services Computing 5 (1), 2–19.

[3] Ashrafi, N., Berman, O., Cutler, M., 1994. Optimal design of large software-systems using n-version programming. IEEE Transactions on Reliability 43 (2), 344–350.

[4] Beloglazov, A., Buyya, R., Lee, Y. C., Zomaya, A. Y., 2011. A taxonomy and survey of energy-efficient data centers and cloud computing systems. Advances in Computers 82 (2), 47–111.

[5] Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E., Moatti, Y., Lorenz, D., 2011. Guaranteeing high availability goals for virtual machine placement. In: 2011 31st International Conference on Distributed Computing Systems. pp. 700–709.

[6] Breitgand, D., Epstein, A., 2011. SLA-aware placement of multi-virtual machine elastic services in compute clouds. In: Agoulmine, N., Bartolini, C., Pfeifer, T., O'Sullivan, D. (Eds.), Integrated Network Management. IEEE, pp. 161–168.

[7] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., Warfield, A., 2005. Live migration of virtual machines. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2. NSDI'05. USENIX, Berkeley, CA, USA, pp. 273–286.

[8] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., Warfield, A., 2008. Remus: High availability via asynchronous virtual machine replication. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. USENIX, Berkeley, CA, USA, pp. 161–174.

[9] Distler, T., Kapitza, R., Popov, I., Reiser, H. P., Schröder-Preikschat, W., 2011. SPARE: Replicas on hold. In:

Proceedings of the 18th Network and Distributed System Security Symposium. The Internet Society, Geneva, Switzerland.

[10] Ferreto, T. C., Netto, M. A., Calheiros, R. N., Rose, C. A. D., 2011. Server consolidation with migration control for virtualized data centers. Future Generation Computer Systems 27 (8), 1027 – 1034.

[11] Goudarzi, H., Pedram, M., 2011. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In: 2011 IEEE 4th International Conference on Cloud Computing. IEEE Computer Society, Los Alamitos, CA, USA, pp. 324–331.

[12] Gullhav, A. N., Nygreen, B., 2015. Deployment of replicated multi–tier services in cloud data centres. International Journal of Cloud Computing 4 (2), 130–149.

[13] Gullhav, A. N., Nygreen, B., Heegaard, P. E., 2013. Approximating the response time distribution of fault-tolerant multi-tier cloud services. In: 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing. IEEE Computer Society, Los Alamitos, CA, USA, pp. 287–291.

[14] Gunnerud, V., Foss, B. A., McKinnon, K. I. M., Nygreen, B., 2012. Oil production optimization solved by piecewise linearization in a branch & price framework. Computers & Operations Research 39 (11), 2469 – 2477.

[15] Irnich, S., Desaulniers, G., 2005. Shortest path problems with resource constraints. In: Desaulniers, G., Desrosiers, J., Solomon, M. M. (Eds.), Column Generation. Springer, New York, USA, pp. 33 – 65.

[16] Iyoob, I., Zarifoglu, E., Dieker, A. B., 2013. Cloud computing operations research. Service Science 5 (2), 88–101.

[17] Jennings, B., Stadler, R., 2015. Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management 23 (3), 567–619.

[18] Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56 (2), 497–511.

[19] Kramer, H. H., Petrucci, V., Subramanian, A., Uchoa, E., 2012. A column generation approach for power-aware optimization of virtualized heterogeneous server clusters. Computers & Industrial Engineering 63 (3), 652 – 662.

[20] Kuo, W., Wan, R., 2007. Recent advances in optimal reliability allocation. In: Levitin, G. (Ed.), Computational Intelligence in Reliability Engineering. Vol. 39 of Studies in Computational Intelligence. Springer Berlin Heidelberg, pp. 1–36.

[21] Meedeniya, I., Buhnova, B., Aleti, A., Grunske, L., 2010. Architecture-driven reliability and energy optimization for complex embedded systems. In: Heineman, G. T., Kofron, J., Plasil, F. (Eds.), Research into Practice – Reality and Gaps. Vol. 6093 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 52–67.

[22] Mell, P., Grance, T., 2011. The NIST definition of cloud computing. NIST SP 800-145.

[23] Menasce, D., 2002. QoS issues in web services. Internet Computing, IEEE 6 (6), 72–75.

[24] Petrucci, V., Loques, O., Mossé, D., 2010. A dynamic optimization model for power and performance management of virtualized clusters. In: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. ACM, New York, NY, USA, pp. 225–233.

[25] ROADEF, 2012. ROADEF/EURO challenge 2012: Machine reassignment. Last visited 2015/10/01.
URL `http://challenge.roadef.org/2012/en/`

[26] Shi, L., Butler, B., Botvich, D., Jennings, B., 2013. Provisioning of requests for virtual machine sets with placement constraints in iaas clouds. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management. IEEE, pp. 499–505.

[27] Speitkamp, B., Bichler, M., 2010. A mathematical programming approach for server consolidation problems in virtualized data centers. IEEE Transactions on Services Computing 3 (4), 266–278.

[28] Undheim, A., Chilwan, A., Heegaard, P. E., 2011. Differentiated availability in cloud computing slas. In: 2011 12th IEEE/ACM International Conference on Grid Computing. IEEE Computer Society, Los Alamitos, CA, USA, pp. 129–136.

[29] Vance, P. H., 1998. Branch-and-price algorithms for the one-dimensional cutting stock problem. Computational Optimization and Applications 9, 211–228.

[30] Vanderbeck, F., 1999. Computational study of a column generation algorithm for bin packing and cutting stock problems. Mathematical Programming 86 (3), 565–594.

[31] Vanderbeck, F., 2000. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Operations Research 48 (1), 111–128.

[32] Vanderbeck, F., 2005. Implementing mixed integer column generation. In: Desaulniers, G., Desrosiers, J., Solomon, M. (Eds.), Column Generation. Springer US, pp. 331–358.

[33] Vanderbeck, F., Wolsey, L. A., 1996. An exact algorithm for ip column generation. Operations Research Letters 19 (4), 151 – 159.