A Desktop Grid Computing Approach for Scientific Computing and Visualization

by

Nicolae-Zoran Constantinescu-Fülöp

Doctoral Thesis

Submitted for the Partial Fulfillment of the Requirements for the Degree of

doctor scientiarium

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

May 2008

ISBN 978-82-471-9158-3 (printed version)
ISBN 978-82-471-9161-3 (electronic version)
ISSN 1503-8181
Thesis at NTNU 2008:153
Printed in Norway by NTNU-trykk, Trondheim

Abstract

Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer, mathematical models of problems from science and engineering, and its main goal is to gain insight in such problems. Generally, it is difficult to understand or communicate information from complex or large datasets generated by Scientific Computing methods and techniques (computational simulations, complex experiments, observational instruments etc.). Therefore, support of Scientific Visualization is needed, to provide the techniques, algorithms, and software tools needed to extract and display appropriately important information from numerical data.

Usually, complex computational and visualization algorithms require large amounts of computational power. The computing power of a single desktop computer is insufficient for running such complex algorithms, and, traditionally, large parallel supercomputers or dedicated clusters were used for this job. However, very high initial investments and maintenance costs limit the availability of such systems. A more convenient solution, which is becoming more and more popular, is based on the use of non-dedicated desktop PCs in a Desktop Grid Computing environment. Harnessing idle CPU cycles, storage space and other resources of networked computers to work together on a particularly computational intensive application does this. Increasing power and communication bandwidth of desktop computers provides for this solution.

In a desktop grid system, the execution of an application is orchestrated by a central scheduler node, which distributes the tasks amongst the worker nodes and awaits workers' results. An application only finishes when all tasks have been completed. The attractiveness of exploiting desktop grids is further reinforced by the fact that costs are highly distributed: every volunteer supports her resources (hardware, power costs and internet connections) while the benefited entity provides management infrastructures, namely network bandwidth, servers and management services, receiving in exchange a massive and otherwise unaffordable computing power. The usefulness of desktop grid computing is not limited to major high throughput public computing projects. Many institutions, ranging from academics to enterprises, hold vast number of desktop machines and could benefit from exploiting the idle cycles of their local machines.

In the work presented in this thesis, the central idea has been to provide a desktop grid computing framework and to prove its viability by testing it in some Scientific Computing and Visualization experiments. We present here QADPZ, an open source system for desktop grid computing that have been developed to meet the above presented needs. QADPZ enables users from a local network or Internet to share their resources. It is a multi-platform, heterogeneous system, where different computing resources from inside an organization can be used. It can be used also for volunteer computing,

where the communication infrastructure is the Internet. QADPZ supports the following native operating systems: Linux, Windows, MacOS and Unix variants. The reason behind natively supporting multiple operating systems, and not only one (Unix or Windows, as other systems do), is that often, in real life, this kind of limitation restricts very much the usability of desktop grid computing.

QADPZ provides a flexible object-oriented software framework that makes it easy for programmers to write various applications, and for researchers to address issues such as adaptive parallelism, fault-tolerance, and scalability. The framework supports also the execution of legacy applications, which for different reasons could not be rewritten, and that makes it suitable for other domains as business. It also supports low-level programming languages as C/C++ or high-level language applications, (e.g. Lisp, Python, and Java), and provides the necessary mechanisms to use such applications in a computation. Consequently, users with various backgrounds can benefit from using QADPZ. The flexible object-oriented structure and the modularity allow facile improvements and further extensions to other programming languages.

We have developed a general-purpose runtime and an API to support new kinds of high performance computing applications, and therefore to benefit from the advantages offered by desktop grid computing. This API directly supports the C/C++ programming language. We have shown how distributed computing extends beyond the master-worker paradigm (typical for such systems) and provided QADPZ with an extended API that supports in addition lightweight tasks and parallel computing (using the message passing paradigm - MPI). This extends the range of applications that can be used to already existing MPI based applications - e.g. parallel numerical solvers used in computational science, or parallel visualization algorithms.

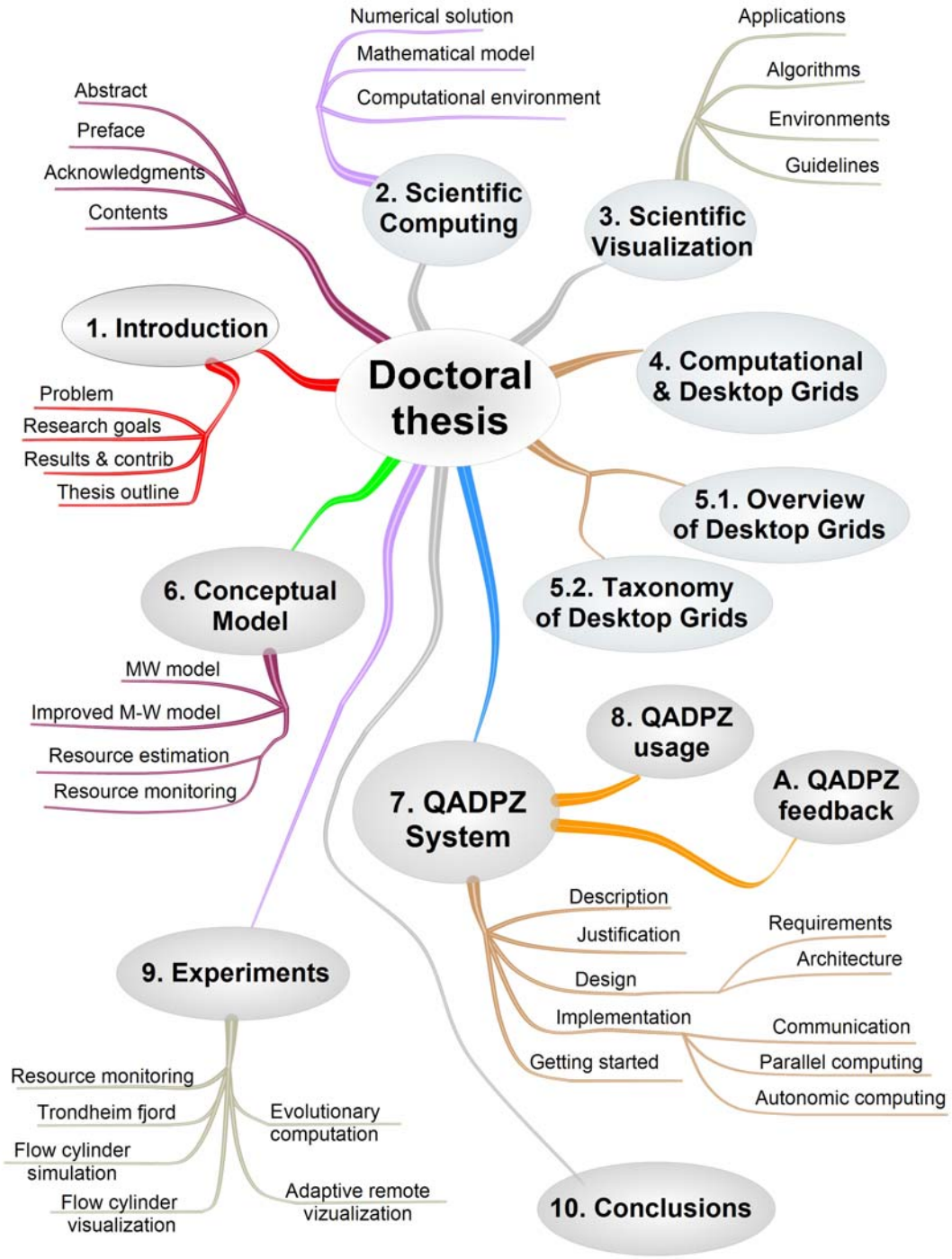
Another restriction of existing systems, especially middleware based, is that each resource provider needs to install a runtime module with administrator privileges. This poses some issues regarding data integrity and accessibility on providers' computers. The QADPZ system tries to overcome this by allowing the middleware module to run as a non-privileged user, even with restricted access, to the local system.

QADPZ provides also low-level optimizations, such as on-the-fly compression and encryption for communication. The user can choose from different algorithms, depending on the application, improving both the communication overhead imposed by large data transfers and keeping privacy of the data. The system goes further, by providing an experimental, adaptive compression algorithm, which can transparently choose different algorithms to improve the application. QADPZ support two different protocols (UDP and TCP/IP) in order to improve the efficiency of communication.

Free source code allows its flexible installations and modifications based on the particular needs of research projects and institutions. In addition to being a very powerful tool for computationally-intensive research, the open-

sourceness makes QADPZ a flexible educational platform for numerous small-size student projects in the areas of operating systems, distributed systems, mobile agents, parallel algorithms, etc. Open source software is a natural choice for modern research as well, because it encourages effectively integration, cooperation and boosting of new ideas.

This thesis proposes also an improved conceptual model (based on the master-worker paradigm), which makes contributions in several directions: pull vs. push work-units, pipelining of work-units, more work-units sent at a time, adaptive number of workers, adaptive time-out interval for work-units, and multithreading. We have also demonstrated that the use of desktop grids should not be limited to only master-worker applications, but it can be used for more fine-grained parallel Scientific Computing and Visualization applications, by performing some specific experiments. This thesis makes supplementary contributions: a hierarchical taxonomy of the main existing desktop grids, and an adaptive compression algorithm for remote visualization. QADPZ has also pioneered autonomic computing approach for desktop grids and presents specific self-management features: self-knowledge, self-configuration, self-optimization and self-healing. It is worth to mention that to the present the QADPZ has over a thousand users who have download it (since July, 2001 when it has been uploaded to sourceforge.net), and many of them use it for their daily tasks (see the appendix). Many of the results have been published or are in course of publishing as it can be seen from the references.


Doctoral thesis
1. Introduction
Abstract
Preface
Acknowledgments
Contents
Problem
Research goals
Results & contrib
Thesis outline
2. Scientific Computing
Numerical solution
Mathematical model
Computational environment
3. Scientific Visualization
Applications
Algorithms
Environments
Guidelines
4. Computational & Desktop Grids
5.1. Overview of Desktop Grids
5.2. Taxonomy of Desktop Grids
6. Conceptual Model
MW model
Improved M-W model
Resource estimation
Resource monitoring
7. QADPZ System
8. QADPZ usage
A. QADPZ feedback
Description
Justification
Design
Requirements
Architecture
Implementation
Communication
Getting started
Parallel computing
Autonomic computing
9. Experiments
Resource monitoring
Trondheim fjord
Flow cylinder simulation
Flow cylinder visualization
Evolutionary computation
Adaptive remote vizualization
10. Conclusions

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of doctor scientiarium.

This doctoral work has been performed at the Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU), Trondheim, under supervision of Prof. Richard E. Blake.

The thesis work has been funded by the Computational Science and Engineering project, having the number #121455 from the Research Council of Norway, during October 1999 – October 2003.

Acknowledgements

I would like to thank you to my supervisor Prof. Richard E. Blake, for challenge, advice and support, to my friends from the 3rd floor for being there for interminable discussions and working sessions (especially Jörg, Gerthory and Pavel – co-worker to the QADPZ system), to all the people at IDI for their support and kindness, and, of course, to my family.

I would also like to express my gratitude to the members of the evaluation committee, Algimantas Juozapavicius, Anne C. Elster, and Brian Vinter, for their valuable comments that have lead to significant improvements of this thesis, and for smooth handling of the evaluation process.

Contents

ABSTRACT I

ACKNOWLEDGEMENTS VII

CONTENTS IX

LIST OF FIGURES XIII

1 INTRODUCTION 1

1.1 Problem description 1

1.2 Research goals 5

1.3 Results and contributions 7

1.4 Outline of the thesis 9

2 SCIENTIFIC COMPUTING 13

2.1 What is Scientific Computing (all about)? 13

2.2 Mathematical Modeling 18

2.2.1 Modeling 18

2.2.2 Validation 19

2.3 The Process of Numerical Solution 21

2.4 The Computational Environment 23

3 SCIENTIFIC VISUALIZATION 27

3.1 Visualization 27

3.2 Scientific Visualization 31

3.3 Applications of Visualization 32

3.4 Algorithms for Scientific Visualization 38

3.5 Visualization Environments 41

3.6 Graphical excellence guidelines 41

4 COMPUTATIONAL GRIDS AND DESKTOP GRIDS 43

4.1 Distributed and Parallel Computing 43

4.2 Computational Grids and Applications 45

4.2.1 A bit of Grid history 45

4.2.2 Need for Computational Grid in Context ... 46
4.2.2.1 Data-Intensive Science ... 46
4.2.2.2 Simulation-Based Science ... 49
4.2.2.3 Remote Access to Experimental Apparatus ... 50
4.2.2.4 Virtual Community Science ... 50
4.2.2.5 Scenarios for grid use in the real-world ... 52
4.3 Premises for Computational Grid ... 53
4.3.1 Technical premises ... 53
4.3.2 Financial premises ... 55
4.3.3 Experiencing premises ... 59
4.4 Computational Grid Definition ... 65
4.5 Short Taxonomy of Grid Applications ... 68
4.6 Grid's Integrability, Efficiency and Quality of Services ... 71
4.7 Desktop Grid Computing ... 74
4.7.1 SCEs' Capabilities and Requirements ... 75
4.7.2 High-Throughput SCEs or Desktop Grids ... 76
4.7.2.1 Key Components for Desktop Grids ... 77
4.7.2.2 Requirements for Desktop Grids ... 79
4.7.2.3 External Interfaces and Guarantees ... 89
4.7.2.4 Hardware Requirements ... 89
4.7.2.5 High-Throughput SCEs in Grids ... 89
4.7.3 High-Reliability SCEs ... 90
4.7.3.1 External Interfaces and Guarantees ... 90
4.7.3.2 Hardware Requirements ... 90
4.7.3.3 High-Reliability SCEs in Grids ... 90
4.7.4 Dedicated High-Performance SCEs ... 91
4.7.4.1 Beowulf Clusters ... 92
4.7.4.2 Commercial Resource Virtualization Systems ... 92
4.7.4.3 External Interfaces and Guarantees ... 93
4.7.4.4 Hardware Requirements ... 93
4.7.4.5 Dedicated High-Performance SCEs in Grids ... 93
4.7.5 Concluding comments ... 93
5 OVERVIEW AND TAXONOMY OF DESKTOP GRID SYSTEMS ... 97
5.1 Overview of Desktop Grid Systems ... 97
5.1.1 SETI@home - BOINC ... 97
5.1.2 distributed.net ... 99
5.1.3 Considerations on parallelism for SETI@home - distributed.net ... 100
5.1.4 PVM ... 101
5.1.5 Entropia ... 104
5.1.6 Condor ... 106

5.2 Hierarchical Taxonomy 107
5.2.1 Level 1, Infrastructure: resource, platform, scalability, security 107
5.2.2 Level 2, Models: computing model, architecture, data model..... 108
5.2.3 Level 3, SW.: application, architecture, administration, license.. 110

6 CONCEPTUAL MODEL 115

6.1 Introduction 115

6.2 The Master-Worker Model 115
6.2.1 Decomposition and Distribution of Work-units 116
6.2.1.1 Static decomposition, static distribution 119
6.2.1.2 Dynamic decomposition, static distribution..... 120
6.2.1.3 Dynamic decomposition, dynamic distribution 122

6.3 Improved Master-Worker Model 124
6.3.1 Pull vs. Push for work-units..... 125
6.3.2 Pipelining of work-units..... 126
6.3.3 Sending more work-units at a time..... 128
6.3.4 Adaptive number of workers..... 129
6.3.5 Adaptive timeout interval for work-units 130
6.3.6 Use of multithreading 131

6.4 Resource Estimation..... 131
6.4.1 Network Performance..... 131
6.4.2 Computing Power 132

6.5 Resource Monitoring 133

6.6 Scheduling 134

7 THE QADPZ SYSTEM..... 135

7.1 Description..... 135

7.2 Justification for a New Desktop Grid System 135

7.3 Design and Implementation..... 137
7.3.1 Requirements 137
7.3.1.1 System Requirements..... 137
7.3.1.2 Interface Requirements 139
7.3.1.3 Non-functional Requirements 139
7.3.2 Architecture 140
7.3.2.1 Job-view of the system 142
7.3.2.2 Slave..... 144
7.3.2.3 Master 145
7.3.2.4 Client 146
7.3.2.5 User Interface 146
7.3.3 Communication 147

7.3.4 Parallel Computing 151
7.3.5 Interplatform operability 152
7.3.6 Security 152
7.3.7 Autonomic Computing Features 153
7.3.7.1 Self-knowledge 154
7.3.7.2 Self-configuration 155
7.3.7.3 Self-optimization 155
7.3.7.4 Self-healing 155
7.4 Get Started with QADPZ 156
7.4.1 User modes 156
7.4.2 Installation and maintenance features 158
7.4.3 Security 158
7.4.4 Architecture 159

8 THE QADPZ USAGE ON SOURCEFORGE.NET 163

9 SCIENTIFIC COMPUTING AND VISUALIZATION EXPERIMENTS 169
9.1 Computational Resource Monitoring 169
9.2 Real word problem - Trondheim fjord 172
9.3 Fluid flow around a cylinder - simulation 174
9.4 Fluid flow around a cylinder - visualization 177
9.5 Utilizing QADPZ for Evolutionary Computation 180
9.6 Adaptive Compression for Remote Visualization 180

10 CONCLUSIONS AND FUTURE WORK 193

SELECTIVE BIBLIOGRAPHY 203

APPENDIX 1. FEEDBACK AND REACTIONS TO QADPZ 209

List of Figures

Figure 2.1 From problem to solution in Scientific Computing 18
Figure 2.2 Validation of a mathematical model 21
Figure 3.1 Cave Painting in Lascaux 28
Figure 3.2 Kotsushika Hokusai - The Greate Wave 28
Figure 3.3 Great wave discretization 30
Figure 5.1 SETI@home screenshot 98
Figure 5.2 SETI@home architecture 98
Figure 5.3 distributed.net statistics screen 101
Figure 5.4 PVM Computing Model 103
Figure 5.5 Entropia Sandbox Model 105
Figure 6.1 Master-Worker model 115
Figure 6.2 Decomposition and Distribution of Work-units 117
Figure.6.4 Dynamic decomposition strategy 118
Figure 6.5 Computation times on workers: static decomp-static distrib 119
Figure 6.6 Computation times on workers: dyn. decomp-static distrib 121
Figure 6.7 Computation times on workers: dyn. decomp-dyn. distrib 124
Figure 6.8 Worker timeline in execution 125
Figure 6.9 Pull vs. Push technology 126
Figure 6.10 Pipelining of worker tasks 126
Figure 6.11 Worker timeline for unit pipeline 127
Figure 6.12 Unit pipeline - worst case 127
Figure 6.13 More results at a time 128
Figure.6.14 Model timeout history 130
Figure 7.1 QADPZ requirements 137
Figure 7.2 QADPZ system requirements 137
Figure 7.3 QADPZ interface requirements 139
Figure 7.4 QADPZ non-functional requirements 140
Figure 7.5 QADPZ coarse architecture 141
Figure 7.6 QADPZ detailed architecture 142
Figure 7.7 QADPZ close-up architecture 143
Figure 7.8 Simplified UML Diagram of QADPZ's architecture 143
Figure 7.9 QADPZ job life 144
Figure 7.10 QADPZ slave info user interface 145
Figure 7.11 QADPZ job monitoring web-interface 146
Figure 7.12 QADPZ resource monitoring web-interface 147
Figure 7.13 QADPZ slave configuration interface 148
Figure 7.14 QADPZ communication layers 150
Figure 7.15 Reliable UDP communication 150
Figure 7.16 MPI communication 151
Figure 8.1 QADPZ in Distributed Computing projects 164
Figure 8.2 QADPZ home page 165

Figure 8.3 QADPZ project summary 165
Figure 8.4 QADPZ download statistics 166
Figure 8.5 QADPZ downloads 166
Figure 8.6 QADPZ webhits statistics 167
Figure 9.1 Available desktop computers in laboratory (day 1) 169
Figure 9.2 Available desktop computers in laboratory (day 2) 170
Figure 9.3 Available desktop computers in laboratory (day 3) 170
Figure 9.4 Available desktop computers in laboratory (day 4) 171
Figure 9.5 Available desktop computers in laboratory (day 5) 171
Figure 9.6 Available desktop computers in laboratory (day 6) 171
Figure 9.7 Maps of the Trondheim fjord 172
Figure 9.8 Trondheim fjord (left: topography, right: grid) 173
Figure 9.9 Grid colored by salinity concentration 173
Figure 9.10 Trondheim fjord model 174
Figure 9.11 Velocity vector field - LIC representation 174
Figure 9.12 3D representation surface- vector field top layer 174
Figure 9.13 Execution times for the solver 176
Figure 9.14 Speedup for the simulation 177
Figure 9.15 Flow around cylinder grid 177
Figure 9.16 Flow around cylinder (measurement and simulation) 178
Figure 9.17 Streamline 179
Figure 9.18 Streakline 179
Figure 9.19 Flow around 3 cylinders 179
Figure.9.20 Remote visualization 182
Figure.9.21 The agent-environment interaction. 186
Figure.9.22 Frame rate variations and average 187
Figure.9.23 The adaptive algorithm 188
Figure.9.24 Vizserver architecture 188
Figure.9.25 Average frame rate - 100 MBps network 190
Figure.9.26 Average frame rate - 100 MBps network. 190

1 Introduction

1.1 Problem description

Technology may be the product of knowledge and intense work. Nevertheless, people want it to work like magic. And when technology users want to accomplish something, the last thing they want to think about is how to do it. That is why the scientific and engineering community, and increasingly the business world, are welcoming Grid computing with the kind of enthusiasm inspired by the Internet not long ago, when its standards and technologies began the march toward near-universal connectivity, broad access to content, and a new model for science, engineering, business and for life itself. That development was extraordinary in many respects, not least because it was a major step in Information Technology's historic evolution toward total integration into our society. In that passage of Information Technology (IT) to mass adoption, Grid computing could be as momentous as the Internet itself.

Grid computing is a model of distributed computing that uses geographically and administratively disparate resources that are found on the network. These resources may include processing power, storage capacity, specific data, and other hardware such as input and output devices. In grid computing, individual users can access computers and data transparently, without having to consider location, operating system, account administration, and other details. Moreover, the details are abstracted, and the resources are virtualized. Grid computing seeks to achieve the secured, controlled and flexible sharing of resources (for example, multiple computers, software and data) among various dynamically created virtual organizations (Foster and Kesselman, 2004) (Cummings, 2007), which are generally setup for collaborative problem solving and access to grid resources are limited to those who are part of the project. The creation of an application that can benefit from Grid computing (faster execution speed, linking of geographically separated resources, interoperation of software, etc.) typically requires the installation of complex supporting software and an in-depth knowledge of how this complex supporting software works.

Grid computing systems can be classified into two broad types. The first type are heavy-weight, feature-rich systems that tend to concern themselves primarily with providing access to large-scale, intra- and inter-institutional resources such as clusters or multiprocessors. The second general class of Grid computing systems is the Desktop Grids, in which cycles are scavenged from idle desktop computers. The typical and most appropriate application for desktop grid is comprised of independent tasks (no communication exists amongst tasks) with a high computation to communication ratio.

In a desktop grid system, the execution of an application is orchestrated by a central scheduler node, which distributes the tasks amongst the worker

nodes and awaits workers' results. It is important to note that an application only finishes when all tasks have been completed. The main difference in the usage of institutional desktop grids relatively to public ones lies in the dimension of the application that can be tackled. In fact, while public projects usually embrace large applications made up of a huge number of tasks, institutional desktop grids, which are much more limited in resources, are more suited for modestly-sized applications. So, whereas in public volunteer projects importance is on the number of tasks carried out per time unit (*throughput*), users of institutional desktop grids are normally more interested in a fast execution of their applications, seeking fast *turnaround time*.

The attractiveness of exploiting desktop grid systems is further reinforced by the fact that costs are highly distributed: every volunteer supports her resources (hardware, power costs and internet connections) while the benefited entity provides management infrastructures, namely network bandwidth, servers and management services, receiving in exchange a massive and otherwise unaffordable computing power. The usefulness of desktop grid computing is not limited to major high throughput public computing projects. Many institutions, ranging from academics to enterprises, hold vast number of desktop machines and could benefit from exploiting the idle cycles of their local machines. In fact, several studies confirm that CPU idleness in desktop machines averages 95% (Heap, 2003), (Domingues et al., 2005).

The needs for more accurate simulations, combined with advances in computer hardware performance, are generating larger and larger amounts of numerical results. Workstations, minicomputers, and image computers are significantly more powerful and effective visualization tools than supercomputers. It is a waste of super-computer cycles to use them to convert this data into new pictures. Specialized graphic processors are more cost-effective than supercomputers for specialized picture processing and/or generation. Researchers must have easy access to local or distributed resources for high quality computations and visualizations.

Scientific Computing (or Computational Science) is the field of study concerned with constructing mathematical models and numerical solution techniques, and with using computers to analyze and solve scientific and engineering problems. In practical use, it is typically the application of computer simulation and other forms of computation to problems in various scientific and engineering disciplines. The field is distinct from computer science (the mathematical study of computation, computers and information processing). It is also different from theory and experiment, which are the traditional forms of science and engineering. The Scientific Computing approach is to gain understanding, mainly through the analysis of mathematical models implemented on computers. As Richard Hamming has observed many year ago, "the purpose of Scientific Computing is insight, not numbers" (McCormick, 1988).

Scientific Computing programs often model real-world changing conditions, such as weather, air flow around a plane, automobile body distortions in a crash, the motion of stars in a galaxy, an explosive device, etc. Such programs might create a 'logical mesh' in computer memory where each item corresponds to an area in space and contains information about that space relevant to the model. For example in weather models, each item might be a square kilometer; with land elevation, current wind direction, humidity, temperature, pressure, etc. The program would calculate the likely next state based on the current state, in simulated time steps, solving equations that describe how the system operates, and then repeat the process to calculate the next state.

Scientists and engineers develop software systems that implement the models of the systems being studied and run these programs with various sets of input parameters. Typically, these models require massive amounts of calculations (usually floating-point) and are often executed on supercomputers or distributed computing platforms.

Visualization could help overcome the dilemma of having information, but not the right interpretation for it. Interactive computing and visualization would be an invaluable aid during the scientific discovery process, as well as a useful tool for gaining insight into scientific anomalies or computational errors. Scientist needs an alternative to numbers. A cognitive possibility and technical reality is the use of images. The ability of scientists to visualize complex computations and simulations is absolutely essential to ensure the integrity of the analysis, to provoke insights, and to communicate about them with others. Scientific and Information Visualization are concerned with presenting data to users by means of images. Both fields seek ways to help users explore, make sense of, and communicate about data. They are active research areas, drawing on theory in information graphics, computer graphics, human-computer interaction and cognitive science.

Information Visualization and Scientific Visualization have overlapping goals and techniques. There is currently no clear consensus on the boundaries between these fields, but broadly speaking the two areas can be distinguished as follows: Scientific Visualization deals primarily with data that has a natural geometric structure (e.g. MRI data or wind flows), and Information Visualization handles more abstract data structures.

A related term, Visual Analytics, focuses on human interaction with visualization systems as part of a larger process of data analysis. Visual analytics is considered the science of analytical reasoning supported by the interactive visual interface. Its focus is on human information discourse (interaction) within massive, dynamically changing information spaces. Visual analytics research concentrates on support for perceptual and cognitive operations that enable users to detect the expected and discover the unexpected in complex information space. Technologies resulting from visual analytics find their application in almost all fields, but are being driven by critical needs (and funding) in biology and national security.

Through a strong US government financial support Scientific Visualization prospered specifically after the mid ´80s. A key event for the growth of Scientific Visualization was the appearance of a report based on an NSF sponsored workshop (McCormick et al., 1987). Scientific Visualization is a relatively new, exciting field of computational science spurred on in large measure by the rapid growth in computer technology, particular in graphics workstation hardware and computer graphics software. Visualization tools are beginning to impact our daily lives through usage in art (e.g. film animation), and they hold great promise for scientific research and education. The goal of visualization is to leverage existing scientific methods by providing new scientific insight through visual methods. An estimated 50 percent of the brain's neurons are associated with vision. Visualization in Scientific Computing aims to put that neurological machinery to work.

Moreover, there is every indication that the number of data sources will multiply, as will the data density of these sources. For example, the definition of a supercomputer is changing from its former meaning of 0.1 - 1.0 gigaflops (billions of floating-point operations per second) to intermediate 1 - 10 gigaflops, and up-to-date 500Tflops (Top500, 2007). Also, current earth resource satellites have resolutions 10 -100 times higher than satellites orbited just a few years ago. Scientists involved in the computational sciences require these data sources to conduct significant research; however, the flood of data generated overwhelms them. Using an exclusively numerical format, the human brain cannot interpret gigabytes of data each day, and therefore much information will goes to waste.

Moreover, scientists not only want to analyze data that results from super-computations, they want to interpret what is happening to the data during super computations. Scientist wants to steer calculations in close-to-real-time, they want to be able to change parameters, resolution, or representation, and see the effects. Basically, scientists want to be able to interact with their data.

Some of the domains and directions in which Scientific Computation and Visualization are able to give valuable insight are listed here: engineering, computational fluid dynamics, finite element analysis, electronic design automation, simulation, medical imaging, geospatial, RF propagation, meteorology, hydrology, data fusion, ground water modeling, oil and gas exploration and production, finance, data mining/OLAP, numerical simulations, orbiting satellites returning earth resource, military intelligence, astronomical data, spacecraft sending planetary and interplanetary data, earthbound radio astronomy arrays, instrumental arrays recording geophysical entities, such as ocean temperatures, ocean floor features, tectonic plate and volcanic movements, and seismic reflections from geological strata, medical scanners employing various imaging modalities, such as computed transmission and emission tomography, and magnetic resonance imagery.

There are many complex computational and visualization algorithms,

which require large amounts of computational power. In many cases the computing power of a single desktop computer is insufficient for generating such complex visualizations, or it takes too long time to generate them. The situation becomes more complicated when the data sets are very large. Traditionally, large parallel supercomputers were used for generating such complex visualizations. However, very high initial investments and maintenance costs limited the availability of such systems to very few research labs.

A more convenient solution, which is becoming more and more popular, is based on commodity clusters, consisting of cheaper personal computers connected by large bandwidth network. In a similar way, the cost of such systems is still high due to the large number of cluster nodes required by the computationally intensive applications.

An alternative approach is the use of non-dedicated desktop PCs in a desktop grid computing environment. Harnessing idle CPU cycles and storage space of networked computers to work together on a particularly computational intensive application does this. Increasing power and communication bandwidth of desktop computers are helping to make distributed computing a more practical idea. By using existing desktop computers from a local network, the cost of such an approach is low compared with parallel supercomputers and dedicated clusters.

We finally conclude that science, industry, business and other domains can benefit from Grids and Desktop Grids. However, at the risk of stating the case too broadly, we make a more comprehensive statement. A primary purpose of information technology and infrastructure is to enable people to perform their daily tasks more efficiently or effectively. To the extent that these tasks are performed in collaboration with others, grids are more than just a niche technology, but rather a direction in which our infrastructure must evolve if it is to support our social structures and the way work gets done in our society.

1.2 Research goals

This thesis mainly deals with processing power as the vital resource. The motivation for harnessing the available processing power on the network is simple: to increase the size of problems that can be solved, to increase performance and obtain results faster. Consider a typical local area network, where many low-price machines on the network will be idle for significant periods of time. If these wasted processor cycles could be utilized, they could represent a significant processing resource. This approach provides a more flexible and cost effective processing system. Normally, workstations may be in use as desktop machines, but become part of a distributed computation resource when not in use, for example, at night or during weekends.

In many universities, research organizations, and, lately, enterprises, the following recent trends can be identified: larger amounts of data are being accumulated and manipulated; hardware performance of desktop computers increases dramatically; new technological advancements stimulate use of computing applications with extreme requirements for computational power; use of computing, simulations, visualizations, and optimization in various research fields and practical applications is accelerating and leads to very high demands on computing power; and the pace of development of high-performance servers hardly equals these trends, but for very high financial costs. Increasing hardware performance of desktop computers accounts for a low-cost high-performance computing potential that is waiting to be efficiently put in use.

Distributed Computing harnesses the idle processing cycles of the available workstations on the network and makes them available for working on computationally intensive problems that would otherwise require a supercomputer or a dedicated cluster of computers to solve. A distributed computing application is divided to smaller computing tasks, which are then distributed to the workstations to process in parallel. Results are sent back to the server, where they are collected. The more PCs in a network, the more processors available to process applications in parallel, and the faster the results are returned. A network of a few thousand PCs can process applications that otherwise can be run only on fast and expensive supercomputers. This kind of computing can transform a local network of workstations into a *virtual supercomputer.*

In this thesis we present a solution that has been developed to meet the above needs. It consists of a conceptual model for desktop grid computing and the system that has been developed according to it, QADPZ ['kwod "pi: 'si:], a modular, open source, object oriented implementation in C++, of a multi-user and multi-platform desktop grid system. The computing power of large number of idle desktop computers is utilized by automatically scheduled tasks that are submitted, monitored, and controlled by users. Flexibility of the system is implied by several user application modes. Task software and hardware requirements and input/output files are handled automatically by the system. Internal communication protocol is based on optionally encrypted XML messages using public/private keys, user names and passwords. QADPZ can operate both in conditions of an open Internet environment and of a closed local network which supports the family of TCP/IP protocols. QAPDZ has important autonomic features as well. The system is currently in use for research tasks in the areas of large-scale Scientific Visualization, evolutionary computation, simulation of complex neural network models, and other computationally intensive applications. Besides that, QADPZ can also be used a as research environment for studying different algorithms related to distributed computing, different scheduling policies etc., or as an educational environment

to study different aspects related to operating systems, distributed systems, mobile agents, parallel algorithms, etc.

The QADPZ design goals have been ease of use at different user skill levels, inter-platform operability, modularity and modifiability, client-master-slave architecture using fast message based communication, security of computers participating in QADPZ, and easy and automatic installation and upgrade. In QADPZ, a small software program (*slave service*) runs on each desktop workstation. As long as the workstation is not being utilized, the slave service accepts tasks sent by the server (*master*). The available computational power is used for executing a task. Human system administration required for the whole system is minimal.

1.3 Results and contributions

In the work presented in this thesis, the central idea has been to provide a desktop grid computing framework and to prove its viability by testing it in some Scientific Computing and Visualization experiments. We present here QADPZ, an open source system for desktop grid computing, which enables users from a local network or even Internet to share their resources. It is a multi-platform, heterogeneous system, where different computing resources from inside an organization can be used. It can also be used for volunteer computing, where the communication infrastructure is the Internet. QADPZ supports the following native operating systems: Linux, Windows, MacOS and Unix variants. The reason behind natively supporting multiple operating systems, and not only one (Unix or Windows, as other systems do), is that often, in real life, this kind of limitation restricts very much the usability of desktop grid computing.

QADPZ provides a flexible object-oriented software framework that makes it easy for programmers to write various applications, and for researchers to address issues such as adaptive parallelism, fault-tolerance, and scalability. The framework supports also the execution of legacy applications, which for different reasons could not be rewritten, and that makes it also suitable for other domains as business. It also supports either low-level programming languages as C and C++ or high-level language applications, like for example Lisp, Python, and Java, providing the necessary mechanisms to use such applications in a computation. Consequently, users with various backgrounds can benefit from using QADPZ. The flexible, object oriented structure, the modularity of the system along with the open-sourceness provide for easy improvements and further extensions to other programming languages.

We developed a general-purpose runtime and an API to support new kind of high performance computing applications, and therefore to benefit from the advantages offered by desktop grid computing. We have shown how distributed computing extends beyond the master-worker paradigm, typical for

such systems, and provided QADPZ with an extended API which supports in addition lightweight tasks creation and parallel computing, using the message passing paradigm (MPI). The API directly supports the C/C++ programming language. QADPZ supports parallel programs running on the desktop grid, by providing an API in the C/C++ language, which implements a subset of the MPI standard. This extends the range of applications that can be used in the system to already existing MPI based applications, e.g. parallel numerical solvers from computational science, or parallel visualization algorithms.

Another restriction of existing systems, especially middleware based, is that each resource provider needs to install a runtime module with administrator privileges (root, supervisor). This poses some issues regarding data integrity and accessibility on providers' computers. The QADPZ system tries to overcome this by allowing the middleware module to run as a non-privileged user, even with restricted access, to the local system.

QADPZ provides also low-level optimizations, such as on-the-fly compression and encryption for communication. The user can choose from different algorithms, depending on the application, improving both the communication overhead imposed by large data transfers and keeping privacy of the data. The system goes further, by providing an experimental, adaptive compression algorithm, which can transparently choose different algorithms to improve the application. QADPZ also support two different communication protocols (UDP and TCP/IP) in order to improve the efficiency of communication.

Free availability of the source code allows its flexible installations and modifications based on the individual needs of research projects and institutions. In addition to being a very powerful tool for computationally-intensive research, the open-source availability makes QADPZ a flexible educational platform for numerous small-size student projects in the areas of operating systems, distributed systems, mobile agents, parallel algorithms, and others. More, free/open source software is a natural choice for modern research, as well, because it encourages integration, cooperation and boosting of new ideas, in a very effective way. We offered the QADPZ system as open source from the beginning, at a time when very few such solution were free, with all the positive implications of this for research and computationally intensive applications.

This thesis proposes an improved conceptual model (based on the master-worker paradigm), which makes contributions in several directions (pull vs. push work-units, pipelining of work-units, more work-units sent at a time, adaptive number of workers, adaptive time-out interval for work-units, and multithreading).

Beside the extended master-worker conceptual model and the QADPZ desktop grid system, this thesis make contributions in form of a hierarchical taxonomy of the main existing desktop grids, and of an adaptive compression algorithm for remote visualization. We have also been trying to demonstrate

that the use of desktop grid computing should not be limited to only master-worker type of application, but can be used also for more fine-grained parallel applications, in the field of Scientific Computing and Visualization, by performing some experiments in those domains. The system is currently used for research tasks in the areas of large-scale scientific visualization, evolutionary computation, simulation of complex neural network models, and other computationally intensive applications. It is worth to mention that to the present, the QADPZ has over a thousand downloads, from users who use it for their tasks, as it can be seen in the appendix.

Some of the results of this thesis have already been published (they are listed in the references) and some are in course of publication. Thus, contributions that are already published concern: the QADPZ system (Constantinescu and Petrovic, 2002) and (Constantinescu *et al.*, 2002), QADPZ proven to be useful in Scientific Computing – example of using it to solve the Navier Stokes equation for fluid dynamics (Constantinescu, 2003), QADPZ as an autonomic distributed computing system (Constantinescu, 2003), and the hierarchical taxonomy of desktop grid systems built from users' perspective (Constantinescu and Vladoiu, 2008). The paper on QADPZ's autonomicity has been highly cited since it has been published and considered as pioneering this approach in desktop grids, as it can be seen in the appendix. The results on QADPZ, as a viable desktop grid/volunteer computing open solution, which can also use parallel computing techniques using the MPI layer - this is a novel approach in desktop grid, on the improved master worker model, on the adaptive compression algorithm for remote visualization, on master virtualization, on QADPZ testing in some experimental scientific visualizations, and on QADPZ development journey are in course of publication.

1.4 Outline of the thesis

This thesis consists of an abstract, ten chapters and one appendix. In the *abstract*, the reader is familiarized briefly with the basic ideas from Scientific Computing and Visualization, then it is shown that the computationally intensive problems from those domains can be solved by using Desktop Grid Computing Systems (that are briefly described), and finally, the desktop grid framework which has been developed during this work, the QADPZ system, is presented, along with the main contributions of this thesis. The first chapter, *Introduction,* establish the boundaries of the problem to be solved, i.e. providing an environment of desktop grid computing and proving its viability by testing it in some computationally intensive experiments. In the beginning, it is shown that, due to the fact that people need the technology to work like magic, Grid and Desktop Grids are welcomed with the kind of enthusiasm inspired by the Internet not so long ago. After that, Grid and Desktop Grids are briefly introduced, with their features and challenges. Then, the Scientific Computing and Scientific Visualization domains are presented in a few words, along with

their specific requirements for huge computing power and other resources. The importance of proper visualization to gain insight in the real world modeled problems is then emphasized. Finally, it is revealed that grids are more than just a niche technology, but rather a direction in which our infrastructure must evolve if it is to support our social structures and the way work gets done in our society. Before its end, Introduction includes also the research goals and the thesis results and contributions.

The second chapter, *Scientific Computing*, tries to respond the question, what is Scientific Computing all about?, and it concludes by the working definition that says that it is the collection of tools, techniques, and theories required to solve on a computer, mathematical models of problems in science and engineering. The path from a scientific or engineering problem, via mathematical modeling, numerical analysis and computer science that converge through Scientific Computing to a solution is illustrated as well. Besides that, specific problems of the process of numerical solution are presented, because of their importance when using computers to find that solution (errors, approximations etc.). Finally, the computing environment in which Scientific Computing takes place, is discussed.

Scientific Visualization is the subject of the third chapter. It starts with establishing the human nature of visualization, and then shifts to presentation of various sorts of computer-based visualizations: interactive visualizations, animations, abstract and model-based visualizations etc. Next, the main reasons for the need of Scientific Visualization are presented, along with the domains from which visualization has been bred. The chapter continues with detailed descriptions of some applications of visualization, from which the critical requirements of these applications arise. Then a few words about visualization algorithms, environments and graphical excellence guidelines are provided.

The fourth chapter, *Computational Grids and Desktop Grids*, introduces the main domain of this thesis. First, some basic ideas about distributed and parallel computing are presented, and then a bit of grid history is brought to the light. Further on, the need for computational grid along with its context is established. Data-intensive science, simulations, remote apparatus and virtual community science are briefly presented, along with their specific needs that demand for grid facilities. Then, an argumentation, based on exemplifying scenarios, that grid is needed also outside the scientific and engineering world, to solve real day to day problems, is given. After establishing the need for it, the premises (technical and financial) for making the Grid happen are summarized. At this point, two milestone definitions for Grid are provided, and their key elements are discussed. The chapter continues with a short taxonomy of grid applications, and concludes the Grid part with some integrability, efficiency and quality of services' issues. The second part of this chapter concerns Desktop Grids, and it starts with introducing small composite elements, as basic elements for desktop grids (high-throughput computing), high-reliability clusters and high-performance clusters. The key components and requirements

for desktop grids are given, along with detailed robustness, communication, and security issues.

The fifth chapter, which is entitled *Overview and Taxonomy of Desktop Grid Systems,* runs a survey of the most remarkable desktop grids (SETI@home-BOINC, distributed.net, PVM, Entropia, and Condor), and, then synthesize a hierarchical three level taxonomy for desktop grids. The first level refers to infrastructure and includes resource type, the platform that runs at the provider, scalability and security issues. The second one includes conceptual model, architecture and data model, under the umbrella of models. The last level concerns aspects related to software: application type, need for administrator privileges, architecture of the support operating system, and licensing. Before the end of the chapter, a table with the classification of the main desktop grid systems according to this taxonomy is provided. Examining few typical application scenarios has eased crafting a user-centric taxonomy. We hope that our approach will help promote the introduced taxonomy as a practice for its potential users.

The *Conceptual Model* chapter present first the master-worker model for distributed computing, then gives some decompositions and distributions of the work-units, and finally introduce an improved conceptual model, which makes contributions in several directions (pull vs. push work-units, pipelining of work-units, more work-units sent at a time, adaptive number of workers, adaptive time-out interval for work-units, multithreading, resource estimation and monitoring, scheduling).

The seventh chapter, *The QADPZ system,* deals with detailed presentation of the desktop grid system that has been developed during this thesis work. In the first place, a justification for the need for a new such system is given. Then, in the Design and Implementation section, the requirements for the system are reviewed, and the QADPZ architecture comes along, with details about various components of this desktop grid framework (master, slave, client). Detailed explanations about the communication mechanism are provided further, here being included the parallel computing feature as well. Interplatform operability, security, and autonomic computing features of QADPZ bring to a close this section. Further on, a brief "get started" documentation is provided.

The eighth chapter is dedicated to the QADPZ users and it refers to *QADPZ usage on sourceforge.net, reactions to it and feedback*. Basically, we present the screenshot-based history of the system since its upload to this site, with emphasis on the number of hits and downloads of the systems. Within the appendix of this thesis, the raw feedback and reactions to the system are listed. These have been categorized into four main categories: feedback and support requests from users who use QADPZ for their research and development tasks, forum discussions, citations in papers, and working assignments, based on QADPZ features, for students from some universities.

The next chapter, *Experiments of Scientific Computing and Visualization*, presents some experiments we have performed by using QADPZ and other support systems: computational resource monitoring, real world problem: Trondheim fjord, fluid flow around a cylinder – simulation and visualization, evolutionary computation, and, finally, an adaptive compression for remote visualization.

The last chapter, *Conclusions,* re-states the need for Grid and Desktop Grid facilities for solving both scientific and engineering problems (with their complicated visualizations) and daily ones, then summarizes the contributions of this thesis work and the future work ideas, and finally concludes with asserting that, as we become capable of doing more and more with our advanced technologies and as we hide those technologies and their complexities from users, the results will indeed seem like magic.

2 Scientific Computing

2.1 What is Scientific Computing (all about)?

The numerous millions of computers now installed worldwide are used for an increasing puzzling variety of tasks: accounting and inventory control for industry and government, airline and other reservation systems, limited translation of natural languages, monitoring of process control, and so on. One of the earliest - and still one of the largest - uses of computers was to solve problems in science and engineering and, more specifically, to obtain solutions of mathematical models that represent some physical situation. The techniques used to obtain such solutions are part of the general area called *Scientific Computing*, and the use of these techniques to obtain insight into scientific or engineering problems is called *computational science* or *engineering*. Scientific Computing is concerned with the design and analysis of algorithms for solving mathematical problems that arise in many fields, especially science and engineering.

Scientific Computing is distinguished from most other parts of computer science in that it deals with quantities that are continuous. as opposed to discrete. It is concerned with functions and equations whose underlying variables time, distance, velocity, temperature, density, pressure, stress, and the like are continuous in nature. Must of the problems of continuous mathematics (for example, almost any problem involving derivatives, integrals, or nonlinearities) cannot be solved exactly, even in principle, in a finite number of steps and thus must be solved by a (theoretically infinite) iterative process that ultimately converges to a solution. In practice one does not iterate forever, of course, but only until the answer is approximately correct, "close enough" to the desired result for practical purposes. Thus, one of the most imperative aspects of Scientific Computing is finding rapidly convergent iterative algorithms and assessing the accuracy of the resulting approximation. If convergence is sufficiently rapid, even some of the problems that can be solved by finite algorithms, such as systems of linear algebraic equations, may in some cases be better solved by iterative methods.

Consequently, a second factor that distinguishes Scientific Computing is its concern with the effects of approximations. Many solution techniques involve a whole series of approximations of various types. Even the arithmetic that is used is only approximate, because digital computers cannot represent all real numbers exactly. In addition to having the usual properties of good algorithms, such as efficiency, numerical algorithms should also be as reliable and accurate as possible despite the various approximations made along the way (Heath, 2002).

Nowadays, there is hardly an area of science or engineering that does not use computers for modeling. Trajectories for earth satellites and for planetary missions are routinely computed. Engineers use computers to simulate the flow of air about a spacecraft or other aerospace vehicle as it passes through the atmosphere, and to verify the structural integrity of aircraft. Such studies are of crucial importance to the aerospace industry in the design of safe and economical aircraft and spacecraft. Modeling new designs on a computer can save many millions of dollars compared to building a series of prototypes. Similar considerations apply to the design of automobiles and many other products, including new computers.

Astronomers and astrophysicists have modeled the evolution of stars, and much of our basic knowledge about such phenomena as red giants and pulsating stars has come from such calculations corroborated with observations. Civil engineers study the structural characteristics of large bridges, buildings, dams, and highways. Meteorologists use large amounts of computer time to predict tomorrow's weather as well as to make much longer range predictions, including the possible change of the earth's climate. Biochemists visualize the effect of drugs on human cells. Ecologists and biologists are increasingly using the computer in such diverse areas as population dynamics (including the study of natural predator and prey relationships), the flow of blood in the human body, and the dispersion of pollutants in the oceans and atmosphere.

As the above examples suggest, many of the problems from Scientific Computing come from science and engineering, in which the ultimate aim is to understand some natural phenomenon or to design some device. Computational simulation, as a representation and an emulation of a physical system or process using a computer, can greatly enhance scientific understanding by allowing the investigation of situations that may he difficult or impossible to investigate by theoretical, observational, or experimental means alone. In astrophysics, for example, the detailed behavior of two colliding black holes is too complicated to determine theoretically and impossible to observe directly or duplicate in the laboratory (Heath, 2002). To simulate it computationally, however, requires only an appropriate mathematical representation (in this case Einstein’s equations of general relativity), an algorithm for solving those equations numerically, and a sufficiently large computer on which to implement the algorithm.

Computational simulation is useful not just for exploring exotic or otherwise inaccessible situations, however, but also for exploring a larger variety of normal scenarios than could otherwise be investigated with unreasonable cost and time. In engineering design, computational simulation allows a large number of design operations to be tried much more rapidly, inexpensively, and safely than with traditional prototyping methods. In this context, computational simulation has become known as virtual prototyping. In improving automobile safety, for example, crash testing is far less expensive

and dangerous on a computer than in real life, and thus the space of all possible design parameters can be explored much more thoroughly to develop an optimal design.

The overall problem solving process in computational simulation usually includes the following steps: *develop a mathematical model* usually expressed by equations of some type of a physical phenomenon or system of interest, *develop algorithms to solve the equations numerically*, *implement the algorithms* in software systems, *run the software* on a computer to simulate the physical process numerically, *represent the computed results* in some comprehensible form such as graphical visualization, and, finally, *interpret and validate the computed results*, repeating any or all of the preceding steps, if necessary.

The first step is often called *mathematical modeling*. It requires, specific knowledge of the particular scientific or engineering disciplines involved, as well as knowledge of applied mathematics. The next two steps that are concerned with designing, analyzing, implementing, and using numerical algorithms and software, are the main subject matter of Scientific Computing. It is essential that all of these steps, from problem formulation to interpretation and validation of results, be done properly for the results to be meaningful and useful. The principles and methods of Scientific Computing can be studied at a fairly broad level of generality, but the specific source of a given problem and the uses to which the results will be put should always be kept in mind, as each aspect affects and is affected by the others (Heath, 2002). For example, the original problem formulation may strongly affect the accuracy of numerical results, which in turn affects the interpretation and validation of those results.

A mathematical problem is said to be well-posed if a solution exists, is unique, and depends continuously on the problem data. The latter condition means that a small change in the problem data does not cause an abrupt, disproportionate change in the solution. This property is especially important for numerical computations, where such perturbations are usually expected. Well-posedness is highly desirable in mathematical models of physical systems, but this is not always achievable. For example, inferring the internal structure of a physical system solely from external observations, as in tomography or seismology, often leads to mathematical problems that are inherently ill-posed in that distinctly different internal configurations may have indistinguishable external appearances.

Even when a problem is well-posed the solution may still respond in a highly sensitive (though continuous) manner to perturbations in the problem data. To assess the effects of such perturbations, one must go beyond the qualitative concept of continuity to define a quantitative measure of the sensitivity of a problem. In addition, one must also take care to ensure that the algorithm that is used to solve a given problem numerically does not make the results more sensitive than is already inherent in the underlying problem. This need leads to the notion of a stable algorithm.

In seeking a solution to a given computational problem, a basic general strategy is to replace a difficult problem with an easier one that has the same solution, or at least a closely related solution. Examples of this approach include: replacing of infinite-dimensional spaces with finite-dimensional spaces, replacing infinite processes with finite processes (such as replacing integrals or infinite series with finite sums, or derivatives with finite differences), replacing differential equations with algebraic equations, replacing nonlinear problems with linear problems, substituting high-order systems with low-order systems, changing complicated functions with simple functions, such its polynomials, and replacing general matrices with simpler form ones.

For example, to solve a system of nonlinear differential equations, one might first replace it with a system of nonlinear algebraic equations, then replace the nonlinear algebraic system with a linear algebraic system, then replace the matrix of the linear system with one of a special form for which the solution is easy to compute At each step of this process, we would need to verify that the solution is unchanged, or is at least within some required tolerance of the true solution.

To make this general strategy work for solving a given problem, we have to have an alternative problem, or class of problems, that is easier to solve, and a transformation of the given problem into a problem of this alternative type that preserves the solution in some sense. Thus, much of the effort will go into identifying suitable problem classes with simple solutions and solution-preserving transformations into those classes.

Ideally, the solution to the transformed problem is identical to that of the original problem, but this is not always possible. In the latter case the solution may only approximate that of the original problem, but the accuracy can be made arbitrarily good at the expensive of additional work and storage. Thus, primary concerns are estimating the accuracy of such an approximate solution and establishing convergence to the true solution in the limit.

The mathematical models of all of these problems are systems of differential equations, either ordinary or partial. Differential equations come in all "sizes and shapes" (Golub and Ortega, 1993) and even with the largest computers we are nowhere near being able to solve many of the problems posed by scientists and engineers. But there is more to Scientific Computing, and the scope of the field is changing rapidly. There are many other mathematical models, each with its own challenges. In operations research and economics, large linear or nonlinear optimization problems need to be solved.

Data reduction, the condensation of a large number of measurements into usable statistics, has always been an important, if somewhat ordinary, part of Scientific Computing. However now there are available new tools (such as earth satellites) that have increased our ability to make measurements faster than our ability to assimilate them. Fresh insights are needed into ways to preserve and use this exceptional information. In more developed areas of engineering, what formerly were difficult problems to solve even once on a

computer are in our days routine problems that are being solved over and over with changes in design parameters. This has given rise to an increasing number of computer-aided design systems. Similar considerations apply in a variety of other areas.

Before presenting a definition, we must mention that delimiting the area of Scientific Computing nowadays is tricky, especially the boundaries and overlaps with other areas. Though we will agree to use the working definition that says that "Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer, mathematical models of problems in science and engineering" (Golub, 1997).

A preponderance of these theories, tools, and techniques was originally developed in mathematics, many of them having their origin long before the dawn of electronic computers. This set of mathematical theories and techniques is called *numerical analysis* (or numerical mathematics) and constitutes a major part of Scientific Computing. The development of the computers, however, indicated a new approach of the solutioning the scientific problems. Many of the numerical methods that had been developed for the purpose of hand calculation (including the use of desk calculators for the actual arithmetic) had to be revised and sometimes abandoned. Considerations that were irrelevant or insignificant for hand calculation now became of chief importance for the efficient and correct use of a large computer system. Many of these considerations with regard to programming languages, operating systems, management of large quantities of data, correctness of programs have been subsumed under the discipline of *computer science*, on which Scientific Computing now depends heavily.

Nevertheless mathematics itself continues to play a major role in Scientific Computing: it provides the language of the mathematical models that are to be solved and information about the appropriateness of a model (*Does it have a solution? Is the solution unique?*), and it provides the theoretical groundwork for the numerical methods and, increasingly, many of the tools from computer science.

In summary, then, Scientific Computing draws on modeling in science and engineering, numerical mathematics, and computer science to develop the best ways to use computer systems to solve problems from science and engineering. This relationship is depicted schematically in Figure 2.1, where the informational flow from problem to solution in Scientific Computing is drawn.

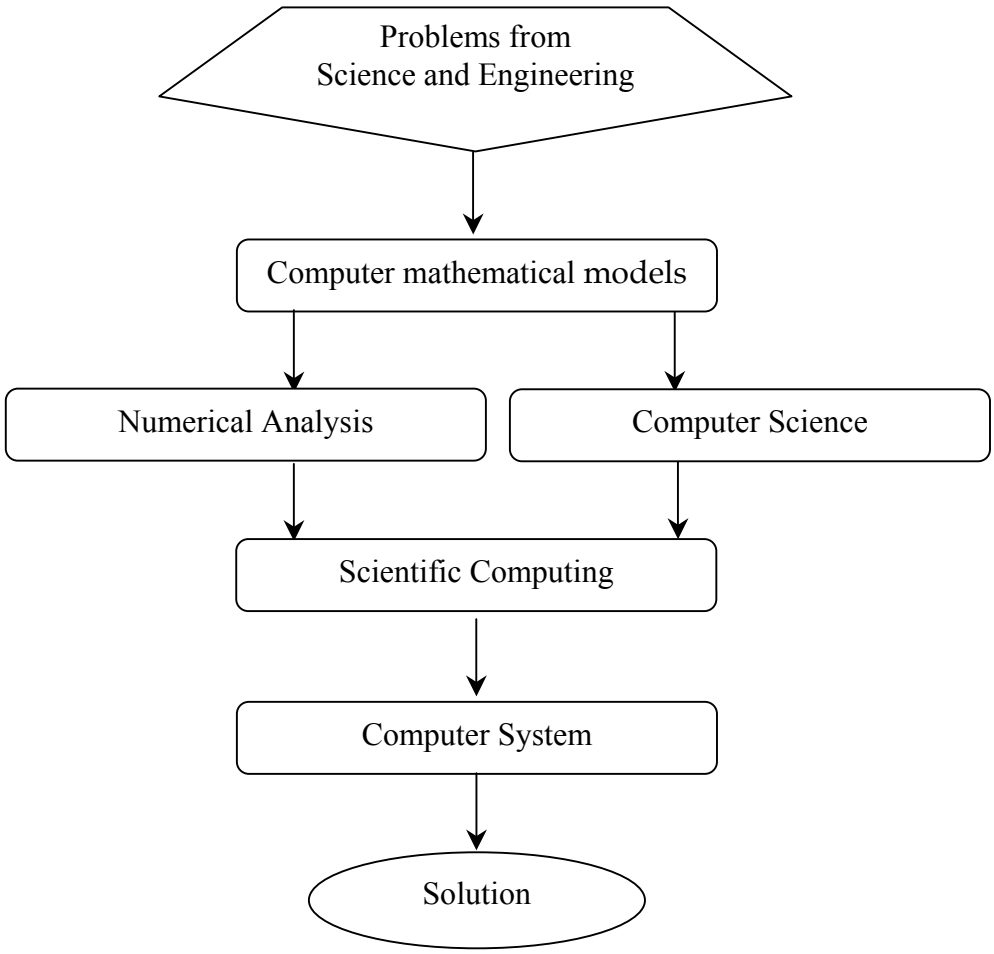


Figure 2.1 From problem to solution in Scientific Computing

2.2 Mathematical Modeling

As it was shown in the previous section, Scientific Computing is seen as the discipline that achieves a computer solution for mathematical models of problems from science and engineering. Therefore the first step in the overall solution process is the formulation of an appropriate mathematical model of the problem to be solved.

2.2.1 Modeling

The formulation of a mathematical model starts with a statement of the factors to be considered. In many physical problems, these factors concern the balance of forces and other conservation laws of physics. For example, in the formulation of a model of a trajectory problem the basic physical law is Newton's second law of motion, which requires that the forces acting on a body equal the rate of change of momentum of the body. This general law must then

be specialized to the particular problem by enumerating and quantifying the forces that will be of importance.

For example, the gravitational attraction of Jupiter will exert a force on a rocket in Earth's atmosphere, but its effect will be so little compared to the earth's gravitational force that it can usually be neglected. Other forces may also be small compared to the dominant ones but their effects not so easily dismissed, and the construction of the model will invariably be a compromise between retaining all factors that could likely have a bearing on the validity of the model and keeping the mathematical model sufficiently simple that it is solvable using the tools at hand. Traditionally, only very simple models of most phenomena were considered since the solutions had to be achieved by hand, either analytically or numerically. As the power of computers and numerical methods has developed, increasingly complicated models have become solvable.

In addition to the indispensable relations of the model - which in most situations in Scientific Computing take the form of differential equations - there usually will be a number of initial or boundary conditions. For example, in the predator-prey problem the initial population of the two species being studied is specified. In studying the flow in a blood vessel, we may require a boundary condition that the flow cannot penetrate the walls of the vessel. In some other cases, boundary conditions may not be so physically evident but are still essential so that the mathematical problem has a unique solution.

Or the mathematical model as first formulated may indeed have many solutions, the one of interest to be selected by some constraint such as a requirement that the solution be positive, or that it be the solution with minimum energy (Golub and Ortega, 1993), (Leopold, 2001). In any case, it is usually assumed that the final mathematical model with all appropriate initial, boundary, and side conditions indeed has a unique solution. The next step, then, is to find this solution. For problems of current interest, such solutions rarely can be obtained in "closed form." The solution must be approximated by some method, and the methods to be considered are numerical methods suitable for a computer. In the next section we will consider the general steps to be taken to achieve such a numerical solution.

2.2.2 Validation

Once we are able to compute solutions of the model, the next step usually is called the *validation of the model*. This means a verification that the computed solution is sufficiently accurate to serve the purposes for which the model was constructed. There are two main sources of possible error. First, there invariably are errors in the numerical solution. The general nature of these errors will be discussed in the next section, and one of the major research themes is a better understanding of the source and control of these numerical errors. But there is also invariably an error in the model itself. As mentioned previously, this is a

necessary aspect of modeling: the modeler has attempted to take into account all the factors in the physical problem but then, in order to keep the model tractable, has neglected or approximated those factors that would seem to have a small effect on the solution. The question is whether neglecting these effects was justified. The first test of the validity of the model is whether the solution satisfies obvious physical and mathematical constraints.

For example, if the problem is to compute a rocket trajectory where the expected maximum height is 100 kilometers and the computed solution shows heights of 200 kilometers, obviously some blunder has been committed. Or, it may be that we are solving a problem for which we know, mathematically, that the solution must be increasing but the computed solution is not increasing. Once such gross errors are eliminated, which is usually fairly easy, the next phase begins, which is, whenever possible, comparison of the computed results with whatever experimental or observational data are available. Many times this is a clever undertaking, since even though the experimental results may have been obtained in a controlled setting, the physics of the experiment may differ from the mathematical model. For example, the mathematical model of airflow over an aircraft wing may assume the idealization of an aircraft flying in an infinite atmosphere, whereas the corresponding experimental results will be obtained from a wind tunnel where there will be effects from the walls of the enclosure. Neither the experiment, nor the mathematical model represents the true situation of an aircraft flying in our finite atmosphere. The experience and intuition of the investigator are required to make a human judgment as to whether the results from the mathematical model are corresponding sufficiently well with observational data (Heath, 1997).

At the outset of an investigation this is quite often not the case, and the model must be modified. This may mean that additional terms - which have been thought insignificant, but may not be - are added to the model. Occasionally a complete revision of the model is required and the physical situation must be approached from an entirely different point of view. In any case, once the model is modified the cycle begins again: a new numerical solution, revalidation, additional modifications, and so on. This process is depicted schematically in Figure 2.2.

Once the model is estimated adequate from the validation and modification process, it is ready to be used for *prediction*. This, of course, was the whole purpose. We should now be able to answer the questions that gave rise to the modeling effort: How high will the rocket go? Will the wolves eat all the rabbits? Of course, we must always take the answers with a sound skepticism. Our physical world is simply too complicated and our knowledge of it too thin for us to be able to predict the future perfectly. Nevertheless, we hope that our computer solutions will give increased insight into the problem being studied, be it a physical phenomenon or an engineering design.

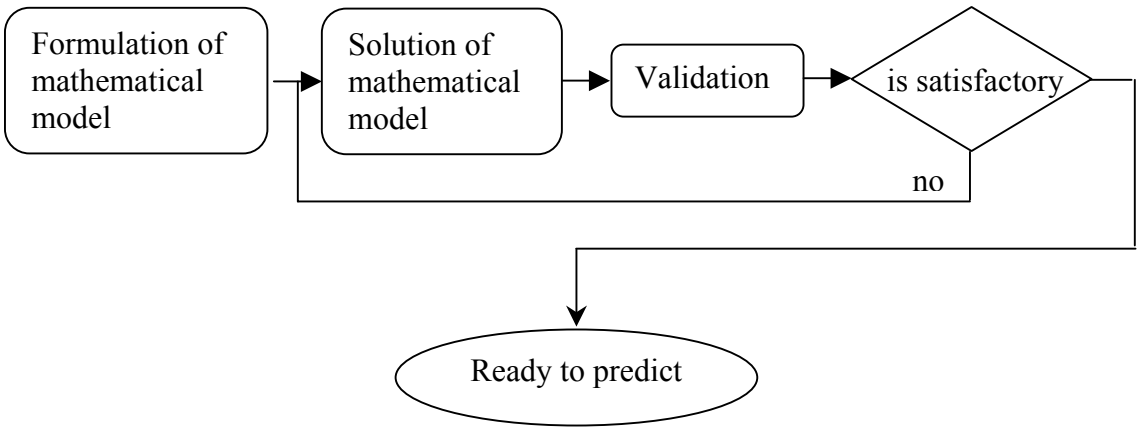


Figure 2.2 Validation of a mathematical model

2.3 The Process of Numerical Solution

This section presents the general considerations that arise in the computer solution of a mathematical model. Once the mathematical model is given, the first thought typically is to try to find an explicit closed-form solution, but such a solution will usually only be possible for certain, perhaps radical, simplifications of the problem. These simplified problems with known solutions may be of great utility in providing "check cases" for the more general problem. After realizing that explicit solutions are not possible, one must turn to the task of developing a numerical method for the solution. Implicit in the thinking at the outset - and increasingly explicit as the development proceeds - will be the computing equipment as well as the software environment that is available. The approach may be quite different for a microcomputer or a cluster than for a supercomputer. Nevertheless certain general factors must be considered apart from the computer system to be used.

Perhaps the most important factor is that computers manipulate only a finite number of digits or characters. Because of this, normally, we cannot do arithmetic within the real number system as we do in pure mathematics. That is, the arithmetic done by a computer is restricted to finitely many digits, whereas the numerical representation of most real numbers requires infinitely many. Therefore *round-off errors* can affect the final computed result in different ways, from the possible accumulation of errors over a large number of operations to catastrophic cancellation. Catastrophic cancellation is one way in which an algorithm can be numerically unstable, although in exact arithmetic it may be a correct algorithm. Indeed, it is possible for the results of a computation to be completely erroneous because of round-off error even though only a small number of arithmetic operations have been performed.

Detailed round-off error analyses have now been completed for a number of the simpler and more basic algorithms such as those that occur in the solution of linear systems of equations. A particular type of analysis that has

proved to be very powerful is backward error analysis. In this approach the round-off errors are shown to have the same effect as that caused by changes in the original problem data. When this analysis is possible, it can be stated that the error in the solution caused by round off is no worse than that caused by certain errors in the original model. The question of errors in the solution is then equivalent to the study of the sensitivity of the solution to perturbations in the model. If the solution is highly sensitive, the problem is said to be ill-posed or ill-conditioned, and numerical solutions are apt to be meaningless.

Another way that the finiteness of computers manifests itself in causing errors in numerical computation is due to the need to replace "continuous" problems by "discrete" ones. This type of error is usually called *discretization error* or *truncation error*, and it affects, except in trivial cases, all numerical solutions of differential equations and other "continuous" problems.

There is one more type of error that is somewhat akin to discretization error. Many numerical methods are based on the idea of an iterative process. In such a process, a sequence of approximations to a solution is generated with the hope that the approximations will converge to the solution; in many cases mathematical proofs of the convergence can be given. However, only finitely many such approximations can ever be generated on a computer, and, therefore, we must necessarily stop short of mathematical convergence, i.e. having a *convergence error*.

If we rule out trivial problems that are of no interest in Scientific Computing, we can summarize the situation with respect to computational errors as follows. Every calculation will be subject to rounding error. Whenever the mathematical model of the problem is a differential equation or other "continuous" problem, there also will be discretization error, and in many cases, especially when the problem is nonlinear, there will be convergence error. These types of errors and methods of analyzing and controlling them need to discussed more fully in concrete situations, but for us it is important to keep in mind that an acceptable error is very much dependent on the particular problem. Rarely is very high accuracy, let's say 16 digits, needed in the final solution; indeed, for many problems arising in industry or other applications two or three digit accuracy is quite acceptable.

The other major consideration besides accuracy in the development of computer methods for the solution of mathematical models is *efficiency*. By this we will mean the amount of effort both human and computer required to solve a given problem. For most problems, such as solving a system of linear algebraic equations, there are a variety of possible methods, some going back many tens or even hundreds of years. Clearly, we would like to choose a method that minimizes the computing time yet retains suitable accuracy in the approximate solution. This turns out to be a surprisingly difficult problem, which involves a number of considerations. Although it is frequently possible to estimate the computing time of an algorithm by counting the required arithmetic operations, the amount of computation necessary to solve a problem

to a given tolerance is still an open question except in a few cases. Even if one ignores the effects of round-off error, surprisingly little is known. In the past several years these questions have spawned the subject of computational complexity. However, even if such theoretical results were known, they would still give only approximations to the actual computing time, which depends on a number of factors involving the computer system. And these factors change as the result of new systems and architectures. Indeed, the design and analysis of numerical algorithms should provide motivation and directions for such changes.

Even if a method is intrinsically "good", it is extremely important to implement the corresponding software in the best way possible, especially if other people are to use it. Some of the criteria for a good software system, besides functionality, are the following: maintainability, reliability, availability, robustness, efficiency, user friendliness, simplicity, readability, validity, verifiability, reusability, compatibility, portability, integrity, and, of course, a well-written documentation.

2.4 The Computational Environment

As indicated in the last section, there is usually a long way from a mathematical model to a successful software system. Such programs are developed within the overall computational environment, which includes the computers to be used, the operating system and other software systems, the languages in which the program is to be written, techniques and software for data management and visualization of the results, and programs that do symbolic computation. In addition, network facilities allow the use of remote computers, as well as the exchange of software and data.

The *computer hardware* itself is of primary importance. Scientific Computing is done on computers ranging from small PC's, which execute a few thousand floating-point operations per second, to supercomputers capable of billions of such operations per second. Supercomputers that utilize hardware vector instructions are called *vector computers*, while those that incorporate multiple processors are called *parallel computers*. In the latter case, the computer system may contain a few, usually very powerful processors or as many as several tens of thousands of relatively simple processors. Generally, algorithms designed for single processor "serial" computers will not be satisfactory, without modification, for parallel computers. Indeed, a very active area of research in Scientific Computing is the development of algorithms suitable for vector and parallel computers, and also for desktop grid or volunteer computing.

It is quite common to do program development on a workstation or PC prior to production runs on a larger computer. Unfortunately, a program will not always produce the same answers on two different machines due to different rounding errors. This, of course, will be the case if different precision

arithmetic is used. However, even when the precision is the same, two machines may produce slightly different results due to different conventions for handling rounding error. This is an unsatisfactory situation that has been addressed by the IEEE standard for floating point arithmetic. Although not all computers currently follow this standard, in the future they probably will, and then machines with the same precision will produce identical results on the same problem. On the other hand, algorithms for parallel computers often do the arithmetic operations in a different order than on a serial machine and this causes different errors to occur.

In order to be useful, computer hardware must be supplemented by *software systems*, including operating systems and compilers for high level languages. Although there are many operating systems, UNIX and its variants have increasingly become the standard for Scientific Computing and essentially all computer manufacturers now offer a version of UNIX for their machines. This is true for vector and parallel computers as well as more conventional ones. The use of a common operating system helps to make programs more portable. The same is true of programming languages. Since its inception in the mid 1950's, Fortran has been the primary programming language for Scientific Computing. It has been continually modified and extended over the years, and now versions of Fortran also exist for parallel and vector computers. Other languages, especially the systems language "C", are sometimes used for Scientific Computing. However, it is expected that Fortran will continue to evolve and be the standard for the foreseeable future, at least in part because of the large investment in existing Fortran programs.

Many of the problems in Scientific Computing require huge amounts of data, both input and output, as well as data generated during the course of the computation. The storing and retrieving of these data in an efficient manner is called *data management*. As an example of this in the area of computer-aided design, a database containing all information relevant to a particular design application, which might be for an aircraft, an automobile, or a dam - may contain several billion characters. An engineer may use this database simply to find all the materials with a certain property. On the other hand, the database will also be used in doing various analyses of the structural properties of the aircraft, which requires the solution of certain linear or nonlinear systems of equations. Large data management programs for use in business applications such as inventory control have been developed over many years, and some of the techniques used there are now being applied to the management of large databases for scientific computation. It is interesting to note that in many Scientific Computing programs the number of lines of code to handle data management is far larger than that for the actual computation.

The results of a scientific computation are numbers that may represent, for example, the solution of a differential equation at selected points. For large computations, such results may consist of the values of four or five functions at a million or more points. Such a volume of data cannot just be printed.

Scientific Visualization techniques allow the results of such computations to be represented pictorially. For example, the output of a fluid flow computation might be a movie, which depicts the flow as a function of time in either two or three dimensions. The results of a calculation of the temperature distribution in a solid might be a color-coded representation in which regions of high temperatures are red and regions of low temperatures are blue, with a gradation of hues between the extremes. Or, a design model may be rotated in three-dimensional space to allow views from any angle. Such visual representations allow a quick understanding of the computation, although more detailed analysis of selected tables of numerical results may be needed for certain purposes, such as error checking.

Another development that is having an increasing impact on Scientific Computing is *symbolic computation*. Systems such as MACSYMA, REDUCE, MAPLE, and MATHEMATICA allow the symbolic (as opposed to numerical) computation of derivatives, integrals and various algebraic quantities. For example, such systems can add, multiply and divide polynomials or rational expressions, differentiate expressions to obtain the same results that one would obtain using pencil and paper, and integrate expressions that have a "closed form" integral. This capability can alleviate the hard work of manipulating by hand lengthy algebraic expressions, perhaps as a prelude to a subsequent numerical computation. In this case, the output of the symbolic computation would ideally be a Fortran program. Symbolic computation systems can also solve certain mathematical problems, such as systems of linear equations, without rounding error. However, their use in this regard is limited since the size of the system must be small. In any case, symbolic computation is continuing to develop and can be expected to play an increasing role in scientific computation.

3 Scientific Visualization

This chapter starts with establishing the human nature of visualization, and then shifts to considerations on various sorts of computer-based visualizations: animations, interactive visualizations, abstract and model-based visualizations etc. Next, the main reasons for the need for Scientific Visualization are presented, along with the domains from which visualization has been bred, and the reasons for which visualization works. The chapter continues with detailed descriptions of some applications of visualization, from which the critical requirements of these applications arise. Then a few words about visualization algorithms, environments and graphical excellence guidelines are provided.

3.1 Visualization

The Merriam-Webster Collegiate Dictionary (Webster, 1998) gives two definitions for the term *visualization*:

1. the formation of mental visual images;
2. the act or process of interpreting in visual terms or of putting into visual form.

The Oxford Dictionary (Oxford, 2002) gives similar definitions for the same term *visualization*:

1. to form a mental vision, image, or picture of (something not visible or present to sight, or of an abstraction);
2. to make visible to the mind or imagination.

Visualization has its ancestry in pictorial representations dating back to the origins of man. Pictographs, for whatever reasons, are human generated images. Through the centuries, we have had maps, human generated imagery of different parts of the world for travel and warfare, paintings; imagery of plans for architectural and novel devices; images to enhance stories, and many more. Visualization is now part of our everyday life. From paintings and photographs to maps, television, and to computer generated graphics and virtual environments, we can see how it is used today in diverse ways. Some impressive examples can be seen in Figure 3.1 and Figure 3.2.

The use of visualization to present information is not a new phenomenon. It has been used in maps, scientific drawings, and data plots for over a thousand years. Examples from cartography include Ptolemy's Geographia (2nd Century AD), a map of China (1137 AD), and Minard's map (1861) of Napoleon's invasion of Russia half a century earlier. Most of the concepts learned in devising these images carry over in a straightforward manner to computer visualization. Edward Tufte has written two critically acclaimed books that explain many of these principles (Tufte, 1997), (Tufte, 2001).

Figure 3.1 Cave Painting in Lascaux

Figure 3.2 Kotsushika Hokusai - The Great Wave

Computer graphics has from its beginning been used to study scientific problems. However, in its early days the lack of graphics power often limited its usefulness. The recent emphasis on visualization started in 1987 with the special issue of Computer Graphics on Visualization in Scientific Computing. Since then there have been several conferences and workshops, co-sponsored by the IEEE Computer Society and ACM SIGGRAPH, devoted to the general topic, and special areas in the field (for example volume visualization).

Most people are familiar with the digital animations produced to present meteorological data during weather reports on television, though few can distinguish between those models of reality and the satellite photos that are also shown on such programs. TV also offers Scientific Visualizations when it shows computer drawn and animated reconstructions of road or airplane accidents. Some of the most popular examples of Scientific Visualizations are computer-generated images that show real spacecraft in action, out in the void far beyond Earth, or on other planets. Dynamic forms of visualization, such as educational animation, have the potential to enhance learning about systems that change over time.

Apart from the distinction between interactive visualizations and animation, the most useful categorization is probably between abstract and model-based Scientific Visualizations. The abstract visualizations show completely conceptual constructs in 2D or 3D. These generated shapes are completely arbitrary. The model-based visualizations either place overlays of data on real or digitally constructed images of reality, or they make a digital construction of a real object directly from the scientific data.

The success of visualization not only depends on the results, which it produces, but also depends on the environment in which it has to be done. This environment is determined by the available hardware, like graphical workstations, disk space, color printers, video editing hardware, and network bandwidth, and by the visualization software. For example, the graphical hardware imposes constraints on interactive speed of visualization and on the size of the data sets, which can be handled. Many different problems

encountered with visualization software must be taken into account. The user interface, programming model, data input, data output, data manipulation facilities, and other related items are all important. The way in which these items are implemented determines the convenience and effectiveness of the use of the software package as seen by the scientist. Furthermore, whether software supports distributive processing and computational steering must be taken into account.

In our context, we consider *visualization* to mean *a computer generated image or collection of images, possibly ordered, using a computer representation of data as its primary source and a human as its primary target* (McCormick et al., 1987). Computer generated data visualizations appeared in the late 40's when tables became much too large for a human to comprehend and manage. These visualizations, then called plots, were followed by the growth of computer graphics and systems that permitted the rapid, often interactive, generation of scientific data sets.

Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way scientists do science. Thus, a new definition has been added: *a tool or method for interpreting image data fed into a computer and for generating images from complex multi-dimensional data sets* (McCormick et al., 1987).

Visualization embraces both image understanding and image synthesis, that is, it is a tool both for interpreting image data fed into a computer and for generating images from complex multidimensional data sets. Visualization studies those mechanisms in humans and computers "which allow them in concert to perceive, use, and communicate visual information" (McCormick, 1988). It unifies the largely independent but converging fields of computer graphics, image processing, computer vision, computer-aided design, signal processing and user interfacing. An inspiring example can be seen in Figure 3.3.

The main reasons of need for Scientific Visualization are the following ones: it will compress a lot of data into one picture (data browsing), it can reveal correlations between different quantities both in space and time, it can furnish new space-like structures beside the ones which are already known from previous calculations, and it opens up the possibility to view the data selectively and interactively in `real time'. By following the formation and the deformation as well as the motions of these structures in time, one will gain insight into the complicated dynamics.

As was mention before, we also want to integrate our simulation codes into a visualization environment in order to analyze the data in real time and to by-pass the need to store every intermediate result for later analysis. This is possible by means of *processing* in which the simulation is distributed over a set of high-performance computers and the actual visualization is done on a

graphical distributive workstation. It is also very useful to have the possibility to interactively change the simulation parameters and immediately see the effect of this change through the new data. This process is called *computational steering* and it will increase the effective use of CPU time.

Figure 3.3 Great wave discretization

Why does visualization work at all? Because humans are inherently visual beings, with over half the brain being dedicated to visual information processing. The bandwidth of the human visual system is greater than any other sense, allowing humans to see and understand huge amounts of complex data quickly and accurately. By transforming data into pictures, visualization takes advantage of this enormous bandwidth and processing power of the human visual system. The data is not only processed faster, but also with a different strategy: instead of using conscious mechanisms (read something, translate it into a mental model, then understand the mental model), visual processing uses preconscious mechanisms, which are "hardwired, highly parallel processes that handle the initial stages of analysis of the retinal patterns" (Friedhoff and Peercy, 2000).

The brain areas involved in higher order visual perception and cognition are highly interlinked, such that the systems for seeing, understanding and remembering are closely associated. This means that in certain situations information can be more readily assimilated and communicated in a visual format than in any other form. For example, we are very good at recognizing objects, faces, and characters. Color information may be more appropriate for categorizing different objects or data types. With good luminance contrast we are very good in making fine spatial discriminations and to accurately determine shape, motion or depth.

How people perceive an image can have a profound effect on the meaning they attach to that image. Vision is also influenced by memory, context and intention. Many visualization methods are exploiting the human perceptual capacities and insensitivities. Since the appearance of the resulting

images as perceived by the human observer is of primary importance, the computation can be focused on those image features that can be readily perceived. This can lead to simplification of computation by omitting from the image details that will not make significant differences in its appearance.

3.2 Scientific Visualization

Today we are presented with a broader context within data visualization fits. It encompasses scientific visualization, information visualization, database visualization, software visualization and all the specific visualizations (including biomedical and geospatial visualizations). Visualization of Scientific Data, or Scientific Visualization *describes the application of graphical methods to enhance interpretation and meaning of scientific data, representation of data graphically as a means of gaining understanding and insight into the data*. This allows the researcher to achieve insight into the system that is studied in ways previously impossible. We will abbreviate Scientific Visualization to simply Visualization throughout this work. Scientific data can be derived from various sources, including measuring instruments, or may be obtained as a result of scientific computations performed on large computers. However, data do not become useful until some (or all) of the information they carry is extracted.

The goal of Scientific Visualization is to provide concepts, methods and tools to create expressive and effective visual representations from scientific data. Such visual representations will convey new insights and an improved understanding of physical processes, mathematical concepts and other quantifiable phenomena expressed in the data (Magnenat-Thalmann and Thalmann, 1991). Together with quantitative analysis of data, such as offered by statistical analysis, image and signal processing, visualization attempts to explore all information inherent in scientific data in the most effective way. Therefore, Scientific Visualization is expected to enhance and increase scientific productivity. The discipline of *Visualization in Scientific Computing* is widely recognized to have begun in the 1980s. Its birth marked by the production of a key report for the US National Science Foundation (NSF). Interest in visualization was stimulated by the happy coincidence of a number of factors. Workstations had become powerful enough to display graphics on the scientist's desktop and algorithmic developments were making the treatment of large datasets tractable. Crucially, supercomputers could now run simulations of complex phenomena and produce more data than could otherwise be assimilated. The NSF report argued that continuing slowly computer graphics provision was equivalent to a waste of these compute resources (Wright, 2007).

Scientific Visualization encompasses and unifies the fields of computer graphics, image processing, high performance computing, computer vision, signal processing, computer aided design, and human-machine interaction. Visualization is a method of extracting meaningful information from complex or voluminous datasets through the use of interactive graphics and imaging. It

provides processes for steering the dataset and seeing the unseen, thereby enriching existing scientific methods.

In this context, it is important to differentiate between Scientific Visualization and presentation graphics. Presentation graphics is primarily concerned with the communication of information and results in ways that are easily understood. In Scientific Visualization, we seek to understand the data. However, often the two methods are intertwined.

3.3 Applications of Visualization

Examples of the power of visualization to gain new insights into scientific data, to understand complex concepts, or to aid in the quest for information are plentiful. In this section we present some of the typical applications of visualization in various fields.

Computation is emerging between theory and experiment as a partner in scientific investigation. Computational science and engineering encompass a broad range of applications with one common denominator: *visualization*. Visualization tools are helping researchers understand and steer computations. The list of research opportunities for visualization in Scientific Computing is long and spans all of contemporary scientific endeavor.

The research opportunities actually described in this section represent a select sampling of advanced scientific and engineering applications. There are scientific opportunities for visualization in molecular modeling, medical imaging, brain structure and function, mathematics, geosciences, space exploration, astrophysics etc. *Engineering* opportunities for visualization consist of computational fluid dynamics and finite element analysis. Images and signals may be captured from cameras or sensors, transformed by image processing, and presented pictorially on hard or soft copy output. Abstractions of these visual representations can be transformed by computer vision to create symbolic representations in the form of symbols and structures. Using computer graphics, symbols or structures can be synthesized into visual representations.

Molecular Modeling. The use of interactive computer graphics to gain insight into chemical complexity began in 1964. Interactive graphics is now an integral part of academic and industrial research on molecular structures and interactions, and the methodology is being successfully combined with supercomputers to model complex systems such as proteins and DNA. Techniques range from simple black-and-white, bitmapped representations of small molecules for substructure searches and synthetic analyses, to the most sophisticated 3D color stereographic displays required for advanced work in genetic engineering and drug design.

The attitude of the research and development community toward molecular modeling has changed. What used to be viewed as a sophisticated and expensive way to make pretty pictures for publication is now seen as a

valuable tool for the analysis and design of experiments. Molecular graphics complements crystallography, sequencing, chromatography, mass spectrometry, magnetic resonance and the other tools of the experimentalist, and is an experimental tool in its own right. The pharmaceutical industry, especially in the new and flourishing fields of genetic and protein engineering, is increasingly using molecular modeling to design modifications to known drugs, and to propose new therapeutic agents.

Molecular modeling supports three general activities: synthesis, analysis and communication. Interactive 3D images are essential to each of these areas, to give scientists control of their data and access to information. Synthesis lets scientists integrate information interactively in real time. The computer is used to build or extend existing models by combining information and knowledge from a variety of sources. Molecular fragments pieced together from a chemical fragment database or a protein structure fitted into a 3D electron density map typifies this modeling activity. Analysis enables scientists to interpret and evaluate data by selectively displaying experimental and/or computational results in a comprehensible framework. The display and comparison of any number of macromolecular properties, such as chemical composition, connectivity, molecular shape, electrostatic properties, or mobility characteristics, all fall into the domain of modeling analysis. As more structures become available for examination, and as more techniques are developed for analysis, new patterns will emerge. This activity then feeds back to the synthesis activities and new models for the next level of biomolecular organization can be constructed. Communication takes place between computer and scientist, and between scientist and scientist. It is important that the information discovered about biological molecules be conveyed not only to the structural scientist, but also to a larger body of scientists whose expertise can add data and knowledge to increase overall understanding. Communication can also bridge the gap between science and the general public, making individuals aware of significant discoveries.

In an effort to make major inroads in these areas, scientists need access to more powerful visualization hardware and software. Effort must be expended on imaginative uses of graphics devices as windows on the microscopic world of the molecule, as well as on integrating the complex numeric and symbolic calculations used to simulate this world. There are currently two types of images one can generate: realistic pictures of molecules (simulations that resemble plastic models), and 3D line drawings (informative images that can be manipulated in real time). Raster equipment is used to create realistic-looking representations and animations, while vector hardware, used for real-time display and interaction, is used to create line drawings. As raster hardware improves, it is expected that raster and vector hardware will merge, allowing increased flexibility in the representations of chemical properties.

Tachistoscopic stereo has been in use for over 15 years and is rapidly gaining acceptance. The introduction of inexpensive liquid polarized screens

and polarized glasses will accelerate this important development. Interaction with the complex 3D world of the molecule is inhibited by the inherent 2D nature of many interactive input devices, such as the mouse. The ability to manipulate the 6 degrees of freedom of a molecule in space so it can interact with another is currently done using dual 3-axis joysticks or similar devices. Imaginative and inventive solutions are needed, such as magnetic motion monitors, to allow multiple interactions.

Medical Imaging. Scientific computation applied to medical imaging has created opportunities in diagnostic medicine, surgical planning for orthopedic prostheses, and radiation treatment planning. In each case, these opportunities have been brought about by 2D and 3D visualizations of portions of the body previously inaccessible to view.

In each of these applications, image processing dominates; both computer vision and computer graphics play a role in orthopedic prostheses and radiation treatment planning. The research activity is experimental, depending on volume-filled images reconstructed from measured data. The bottleneck in each of these examples is in the generation of useful 3D images, which requires further visualization research to increase spatial and temporal resolution. Useful 3D visualization algorithms, the development of powerful and portable visualization software, and relevant experimentation in visual psychophysics are all areas of visualization research.

Diagnostic medicine. The imaging modalities of computed transmission and emission tomography, magnetic resonance imaging and ultrasound, enhanced at times by contrast agents or monoclonal antibodies, are leading to a new understanding of both clinical and research questions in diagnosis. Improved 3D visualization techniques are essential for the comprehension of complex spatial and, in some cases, temporal relationships between anatomical features both within and across imaging modalities. Computation will play an increasingly central role in diagnostic medicine as information is integrated from multiple images and modalities. Visualization, the cornerstone of diagnostic radiology, must be smoothly combined with computation to yield natural and accurate images that a diagnostician can understand in 3D.

Orthopedic prostheses. An emerging visualization application is the fitting of prostheses to individuals for orthopedic reconstructions, such as hip replacements. The 3D fit must be precisely individualized to minimize rejection. Only through non-invasive 3D imaging can accurate specifications be obtained, so that a custom hip replacement can be fabricated in advance of a surgical procedure.

Radiation treatment planning. The use of ionizing radiation to destroy or inhibit the growth of malignant tumors requires careful planning. Misapplication of the radiation beam can jeopardize nearby normal tissue or render therapy ineffective. The precision required for safe but effective treatment is surprisingly high. Fortunately, recent computational advances have made practical the extensive computations that are essential to predict

radiation dosage accurately. Medical imaging allows these predictions to be based on a patient's own anatomy. Before the radiation treatment can be confidently applied, however, an effective means of visualizing the treatment dosage in relation to the tumor and neighboring normal tissue must be developed.

Brain structure and function. Visualization in 3D of human brain structure and function is a research frontier of far-reaching importance. The complexity of the brain limits understanding gained from the purely reductionistic approach familiar to neurobiologists. For continued progress in brain research, it will be necessary to integrate structural and functional information at many levels of abstraction. Work on brain structure and function requires computational support in four areas:

- acquisition of experimental data in digital image form from serial histological sections, medical imaging instruments, drug receptor studies and neurophysiological experiments;
- extraction of features from measured digital images to produce a 3D map of brain structure and function;
- analysis of the abstract brain map to relate measured ~mages and parameters to a standard brain geometry, to provide statistical summaries across a series of brains and to compare an individual brain with such statistical summaries.
- visualization of the results of data acquisition, feature extraction and map analysis in a proper 3D context.

The massive data input needed to map the brain will eventually lead to the invention or use of more advanced storage technologies. However, there is also a gap in our present ability to do automatic feature extraction. Manual recognition of features is inadequate for so large a problem. Automation will require major advances in the field of volume image abstraction and modeling. While the accomplishment of these tasks remains a monumental challenge, selected pilot studies promise near-term resolution of several pivotal issues. Brain mapping is a necessary first step toward modeling and simulating biological brain functions at a systems level of description.

3D brain visualization. Large-scale volume memory modules capable of storing 1 gigabyte of volume data are under design to model the brain from serial histological sections. Renderings are then constructed from the 1024x1024x1024 array of voxels to compare the spatial distribution of neurons and brain structures within one brain or, at increased resolution, within one portion of the brain. Image processing can create complex representations with brain parts made transparent, translucent or opaque. The volume memory module will aid in the use and evaluation of spatial filtering and boundary detection techniques in 3D brain imagery.

3D image understanding. A neuroanatomist locates objects in a brain section by matching the contour and regional data with a memorized model drawn from a visual knowledge base, which can include references to atlases. An image understanding system provides similar computer-based guidance for the semi-automated image analysis task – e.g. at the stage of data input, an individual can first identify a feature in one section and then let the system track the feature in succeeding serial sections, label the sectional data and store the information about the feature in a hierarchical database. Mapping of brain interconnections in parallel throughout the brain, at least at the level of nerve fiber tracts, remains a shortcoming in our mapping technology.

Brain mapping factory. Useful statistical experiments will eventually require analysis of 50-100 brains or major brain portions. The magnitude of this task is such that special automated facilities, called brain mapping factories, will inevitably be required to map one brain per month. By centralizing the volume image analysis it becomes possible to integrate and standardize scanning instrumentation, stain technology, image analysis software, geometric modeling and other brain mapping technologies, and to exploit economies of scale. Subsequent brain analysis could take place at distributed workstations with enhanced tools for image analysis.

Mathematics. In the computational study of partial differential equations associated with gas dynamics, vortex formation, combustion and fluid flow, the most effective means of analyzing output has been visualization. Using modern supercomputers, novel parallel architectures and new mathematical algorithms, important 3D physical processes can now be simulated. Geometric problems, such as the generation of body centered coordinates, automatic mesh generation, and so on, are mathematical in nature and will have to be solved efficiently to conduct research in applied mathematics.

Visualization is making a tremendous impact in the mathematical study of optimal form and more generally the calculus of variations, including the theory of minimal surfaces, and surfaces of constant mean curvature. Computer graphics has become an essential research tool in this and many other areas of pure and applied mathematics. Hard problems, such as eigen value optimization for regions with partially-free boundaries, are being attacked successfully for the first time with these visualization tools. Mathematics is one of the last sciences to become computerized; yet, it is already clear that visualization, coupled with very high-speed numerical simulations, is having a major influence in the field, even in areas long considered to be abstract. This new mode of investigation makes collaboration with scientists in other disciplines much easier for the mathematician; there is a common language of computation and images. There is a strong need to increase the availability and power of visualization tools for researchers within this discipline.

Computer Simulation. Science and engineering have undergone a major transformation at the research level as well as at the development and technology level. The modern scientist and engineer spend more and more time

in front of a laptop, a workstation, or a parallel supercomputer and less and less time in the physical laboratory or in the workshop. The virtual wind tunnel and the virtual biology laboratory are not a thing of the future: they are already here. The old approach of "cut and try" has been replaced by "simulate and analyze" in several key technological areas such as aerospace applications, synthesis of new materials, design of new drugs, and chip processing and microfabrication. The new discipline of nanotechnology will be based primarily on large-scale computations and numerical experiments. The methods of scientific analysis and engineering design are changing continuously, affecting both our approach to the phenomena that we study as well as the range of applications that we address. Whereas there is an abundance of software available to be used as almost a "black box", working in new application areas requires good knowledge of fundamentals and mastering of effective new tools.

In the classical scientific approach, the physical system is first simplified and set in a form that suggests what type of phenomena and processes may be important and, correspondingly, what experiments are to be conducted. In the absence of any known type of governing equations, dimensional inter dependence between physical parameters can guide laboratory experiments in identifying key parametric studies. The database produced in the laboratory is then used to construct a simplified "engineering" model that, after field-test validation, will be used in other areas of research, product development, and design and possibly lead to now technological applications. This approach has been used almost invariably in every scientific discipline, from engineering and physics to chemistry and biology.

The simulation approach follows a parallel path but with some significant differences. First, the phase of the physical model analysis is more elaborate: the physical system is cast in a form governed by a set of partial differential equations, which represent continuum approximations to microscopic models. Such approximations are not possible for all systems, and sometimes the microscopic model should be used directly. Second, the laboratory experiment is replaced by simulation, that is, by a numerical experiment based on a discrete model. Such a model may represent a discrete approximation of the continuum partial differential equations, or it may simply represent a statistical representation of the microscopic model.

Finite difference approximations on a grid are examples of the first case, and Monte Carlo methods are examples of the second case. In either case, these algorithms have to he converted to software using an appropriate computer language, debugged, and run on a workstation, parallel super-computer or a grid platform. The output is usually a large number of files of a few megabytes to hundreds of gigabytes, being especially large for simulations of time-dependent phenomena. To be useful, this numerical database needs to be put into graphical form using various visualization tools, which may not always be suited for the particular application considered. Visualization can be especially useful during simulations where interactivity is required as the grid may be

changing or the number of molecules may be increasing. The majority of researchers have already followed the simulation approach across disciplines in the past few decades.

Let us reexamine some of the requirements following the various steps in the simulation approach. The first task is to select the right representation of the physical system by making consistent assumptions to derive the governing equations and the associated boundary conditions. The conservation laws should be satisfied, the entropy condition should not be violated, and the uncertainty principle should be honored.

The second task is to develop the right algorithmic procedure to discretize the continuum model or represent the dynamics of the atomistic model. The choices are many, but which algorithm is the most accurate one, or the simplest one, or the most efficient one? These algorithms do not belong to a discipline! Finite elements, first developed by the famous mathematician Richard Courant and rediscovered by civil engineers, have found their way into every engineering discipline as well as into physics, geology, and other fields. Chemists, biologists, material scientists, and others practice molecular dynamics simulations.

The third task is to compute efficiently in the ever-changing world of supercomputing. How efficient the computation is translates to how realistic of a problem is solved and therefore how useful the results can be to applications. The fourth task is to assess the accuracy of the results in cases where no direct confirmation from physical experiments is possible, such as in nanotechnology, in bio-systems or in astrophysics. Reliability of the predicted numerical answer is an important issue in the simulation approach because some of the answers may lead to new physics or false physics contained in the discrete model or induced by the algorithm but not derived from the physical problem. Finally, visualizing the simulated phenomenon, in most cases in three-dimensional space and in time, by employing proper computer visualization completes the full simulation cycle the rest of the steps followed are similar to those of the classical scientific approach.

3.4 Algorithms for Scientific Visualization

In this section we look at basic algorithms for Scientific Visualization. In practice, a typical algorithm can be thought of as a transformation from one data form into another. These operations may also change the dimensionality of the data. For example, generating a streamline from a specification of a starting point in an input 3D dataset produces a one-dimensional curve. The input may be represented as a finite element mesh, while the output may be represented as a polyline. Such operations are typical of Scientific Visualization systems that repeatedly transform data into different forms and ultimately transform it into a representation that can be rendered by the computer system. The algorithms that transform data are the heart of data visualization.

To describe the various transformations available, algorithms need to be categorized according to the structure and type of transformation. Structure refers to the effects that transformation has on the topology and geometry of the dataset, and type means the type of dataset that the algorithm operates on. Structural transformations can he classified in four ways, depending on how they affect the geometry, topology, and attributes of a dataset. Here, we consider the topology of the dataset as the relationship of discrete data samples (one to another) that are invariant with respect to geometric transformation. For example, a regular, axis-aligned sampling of data in three dimensions is referred to as a volume, and its topology is a rectangular (structured) lattice with clearly defined neighborhood voxels and samples.

On the other hand, the topology of a finite element mesh is represented by a (unstructured) list of elements, each defined by an ordered list of points. Geometry is a specification of the topology in space (typically 3D), including point coordinates and interpolation functions. Attributes are data associated with the topology and/or geometry of the dataset, such as temperature, pressure, or velocity. Attributes are typically categorized as being scalar (single value per sample), vectors (n-vector of values), tensor (matrix), surface normals, texture coordinates, or general field data.

Given these terms, the following transformations are typical of Scientific Visualization systems (Hansen and Johnson, 2005):

- *Geometric transformations* alter input geometry but do not change the topology of the dataset. For example, if we translate, rotate, and/or scale the points of a polygonal dataset, the topology does not change, but the point coordinates, and therefore the geometry, do change;
- *Topological transformations* alter input topology but do not change geometry and attribute data. Converting a dataset type from polygonal to unstructured grid, or from image to unstructured grid, changes the topology but not the geometry. More often, however, the geometry changes whenever the topology does, so topological transformation is uncommon;
- *Attribute transformations* convert data attributes from one form to another, or create new attributes from the input data. The structure of the dataset remains unaffected. Computing vector magnitude and creating scalars based on elevation are data attribute transformations;
- *Combined transformations* change both dataset structure and attribute data. For example, computing contour lines or surfaces is a combined transformation.

We also may classify algorithms according to the type of data they operate on. The meaning of the word "type" is often somewhat vague. Typically, "type" means the type of attribute data, such as scalars or vectors. These categories include the following (Hansen and Johnson, 2005):

- *Scalar algorithms* that operate on scalar data. An example is the generation of contour lines of temperature on a weather map;
- *Vector algorithms* that operate on vector data. Showing oriented arrows of airflow (direction and magnitude) is an example of vector visualization;
- *Tensor algorithms* operate on tensor matrices. One example of a tensor algorithm is to show the components of stress or strain in a material using oriented icons;
- *Modeling algorithms* generate dataset topology or geometry, or surface normals or texture data. "Modeling algorithms" tends to be the catch-all category for algorithms that do not fit neatly into any single category mentioned above. For example, generating glyphs oriented according to the vector direction and then scaled according to the scalar value is a combined scalar/vector algorithm. For convenience, we classify such an algorithm as a modeling algorithm because it does not fit squarely into any other category.

Note that an alternative classification scheme is to refer to the topological type of the input data (e.g.. image, volume, or unstructured mesh) that a particular algorithm operates on. In the remainder of the chapter we will classify the type of the algorithm as the type of attribute data on which it operates. Though, we should be aware that alternative classification schemes do exist and may he better suited to describing the true nature of the algorithm.

Most algorithms can be implemented specifically for a particular data type or, mote generally, for treating any data type. The advantage of a specific algorithm is that it is usually faster than a comparable general algorithm. An implementation of a specific algorithm may also be more memory-efficient, and it may better reflect the relationship between the algorithm and the dataset type it operates on.

One example of this is contour surface creation. Algorithms for extracting contour surfaces were originally developed for volume data, mainly for medical applications. The regularity of volumes lends itself to efficient algorithms. However, the specialization of volume-based algorithms precludes their use for more general datasets such as structured or unstructured grids. Although the contour algorithms can be adapted to these other dataset types, they are less efficient than those for volume datasets. The presentation of algorithms in this section favors more general implementations. In some special cases, there exist performance-improving techniques for particular dataset types.

In a typical visualization system, algorithms are implemented as filters that operate on data. This approach is due in some part lo the success of early systems like the Application Visualization System and Data Explorer and the popularity of systems like SCIRun and the Visualization Toolkit that are built

around the abstraction of data flow. This abstraction is natural because of the transformative nature of visualization. The basic idea is that two types of objects (data objects and process objects) are connected together into visualization pipelines. The process objects, or filters, are the algorithms that operate on the data objects and in turn produce data objects as output. In this abstraction, filters that initiate the pipeline are referred to as sources and filters that terminate the pipeline are known as sinks (or mappers). Depending on their particular implementation, filters may have multiple inputs and/or may produce multiple outputs.

3.5 Visualization Environments

In this section we present briefly some visualization environments, which are distinguished by their cost, location and visualization technologies:

	Location	**Pros**	**Cons**	**Examples**
Supercomputer	machine room	very specialized	centralization	flight simulator
Minicomputer	laboratory high speed LAN	specific visualization	small support staff	Viz lab, CAVE
Distributed	laboratory	lower cost performance	small support staff	Viz lab, CAVE
Workstation	laboratory	decentralization	no support staff	high performance graphics desktop
Remote	desktop	very low cost	limited support staff	low performance desktop

3.6 Graphical excellence guidelines

According to Edward Tufte, "graphical excellence consists of complex ideas, situations, phenomenon etc. communicated with clarity, precision, and efficiency" (Tufte, 2001). Thus, graphical and visualization applications are supposed to do the following:

- show the data;
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production, or something else;
- avoid distorting what the data have to say;
- make large data sets coherent;
- encourage the eye to compare different pieces of data;
- reveal and distinguish the data at several levels of detail, from a broad overview to the fine structure;
- serve a reasonably clear purpose: description, exploration, and so on;
- be closely integrated with the meaning of the data set.

Graphical elegance is often found in simplicity of design and complexity of data. Design is choice. The theory of the visual display of quantitative information consist of principles that generate design options and that guide choices among options. The principles should not be applied rigidly or in a peevish spirit; they are not logically or mathematically certain; and it is better to violate any principle than to place graceless or inelegant marks on screen or paper. Most principles of design should be greeted with some skepticism, for word authority can dominate our vision, and we may come to see only through the lenses of word authority rather than with our own eyes.

What is to be sought in designs for the display of information is the clear portrayal of complexity. Not the complication of the simple; rather the task of the designer is to give visual access to the subtle and the difficult, that is the revelation of the complex.

4 Computational Grids and Desktop Grids

4.1 Distributed and Parallel Computing

Distributed computing arises as soon as one has to solve a problem in terms of processes that individually have only a partial knowledge of the several parameters associated with the problem. Thus, distributed computing appears both in computer world and in real world, and is at the heart of lots of applications. Whereas parallel computing is mainly concerned with efficiency, distributed computing addresses uncertainty generated by the multiplicity of control flows, the absence of shared memory and global time, and the occurrence of failures.

Distributed systems can be defined as computer systems that contain multiple processors connected by a communication network. The processors communicate with each other using messages that are sent over the network. Such systems are increasingly available due to decrease of prices of computer processors and the availability of high-bandwidth links to connect them. However, despite the availability of hardware for distributed systems, there are only few software applications that exploit that hardware. One important reason is that distributed software requires a different set of tools and techniques than that required by the traditional sequential software. Although distributed algorithms are often made up of only few lines, their behaviors can be difficult to understand and their properties hard to state and prove.

A distinction has to be made between distributed systems and parallel systems; the later ones consist of multiple processors that communicate with each other using shared memory. This distinction is only at a logical level. Given a physical system in which processors have shared memory, it is easy to simulate messages. Conversely, given a physical system in which processors are connected by a network, it is possible to simulate shared memory. Thus a parallel hardware system may run distributed software and vice versa. This distinction raises two important issues. One regards the question on which hardware to build: parallel or distributed. The other refers to the way we write applications, i.e. assuming shared memory or not. At the hardware level, we would expect that the prevalent model would be multiprocessor workstations connected by a network. Thus the system is both parallel and distributed. Further, two questions need to be addressed. The first one is "Why would the system not be completely parallel?" There are many reasons that follow below (Garg, 2002):

- *Scalability.* Distributed systems are inherently more scalable than parallel systems. In parallel systems shared memory becomes a bottleneck when the number of processors is increased;

- *Modularity and heterogeneity.* A distributed system is more flexible because a single processor can be added or deleted easily. Furthermore, this processor can be of a different type than the existing processors;
- *Data sharing*. Distributed systems provide data sharing as in distributed databases. Thus multiple organizations can share their data;
- *Resource sharing.* Distributed systems provide resource sharing – e.g. an expensive special purpose processor can be shared by organizations;
- *Geographical structure*. The geographical structure of an application may be inherently distributed. The low communication bandwidth may force local processing. This is especially true for wireless networks;
- *Reliability*: Distributed systems are more reliable than parallel systems because the failure of a single computer does not affect others;
- *Low cost*: Availability of high-bandwidth networks and inexpensive workstations also favors distributed computing for economic reasons.

The other essential question is "Why would the system not be purely a distributed one?" The reasons for keeping a parallel system at each node are mainly of a technological nature. With the current technology it is faster to update a shared memory location than to send a message to some other processor. This is especially true when the new value of the variable must be sent to multiple processors. Consequently, it is more efficient to get fine grain parallelism from a parallel system than from a distributed system (Garg, 1996).

So far the argumentation has been at the hardware level. Nevertheless, the interface provided to the programmer can actually be independent of the underlying hardware. So which model the programmer should better use? At the programming level, is expected that programs will be written using multithreaded distributed objects. In this model, an application consists of multiple heavyweight processes that communicate using messages (or remote method invocations). Each heavyweight process consists of multiple lightweight processes called *threads*. Threads communicate through the shared memory. This software model mirrors the hardware that is expected to be widely available. By assuming that there is at most one thread per process (or by ignoring the parallelism within one process), we get the usual model of a distributed system.

By restricting the focus to a single heavyweight process, we get the usual model of a parallel system. Though, the system will have aspects of distributed objects. The main reason is the logical simplicity of the distributed object model. A distributed program is more object-oriented because data in a remote object can only be accessed through an explicit message (or a remote procedure call). Conversely, threads are also useful to provide efficient objects. For many applications such as servers, it is useful to have a large shared data structure, because it is a programming burden to split the data structure across multiple heavyweight processes.

Summing up, aspects of both parallel processing and distributed processing will be seen both in hardware as well as software. To define the distributed systems we can consider the following features (Garg, 2002):

- *Absence of a shared clock* - in a distributed system, it is not possible to synchronize the clocks of different processors precisely because of uncertainty in communication delays between them. Consequently, it is rare to use physical clocks for synchronization. The concept of causality can be used instead of time to tackle this problem;
- *Absence of shared memory* - in a distributed system, it is impossible for any particular processor to know the global state of the system. Therefore it is difficult to observe any global property of the system. Though efficient algorithms can be developed for evaluating a suitably restricted set of global properties;
- *Absence of accurate failure detection* - in an asynchronous distributed system (a distributed system is asynchronous if there is no upper bound on the message communication time), the distinction between a slow processor and a failed processor cannot be done. This leads to many difficulties in developing algorithms for consensus, election, etc. Failure detectors can be built to alleviate some of these problems.

4.2 Computational Grids and Applications

4.2.1 A bit of Grid history

Similar to many momentous concepts and technologies that we now take for granted, Grid ideas have been inspired by, and were first applied to, problems faced by researchers tackling fundamental problems in science and engineering. Starting with ideas first expounded in the 1960s and given concrete form by Grid pioneers in the 1990s, the scientific community continues to lead the development of Grid technologies that will act as a computational and data management infrastructure that will be a key enabler for twenty-first-century science and society.

The origins of the idea of a "Grid" to support scientific research can be traced back to the Internet pioneer J. C. R. Licklider, who has began his career as an experimental psychologist studying psychoacoustics - how the human ear and brain convert air vibrations into the perception of sound. In the 1950s, he was a human factor researcher on the famous SAGE project at MIT: an air defense system designed to use real-time information on Soviet bombers. Coming from this experience, Licklider has written a groundbreaking paper in which he argued that computers should be developed to enable people and computers to cooperate in making decisions and controlling complex situations without inflexible dependence on predetermined programs (Waldrop, 2001).

Larry Roberts, the principal ARPANET architect, has recalled the importance of Licklider's ideas: *Lick had this concept of the intergalactic network*

which he believed was everybody could use computers anywhere and get at data anywhere in the world. He didn't envision the number of computers we have today by any means, but he had the same concept – all of the stuff linked together throughout the world, that you can use a remote computer, get data from a remote computer, or use lots of computers in your job. The vision was really Lick's originally. None of us can really claim to have seen that before him nor can anybody in the world. Lick saw this vision in the early sixties. He didn't have a clue how to build it. He didn't have any idea how to make this happen. But he knew it was important, so he sat down with me and really convinced me that it was important and convinced me into making it happen (Foster and Kesselman, 2004).

Since the beginnings of the ARPANET very much has changed. The Internet has become a reality, and e-mail and Web browsers have emerged as killer applications. Moore's law has prevailed for more than 30 years, with the result that computers are no longer rare, expensive resources. Nonetheless, Licklider's vision of a global network of computers and data resources that can be accessed seamlessly from anywhere in the world remains valid. The Grid is our latest and most promising attempt to realize Licklider's vision.

4.2.2 Need for Computational Grid in Context

Constant exponential technology improvements, new collaborative modalities enabled by the quasi-ubiquitous Internet, and the demands of increasingly complex problems have, over recent decades, fueled a revolution in the practice of science and engineering. Today's science is as much based on large-scale numerical simulation, data analysis, and collaboration as it is on the efforts of individual experimentalists and theorists. Further on we briefly review some of the new modes of inquiry that more and more define twenty-first-century science and engineering.

4.2.2.1 Data-Intensive Science

Impressive improvements in the capability and capacity of sensors, storage systems, computers, and networks are enabling the construction of data archives of mammoth size and value. Multipetabyte (10^{15} bytes) archives will soon be in place in fields as diverse as astronomy, biology, medicine, the environment, engineering, and high energy physics. Analysis of these vast quantities of data can yield profound new insights into the nature of matter, life, the environment, or other aspects of the physical world.

The sources of these huge quantities of data span a broad spectrum. At one extreme, we have individual, highly specialized, and expensive scientific devices that generate large quantities of data at a single location. For example, the worldwide particle physics community is planning an ambitious set of experiments at the Large Hadron Collider (LHC) experimental facility under construction at CERN in Geneva. The goal of this work is to find signs of the Higgs boson, key to the generation of mass for both the vector bosons and the fermions of the standard model of weak and electromagnetic interactions.

Particle physicists are also hoping for indications of other new types of matter - such as super symmetric particles - that may shed light on the "dark matter" problem of cosmology. These LHC experiments are on a scale never before seen in physics, with each experiment involving a collaboration of hundreds of institutions and over 5,000 physicists around the globe (LHC, 2007).

When operational in 2008, each of the LHC experiments will generate several petabytes of experimental data per year. This vast amount of data needs to be preprocessed and distributed for further analysis by all members of the consortia to search for signals betraying the presence of the Higgs boson or other revelations. The physicists need to put in place an LHC Grid infrastructure that will permit the transport and data mining of extremely large and distributed datasets.

The creation of this infrastructure is being pursued in collaboration with major Grid projects in the United States (NSF Grid Physics Network, DOE Particle Physics Data Grid, NSF International Virtual Data Grid Laboratory) and Europe (the EU DataGrid project and national Grid projects such as the UK Grid PP, Italian INFN Grid, and NorduGrid). The Importance of transoceanic bandwidth is recognized via the EU-funded DataTAG project for trans-Atlantic networks and the U.S.-funded STAR-TAP and StarLight international interconnection point (Foster and Kesselman, 2004).

Another significant source of immense quantities of data is the monitoring of industrial equipment. For example, pressure, temperature, and vibration sensors in each of the many thousands of Rolls-Royce engines currently in service generate about a gigabyte of data per engine on each trans-Atlantic flight, which translates to petabytes of data per year. The UK e-Science Distributed Aircraft Maintenance Environment project is working to aggregate these data so that they can be mined to detect indications of potential problems. The objective is to transmit a subset of the primary data for analysis and comparison with engine data stored in one of several data centers located around the world. By identifying the early problems, Rolls-Royce hopes to be able to lengthen the period between scheduled maintenance periods, thus increasing profitability. Decisions need to be taken in real time as to how much of the petabytes of data to analyze, how much to transmit for further analysis, and how much to get archived (Foster and Kesselman, 2004).

Similar (or even larger) data volumes are being generated by other high-throughput sensors in fields as varied as environmental and earth observation, astronomy, and human health-care monitoring. For example, in astronomy, individual "digital sky surveys" are creating data archives that will scale from a maximum of 10 terabytes today to petabytes within the next decade. It is estimated that the U.S. National Virtual Observatory project alone has stored 500 terabytes per year from 2004. Similarly, the Laser Interferometer Gravitational Observatory project is estimated to generate 250 terabytes per year beginning with 2002. A new generation of astronomical surveys such as the VISTA project in the visible and infrared regions will also contribute to the

transformation of the data requirements of the astronomy community. The VISTA telescope, which is operational since 2004 and will generate 250 gigabytes of raw data per night and around 10 terabytes of stored data per year. There will be several petabytes of data in the VISTA archive within 10 years (Foster and Kesselman, 1999).

Although these data volumes are impressive enough by themselves, what has astronomers really eager is the prospect of federating many such archives to create a uniformly accessible, globally distributed repository of astronomical data spanning all wavelengths, from radio waves to X-rays. The worldwide astronomy community is working to create such a globally distributed, multiwavelength "virtual observatory". For the time being, astronomical data using different wavelengths are captured by different telescopes and stored in a diversity of formats. The creation of such a multiwavelength "data warehouse" for astronomical data will enable new types of astrophysical studies.

Similar opportunities are arising in medicine, where all-digital scanning technologies allow CT scans, mammograms, MRI scans, and other medical images to be stored online rather than in film libraries. Multiterabyte databases being assembled within hospitals and research laboratories are making it far easier to compare images both across time for individuals and across populations. The linking of these databases with advanced analytical tools offers the potential for automated diagnosis in support of the individual physician, while the federation of multiple databases - potentially on a national or international scale - promises to enable epidemiological studies of unprecedented scope and scale that will provide new insights into the impact of environment and life cycle on disease.

The UK e-Diamond project is one project working to exploit these opportunities. Others include the Biomedical Informatics Research Network, the U.S. National Digital Mammography Archive, and the EU MammoGrid. e-Diamond brings together medical image analysis expertise from Mirada Solutions Ltd. and the MIAS Interdisciplinary Research Collaboration, computer science expertise from IBM and the Oxford e-Science Center, and clinical expertise from hospitals in London. Oxford, and Scotland. The goal is to provide an exemplar of the dynamic, best-evidence-based approach to diagnosis and treatment, which is made possible through a Grid middleware infrastructure.

The scope of the project is broad, and includes distributed data management and analysis, ontologies and metadata to describe both the physics that support the imaging process and the key features within images, as well as the capture of relevant demographic and clinical information. Technologies for data compression and data transfer that allow rapid data mining of the resulting large, federated databases of both metadata and images are also a key research area. Security and privacy are of paramount importance, and any grid infrastructure must be able to combine databases of information

based in hospitals protected by firewalls. The creation of such a large federated database of annotated, digitized, and standardized mammograms will provide for new applications in teaching and aiding both detection and diagnosis.

4.2.2.2 Simulation-Based Science

Numerical simulation represents another new problem-solving methodology in its own right, which continues to grow in importance The extensive use of supercomputers (an important class of "central power plant" in a scientific Grid) has been fundamental to scientific disciplines such as climatology and astrophysics, in which physical experiments cannot easily be performed but computational simulations are feasible. Indeed, supercomputers have emerged as an important class of "extreme scientific instrumentation." For example, the supercomputers can run numerical simulations of at a sustained rate of 400 teraflop/sec, and generating hundreds of terabytes of data in a single run. Other extremely capable systems are being operated by the U.S. ASCI program, DOE and NSF supercomputer centers in the United States, and supercomputer centers in Europe and elsewhere.

These tremendous investments in high-end supercomputers are just one indication of a broad phenomenon, which is that as a result of advances in computer performance and computational techniques, computational approaches are increasingly being applied even in fields long dominated by detriment. For example, in chemistry, combinatorial methods provide new opportunities for the generation, via computation rather than experiment, of large amounts of new chemical knowledge. The UK Comb-e-Chem project is illustrative of what is being done to exploit this opportunity. The goal of this project is to synthesize large numbers of new compounds by high-throughput combinatorial methods and then map their structure and properties. Such a parallel synthetic approach creates hundreds of thousands of new compounds at a time, leading to an explosive growth in the volume of data generated. Each new compound needs to he screened for potential usefulness, and properties and structure must be identified and recorded for promising candidates. Thus, an extensive range of primary data needs to be accumulated, integrated with information in existing databases, and enhanced with accurate models of the various relationships and properties. Comb-e-Chem is developing an integrated platform that combines existing structure and property data sources within a Grid-based information- and knowledge-sharing environment (Comb-e-Chem, 2005).

Similar transformations are occurring in the life sciences, as illustrated by the U.S. Encyclopedia of Life (EOL) project, which seeks to produce a database of reputed functional and 3D structure assignments for all known publicly available complete or partial genomes. Considerable computational capacity is required to update data as new genome sequences become available. This computation converts the rather sparse information contained in the linear

sequence of DNA bases into human-readable information that can be inferred by conversion to the amino acid sequence. Each genome must be subjected to a computation that is built up from loosely structured workflow, with analysis performed by a collection of algorithms that build on information in the EOL database.

4.2.2.3 Remote Access to Experimental Apparatus

The increasing prominence of simulation- and data-driven science does not mean that experimental science has become less important. On the contrary, the advance of technology is also producing revolutionary new experimental apparatus, and the emergence of high-speed networks makes it feasible to integrate those apparatus into the scientific problem-solving process in ways not previously imaginable.

Thus, for instance, we see the earthquake engineering community deploying telepresence capabilities that allow remote participants to design, execute, and monitor experiments without traveling to experimental facilities. The National Science Foundation's George E. Brown Jr. Network for Earthquake Engineering Simulation (NEES) is an ambitious national program whose purpose is to advance the study of earthquake engineering and to find new ways to reduce the hazard earthquakes represent to life and property. Its goal is to encourage the use of both physical and numerical simulation to develop increasingly complex, comprehensive, and accurate models of how the built infrastructure responds to earthquake loadings. NEESgrid is integrating and deploying Grid technologies to link earthquake engineering researchers across the United States with shared engineering research equipment, data resources, and leading-edge computing resources.

The NEESgrid middleware infrastructure allows collaborative teams (including remote participants) to plan, perform experiments, and publish and share their data and results. Collaborative tools assist experiment planning and allow engineers at remote sites to perform teleobservation and teleoperation of experiments, and enable access to computational resources and open source analytical tools for simulation and analysis of experimental data. The middleware also supports the publishing of results in a curated data repository using standard data and metadata vocabularies and formats.

Similar technologies have been applied successfully for some time to the remote operation of specialized scientific instrumentation. Grid technologies introduce the possibility of making these specialized usage scenarios routine.

4.2.2.4 Virtual Community Science

Besides the new modes of inquiry enabled by flexible and pervasive access to massive amounts of data, large amounts of computation, and specialized experimental apparatus, equally significant to 21st century's science and engineering is the increasingly collaborative and distributed nature of the teams. Moreover, the collaborative nature of science is in many respects

inseparable from the new capabilities. The most significant impact of grid technologies on science will probably be global virtual communities of scientists able to address the fundamental problems of today and tomorrow. Hopefully, this will hold also for virtual communities of non-scholar users that will turn to grid applications or opportunities to solve some of their day-to-day problems.

In data-driven science, the high scientific value of large data archives means that they are, increasingly, viewed as major strategic assets by their user communities, who devote considerable effort to establishing, managing, controlling, and exploiting those archives. Similar to the Encyclopedia of Life project mentioned previously, the UK eScience myGrid project is working to design, develop, and demonstrate higher-level grid middleware to support the use of complex distributed resources for bioinformatics, with particular applications being the analysis of functional genomic data and the annotation of a pattern database. The myGrid project is developing an e-Scientist's workbench to support experimental investigation, evidence accumulation, and result assimilation. The goal is to help scientists use community information (e.g., "gray literature") and enhance scientific collaboration by assisting the formation of dynamic groupings to tackle emergent research problem (myGrid, 2007)

Personalization facilities relating to resource selection, data management, and performing of processes provide for the dynamic creation of personal datasets and personal views over repositories, as well as both the addition of personal annotations to datasets and a personalized notification service about changes in relevant databases. The myGrid project is also developing tools and techniques to support the creation of personalized workflows that capture the biologist's know-how and enable reuse of patterns of knowledge discovery. These services can also associate base resources with the derived data, an aspect of the important area of provenance. The project is developing mechanisms to track the creation of knowledge, to automate the association of metadata with the production of primary experimental data, and to develop ontologies that facilitate automated reasoning about information from different communities.

Similar observations can be made about large-scale experimental and simulation science, as increasingly large teams devote considerable effort to establish and operate diverse apparatus, such as particle accelerators and climate simulation codes. For example, both the fusion and high-energy physics communities are planning future experimental facilities of unprecedented international scope and scale, and featuring “distributed control rooms" that will allow control of long-running experiments to be passed from one time zone to another over the course of a day.

The Comb-e-Chem project presented previously illustrates some of these issues. As much attention is given to the needs of the end-user community as to basic computational issues. Thus, work on the collection of new data, addresses support for both process and product data, and integrates electronic laboratory and e-logbook facilities. Also, interfaces are being developed to provide the user community with a unified view of resources and transparent access to data

retrieval, online modeling, and experiment design tools. The Comb-e-Chem service-based infrastructure extends to devices in the laboratory as well as to databases and computational resources. An important component of the project is the support of remote users of the UK National Criystallographic Service, physically located in Southampton. The service extends both to the portal access to the apparatus and to the support of resulting workflow. This scientific workflow corresponds to the sequence of linked operations necessary to get the desired result (use of the X-ray e-Laboratory), access to structures' databases, and admission to computing facilities for simulation and analysis in a specified sequence of operations. The goal is to provide shared, secure access to all of these resources in a supportive collaborative e-science environment.

The EOL project has similar goals. Information produced by EOL software is stored in a data warehouse and offered to the public through the EOL notebook, which accesses subservient, high-performance MySQL data marts. The EOL notebook portal provides users with data-mining capability that allows extensive, distributed data analyses. Users can gather information with regard to protein function over a wide variety of species and then run complex analysis applications on the combined dataset. An example could be an analysis of variations in structure and function over the evolutionary history of organisms. Such applications require high rates of data transfer and access to a large amount of computation, and thus EOL is both data- and compute-intensive. Because of the sheer number of current and future genomes available and the need for constantly up-to-date and synthesized information, EOL represents a growing class of applications for which a global grid infrastructure could be critical to enabling new advances in biology.

4.2.2.5 Scenarios for grid use in the real-world

Further on some grid use scenarios will be presented in order to emphasize that the need for grid systems go beyond the scientific world, to the people and their daily problems (Foster and Kesselman, 2004).

Scenario 1. A holding that want to reach a decision on the placement of a new industrial unit invokes a sophisticated financial forecasting model from an Application Service Provider (ASP), providing it with access to appropriate proprietary historical data from a corporate database on storage systems operated by a storage service provider. During the decision-making meeting, what-if scenarios are run collaboratively and interactively, even though the division heads participating in the decision are located in different locations. The ASP itself contracts with an on-demand cycle provider for additional "power" during particularly demanding scenarios, requiring of course that cycles meet desired security and performance requirements.

Scenario 2. An industrial consortium formed to develop a feasibility study for a next-generation supersonic spacecraft undertakes a highly accurate multidisciplinary simulation of the entire spacecraft. This simulation integrates

proprietary software components developed by different participants, with each component operating on that participant's computers and having access to appropriate design databases and other data made available to the consortium by its members.

Scenario 3. A crisis management team responds to a toxic waste accident by using local weather and soil models to estimate the spread of the waste, determining the impact based on population location and geographic features such as rivers and water supplies, creating a short-term mitigation plan (based on chemical reaction models), and tasking emergency response personnel by planning and coordinating evacuation, notifying hospitals, and so on.

Scenario 4. A large-scale Internet game consists of many virtual worlds, each with its own physical laws and consequences. Each world may have a large number of inhabitants that interact with one another and move from one world to another. Each virtual world may expand in an on-demand basis to accommodate population growth, new simulation technology to model the physical laws of the world will need to be added, and simulations need to be coupled to determine what happens when worlds collide.

These scenarios differ in many aspects: the number and type of participants, the types of activities, the duration and scale of the interaction, and the resources being pooled. However, they also have much in common. In each case, the participants who have varying degrees of prior relationship (perhaps none at all) want to share resources in order to perform some real-world complex problem within a powerful infrastructure.

4.3 Premises for Computational Grids

Computational approaches to solve various problems have proven their worth in almost every field of human endeavor. Computers are used for modeling and simulating complex scientific and engineering problems, diagnosing medical conditions, controlling industrial equipment, forecasting the weather, managing stock portfolios, and so on. However, although there are certainly challenging problems that exceed our ability to solve them, computers are still used much less extensively than they could be. For example, university researchers make extensive use of computers when studying the impact of changes in land use on biodiversity, but city planners selecting routes for new roads or planning new zoning ordinances do not. Nevertheless, it is local decisions such as these ones that, ultimately, shape our future.

4.3.1 Technical premises

There are a variety of reasons for this relative lack of use of computational problem-solving methods, including lack of appropriate education and tools. But one important factor is that the average computing environment remains inadequate for such computationally sophisticated goals. Though, the opportunity to provide users – whether city planners, engineers, or scientists –

with substantially more computational power: an increase of three orders of magnitude within five years, and five orders in a decade. These dramatic increases will be achieved by innovations in a wide range of areas (Foster and Kesselman, 1999):

- *technology improvement:* evolutionary changes in VLSI technology and microprocessor architecture can be expected to result in a factor of 10 increase in computational capabilities in the next five years, and a factor of 100 increase in the next ten;
- *increase in demand-driven access to computational power:* many applications have only episodic requirements for substantial computational resources. For instance, a medical diagnosis system may be run only when a CT scan is performed, a stock market simulation when a user re-computes some specific benefits, or a seismic simulation when an earthquake is studied. If mechanisms are available to allow reliable, instantaneous, and transparent access to high-end resources, then from the perspective of these applications it is as if those resources are dedicated to them. Given the existence of multiteraFLOPS systems, an increase in apparent computational power of three or more orders of magnitude is feasible;
- *increased utilization of idle capacity* - most low-end computers (workstations or PCs) are often idle, various studies reporting that around 30% of processor time is used in academic and commercial environment. Utilization can be doubled, even for parallel programs, without having a significant effect on productivity. The benefit to individual users can be substantially greater: factors of 100 or 1000 increase in peak computational capacity have been reported;
- *greater sharing of computational results* – the daily weather forecast involves probably 10^{14} numerical operations. If we assume that the forecast is of benefit to 10^7 people, we have 10^{21} effective operations – comparable to the computation performed each day on all the world's PCs. Few other computational results or facilities are shared so effectively today, but they may be in the future as other scientific communities adopt a "big science" approach to computation. The key for more sharing could be the development of "collaboratories" i.e. center(s) without walls, in which the researchers can do their research without regard to geographical location – interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries;
- *new problem-solving techniques and tools*: a variety of approaches can increase the efficiency with which computation is applied to problem solving. For instance, network-enabled solvers allow users to invoke advanced numerical solution methods without having to install sophisticated software. Teleimmersion techniques facilitate the

sharing of computational results by supporting collaborative steering of simulation and exploration of data sets.

Underlying each of these advances is the synergistic use of high-performance networking, computing, and advanced software to provide access to sophisticated computational capabilities, regardless of the location of both users and resources.

4.3.2 Financial premises

The new modes of inquiry and application scenarios presented in the preceding sections will transform the practice of science and engineering. Nevertheless, achieving these transformations requires major investments in physical infrastructure (petabyte archival storage, terabit networks, sensor networks, teraop supercomputers), software infrastructure (grid middleware, collaboratories), and new application concepts and software. Governments are realizing the importance of these investments its a means of enabling scientific progress and enhancing national competitiveness. To this end, major initiatives are under way worldwide, aimed variously at supporting major science and research grid projects, establishing and enhancing national grid resources and instruments, developing grid and middleware technologies, and/or coordinating and facilitating grid technologies and activities.

John Taylor, Director General of the United Kingdom's Office of Science and Technology, was an early proponent of this idea, coining in 1999 the term *e-science* to denote a new field of endeavor, writing that "e-science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it". He was also successful in obtaining significant funding to realize his concept. The first phase of the UK e-Science programme, launched in 2001 with a budget of £120 M over three years, has established projects spanning many areas of science and engineering. A key feature of the program is the active engagement of early adopters from industry: over 80 companies are contributing a total of £30 M in collaborative e-science projects. The industries represented range from IT, pharmaceutical, engineering, and petrochemical companies to financial modeling and media. These projects define middleware infrastructure requirements that far exceed the capability of present grid middleware. The UK e-Science Core Programme is tasked with identifying the elements of a generic grid middleware stack that will not only support UK science but also be of interest to industry. The UK program revived in 2003 a second investment of about £120 M for a further three years, till 2006 (eScience, 2007).

In the United Slates, a National Science Foundation (NSF) Blue-Ribbon Advisory Panel was convened in 2001 to inventory and explore advances in computational technology and to make strategic recommendations on the nature and form of programs that the NSF should take in response to converging technology trends. The Panel observed that digital computation,

data, information, and networks are now increasingly replacing and extending traditional methods of science and engineering research. In silico simulation and modeling at new levels of resolution and fidelity arc providing a complementary approach to scientific exploration to contrast with the traditional theoretical/analytical and experimental/observational modes. The Panel states that "a new age has dawned in scientific and engineering research, pushed by continuing progress in computing, information and communication technology, and pulled by the expanding complexity, scope, and scale of today's challenges" and concludes that new technologies have progressed to the extent that it is now possible to envisage creating a global "cyber infrastructure" on which new types of scientific and engineering knowledge environments and "virtual organizations' can be built The realization of such cyber infrastructure would allow research to be pursued in new ways and with increased efficiency. The blue-ribbon report recommends that NSF should lead a large ($1 billion per year), interagency and internationally coordinated Advanced Cyber infrastructure Program (ACP) to "create, deploy, and apply cyber infrastructure in ways that radically empower all scientific and engineering research and allied education.

The European Union's 6th and 7th Framework Programmes (FP6, FP7) also devote substantial sums to research infrastructure and grid computing, through their specific programmes, which aim to "to optimize the use and development of the best research infrastructures existing in Europe. Furthermore, it aims to help to create new research infrastructures of pan-European interest in all fields of science and technology. The European scientific community needs these to remain at the forefront of the advancement of research, and they will help industry to strengthen its base of knowledge and technological know how". In the case of FP7, the specific program is called "Research Infrastructure" and the budget is € 1.8 billion for funding this theme over the duration of FP7 (2007-2013) (FP7, 2007).

Within the scope of this European Community action, the term "research infrastructures" refers to facilities or resources that provide essential services to the scientific community for basic or applied research in all scientific and technological fields. Such research infrastructures may be "single-sited" or distributed (a network of resources). Including the associated human resources, this definition covers:

- major equipment or sets of instruments used for research purposes;
- knowledge-based resources such as collections, archives, structures information or systems related to data management, used in research;
- enabling Information and Communication Technology-based infrastructures such as Grid, computing, software and communications;
- any other entity of a unique nature that is used for scientific research.

The optimization, or emergence, of research infrastructures with a clear European dimension and added value in terms of performance and access will be considered for support. These infrastructures must contribute significantly to the development of European research capacities. The activities to be supported are identified under three main lines of action as described below:

1. *Optimizing the utilization of existing research infrastructures and improving their performance*. The objective is to strengthen European capacities and performance of specific research infrastructures, and increase user communities' involvement in opportunities offered by research infrastructures and their commitment towards investment in top-level research. This line of action represents the majority of the efforts (more than 60% of the operational funds) to be carried out under this part of the Specific Programme 'Capacities'. Support will be provided for integrating activities to structure better, on a European scale, the way research infrastructures operate in a given field, to foster their joint development in terms of capacity and performance and to promote their coherent and cross-disciplinary use. Emphasis should be given to the efficient and coordinated implementation of trans-national access and service activities, to ensure that European researchers, including researchers from industry and SMEs, may have access to the best research infrastructures to conduct their research, irrespective of their location. This action is both a bottom-up and a targeted approach:
   - bottom-up to respond to the needs of the scientific community in all fields of science and technology, without any preference for one field over another;
   - targeted to respond to the strategic research needs of the thematic priority areas and thereby strengthen the consistency of actions within the FP7.
2. *Strengthening research e-infrastructures by fostering further development and global connectivity of high-capacity and high-performance communication and grid-empowered infrastructures, and by reinforcing distributed supercomputing and data storage facilities.* The aim is to develop a new research environment, building upon the capabilities of GÉANT, the multi-gigabit pan-European data communications network reserved specifically for research and education and existing grid infrastructures, in which all scientists have easy-to-use, controlled access, regardless of their location in the world. It will be necessary to support, in a coordinated way, digital libraries, archives, data storage and curation activities and the essential pooling of resources at European level. Finally, the activities aim at fostering the adoption of e-infrastructures by user communities where appropriate, enhancing their global relevance and increasing the level of trust and confidence. The development of new research

infrastructures of pan-European interest, will build primarily on the work of the 'European strategy forum on research infrastructures' (ESFRI), which aims to promote the creation of new research infrastructures with a crucial and pan-European impact for the development of relevant scientific fields of science and technology, and to be able to help industry to strengthen its knowledge base and technological know-how. This action will also examine the opportunities for exploiting the potential of scientific excellence of the converging and outermost regions through new infrastructures. This line of action represents about one third of the total financial resources available for in this part of the Specific Programme 'Capacities'. Support will be provided for designs investigated for new research infrastructures that demonstrate a clear European dimension and interest, through a bottom-up approach of 'calls for proposals' and construction of critical new research infrastructures, building upon the work conducted by the ESFRI on the development of a European roadmap for new research infrastructures. This activity will follow a two-stage approach. The first phase will involve the preparation of the detailed construction plans, of the legal organization, of the management and multi-annual financial planning and the final agreement between stakeholders. In the second stage, the construction plans will be implemented with the possible involvement of private financial institutions, building on the achieved technical, legal and financial agreements.

3. *Support measures for policy development and programme implementation, including support for emerging needs.* Strong coordination within the EU in formulating and adopting a European policy on research infrastructures is the key to the success of this activity. Throughout the whole programme there will be measures to enhance the effectiveness and coherence of national and Community research policies and the development of international co-operation.

These activities will be carried out mainly following periodic 'calls for proposals' that aim to stimulate the coordination of national programmes through ERA-NET actions and support the work of ESFRI and 'e-infrastructure reflection group' (eIRG). In the context of international co-operation, they will also allow the identification of the needs of specific third countries and mutual interests on which specific co-operation actions could be based and, on the basis of targeted calls, the development of cross-links between key research infrastructures in third countries and those within the European Research Area (ERA) (FP7, 2007).

4.3.3 Experiencing premises

For the time being, there are some remarkable American and European experiences in development of grid computing solutions that will be presented further on. Similar initiatives are underway in Japan, Singapore, and China. Although active research on grid technology has been conducted among various countries, mainly in European countries and USA, research in Japan is rather behind compared with these countries. Though, in the meantime the interest of the people in Japan is rapidly getting higher, and it is commonly and basically recognized that it is necessary to accelerate the grid research (JPGRID, 2007).

Through developing corresponding grid middleware and cooperating with the application of Network Computer, ChinaGrid aims to integrate heterogeneous mass resources distributed in the China Education and Research Network (CERNET), shares those resources in the CERNET environment effectively and avoids the resource islands, provides useful services, finally forms the public platform for research and education in China (ChinaGRID, 2007).

gLite is the next generation lightweight middleware for grid computing. Born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers as part of the EGEE Project, gLite provides a framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet (gLite, 2008). gLite middleware is currently deployed on hundreds of sites as part of the EGEE project and enables global science in a number of disciplines, notably serving the LCG project (LCG, 2008). The EGEE project brings together experts from over 27 countries with the common aim of building on recent advances in Grid technology and developing a service Grid infrastructure which is available to scientists 24 hours-a-day. The project aims to provide researchers in academia and industry with access to major computing resources, independent of their geographic location. The EGEE project will also focus on attracting a wide range of new users to the Grid. The project primarily concentrates on three core areas (EGEE, 2008):

- The first area is to build a consistent, robust and secure Grid network that will attract additional computing resources.
- The second area is to continuously improve and maintain the middleware in order to deliver a reliable service to users.
- The third area is to attract new users from industry as well as science and ensure they receive the high standard of training and support they need.

The EGEE Grid is being built on the EU Research Network GÉANT and exploit Grid expertise generated by many EU, national and international Grid projects to date. Funded by the European Commission, the EGEE project community has been divided into 12 partner federations, consisting of over 70 contractors

and over 30 non-contacting participants covering a wide-range of both scientific and industrial applications.

The work being carried out in the project is organized into 11 activities. Two pilot application domains were selected to guide the implementation and certify the performance and functionality of the evolving infrastructure. One is the Large Hadron Collider Computing Grid supporting physics experiments and the other is Biomedical Grids, where several communities are facing equally daunting challenges to cope with the flood of bioinformatics and healthcare data. With funding of over 30 million Euro from the European Commission, the project is one of the largest of its kind. EGEE is a two-year project conceived as part of a four-year programme (2004-2008), where the results of the first two years will provide the basis for assessing subsequent objectives and funding needs.

NorduGrid/ARC is a Grid Research and Development collaboration aiming at development, maintenance and support of the free Grid middleware, known as the Advance Resource Connector (ARC). The NorduGrid collaborative activity is based on the success of the project known as the "Nordic Testbed for Wide Area Computing and Data Handling", and aims at continuation and development of its achievements. That project was launched in May 2001, aiming to build a Grid infrastructure suitable for production-level research tasks. The project developers came up with an original architecture and implementation, which allowed the testbed to be set up accordingly in May 2002, and remain in continuous operation and development since August 2002. The aim of the NorduGrid collaboration is to deliver a robust, scalable, portable and fully featured solution for a global computational and data Grid system. NorduGrid develops and deploys a set of tools and services – the so-called ARC middleware, which is a free software. The goals are (NorduGrid, 2008):

- Develop and support the ARC middleware.
- Coordinate contributions to the ARC code.
- Define strategical directions for development of the ARC middleware following latest tendencies in the Grid technologies.
- Promote ARC middleware solutions in such areas as Grid development, deployment and usage.
- Contribute to development of Grid standards, e.g. via GGF.

UNICORE (Uniform Interface to Computing Resources) offers a ready-to-run Grid system including client and server software. UNICORE makes distributed computing and data resources available in a seamless and secure way in intranets and the Internet. UNICORE has special characteristics that make it unique among Grid middleware systems. The UNICORE design is based on several guiding principles that serve as key objectives for further enhancements (UNICORE, 2008):

- *Abstraction*: UNICORE users need not know details about the system that they use. UNICORE provides abstractions for concepts such as application software and storage locations. Thus, UNICORE allows seamless access to heterogeneous environments;
- *Security*: UNICORE offers strong security based on industry standards such as the X.509 PKI. Communication over the Internet is protected by mutual authentication;
- *Site autonomy*: when making resources available on the Grid, administrators keep fine-grained control about their resources. Local policies are respected;
- *Ease of use*: A powerful GUI client covers the most common usage scenarios, e.g application execution and multi-step, multi-site workflows;
- *Ease of installation*: UNICORE is simple to install, with minimal external dependencies. Quick start bundles exist that allow getting up-and-running quickly.

The development of the UNICORE system was initiated in 1997 to enable German supercomputer centers to provide their users with a seamless, secure, and intuitive access to their heterogeneous computing resources. As in the case of the Globus Toolkit, UNICORE was started before “Grid computing” became the accepted new paradigm for distributed computing (Foster and Kesselman, 1999). The UNICORE vision was proposed to the German Ministry for Education and Research (BMBF) and received funding. A first prototype was developed in the "UNICORE" project. The foundations for the current production version were laid in the follow-up project "UNICORE Plus", which was successfully completed in 2002. In recent years, UNICORE has undergone a major restructuring and re-implementation of core components. This has been done in the European UniGrids project. Now, UNICORE is based on Web Services as proposed by the Open Grid Services Architecture maintained by the Open Grid Forum. In fact, UNICORE 6 is the most up-to-date implementation of the core specifications (such as WS-RF).

MiG - Minimum intrusion Grid is an attempt to design a new platform for Grid computing which is driven by a stand-alone approach to Grid, rather than integration with existing systems. The goal of the MiG project is to provide Grid infrastructure where the requirements on users and resources alike is as small as possible (minimum intrusion). MiG strives for minimum intrusion but will seek to provide a feature rich and dependable Grid solution (MiG, 2008).

MiG's main features (when fully implemented) will be as follows: minimum intrusion on user and resource, few dependencies on user and resource, scalable, autonomous - updating grid without user/resource software, anonymous - users and resources can't see identity of each other if desired, fault tolerance, load balancing, firewall compliant, strong scheduling (grid level), simple implementation, cooperative support (user-defined data-structures), and banking/accounting. MiG sandboxes make it easy to donate the spare

computing power in your computer to scientific research. Sandboxes provides a secure execution environment, which ensures that your computer is in no danger for being exposed for virus or other malware from the work it performs for the Grid. The sandbox also makes sure that both the personal files and other information cannot be seen by the Grid. Thus the name sandbox, the Grid really cannot see or access anything outside the sandbox. In MiG, there are 3 ways to contribute to the researchers work and still be fully protected by the sandboxes: One-Click, MiG-SSS, and MiG-PS3.

WebCom-G is a fledgling Grid Operating System, designed to provide independent service access through interoperability with existing middlewares (Morrison *et al.,* 2004). Metacomputing systems were developed to harness the power of geographically distributed computing resources. Such resources generally consisted of machines connected to intranets, the Internet and World Wide Web. WebCom separates the application and execution environments by providing both an execution platform, and a development platform. The independence provided by separating these two environments facilitates computation in heterogeneous environments. WebCom uses a server/client model for task distribution. Clients consist of Abstract Machines (AM's) that can be either pre-installed or downloaded dynamically form a WebCom server. AM's are uniquely comprised of both volunteers and conscripts.

Volunteers donate compute cycles by instantiating a web-based connection to a WebCom server and dynamically downloading the client abstract machine. These clients, constrained to run in the browsers' sandbox, will execute tasks on behalf of the server. Task communication is carried out over dedicated sockets. Pre-installed clients, also communicate over dedicated sockets. Upon receipt of a task representing a Condensed Graph (the task can be partitioned for further distributed execution), such clients are promoted to act as other WebCom servers. The returning of a result causes a promoted AM to be demoted, and act as a simple client once more. The execution platform consists of a network of dynamically managed machines, each running WebCom. WebCom can assume a traditional client server connection model or the more contemporary peer-to-peer model. WebCom sees each abstract machine as a "unit". Each unit contains a number of modules. The modules include an execution engine module and others for communication, load balancing, fault tolerance, scheduling and security. These modules are plugins to a backplane. Communications between WebCom units use a messaging system. Plugins can send messages between themselves on the local unit or to any plugin on other connected units. The default engine module is capable of executing Condensed Graphs applications.

By using WebCom, a whole grid can be viewed as a single WebCom unit with a specific computation engine plugin. This evolution of WebCom is the first step of producing a grid-enabled middleware: WebCom-G. Multiple grids are themselves viewed as independent WebCom-G units. When an instruction is sent to WebCom-G all the information is supplied to either dynamically

create or invoke an RSL script or to execute the job directly. When a WebCom-G unit receives an instruction it is passed to the grid engine module. This module unwraps the instruction, creates the RSL script and directs the gatekeeper to execute it. Once the Gatekeeper has completed execution, the result is passed back to the unit that generated the instruction. As WebCom-G uses the underlying grid architectures, failures are detected only at the higher level. In this case WebCom-G's fault tolerance will cause the complete job to be re scheduled. A WebCom compute engine plugin is implemented as a module that interacts with WebCom via a well-defined interface. This gives a platform independent grid compute engine. Although a Condensed Graph may be developed on a platform that is not grid enabled, the execution of grid operations will be targeted to grid-enabled platforms. The WebCom-G Operating System is proposed as a Grid Operating System. It is modular and constructed around a WebCom kernel, offering a rich suite of features to enable the Grid. It will utilize the tested benefits of the WebCom metacomputer and will leverage existing grid technologies such as Globus and MPI. The aim of the WebCom-G OS is to hide the low level details from the programmer while providing the benefits of distributed computing.

Office Grid relies on the fact that the concept of grids of office computers is especially desirable for many enterprises. One can look at a department or division in a corporation as a computational unit. The structure of today's businesses implies a heterogeneous setup and machine park, and centralized management. In addition the internal network of computers in an office is normally trusted and thought of as secure. The centralized management and the trusted status of the computers make the task of joining the power of the office computers more manageable, while the heterogeneous nature of the network is an obstacle. The vision of Office Grid is to allow for the constructive use of these hours of wasted time. OfficeGRID glues this pool of unused computational power together, thus making in possible to utilize the unused 75% (or more) computer time. OfficeGRID contains a system that allows the user to start jobs that run on many computers with a single OfficeGRID command and collect the results on his local computer, thereby giving him an easy way to use the grid of office computers as a supercomputer.

OfficeGRID can be run explicitly at any time or it can be limited to run when the screen saver on a given computer is running - that is, when the user is not using it. This gives the possibility to ensure that OfficeGRID will not bother the user, while still running more than 75% of the time. OfficeGRID BLAST is an OfficeGRID application, which has been developed by using the OfficeGRID Development Package. BLAST is an algorithm that is used to compare biological sequences, such as DNA sequences of different genes or the amino-acid sequence of different proteins. Several computer programs that implement the BLAST algorithm has been created, but generally these BLAST implementations are not built to run in parallel on several computational hosts. This is a problem because BLAST normally has a very long running time and

requires a lot of memory to run efficiently. OfficeGRID BLAST is a parallellization of NCBI BLAST, which is developed by the U.S. National Center for Biotechnology Information. There is both a Windows and a Linux version of the OfficeGRID Development Package available and complete interoperability of the two is possible. This makes it possible to develop OfficeGRID applications that will run on both the Windows and the Linux operating system as well as a mix of them. It could be used also for Mac OSX or Solaris (Office Grid, 2008).

The Globus Toolkit is an open source software toolkit used for building Grid systems and applications. The Globus Alliance and many others all over the world are developing it. A growing number of projects and companies are using the Globus Toolkit to unlock the potential of grids for their cause. *The Globus Alliance* is a community of organizations and individuals developing fundamental technologies behind the "Grid," which lets people share computing power, databases, instruments, and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The Globus Alliance is an active member in the community of Grid Software developers. As partners in e-Science and e-Business projects, they have built Grid Solutions for a variety of challenges that come up when people share resources. The open source Globus Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. In addition to being a central part of science and engineering projects that total nearly a half-billion dollars internationally, the Globus Toolkit is a substrate on which leading IT companies are building significant commercial Grid products.

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when. The Globus Toolkit has grown through an open-source strategy similar to the Linux OS, and distinct from proprietary attempts at resource-sharing software. This encourages broader, more rapid adoption and leads to greater technical innovation, as the open-source community provides continual enhancements to the product.

The Globus Alliance and the Globus Toolkit have enabled many exciting new scientific and business applications, as it is presented further on (Globus, 2007). Computational scientists at Brown University are using the Globus Toolkit and MPICH-G2 to simulate the flow of blood through human arteries. Globus Toolkit-driven Grid computing is central to management of large datasets generated by colliders such as those at CERN (LCH, 2007). The Southern California Earthquake Center uses Globus software to visualize earthquake simulation data. Scientists simulate earthquakes by calculating the effect of shock waves as they propagate through various layers of a geological model. SCEC simulations cover a very large space with very high resolution and can generate up to 40TB of data per simulation run. Scientists in the National Fusion Collaboratory are learning to use the Access Grid and Globus Web services to participate remotely in pulsed plasma fusion experiments. The remote interface provides sensor readings, data analysis, audio, and video available in the control room and allows the team to discuss what is happening. The Access Grid is integrated with Grid services and applications using the Globus Toolkit's security and communication libraries.

Physicists used the Globus Toolkit and MPICH-G2 to harness the power of multiple supercomputers to simulate the gravitational effects of black hole collisions. The team, which included researchers from Argonne National Laboratory, the University of Chicago, Northern Illinois University, and the Max Planck Institute for Gravitational Physics in Germany, was awarded a prestigious Gordon Bell prize for its work. Scientists in the Earth System Grid (ESG) are producing, archiving, and providing access to climate data that advances our understanding of global climate change. ESG uses Globus software for security, data movement, and system monitoring.

4.4 Computational Grid Definition

The current status of computation is analogous in some respects to that of electricity at its beginnings. At that early time (around 1910), electric power generation was possible, and new devices that depended on electric power were becoming available, but the need for each user to build and operate a new generator hindered use. The truly revolutionary achievement has not been, in fact, electricity, but the electric power grid and the associated transmission and distribution technologies. Together, these revolutionary developments provided reliable, low-cost access to a standardized service, with the result that power that, for most of the human history has been accessible only in non-portable forms (human effort, horses, steam engines, water power etc.), has become universally accessible. By permitting both individuals and industries to take for granted the accessibility of cheap, reliable power, the electric power grid has made possible both new devices and the new industries that manufactured them.

Analogously, the term of *computational grid* can be adopted for the infrastructure that will enable the increases of computation presented above. An early attempt to define a *computational grid* states that is *a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities* (Foster and Kesselman, 1999). *Infrastructure* is needed because the computational grid is concerned, above all, with large-scale pooling of resources, whether compute cycles, data, sensors and people. Such performant pooling requires significant hardware infrastructure to get hold of the necessary interconnections and software infrastructure to monitor and control the resulting assembly.

The requirement for *dependable service* is fundamental. Users need assurances that they will receive predictable, sustained, and high levels of performance from the various components that constitute the grid. In the absence of these guarantees, application will not be written or used. The performance characteristics that are of interest will vary widely from application to application, but may include network bandwidth, latency, jitter, computer power, software services, reliability and security.

The need for *consistency of service* is an essential concern as well. As with electric power, we need standard services, which are accessible via standard interfaces, and operating within standard parameters. Without such standards, pervasive use and application development are unrealistic. A significant challenge when developing standards is to encapsulate heterogeneity without compromising high performance execution.

Pervasive access provides services that are always available, within whatever environment we expect to move. Pervasiveness does not, of course, imply that resources are everywhere or are universally accessible. Similar to electricity services, computational grids will have circumscribed availability and access. Finally, the grid must offer *inexpensive access* if it is to be broadly accepted and used.

It is the combination of dependability, consistency and pervasiveness that will cause computational grids to have a transforming effect on how computation is performed and used. By increasing the set of capabilities that can be taken for granted to the extent that they are noticed only by their absence, grids allow new tools to be developed and widely deployed. Computational grids have the potential to change fundamentally the way we think about and relate to computation and resources.

The term "the Grid" was coined in the mid-1990s to denote a (then) proposed distributed computing infrastructure for advanced science and engineering. Much progress has since been made, on the construction of such an infrastructure and on its extension and application to commercial computing problems. And while, the term “Grid” has also been on occasion conflated to embrace everything from advanced networking and computing clusters to artificial intelligence, there has also emerged a good understanding of the

problems that grid technologies address, and at least a first set of applications for which they are suited (Foster and Kesselman, 1999).

Grid concepts and technologies were originally developed to enable resource sharing within scientific collaborations, first within early gigabit/sec testbeds and then on increasingly larger scales. Applications in this context include distributed computing for computationally demanding data analyses (pooling of compute power and storage), the federation of diverse distributed datasets, collaborative visualization of large scientific datasets (pooling of expertise), and coupling of scientific instruments with remote computers and archives (increasing functionality as well as availability). A common theme underlying these different usage modalities is a need for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. More recently, it has become clear that similar requirements arise in commercial settings, not only for scientific and technical computing applications but also for commercial distributed computing applications, including enterprise application integration and business-to-business partner collaboration over the Internet.

A more recent definition of Grid states it is a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service. Let us examine the key elements of this definition (Foster and Kesselman, 2004):

- *Coordinates distributed resources* - a Grid integrates and coordinates resources and users that live within different control domains - for example, the user's desktop versus central computing, different administrative units of the same company, and/or different companies - and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings. Otherwise, we are dealing with a local management system;
- *Using standard, open, general-purpose protocols and interfaces* - a Grid is built from multipurpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. As we discuss in material to follow, it is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application-specific system;
- *To deliver nontrivial qualities of service* - a Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating, for example, to response time, throughput, availability, and security - and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

The second point is of particular importance. Standard protocols (and interfaces and policies) allow us to establish resource-sharing arrangements dynamically with any interested party and thus to create something more than a plethora of balkanized, incompatible, non-interoperable distributed systems. Relevant standards are being developed rapidly within the Global Grid forum and other bodies (Globus, 2007). For an entity to be part of the Grid it must implement these inter-Grid protocols, just as to be part of the Internet an entity must speak IP (among other things).

Both open source and commercial products can interoperate effectively in this heterogeneous, multivendor Grid world, thus providing the pervasive infrastructure that will enable successful Grid applications. In the Internet, it is not uncommon that a specific set of hosts is disconnected from other hosts within an Intranet. However, this partitioning occurs as a result of policy and not because of implementation. In general, all networked computers use TCP/IP and its associated protocols; and despite these policy restrictions, we still talk about a single Internet.

Similarly, we speak about the Grid as a single entity, even though different organizations and communities use Grid protocols to create disconnected Grids for specific purposes. As with the Internet, it is policy issues (e.g., security, cost, operational mode), not implementation issues that prevent a service or resource from being accessible. The success of the Grid to date owes much to the relatively early emergence of clean architectural principles, de facto standard software, aggressive early adopters with challenging application problems, and a vibrant international community of developers and users.

4.5 Short Taxonomy of Grid Applications

The history of network computing shows that orders-of-magnitude improvements in underlying technology, invariably enable revolutionary, often unanticipated applications of that technology, which in turn motivate further technological improvements. The integrated computational grids are expected to provide dependable and pervasive computational capabilities and consistent interfaces. The applications will follow this revolutionary path. There are several major classes of grid applications: high-throughput computing, distributed supercomputing, on-demand computing, data intensive computing and collaborative computing (Foster and Kesselman, 1999).

In *high-throughput computing (desktop grid computing)* the grid is used to schedule large numbers of loosely coupled or independent tasks with the goal of putting unused processor cycles to work, much often those cycles coming from idle workstations. The result may be, as in distributed supercomputing, the focusing of available resources on a single problem. The quasi-independent nature of the involved tasks leads to very different types of problems and problem-solving methods. Such a system is Condor from the University of Wisconsin, which is used to manage pools of hundreds of workstations at

universities around the world. These resources have been used for studies as diverse as ground-penetrating radar, design of diesel engines or various molecular simulations of liquid crystals. Cryptographic problems or design of the processors are other common applications.

Distributed supercomputing applications use grids to aggregate substantial computational resources in order to tackle problems that cannot be solved on a single system. Depending on the grid on which one works, these aggregated resources might include the majority of supercomputers in a country or simply all of the workstations within a company. Some examples include distributed interactive or complex physical processes simulations. Distributed interactive simulation is a technique used for training and planning in the military. Realistic scenarios may involve hundreds of thousands of entities, each having potentially complex behavior patterns. Complex physical processes require high spatial and temporal resolution in order to resolve fine-scale detail. Although high latencies can pose significant difficulties, coupled supercomputers have been used successfully in climate modeling, cosmology, and high-resolution computational chemistry applications.

On-demand applications need grid capabilities to meet short-term requirements for resources (software, data repositories, sensors, and so on) that cannot be cost-effectively or conveniently positioned locally. In contrast to distributed supercomputing, these applications are often driven by cost-efficiency concerns rather than absolute performance. Such applications include a system developed at the Aerospace Corporation for processing of data from meteorological satellites that uses dynamically acquired supercomputer resources to deliver the result of a cloud detection algorithm to remote meteorologists in quasi real time. The NEOS and NetSolve network-enhanced numerical solver systems allow users to couple remote software and resources into desktop applications, dispatching to remote servers calculations that are computationally demanding or that require specialized software. A computer-enhanced MRI machine and scanning tunneling microscope developed at the National Center for Supercomputing Applications use supercomputers to achieve real-time image processing.

In *data-intensive applications*, the spotlight is on synthesizing new information from data that is maintained in geographically distributed repositories, digital libraries and databases. The synthesis process is habitually computationally and communication intensive as well. Future high-energy physics experiments will generate terabytes of data per day, or around a petabyte per year. The complex queries that are used to detect attention-grabbing events may need to access large fractions of this data. The scientific collaborators who will access this data are widely distributed, and hence the data systems in which data is placed are likely to be distributed as well. Modern meteorological forecasting systems make extensive use of data assimilation to incorporate remote satellite observations. The process involves the movement and processing of many gigabytes of data. The Digital Sky Survey will make

also many terabytes of astronomical photographic data, which will be available in numerous network-accessible databases. This facility provides for new approaches to astronomical research that are based on distributed analysis, assuming that suitable computational grid facilities exist.

Collaborative computing applications are concerned primarily with enabling and enhancing human-to-human interactions and sharing. Such applications are often structured in terms of a virtual shared space. Many collaborative applications are concerned with making possible the shared use of computational resources such as data archives and simulations. Moreover, such applications have common features with other application classes, which are presented above. For example, the CAVE5D system supports both remote, collaborative exploration of large geophysical data sets and the models that generate them. The BoilerMaker system developed at Argonne National Laboratory allows multiple users to cooperate on the design of emission control systems in industrial incinerators. The different users interact with each other and with a simulation of the incinerator. The NICE system from University of Illinois at Chicago permits children to participate in the creation and maintenance of realistic virtual worlds, for entertainment and education. The grid use scenarios are also included in this class of applications.

Sharing is not simply document exchange, it can rather involve direct access to remote software, computers, data, sensors, and other resources. For example, members of a consortium may provide access to specialized software and data and/or share their computational resources. More abstractly, what these collaborative application domains have in common is a need for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.

The sharing that Grid is concerned with is direct access to computers, software, data, and other resources, as it is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs.

A set of individuals and/or institutions defined by such sharing rules form what we call a *Virtual Organization (VO)*, a concept that is becoming fundamental to modern computing world. VOs enable disparate groups of organizations and/or individuals to share resources in a controlled fashion, so that members may collaborate to achieve a shared goal. As the examples show, VOs can vary greatly in their purpose, scope, size, duration, structure, community, and sociology (Foster and Kesselman, 2004). Nevertheless, a broad set of common concerns and technology requirements can be identified.

In particular, we see a need for highly flexible sharing relationships, ranging from client-server to peer-to-peer; for sophisticated and precise levels of control over how shared resources are used, including fine-grained and multistakeholder access control, delegation, and application of local and global

policies; for sharing of varied resources, ranging from programs, files, and data to computers, sensors, and networks; for virtualization of resources as services, so that diverse capabilities can be delivered in standard ways, without regard to physical location and implementation; and for diverse usage modes, ranging from single-user to multi-user and from performance-sensitive to cost-sensitive and hence embracing issues of quality of service, scheduling, co-allocation, etc.

To conclude this sections three remarks are necessary: first, we should notice that a large variety of successful grid applications exists already and that has been possible despite of the significant difficulties faced by developers of grid applications, in the absence of a mature grid infrastructure. As this will evolve, the range and sophistication of applications is expected to increase tremendously. Secondly, we must point out that almost all of the previous presented applications crave for computational resources that will not be provided by expected growth in single-system performance. This emphasizes the importance of grid technologies which will allow sharing of computation, data access and communication medium. Finally, we notice that many of the above applications are interactive, or dependable on tight synchronization with computational component. Therefore, the grid infrastructure is expected to provide for robust performance guarantees.

4.6 Grid's Integrability, Efficiency and Quality of Services

The plethora of powerful technologies emanating from the industry's laboratories will let us do new and great things, but realizing that potential depends on our ability to integrate these technologies. Such *integration* will grow increasingly easier as open standards become more and more common. Historically, the success of most technologies has depended on the availability of a small number of commonly agreed-upon standards. IT today is moving in the same way, especially with the increase of open standards and the growing trend toward open source software -be it Linux, Grid protocols or Web services. Standardization has the feel of historical inevitability, because it is the only way to integrate that incredibly diverse abundance of technologies. Standards bring the kind of flexibility and modularity that allow technology to be absorbed and managed smoothly, that make it commonplace and unremarkable and permit people to pay attention to what it does, rather then what it is.

The cause of Grid standards took a major step forward in 2002, as open Grid protocols were brought together with Web services in the Open Grid Services Architecture (OGSA), which represents an evolution towards a Grid system architecture based on Web services concepts and technologies and is provided by GLOBUS (Globus, 2007). The Grid allows people share computing power, databases, and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy. Web services' XML-based technologies, such as WSDL, UDDI, and SOAP, can now be used as the language in which to express Grid protocols. Clearly, this

development indicates levels of integration inconceivable just a few years ago, integration at every level that is increasingly dynamic. Such integration will be a major step forward for e-enterprises of any kind. Organizations, universities, departments, divisions, people, and processes will be united as never before. Together, they will be capable of prompt action and reaction, of quickly forming alliances with other organizations, companies or individuals in search of common interests. Standardization and integration, however, are not synonymous with simplicity. The ever-growing volume of technology and the constant spreading out of a heterogeneous infrastructure, no matter how smoothly integrated, lead to profound levels of complexity. And while, the availability of technology and growing standardization continue to push IT toward mass adoption in a post-technology era, the industry must find ways to deal with that complexity, keep it from intruding on the user, and make the infrastructure perform efficiently.

Efficiency poses the same challenge, on a smaller scale, that the IT industry faced in earlier times, when systems like mainframes addressed one job at a time and operating systems were relatively simple. However, computers and their applications eventually had to be shared among many users, and with sharing the efficient allocation of physical resources became extremely tricky. The solution to this problem stands as one of the more influential breakthroughs in the history of computer science. It was the notion of virtualization that, fueled by increasingly powerful and sophisticated operating systems, provided people with their own machines - virtual systems, consisting of virtual I/O, virtual memory, and virtual storage. Virtualization has enabled people to share an expensive and complex resource, as well as the applications and data they were all working with, without worrying about what was there physically, how it did what it did, or even where it was. Increasingly sophisticated operating systems allowed users to invoke a service that then provided and managed the resources needed by these users.

Thirty or so years ago, virtualization within a single system capitalized on a very expensive resource by making it available to users without their needing a deep knowledge of programming in order to use it. Today, the challenge is to virtualize computing re-sources over the Internet. This is the essence of Grid computing, and applying a layer of open Grid protocols to every local operating system, for example, Linux, Windows, AIX, Solaris, and z/OS are accomplishing it. Thus, we will make the sharing of resources over the Internet (or through a private intranet) a reality, while also hiding the vast, complex, global Infrastructure supporting the user.

That transparency is essential to accelerate the move to the post-technology era while enabling businesses and other institutions to make the most of substantial investments in heterogeneous systems. Moreover, since grids bring to bear not just the resources immediately at hand but also those that are distributed all over the world, users on become more productive, paying further dividends on an enterprise's investment in people and

technology. Increased efficiency is the reason so many are turning to open grid protocols to share resources.

In addition to a much more integrated environment and marked increases in efficiency attributable to a shared infrastructure, we can expect considerable, though gradual, gains in the quality of service provided to the enterprise. This will be due primarily to the increasingly autonomic characteristics that will characterize the infrastructure. For the colossal volumes of technology being produced every year to be useful to a human activity or undertaking, all this new, sophisticated technology must be integrable, efficient and manageable. It must get to its users smoothly and quietly, almost unnoticed because it is delivered with a superb quality of service. Grids, because their open standards are running on every system in the infrastructure, will enable increasingly sophisticated levels of integration and management for distributed resources, and the delivery of a great quality-of-service.

Certainly, the level of management today leaves much to be desired, especially in the world of distributed computing and the Internet. In fact, it is a grand challenge for the industry, which must bring to bear more and more sophisticated technologies to provide a very high quality of service at an affordable price. The answer lies in creating highly sophisticated, end-to-end resource management. The system itself should be able to schedule not just one computer at a time but also multiple computers along the path of a particular transaction, enabling truly global collaboration.

For the different nodes to collaborate (whether for availability, scheduling, or anything else), they must exchange information. All the nodes in the infrastructure must be addressed as if by a single operating system managing the resources under its control, the difference being that unlike the resources addressed by a conventional operating system, these are distributed and heterogeneous. They come from different vendors, are the products of different architectures, and are totally reliant on a common set of open protocols to feed back information about the state of the system. All that should take part in an open architecture for grid services. That will work as a virtual operating system working on top of all the local operating systems and permitting the resources of the entire aggregation of heterogeneous architectures to be managed in an automated fashion. Its open protocols will allow management to become more autonomic in nature and be carried out much the way biological systems regulate themselves - unconsciously and autonomically. It will configure, optimize, heal, and protect itself with minimal human intervention. In short, it will be self-managing.

Greater integration, efficiency, and a far higher quality of service are some of the more significant ways in which e-activities or e-undertakings will benefit from grid computing. They are the direct result of the Grid's ability to balance infrastructure needs and costs and to deliver a quality of service that truly unlocks the substantial, unrealized value of the infrastructure. These vast

new levels of integration, efficiency, and resiliency will combine to bring a new, far more flexible computing model to various human e-endeavors.

4.7 Desktop Grid Computing

Distributed computing systems are constructed by integrating diverse end systems, and therefore it is important to understand key characteristics of these systems with respect to both current and expected future capabilities. This section presents briefly *Simple Composite Elements (SCEs)* (Foster and Kesselman, 1999), (Foster and Kesselman, 2004). They are important, because, in fact, the similarity between a national-scale Grid and simple composite elements reflects the "fractal nature" of Grids. Simple composite elements are collections of basic elements, aggregated with software and sometimes, special hardware, to provide a qualitatively different interface and capability. Examples of simple composite elements include high-throughput, high-reliability, dedicated high-performance, and shared controllable system components. These composite elements can be employed in functions suited to their capabilities.

The capabilities of basic elements, the building blocks for all computing systems, have improved at geometric rates for the past three decides. Such rapid change produces not only tremendous quantitative changes in capability (1,000-10,000 times) but also, even more important, qualitative changes. Computer systems were once of the size of a small building and now are wristwatch-sized gadgets. Multiplying the revolutionary changes enabled by size reductions are equally dramatic increases in storage, compute power, and networking capability.

Simple composite elements are richly connected, relatively homogeneous collections of basic system elements (compute, memory, communication, and storage). They are often housed within single administrative domains and in many cases are already thought of as a single system. SCEs are building blocks for wide-area, national, and international Grids. SCEs are worthy of particular study far several reasons. First, local grid technologies can reduce the number of problems higher-level grids must solve. Second, local grids use resources and software to implement the external properties of the composite element affecting its utility or integration into larger grids. And, finally, local grids form the basis for larger grids. Thus, their evolution is an integral part of the challenges in building larger grids.

Together, these integral relationships make understanding technologies for SCEs and their capabilities a crucial element of understanding issues in building grids. For example, in a national Grid, reliable composite elements can be used to provide management (access control and scheduling) and basic services (naming and routing). Other composite elements can provide resource pools with distinct computation power. Composing two SCEs together may be challenging however, if they correspond to different administrative domains or employ distinct data representations or different network protocols. Further on,

we first describe the two key distinguishing features of SCEs: their external interfaces and guarantees and their hardware requirements. External interfaces and guarantees affect the use and utility of SCEs in the larger grid context. Hardware requirements determine the SCE capabilities that can be exploited for building grids. Then we describe a series of state-of-the-art SCEs and technologies: high-throughput clusters, reliable clusters, dedicated high-performance clusters, and shared controllable-performance clusters.

4.7.1 SCEs' Capabilities and Requirements

SCEs can be defined by their external interfaces and guarantees, by their internal hardware requirements, and by their ability to deliver efficient, flexible use of their internal capabilities to applications. The focus here is on classifying these interfaces and guarantees. In addition, because an SCE technology's internal hardware requirements and capabilities are integrally related to its applicability, we also provide a classification of hardware requirements. Together, the two classifications delineate both current-day SCE systems and many other systems under development.

External interfaces and guarantees define how SCE are used by applications and how they can be integrated into larger Grids. Five attributes capture the important distinctions among a wide range of SCEs: *capacity, aggregate performance, reliability, predictability,* and *sharability*. *Capacity* corresponds to the total throughput of the SCE in dimensions of compute, memory, communication, and storage. *Aggregate performance* corresponds to the SCE's ability to deliver compute, memory, communication, and storage performance. *Reliability* reflects the likelihood of unavailability of resource or unavailability (or loss) of data. *Predictability* captures an application's ability to predict the delivered capacity or performance. *Sharability* refers to whether the SCE can be shared, integrating a number of computations on one resource for tighter coupling or simply multitasking.

Hardware requirements and capabilities distinguish the range and capability of the technologies used to build an SCE. These constrain the range of an SCE and distinguish it from higher-level grids. Five attributes capture many important distinctions in applicability: heterogeneity, networking requirements, distributed resources, changes in constituent systems, and scalability. Heterogeneity in compute, networking, and storage elements influences the inclusiveness of an SCE environment and its ability to encompass both legacy and new systems. Networking requirements, as special hardware, link length limited, high bandwidth, and so on, all limit the locales and cost constraints under which an SCE technology can be deployed.

Whether an SCE technology can exploit distributed resources (links tens of meters or thousands of kilometers) limits the geographical extent of the SCE. Whether an SCE technology requires changes in its constituent systems has a significant effect on the deployment requirements and technology insertion

cost, and therefore on deployability of a technology. Scalability of a system influences the number of nodes that can be deployed and the SCE's ability to manage and deliver their performance.

In the following, we use a two-part framework, external interfaces and hardware requirements, to understand and distinguish the wide range of cluster systems and systems that have been built by both researchers and commercial vendors. Each type has distinct capabilities and provides different challenges and advantages for integration into a Grid. They include high-throughput, high-reliability, and dedicated high-performance SCEs.

4.7.2 High-Throughput SCEs or Desktop Grids

In high-throughput computing or desktop grid systems, pooled resources are used to achieve high throughput on a set of compute jobs. Such systems allow large numbers of machines to be added as a single resource in a higher-level grid system, achieving significant benefits in reduced management effort and grid complexity. Systems such as BOINC, Condor, and LSF manage clusters of workstations as pooled resource servers, with the primary application being compute-bound sequential jobs. Although all three systems provide cluster access and resource management facilities, Entropia and Condor also increase the pool of available resources by allowing desktop machines to be added as resources and by ensuring that those resources can be gathered without interfering with the desktop users. Whereas early definitions of high throughput involved only long-running (multiday) jobs, more recent systems such as Entropia have focused on achieving rapid turnaround to enhance scientific or engineering productivity.

Desktop Grids (DGs) evolve in two major directions: *institution- or enterprise-wide desktop grid computing environment* and *volunteer computing*. The former, usually called simply *desktop grid*, refers to a grid infrastructure that is confined to an institutional boundary, where the spare processing capacity of an enterprise's desktop PCs are used to support the execution of the enterprise's applications. User participation in such a grid is not usually voluntary and is governed by enterprise policy. Applications like CONDOR, Platform LSF, DCGrid and GridMP are all such examples. Unlike the PRC model these applications usually allow users to submit jobs for processing.

The later is an arrangement in which volunteers provide computing resources to projects, which use the resources to do distributed computing and/or storage. Volunteers are typically members of the general public who own Internet-connected PCs. Organizations such as schools and businesses may also volunteer the use of their computers. Projects are typically academic (university-based) and do scientific research. But there are exceptions, e.g. GIMPS and distributed.net (two major projects) are not academic. Several aspects of the project/volunteer relationship are worth noting (BOINC, 2006):

- volunteers are effectively anonymous; although they may be required to register and supply email address or other information, there is no way for a project to link them to a real-world identity;
- due to their anonymity, volunteers are not accountable to projects. If a volunteer misbehaves in some way (for example, by intentionally returning incorrect computational results) the project cannot prosecute or discipline the volunteer;
- volunteers must trust projects in several ways: 1) the volunteer trusts the project to provide applications that don't damage their computer or invade their privacy; 2) the volunteer trusts that the project is truthful about what work is being done by its applications, and how the resulting intellectual property will be used; 3) the volunteer trusts the project to follow proper security practices, so that hackers cannot use the project as a vehicle for malicious activities.

The first volunteer computing project was GIMPS (Great Internet Mersenne Prime Search), which started in 1995. Other early projects include distributed.net, SETI@home, and Folding@home. Today there are at least 50 active projects. Desktop grid differs from volunteer computing in several ways (BOINC, 2006):

- The computing resources can be trusted; i.e. one can assume that the PCs don't return results that are wrong either intentionally or due to hardware malfunction, and that they don't falsify credit. Hence there is typically no need for redundant computing;
- There is no need for using screensaver graphics whatsoever; in fact it may be desirable to have the computation be completely invisible and out of the control of the PCs' users;
- Client deployment is typically automated.

4.7.2.1 Key Components for Desktop Grids

The key elements in a desktop grid system include physical node management, resource scheduling, and job scheduling. In addition, systems that also support data-intensive computations include facilities for data management.

Physical node management. The desktop environment presents several unique challenges to reliable computing. Individual client machines are under the control of the desktop user or IT manager. As such, they can be at any time shut down, rebooted, reconfigured, and disconnected from the network. Laptops may be offline or just off for long periods of time. The physical node management layer is supposed to manage these and other low-level reliability issues. It is also expected to provide naming, communication, resource management, application control, and security. The resource management services capture a large amount of node information (e.g., physical memory, CPU, disk size and free space, software version, data cached) and collect it in

the system manager. This layer also should provide basic facilities for process management including file staging, application initiation and termination, and error reporting. In addition, the physical node management layer must ensure node recovery, terminating runaway and poorly behaving applications.

The security services employ a range of encryption and technologies to protect both distributed computing applications and the underlying physical node. Application communications and data are protected with high-quality cryptographic techniques. The control of the operations and resources visible to distributed applications on the physical nodes is necessary in order to protect the software and hardware of the underlying machine. At this level regulation of the usage of resources by the distributed computing application is also expected. This ensures that the application does not interfere with the primary users of the system - it is unobtrusive - without requiring a rewrite of the application for good behavior.

Resource scheduling. A DG system consists of resources with a broad diversity of configurations and capabilities. The resource-scheduling layer accepts units of computation from the user or job management system, matches them to appropriate client resources, and schedules them for execution. Despite the resource conditioning provided by the physical node management layer, the resources may still be unreliable (e.g. the application software itself may be unreliable). Therefore, the resource-scheduling layer must adapt to all kind of changes in resource status and availability, and also to high failure rates. To meet these challenging requirements, multiple instances of heterogeneous schedulers can be supported. This layer also provides simple abstractions for IT administrators, abstractions that automate the majority of administration tasks with reasonable defaults but allow detailed control as desired.

Job management. Distributed computing applications often involve large overall computation (thousands to millions of CPU hours) submitted as a single large job. These jobs consist of thousands to millions of smaller computations and often arise from statistical studies (e.g., genetic algorithms or statistical simulations), parameter sweeps, or database searches (bioinformatics, combinatorial chemistry, Google search etc.). Because so many computations are involved, tools to manage the progress and status of each piece - in addition to the performance of the aggregate job in order to provide short, predictable turnaround times - are provided by the job management layer. The job manager provides simple abstractions for users, delivering a high degree of usability in an environment where it is easy to drown in the data, computation, and the vast number of activities.

4.7.2.2 Requirements for Desktop Grids

Desktop Grid (DG) systems aggregate large numbers of machines (tens of thousands to millions) into a single high-throughput SCE. Such systems allow the desktop systems to be incorporated into a larger grid at a low management effort. Desktop grid systems begin with a collection of computing resources - heterogeneous in hardware and software configuration, distributed throughout a corporate network and subject to varied management and use regimens - and aggregate them into an easily manageable and usable single resource. Furthermore, a desktop grid system must do this in a fashion that ensures there is little or no detectable impact on the use of the computing resources for other purposes. For end users of distributed computing and higher-level grids, the aggregated resources must be presented as a simple-to-use, robust resource that can be easily integrated into larger-scale Grids. A matrix of key requirements for desktop Grids is shown in next table (Foster and Kesselman, 2004), (Browne et al., 2004), (Domingues et al., 2007):

Requirement	**Brief description**
Efficient	A DG should harvest virtually all of the idle resources available
Robust	Computational jobs must complete with predictable performance, masking underlying resource failures. DGs must tolerate job, machine, and network failures and includes a variety of mechanisms for ensuring timely completion of a larger job in the presence of such failures
Secure	The system must protect the integrity of the distributed computation (tampering with or disclosure of the application data and program must be prevented). In addition, the DG must protect the integrity of the desktops, preventing applications from accessing or modifying desktop data
Scalable	DGs must scale to the 1,000s, 10,000s, and even 100,000s of desktop PCs deployed in enterprise networks. Systems must scale both upward and downward, performing well with reasonable effort at a variety of system scales
Manageable	With thousands to hundreds of thousands of computing resources, management and administration effort in a DG cannot scale up with the number of resources. DGs systems must achieve manageability that requires no incremental human effort as clients are added to the system. A key leverage for including DGs as single entities in larger grids is to reduce the management effort
Unobtrusive	DGs share resources (computing, storage, and network resources) with other usage in the corporate IT environment.

	The DG's use of these resources should be unobtrusive, so as not to interfere with the primary use of desktops by their primary owners and networks by other activities
Communicative	DGs effectively execute iterative parallel computations requiring communication among hosts that is anonymous, scalable and fault-tolerant.
Open/Easy to integrate applications	DG software is a platform that supports applications that in turn provide value to the end users. Distributed computing systems must support applications developed with varied programming languages, models, and tools - all with minimal development effort
Multiple-project participation	The rationale for promoting multiple projects, which from the individual point of view of a project might seem counterproductive since the project loses exclusivity of resources, lies in the fact that many projects have downtime (for hardware and software maintenance and reparation of the server infrastructure), and shortage of tasks (for instance, when transitioning from one stage to another). Thus, participation in multiple projects helps to cope with a particular project downtime, besides permitting the volunteers to donate resources for several causes they might find worthy

Table 4.1 Requirements for Desktop Grids

More on robustness issues. Desktop grids, which harvest volunteer or enterprise computing resources, have gained tremendous momentum in recent years attracting hundreds of thousand of enlistees. Currently, more than a dozen large-scale projects exist, and new ones are being created regularly. The advent of open source and easy-to-setup middleware frameworks like have lowered the requirements and skills needed to exploit volunteered resources. To encourage volunteers, projects publish online rankings of contributed work. Interestingly, these rankings cause fierce competition, and attract even more dedicated enlistees.

Although desktop grids have a high return-on-investment, they also have major limitations, namely *resource volatility* and *result correctness*. The volatility of desktop grids is caused not only by hardware and software faults of computing systems, but also by resource owners who retain full priority in accessing and managing their desktops. Thus, owners reclaiming their resources might force hosted applications to be interrupted. Checkpointing is a common solution to cope with volatility, and some support exists for application-level checkpointing in existing desktop grid middleware. It consists in periodically saving the state of the executing task to stable storage, usually the executing machine's local disk. Whenever, the execution recovers from a failure, the last stable checkpoint (a checkpoint can get corrupted, for instance,

if a failure occurs during checkpointing) can be used to resume the execution, reducing the prejudices of the failure.

Two main types of checkpoint exist: *system-level* and *user-level*. The former relies on operating system mechanisms to take a full snapshot of the target process. While it is transparent to the user, it usually generates huge checkpoint files since the whole process image needs to be saved. It also requires support from the operating system (a support that does not exist for instance on Windows) and saved checkpoints are non-portable across operating systems and platforms. On the other hand, user-level checkpointing is application specific and is non-transparent since it requires the involvement of the application programmer. However, the application programmer can select only the data and states deemed relevant, yielding a much lighter checkpoint.

Moreover, if appropriate care is taken in data representation, checkpoints can be used to resume applications across heterogeneous platforms. Apart from Condor which supports system level checkpoint desktop grids middleware like BOINC and XtremWeb resort to user-level checkpoint. A usual limitation of volunteer computing is that checkpoints are private, i.e. a checkpoint taken in a given machine will only be used to resume the application in that machine. Sharing checkpoints in a desktop grid environment for the purpose of optimizing turnaround time could be useful. Under this approach, portable checkpoints are saved in a central storage and can be used for restoring, moving or replicating tasks to other machines (Domingues et al., 2006).

Result correctness of computations performed on volunteer resources is an important issue, since interpreting incorrect results as correct can be worse than no results at all. A major source of result incorrectness is faulty hardware. Often overclocking is a significant cause of faulty computations in projects that resort to the BOINC framework. The fierce competition and rivalry among volunteers sometimes may also cause unhealthy behavior. Some users try to increase, not always by honest means, their credits. In some extreme cases, users resort to dishonest tricks to collect undue credits, like fabricating results that require much less computation than the real ones. These users are known as lazy cheaters. Finally, another type of malicious user, the saboteur, might simply act for the sole purpose of ruining the computation, without concern for credits. In contrast to lazy cheaters, saboteurs may be difficult to counter since they may be resourceful and committed to perform everything they can to disrupt the computation (Domingues et al., 2006)

More on security issues. The verification of results is an important issue that needs to be addressed in any volunteer computation. Indeed, hardware and software mishaps as well as malicious volunteers can falsify the outcome of computations, rendering the results useless. Thus, a major concern of middleware tools supporting volunteer computation is to provide results validation and sabotage tolerance mechanisms. Since computations are run in open and non-trustable environments, it is necessary to protect the integrity of

data and to validate the computation results. Without a sabotage detection mechanism, a malicious user can potentially undermine a computation that may have been executing for weeks or even months. Therefore, it is no surprise that users with computationally demanding applications do not easily trust open environments, rather preferring to have their applications executed over more controlled clusters which offer some reliability and trustability. This means that sabotage-tolerance is a mandatory issue in desktop grids in order to make them trustable and dependable.

Along with sabotage-tolerance techniques, it is crucial to devise protocols for trust management in desktop grids. For this purpose, low-level techniques are employed to gather valuable information for the creation and maintenance of local reputation lists. On top of that, higher level protocols are needed for globally sharing and maintaining an updated view of the participants' reputation. Some trust management systems have already been proposed in the area of Grid, like the Grid EigenTrust framework and the EigenTrust system for P2P networks, among some other proposals. However, these trust management systems do not properly exploit the computational paradigm of desktop-based computing.

Sabotage-tolerance techniques The master–worker model is the common paradigm for computing over desktop grids. Under this model, an application is broken into a large set of individual tasks, with tasks being distributed for computation by the master (also referred to as the supervisor) to request workers. After having processed a task, a worker sends the computed results to the supervisor. In an open environment like the Internet, it is necessary to assess the integrity and correctness of the results, since any host can run a worker. The taxonomy of the sabotage-tolerance techniques can be classified in three distinct groups: *replication and voting*; *sampling*; and *checkpoint-based techniques* (Domingues et al., 2007).

Replication and voting (also known as double-check or as majority voting) was first deployed on a wide-scale by the SETI@home project to cope with erroneous results provoked by faulty hardware and malicious users eager to claim credits for work not performed. The technique is based on the replication of individual tasks to different and preferably non-related workers. When completed, the results of the N replicas are compared and a majority voting is applied. The results that do not agree with the majority are marked as erroneous. If no majority can be determined (e.g. all results disagree), results are classified as erroneous and the task needs to be re-executed. N corresponds to the replication factor, and should be at least equal to two. The error rate of the replication method is determined by the replication factor N and by the percentage of erroneous/malicious volunteers. High levels of redundancy augment the resiliency at the cost of higher impact in the overall performance. For instance, the Einstein@home project diminished its replication factor from 3 to 2 when it switched to a more computational demanding stage (S5), and evidence that replication can significantly consume computing resources.

The main benefits of the replication approach are its support for generic computation and its simplicity, which eases its implementation — the technique is supported by the main desktop grid middleware, and employed by all major public computing projects. On the contrary, a major weakness lies in the wasting of resources, since to complete a task, at least N instances need to be effectively computed. Furthermore, in computations that produce results sensible to hardware and software specificities, some further restrictions might be needed to support replication. For instance, some applications are extremely susceptible to floating-point implementations, and the same task run over different machines can yield different numerical results.

A viable workaround is homogeneous redundancy, upon which replicas of a task are only assigned to homogeneous systems. Regarding sabotage, smart colluding saboteurs can bypass the replication technique as long as they manage to control a majority of replicas of a task. A more subtle limitation of replication-based validation for public computing environments is the potentially long interval that might elapse between the completion of the first result and the existence of enough results for majority voting. This is relevant in credit-based projects, where the effort of volunteers is rewarded through virtual credits. Indeed, credit assignment for a given task is only performed after the result has been validated, that is, after a majority of results matched and a so-called canonical result exists. This means that the worker of the first result might wait a significant amount of time for receiving its due credits. Although this might be perceived as an irrelevant issue, credits and the associated tops, where users are ranked according to their earned credits, are major motivation factors for volunteers to participate in projects and thus everything related to credits should be treated carefully to avoid disgruntled volunteers.

Sampling techniques were developed to overcome the limitations of replication, namely its inefficient usage of resources. Sampling techniques are proposed under four different approaches: *naive; quizzes; spot checks with black lists;* and *ringers* (Domingues et al., 2007). The *naive sample* is a simple technique, which uses probes to test the trustworthiness of participants. Basically, the supervisor sends some test samples to the participants and then checks the results sent back by the assessed workers. However, malicious workers can easily compromise the technique if they are able to distinguish test samples from real application tasks. Indeed, a malicious worker can compute correctly the test samples, only faking application tasks, with its dual behavior possibly going unnoticed. The fact that test samples are computationally less demanding than real tasks makes the identification of test samples relatively easy and thus seriously compromises the usefulness of the technique.

Further, if the test samples are sent separately from the batch of real tasks, the detection of samples is even easier and the technique becomes almost useless in a hostile environment, as occurred in early versions of SETI@home. The naive sample technique can be extended by proposing the Commitment-Based Sampling (CBS) approach for strictly one-way functions f(x). Their goal is

to hide the test samples, making them indistinguishable from real tasks. CBS requires that a host, which computes f (x) in the domain of D, saves all the intermediate results of its computation and builds a Merkle tree to prove that it effectively computed every input x. A Merkle tree is a hash-indexed binary tree, where data is kept on leafs and sibling nodes are built through a hash function. The CBS method involves the following four steps: (1) a participant computes its assigned tasks, locally building a Merkle tree which holds the intermediate results of the computation; (2) the supervisor sends a set of selected samples to the participant; (3) the participant proves its honesty by returning, along with the computed results, the Merkle tree's path up to the leaf; (4) the supervisor verifies the results to check whether the participant is cheating or not. For that purpose, the supervisor reconstructs the Merkle tree. If the hash root node differs from the one reported by the participant, the participant is labeled as a cheater. The main drawbacks of the CBS method are its limited applicability to one-way functions and the requirement that every worker builds and holds a possibly huge Merkle tree. Additionally, it induces a severe computational overhead on the supervisor due to the reconstruction of the Merkle tree.

Quizzes. The naïve sample method can be further extended by hardening the detection of samples. For that purpose, quizzes are mixed along with tasks. When a batch of tasks is finished, the supervisor checks the results related to the quizzes and accepts the results if all quizzes are correct. Otherwise, the results are discarded and the tasks rescheduled for another execution. This method is resilient to collusion and presents the advantage that the samples' outcome can be verified before the end of a task. However, no efficient method exists for generating quizzes in an automatic way, therefore preventing the use of this technique in wide-scale projects.

Spot checks with blacklists were proposed by Sarmenta (Sarmenta, 2001). This technique works similarly to quizzes. The main novelty is the tight integration of the technique with blacklists, which helps to filter out malicious users over time. When a participant is caught cheating, all her contributions until then are invalidated, and the participant is blacklisted and will be left out of any further computations. The implementation of spot-checking with blacklists faces some subtle problems, mainly the requirement of uniquely identifying participants over time. In fact, identification through email addresses, as it is commonly used by most volunteer projects is unreliable, since a malicious participant can easily and quickly obtain new email addresses.

Ringers. Ringers were introduced to protect against coalitions of lazy cheaters assuming that all computational tasks involve the inversion of a strictly one-way function, f(x), for a given value y. An example of the applicability of one-way functions is the attempt to break cryptographic functions through a brute-force approach, as is undertaken by Distributed.net (distributed.net, 2004). Under the ringer approach, the supervisor creates individual tasks, each one involving a part D_i of the whole domain D. Before assigning a task, the supervisor adds to D_i a set of test samples (ringers) y_i, which are inverted

values of D, computed through $y_i = f(x_i)$. Each task is then assigned to a worker w_i, which computes f (x) for all x in its sub-domain D_i. A ringer y_i yields x_i, since $f(f(x_i)) = x_i$. Thus, to check the integrity of results, the supervisor just has to assess the xi , which should correspond to the sent ringers y_i.

Two ringer-based versions have been proposed: basic and bogus. In the basic approach, when the supervisor assigns work to the participants, it includes a list of input values, for which it already knows the outcome, to be computed along with ringers. Each participant must then return the results yielded by the computation of input values and ringers, receiving credit only if all the ringers are effectively committed to the supervisor. A feebleness of this method is that the participant knows the number of ringers. Therefore, a malicious participant can halt computation and return faked results as soon as all ringers of a task have been found. The bogus ringer version surmounts the limitations of the basic version by concealing the real number of ringers from the worker. For this purpose, a randomly chosen number of ringers whose results are of no interest ("bogus") are inserted in the computation set.

Szajda *et al.* tried to extend the ringers technique to generic computations, overcoming the one-way function limitation. In their approach, the supervisor plants ringers on the domain of values to be checked, with participants computing the values in the domain and the inserted ringers. Though, their approach is hardly feasible due to the hardness of generating an automatic method for creating the indistinguishable ringers (Szajda et al., 2003).

Checkpoint-based verification proposes the (a) basic checkpoint verification and the (b) distributed checkpoint verification. Both schemes are checkpoint based techniques for sabotage-tolerance and address sequential computations that can be broken into multiple temporal segments (S_{t1} , ... , S_{ti} , ... , S_{tn}). At the end of each segment, a checkpoint $C(St_i)$ of the task can be committed to stable storage. Next, a brief review of both techniques is given.

Basic checkpoint verification. Under this technique, each worker periodically saves the state of its task in a checkpoint, computes its hash code and submits it to the supervisor. The supervisor randomly chooses a checkpoint-time S_{ti} and requests the corresponding checkpoint $C(S_{ti})$ from the worker. Then, the supervisor computes the partial execution of the task, from S_{ti} up to the next checkpoint $C(S_{ti+1})$. Finally, the hash code of $C(S_{ti+1})$, that is, $H(C(S_{ti+1}))$, is compared with the corresponding hash code sent by the worker. The error rate of the basic checkpoint method depends on the number of checkpoints verified by the supervisor: a high percentage of verified checkpoints yields a low error rate at the cost of increased computation (for the partial computation of the task) and bandwidth (for having the checkpoint S_{ti} transferred from the worker to the supervisor). Since, the entire overhead (computation and bandwidth) needs to be supported by the supervisor, this technique might induce an unbearable overhead to the supervisor, especially in wide-scale systems.

Distributed checkpoint verification extends the basic verification technique by distributing the partial computation over workers, in six steps. Firstly, (1) the supervisor sends a task to the participant. (2) The worker then computes the results along with a list of the partial checkpoint hashes, sending both to the supervisor. (3) The supervisor stores the received hash list and selects a worker (henceforth the verifier) to verify it. The supervisor identifies the partial execution to be computed by the verifier and sends to the verifier the necessary data, namely how to contact the worker being scrutinized, so that it can obtain the checkpoint to load for the partial execution. (4) The verifier requests the initial checkpoint from the original participant, and then it (5) computes the partial task up to the next checkpoint, taking a hash code of this new checkpoint. Finally, (6) this hash code is sent to the supervisor, which compares it with the one it received from the worker under assessment.

The distributed checkpoint verification method allows the verifications without overloading the supervisor. The intermediate steps can also be checked, allowing for the detection of a malicious worker before the completion of a task. The price for this technique is the redundancy required for checkpoint comparison, the cost of communications and the capability of participants to communicate directly with each other, a requirement that can be difficult to achieve when connectivity of hosts is restricted by firewalls and Network Address Translation (NAT) schemes. Even if both machines can contact with each other, promoting direct contact between worker and verifier might create opportunities for collusion by the supervisor, this technique might induce an unbearable overhead to the supervisor, especially in wide-scale systems.

A combination of replication with checkpoint based comparison to promote early detection and finer localization of errors in volunteer computations has been proposed (Domingues et al., 2007). Specifically, they proposed the compare replicated checkpoint hashes technique, and complemented it with trickle messaging to permit early detection of divergent computations. They targeted public computing projects, assuming that a N-level replication is used for results validation.

Under the *compare replicated checkpoint hashes* (CRCH) approach, a worker is requested to return, along with the results of its task, a selected set of hashes of the checkpoints saved along the computation. The list of checkpoints whose hashes are requested is defined at task creation time, so that redundant instances of a task share the same set of requested checkpoint hashes. When a majority of replicated executions are completed, and thus the supervisor holds enough results for meaningful comparisons, the hashes from equivalent checkpoints are compared to each other. If a divergence occurs, the execution point where the differences were detected is marked as suspicious. Comparatively to the result comparisons detection level, since an erroneous computation can be detected right after the first divergent checkpoint. For deterministic errors this might speed up the debugging process, since the

temporal location of the fault is known with some precision, permitting a faster reproduction on of the error.

Relatively to the basic checkpoint and to the distributed checkpoint techniques, CRCH requires no extra communications since the lightweight hashes can be sent to the supervisor along with the results. Additionally, the traditional communication model is not disrupted, since no contact is required between workers, contrary to the distributed checkpoint verification technique. Selective checkpoint hashing is also much less demanding for the supervisor, since no task computation (partial or complete) needs to be performed by the supervisor. Although the CRCH strategy allows for result verification with practically no overhead at the server-side, and permits a more precise location of error occurrence, it does not speed up the detection of incorrect computations, since error detection can only occur after, at least, two replicas of the task have terminated. A more proactive variant is to have workers returning available checkpoint hashes during the computation. Ideally, from detection point-of-view, the worker should send to the supervisor a hash immediately after its computation. However, such an attitude would increase the number of messages and consequently stress the supervisor network, possibly disturbing the whole system performance.

A more realistic approach is to use the so-called *trickle messages* to send checkpoint digests to the supervisor. A trickle message is sent by a worker to the supervisor and provides some status information about the worker. The trickle notification mechanism is used by projects like climateprediction.net, which have lengthy tasks (weeks or months long). It permits workers to update their progression status and to claim pending credits. Although the trickle designation covers a BOINC specific characteristic, the importance of this feedback mechanism for projects with long running tasks renders it mandatory for any serious desktop grid middleware.

Thus, an improvement to the CRCH is to take advantage of the trickle messages, which are already sent by workers to report status, for sending the hashes of the selected checkpoints without additional communication costs. This way, the supervisor can spot an error as soon as a majority of checkpoint digests is available for the considered execution point. Thus, upon detection of a divergent computation, the supervisor can immediately trigger corrective measures. For instance, an additional instance of the task can be scheduled to replace the faulty task. Additionally, the thought-to-be faulty worker can be marked as suspect and further probed to assess its computational honesty, or, if repeating a faulty behavior, can be back listed altogether.

Human-based trusting emphasizes the importance of human factors in security and trust management (Domingues et al., 2007). They point out that the auction site eBay is a live example of the importance of reputation systems to promote transactions among individuals that do not know each other. Indeed, reputation systems are important because they collect, distribute and aggregate feedback about participant's behavior and help to decide whom to trust,

implicitly encouraging trustworthy behaviors. Further they propose the *Volunteer Invitation-based System* (VIS) for trust management targeted at volunteer DGs. The protocol establishes and updates the reputation of the participants according to their relationship in the volunteer chain, using underlying sabotage-tolerance mechanisms to detect sabotage attempts to undermine the computations, or simply, computation errors due to faulty hardware. This system aims at building trustable networks of volunteers resorting to invitations. The invitation-based system can be extended so that it supports recommendations of participants across multiple volunteer projects. The basic goal is to permit a volunteer who is already participating in a public project (or has participated in the past), to apply for an invitation in another project (from which the volunteer does not know anyone to ask directly for an invitation), presenting as references a virtual certificate provided by the project(s) s/he is currently participating in or has participated in the past. This virtual certificate would include the worker performance and trustability metrics, such as the ratio of successful tasks completed, earned credits, and errors. Note that a certificate-based scheme could attenuate the possibly slow growth endured by a VIS-based system in its early stage, when the number of volunteers with invitation cards is still small.

More on communications requirements. Desktop grids have been by now very successful in cost effective computation of fully partitionable computations. But is also clear that desktop grids cannot be applied to general parallel computations as long as communication is restricted to the master-slave model of parallelism and communication and current parallel computational infrastructures, which for the most part rely on synchronous algorithms, executing in a fully reliable resource environment. The requirements for desktop grids which can effectively execute iterative parallel computations requiring communication are anonymous, scalable and fault-tolerant communication among the hosts of a scalable desktop grid systems and fault-tolerant computational algorithms, which are insensitive to heterogeneity in processing power of hosts and communication speeds among hosts. The requirements on the computational algorithms are obvious from the nature of desktop grids as are the requirements for scalable and fault-tolerant communication. Anonymity is required of the communication mechanism among the hosts in a desktop grid because the software executed on hosts is written by users and poses security and privacy risks even when encapsulated by desktop grid client agents. Anonymity among the desktop resources minimizes security and privacy violation (Browne et al., 2004).

Multiple-project participation. The participation of a volunteer in multiple projects is not a novelty, and is actually promoted by the BOINC platform, which permits that a volunteer donates resources to several projects, specifying the CPU time distribution to be allocated to each project. The rationale for

promoting multiple projects, which from the individual point of view of a project might seem counterproductive since the project loses exclusivity of resources, lies in the fact that many projects have downtime (for hardware and software maintenance and reparation of the server infrastructure), and shortage of tasks (for instance, when transitioning from one stage to another). Thus, participation in multiple projects helps to cope with a particular project downtime, besides permitting the volunteers to donate resources for several causes they might find worthy (Domingues et al., 2007).

4.7.2.3 External Interfaces and Guarantees

A high-throughput cluster provides high computational capacity and is, obviously, a sharable resource. The job manager or resource scheduler might present its interface in the Entropia system. In fact, most such systems share resources with interactive users, and some include elaborate mechanisms for ensuring good interactive response. While these systems provide: no special support for aggregate performance, interfaces for loosely coupled parallel computing such as PVM (Sunderam, 1990) are now becoming available. No special support for reliability or predictability is provided.

4.7.2.4 Hardware Requirements

High-throughput SCEs have minimal hardware requirements, running on a wide range of processor and network environments and tolerating both processor and network heterogeneity in type and speed. High-throughput systems are also used on widely distributed resources, such as for pooling workstation resources across the worldwide sites for a corporation. In addition, high-throughput SCEs do not require significant change to the underlying systems (depending only on some common job controls and special system libraries) and can scale to larger numbers of processors (hundreds lo thousands) with little difficulty.

4.7.2.5 High-Throughput SCEs in Grids

High-throughput SCEs are flexible, powerful systems for achieving high throughput on large numbers of sequential jobs. Thus, they are very suitable grid elements for such tasks. These SCEs manage a wide range of heterogeneity automatically (instruction set, memory configuration, network, etc.) and schedule compute resources efficiently to reduce turnaround title for jobs. In addition, effective sharing of resources with interactive users increases the pool of resources available to the SCE dramatically. The primary benefit of using DGs to organize large numbers of small resources is that the complexity of the higher-level grid is reduced (dramatically fewer SCEs), and the usability of the small resources is enhanced through the sophisticated management that the high-throughput SCE software provides.

However, because high-throughput SCEs primarily focus on processing large numbers of small but compute-bound jobs, such SCEs do little to efficiently aggregate resources for larger computations, enhance reliability, or improve performance predictability. Each of these issues is addressed by at least one of the other types of SCEs described next.

4.7.3 High-Reliability SCEs

High-reliability SCEs provide computational resources with extremely low probability of service interruption and data loss. In high-reliability SCEs, commonly called *reliable clusters,* additional computing resources are deployed to replicate the state of an application, and responsibility for the computation is 'failed over' automatically in the case of software, hardware, or any other failure. Failover transfers responsibility for the computation to the additional hardware, which takes up the task seamlessly, so clients see no interruption of service. This approach, typified by Tandem's Guardian system, has been adopted by a wide variety of vendors for highly available systems.

4.7.3.1 External Interfaces and Guarantees

Reliable clusters use replication for reliability but can also add resources for scalability for many kinds of applications. With the exception of a few large data manipulation applications, however, the scalability is generally used to increase system capacity, not to scale to support large jobs. Of course, reliable clusters provide a reliability guarantee to applications and are generally sharable resources. Because of failover delays and dynamic load sharing, most reliable systems do not provide strong guarantees of predictable response.

4.7.3.2 Hardware Requirements

Reliable clusters generally prefer compatible hardware to enable failover, data sharing, and convenient restoration from checkpoints. For cold standbys, however, less powerful configurations can be deployed to reduce cost, provided lower performance is tolerable in a failover situation. Custom networking is employed among cluster nodes and between primaries and standbys to ensure fault detection and isolation at the earliest possible time. Reliable clusters can be physically localized or distributed over a wide area network. Traditionally, reliable systems use special operating systems (e.g.. Tandem NonStop kernel), but many recent systems have been implemented as a middleware layer, so a specialized operating system is no longer required. Finally, reliable dusters can also include multiple nodes for scalability in capacity.

4.7.3.3 High-Reliability SCEs in Grids

High-reliability elements are a natural choice for simple composite elements. The internal substructure of such elements is encapsulated, allowing them to be

viewed as reliable, high-capacity systems. Such systems can provide a wealth of important grid services reliably, facilitating rigorous reasoning about operation, bootstrap procedures, reconfiguration, failure modes, and so on.

4.7.4 Dedicated High-Performance SCEs

Dedicated high-performance SCEs merge basic computing elements into a single resource pool for computing, memory, and storage, allowing these resources to be applied to a single computational application. Since microprocessors have become the fastest processors available, collections of microprocessors (especially parallel processors) or entire systems (scalable clusters or networks of workstations) have become an attractive and cost-effective way to achieve very high performance.

By employing standard workstation or PC building blocks and scalable networks, these dedicated high-performance clusters can be scaled to arbitrarily large and complex configurations. These systems were initially applied to supercomputing tasks and were operated as dedicated systems, with space sharing used to run two or more applications simultaneously. To connect hundreds or thousands of nodes together with high efficiency, dedicated high-performance systems employ high-speed custom networks with limited physical extent (tens of meters). These networks employ parallel data links and custom signaling to deliver high performance, as with the custom cluster networks described in a previous section.

In case of the IBM Blue Horizon machine, it uses high-volume microprocessors as their basic computation engines. It employs 8-processor SMP servers as the basic building blocks, with custom interconnects delivering ~350 megabit/sec of network bandwidth to each node in the system and latencies as low as 20 microseconds. This system uses a standard AIX (IBM's UNIX) workstation operating system and a collection of middleware to provide, job scheduling, program loading, file input/output, and so on (Foster and Kesselman, 2004). Allowing a single job on each node enables high performance on dedicated jobs-direct access to networks, management of local memory, and so on. Operating system services such as file access and external network input/output are hosted on system service nodes.

The main programming model on these systems is explicit message passing, typically via a standard interface as MPI. This model enables the achievement of high performance at the price of explicit programmer management of naming and data movement. Higher-level interfaces, such as HPF and distributed shared memory, are used to a lesser degree. For a large number of applications, high-performance MPI implementations have been built, and therefore for these applications, dedicated high-performance SCEs can effectively aggregate their compute performance.

These systems provide single-system image (uniform monitoring, resource usage, file system access, etc.). Distributed shared-memory systems

have demonstrated techniques for efficient memory pooling, although significant issues remain about how to manage and share such pools as well as how to best implement virtual memory in such an environment. Efficient scheduling remains a difficult challenge, as schedulers typically focus disappointingly on processors, utilizing memory, network, disk, and other resources (Foster and Kesselman, 2004).

4.7.4.1 Beowulf Clusters

An increasingly popular dedicated high-performance SCE is a PC cluster, commonly known as a Beowulf cluster (Sterling 1999), which consists of high-volume products such as dual-processor desktop or server systems, networked by low-cost, commodity fast Ethernet or gigabit Ethernet networking. These systems are predominantly Linux based (Redhat, Debian, and Suse being popular) and there are a wide variety of both commercial and research or academic software systems for assembling and managing such cluster systems.

Commercial systems include Scyld, Scali, Platform Computing, VA Cluster, and Score. Research and academic systems include Oscar and NPACI Rocks. Although the functionality in these systems varies generally, they all address elements of the key challenges in building dedicated high-performance elements from commodity components: configuration management, scheduling, single-system image, and a shared file system. These software packages allow a Beowulf cluster to be viewed as an aggregate resource with a single point of access for inclusion as a dedicated, high-performance SCE into the Grid. Typical Beowulf cluster systems are anywhere from 8 to about 128 nodes, with the majority of the systems being in the range of 16 to 64 nodes. Above 64 nodes, the complexity of physical machine maintenance, configuration management, and even network wiring becomes significant, and the advantages of custom-engineered systems are more pronounced. Even at 64 nodes, however, Beowulf systems can have substantial compute, memory, and storage capabilities.

4.7.4.2 Commercial Resource Virtualization Systems

Lately, a number of commercial vendors have introduced resource virtualization systems that increase the manageability of resources and the applications deployed on them. Examples of such commercial systems include IBM's Oceano, Hewlett-Packard's Utility Data Center, and Sun Microsystem's N1. Although public information on these systems is limited at present, each of these systems purports to support "wire once" approaches to hardware in large server complexes, automated deployment of applications, monitoring, provisioning, and evolution as application needs evolve. Many of these systems are advertised as providing a single-system view of an entire data center, much as cluster software packages provide a single-system view of a cluster. As the commercial virtualization systems become more widespread, they will not only

support dedicated high-performance SCEs as grid elements but will also extend their capabilities to include dynamic deployment of applications.

4.7.4.3 External Interfaces and Guarantees

Dedicated high-performance SCEs aggregate resources to speed up individual computational applications and are scalable to hundreds or thousands of nodes. Hence, they provide both pooled capacity for sequential jobs and high performance for parallel computations. In fact, many scheduling systems such as Sun's Grid Engine, IBM's LoadLeveler, and Platform Computing's LSF will schedule in combination both uniprocessor and dedicated parallel jobs. Because they focus on highest single job performance (supercomputing), however, dedicated SCEs have not delivered reliability or predictability and are not generally sharable (other than via space partitioning) (Foster and Kesselman, 2004). More, many commercial reliable cluster products have an element of scalability but generally do not deliver the levels of performance described for dedicated high-performance SCEs.

4.7.4.4 Hardware Requirements

In dedicated high-performance SCEs, aggregate performance is the primary objective, so scalability to hundreds or thousands of nodes is a must. Hardware attributes (e.g., heterogeneity) that degrade performance are not generally included. Software features (e.g., process pairing for reliability) that reduce performance are not included either. Further, because the SCE is viewed as a single system, changes to the underlying systems (operating system and motherboard) are sometimes required. Networks and interfaces are virtually always customized. Recently, under pressure from low-cost, high-volume products, many vendors have chosen to use unmodified workstations and operating systems, differentiating only with modest scheduling and middleware software.

4.7.4.5 Dedicated High-Performance SCEs in Grids

Dedicated high-performance SCEs can be real assets in a grid environment. Indeed, many such systems are deployed in production grids nowadays. However, their dedicated-use model significantly reduces their effectiveness. This observation provides a major impulsion for the development of shared controllable-performance systems. For the future, broadening the model of use is essential both for improving resource utilization and for supporting a broader class of resource-intensive online and interactive applications.

4.7.5 Concluding comments

In this section, we focused on the capabilities of basic elements and SCEs. The capabilities of basic computation, communication, and storage elements continue to improve geometrically, producing a grid wealthy in resources and

capable of substantial sharing because of the availability of high-bandwidth links. SCEs provide both aggregate capabilities and qualitatively different capabilities, such as reliability. Thus, high-throughput SCEs (desktop grids) are scalable, non-aggregatable, partially reliable, non-predictable and sharable. Reliable SCEs have limited scalability, are reliable and sharable, but non-aggregatable and non-predictable. Finally, dedicated high-performance SCEs are scalable and aggregatable, but non reliable, predictable or sharable. These capabilities will determine their role and contribution to larger grids.

Grids based on tile Globus Toolkit are now moving from a resource to a services model in which all capabilities are presented as network grid services. In such a model, SCEs can provide compute resources, data/storage resources, and application services. As compute resources, they will form dynamic grid application servers, allowing compute-oriented applications to be dynamically instantiated and provide grid compute application services. As data/storage resources, they will provide a wealth of data Grid services. As these capabilities are combined with the increasingly popular commercial resource virtualization systems, applications will increasingly be expressed in a fashion independent of the detailed platform environment. That is, if they do not need the greatest possible performance or access to unique services, they can be expressed against a virtualized interface. Such an approach will further increase the liquidity of applications and their flexible deployment, enabling further progress in achieving the grid vision of computing and application services as a fungible resource (Foster and Kesselman, 2004) (Globus, 2007).

A number of important challenges arise in building bridges from useful SCEs to large-scale grids. Three major elements of these challenges are composition (interfacing), performance guarantees, and security and data integrity. Composing both basic elements and SCEs into larger grids is a complex challenge, which requires directory services, protocols, compatible services and data representations, and conversion. Although a general solution must involve all element types, special problems are raised here by simple composite elements. They include interfaces, semantics, scheduling, and management lot aggregated memory resources, aggregated persistent storage resources, aggregated communication resources, and migration/interoperation. The distinct challenges for SCEs here include developing interfaces to aggregate resources that provide simple semantics and high performance and still reflect the fact that, even within an SCE, the hardware elements come and go dynamically. A number of lower-level issues are also critical: how to map aggregated resources to the disjoint basic elements within an SCE (e.g., where do the I/O requests and computations go?), whether such mappings are static or dynamic, and how users can manage their usage of the resources.

Another important challenge with SCEs is allowing grid computations to achieve reasonable overall performance. In most current approaches, this is predicated on some ability to guarantee performance from grid elements. Nevertheless, such techniques must be reconciled with local resource

management policies designed to achieve local resource and computational efficiency. Therefore, challenge for SCEs in grids include the following (Foster and Kesselman, 2004):

- Mechanisms to ensure predictable performance for memory, computation, and storage both for basic and for aggregated resources;
- Techniques for coordinated scheduling across basic (within an SCE) and global memory, computation, and storage resources;
- Policies for predictable performance, which manage the needs of global and local computations against available resources.

After all, if users are to achieve high productivity in a computing environment spanning numerous physical resources and administrative domains, they must be shielded from a wealth of security and data integrity concerns, for example, users may wish to use remote storage facilities for performance or even cost advantages. Conversely, if we have them worry about unauthorized data disclosure or unexpected loss of data, it is very likely that they will be rare. Thus, important challenges for SCEs in grids include both providing safe access to resources and data security for participating computational applications and providing data integrity guarantees for data, independent of availability or failure of any individual data repository.

5 Overview and Taxonomy of Desktop Grid Systems

Desktop Grid (DG) has recently received the rapidly growing interest and attraction because of the success of the most popular examples such as SETI@Home and distributed.net. SETI@home is one of the most successful projects that use such a model. One of the reasons for this success is its simplicity in enabling contributors to donate computational resources—when the computer screensaver is activated the application starts by making a request to a remote server to download tasks to be processed. Another reason is its support for Windows operating system, since the majority of the desktop machines around the world run Windows. Based on the same concept, there are other @home projects: FightAIDS@home, Folding@home, evolution@home, etc. All of these projects are primarily targeted for applications that can be expressed as parameter-sweep applications. They have no or lack of support for creating applications consisting of tasks that need to communicate and coordinate their activities by exchanging messages among themselves (distributedcomputing.info, 2007). In this section we will present first an overview of some of the most well-known and used desktop grid systems, and will conclude with a taxonomy of these systems.

5.1 Overview of Desktop Grid Systems

5.1.1 SETI@home - BOINC

SETI, or the Search for Extraterrestrial Intelligence, is a scientific effort seeking to determine if there is intelligent life outside Earth. One popular method SETI researchers use is radio SETI, which involves listening for artificial radio signals coming from other stars. Previous radio SETI projects have used special-purpose supercomputers, located at the telescope, to do the bulk of the data analysis. In 1995, a new idea was proposed to do radio SETI using a virtual supercomputer composed of large numbers of Internet-connected computers.

SETI@home, developed at the University of California in Berkley, is a radio SETI project that lets anyone with a computer and an Internet connection participate. The method they use to do this is with a screen saver that can go get a chunk of data from a central server over the Internet, analyze that data, and then report the results back. When the computer is needed back, the screen saver instantly gets out of the way and only continues it's analysis when the computer is not anymore used. The program that runs on each client computer looks and behaves like a captivating screen saver. It runs only when the machine is idle, and the user can choose from several different colorful and dynamic "visualizations" of the SETI process. Some of these visualizations will look technical, some will look abstract, and some will look decidedly artistic, as it can be seen in the screenshot from Figure 5.1.

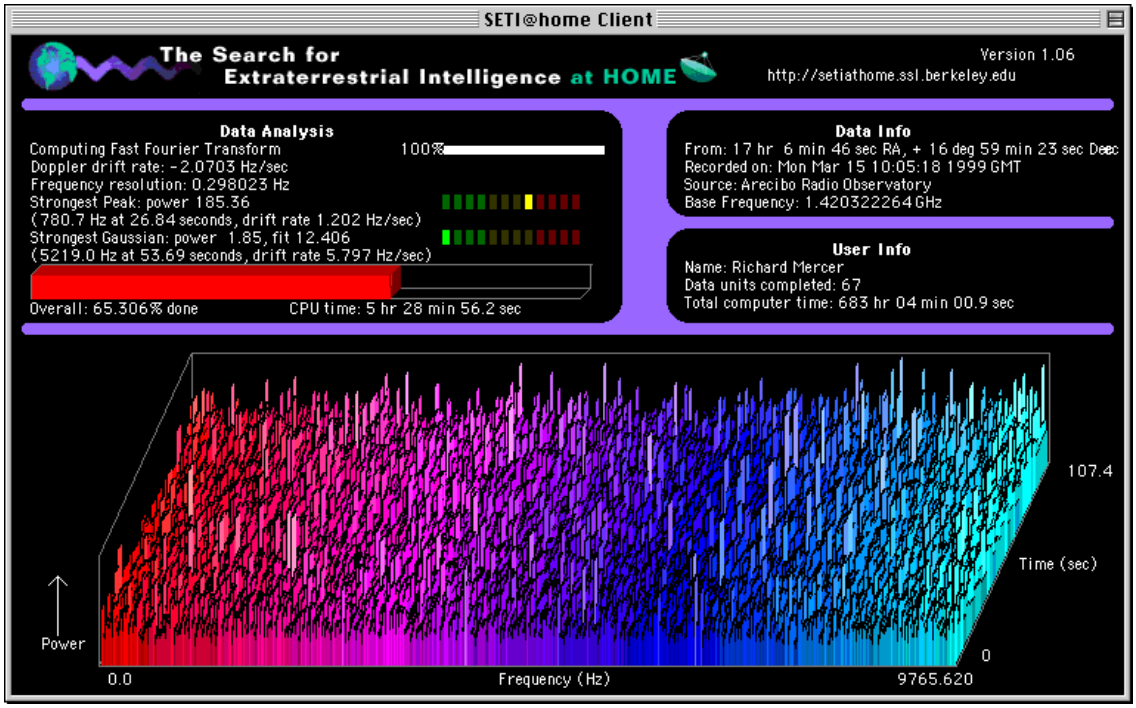


Figure 5.1 SETI@home screenshot

The data analysis task can be easily broken up into little pieces that can all be worked on separately and in parallel. None of the pieces depends on the other pieces, which makes large deployment of clients and computations very easy over the Internet. SETI@home needs network connection only when transferring data. This occurs only when the screen saver has finished analyzing the work-unit and wants to send back the results. Each work unit is sent multiple times to different users in order to make sure that the data is processed correctly. The system architecture is depicted in Figure 5.2.

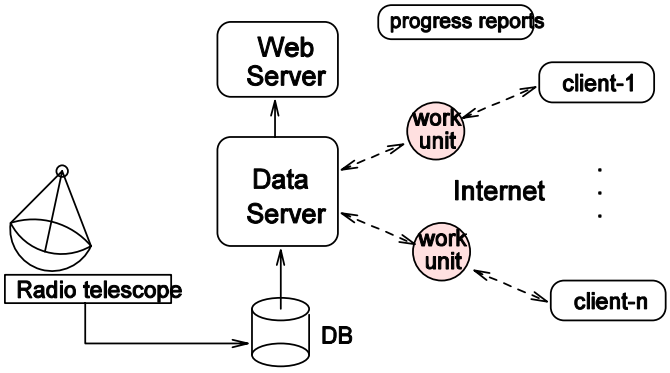


Figure 5.2 SETI@home architecture

While the "screen saver" is running, the client would be processing the quarter-megabyte data block (work-unit), which would contain 50 seconds within a 20-kilohertz range. The algorithm examines this data for strong signals

or "chirps" while taking Doppler shifting into account. False alarms would be prevented by tests for terrestrial interference. Once a block was processed, it would be returned to a centralized SETI@home computer where the results would be stored and organized. This process, when replicated tens or hundreds of thousands of times, has the capacity to analyze the data much more closely than before, perhaps noticing subtle patterns that real-time signal processing missed. The overall results of the search would appear on the SETI@home web site, making the findings immediately available to the public and to the participants. SETI@home is the largest public distributed computing project in terms of computing power: on September 26, 2001 it reached the ZettaFLOP (10^{21} floating point operations) mark, a new world record, performing calculations at an average of 71 TeraFLOPs/second. For comparison, the fastest individual computer at that time in the world was IBM's ASCI White, which runs at 12.3 TeraFLOPs/second. On June 1, 2002, the project completed over 1 million CPU years of computation.

SETI@home was not without problems. For all the media attention and public interest, funding has not been forthcoming. Developing new software to run the distributed system and to perform the analysis on the client side is a difficult and expensive process. The SETI@home project has been delayed repeatedly due to lack of corporate sponsorship. "People time", rather than computer power, has proven to be hard to come by, and in the end it seems that expense - the very thing that SETI@home and distributed computing are meant to escape - may be a force as inexorable as gravity. The *SETI@home* project is for a very specific problem, as described above. There was no general framework for the system, which can be used by other types of applications, and it became SETI@home Classic. Then new funding came for the BOINC project and SETI@home was rewritten for the new framework and it became SETI@home II in 2005. BOINC is open-source software for volunteer computing and desktop grid computing. It includes the following features: project autonomy, volunteer flexibility: flexible application framework, security, server performance and scalability, source code availability, support for large data, multiple participant platforms, open, extensible software architecture, and volunteer community features. BOINC is designed to support applications that have large computation requirements, storage requirements, or both. The main requirement of the application is that it be divisible into a large number (thousands or millions) of jobs that can be done independently. If the project is going to use volunteered resources, there are additional requirements as public appeal and low data/compute ratio (BOINC, 2006).

5.1.2 distributed.net

A very similar project is the *distributed.net* project (distributed.net, 2008). It takes up challenges and run projects which require a lot of computing power. Utilizing the combined idle processing cycles of the members' computers solves

these. The collective-computing projects that have attracted the most participants have been attempts to decipher encrypted messages. RSA Security (RSA, 2005) a commercial company has posted a number of cryptographic puzzles, with cash prizes for those who solve them. The company's aim is to test the security of their own products and to demonstrate the vulnerability of encryption schemes they consider inadequate. The focus of the *distributed.net* project is on very few specialized computing challenges. Furthermore, the project releases only binary code of the clients and no server code, making impossible the adaptation of this to other types of projects.

Typical RSA challenges could either involve factoring, or call for a more direct attack on an encrypted text. In one challenge the message was encoded with DES, the Data Encryption Standard, a cipher developed in the 1970s under U.S. government sponsorship. The key that unlocks a DES message is a binary number of 56 bits (or larger: 64, 72 bits). In general the only way to crack the code is to try all possible keys, of which there are 256, or about 7 * 1016. Another RSA challenge also employed a 56-bit key, but with an encryption algorithm called RC5. Compared with earlier distributed-computing projects, the RC5 efforts were not only technically sophisticated but also reached a new level of promotional and motivational slickness.

For example, they kept statistics on the contributions of individuals and teams, adding an element of competition between teams, as it can be seen in Figure 5.3. The RSA Challenge numbers are the kind, which are believed to be the hardest to factor; these numbers should be particularly challenging. These are the kind of numbers used in devising secure RSA cryptosystems. The challenges are an effort to learn about the actual difficulty of factoring large numbers of the type used in RSA keys.

Another type of project, which involves a lot of computing power, is the optimal Golomb Ruler (OGL) (Gardner, 1972). Essentially, a Golomb Ruler is a mathematical term given to a set of whole numbers where no two pairs of numbers have the same difference. An Optimal Golomb Ruler is just like an everyday ruler, except that the marks are placed so that no two pairs of marks measure the same distance. OGRs have many uses in the real world, including sensor placements for X-ray crystallography and radio astronomy. Golomb rulers can also play a significant role in combinatorics, coding theory and communications. The search for OGRs becomes exponentially more difficult as the number of marks increases ("NP complete" problem).

5.1.3 Considerations on parallelism for SETI@home - distributed.net

None of these two systems provide support for parallel application, when communication between programs running on different computers is necessary during the computation. This makes difficult to use such systems for our purpose, where more than one desktop computer are needed to solve a certain problem. Tasks with independent parallelism are suited for this type of

computing. In SETI@home, work unit computations are independent, so participant computers never have to wait for or communicate with one another. If a computer fails while processing a work unit, the work unit is eventually sent to another computer. Public-resource computing, with its frequent computer outages and network disconnections, seems ill-suited to parallel applications that require frequent synchronization and communication between nodes. However, scheduling mechanisms that find and exploit groups of LAN-connected machines may eliminate these difficulties.

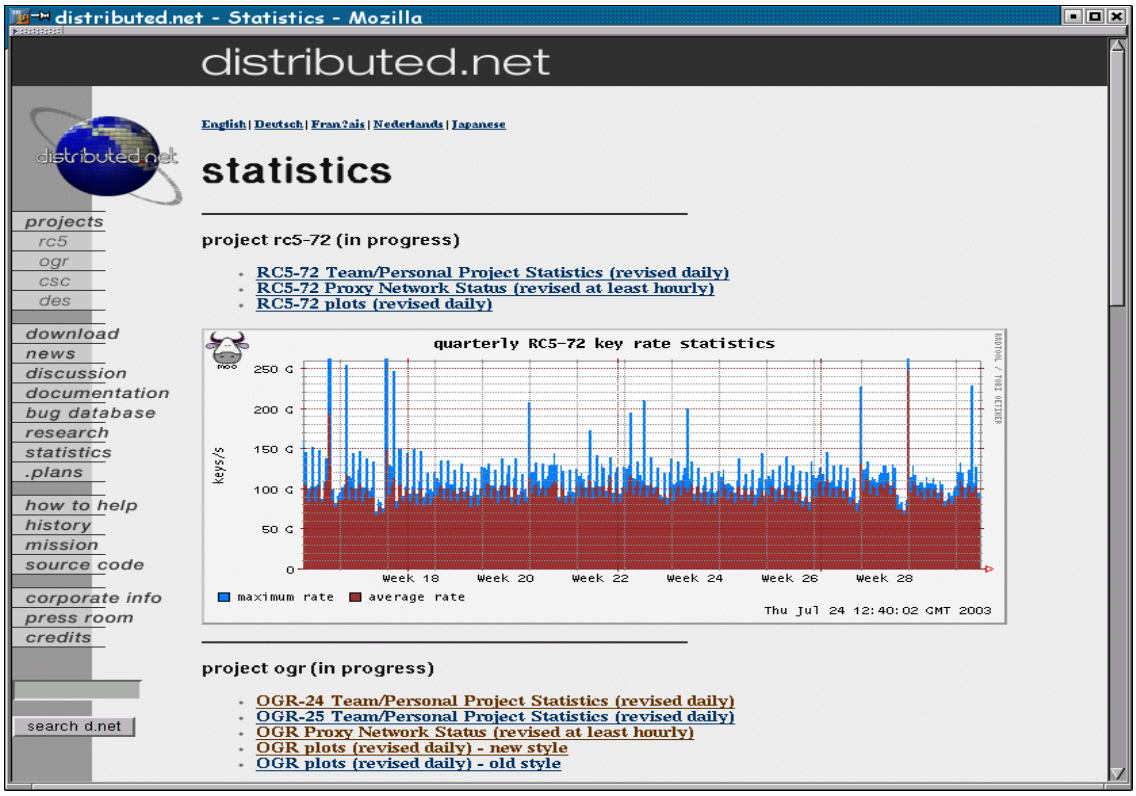


Figure 5.3 distributed.net statistics screen

5.1.4 PVM

PVM (Parallel Virtual Machine) is a portable message-passing programming system, designed to link separate host machines to form a virtual machine, which is a single, manageable computing resource. The virtual machine can be composed of hosts of varying types. The general goals of this project are to investigate issues in, and develop solutions for, heterogeneous concurrent computing. PVM is an integrated set of software tools and libraries that emulates a general-purpose, flexible, heterogeneous concurrent computing framework on interconnected computers of varied architecture. The overall objective of the PVM system is to enable such a collection of computers to be used cooperatively for concurrent or parallel computation.

Applications can be composed of any number of separate processes and are provided access to PVM through the use of calls to PVM library routines for

functions such as process initiation, message transmission and reception, and synchronization via barriers or rendezvous. PVM is effective for heterogeneous applications that exploit specific strengths of individual machines on a network.

The PVM system is composed of two parts. The first part is a daemon (called pvmd) that resides on all the computers making up the virtual machine. This is designed in such a way that any user with a valid login can install this daemon on a machine. When a user wishes to run a PVM application, he first creates a virtual machine by starting up PVM. Multiple users can configure overlapping virtual machines, and each user can execute several PVM applications simultaneously.

The second part of the system is a library of PVM interface routines. It contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This library contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine.

The PVM computing model described in Figure 5.4 is based on the notion that an application consists of several tasks. Each task is responsible for a part of the application's computational workload. Sometimes an application is parallelized along its functions; that is, each task performs a different function, for example, input, problem setup, solution, output, and display. This process is often called functional (task) parallelism. A more common method of parallelizing an application is called data parallelism. In this method all the tasks are the same, but each one only knows and solves a small part of the data. This is also referred to as the SPMD (single-program multiple-data) model of computing. PVM supports either or a mixture of these methods. Depending on their functions, tasks may execute in parallel and may need to synchronize or exchange data, although this is not always the case.

The general paradigm for application programming with PVM is as follows. A user writes one or more sequential programs in C/C++, or Fortran 77 that contain embedded calls to the PVM library. Each program corresponds to a task making up the application. These programs are compiled for each architecture in the host pool, and the resulting object files are placed at a location accessible from machines in the host pool. To execute an application, a user typically starts one copy of one task (usually the "master" or "initiating" task) by hand from a machine within the host pool.

This process subsequently starts other PVM tasks, eventually resulting in a collection of active tasks that then compute locally and exchange messages with each other to solve the problem. Note that while the above is a typical scenario, as many tasks as appropriate may be started manually. Tasks interact through explicit message passing, identifying each other with a system-assigned, opaque task identifier. PVM support both task- and data-parallelism. An advantage of the PVM system is that it is quite popular today and has become a de-facto standard for message passing. There are many algorithms implemented using PVM, and there is a large body of experience in using it.

It is well known and accepted in the academic environment, due to its easiness of use and the availability of source code from the public domain. However, it is not very widespread in industry.

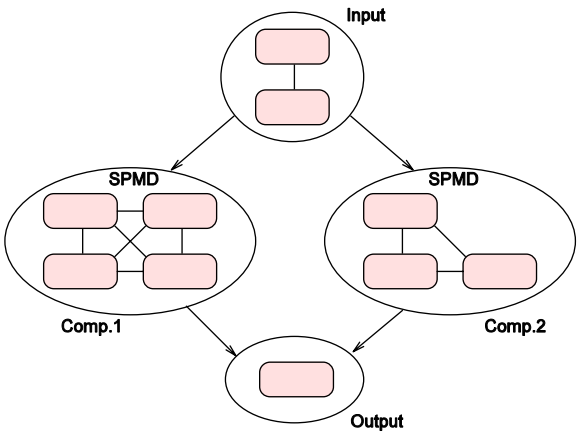


Figure 5.4 PVM Computing Model

Alas, PVM provides only the parallel programming environment and does not offer resource management. This means that the system could not prevent access of different users to the same computing resource. Two or more users could share the same CPU without even knowing that. Such sharing of CPU could result in an inefficient use of resources, especially when running a data-parallel application with uniform computational requirements per task. Further job/resource scheduling systems are required to provide exclusive access to CPU resources in a PVM environment.

Another disadvantage of the PVM system is that the user needs to have 'login' access to each of the computers involved in a computation. From the user's point of view this is done in a transparent way, by automatically using remote login (usually Rush or ssh) to start the application on each computer. There are certain problems with this, which limits a large-scale deployment of the system in many situations. One is that in a completely heterogeneous environment, consisting of operating systems with different types of user authentication (e.g. Unix, Windows and Mac), allowing users' login access to each computer on the network can be extremely difficult to set up and later maintain it. This could also easily be the cause of a potential security problem. For this reason, in many real-life situations, users are not allow to remotely login to the computers in the network, or if so, to only a very few servers. It has also been found to be difficult to install PVM for recent versions of Windows, making it very hard to deploy and use in a today's typical large-scale corporate network, where desktop machines running different operating systems are usually available. So PVM need 'more' heterogeneity than just Unix systems.

5.1.5 Entropia

DCGrid, developed by the company called Entropia, was a PC grid computing platform that provides high performance computing capabilities by aggregating the unused processing cycles of networks of existing Windows-based PCs. The system is no longer used due to the fact that the system was thought in the first place as being commercial. We have chosen to still present it since it was a major desktop grid system, which has had significant contributions to the field.

Existing proprietary and third party applications could be deployed on the DCGrid platform quickly and easily using DCGrid's rapid integration features, which allow enterprises to achieve business objectives faster, with higher throughput, increased precision and more meaningful results in less time than previously possible. DCGrid solutions enabled new and more difficult problems to be solved. Unused PC resources are harvested based on user and organization policies, with settings centrally monitored and managed with a web-based grid management interface. Work is scheduled to PCs based on application resource requirements, and is monitored and rescheduled as necessary if there are system disruptions or resource unavailability. Any native Win32 application could be deployed and executed on the DCGrid platform, and applications are enabled for the platform at the binary code level.

DCGrid contained an isolation technology, which provides full and unobtrusive protection for the grid as well as the underlying resources. DCGrid protected the desktop configuration, programs, and data from corruption by grid application errors as well as the privacy of desktop users from snooping. The grid application could not accidentally or intentionally access or modify the PC configuration or data files. Unlike other error-prone approaches, DCGrid presented a cleanly isolated, corruption-free environment. DCGrid shielded applications, proprietary data, and resources distributed to the desktop PCs by using encryption and tamper detection. Proprietary data and research sent out to hundreds of PCs in an enterprise could be protected from desktop user inspection or malicious corruption. DCGrid automatically monitored and limited grid work so it does not intrude on the PC user. DCGrid remained invisible at all times, never demanding inputs or responses from the desktop user, and never impacting the user's performance.

The approach is to automatically wrap an application in a virtual machine technology (Figure 5.5). When an application program is registered or submitted to the Entropia system, it is automatically wrapped inside the virtual machine. This isolation is called sandboxing. The application is contained within a sandbox and is not allowed to modify resources outside the sandbox. The application is fully unaware of being running within a sandbox, since its interaction with the OS is automatically controlled by the virtual machine. The virtual machine intercepts system calls the application makes. This ensures that the virtual machine has complete control over the applications' interaction with the operating system and access to the desktop resources.

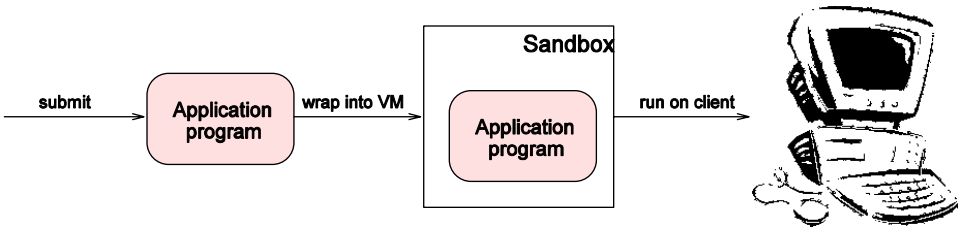


Figure 5.5 Entropia Sandbox Model

The Entropia system architecture consisted of three layers: physical management, scheduling, and job management. The physical node management layer, provided basic communication and naming, security, resource management, and application control. The second layer was resource scheduling, providing resource matching, scheduling, and fault tolerance. Users could interact directly with the resource-scheduling layer through the available APIs or alternatively through the third layer management, which provides management facilities for handling large numbers of computations and files. Entropia provided a job management system, but existing job management systems can also be used.

The physical node management layer of the Entropia system managed these and other low-level reliability issues. The physical node management layer provided naming, communication, resource management, application control, and security. The resource management services captured a wealth of node information (e.g., physical memory, CPU, disk size and free space, software version, data cached) and collected it in the system manager. This layer also provided basic facilities for process management including file staging, application initiation and termination, and error reporting. In addition, the physical node management layer ensures node recovery, terminating runaway and poorly behaving applications.

The security services employed a range of encryption and binary sandboxing technologies to protect both distributed computing applications and the underlying physical node. Application communications and data were protected with high-quality cryptographic techniques. A binary sandbox controlled the operations and resources visible to distributed applications on the physical nodes in order to protect the software and hardware of the underlying machine. The binary sandbox also regulated the usage of resources by the distributed computing application. This ensured that the application did not interfere with the primary users of the system without requiring a rewrite of the application for good behavior (Foster and Kesselman, 2004).

The resource-scheduling layer of Entropia accepted units of computation from the user or job management system, matched them to appropriate client resources, and scheduled them for execution. The resource-scheduling layer adapted to changes in resource status and availability and to high failure rates. To meet these challenging requirements, the Entropia system supported

multiple instances of heterogeneous schedulers. This layer also provided simple abstractions for IT administrators, abstractions that automate the majority of admins' tasks with reasonable defaults but allow detailed control as desired.

Entropia's three-layer architecture provided a wealth of benefits in system capability, ease of use by users and IT administrators, and internal implementation. The physical node layer managed many of the complexities of the communication, security, and management, allowing the layers above to operate with simpler abstractions. The resource-scheduling layer dealt with unique challenges of the breadth and diversity of resources but need not deal with a wide range of lower-level issues. Above the resource-scheduling layer, the job management layer dealt with mostly conventional job management issues. Finally, the higher-level abstractions presented by each layer did simplify application development. One disadvantage of the Entropia system was that it did not support heterogeneous systems. The only platform was Windows that limited the usability of this system in a research environment.

5.1.6 Condor

Condor, developed at the department of Computer Science, University of Wisconsin, Madison, is a High Throughput Computing (HTC) environment that can manage very large collections of distributive owned workstations (Litzkow and Mutka, 1998). This is a computing environment that delivers large amounts of computational power over a long period of time, usually weeks or months. In contrast, High Performance Computing (HPC) environments deliver a tremendous amount of compute power over a short period of time. In a high throughput environment, researchers are more interested in how many jobs they can complete over a long period of time instead of how fast an individual job can complete. HTC is more concerned to efficiently harness the use of all available resources.

The Condor environment is based on a layered architecture that enables it to provide a powerful and flexible suite of resource management services to sequential and parallel applications. Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor provides a powerful resource management by match-making resource owners with resource consumers. This is the cornerstone of a successful HTC environment. Other compute cluster resource management systems attach properties to the job queues themselves, resulting in user confusion over which queue to use as well as administrative hassle in constantly adding and editing queue properties to satisfy user demands.

Condor implements ClassAds, which simplifies the user's submission of jobs. ClassAds work in a fashion similar to the newspaper classified advertising want-ads. All machines in the Condor pool advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average, in a resource offer ad. A user specifies a resource request ad when submitting a job. The request defines both the required and a desired set of properties of the resource to run the job. Condor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied. During this match-making process, Condor also considers several layers of priority values: the priority the user assigned to the resource request ad, the priority of the user which submitted the ad, and desire of machines in the pool to accept certain types of ads over others.

5.2 Hierarchical Taxonomy

In this section we introduce our three-level hierarchical taxonomy on desktop grid systems. The first level refers to infrastructure and includes resource type, the platform that runs at the provider, scalability and security issues. The second one includes conceptual model, architecture and data model, under the umbrella of models. The last level concerns aspects related to software: application type, architecture of the support operating system, the need for administrator privileges, and whether a license is needed or not. At the end of this section, a table with the classification of the main desktop grid systems according to this taxonomy will be provided.

5.2.1 Level 1, Infrastructure: resource, platform, scalability, security

Resource type specifies how resources are provided to the system. There are two main trends: *volunteer* and *enterprise* resources. Volunteer desktop grid is based on voluntary participants, while enterprise desktop grid is based on non-voluntary participants usually within a corporation, research lab or university. Mostly, volunteer desktop grid is Internet-based, while enterprise desktop grid is LAN-based. Volunteer DG is more volatile, malicious, and faulty, whereas enterprise DG is more controllable because its resource providers are located in the same administrative domain. Typical examples of volunteer DG are SETI@home, BOINC, XtremWeb (XtremWeb, 2008), and Bayanihan (Bayanihan, 2008). Enterprise DG examples can be Entropia (Entropia, 2003) and Condor.

Desktop grids are classified based on the **platform** running on the resource provider. This can be *web-based*, where the applications are run into the web browser (can be Java applets or ActiveX controls), or *middleware based*, where the user must install a specific middleware application, that provides the functionality and services required to later execute computing applications on the provider's resource. In the web-based situation, the users only need to load

a specific web page, containing an applet, which is automatically downloaded and executed by the resource provider. Typical examples of such web-based systems are Bayanihan, Javelin, while middleware-based systems are SETI@home, BOINC, XtremWeb, Entropia and Condor.

Scalability divides desktop grids into two groups: *Internet-based* and *LAN-based*. Internet based desktop grids are characterized by anonymous resource providers, connectivity issues (firewall, NAT, dynamic addressing, possibly poor bandwidth and unreliable connection), possibly malicious resources, high security risks. In contrast, LAN-based desktop grids are characterized by more constant and reliable connectivity, lower security risks or under certain degree of control. Mainly, volunteer desktop grids fall in the first group, and enterprise desktop grids in to the second one.

Security in desktop grids deals with aspects of access to the computational resources by using some form of authentication and authorization; and access to the computational data, input and results, by providing data integrity and encryption. The verification of results is also an important issue that needs to be addressed in any volunteer computation. Hardware and software mishaps as well as malicious volunteers can falsify the outcome of computations, rendering the results useless. Thus, a major concern of middleware tools supporting volunteer computation is to provide results validation and sabotage tolerance mechanisms. Since computations are run in open and non-trustable environments, it is necessary to protect the integrity of data and to validate the computation results. Without a sabotage detection mechanism, a malicious user can potentially undermine a computation that may have been executing for weeks or even months. In contrast, applications executed over more controlled clusters offer some reliability and trustability.

5.2.2 Level 2, Models: computing model, architecture, data model

According to the **computing model** we can group desktop grids into two main categories: one is the typical, *master-worker computing model*, consisting of independent tasks, and the other one involves *parallel paradigms* with communication between the tasks. The master-worker (M-W) model includes a master (server) process which sends tasks to a set of worker processes, then each worker makes some kind of computation on some tasks, a computation that generally requires a variable and unpredictable time. The master then waits for the answer from each individual worker before sending a new task to that worker. This is a typical form of embarrassingly parallel pattern, where tasks are mutually independent, and can be executed in parallel. The other category involves tasks which depend on each other: there is either an execution flow between the tasks, such that one task needs to be executed only after other tasks are finished (typically accomplished using some sort of task-dependency

graph), or the tasks are run in parallel, with data communication between each task (typical paradigms involved are PVM, MPI, BSP).

Desktop grids can be categorized into *centralized*, *hierarchical* and *peer-to-peer* (distributed) according to the **architecture** of the components of each system. A centralized DG consists of a central server, where resource providers donate computing resources during their idle time, and job submitters send their computing requests (jobs). Usually a job is divided into smaller, independent computing units, called tasks, with their own input data. The server distributes these tasks to the available resources, based on some scheduling algorithm. Typical examples are BOINC, XtremWeb, and Entropia etc. In a hierarchical DG, desktop grids on the lower level can ask for work from higher level, or vice versa, desktop grids on the higher level can send work to the lower levels. The control of work at the higher level can be realized with priority handling at the lower level. A basic DG can be configured to participate in a hierarchy, that is, to connect to a higher-level instance of DG (parent node in the tree of the hierarchy). When the child node (a stand-alone desktop grid) has less work than resources available, it asks for work from the parent. The parent node can see the child as one powerful client. An example of such hierarchical DG is the SZTAKI Desktop Grid (SZTAKI, 2008).

In a peer-to-peer DG, there is no central server, in contrast with the centralized type. Resource providers have only partial information of other providers. They are also responsible for constructing the computational overlay network and for scheduling a job in a distributed way, according to each other's capability, availability, reputation or trust. The reliability and performance of such P2P systems depend on how the overlay network is constructed, because there is no reliable central server. Examples of such systems are CCOF, Messor, Paradropper, and Organic Grid.

Data model concerns classifying of desktop grids based on how computational data (both input and output data) is transferred between the components of the DG. We are concerned here with data communication between job submitter and resource provider on one hand, and between different resource providers on the other hand, in the situation when communication between running tasks is required (parallel models). We identified three data model types: *middleware*, *data servers*, and *direct communication*. In the first situation, using the middleware, which connects the two components, transfers data. This could be the master (server) in a centralized configuration, or all the involved nodes in a P2P configuration. The downside of this approach is that there could be a bottleneck in the case of large data sets involved, which could affect other communication between the components (control, discovery, status, etc.).

In the data server model, all the data is transferred using another type of component in the system: a data server, which is a repository of both input and output data. In this case, the job submitter is responsible for uploading the input data to the data server, and for retrieving the results, while the job

running on the resource provider's computer is responsible for downloading the input data and storing the results on the server after finishing the job. This model has the advantage of moving the burden of data transfer wrt communication and complexity from the central node to a more dedicated, and optimized component. However, there is a complexity added in maintaining such a data server, which in some situations might not be necessary.

The third data model involves direct data communication between the components. This could be done either by using a common network file system, where each component has access to it, by using a distributed file sharing mechanism (P2P Bittorrent), or by using lower lever network based communication for data transfer. The type of direct data communication could be chosen based on the amount of data transferred, and the frequency with which data transfers occur. We can also have the situation when the submitted job contains also the input data for the computation.

5.2.3 Level 3, SW.: application, architecture, administration, license

SW applications to be run on desktop grids can be of different types: legacy applications that already exist and are inherited from languages, platforms, and techniques earlier than current technology. Most enterprises that use computers have legacy applications that serve critical business needs. In order to run such application, some kind of virtualization could be necessary, depending on the complexity of the application and the resources it needs (third party applications or libraries, file system access, specific operating system, etc.). This could range from simple, virtual file systems, to more complex virtual environments (virtual machines emulating an operating system).

Another class includes applications written in a high level or interpreted programming language, like Lisp, Perl, Java, where in order to run the program, a specific run-time environment should be present. The computing jobs must be distributed according to each processing resource's capabilities, and provide the appropriate starting mechanism. Web-based and Java-based systems have their own drawbacks, e.g. the historically slow execution speed of the Java Virtual Machine (JVM) that executes the platform-independent bytecode. Another problem comes from the security restrictions imposed on Java applets that prevent them to access local storage space or communicating with machines other than the host from which they came. Together, these two problems may limit the performance and scalability of Java-based systems.

A whole class of application includes those where the programs could be compiled in a programming language (C/C++, Fortran), and where additional support for desktop grids could be included. This allows fine tuning the application in term of computing performance, but requires an API from the DG to be provided. This includes also parallel applications, where different communication paradigms are required (message passing, shared memory,

etc.). A last type of applications concerns lightweight programs that are highly optimized for performance and DG specific. For example, computational applications could be made in form of plugins (or shared libraries), which contains only the computational problem, the rest of the communication, file access, and other access to resources are handled by the supporting middle layer application running on the resource provider. In this case, more complex abstraction and API are needed from the underlying DG system.

SW architecture in desktop grids concerns with the operating system of different components of the system. This could be Linux, or other Unix versions (BSD, IRIX, etc.) when resource providers are nodes from a cluster, or Linux, Windows, Mac if resource providers are desktop computers. Thus, a desktop grid system should have support for the different operating systems. Many desktop grid systems are based on Java for portability.

SW administration - during the QADPZ development, we have learned that another restriction of existing systems, especially middleware based, is that each resource provider needs to install a runtime module as administrator. This poses some issues regarding data integrity and accessibility on providers' computers. QADPZ tries to overcome this by allowing the middleware module to run as a non-priviledged user to the local system.

SW license can be necessary if the desktop grid system is a commercial one or not, in case of open source software systems. Beneath a table with the classification of the main desktop grid systems according to the above-introduced taxonomy is presented (Table 5.1).

	Infrastructure	**Models**	**Software**
DG system	**Resource** **Platform** **Scalability** **Security**	**Computing model** **Architecture** **Data comm.. model**	**SW application** **SW platform** **SW administration** **SW license**
distributed.net	- volunteer - middleware - Internet - trust	- master-worker (M-W) - centralized - data server	- set of dedicated only - all OS - non admin - closed
Entropia	- volunteer - middleware - Internet - trust	- master-worker - centralized - data server	- set of dedicated only - Windows - non admin - closed
SETI@home	- volunteer - middleware - Internet - trust	- master-worker - centralized - data server	- set of dedicated only - Linux, Win, Mac - non admin - closed
Bayanihan	- volunteer - web-based - Internet - Java sandbox	- master-worker - centralized - middleware	- Java applet - all OS (Java) - non admin - open source

Condor	- enterprise - middleware - LAN, Internet - authentication	- M-W, PVM, MPI - centralized, (hierarchical) - file system	- legacy, script, compiled - Linux, Win, Mac - admin - license
XtremWeb	- enterprise - middleware - LAN, Internet? - authentication	- M-W, MPI - centralized, (hierarchical) - middleware	- Java applet - all OS (Java) - admin? - open source
QADPZ	- enterprise - middleware - LAN, Internet - authentication	- M-W, MPI, PVM - centralized - file system, data server	- legacy,script,compiled, lightweight - Linux,Win,Mac,Unix - non admin, admin - open source
BOINC	- enterprise - middleware - LAN, Internet - authentication	- M-W - centralized - data server	- legacy, script, compiled - Linux,Win,Mac,Solars - admin - open source
SZTAKI LDG (BOINC based)	- enterprise - middleware - LAN, Internet - authentication	- M-W - hierarchical - data server	- legacy, script, compiled -Linux,Win, Mac, Solaris - admin - open source
Javelin Javelin++	- volunteer - web-based - Internet - Java sandbox	- M-W - centralized - middleware	- Java applet - all OS (Java) - non admin - open source

Table 5.1. Classification of the main DG systems according to the taxonomy

Hints to choose the most suitable DG for a given problem: if we consider the four scenarios that have been presented in Section 4.2.2.5 and try to decide what is the best DG for each scenario, we first look at the first column, first entry (resource) and if the project is requested to have robustness and reliability (major issue for first 3 scenarios) we would better choose the enterprise DG as it overcome the volatility of volunteer computing. More, it has accountability and, depending on the type of the organization, lacks anonymity (except for universities or alike organizations). On the second choice (middleware vs. web-based), if the ensuring of control and security is crucial we should go for middleware platform (first 3 scenarios), while for the 4th scenario we could use both. Though, we must remind that the enterprise desktop grid is limited in power, and the volunteer computing has virtually unlimited resources.

As for the scale and security, probably the best option for the first two problems is the LAN solution as it ensures privacy and keeps the secrets of the application away from un-authorized eyes. The last two, on the other hand can go both ways. The models from the second column are strongly influenced by the nature and complexity of the application. One choice would be suitable in the case of an application that can be broken in small tasks that can run parallel, with no communication between them (master-worker), and another for a

different type of application, in which tasks can communicate with each other (Message Passing Interface - MPI). Moreover, if the application needs a huge computational power, we would probably prefer a hierarchical DG, as it can borrow power from third parties.

As for the data communication model, one has to consider the difficulty of developing the software that will manipulate the data and the technical limitations (within a virtual file system vs. "back and forth" from a data server). The main difference in the usage of institutional DGs relatively to public ones lies in the dimension of the application that can be tackled. In fact, while public projects usually embrace massive applications made up of an enormous number of tasks, institutional DGs (much more limited in resources) are better matched for small size applications. So, whereas in public volunteer projects importance is on the number of tasks carried out per time unit (throughput), users of institutional desktop grids are normally more interested in a fast execution of their applications, seeking fast turnaround time.

The last column is easier to work with as many of the issues involved here are known before starting to solve a given problem: we have our own application or we want to run a pre-defined one, if we have our own, which kind it is (Java applet, legacy, script etc.), what platform we use (Linux, Windows, Mac etc.), what are the needed administration privileges (admin or user), whether we are interested in access to the source code or not, and finally if we need a desktop grid for which a commercial license is requested.

6 Conceptual Model

6.1 Introduction

This chapter describes the framework that was developed with the purpose of using distributed computing for large-scale Scientific Computing and Visualization. The framework is based on the master-worker paradigm, where worker nodes download small tasks from a central master node, execute them, and send back the results to the master. Some of the disadvantages in using this model in a heterogeneous and dynamic environment are described, together with some problems in using it for visualization purposes. Improvements to this model are presented, which are meant to increase the performance and efficiency. The idea of using dynamic creation of subtasks is presented. Subtasks are generated according to the problem's requirements, taking into consideration the available performance parameters of the system (network bandwidth, latency, CPU availability and performance).

6.2 The Master-Worker Model

Our conceptual model is based on the *master-worker* paradigm. The master-worker computing paradigm also called *replicated worker computing* is built on the observation that many computational problems can be broken into smaller pieces that can be computed by one or more processes in parallel. That is, the computations are fairly simple consisting of a loop over a common, usually compute-intensive, region of code. The size of this loop is usually considered to be long. In this model, a number of worker processes are available, each capable of performing any one of the steps in a particular computation. The computation is divided into a set of mutually independent *work units* by a master node, as it can be seen in Figure 6.1. Worker nodes then execute these work units in parallel. A worker repeatedly gets a work unit from its master, carries it out and sends back the result. The master keeps a record of all the work units of the computation it is designed to perform.

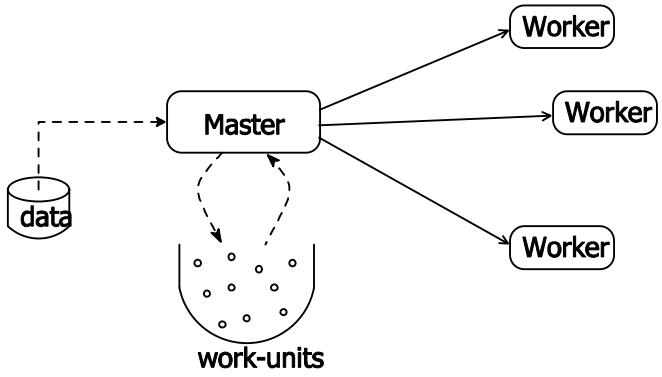


Figure 6.1 Master-Worker model

As each work unit is completed by one of the workers, the master records the result. Finally when all the work units have been completed, the master produces the complete result. The program works in the same way irrespective of the number of workers available - the master just gives out a new work unit to any worker who has completed the previous one.

Whereas the master worker model is easily programmed to run on a single parallel platform, running such a model for a single application across distributed machines presents interesting challenges. On a parallel platform, the processors are always considered identical in performance. In a distributed environment, and especially in a heterogeneous one, processors usually have different types and performance. This raises the problem of load balancing of work-units between the workers in such a way to minimize the total computing time of the application.

The ideal application is coarse-grain and embarrassingly parallel. Granularity is defined as the computation-to-communication ratio, with coarse grain applications involving small communication time compared to computation time, and fine grained application requiring much more time for communication than computation. Coarse-grain applications are ideal for desktop grid computing because most such computing systems employ commodity network links, which have limited bandwidth and high latencies. Embarrassingly parallel applications are those problems that easily decompose into a collection of completely independent tasks. Examples of such scientific problems are: genetic and evolutionary algorithms, Monte Carlo simulations, distributed web crawling, image processing, image rendering.

In a heterogeneous environment, *scheduling* that includes both problem decomposition and work-unit distribution (placement to workers), has a dramatic effect on the program's performance. An inappropriate decomposition or distribution decision can result in poor performance due to load imbalance. Effective scheduling in such heterogeneous environments is difficult. We will show that this problem can be overcome using relatively simple heuristics if appropriate mechanisms are provided to the scheduler in order to determine the computation and communication complexity of the problem. This information is then used to decompose the problem and schedule the work-units in a way that provides good load balance, and thus good performance.

6.2.1 Decomposition and Distribution of Work-units

The most important part of parallel programming is to map out a particular problem on a multiprocessor environment. The problem must be broken down into a set of tasks that can be solved concurrently. The choice of an approach to the *problem decomposition* depends on the computational scheme. A parallel program is only useful if it scales efficiently with the number of processing elements, in terms of reduced runtime. For the problem's decomposition, this means enough tasks are needed to keep all the processing elements busy with

enough work per task to compensate for overhead incurred to manage dependencies and communication. The drive for efficiency can lead to complex decompositions that lack flexibility. There are two different ways of decomposing a parallel problem, based on the way and time when the work-units are created: static and dynamic.

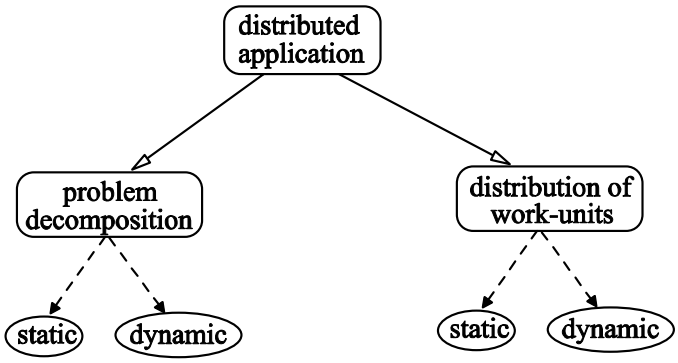


Figure 6.2 Decomposition and Distribution of Work-units

Static decomposition - the master generates all the work-units in the beginning of the computation, as it is shown in Figure 6.3.

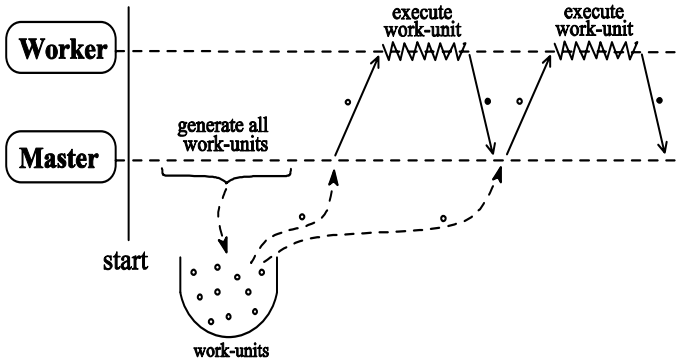


Figure 6.3 Static decomposition strategy

Dynamic decomposition - not all work-units can be generated in the beginning; instead, the computation starts with a small number of work-units, and later new work-units are created, depending on the results of already executed work-units; the master can create or delete dynamically work-units.

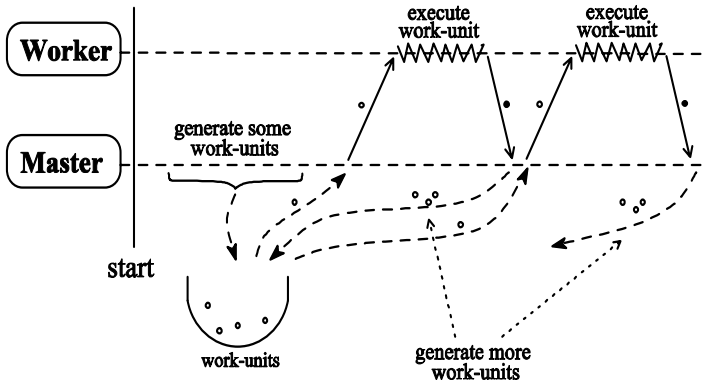


Figure.6.4 Dynamic decomposition strategy

There are applications where an easy, static decomposition is not possible, and a more complicated dynamic decomposition is necessary. This could lead to complex decomposition schemes that lack flexibility. The decomposition needs to be complex enough to get the job done, but sufficiently simple to allow easy maintenance of the application. After decomposing the problem, the work-units need to be distributed to the work-units, or *scheduled for execution*. The key to making a parallel program work well is to schedule their execution so that the load is balanced between the processing elements. Distribution of work-units to the workers can be of two types:

Static distribution - the master processor decides on the distribution of work at the start of the computation, by assigning the work-units to the workers; this is suitable in those situations where the relative amount of time required for each work-unit is known and the workers have a well known and stable load. This method works when it is possible to statically determine how many work-units to assign per worker in order to achieve a balanced load.

Dynamic distribution - the distribution of work-units varies between workers as the computation proceeds; this is a good strategy when the execution time of each work-unit is unpredictable, especially when the processing elements are different or when the amount of load that can be supported by each worker is unknown and possibly changing. The most common approach used for this is to use a queue of work-units at the master; after execution, each work-unit is removed from the queue. Workers which are faster or which receive work-units with shorter execution times will get more work-units.

The static distribution approach is more suited to homogeneous environments, where all processing elements are the same, and the work-units have a similar execution time. In contrast, dynamic distribution is most suited to heterogeneous environments, with applications where each individual work-unit can have a different execution time. This strategy works also in the case where the number of workers is changing during the computation.

6.2.1.1 Static decomposition, static distribution

We describe first the simplest master-worker algorithm. We consider a fixed, known from the beginning, number of workers. The problem is decomposed into a fixed number or work-units, usually dependent on workers' number (Figure 6.5). The decomposition is made considering that each worker has the same processing power and that each work-unit requires the same computing time. All work-units are handed out to the workers at the beginning of the overall computation. The workers start executing work-units, and will contact the master each time when finish execution of a individual work-unit, to send back the results of the computation. The master will assemble all partial results received from the workers into the final result of the application. The algorithm is described in the pseudo-code below.

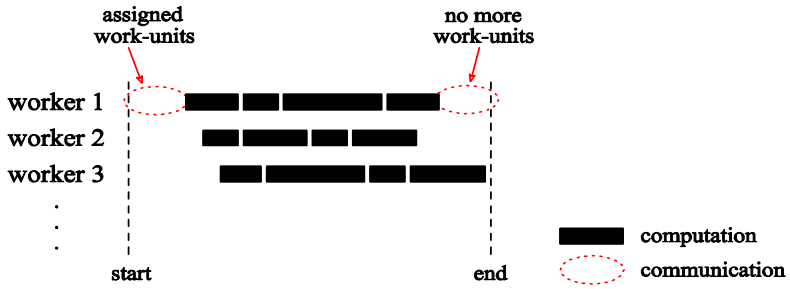


Figure 6.5 Computation times on workers:
static decomp-static distrib

Master - static decomposition, static distribution

```
read data
create all work-units
assume fixed number of workers
assign equally work-units to workers
FOR each worker
        send all assigned work-units to worker
ENDFOR
WHILE not all results received
        receive result from worker
        process result
ENDWHILE
assemble final result
```

Worker - static decomposition, static distribution

```
receive all work-units from master
FOR each work-unit
        execute work-unit
        send result to master
ENDFOR
```

The simplicity of the algorithm makes it very easy to implement. Each worker knows at the beginning exactly what it needs to compute, and doesn't require additional communication with the master to get new work-units. This makes the algorithm very efficient, by minimizing the time spent in communication and maximizing the total time spent on doing computation at the workers. Unfortunately, the algorithm works only for a limited number of applications and cannot be used in many situations. The algorithm is not very flexible, especially in a heterogeneous environment, where different processors can have very different computing power, such that the same work-unit can take different amounts of time to compute on different workers. Quite often, in many applications, work-units have different computing time, even on the same processor. This could result in a large imbalance in the computation time spent by different workers. Another disadvantage of the static algorithm is that it cannot handle a dynamic pool of workers. This situation can occur when computing power is harvested from the idle CPU cycles of desktop computers. Available workers can appear and/or disappear, thus the total number of workers, which can be used, is varying over time.

Visualization algorithms that can use this type of master-worker are those, which are easily decomposed into independent tasks. Here, we can mention ray tracing and volume visualization. Ray tracing is a widely used technique to generate realistic looking images on a computer, and is recognized as a powerful technique. Rays are reflected and refracted according to the reflectivity and transparency of the surfaces. The process is repeated recursively with the reflected or refracted rays changing the light intensity at all intersection points. However, the ray tracing techniques require heavy computing power, since they deal with a large number of floating-point calculations for the movement of millions of rays. The required computing power increases especially sharply when many objects are needed to be rendered. Parallelism inside a ray tracing algorithm is observed in computing individual rays. The goal is to distribute all pixels into a number of processors in an efficient manner.

6.2.1.2 Dynamic decomposition, static distribution

A more advanced master-worker algorithm is needed in the case of dynamic decomposition of the problem. We still consider a fixed number of workers, known from the beginning. The problem however is considered as not possible to be decomposed into all work-units from the beginning. This can either because not all the work-units are known from the beginning, or because there are too many of them, and would be inefficient to store them in memory at once. Instead, only a smaller subset of work-units is generated, based on the number of available workers. The master starts by sending out one work-unit for each worker, then waits for the results. Once a worker finishes its work-unit, it sends the results back to the master and requests a new work-unit. The master receives the result and sends a new work-unit to the worker (Figure 6.6).

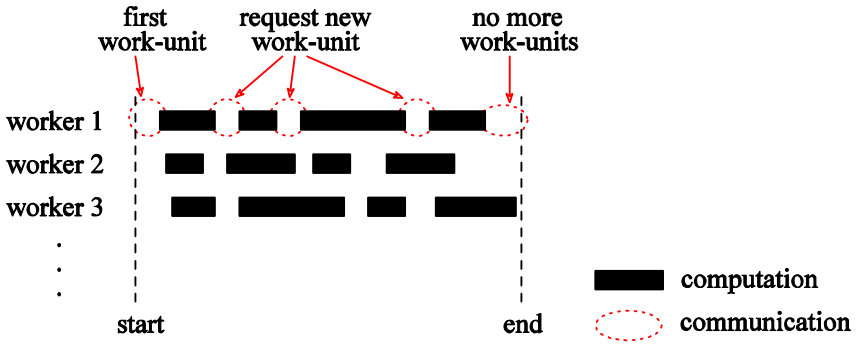


Figure 6.6 Computation times on workers: dyn. decomp-static distrib

Based on this result, the master can also generate new work-units or delete existing ones. After the workers execute all work-units, the master assembles the final result and notifies all the workers about the termination of the application. The algorithm is described in pseudocode below:

Master - dynamic decomposition, static distribution

```
read data
create a set of work-units
assume fixed number of workers
FOR each worker
        send one work-unit to worker
ENDFOR
WHILE not all results received for existing work-units
        receive message from worker
        IF message is request
                send one work-unit to worker
        ELSIF message is result
                process result
                IF necessary
                        create new work-units
                ENDIF
        ENDIF
ENDWHILE
FOR each worker
        send stop message to worker
ENDFOR
assemble final result
```

Worker - static decomposition, static distribution

```
REPEAT
        send request for work-unit to master
        receive message from master
        IF message is work-unit
```

```
            execute work-unit
        send result to master
        ENDIF
    UNTIL message is stop
```

Work units are handed out to the workers upon request. Each time a worker is idle, it sends a request for new work-unit to the master. The master answers by sending a work-unit from the pool. When receiving the results back from the workers, the master can create new work-units, or delete existing ones. The master will give work-units to the workers until all are solved and no more new work-units are created. The master assembles the final result and notifies all workers to stop. The advantage of this algorithm is that it provides certain amount of load balancing. Each time a worker is idle, it requests a new work-unit. This way the workers are doing computation most of the time. However, there is a waiting period of time before starting each work-unit, after the request is sent to the master. This waiting time depends on many variables, for example how busy the master is, how large the work-unit messages are, etc. The algorithm is more complex than the previous one, however many more applications are suitable for this model. Visualization algorithms that can use this type of master-worker are for example line integral convolution.

6.2.1.3 Dynamic decomposition, dynamic distribution

We consider now the situation where the number of workers is changing during the computation. Often, when using idle computational power of desktop computers, the availability of individual computers can vary over time. In our master-worker model it means that new workers can appear in the system and/or other workers can disappear. This can happen for example when the owners are using computers, or when computers are started or stopped. The algorithm is described in pseudocode that is presented beneath.

```
Master - dynamic decomposition, dynamic distribution
    read data
    create a set of work-units in state "new"
    WHILE not all results received for existing work-units
        t = time interval to next timeout
        wait(t) for message from worker
        IF timeout from wait
            change all timed-out work-units from state "exec" to state
        "new"
        ELSE IF message is requested
            send one "new" work-unit to worker
            change work-unit to state "exec" and set timeout interval
            ELSIF message is result
```

```
                    process result
                    remove work-unit
                    IF necessary
                        create new work-units
                    ENDIF
                ENDIF
            ENDIF
        ENDWHILE
        FOR each worker
            send stop message to worker
        ENDFOR
        assemble final result
```

Worker - dynamic decomposition, dynamic distribution

```
    REPEAT
        send request for work-unit to master
        receive message from master
        IF message is work-unit
            execute work-unit
            send result to master
        ENDIF
    UNTIL message is stop
```

The algorithm is an extension of the one presented in the previous section, with static distribution of work-units. The problem is split up into work-units dynamically, by generating a few work-units at the beginning and more during the processing. Work-units are sent to each worker upon request, when these are available for computation. The algorithm takes into account the fact that workers can become available during the computation. This can happen either after a work-unit has been successfully processed, or in the middle of its computation. In order to be able to deal with this situation, the master assigns to each work-unit a timeout interval for processing. This is done when it is sent to a worker. If the master doesn't receive the result from the worker after this timeout interval, the work-unit is resent to another worker.

The work-units can have two possible states:

- **new** - a work-unit has the state *new* when it is created or when it is timed-out (i.e. it was sent to a worker, but the result was not received in the specified amount of time);
- **exec** - a work-unit has the state *exec* when it is sent to a worker for processing; the work-unit gets also associated a timeout interval for processing.

The algorithm is able to handle the situation of having a variable number of workers available. This is very useful when workers are ordinary desktop computers. The algorithm is assigning a timeout interval to each work-unit when sent to a worker, which has to complete the computation within this interval. The problem, which arises here, is in deciding the value of this timeout. The simplest solution is to use a fixed, predetermined interval, based on an estimation of computation time on the worker. This, however, is not always possible. A special case is when having a heterogeneous environment, with workers having different processing power. Assigning the same timeout interval to work-units assigned to slower and faster workers is not a good choice. The master needs to detect in time that a worker has failed to compute a work-unit, so that it can send it to another worker (Figure 6.7).

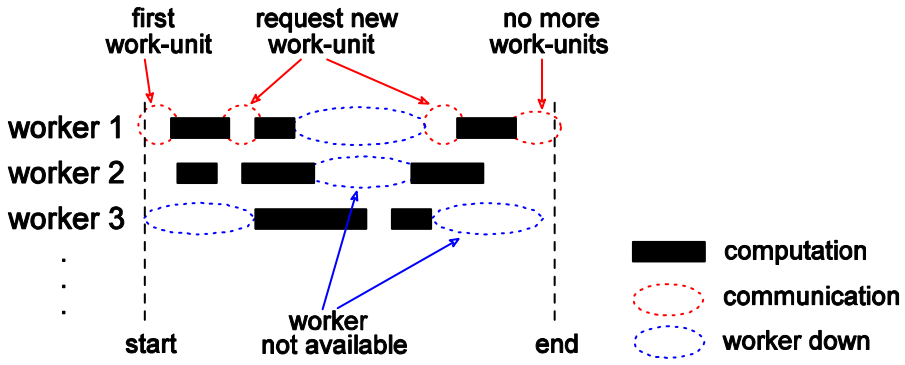


Figure 6.7 Computation times on workers: dyn. decomp-dyn. distrib

A disadvantage of the algorithm consists in its increased complexity on the master side. The master has to keep track of each work-unit, its completion or failure. This might increase the CPU time used by the master itself for managing the work-units, creating a possible bottleneck in the system. Optimized data structures are required for an efficient implementation.

6.3 Improved Master-Worker Model

We present further an improved version of the master-worker model. The model is based on the algorithm with dynamic decomposition of the problem and dynamic number of workers. The improvements concern increasing the performance of the original model, by increasing the time workers are doing computations, and decreasing the time used for communication delays. This is achieved by using different techniques, such as pipelining of the work-units at the worker, redundant computation of the last work-units to decrease the time to finish, overlapped communication and computation at the workers and the master, use of compression to reduce the size of messages. We will describe in the following subsections each of these techniques.

We define the efficiency of a worker as being the ratio between the amount of time the worker spends doing computation and the amount of time the worker is available for doing any work:

$$E = \frac{t_{exec}}{t_{available}} = \frac{t_{exec}}{t_{exec} + t_{communic}}$$

We will use the efficiency as a measure to compare the improved master-worker model with the original master-worker model. The new model tries to improve the efficiency of the workers.

6.3.1 Pull vs. Push for work-units

In the original master-worker model, each time a worker finishes a work-unit, it has to wait until it receives the next work-unit for processing. In situations where this communication time is comparable with the time needed for executing a work-unit, the efficiency of the worker is reduced very much. The time intervals used for communication and computation (processing) are described in Figure 6.8.

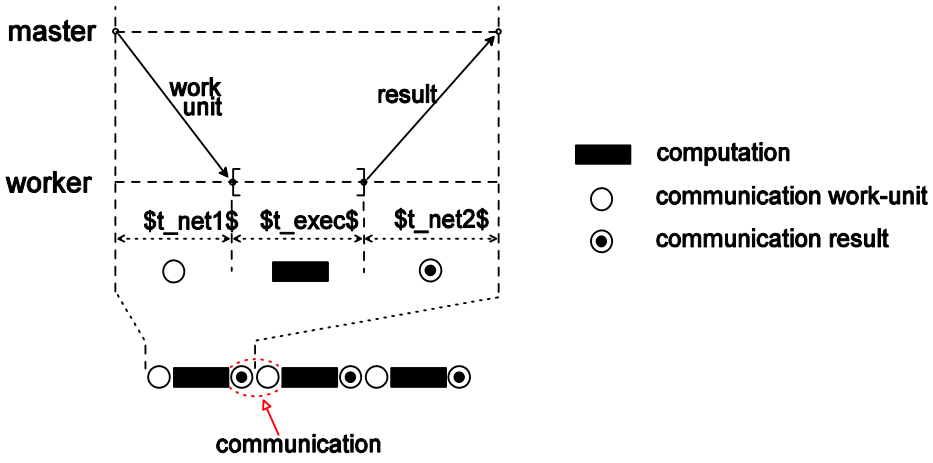


Figure 6.8 Worker timeline in execution

The master-worker model is using the *pull technology*, which is based on the *request/response paradigm*. This is typically used to perform data polling. The user (in our case the worker) is requesting data from the publisher (in our case the master). The user is the initiator of the transaction. In contrast, a *push technology* is using a different approach, which relies on the *publish/subscribe/distribute paradigm*. The user subscribes once to the publisher, and the publisher will initiate all further data transfers to the user. This is better suited in certain situations. We first extend the master-worker model by replacing the pull technology with the push technology, as it is illustrated in Figure 6.9. In this model, the worker doesn't send any more requests for work-units. Instead, it first announces its availability to the master when it starts, and

the master is responsible for sending further work-units. The workers just wait for work-units, and process them when received. At the end of each work-unit, it sends back the results to the master. The master will further send more work-units to the worker. This moves all decisions about initiating work-units transfers to the master, allowing a better control and monitoring of the overall computation. The use of the push technology also allows further improvements to the master-worker model and will be detailed in the following sections.

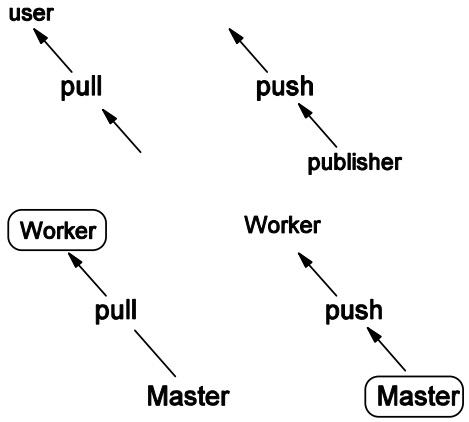


Figure 6.9 Pull vs. Push technology

6.3.2 Pipelining of work-units

Reducing the total time spent in waiting for communication can increase the efficiency of the worker. One method to do that is to use work-units pipelining at the worker, thus making sure that the worker has a new work-unit available when it finishes the processing of the current work-unit. Pipelining is achieved by sending more that one work-unit to the workers, as shown in Figure 6.10.

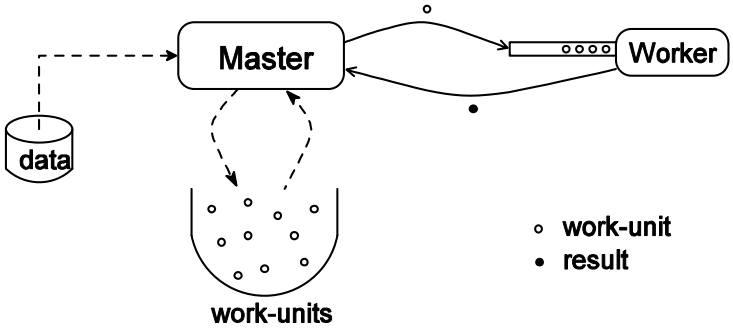


Figure 6.10 Pipelining of worker tasks

Each worker will have at least one more work-unit in addition to the one being processed at that worker. This is done so that the worker, after finishing a work-unit, will have ready the next one for processing. In the beginning, the master sends more than one work-units to the worker, then after each received result, sends another work-unit to be queued on the worker. The worker does

not need to wait again for a new work-unit from the master after sending the result, the next work-unit being already available for processing.

The immediate advantage of pipelining is that the waiting time for a new work-unit is eliminated. This is described in Figure 6.11. While the worker is processing the next work-unit, a new work-unit is sent by the master and is queued in the operating system. When using non-blocking communication, the waiting time for sending the result to the master after finishing a computation can be also eliminated.

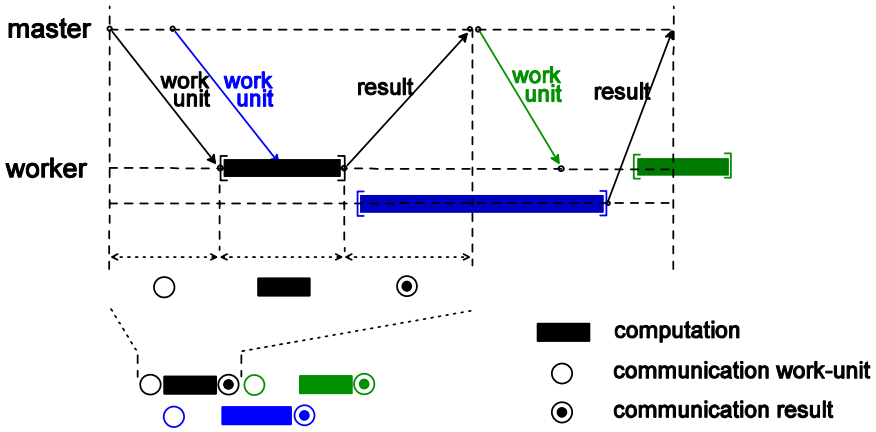


Figure 6.11 Worker timeline for unit pipeline

Keeping one new work-unit available at the worker seems to be enough to reduce the waiting time for communication. However, there is a situation when this is not adequate. It can happen that the execution time of a work-unit is much shorter than the communication time (consisting of sending back the result and receiving the new work-unit). In this case, the worker finishes the current work-unit, but the new one is not yet received. Thus, a certain waiting time is involved for receiving it (see Figure 6.12).

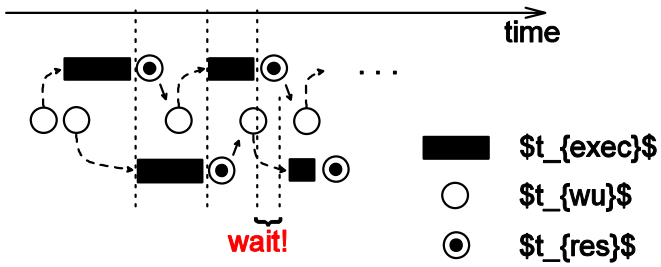


Figure 6.12 Unit pipeline - worst case

If there are many work-units with short execution times, than the overall waiting time can increase significantly, reducing the efficiency of the worker. The condition for this not to happen is the following:

$$t_{wu} + t_{res} \leq t_{exec} \quad \text{for the average time values, or}$$

$$\sum_{work-units} \left(t_{wu} + t_{res}\right) \leq \sum_{work-units} t_{exec} \quad \text{for the individual time values.}$$

This situation can be improved by pipelining more than two work-units at the worker, thus using a larger pipeline. The master starts by sending out a number of work-units to the worker to fill the pipeline. Each time a result is received back from the worker, the master sends a new work-unit, thus keeping the pipeline full. This algorithm works as long as the average execution time for a work-unit is larger than the average communication time for sending a result and a new work-unit between the worker and the master. If the communication time is too large, the pipeline will eventually become empty and the worker will need to wait for new work-units.

6.3.3 Sending more work-units at a time

To overcome this situation, the master needs to send more than one work-units per each message. The master starts by sending a message containing more than one work-unit, and then keeps sending them as long as the pipeline is not full. Each time it receives a result, it sends another work-unit, to compensate the decreasing number of work-units from the pipe. If the worker sends only one result per message back to the master, and this only one new work-unit, then eventually the pipeline will become empty. In order to prevent that, the worker will need to send back more results at a time.

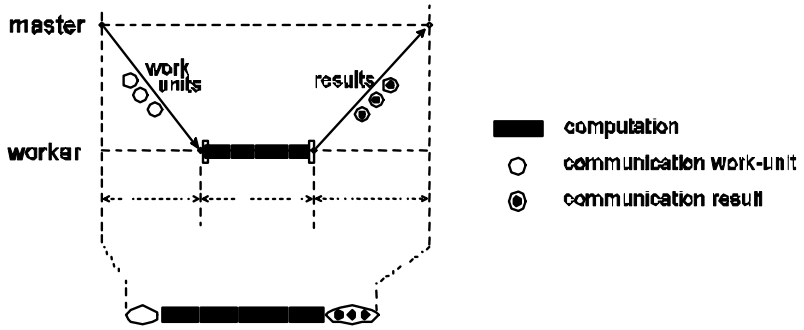


Figure 6.13 More results at a time

We could consider, for example, that the number of results per message is equal to the number of work-units per message sent from the master. In this case, all results from the work-units, which came in one message, are sent back to the master the same way in one message after all of them were successfully computed. This would solve the previous problem if the time to send a larger message (with more work-units) is much smaller than the time to send the

individual messages (for each work-unit). This is usually possible if the data required to describe one work-unit is small enough, so the messages are kept short. However, it could still happen that communication time is larger than the execution time, so that the worker will end up waiting for new work-units. The condition for this not to happen is the following:

$t_{wu,n} + t_{res,m} \leq t_{exec,n}$ for the average time values of multiple work-units per message and execution.

6.3.4 Adaptive number of workers

As mentioned before, in a heterogeneous environment based on idle desktop computers, the total number of workers available could be changing during the computation. New workers can register to the master, and other can become temporarily unavailable. The master controls the total number of workers used for computation, since he is the one sending out work-units to the workers. If necessary, the master can choose not to use all the available workers for computation, only a few of them. This might be for different reasons, as described further. In the master-worker model, the master can become a bottleneck, especially when there are a lot of workers, which are connecting to get work-units and send results back. Overloading the master could cause the bottleneck. Because the master has also to do a small amount of processing each time when it receives results from the workers, if too many workers connect to the master, the processing resource available might not be enough and the request will be delayed. There is an upper limit on the number of workers, which can connect to the master without overloading it. Finding out this number is not easy at all, and it depends on a variety of parameters from the entire system: computational capabilities of the workers and the master, communication delays, the amount of processing involved for each results, etc.

Another bottleneck in the system could be caused by too much communication. Considering that there is enough computational power on the master to serve a large number of workers, it could happened that there are too many messages exchanged between the master and workers, thus communication delays can occur. This might happen either because there are too many messages per time unit, or because the amount of data transferred is too high, exceeding thus the available network bandwidth. On the contrary, if there is too few workers used, then the total execution time of the computation will be too large, not exploiting all the resources available. This suggests that there is some optimum for the number of workers, which can increase the overall efficiency of the whole computation, and reduce the time to complete it.
We define the overall efficiency of the computation as being the ratio between the total amount of time since the beginning of the computation and the sum of execution times for all completed work-units on all workers:

$$E_{system} = \frac{t_{parallel}}{t_{serial}} = \frac{t_{total}}{\sum_{work-units} t_{exec}}$$

We propose an adaptive algorithm for the number of workers, based on performance measures and estimates of execution times for work-units and communication times for sending/receiving messages. The number of workers used is automatically reduced if the efficiency of the computation decreases. We employ a heuristic-based method that uses historical data about the behavior of the application. It dynamically collects statistical data about the average execution times on each worker.

6.3.5 Adaptive timeout interval for work-units

In the master-worker algorithm described for the dynamic number of workers in pseudocode, the suggested approach for selecting the timeout interval for the work-units was to fix it in the beginning of the computation for each worker. We propose here an adaptive algorithm for changing dynamically this timeout value for each individual worker. Each new timeout value is based on the average processing times for the last work-units at that worker.

The processing time for each work-unit is the time interval from the point the work-unit is sent out to the worker and until the result is received back. It consists of the communication times used for sending the work-unit and receiving back the result, plus the execution time of that work-unit. The timeout is recalculated each time a new result is received from the worker. Each result message will carry in addition the effective execution time for that work-unit on that particular worker.

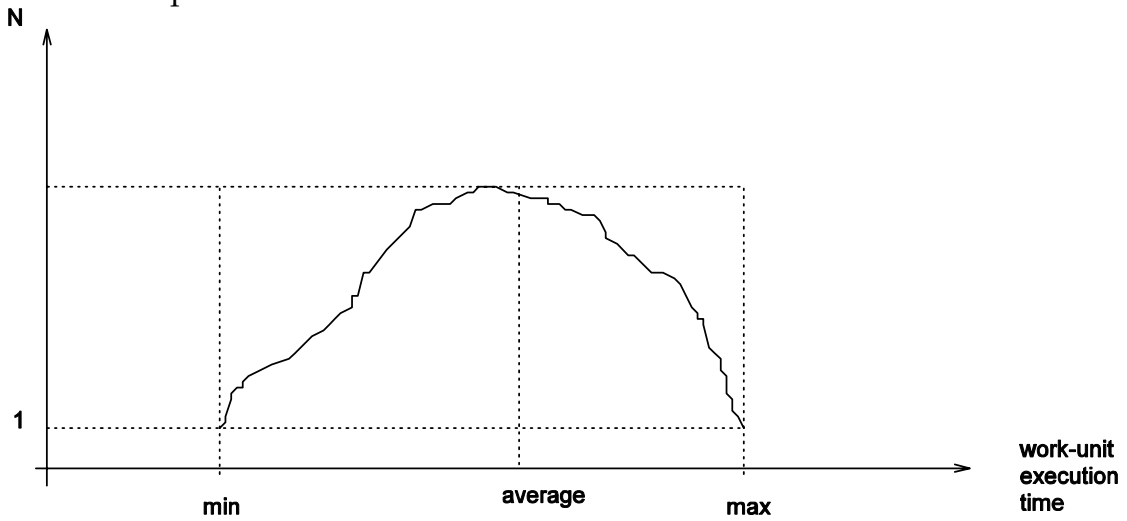


Figure.6.14 Model timeout history

6.3.6 Use of multithreading

The multithreaded programming paradigm allows the programmer to indicate to the run-time system, which portions of an application can occur concurrently. Synchronization variables control the access to shared resources and allow different threads to coordinate during execution. The paradigm has been successfully used to introduce latency hiding in distributed systems or in a single system where different components operate at different speeds.

The paradigm of programming with multiple threads of execution can provide many benefits for the applications. In our situation, it can provide good runtime concurrency, while parallel programming techniques can be easier implemented. The most interesting and probably most important advantages are performance gains and reduced resource consumption. Operating system kernels supporting multithreaded application perform thread switching to keep the system reactive while waiting on slow I/O services, including networks. In this way, the system continues to perform useful work while the network or other hardware is transmitting or receiving information at a relatively slow rate.

Another benefit of multithreaded programming is in the simplification of the application structure. Threads can be used to simplify the structure of complex, server-type applications. Simple routines can be written for each activity (thread), making complex programs easier to design and code, and more adaptive to a wide variation in user demands. This has further advantages in the maintainability of the application and future extensions. The multithreaded paradigm can also improve server responsiveness. Complex requests or slow clients don't block other requests for service, the overall throughput of the server being increased.

6.4 Resource Estimation

The user of the desktop grid systems needs to know what resources are available in the grid, so that it can formulate more efficiently the requirements for the actual computing job. The system should be able to provide an accurate overview of the available resources. Compared with a classic parallel system, or a dedicated computer cluster, where the available resources are well known, in a desktop grid, their availability is very dynamic: new users can join to the system, other users might use their desktop, making unavailable computing resources, network resources can change, due to different, uncontrollable traffic on the net and so on. We are mainly concerned with two types of resources in the system: network and computing.

6.4.1 Network Performance

Distributed systems are becoming increasingly dependent on network estimation: the ability to determine performance along one or more network paths. These include both estimates of network latency and bandwidth. Producing such quality estimates is challenging because network observations

in distributed systems are noisy, and could be influenced by other communication. The master can measure to the latency and bandwidth between it and the workers by observing the exchanges with them. This can be done in an active way, where the master generates benchmark traffic to the workers and measures the parameters. This can be done with simple ping-like messages to measure the roundtrip time, and implicitly the latency, or with more complex data traffic to estimate the bandwidth.

The other way of measuring is the passive way, where the master measures different times during the real execution of the job. The master measures the elapsed time between submitting a task and receiving an acknowledge response. The worker responds also with the time spent between receiving the request and issuing the response, i.e. the service time. This allows the master to consider networking costs separately from other delays. This method provides insight to the master about possible bottlenecks in the communication, and can decide for example not to send anymore tasks to workers having large communication delays.

6.4.2 Computing Power

We need also to estimate the computing power of each worker that might be used for computation. This is needed only if different types of workers are involved in a job (distributed computation). If all the workers have the same hardware and software architecture, then most likely their performance is identical. A very simple, rough estimate could be based on the clock frequency of the worker's CPU. This, however, does not take into consideration the different CPU architectures, different clock speed, or different cache memories. A more realistic estimate would be to run a standardized CPU benchmark on each worker in order to have an estimate of the workers' potential. In reality, different benchmarks can give different results when comparing two processors, depending on which algorithms each benchmark is using. It also depends on what optimizations are possible for each processor type for the given benchmark. The situation where a certain benchmark is favoring one type or architecture can occur. Thus, choosing the right benchmark is essential to obtain a good estimate of the performance for comparing different workers.

We propose a more realistic approach for estimating the computing power of each worker: at the beginning of each job, the master sends to each worker the same, initial task. Based on the computing time of each worker for this task, the master can have a much better overview of the real computing capabilities of each worker. The size of the initial should be relatively small, such that not too much time is wasted for the benchmark, and also, it should be based on the same algorithm as the rest of the tasks from the job to be run. The advantage of this approach in comparison with a standardized benchmark is that we have a much better, problem dependent benchmark, which is more relevant in comparing the different workers. The master can make a better comparison of the computing performance of each worker for the given job.

There is however a slight increase in the computational time of each job, due to the initial benchmark. Another disadvantage is that the job submitter (user) should provide a smaller initial task. This can be done either explicitly by the user, in the job description, when submitting the job, or the decision of selecting the initial task could be done by the master. The master can simply choose the first task from the job to be submitted as the initial benchmark task to each worker, or it can choose a random task from the job description.

Based on the results of the benchmark, the master can make different decisions to improve the overall throughput of the job to be submitted. For example, it can discard very slow workers from taking part in the job. In case of a parallel job, where tasks submitted to each worker are relatively similar in size, the master could choose similar workers in terms of computing performance. This can minimize the wasted computing power when tasks of similar size are executed and the more powerful workers are waiting results from the slower workers. The user can provide such information about the jobs in the task to the master, in the job description.

6.5 Resource Monitoring

To address efficient usage of networked resources, like computing, storage, and communication resources, it is compulsory to know the availability and usage of the resources in a continuous way, rather than isolated. Monitoring and profiling would provide detailed information with an unobtrusive, continuous, and application independent view for the monitored nodes. In desktop grid environments this is particularly challenging because the desktop PCs are volatile, frequently leaving and joining the system, thus making it difficult to locate all the monitored nodes at any time. The knowledge of dynamic resource properties is vital for improving application performance.

In our approach, the resources under investigation for monitoring are: computing (CPU, memory) and networking. For each such resource we have identified a set of metrics to capture the dynamics of resources: (1) **CPU**: idle time, user time, system time, number of processes, load; (2) **memory**: available, max. used by processes, cache, page faults; and (3) **network**: bandwidth, packets transferred, bytes transferred, packets dropped. Periodically, the metrics for these parameters are sampled and the resulting values are centralized and made available to the user in a friendly way. Depending of the usage of resources, some parameters are updated more often then the others.

The required network communication for transmission of monitoring parameters should interfere as little as possible with the rest of the job communications. This is especially important in the case of fine-grain parallel applications, where communication is crucial. Monitoring data packets should be small, and the frequency with which they are transmitted should depend on the usage of the resources. It is also possible to send such information after jobs are finished, and before others are started, minimizing the interference.

We need to maximize the time spend for computation and minimize the time spent for communication, by using some of the techniques that are beneath:

- overlap computation with the communication on the worker by using separate threads for them (both send and receive);
- minimize time spent doing communication by reducing the size of the transferred messages: efficient packing of information, use of compression;
- pipeline work-units on the worker - minimize the time a worker has to wait for getting a new work-unit;
- make workers inter-communicate - if this reduces redundant computations.

6.6 Scheduling

In this section we present the scheduling problem adopted in this work and we present also our proposed policy to solve it. Efficient scheduling of a master-worker application in a cluster of distributive owned resources should provide answers to the following questions:

- *How many workers should be allocated to the application?* A simple approach would consist of allocating as many workers as tasks are generated by the application at each iteration. However, this policy will result, in general, in poor resource utilization because some workers may be idle if they are assigned a short task while other workers may be busy if they are assigned long tasks;
- *How should tasks be assigned to the workers?* When the execution time incurred by the tasks of a single iteration is not the same, the total time for completing a batch of tasks depends on the order in which tasks are assigned to workers.

The problem of scheduling master-worker applications on cluster environments has been investigated recently in the framework of middleware environments that allow the development of adaptive parallel applications running on distributed clusters. They include NetSolve, Nimrod and AppLeS. NetSolve and Nimrod provide APIs for creating task farms that can only be decomposed by a single bag of tasks. Therefore, no historical data can be used to allocate workers. The AppLeS (Application-Level Scheduling) system focuses on the development of scheduling agents for parallel applications but in a case-by-case basis, taking into account the requirements of the application and the predicted load and availability of the system resources at scheduling time.

There are other works in the literature that have studied the use of parallel application characteristics by processor schedulers of multi-programmed multiprocessor systems, typically with the goal of minimizing average response time. The results from these studies are not directly applicable in our case because they were focused on the allocation of jobs in shared memory multiprocessors without considering the problem of task scheduling within a fixed number of processors. However, their experimental results also confirm that iterative parallel applications usually exhibit regular behaviors that can be used by an adaptive scheduler.

7 The QADPZ System

This chapter describes the QADPZ ['kwod 'pi: 'si:] system, a desktop grid environment for running compute-intensive tasks in a distributed way, using the computational resources from already existing desktop-class machines from an Intranet (corporate-wide) or from the Internet (worldwide). Although the idea of using idle computational resources from existing computers is not new, few systems exist today which could easily provide the necessary support for our Scientific Computing and Visualization needs. Besides the distributed capabilities, the system provides for parallel computing as well. The reasons for building a new such system are described. Furthermore, the chapter describes the design and implementation of the QADPZ system, with details of the requirements, architecture, communication, security and user interface. We explain how this system supports the conceptual model described, and present some of the more advanced features of the system. The system described is not limited to Scientific Computing and Visualization, and it can be used for other types of computational intensive applications.

7.1 Description

QADPZ (Quite Advanced Distributed Parallel Zystem) is a system for heterogeneous desktop grid computing. The system allows a centralized management and use of the computational power of idle computers from a network of desktop computers. Users of the system can submit compute-intensive applications to the system, which are then automatically scheduled for execution. The scheduling is made based on the hardware and software requirements of the application. Users can later monitor and control the execution of the applications. Each application consists of one or more tasks, the smallest execution unit of the system. Applications can be independent, when the composing tasks do not require any interaction. They can also be parallel, when the tasks communicate between each other during the computation. Parallel communication is done using a subset of the widely used MPI standard. Thus, the system provides for both task- and data-parallelism. QADPZ can operate both in conditions of an open Internet environment and of a closed local network which supports the family of TCP/IP protocols.

7.2 Justification for a New Desktop Grid System

Although the idea of using the idle computational resources from existing desktop computers is not new, the use of such distributed systems, especially in a research environment, has been limited, however, due to a lack of supporting applications, and because of security, management, and standardization challenges. Existing systems that were available in July, 2001, the date of first QADPZ release, were specialized towards a very limited number of

computational intensive problems, or too general to provide the necessary support for specific type of applications. For the purpose of this thesis, a flexible tool was needed to conduct experiments that concern Scientific Computing and Visualization. Also, at the time QADPZ has been developed, most of the existing systems have had very restrictive licenses, which had not permitted adaptation of the code to different requirements.

More, the majority of the existing desktop grid systems, as it has been shown in the previous chapter, provide no support for parallel application, when communication between programs running on different computers is necessary during the computation. This makes difficult to use such systems for our purpose, where more than one desktop computer are needed to solve a certain problem. Tasks with independent parallelism are suited for this type of computing. For example, in SETI@home, work unit computations are independent, so participant computers never have to wait for or communicate with one another. If a computer fails while processing a work unit, the work unit is eventually sent to another computer. Public-resource computing, with its frequent computer outages and network disconnections, seems ill-suited to parallel applications that require frequent synchronization and communication between nodes. However, scheduling mechanisms that find and exploit groups of LAN-connected machines may eliminate these difficulties.

To summarize, the need to develop the QADPZ desktop grid system has arisen from the following main reasons:

- many existing systems tended to be highly specialized towards a very limited number of computationally challenging problems, and hence did not allow the degree of flexibility that was desired;
- source code was generally not available, hence making any novel non-standard applications, extensions and analyzes difficult;
- very few of the existing systems allowed specific considerations to be made wrt the challenges of scientific computing and visualization;
- most of the existing systems usually have a complicated deployment procedure, requiring high-level, privileged access to the desktop computers; this makes very hard to use such systems on a larger scale, and also makes further maintenance of the computers more complicated;
- the front-ends of most existing systems did not match up to current expectations of user-friendliness, which limits very much the possibility of using these systems on a day by day basis;
- many of today's networks of desktop computers are heterogeneous, thus requiring a distributed computing system with support for different architectures and different kind of operating systems.

All these reasons have lead to the development of a new tool for desktop grid computing: the QADPZ system.

7.3 Design and Implementation

7.3.1 Requirements

Based on the reasons mentioned earlier, we have set up a set of requirements that a successful desktop grid computing system should satisfy in order to support applications in Scientific Computing and Visualization. The overall goal of the system was to be friendly, flexible and tailorable to many different requirements. The main prerequisite is therefore an open architecture that can evolve in pace with the need and challenges of the real world.

We specify two sets of requirements for the system, as it can be seen from Figure 7.1- one for the whole system, mostly from a functionally point of view, and another set for the interface of the system. The interface of the system covers both user interfaces and programming interfaces. Additionally, we describe a set of non-functional requirements, concerning more the development of the system itself.

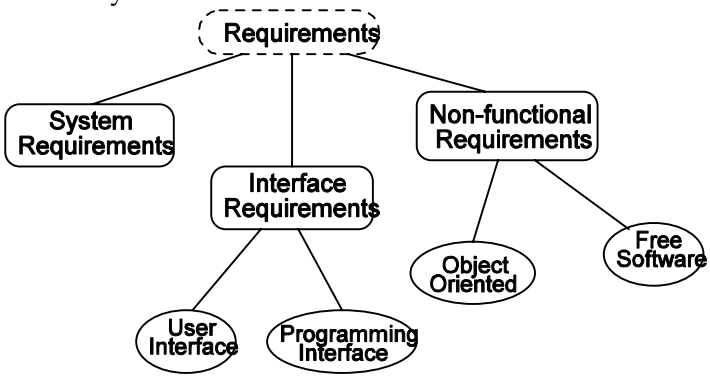


Figure 7.1 QADPZ requirements

7.3.1.1 System Requirements

System requirements are related to the core of the system and are concerning mainly with the sharing and management of resources and application jobs in a heterogeneous environment, but also involve performance and usability of the system as required by our conceptual model (see Figure 7.2).

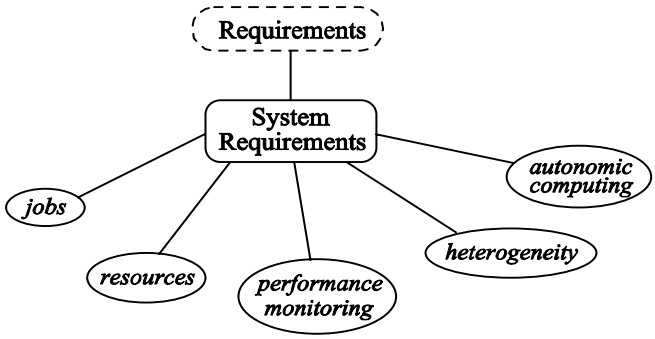


Figure 7.2 QADPZ system requirements

These requirements are listed further on:

- *resource sharing*: the most important resources that need to be shared are the idle computational cycles of the desktop machines which contribute to the system; it should also be possible to share other kind of resources from the computers, like for example storage space;
- *resource management*: the system should be able to manage efficiently the available shared resources; furthermore, owners of desktop computers which are sharing resources should keep control of them, by allowing owners to define use policies and retract the resources if they want that;
- *job management*: the user should be able to submit computational jobs to the system, which will be executed using the shared computational cycles; it should be possible to monitor and control job executions;
- *heterogeneity*: the system should be possible to deploy on a network of heterogeneous desktop computers, with different architectures (Intel, RISC, etc.) and different operating systems (UNIX type, Windows, Mac OS); it is the responsibility of the user submitting the jobs to provide the appropriate binary files for execution on the different platforms;
- *simple installation and maintenance*: the system should be easy to install on a large number of computers in a network, and further maintenance of the installed programs should be minimal;
- *parallel programming support*: the system should support different kind of parallel programming paradigms, for example both task- and data-parallelism; it should be preferable to use well known standards for this; MPI is an example of parallel programming standard;
- *network support*: the system should work in a LAN environment, but it should also allow the possibility to be used over the Internet; the higher level communication protocol used between different components of the system should be based on both TCP/IP and UDP/IP families of protocols. The reasoning for this dual support will be given later;
- *autonomous features*: the system should be able to deal with its own complexity, by supporting different autonomic features: self-healing, self-management, self-knowledge, self-configuration, self-optimization;
- *provide performance measurements*: the system should be able to provide to the user some information about the performance of the system, which could be used for better usage of the available resources;
- *multi-project use*: many projects have downtime and shortage of tasks. Participation in multiple projects helps to cope with projects' downtime;
- *on-line/off-line support*: the system should provide support for both batch and interactive type of applications; in a batch setting, the user submits jobs which will be executed at a later time, when resources become available; in contrast, interactive jobs provide real-time feedback of the execution; the user can inspect the partial result and interact with the execution of the application.

7.3.1.2 Interface Requirements

The interface requirements of the QADPZ system can be split up into two parts. One is for the *user interface*, concerned with the graphical interface by which the human user accesses the system. With this interface, the user can either monitor, or control the behavior of the system. The other interface is the *programming interface (API),* which allows different user applications to interact with the QADPZ system (see Figure 7.3).

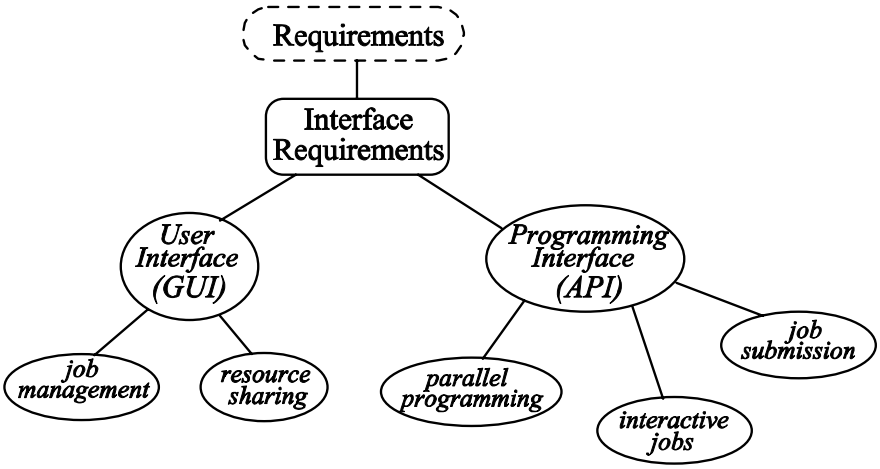


Figure 7.3 QADPZ interface requirements

All of these requirements are enlisted beneath:

- *personalization*: the system should provide different levels of access to the users, according to their skills and personal preferences; users without strong programming skills should be able to use the system without difficulty, even if only for easy tasks; more advanced users should be provided with a programming API to interface more efficiently with the system and use the full capabilities of it;
- *job management interface*: there should be a simple, preferably platform independent graphical user interface, to allow submission, monitoring and control of the different computational jobs in the system;
- *resource sharing interface*: the owner of a desktop computer should be provided with a simple and intuitive graphical user interface that allows her to control the sharing of his computational and storage resources;

7.3.1.3 Non-functional Requirements

Non-functional requirements of the system are constraints on various attributes of these functions of the system (see Figure 7.4). The software tools have been developed as free/open source software, which is a natural choice for modern research - it encourages integration, cooperation and boosting of new ideas, in a very effective way. We have decided to build QADPZ using C/C++, both for reasons of high performance and object-orientedness of the language.

High performance was required by the nature of Scientific Computing and Visualization, especially when handling large data sets. The object-oriented features provided by the C++ language were fully employed, together with suitable advanced object oriented design patterns. This promoted software component reuse and significantly contributed to the *maintainability*, *flexibility* and *extensibility* of the system, all very important requirements of such a distributed and heterogeneous system.

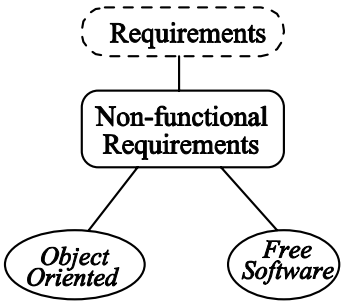


Figure 7.4 QADPZ non-functional requirements

Modularity of the system was another significant requirement. This provided for the source code for an object to be written and maintained independently of the source code for other objects. It also allowed an object to be easily passed around in the system. Important was also *simplicity*: software objects were design to model real world objects, so the complexity of the system has been reduced and the structure has become much more clear.

7.3.2 Architecture

The QADPZ system has a centralized architecture, based on the client-server model, which is the most common paradigm used in distributed computing. The paradigm describes an asymmetric relationship between two processes, of which one is the client, and the other is the server. Almost all applications based on this paradigm involve multiple clients, however they can involve one or multiple servers. In our case, the server manages the computational resources available from the desktop computers. Offering a service, which can be used by other processes, does this. The client is a process that needs the service in order to accomplish a certain work. It sends a request to the server, in which it asks for the execution of a concrete task that is covered by the service. Usually, the server carries out the task and sends back the result to the client.

In our situation, the server has two parts: a single master that accepts new requests from the clients, and multiple slaves, which handle those requests. The system consists of three types of entities: master, client, and slave (Figure 7.5). Each computer contributing with computing power to the system is called a *slave*, and is running a small background process in the form of a UNIX daemon or a Windows system service. The process can be run with the privileges of an ordinary user, it doesn't need to be run with administrative

rights. This process is responsible for reporting the computer's resources and status to a central server, the *master*. It also accepts computational requests from the master, downloads the corresponding binaries and data files for the tasks, executes the task, and then uploads the result files when finished. The slave also downloads the task to be executed together with the input data, and starts the computation. The presence of a user logging into a slave computer is automatically detected and the task is killed or moved to another slave to minimize the disturbance of the regular computer users.

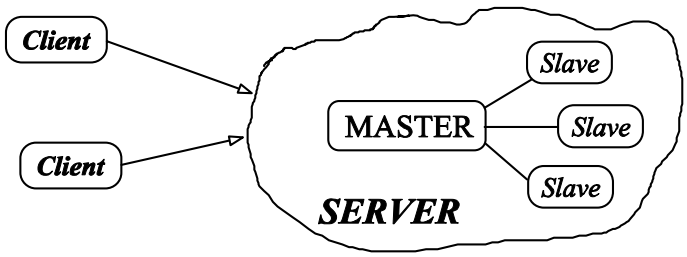


Figure 7.5 QADPZ coarse architecture

The main role of the master is to maintain the current availability status of the slaves, and to start and control the tasks. The master knows about all the resources and jobs in the system. The *master* is responsible for managing the available resources, keeping track of the available slaves, their capabilities and configuration. It also schedules the computational tasks submitted by any authorized user of the system, according to the required resources. Tasks can be started, stopped, or rescheduled by the master. There are two ways of doing this: *a batch mode*, and an *interactive mode*. In the batch mode, which is using our universal client, a project file, specifying the required resources and how to start the tasks, describes tasks. This information is then sent to the master, which is responsible for scheduling it. In the interactive mode, the client has much more freedom over the creation and controlling of new tasks. It can have also direct feedback from the running tasks, either through the master node, or communicating directly with the slaves. This is more suited for applications where interactivity with the running computation is required.

The *client* is the interface by which a user interacts with the system. Its main purpose is to allow the human user to create new computational jobs in the system. It allows also monitoring them, and controlling their execution. A client does not communicate with the slaves directly, instead it sends all its requests to the master. A more detailed architecture of the system is described in Figure 7.6. The control and data flow in the system are separated. Data files, represented by binary, input, and output files, needed to run the applications are not sent to the master. They are stored on one or more data servers. An even more comprehensive view of the architecture is presented in Figure. 7.7.

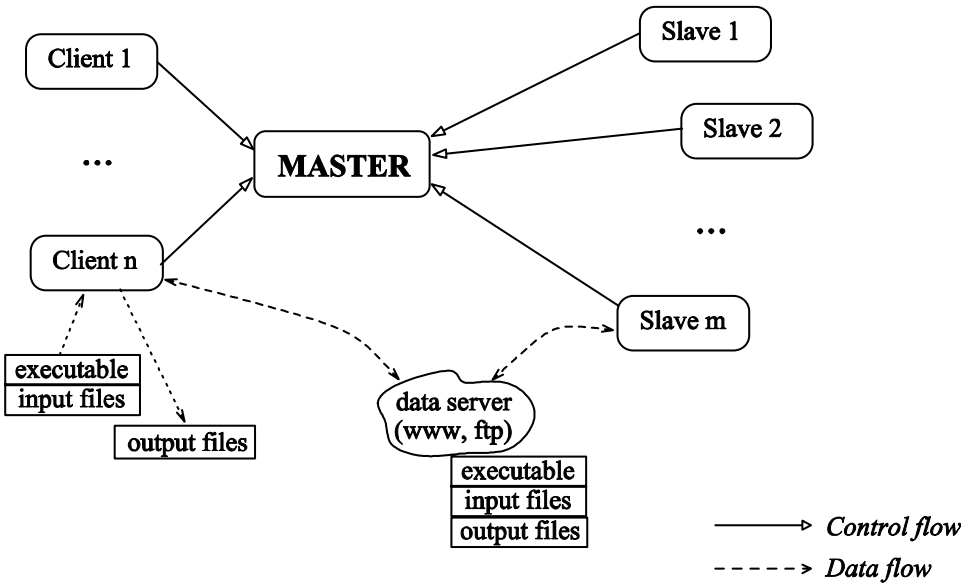


Figure 7.6 QADPZ detailed architecture

Multiple tasks can be grouped into jobs, for an easier management. Different types of jobs can be submitted to the system. A job can consist of independent tasks, which do not require any kind of communication between each other. This is usually referred to as task parallelism. Jobs can also consist of parallel tasks, where different tasks running on different computers can communicate with each other. Inter-slave communication is accomplished using a subset of the MPI standard. The current implementation of the system is made considering only one central master node. This can be an inconvenience in certain situations, where computers located in different networks are used together. However, our high level communication protocol between the entities, especially between the client and master, allows a master to act as a client to another master, thus making possible to create some sort of virtual master, consisting of independent master nodes, which communicate with each other.

7.3.2.1 Job-view of the system

The users of the QADPZ system can submit, monitor, and control computing applications to be executed on the computers sharing computational resources (Figure 7.9, Table 7.1). The smallest independent execution unit is called a task. Tasks are binary programs, which can run on any of the platforms sharing computing resources. A task comes in the form of an executable program, compiled for a specific architecture and OS. When better performance is required, a task can be also in the form of a shared (dynamic) library, which can be more efficiently loaded by the slave program running on a computer sharing resources. As an alternative to native binary programs for a specific platform, a task can also be an interpreted or precompiled program. For example, it can be a compiled Java application, which further needs a Java Virtual Machine on the host computer; it can also be an interpreted program (e.g. Perl, Python), but then an interpreter for that specific language is required on the host computer.

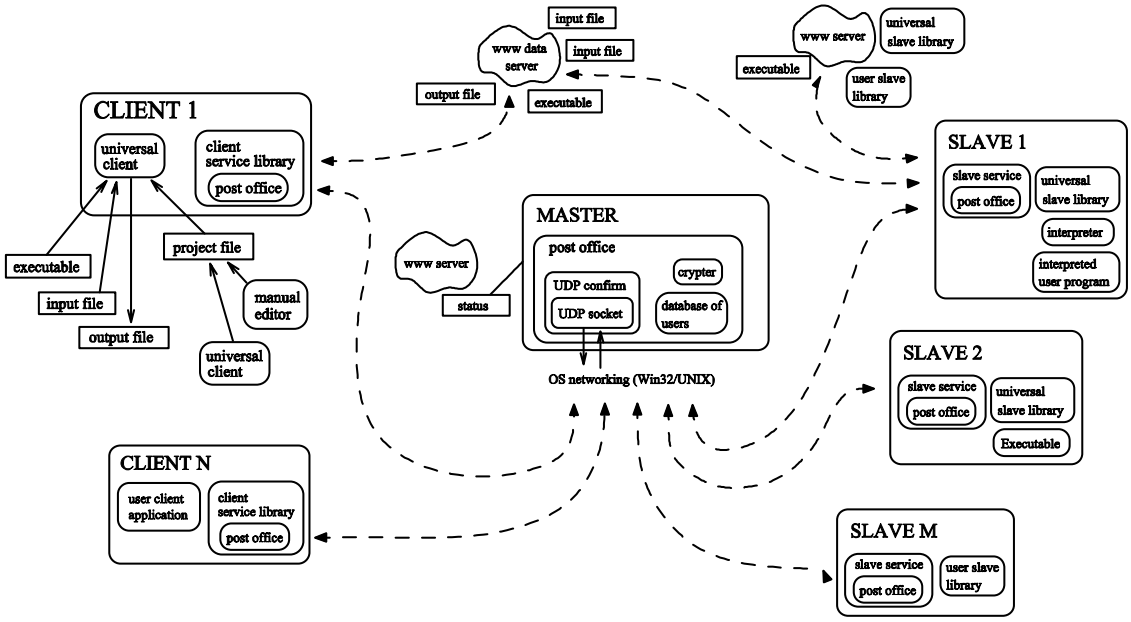


Figure 7.7 QADPZ close-up architecture

A simplified UML Diagram of QADPZ's architecture is depicted in Figure 7.8.

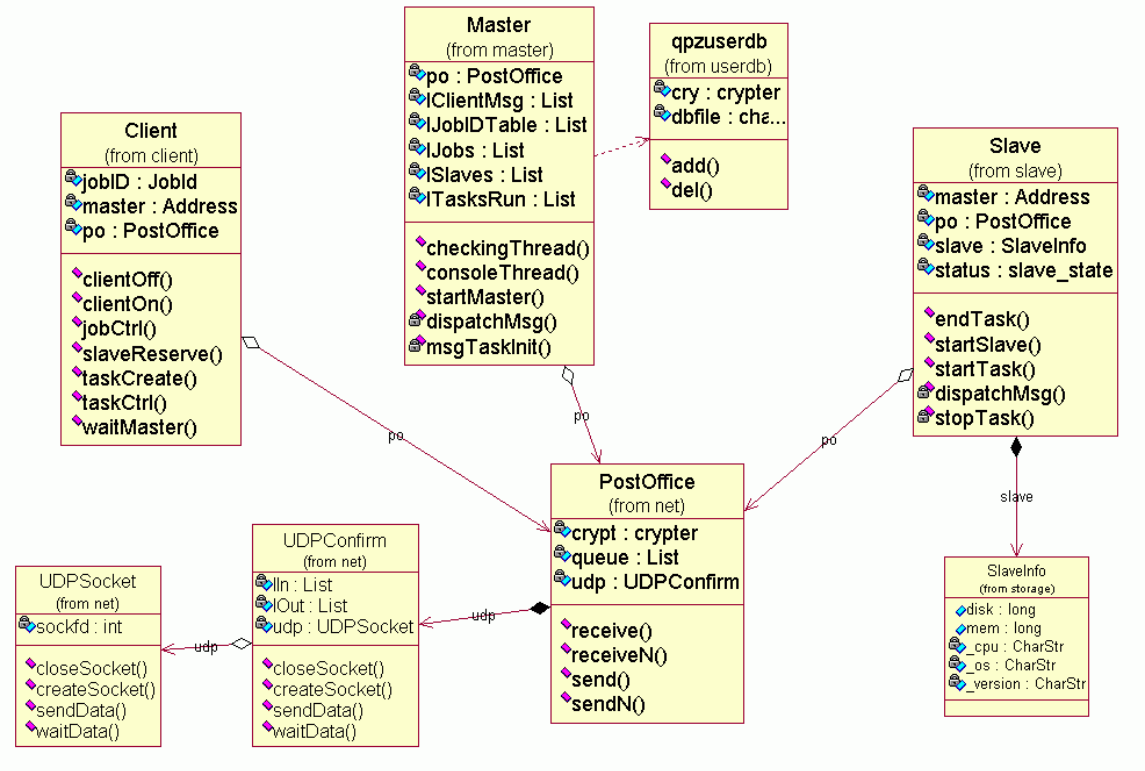


Figure 7.8 Simplified UML Diagram of QADPZ's architecture

Multiple tasks, which are related to each other, can be grouped into a so-called *job*. This is actually what a user submits to the system. A job can be composed of one or more tasks, and allows an easier structuring and management of the computational applications, both from the user's and the system's point of view. Each job is assigned uniquely to one user; however, a user can have multiple

jobs submitted at the same time to QADPZ. The tasks part of a job can be either independent from each other, or they can depend on each other at execution time. Tasks can further be divided into *subtasks*, consisting of finer work units executed within a task. A task can contain different types of subtasks. Subtasks are used for interactive applications, which require permanent connection between a client and the slaves. They are usually generated at the client and send for execution to an already running task, which can solve it. The main reason for having subtasks is to improve the efficiency of smaller executional units without having the overhead of starting a new task each time.

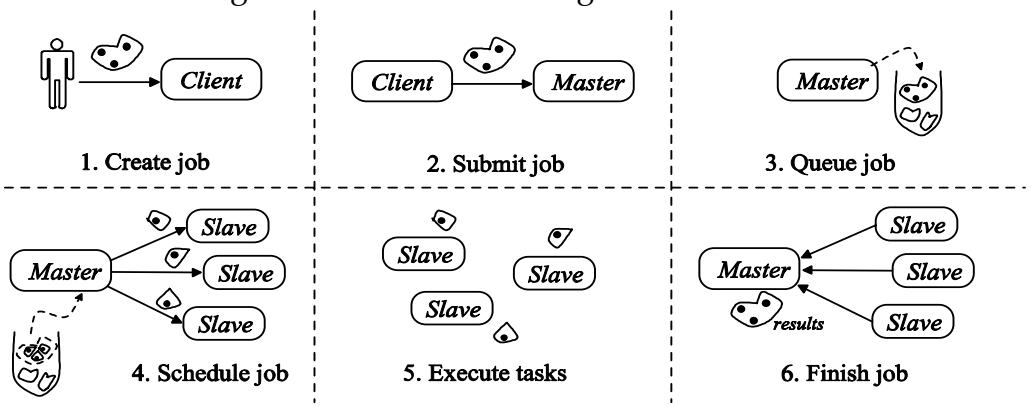


Figure 7.9 QADPZ job life

Operation	Where	Description
Create job	Client	the user creates a job
Submit job	Client	the job is submitted to the master
Queue job	Master	the job is queued for execution
Schedule job	Master	the job is scheduled for execution
Execute tasks	Slave	component tasks of the job are executed
Finish job	Master	informs client about job completion

Table 7.1. - The life of a job

7.3.2.2 Slave

The *slave* component of the system has two roles. On one hand it has to report to the configured master node the resources shared. These are mainly computational resources (CPU cycles), but can also be for example storage space. The slave sends information about the system periodically to the master. The information (see Figure 7.10) describes the hardware architecture of the slave (cpu type, cpu speed, physical memory, etc.), the software environment available on that architecture (operating system, different application or libraries available), and the resources available on that slave.

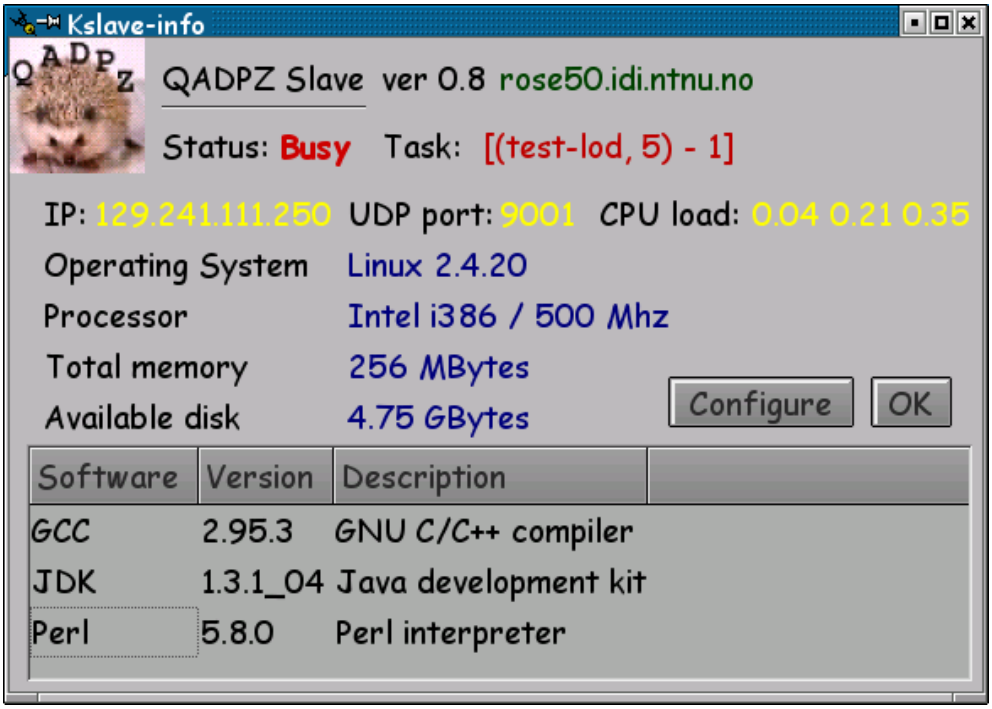


Figure 7.10 QADPZ slave info user interface

On the other hand, the slave can accept computational jobs from the master. This is done only when the slave is free, and any interactive, local user does not use resources. The slave decides for itself if it should accept or not a computational job to be run by setting some configuration parameters. The user can configure different times of day when the slave can accept computational jobs. It can also disable the slave at any time. The slave component runs as a small background process on the user's desktop. It starts automatically when the system starts. The program does not need any special privileges to run, which makes it very easy to install and control by any ordinary user.

7.3.2.3 Master

The *master* is responsible for managing the available resources, keeping track of the available slaves, their capabilities and configuration. It has always an up-to-date overview of the resources. Basically it knows which slaves can accept jobs for execution and how to contact them. The master is also responsible for scheduling the computational tasks submitted by any authorized user of the system. Jobs are sent to the appropriate slave based on the hardware and software requirements from the jobs' description. Tasks can be started, stopped, or rescheduled by the master. Tasks are created by users, who can submit them to the master by means of the *client* as an interface to the QADPZ system. For this, the master needs also to keep a database of authorized users.

7.3.2.4 Client

The client represents the interface for submitting jobs in the system. There are two execution modes for the client: a batch mode and an interactive mode. In the batch mode, which can be done using the universal client, a project file, specifying the required resources and how to start the tasks, describes tasks. This information is then sent to the master, which is responsible for scheduling the tasks. The client can detach from the master and connect later for the results. Each project is described by using the XML language, as it will be seen in some examples later in this chapter. In the interactive mode, the client stays connected to the master for the entire time of the execution of the job. The client can also get direct connection to each of the slaves involved in the computation. The client has much more freedom over the creation and controlling of new tasks: it can dynamically create new tasks, send messages to already executing tasks, and can receive feedback from the running tasks, either through the master node, or communicating directly with the slaves running the respective tasks. This execution mode is more suited for applications where interactivity with the running computation is required.

7.3.2.5 User Interface

The purpose of the user interface in QADPZ is to give a user-friendly environment in which the user can interact with the system. This mainly involves submission, monitoring, and management of submitted computational applications. It also involves resource monitoring and controlling. The first interface is the job monitoring interface, described in Figure 7.11.

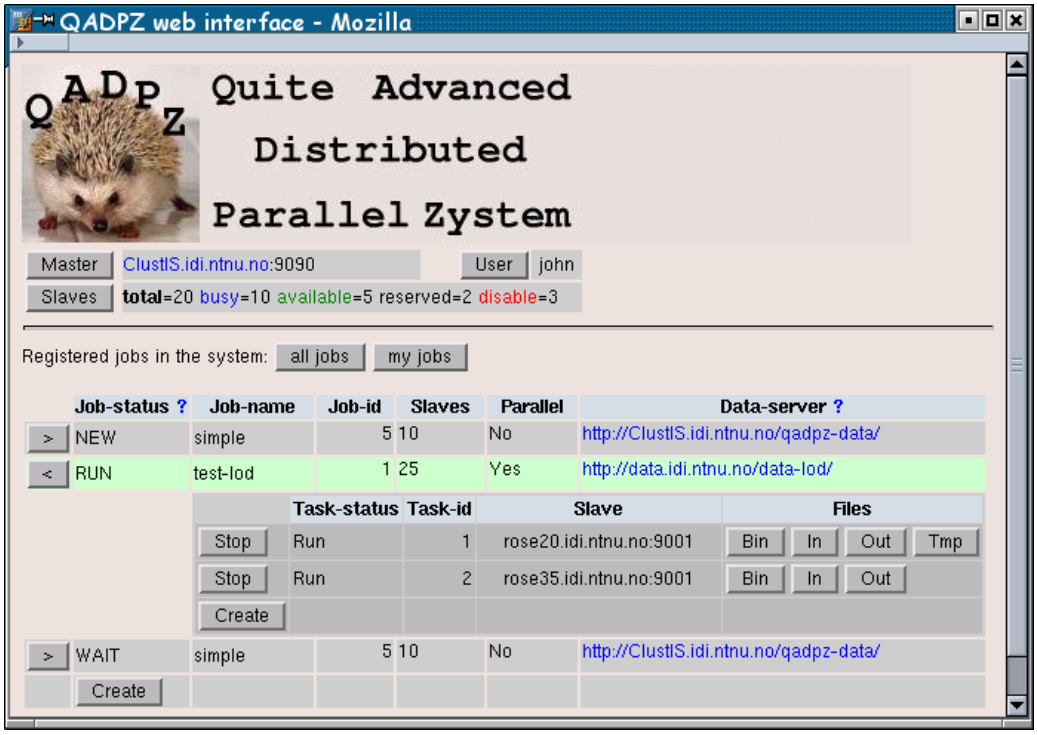


Figure 7.11 QADPZ job monitoring web-interface

It is a web-based interface, which provides detailed information about all existing jobs in the system. The user can browse through each of the jobs, and see their status, and the tasks, which are part of the jobs. It can also easily create new jobs and tasks. Each job can be stopped or deleted by using this interface. The second interface is also web based and provides information related to the resources in the system. Basically it gives a list of the slaves registered in the system and their current status (see
Figure 7.12).

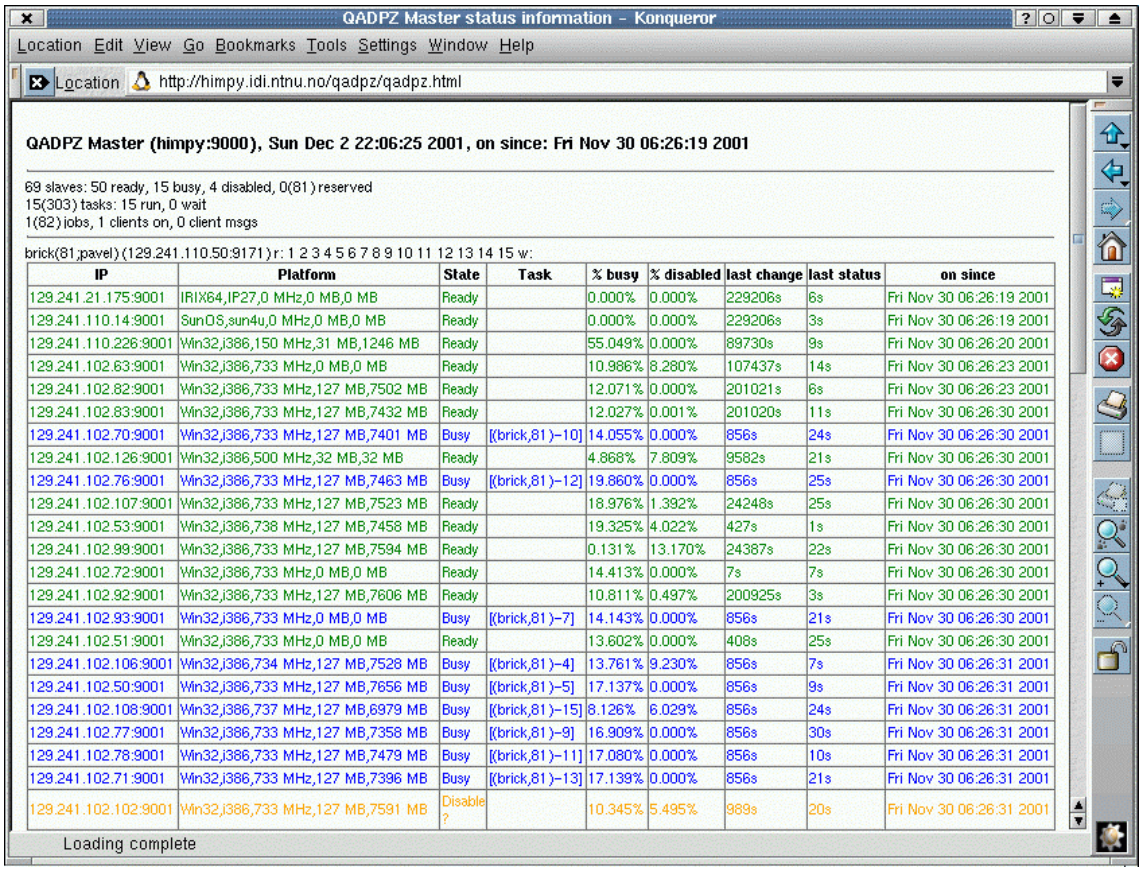


Figure 7.12 QADPZ resource monitoring web-interface

The owner of a desktop computer running a slave is given an interactive application, which permits easy configuration of the slave (Figure 7.13). It allows the user to specify which time of day the slave should accept jobs for execution, and also other configuration parameters. The user has complete control over the slave running on his computer.

7.3.3 Communication

The various components of the QADPZ system (master, slaves, clients) must communicate with one-another using IP connections over a LAN or WAN, depending on the deployment of the components. Single-stream TCP performance on the WAN is often disappointing. Even with aggressive tuning

of the TCP window size, buffer sizes, and chunking of transfers, typical performance is still a fraction of the available bandwidth on the WAN on OC-12 or faster links. While there are a number of factors involved, the behavior of the TCP congestion avoidance algorithm has been implicated as a leading cause of this performance deficit.

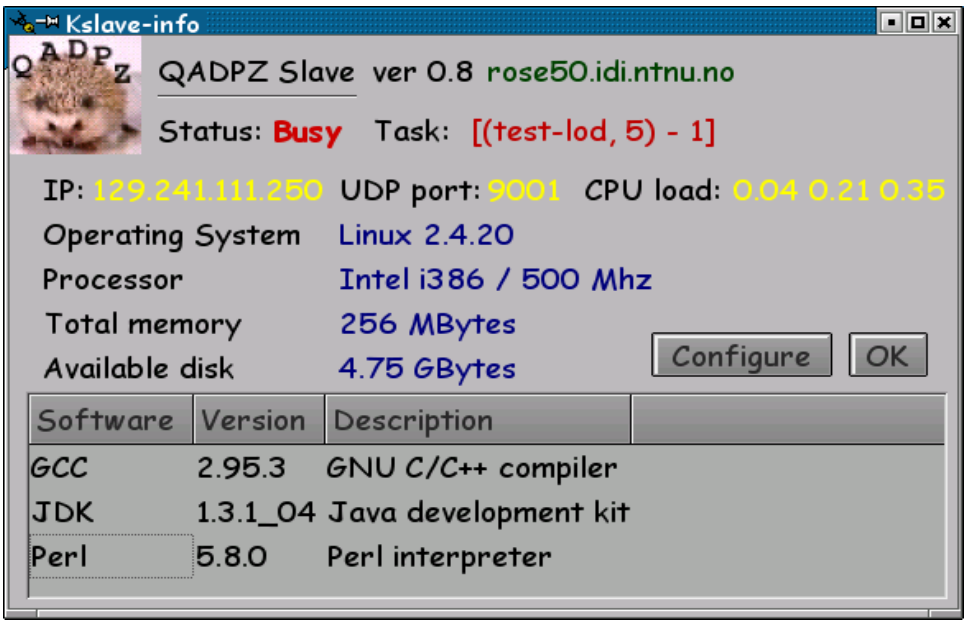


Figure 7.13 QADPZ slave configuration interface

TCP congestion avoidance algorithms assume that any packet loss is due to congestion, which we define to be over subscription of bandwidth on any switch or link along the path the packet stream takes on the WAN. However, it is increasingly the case that packet loss is caused by events that are unrelated to congestion. The sensitivity of TCP to loss is further exacerbated as the bandwidth of the network is increased, so solutions to remedy poor TCP performance will be increasingly important to distributed computing applications on the WAN. Simulations of the TCP protocol performed by Jacobson and Floyd show a high sensitivity to loss, but also demonstrate the fact that from a control theory standpoint, that the TCP congestion avoidance algorithm results in periodic fluctuations that resonate with the deterministic control mechanisms of switching fabric in a highly non-linear and unstable fashion. The default "taildropping" behavior of packet switch input-queues leads to significant degradation in performance.

The conclusion is that TCP congestion control, while adequate for the 10megabit networks it was originally designed for, is failing completely on today's multiple gigabit networks and multiple efforts are underway to work around these problems. Loss-tolerant UDP-based protocols will play an increasingly important role in high throughput network applications of the near future. With custom tools, it is possible to find out that UDP packet loss rates are consistently low until you reach a critical limit, which is the available

bandwidth of the slowest/most congested link in the network path. At the critical point, when the slowest link in the path across the WAN becomes congested, the loss rate climbs almost linearly in proportion to the increase in output rate. It is interesting to note that frame loss rates, even at low data rates, exhibit some background loss. This is counterintuitive, as switches should be less congested when exposed to the slower stream, based on the models of network loss assumed by TCP congestion avoidance algorithm.

For data transfer and replication, file integrity is paramount. Response time and performance is of comparable importance to file integrity for visualizations. Visualization tools almost invariably use reliable transport protocols to connect distributed components, since there is a general concern that lifting the guarantee of data integrity would compromise the effectiveness of the data analysis. However, visualization researchers find acceptable other forms of lossy data compression like JPEG, wavelet compression and even data resampling. Acceptance may be due to the fact that degradation in visual quality is well behaved in these cases. Therefore, an unreliable transport mechanism that deals with packet loss gracefully and doesn't exhibit extreme visual artifacts will compete well with other well-accepted data reduction techniques. Furthermore, when tuned to fit within the available bandwidth of a dedicated network connection, the loss rates for unreliable transport are extremely small - a few tenths of a percent of all packets sent if the packets are paced to stay within the limits of the slowest link in the network path.

In QADPZ, messages exchanged between entities in the system are in XML format, in accordance with a strictly defined communication protocol between client and master, and between master and slave (the QADPZ protocol). For our current implementation, the low level communication is based on both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). We have chosen to implement both protocols to be able to make performance measurements and comparisons. Nevertheless, for the above reasons, UDP is our first option for the low-level communication protocol.

As shown previously, the UDP is an unreliable communication protocol, in which packets are not guaranteed to arrive and if they do, they may arrive out of order. We have outcome this by adding a new layer that ensures reliable communication with UDP. Our higher-level communication abstraction implements a reliable, confirmation based message exchange protocol. Messages are represented in an XML format for easier extensibility and interconnection with other potential systems.

The advantage of UDP over TCP/IP is that UDP is fast, reducing the connection setup and tear-down overhead, and is connectionless, making the scalability of the system much easier. The higher-level protocol is message based, and the messages exchanged between the components of the system are of a small size. Also, messages are exchanged only for control purposes.

Because of the unreliable nature of the UDP protocol, an additional, more reliable level of communication is needed (see Figure 7.14). This is based

on message confirmation. Each message contains a sequence number, and each time it is sent, it is followed by an acknowledgment from the receiver. Each sent or received message is accounted, together with the corresponding acknowledge, and in case of not receiving an acknowledgment, the message is resent a few more times. An acknowledge and a normal message can be combined into one message to reduce the network traffic.

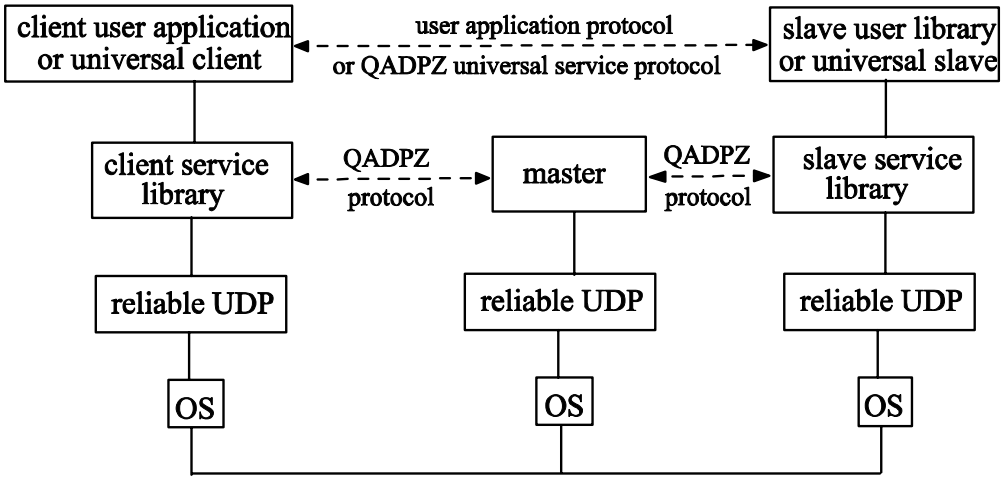


Figure 7.14 QADPZ communication layers

The abstraction used in our reliable communication layer (Figure 7.15). is similar in functionality with the real life postal service. It delivers and receives high level messages. We use two types of high level messages: an XML format for control messages between entities in the system, and a plain binary format, used for any kind of data transfer, for example when slaves inter-communicate.

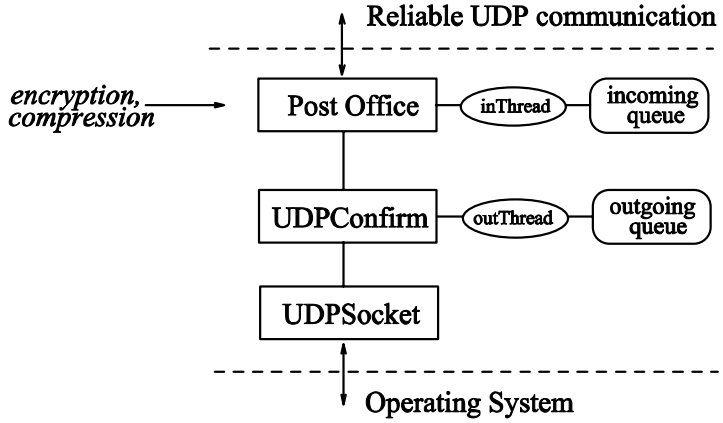


Figure 7.15 Reliable UDP communication

The PostOffice module supports both blocking and non-blocking delivery of messages. Messages have a source and a destination address. Received messages can be kept by the PostOffice as long as needed, the upper layers in

the system having the possibility to retrieve only certain messages, based on the sender's address. This layer provides also support for encryption of messages, providing a certain amount of security to the system. Another feature provided is the possibility of using compression, this way reducing the size of the messages by using some more CPU power.

The PostOffice module is using the capabilities of the UDPConfirm layer for sending and receiving messages. This layer is responsible for resending any previously sent and unconfirmed messages. A separate thread is checking periodically these messages. It provides both blocking and non-blocking message sending. The lowest level module is called UDPSocket, and is responsible for hiding operating system specific function calls, making a more general interface for UDP socket communication.

7.3.4 Parallel Computing

The Message Passing Interface implemented is based on the previously described reliable UDP communication mechanism, and is shown in Figure 7.16. It is using the same message based protocol as the one used for the communication between the entities in the system. The PostOffice abstraction provides a way to send/receive messages in a blocking or non-blocking mode. This provides an easy way to implement the different types of *MPI_send()* and *MPI_recv()* functions from the MPI standard, and the *MPI_wait()*.

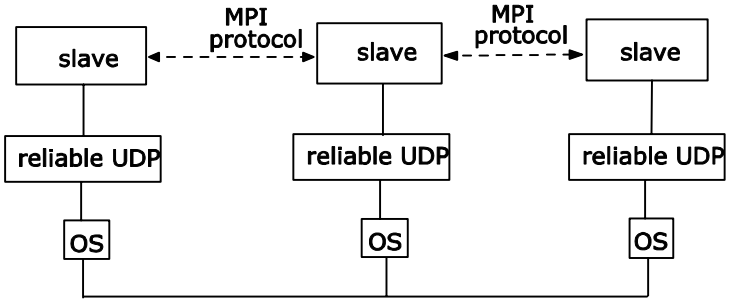


Figure 7.16 MPI communication

The initialization of the MPI communication between slaves is done with the help of the master node: when executing the *MPI_Init()* routine, each slave sends a message to the master, specifying the address and port of its UDP socket used for sending and receiving messages. The master waits for this message from all of the slaves, then gives each slave a rank and distributes a list of all MPI nodes to each of the slaves. Our current implementation does not support collective communication, only the MPI_COMM_WORLD communicator. However, a complete library of the collective communication routines can be written entirely using the point-to-point communication functions and a few auxiliary functions. The implementation provided is limited to a small subset of the MPI. It contains only the most used functions,

and is intended only for testing purposes and evaluation of the parallel communication. More complete implementation based on existing libraries is possible, but is outside the scope of this thesis. The MPI functions implemented provide sufficient features for our parallel experiments.

7.3.5 Interplatform operability

The pool of computers in a network that can run different operating systems and have different hardware architectures achieves inter-platform operability. QADPZ handles task submissions with platform specifications, and the appropriate library or executable is automatically used. It was compiled and tested on many different platforms: Linux, Windows, FreeBSD, MacOS X, IRIX, and SunOS. Our current installation runs on a pool of computers consisting of 80 Pentium III computers from one of our labs, and the nodes of VI a Beowulf cluster of 40 Athlon computers. Currently, we use a daemon process on Unix environments, or a system service on Windows. At the time of writing, we have successfully tested the system on the following hardware platforms: Linux/iX86,sparc,sparc64, FreeBSD/iX86, SunOS/sun4m,sun4u, IRIX64/IP27, and Win32/x86,x86_64. Most of the code is ANSI C++ and POSIX.1 compliant and therefore porting to a new platform does not require too much efforts. We use the POSIX threads API.

7.3.6 Security

Because of the unreliability of the UDP protocol, it is not guaranteed that the executional tasks arriving to the slave computers are undoubtedly sent by master. This is a serious security threat since it allows for a malicious hacker to submit any piece of code to the slave nodes (IPspoofing). For that reason, and on the cost of a decreased performance, all communication from clients to master and from master to slaves is crypted or signed. Particularly, the data flow from client to master has to be authorized by a QADPZ user name and password and crypted by a master public key. A master private key signs the data flow from master to slaves and the authenticity is verified by master public key on slave nodes.

It is important to note that the data flow from slaves to master and from master to clients is not crypted nor signed, which means that a malicious hacker can monitor (packet sniffing) or alter (IPspoofing) the data or control information arriving back to master or client nodes and thus: put the slave nodes out of operation, put the master node out of operation, modify the result data submitted by slaves, or do any other kind of harm to the computational process. In other words, the current QADPZ security scheme is designed to protect the security of the computers in the network, i.e. a malicious hacker cannot submit an alien piece of code to be executed instead of a user computational task. However, this scheme doesn't protect the QADPZ user data. We are considering allowing optional data integrity in the future versions

of the QADPZ system.
Security of the system is handled in two ways. On one hand, only users registered to the QADPZ system are allowed to submit applications for execution. This is done by using a user/password scheme, and allows a simple access control to the computational resources. The QADPZ system manages its own user database, completely independent of any of the underlying operating systems, thus simplifying users' access to the system. The QADPZ system administrator creates new users by maintaining this user database. Tools are provided to minimize the effort in doing this. The other type of security used by the system involves the encryption of messages exchanged between components of the QADPZ system. This is done using public key encryption, and provides an additional level of protection against malicious attacks.

7.3.7 Autonomic Computing Features

IBM's manifesto on autonomic computing (Kephart and Chess, 2003) points out that the difficulty of managing today's computing systems is not only because of the administration of individual software environments, but also because of the need to integrate multiple heterogeneous environments, and to extend beyond company boundaries into the Internet. All these factors contribute to increased levels of complexity in computing systems. Installing, configuring, and maintaining such large systems is becoming an increased challenge even for experts. A possible solution to this problem is to embed the complexity in the system infrastructure itself (both hardware and software), then automating its management. This is in a way similar to the human system, with its autonomic nervous system, which provides automatic, involuntary regulation of the major physiological functions. The essence of autonomic computing systems is *self-management*, the intent of which is to free system administrators from the details of system operation and maintenance. In a similar way to the biological systems, autonomic systems will maintain and adjust their operation in the face of changing components, demands, workloads, and external conditions, and also will be able to handle hardware or software failures. Such systems will be able to monitor their use and interact with other systems. The following is a list of defining characteristics for an autonomic computing system, according to the IBM manifesto (IBM, 2001):

- *know itself*: the system should have detailed knowledge of its components, status, capacity, and connections with other systems; it will need to know the extent of its owned resources, those it can lend, and those that can be shared or should be isolated.
- *configure itself*: the system configuration should be done automatically, as must dynamic adjustments to that configuration to handle changing environments.

- *optimize itself*: the system should monitor its components and look for ways to optimize its working, like resource allocations, load balancing, different network traffic optimizations.
- *heal itself*: the system should be able to recover from faults that might cause some parts of it to malfunction.
- *protect itself*: the system should be capable of detecting and protecting resources from both internal and external attacks, thus maintaining overall system integrity.
- *adapt itself*: the system should be aware of its environment and the context surrounding its activity, and act accordingly, by finding rules for how best to interact with neighboring systems.
- *open standards*: the system should work in a heterogeneous environment and implement open standards; it cannot be a proprietary solution.
- *anticipatory*: an autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden; both the users and applications in the system should be unaware of the presence of the technology used to perform their functions.

Further on, we will describe how the different component types of the QADPZ system manifest autonomic characteristics (Constantinescu, 2003).

7.3.7.1 Self-knowledge

First, the system must have detailed knowledge about itself. In QADPZ this is accomplished by detecting all available computing resources and their current status. Each slave knows about its own local resources, while the master knows about all the available resources provided by the slaves contributing to the system. When the slave background application is started on one of the computers in the network, it automatically detects the hardware and software resources available on that computer. Hardware resources are, for example, system architecture, CPU type and speed, available physical memory, and available disk space. These characteristics of the computer can be obtained in different ways: by inquiring the operating system (e.g. the available memory and disk space), or by running some benchmark tests (e.g. CPU speed). Each operating system has its own way of providing such information, so that this auto-detection feature of the slave is dependent on the operating system.

However, it is a small part of the code and can be easily adapted for a new system. Software resources can be, for example, the operating system type and version, different shared system libraries and software applications available on the system. The slave is pre-configured to detect if certain software applications (e.g. compilers, interpreters, etc.) are available, and determines the installed version on that computer. Using this information, the slave service is creating a description of the computer and registers it to the master. In this way, the master will collect detailed information about each of the slaves

participating in the QADPZ system, keeping an overall knowledge about the whole system's resources, thus creating knowledge about itself.

7.3.7.2 Self-configuration

The software running on each slave computer is capable of upgrading itself whenever there is a new version of the software. This is done automatically on the slave side, without any user intervention, or system restart. The user only needs to specify to the master the new version of the slave program and its location for the different operating systems. The master will notify the slaves about the availability of a new version. Each slave will upgrade itself if it has an older version. However, the upgrade can be delayed if a specific slave is running a task, until the computation is finished. Any additional new slave, which connects to master will also be notified about a possible upgrade.

7.3.7.3 Self-optimization

The slave is also responsible for detecting if the computer is in use by any interactive user, or if the CPU resource is used by other applications. The first situation is detected by monitoring if there is an interactive session started on the computer: in Windows this is done by checking if the explorer application is running, while in Unix by checking for an X-Windows session. The second situation is detected by measuring the CPU load over a longer period of time (seconds, a few minutes). In any of these situations, the slave is considered unavailable, and will not be scheduled for executing computational tasks. Once the computer becomes available, its new status is reported to the master and scheduling of tasks becomes possible. This monitoring feature of the slaves is the first step in gathering information about resource utilization for the purpose of self-optimization of the system. The information is used by the master for scheduling the distribution of tasks to the slaves.

7.3.7.4 Self-healing

When a task is scheduled on one of the slaves, that slave receives a description of the task, which contains all the information needed to start it: the download addresses for the task to be executed and all the input files needed. All the files are downloaded locally on the slave and the computation is started. When the task is finished, the results are uploaded, every temporary files are removed and the master is notified about the end of the computation.

There are however certain situations when the execution of the task is interrupted, and which requires some kind self-healing mechanisms. One such situation is when the task started by the slave is crashing, due to a software problem in the executed program. The slave will detect such failure, then it will clean up any local temporary files, and notify the master about this. The master can either notify the user about the situation, or try to execute the task on a different platform slave, if possible.

Another situation is when a task is running and a user is starting an interactive session on that slave computer. Since interactive users have priority over any executing tasks, the running task will be interrupted. The task can be migrated to a different slave, or restarted, if migration is not possible, on a different slave. Migration can be done if the task program can provide the means to save the current state of the program and continue the execution from this point on a different computer. This has to be done inside each task. A future extension we are investigating now is to use check-pointing techniques. When the task needs to be interrupted, it is first check-pointed, the resulting memory footprint is transferred on the new slave, where the computation is resumed. Another self-healing situation is necessary when a task is running for too long and the local slave will stop the execution and notify this to the master.

The current implementation considers only one central master node. This can be a shortcoming in certain situations, where computers located in different networks are used together. The master node can also be subject to failures, software or hardware. A more decentralized approach is needed in this case. Currently, our high-level communication protocol between the entities, especially between the client and master, allows a master to act as a client to another master, thus making possible to create a *distributed master*, consisting of independent master nodes, which inter-communicate. Ideas from peer-to-peer computing will be used for implementing such a decentralized approach.

7.4 Get Started with QADPZ

In QADPZ, a small software program (*slave service*) runs on each desktop workstation. As long as the workstation is not being utilized, the slave service accepts tasks sent by the server (*master*). The available computational power is used for executing a task. Human system administration required for the whole system is minimal. We will now describe the features in detail.

7.4.1 User modes

Each installation of the system requires a local administrator, who is responsible for configuring the system and installing the *slave service* on desktop computers, and the *universal client* on user computers. Individual users, however, do not need to have any knowledge about the system internals. On the contrary, they are able to simply submit their executable or interpreted (such as Lisp or Java) program from a menu-driven command-line application, where they can specify:

- number of runs of the application;
- file path to the executable and command line arguments;
- input and output files (their names are automatically generated from the run number) either for all runs or for specified subset of runs;
- directories where the files reside;

- utilities to be run after individual tasks (typically to process the output files before another task is started);
- maximum time allowed for a task to execute;
- in what order ought the task groups be executed;
- hardware (disk, memory, CPU type and speed) and software (operating system, and installed programs) requirements of the application.

These project configuration parameters are saved into XML-structured file. The executable can be taken from a local disk or downloaded from any URL-specified address. The input and output data files are automatically transferred to slaves using a dedicated data www-server. The progress of execution can be viewed in any web browser. Each run corresponds to a task – the smallest computational unit in QADPZ. Tasks are grouped into jobs – identified by a group name and a job number. System allows control operations on the level of tasks, jobs, job groups, or users. If preferred by advanced users, the project file may be edited manually or generated automatically, see the example 1 and 2 below.

Example1: Simple library-type project file.

```
<Job Name="example">
<Task ID="1" Type="Library">
<RunCount>1</RunCount>
<TaskInfo>
<Memory
Unit="MB">64</Memory>
<Disk Unit="MB">5</Disk>
<TimeOut>3600</TimeOut>
<OS>Linux</OS>
<CPU>i386</CPU>
<URL>http://server/lib-
example.so</URL>
</TaskInfo>
</Task>
</Job>
```

Example 2: Simple executable-type project file.

```
<Job Name="brick">
<Task ID="1" Type="Executable">
<RunCount>15</RunCount>
<FilesURL>http://server/cgi-bin/</FilesURL>
<TaskInfo>
<TimeOut>7200</TimeOut>
<OS>Win32</OS>
<CPU Speed="500">i386</CPU>
<Memory>64</Memory>
<Disk>5</Disk>
<URL>http://server/slave_app.dll</URL>
<Executable Type="File">../bin/evolve_layer.exe
</Executable>
<CmdLine>sphere.prj 2 50</CmdLine>
</TaskInfo>
<InputFile Constant="Yes">sphere.prj</InputFile>
<OutputFile>sph/layout/layout.2</OutputFile>
<InputFile Constant="Yes">sph/sphere.1</InputFile>
<InputFile Constant="Yes">sph/sphere.2</InputFile>
<InputFile Constant="Yes">sph/sphere.3</InputFile>
<OutputFile>sph/logs/evolve_layer.log.2
</OutputFile>
<InputFile>sph/layout/layout.1</InputFile>
</Task>
</Job>
```

More advanced users can write their own client application that communicates directly with the master using API of the *client service library*. This allows submitting tasks with appropriate data dynamically. Finally, advanced users can write their own slave libraries that are relatively faster than executable programs and very suitable for applications with many short-term small-size tasks, i.e. with a high degree of parallelism. The communication between the system components is in human readable XML format and can optionally be saved into log-files, so that all the activity and possible failures can be traced. Extensive debug logs can be produced as well. The system provides basic statistics information on usage accounting.

7.4.2 Installation and maintenance features

All three main components of the system – client, master, and slave have their configuration files, which are well-documented and pre-configured for normal operation (only the IP address of the master needs to be modified). Each user of the system is authorized by user name and password and a special administration utility for their maintenance is provided. Manual configuration of the data www-server and master automatic startup is currently required, however automatic installation of the slave service on multiple PC workstations is solved for Win32/iX86 platform and is easy to setup for UNIX platforms.

Upgrade of the slave service is automatic, it is started by administration utility program – a new version is downloaded and started by each slave service. This allows large number of network computers to be easily integrated. The computers submitting jobs (the clients) can be offline while their tasks are running on slave machines. The master keeps track of the jobs and caches computation results when needed. In addition to a flexible storage place for the pre- or post-computational data, computational nodes can use common Internet protocols for data transfer to or from any other computer, including those not involved in the QADPZ system. Tasks are automatically stopped or moved to another slave when a user logs on to one of the slave workstations. The system does not support job checkpointing yet and does not handle restart of master computer. Adding these features has high priority. However, tasks can be moved from one slave to another at the request of the running task. This is equivalent to resubmitting a task with the addition that initial input data can be different from the original task.

The system installation, administration and use, and system internals are documented in the manual that is available from the project webpage.

7.4.3 Security

There are two conceptually different parts about security: system integrity and data integrity. In QADPZ we have primarily focused on system integrity, i.e. it should not be possible to use the system to gain access to any of the machines involved. Based on this we have the following requirements: only registered

users should be able to upload code to the slave machines, and slave code has limited access to the host environment.

In order to reach the first requirement, the master is fitted with a private/public key pair using the OpenSSL library (OpenSSL, 2007). All commands from clients to the master are signed with a username/password pair, so that only registered users can submit work. The passwords are saved in an encrypted form on the master host system.

The transmission of the username with password is always encrypted. Likewise, all commands from the master to the slaves are signed using the master's private key. The key-pair is defined at install-time. Slave code access to the system is defined by the owner of the system hosting the slave, and is thus outside QADPZ's control. The slave can be requested to download codeblocks from other locations. These locations are also outside the control of QADPZ. This means that if the system administrators of slave hosts give the software unnecessary system access, these computers will be vulnerable to unlawful users and to users ignorant of security issues. We pay this price for flexibility. In our setup, the slave is started under separate network user that has the disk read and write access only in a special temporary directory.

7.4.4 **Architecture**

The system consists of a central process called "master", a variable (high) number of computing processes on different computers in the network called "slaves", and a number of "client" processes, user applications, which generate tasks grouped in jobs. Slave component is run as a daemon or Windows service. Its first role is to notify the central master about its status and the available resources. These include: operating system type, processor information, CPU type, CPU speed nodes, physical memory available, local disk available, and existing software on the local system. An example of slave status message is shown beneath. Slave status message is sent from all computational nodes at regular intervals.

```
<Message Type="M_SLAVE_STATUS">
<Status>Ready</Status>
<SlaveInfo>
<Version>0.5</Version>
<OS>Win32</OS>
<CPU Speed="500">i386</CPU>
<Memory Unit="MB">32</Memory>
<Disk Unit="MB">32</Disk>
<Software Version="1.3.0">JDK</Software>
<Software Version="2.95.2">GCC</Software>
<Address>129.241.102.126:9001</Address>
</SlaveInfo>
</Message>
```

Another role of the slave is to launch an application (task) as a consequence of master's request. The application, in form of a library, executable, or interpreted program, is transferred from a server according to the description of the task, then it is launched with the arguments from the same task description. In case of executable and interpreted tasks, *universal slave library* is used. After the slave service library launches it, it first downloads the executable or interpreted program, either from an automatic data store (now implemented on top of www-server in Perl), or from a specified URL location. The universal library proceeds with downloading and preparing all the required input files. After the executable or interpreted program terminates, the generated output files are uploaded to the data store to be picked up later by the universal client, which originated the task. On Win32 platform, the user (or universal) slave libraries come in form of DLL module, while on UNIX platform they are dynamic libraries (this makes it difficult to port the application for example to Darwin/Mac OS, which doesn't support dynamic libraries).

The master is listening to all the slaves. This way, it has an overview of all the resources available in the system, similar to a centralized information resource center. It accepts requests for tasks from clients and assigns the most suitable computational nodes (slaves) to them. The matching is based on task and slave specifications and the history of slave availability. In addition, master accepts reservations for serial or parallel groups of computational nodes: clients are notified after resources become available. Master generates a report on current status of the system either directly on a text console – possibly redirected to a (special) file, or in form of an HTML document.

The client consists of the service library and a client user application or the universal client application. The client service library provides a convenient C++ API for a communication with the master, allowing controlling and starting jobs and tasks and retrieving the results. Users can either use this API directly from their application or utilize the universal client, which submits and controls the tasks based on an XML-formatted project file. In version 0.6 of the system, each job needs a different client process, although we are working on extending the client functionality to allow single instance of client to optionally connect to multiple masters and handle multiple jobs.

Communication in QADPZ is based on TCP/UDP, an unreliable communication protocol, in which packets are not guaranteed to arrive and if they do, they may arrive out of order. The advantage of UDP/IP over TCP/IP is that UDP is fast, reducing the connection setup and teardown overhead, and is connectionless, making the scalability of the system easier. The higher-level protocol is message based, and the size of the messages exchanged between the components of the system is small. Also, messages are exchanged only for control purposes. This makes UDP a very good option for our low-level communication protocol.

This layer, called UDPSocket, is also responsible for hiding operating system specific function calls, and making a more general interface for

communication. Because of the unreliable nature of the UDP protocol, an additional, more reliable level of communication is needed. This is based on message confirmation. Each message contains a sequence number, and each time it is sent, it is followed by an acknowledgement from the receiver. Each sent or received message is accounted, together with the corresponding acknowledge, and in case of not receiving an acknowledgement, the message is resent a few more times. An acknowledge and a normal message can be combined into one message to reduce the network traffic. This layer is called UDPConfirm, and permits both synchronous and asynchronous message sending. The next communication layer, *PostOffice*, has a similar functionality as the real life post office service. It delivers and receives high level messages – XML elements represented as instances of XMLData class. Both blocking and non-blocking modes are supported. Messages have a source and a destination address. Received messages can be kept by the PostOffice as long as needed, the upper layers in the system having the possibility to retrieve only certain messages, based on the sender's address. The PostOffice is also responsible for the encryption and decryption of messages, if necessary.

Messages exchanged are in XML format, in accordance with a strictly defined communication protocol between client and master, and between master and slave. Each message is represented as an XML element <Message Type="message_type">, see the example 3. XML elements are internally stored as objects of class XMLData, which in turn contain their sub-elements – other XMLData objects. Element attributes are instances of XMLAttrib class. These classes provide extensive functionality for manipulation with XML elements including input/output string and stream operations. When the data for slave user library are sent in the message, they are encapsulated inside of standard <![CDATA[]]> XML elements. We chose to implement our own lightweight class in order to achieve flexibility and easy extensibility of its functionality. Message based communication is used only for controlling the entities in the system. Shared libraries and executable files for task execution on the slaves, as well as data files for the computations are transferred using standard Internet protocols, like for example http, ftp, ldap, etc. For this, we are using the open source library "cURL" (cURL, 2007). Currently, the slave is using http to download files from a server (which can be the master itself, or another, specialized data server), but this can easily be changed to a different protocol.

8 The QADPZ usage on sourceforge.net

In this section, we present briefly the "history" of the QADPZ system since it has been uploaded to sourceforge.net (July, 2001), as well as its users' feedback and some other interesting reactions to this open source system. Within the appendix of this thesis, some raw feedback and reactions to the system are listed. These can be categorized into four main categories: feedback and support requests from users who use QADPZ for their research and development tasks, forum discussions, citations in papers, and working assignments, based on QADPZ features, for students from some universities. Links to the QADPZ system can be found also in distributed computing directories on the web and in several blogs.

SourceForge.net is the world's largest Open Source software development web site. SourceForge.net provides free hosting to Open Source software development projects with a centralized resource for managing projects, issues, communications, and code. SourceForge.net is a centralized location for software developers to control and manage open source software development, and acts as a source code repository. Furthermore, SourceForge is a collaborative revision control and software development management system. It provides a front-end to a range of software development lifecycle services and integrates with a number of free software/open source software applications (such as PostgreSQL and Subversion).

SourceForge.net has offered free access to hosting and tools for developers of free software/open source software for several years, and has become well-known within such development communities for these services. Just as important, SourceForge is the place to "see and be seen" for up and coming open source projects. Here, developers are chatting, sharing, rubbing elbows, strutting their stuff with other developers or watching each other build. It's a global community of coder geeks, just jonesing to give birth to that next line of Java or PHP or Perl.

SourceForge.net is operated by Sourceforge, Inc. (formerly VA Software) and runs a version of the SourceForge software, forked from the last open-source version available. A large number of open source projects are hosted on the site (it had reached 155,585 projects and 1,658,777 registered users as of August 2007), although it does contain many dormant or single-user projects.

When the site opened in November 1999, growth was respectable, if modest. At the time, only those with a deep technical background knew the term "open source". Though the site offered myriad free tools, only a small crowd of projects registered by the end of the year. That soon changed. By the end of 2000, SourceForge had thousands of projects registered; by the end of 2001, almost 30,000 were coding away. And the following year, the flood commenced. Since 2002, they claim that a hundred projects a day are added. Fast forward to 2007 and SourceForge is now home to a sprawling universe of

open source developers. It's an intense hive of software creators. Some 150,000 projects – and growing – reside there, covering every conceivable computing function. In the Figure 8.1, a sourceforge sample page on distributed computing projects can be seen.

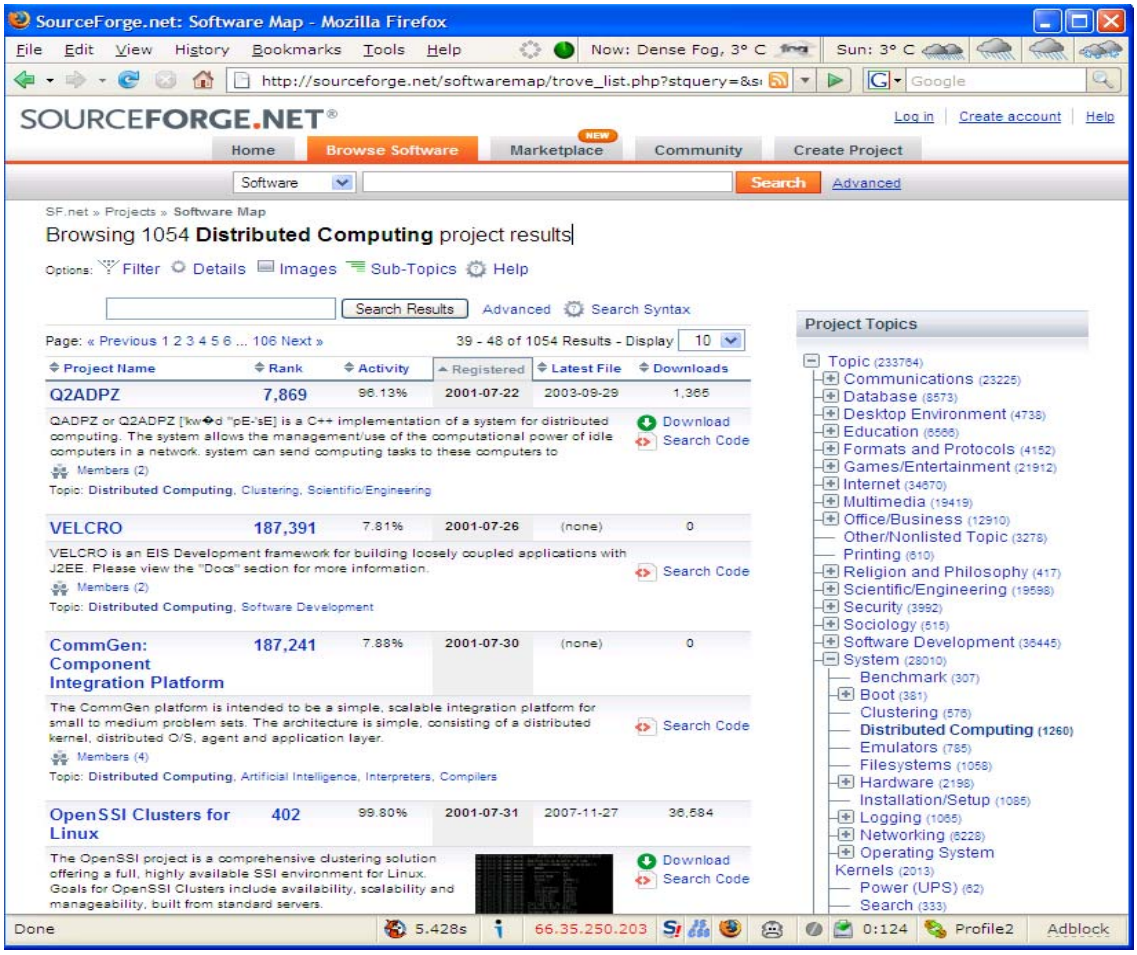


Figure 8.1 QADPZ in Distributed Computing projects

The QADPZ system had registered on sourceforge.net on July 2001, and uploaded its first public version (v 0.4) in September 2001. The QADPZ system is one of the first projects on desktop grid computing that have been registered on the sourceforge site. On the site there are available the source code files, user and developer manual, system documentation and papers about the system. The main page of the project on the sourceforge.net website is presented below (Figure 8.2).

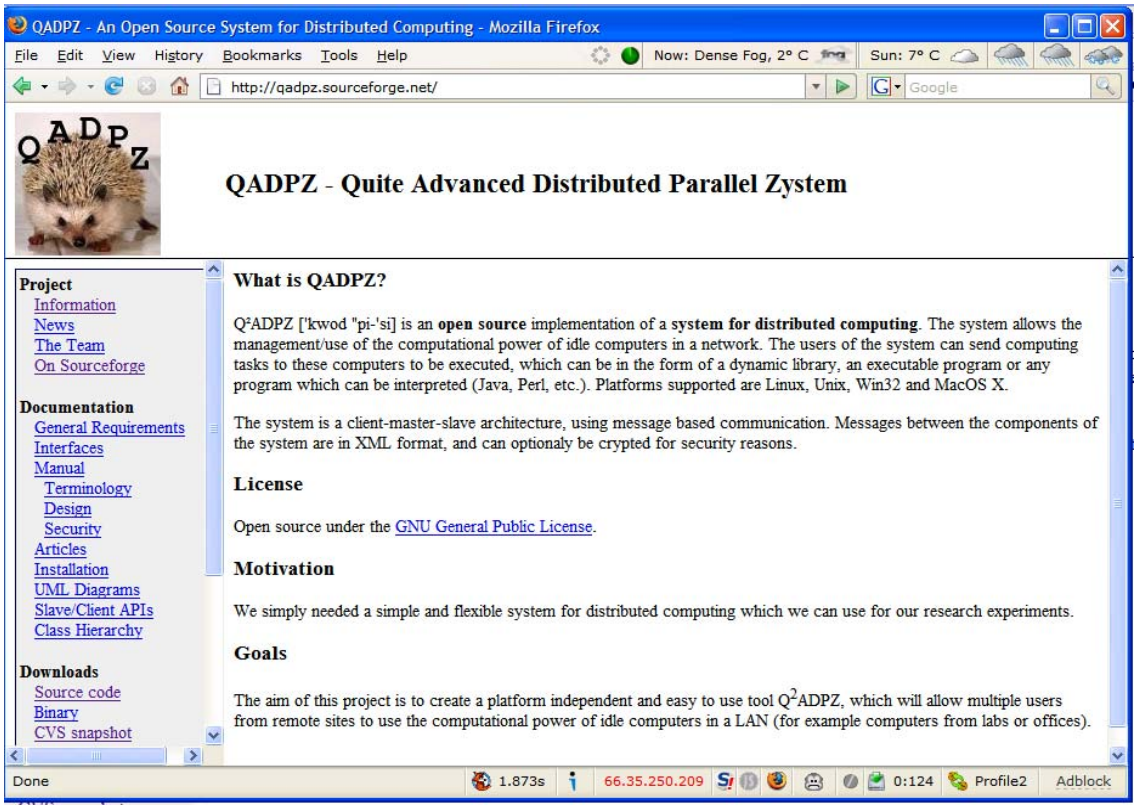


Figure 8.2 QADPZ home page

A summary on the QADPZ system can be seen also on the sourceforge.net website, as it is illustrated with the screenshot underneath (Figure 8.3):

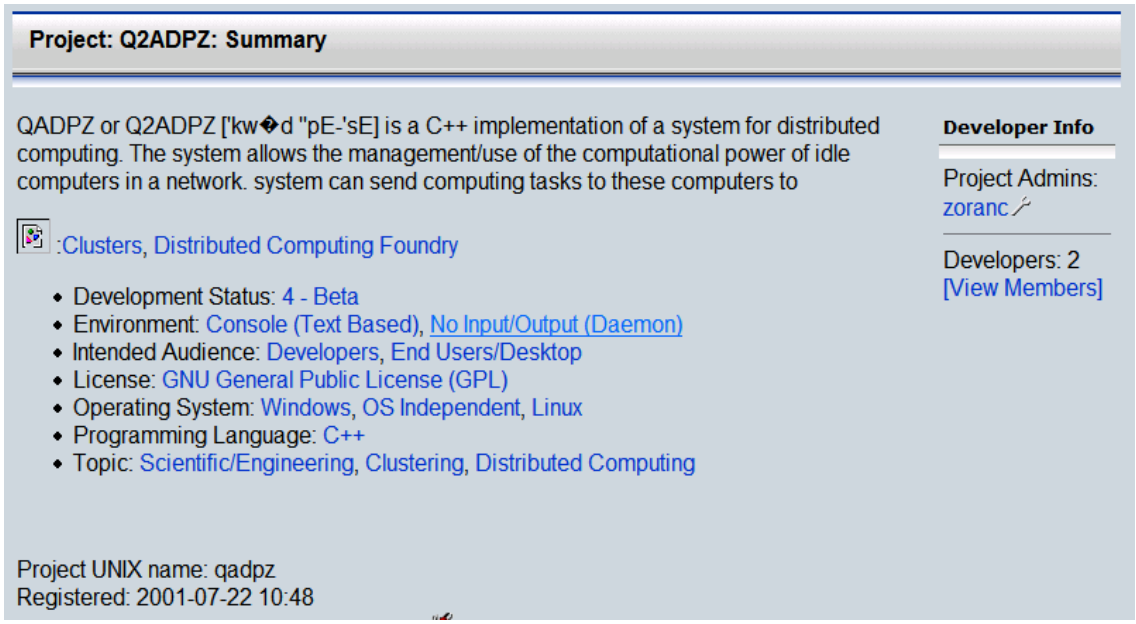


Figure 8.3 QADPZ project summary

Some statistics with source code that have been downloaded since project's registration time on sourceforge.net are presented in Figure 8.4 (December, 2007):

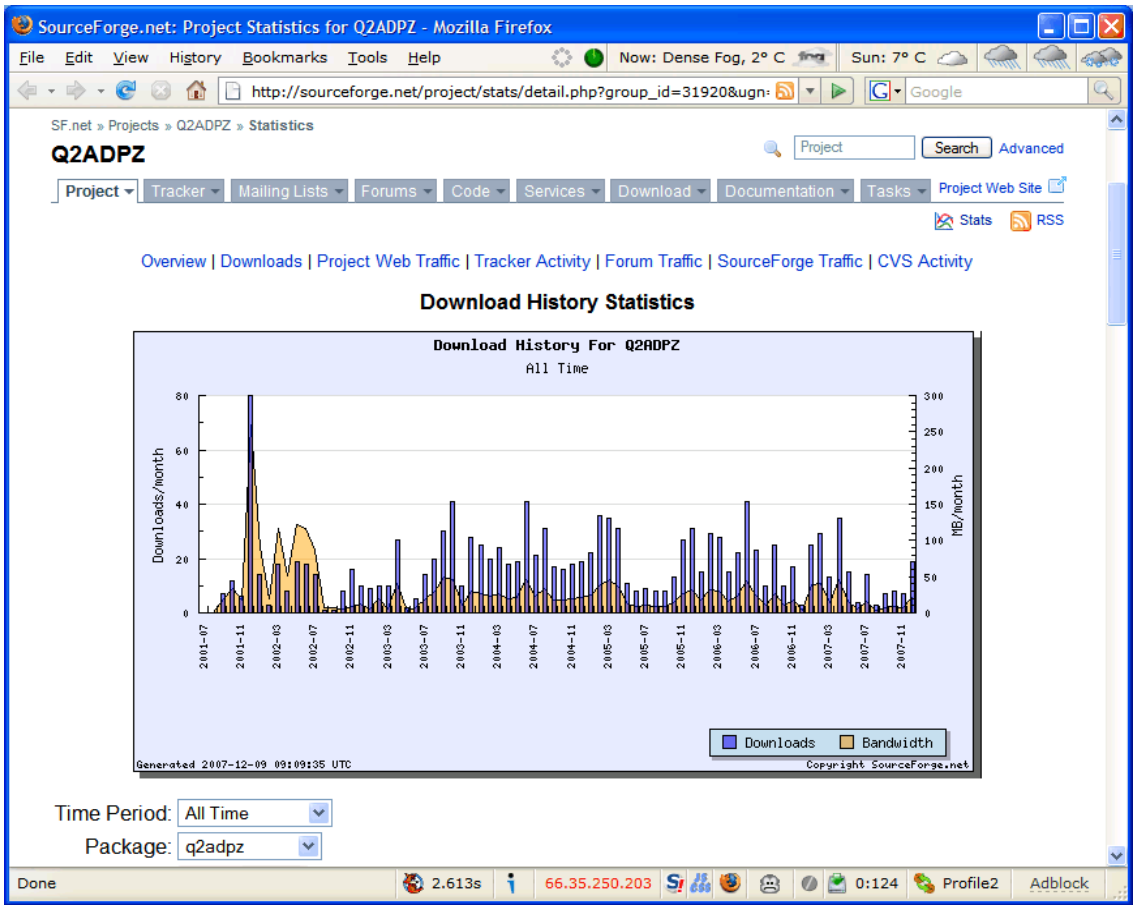


Figure 8.4 QADPZ download statistics

Each version of the program have been downloaded several times, as it can be seen in the following Figure 8.5.

Package	Release (date)	Filename	Size (bytes)	Downloads	Architecture	Type
q2adpz						
Latest	0.8 (2003-09-29 17:00)					
		q2adpz-0.8-20030930.tar.gz	1082172	927	Platform-Independent	Source .gz
	0.8beta (2003-06-03 17:00)					
		q2adpz-0.8beta-20030604.tar.gz	1742491	76	Platform-Independent	Source .gz
	0.7 (2002-10-09 17:00)					
		q2adpz-0.7-20021010.tar.gz	599099	128	Platform-Independent	Source .gz
	0.6 (2001-12-17 16:00)					
		q2adpz-0.6-20011218.tar.gz	6439390	108	Other	Source .gz
	0.5beta (2001-10-15 15:28)					
		q2adpz-0.5beta-20011015.tar.gz	2997486	101	Any	Source .gz
	0.4 (2001-09-22 17:00)					
		q2adpz-0.4-20010923.tar.gz	2666269	23	Any	Source .gz
Totals:	6	6	15526907	1363		

Figure 8.5 QADPZ downloads

And, finally, we can see beneath some statistics with project's webhits since QADPZ project has been registries on sourceforge.net.

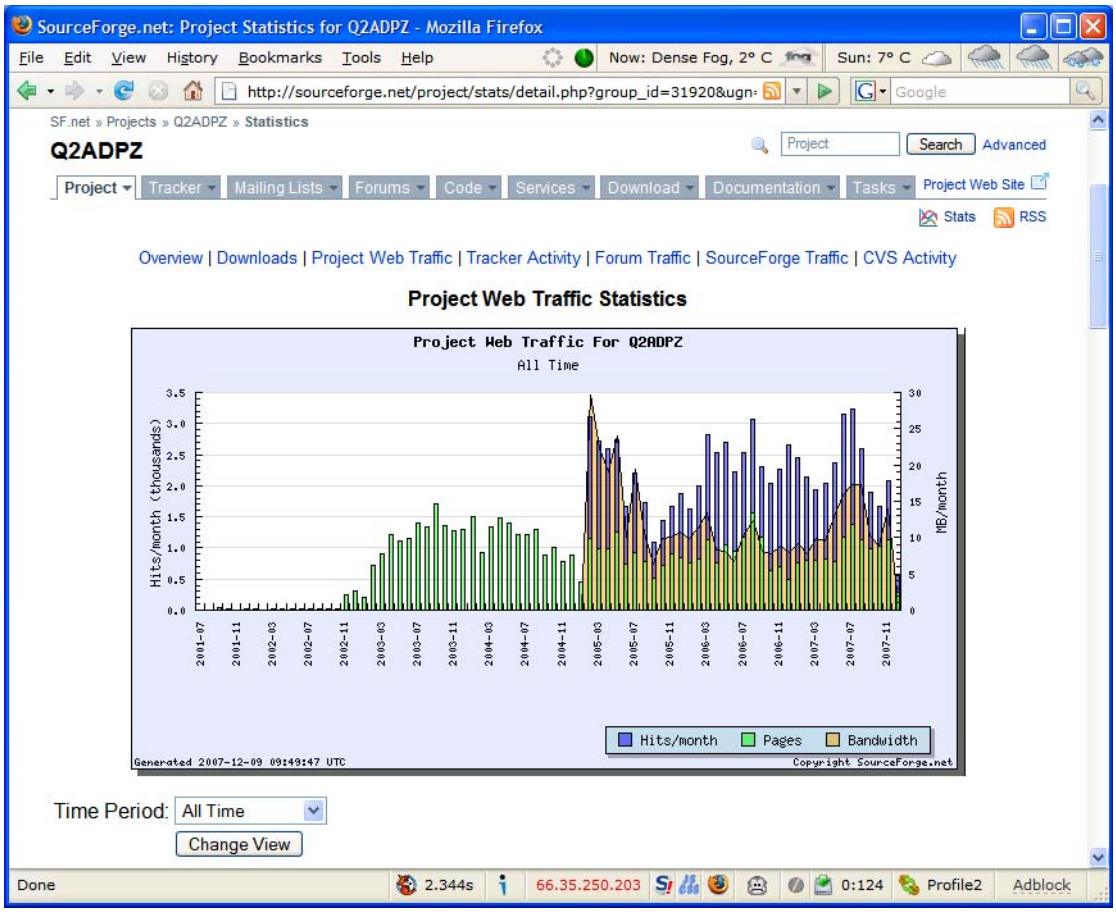


Figure 8.6 QADPZ webhits statistics

In this chapter we presented briefly the "history" of the QADPZ system since it has been uploaded to sourceforge.net. Within the appendix of this thesis users' feedback and some other interesting reactions to this open source system are listed. These can be categorized into four main categories: feedback and support requests from users who use QADPZ for their research and development tasks, forum discussions, citations in papers, and working assignments for students from some universities, based on QADPZ features.

To conclude we should mention that from interaction with the system users, we have gained insight in their perspective and needs, and we have used that feedback to improve our conceptual model, design and implementation of the QADPZ system.

9 Scientific Computing and Visualization Experiments

9.1 Computational Resource Monitoring

Several desktop grid systems have been successfully used for many high throughput applications. Yet, in an organization or enterprise setup there has been little insight into the temporal structure of resource availability. We present some of our observations regarding our desktop grid's availability of computing resources. The results are from an undergraduate student laboratory of 70 personal computers from the university. It is well known that the highest availability of computers is during night, when few of the students are using lab computers. We are more concerned about resource availability during work hours, with the idea of using desktop grid resources for interactive tasks, like group visualizations or interactive presentations, where a group of researchers and students are working together.

We present our observations between 08:00 in the morning and 20:00 in the evening during several days, when computers from labs are actually used intensively (e.g. project deadlines, homeworks, lab hours). The results from the following plots show the actual number of computers available for computations from the running desktop grid. From a rough estimate, we can say that computers are available for computations about 50-60% of the time during week-days, between 08:00 and 20:00. During the night, the availability is close to 95-100%. This amounts to approximately 75-80% availability of computers during a 24hours interval of a working day, growing to 90-95% during weekends. As a conclusion, we can say that, based on our available measurements, we can say that there is a lot of computing power available in such laboratories, which can easily be used for scientific experiments, provided that an appropriate resource-harvesting framework is available.

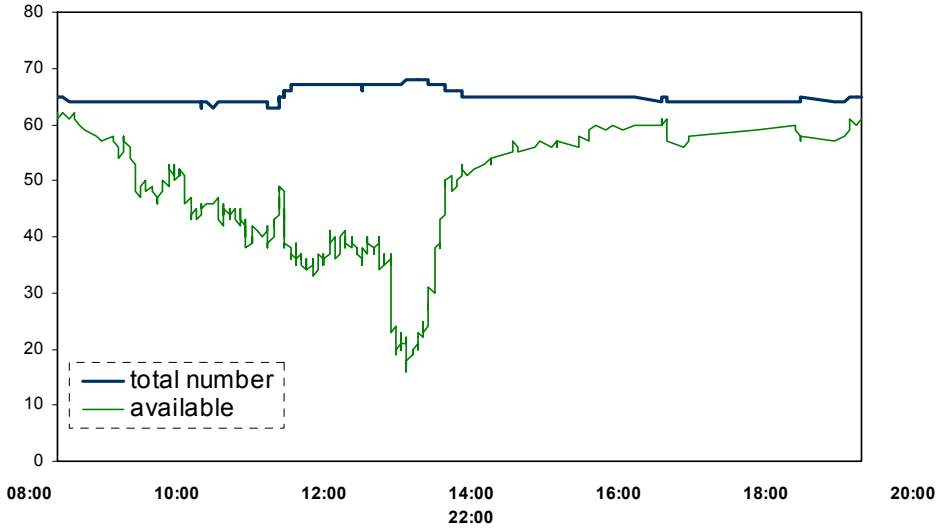


Figure 9.1 Available desktop computers in laboratory (day 1)

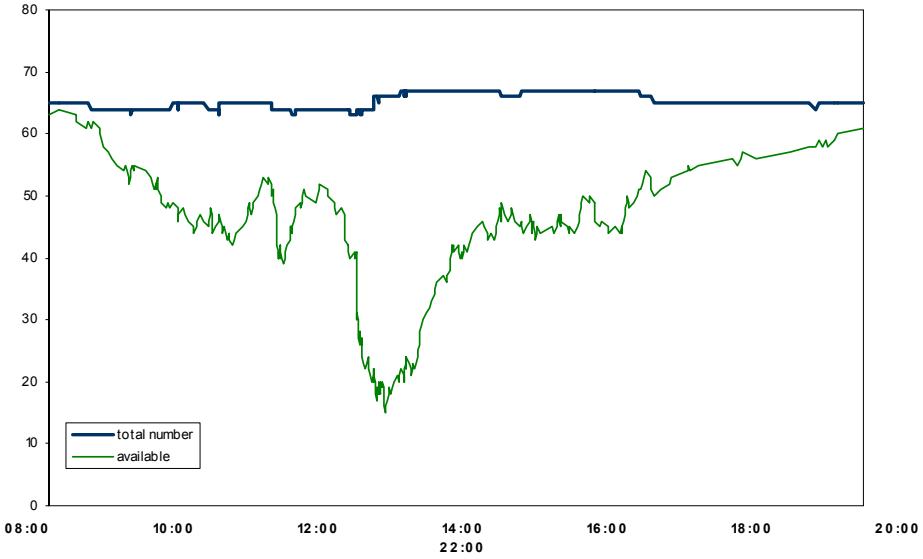


Figure 9.2 Available desktop computers in laboratory (day 2)

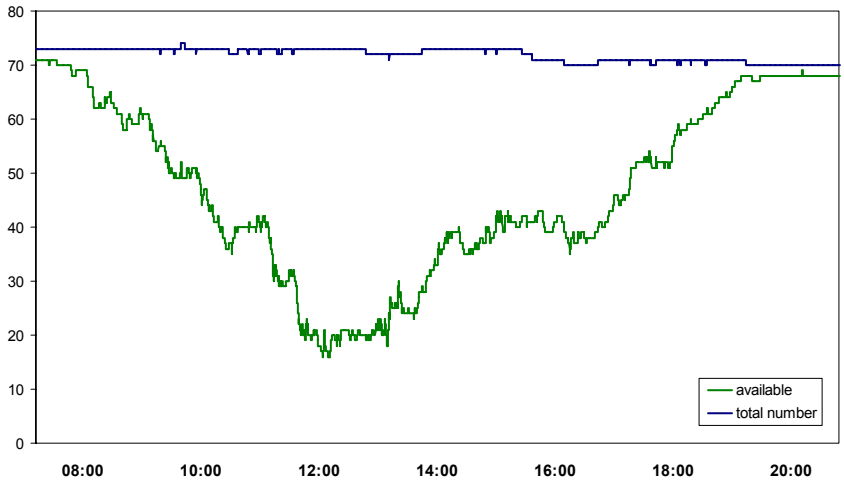


Figure 9.3 Available desktop computers in laboratory (day 3)

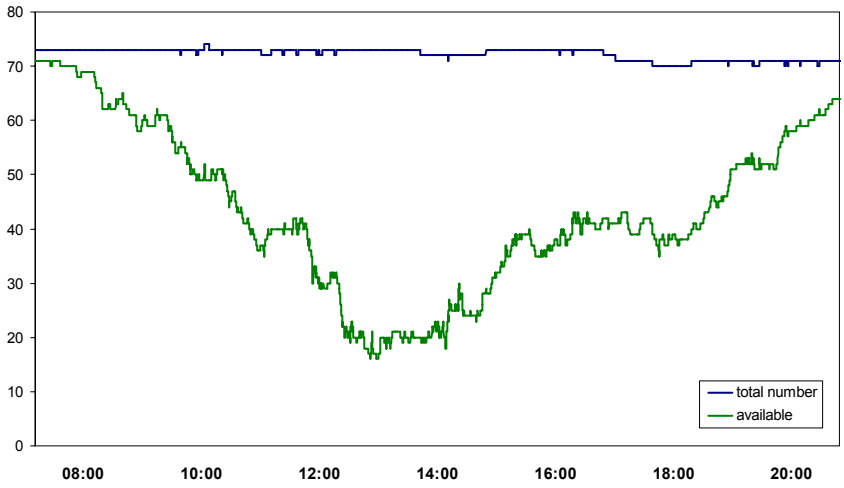


Figure 9.4 Available desktop computers in laboratory (day 4)

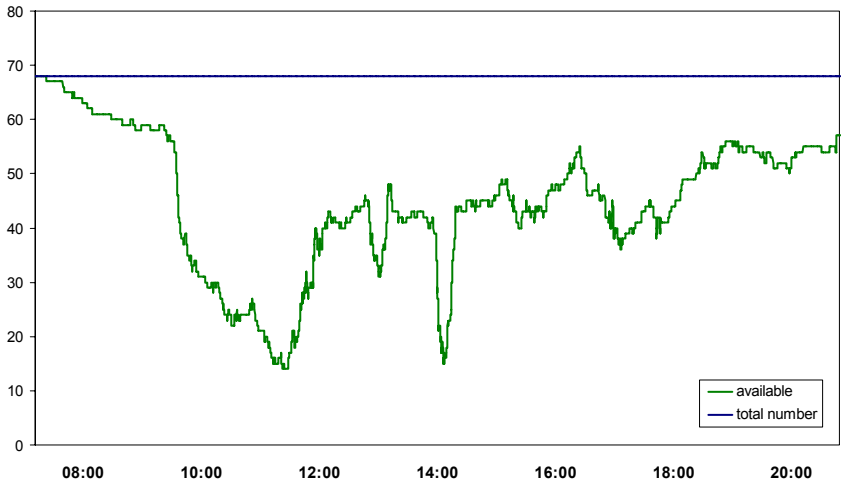


Figure 9.5 Available desktop computers in laboratory (day 5)

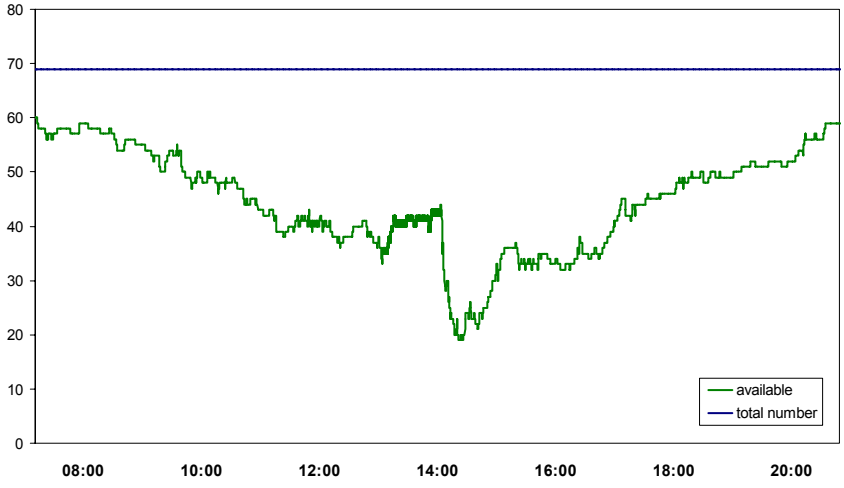


Figure 9.6 Available desktop computers in laboratory (day 6)

9.2 Real word problem - Trondheim fjord

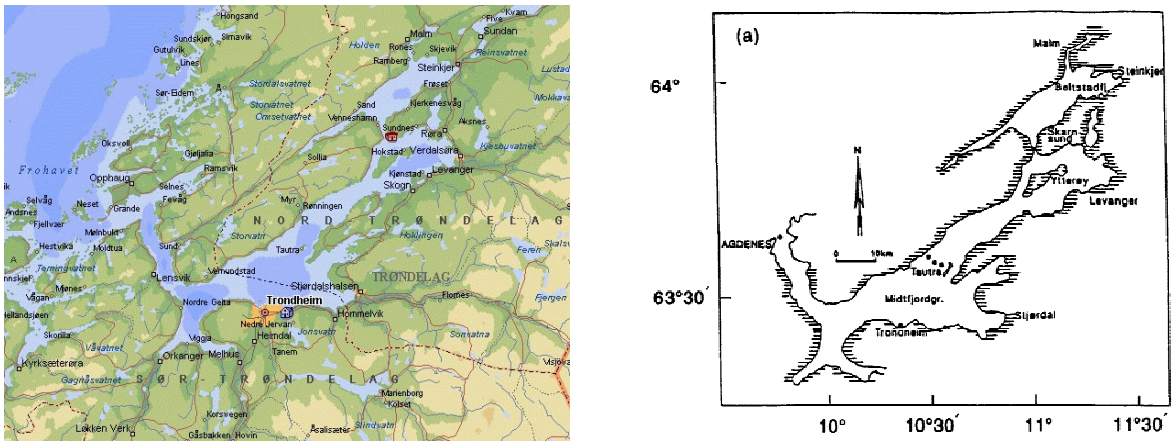


Figure 9.7 Maps of the Trondheim fjord

Geophysical circulation modeling is an increasingly important area for several reasons. One is the growing concern for environmental and ecological issues. This relates to problems of different scales, from global issues to more local questions about water pollution in coastal areas, estuaries, fjords, lakes, etc. To analyze such problems, there is a need to predict the flow circulation and transport of different materials, either suspended in water or moving along the free surface or bottom. The numerical model is based on a finite element formulation. It is believed that the finite element flexibility is advantageous for applications in restricted waters, where the topography is usually complex. The basic mathematical formulation is given by the Navier-Stokes equations. (Utnes and Brors, 1993). The figures above show a map of Trondheimsfjorden, a typical Norwegian fjord that is located on the coast of central Norway. Detailed topographical data are used to interpolate the depth data to the element mesh. Figure 9.7 illustrates the topography of the actual domain. The horizontal element mesh is shown in Figure 9.8. It consists of 813 biquadratic elements with 3683 nodes, and there are 17 levels in the vertical direction with fine grading close to the bottom boundary. This grid is assumed to be detailed enough to describe the main flow field of the fjord.

Shading the discretized cells according to the value of the scalar data field performs color coding. For better appreciation of continuum data the color allocation is linearly graduated. Using directed arrows also represents the velocity vector field. Large vector fields, vector fields with wide dynamic ranges in magnitude, and vector fields representing turbulent flows can be difficult to visualize effectively using common techniques such as drawing arrows or other icons at each data point or drawing streamlines. Drawing arrows of length proportional to vector magnitude at every data point can produce cluttered and confusing images. In areas of turbulence, arrows and streamlines can be difficult to interpret. Line Integral Convolution (LIC) (Cabral and Leedom, 1993) is a powerful technique for imaging and animating vector

fields. The image is created beginning with a white noise that is then convoluted along integral lines of the given vector field. That creates a visible correlation between image pixels that lie on the same integral line. The local nature of the LIC algorithm suggests a parallel implementation, which could, in principle, compute all pixels simultaneously. This would allow for interactive generation of periodic motion animations and special effects.

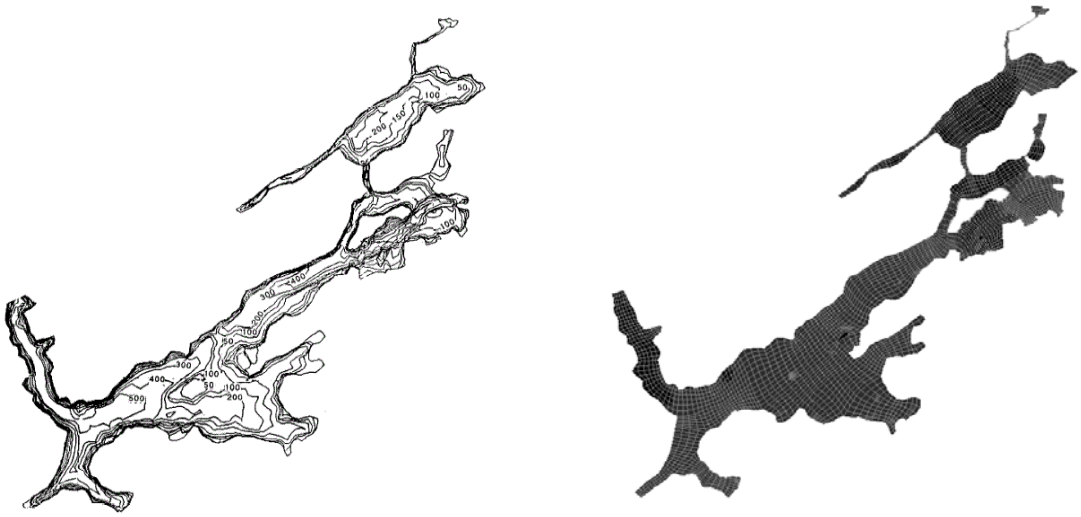


Figure 9.8 Trondheim fjord (left: topography, right: grid)

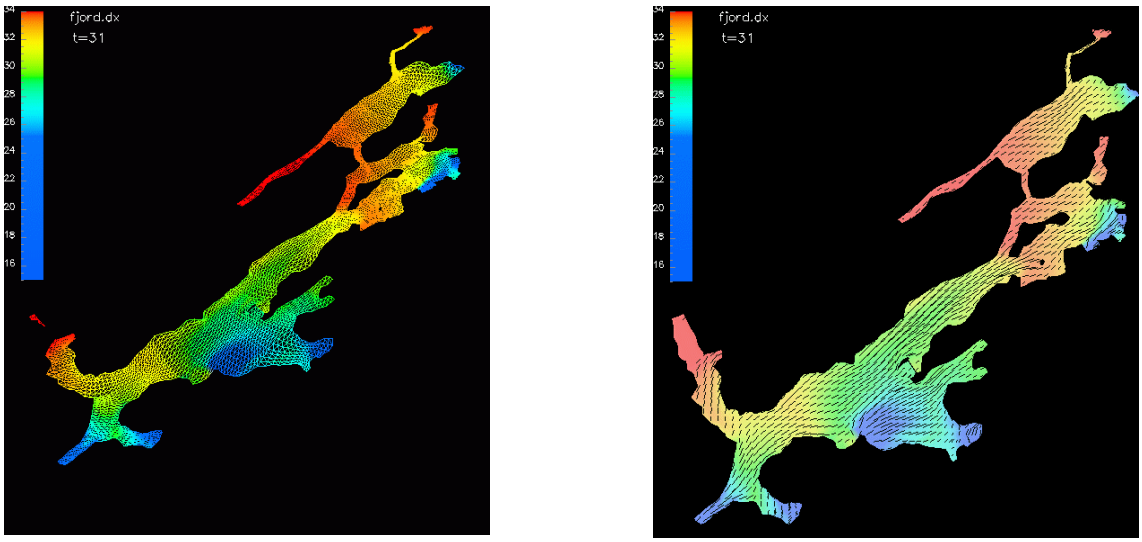


Figure 9.9 Grid colored by salinity concentration

Note. Grid colored by salinity concentration; Max 34 ppt - parts per thousand (mg/l)

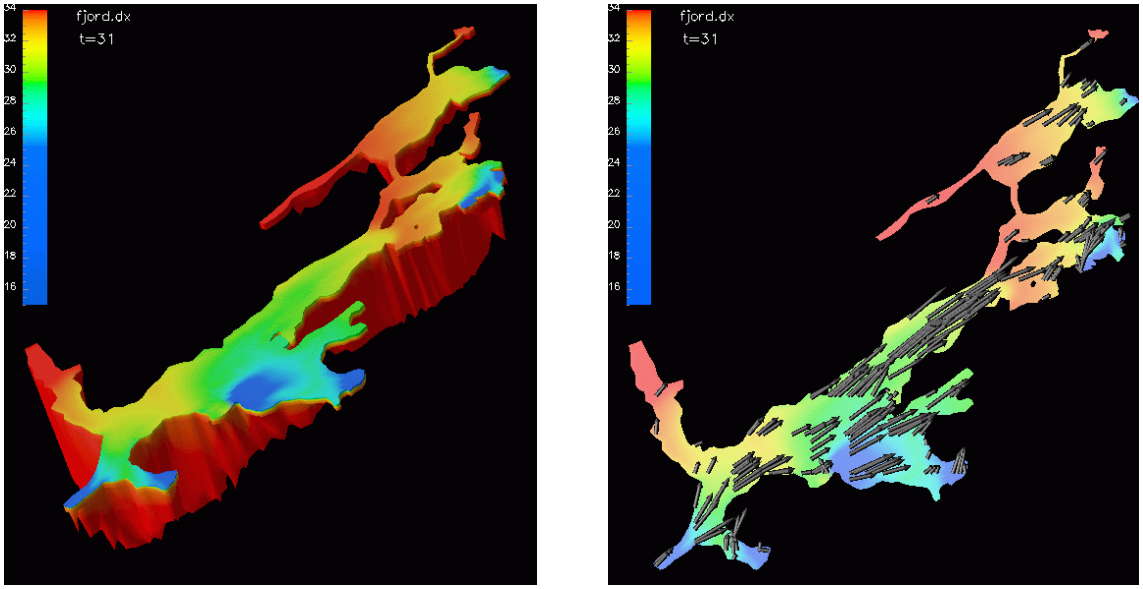


Figure 9.10 Trondheim fjord model

Note. Color by salinity concentration - 3D model representation with isosurface representation.

Figure 9.11 Velocity vector field - LIC representation

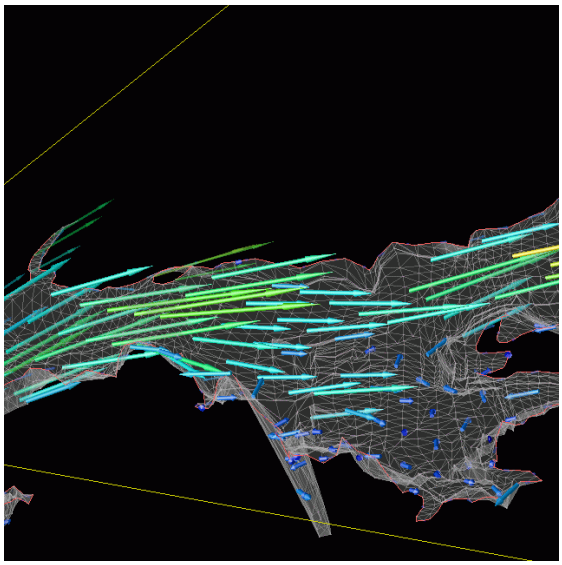


Figure 9.12 3D representation surface- vector field top layer

9.3 Fluid flow around a cylinder - simulation

We present some experiences with running a typical Computational Fluid Dynamics problem in QADPZ environment. The numerical method for solving the incompressible Navier–Stokes equations is used in a test case for the system. An evaluation of the performance of the system is presented, by comparing it with running the same simulation on a typical dedicated cluster environment.

To solve the incompressible Navier–Stokes equations we use a version the well–known projection method, in which equations for the velocity

components and pressure are solved sequentially at each time step. Solving the discretized, fully coupled equations can be very expensive due to the nested iterations, and this decoupling procedure is found to be computationally efficient for transient problems, particularly for higher Reynolds numbers. In general, the separation of pressure and velocity can be performed on both the continuous equations and the discretized equations. While the latter is attractive due to the straightforward interpretation of boundary conditions, these methods give a significantly more complicated pressure equation and we therefore prefer a splitting at the differential level. The Galerkin finite element method is used to discretize in space. The pressure is fixed in one node to ensure a unique solution, and has homogeneous Neumann conditions on all boundaries. Implementation of the flow solver was done is C++ using the object oriented numerical library Diffpack. A parallel version of Diffpack (Langtangen et al., 2000) was used, which is based on a standard Message Passing Interface (MPI) for communication. We used Linux as a development platform. An MPI library with a subset of the most used MPI calls was implemented on top of QADPZ's communication protocol. This QADPZ-MPI library allows us to use QADPZ as a straightforward replacement communication system for the MPI based Diffpack library (Karniadakis and Kirby, 2003).

A series of tests were performed on a dedicated cluster of 40 PCs, with Athlon XP 1.46GHz CPU, 1 GByte memory, and 100 MBps network interconnection between nodes, running a Linux distribution. First, we used the original implementation of the solver software, which is using MPICH as a parallel communication protocol, to run the cluster version of the simulation. Second, we recompiled the solver using the QADPZ-MPI library to create the distributed computing version of the simulation. The solver was run using exactly the same computers from the cluster (i.e. identical hardware setup). A third test case was using a pool of 8 computers with similar hardware specifications. These computers were ordinary desktop PCs from our labs, connected to our LAN, together with other computers. Simulations were done in two different times of the day: during the night, when network traffic in the LAN is minimal, and during working hours, when LAN traffic is much higher.

The first set of results is from simulating some time steps of an oscillating flow around a fixed cylinder in three dimensions. The grid has 81600 nodes and is made of 8-node isoparametric elements, while the coarse mesh (used for pressure preconditiong) has approximately 2000 nodes. Results of the execution times are presented in Figure 9.13. We run the same simulation in three different parallel settings for the underlying MPI library:

- the MPICH library from Argone National Laboratory (Gropp and Lusk, 1996, Gropp et al., 1996),
- the QADPZ MPI library using LZO based compression for communication,
- the QADPZ MPI library using bzip2 based compression for communication.

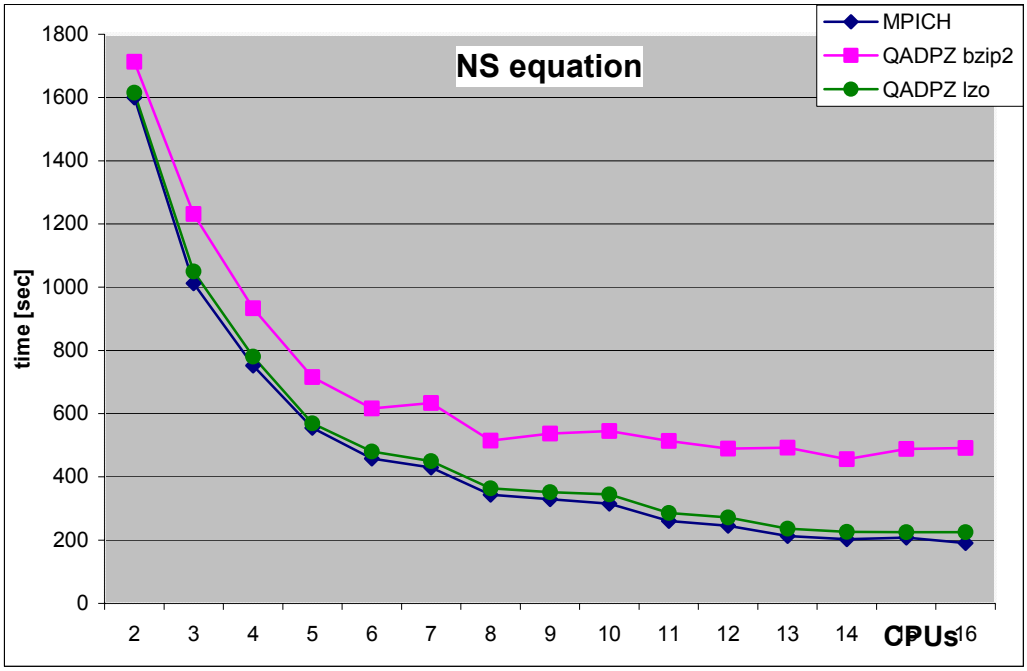
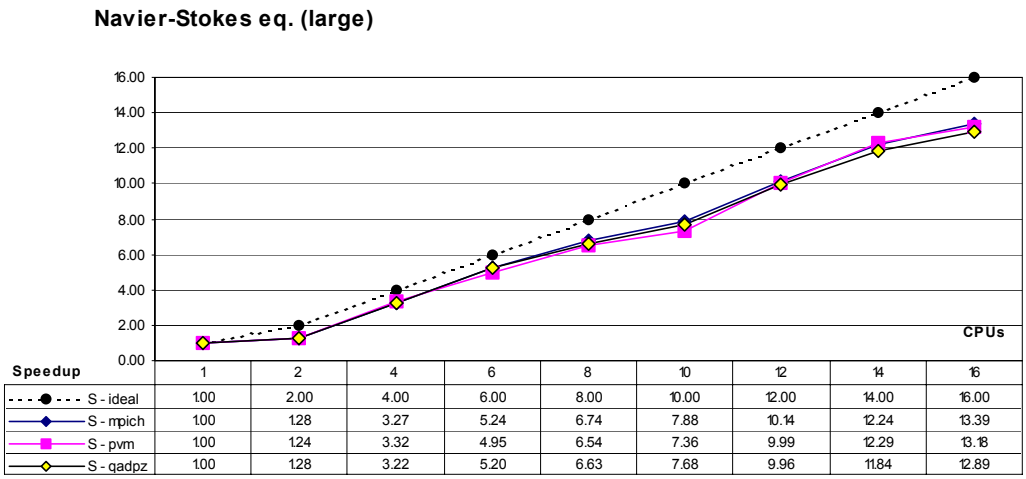


Figure 9.13 Execution times for the solver

A second set of results presented is from simulating some time steps of an oscillating flow around a fixed cylinder in three dimensions. The grid has 307296 nodes and is made of 294912 elements (8- node isoparametric). A coarse grid was used for pressure preconditioning, which had 2184 nodes and 1728 elements. Three sets of simulations were done, using MPICH, PVM and QADPZ-MPI as communication libraries. For each set of simulation, a maximum of 16 processors (Athlon AMD 1.466 GHz) were used. Running times were between around 240 minutes (1 processor) and 18 minutes (16 processors). Speedup results from these simulations, presented in figure 2, show that the performance of our system is comparable to other similar systems based on the message-passing interface. The advantages of our system are as follow: The installation of the slave on the computational nodes is extremely easy, only one executable and one config file being needed, and no root/admin access is necessary. Upgrade of the slaves is done automatically from the master, without any administrator intervention. Also, there is no need for a shared file system, since each of the slaves is downloading by itself the needed files. The slaves systems can have different operating systems, and there is no need for remote access to them (using rsh/ssh type of protocols).



Speedup	1	2	4	6	8	10	12	14	16
S - ideal	1.00	2.00	4.00	6.00	8.00	10.00	12.00	14.00	16.00
S - mpich	1.00	1.28	3.27	5.24	6.74	7.88	10.14	12.24	13.39
S - pvm	1.00	1.24	3.32	4.95	6.54	7.36	9.99	12.29	13.18
S - qadpz	1.00	1.28	3.22	5.20	6.63	7.68	9.96	11.84	12.89

Figure 9.14 Speedup for the simulation

9.4 Fluid flow around a cylinder - visualization

The results are from simulating some time steps of an oscillating flow around a fixed cylinder in three dimensions. The second experiment is with three cylinders. The grid has 81600 nodes and is made of 8-node isoparametric elements, while the coarse mesh (used for pressure preconditiong) has approximately 2000 nodes. As in the previous case, numerical simulation is done using the Navier Stokes equations.

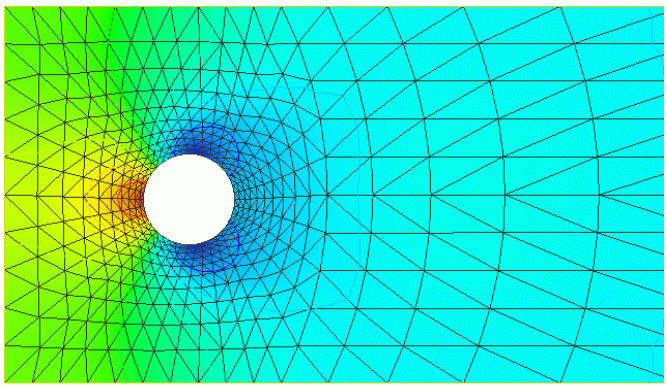


Figure 9.15 Flow around cylinder grid

In the above figure the 2D domain grid is represented using triangular elements, and the coloring is done using the pressure scalar field.

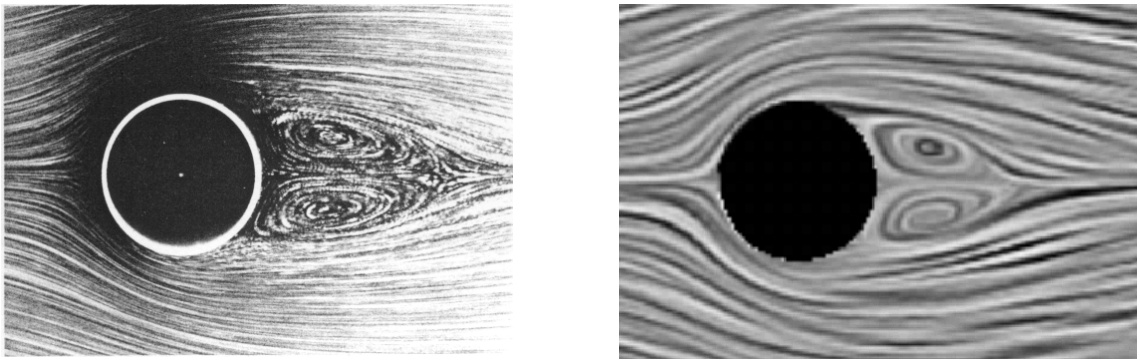
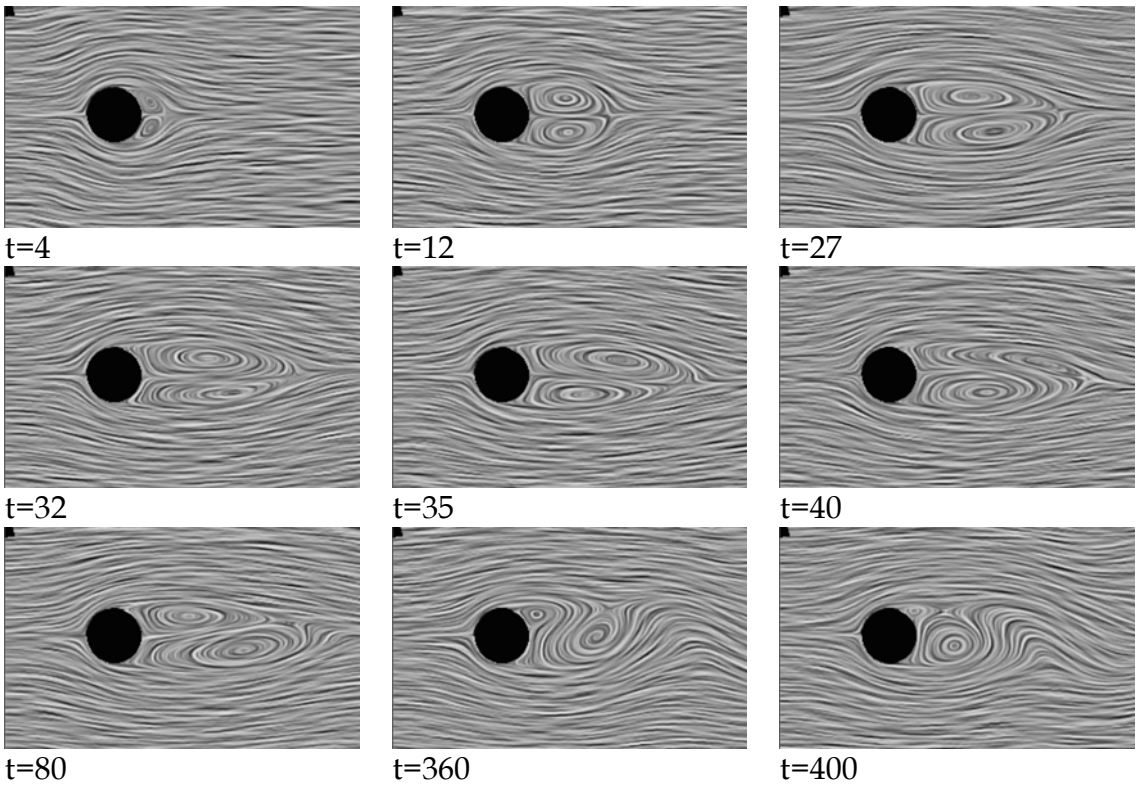


Figure 9.16 Flow around cylinder (measurement and simulation)

The left figure is a real life experimental image for Re=26 (Van Dyke, An Album of Fluid Motion), and the right is a simulation image using LIC vector field representation for Re=20.

Note. Simulation experiment for Re=100 at different time steps.



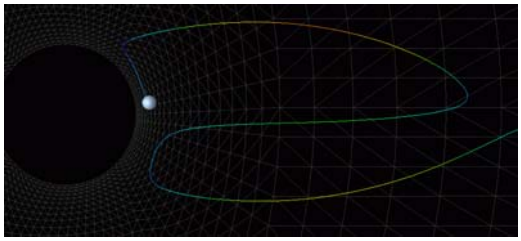


Figure 9.17 Streamline

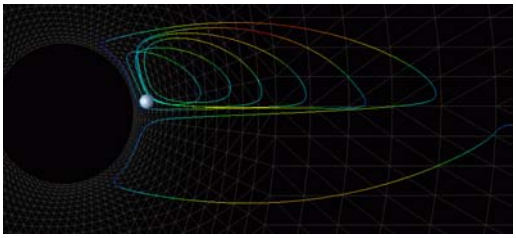


Figure 9.18 Streakline

Streamlines are a family of curves that are instantaneously tangent to the velocity vector of the flow. This means that if a point is picked then at that point the flow moves in a certain direction. Moving a small distance along this direction and then finding out where the flow now points would draw out a streamline.

Streaklines are the locus of points of all the fluid particles that have passed continuously through a particular spatial point in the past. This can be found experimentally by releasing dye into the fluid in a time period at a fixed point and then at a later time finding out where the dye was.

Note. Simulation experiment for Re=100 at different time steps, with 3 cylinders.

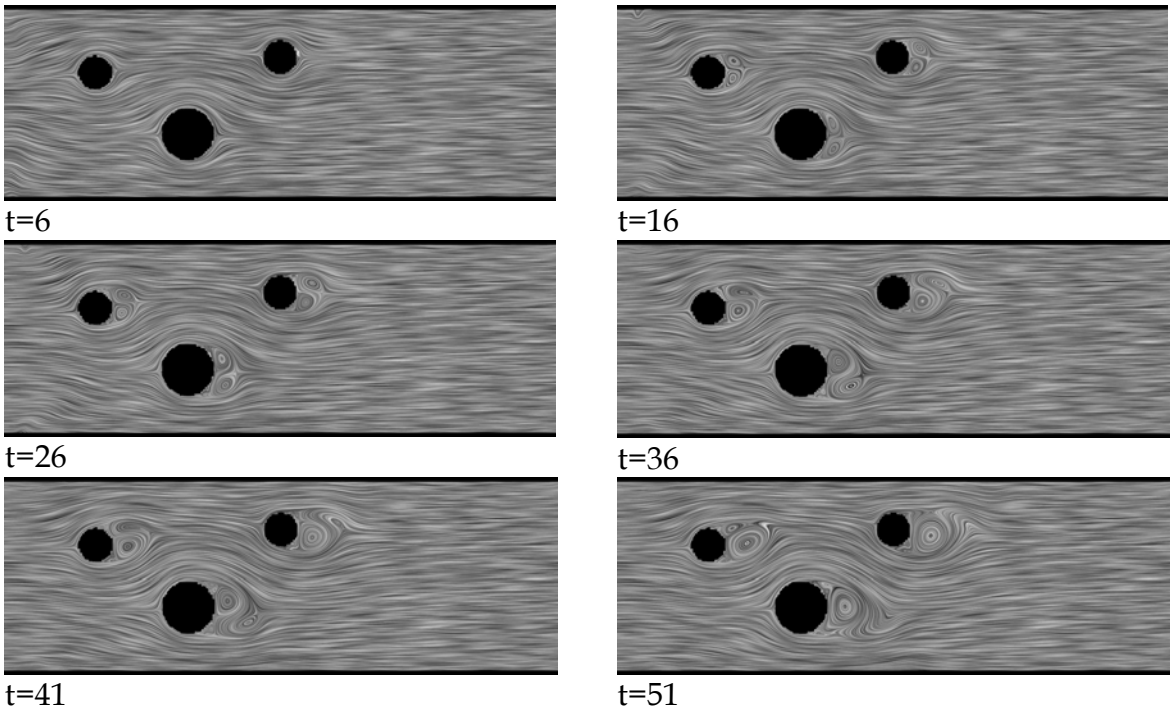


Figure 9.19 Flow around 3 cylinders

9.5 Utilizing QADPZ for Evolutionary Computation

To evaluate the system, the version 0.6 of the system in artificial evolution of layers of 3D LEGO models has been used (Petrovic, 2007). A 3D model was decomposed into individual layers. The layout of each layer, i.e. the placement of LEGO bricks was evolved by a separate task. The input and output files were automatically transferred by the universal client. To obtain statistically significant data, tens of independent runs were required. QADPZ installation included 70 high-performance PentiumIII 733MHz workstations located in a student laboratory. Their status can be on or idle during the night, and except of the exercise deadline season approximately 30-50% idle also during the day.

They received results worth many weeks of single computational time within approximately 3 days time with no configuration overhead, by simply submitting their executable to QADPZ. Evolutionary Algorithms (EA) are highly parallel stochastic search methods for finding approximate solutions useful when no deterministic algorithm generating good solutions is known. They are inspired by the Darwinian natural evolution principles, and work with a population (a set) of solutions that survive, mate and get mutated from generation to generation based on their performance (fitness). In each generation, all individuals in the population have to be evaluated independently. That is where it is natural to parallelize the execution of the evolutionary algorithms. Some flavors of EA work on multiple populations that evolve independently (island models) – and for them another natural place for parallelizing is allocating one (or several CPUs) for each sub-population. Alternately, evaluating a single individual can also be performed in parallel on several CPUs, if the objective function is suitable for parallelizing.

When setting up a distributed EA, one typically uses a combination of a package for distributed computation and a package for EA. The distributed computation package will be responsible for delivering the inputs/outputs to/from the computational nodes, and submitting the tasks to the nodes automatically. The user has to configure which code and data have to be processed. Usually the user specifies at how many nodes he runs a particular application, or optionally what would be the topology of the parallel virtual computer. The user can implement the communication between the computational nodes either through the shared file system, message-passing, or sockets, or simply rely on the parallelization features provided by the chosen EA package. In our case, the evolutionary robotics experiment was based on the GaLib package, which does not support parallelization, and thus we needed a distributed computing package.

9.6 Adaptive Compression for Remote Visualization

In order to understand better the issues from Scientific Visualization and to match them with the capabilities provided by QADPZ we did make an

experiment of remote visualization. Remote visualization using client-server environments allow users to access large datasets. One possible solution for that is the use of compression techniques, where images are generated and compressed at the servers' side, then the encoded images are transferred over a data network, decompressed and displayed at the clients side. One of the problems in remote visualization is to increase the frame rate for the user.

A possible solution is to reduce the amount of data transferred over the network, in our case to choose an efficient compression algorithm. However, the better a compression algorithm is, the more computing is necessary for both compression and decompression, increasing the time needed to process the image. The choice of the most efficient compression depends also on the content of the image. Therefore we have proposed an innovative intelligent adaptive compression method for selecting different image compression algorithms for remote visualization. The selection is made based on the performance of previously compressed frames and network transfer delays. We have used a reinforcement learning technique to select the compression algorithm for each individual frame. The algorithm was tested using SGI OpenGL Vizserver, but it can be easily adapted to other remote visualization systems (Vizserver, 2004).

Transferring the full data set to the researcher's desktop for visualization purposes is most of the time impossible, due to the lack of memory and storage space of local desktop computers. Scientific Visualization research applies a client-server approach to this problem. Remote visualization can be done using different strategies. In a first scenario, the server renders the images and streams them to the client. In a second scenario, the server is doing some of the rendering calculations, such as geometry transformations or visibility determination, while the client is doing the final rendering. Another scenario is where the client is doing all the rendering computations.

Each of these scenarios has tradeoffs. For example, performing the rendering completely on the client side requires high-end desktop computers, not always available to researchers. Performing some of the rendering on the server can greatly improve the visualization, however the low-end client resources may not provide sufficient power to finish the rendering in time.
In the rest of the article we consider only the first scenario, where the server is doing all the computations including the rendering, and the client is responsible only with the display of the final image. Image streaming makes possible remote visualization using low-end desktop computers (thin clients), and can be made independently from any visualization algorithm used.

However, image streaming can require significant network bandwidth. For example, if we consider that the resolution of displayed image is 640x512 pixels with 4 bytes for the RGB colors and the alpha channel, then the size of such an image is 1.25 MBytes. The maximum theoretical frame rate that can be obtained using a 100 MBps bandwidth network, is 10 frames per second, if the full bandwidth could be used. If we consider remote visualization over a wide area network, then the achievable frame rate will be much lower.

One possible solution to this problem, when using image streaming over the network, is to use compression algorithms. The server renders the image, then compresses that image and sends it over the network. The client is then responsible for decompressing the encoded image and displaying it. Using different compression techniques for the images, the amount of data transferred over the network can be significantly reduced. How much an image can be compressed depends essentially on the image content. This means that by using different compression algorithm for the same image, different compression rates can be obtained. The problem is to find an automatic way of selecting the right compression algorithm. Different methods based on analyzing the image and using the compression algorithm that gives the best compressed size are presented in the literature. However, most of these methods can be used only for certain types of images. In this work we present an algorithm for selecting the compression methods during a remote visualization session without analyzing the content of the image.

We propose an adaptive algorithm for dynamically selecting one of the compression algorithms to be used for each individual frame. The selection is done using a reinforcement-learning algorithm, and it is based on different performance measures from the environment: past and present frame rates, compressed image sizes, compression times, estimated bandwidth. The compression method, which increases the overall frame rate, is chosen. However, from time to time, other compression methods are also used for short periods of time, in order to estimate the potential benefit of selecting them.

Reinforcement learning is a computational approach to learning whereby an agent tries to maximize the total amount of reward (the frame rate in our situation) it receives when acting with a complex, uncertain environment. As opposed to other machine learning methods, in this method the learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In many cases, the actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards.

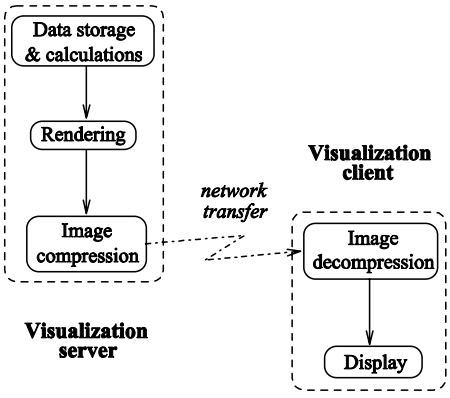


Figure.9.20 Remote visualization

Related work. Renderer implementations exploiting image compression have mostly adopted relatively simple lossless schemes, which rely on frame differencing and run-length encoding. While these techniques can deliver acceptable frame rates over local area networks, their compression ratios are highly dependent on image content, and are insufficient in slower networks. SGI OpenGL Vizserver is a product developed by Silicon Graphics, Inc., to enable remote-visualization applications. Specifically, OpenGL Vizserver is designed to provide users remote access to graphics pipelines of Onyx2 Infinite Reality machines so that they may view rendered output from visualization applications at geographically remote locations while utilizing the powerful pipeline and memory of an Onyx2 machine located at some centralized place. OpenGL Vizserver uses programmable compression modules to compress and decompress frames of the rendered scene. It comes with five standard modules (CCC, ICC, SCC, SICC, LCC) and an API that provides the capability to develop new modules with user defined functionality.

Each compression module has the capability of taking advantage of frame-to-frame coherency inherent in most visualizations, by implementing an inter-frame compression scheme where only the changing portions of each frame are compressed and sent to the clients. The CCC, ICC, SCC, and SICC compression modules implement lossy compression algorithms. These four schemes are derived from the Block Truncation Coding (BTC) algorithm that compresses a 4x4 pixel block down to two colors plus a 4x4 pixel mask. In addition to lossy compressors, there is also a lossless compression module called LCC. This preserves the original image quality while still saving bandwidth. In many cases the savings are as high as 4x without any reduction in image quality. A similar framework exists which provides remote control to Open Inventor or Cosmo3D based visualization applications (Engel et al., 2000). It allows transparent access to remote visualization capabilities and allows sharing of expensive resources. A visualization server distributes a visualization session to Java based clients by transmitting compressed images from the server frame buffer. Visualization parameters and GUI events from the clients are applied to the server application by sending CORBA requests.

Both of these two solutions require the user to explicitly select the compression algorithm to be used. In most of the situations, the user does not have any knowledge about the compression algorithm. An adaptive compression algorithm for medical images was presented in (Hludov et al. 1998). The adaptive algorithm presented is based on a classification of digital images into three classes and followed by the compression of the image by a suitable compression algorithm. The content of the image is analyses based on a validation of the relative number and absolute values of the wavelet coefficients. A comparison between the original image and the decoded image will be done by a difference criteria calculated by the wavelet coefficients of the original image and the decoded of the first and second iteration step of the wavelet transform. Compression of images was used in (Ma et al., 2000) for

visualizing time varying volume data over a wide area network. The rendering was done on a remote parallel computer and compression of the images was used for significantly reducing the cost of transferring output images from the parallel computer to the local display. They used lossy compression methods combined with lossless compression methods, which were capable of providing acceptable image quality for many applications, while retaining desirable properties such as efficient parallel compression and fast decompression. They experimented with different combinations of the JPEG, BZIP and LZO compression algorithms, and then selected the combination of JPEG and LZO as giving the best frame rates for their system.

Image compression. The use of image compression algorithms can significantly improve the amount of data transmitted over the network. All compression algorithms are based on the same principle: compressing data by removing redundancy from the original data. Any nonrandom collection data has some structure, and this structure can be exploited to achieve a smaller representation of the data, where no structure is discernible. This is the case of using lossless compression algorithms. An important feature of image compression is that in many situations it can be lossy, being acceptable to lose image features to which the human eye is not sensitive. Images can be loss compressed by removing irrelevant information even if the original image does not have any redundancy. Different image compression algorithms can be used for different types of images. Each type of image may feature redundancy, but they are redundant in different way. This is why any given compression method may not perform well for all images, and why different methods are needed to compress the different image types. The choice of the best algorithm is not trivial, most of the time requiring a certain experience with the algorithms. During a visualization session, the type of image can also change, making even more difficult to choose the appropriate algorithm. One important factor, which is important in choosing the compression algorithm, is the amount of computation needed for both compressing and decompressing the image. More efficient algorithms, capable of generating smaller compressed images are usually requiring more CPU power. This becomes very critical, especially for high-resolution images. There is a tradeoff between the amount of computation time needed to generate the compressed image and the amount of time used to transfer it over the network.

There are cases when an investment in a more efficient compression algorithm can result in a higher frame rate, especially when the remote visualization is done over low bandwidth networks. In many situations, the actual network bandwidth available, which can be used, is less that the maximum bandwidth. This is the case when the remote visualization is done without having a dedicated network connection between the visualization server and client, especially when using wide area networks for visualization over long distance. An additional problem is that this available network bandwidth can change significantly during a remote visualization session. This

can be due to other data traffic in the network.

For our study, we used four lossless compression algorithms. The choice was mainly made based on the performance of these algorithms for general image compression and the availability of optimal implementations as software libraries. The first algorithm (ZLIB) is the so-called "deflation" algorithm, which is used in the popular programs zip and gzip. This is a dictionary based compression method: it selects strings of symbols and encodes each string as a token using a dictionary. It is based on the LZ77 compression method combined with static Huffman encoding. The compression time and image sizes are pretty good, however for certain image type compression can be very poor.

The second algorithm called Lempel-Ziv-Oberhumer (LZO), an optimized dictionary based method, which is more suited for real-time compression-decompression. It offers pretty fast compression and very fast decompression, however it favors speed over compression ratio. The resulting compressed images can be very large, thus increasing the transfer time over the network. The third algorithm used (BZIP2) is based on the Burrows-Wheeler method, which is a compression method using block sorting. The input stream is read block by block and each block is encoded separately as one string. The main idea is to start with a string S of n symbols and to scramble (permute) them into another string L, which satisfies: (1) any area of L will tend to have a concentration of just a few symbols; (2) it is possible to reconstruct the original string S from L. The method is a general-purpose method, which works well on images and can achieve very high compression ratios.

The disadvantage of this algorithm is that it requires a lot of computing, both compression and decompression being slow. Since the algorithm is compressing individual blocks independently, it is possible to use a parallel version of the compression to reduce the time. The last algorithm we used is a simple Run Length Encoder (RLE). The idea behind this approach is the following: if a data item d occurs n consecutive times in the input stream, replace the n occurrences with the single pair nd. This is well suited for certain types of images, with large areas containing the same pixel value. The size of the compressed stream depends on the complexity of the image. The more detail we have, the worse the compression is. The algorithm being extremely simple, very efficient implementations could be implemented. It is also well suited for a parallel encoding.

Reinforcement learning (Sutton and Barto, 1998) is a computational approach for goal directed learning from interaction. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Reinforcement learning is different from supervised learning, the kind of learning from examples provided by a knowledgeable external supervisor. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations. In uncharted situations, where one would expect learning to be most beneficial, an agent must be able to learn from its own experience.

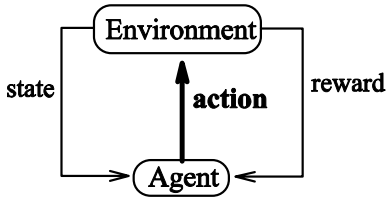


Figure.9.21 The agent-environment interaction.

In reinforcement learning, the learner and decision maker is called the *agent*. The thing it interacts with, comprising everything outside the agent, is called the *environment*. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. More specifically, the agent and environment interact at each of a sequence of discrete time steps, t. At each time step t, the agent receives some representation of the environment's state, st, and on that basis selects an action, at. One time step later, in part as a consequence of its action, the agent receives a numerical reward, rt+1, and finds itself in a new state, st+1. At each time step, the agent implements a mapping from states to probabilities of selecting each possible action.

This mapping is called the agent's policy. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run. One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to *exploit* what it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The agent must try a variety of actions and progressively favor those that appear to be best.

Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. All reinforcement-learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. It is usually assumed that the agent has to operate despite significant uncertainty about the environment it faces.

Adaptive compression. The adaptive compression algorithm we are proposing is using a reinforcement algorithm as presented in the previous section. We consider the frame rates as the rewards for each time step. An example of frame rate variation during a typical visualization session is presented in figure 3. The figure shows the current and average frame rates obtained by using the RLE (Run Length Encoding) compression algorithm for

two situations: one 100 MBps and one 10 MBps network connection of the client to the LAN. There are large variations in the current frame rate, especially when there is enough available network bandwidth (in the left and right regions of the figure). The average is done using the last ten frame rates.

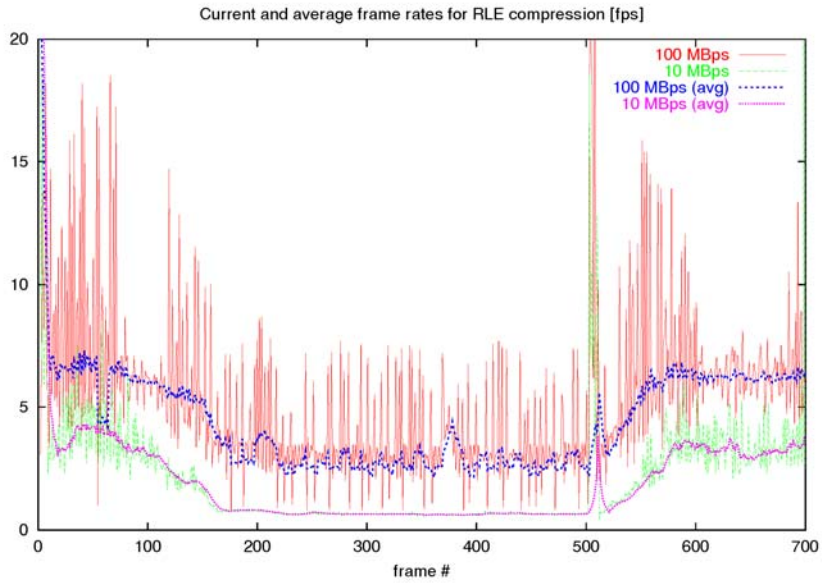


Figure.9.22 Frame rate variations and average

Due to these large variations, the algorithm is making the selection of the compression algorithm based on these average values. For each selected compression method, at least 10 frames will be rendered using this method, providing this way a better estimate of performance of the algorithm. The adaptive algorithm works as follows: it starts with one of the compression methods (LZO in our case) and it uses it for the next 10 frames to get an estimate of its performance. After that, it is trying in a similar way the other compression methods, and when all the methods are tested it is choosing the best of the algorithms. From time to time, another compression method, different from the current one, is selected randomly and evaluated. If the new method is providing a better performance, i.e. increased frame rate, then it is selected as the next compression method. We used an interval of 50 frames between trying another compression method.

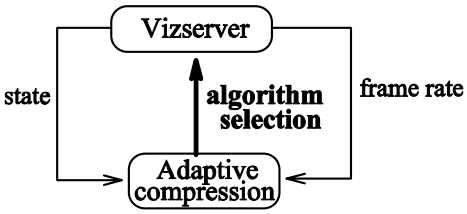


Figure.9.23 The adaptive algorithm

Experimental results. We conducted tests using an SGI Onyx2 2400 parallel computer as the remote visualization server. This computer consists of 32 R12000 RISC processors at 300 MHz, with a total memory of 16 GBytes and two Infinite Reality3 graphic pipelines. For the local visualization client we used a desktop PC with a Pentium 3 processor, running at 500 MHz, with 256 MBytes of memory. The operating system used was Linux with a 2.4.19 kernel. As a remote visualization system, we used the SGI OpenGL Vizserver software. This software allows remote rendering of the images on the SGI server, which are then compressed and sent over the network to the client for display. The SGI Vizserver offers an API for writing additional compression modules to be used.

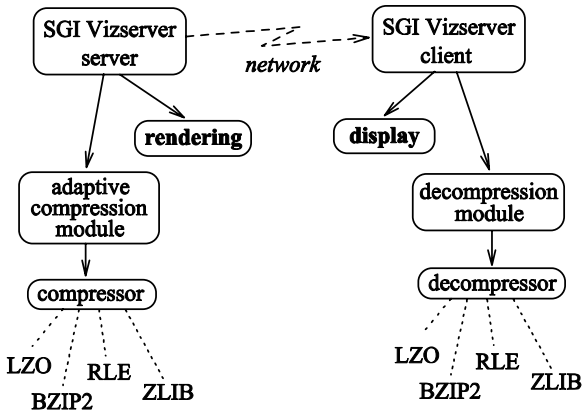


Figure.9.24 Vizserver architecture

We implemented four compression modules using the four lossless methods described above. These modules are basically wrappers for existing software libraries, which implement the compression methods. The modules give a simple interface to both the compression and decompression, which is used by the adaptive algorithm. This is implemented as a compression module for the SGI Vizserver using the development API provided with the software. The adaptive algorithm was implemented using the C++ programming language.

We chose the SGI Vizserver for several reasons. First because we had access to an SGI parallel visualization server which had it available, and second, because the API used for the compression modules is very simple, making it very easy the implementation of different compression techniques. Another reason was that the use of the Vizserver is transparent to the applications used.

There was however some problems we experienced. One of them is that the version we were using (3.1 beta) was quite unstable. We had to go through many crashes of the server software while developing and experimenting with different compression algorithms. One of the disadvantages in using SGI Vizserver is that the server hardware must be an SGI computer. However, the algorithm we implemented for the adaptive compression, together with the four compression modules are very easy to adapt to other similar remote visualization systems, due to the modular of implementation.

One possible useful parameter we did not have access to while using the SGI Vizserver framework was the effective time required for sending each of the compressed frames over the network. The only available parameters we could use were the compression time for the frames and the time between two consecutive calls for the frame compression algorithm.

In our experiment, the size of each frame was 640x512 pixels with 4 bytes per pixel (RGB plus alpha channels). We used the Volview program for visualizing a volume data set of 256x256x77 voxels of a CT scan. The Volview is part of the SGI Volumizer2 software, and uses hardware accelerated 3D texturing for volume visualization. This is a direct data visualization technique that uses textured data slices, which are combined, is a specific order using a blending operator. This technique takes advantage of graphics hardware and resources by using OpenGL 3D-texture rendering, allowing applications to obtain high interactive performances.

The experiments were conducted using two different network connections between the client and the 100 MBps LAN containing the server. In the first situation, we connected the client using a 100 MBps network card to the LAN. In the second situation, we used a 10 MBps network card for connecting the client. Using a modified version of the Volview program, we recorded the translation and rotation vectors of the volume data for each frame generated during a typical interactive visualization session. We then played back the same session using the four different compression methods and then the adaptive algorithm. Frame rate averages for all five situations are presented in Figure.9.25 and Figure.9.26, for the two network connection situations.

In both situations, the adaptive algorithm is searching for the best algorithm in the beginning, thus giving low frame rates. However, when it finds the best algorithm, it keeps it for the rest of the visualization session.

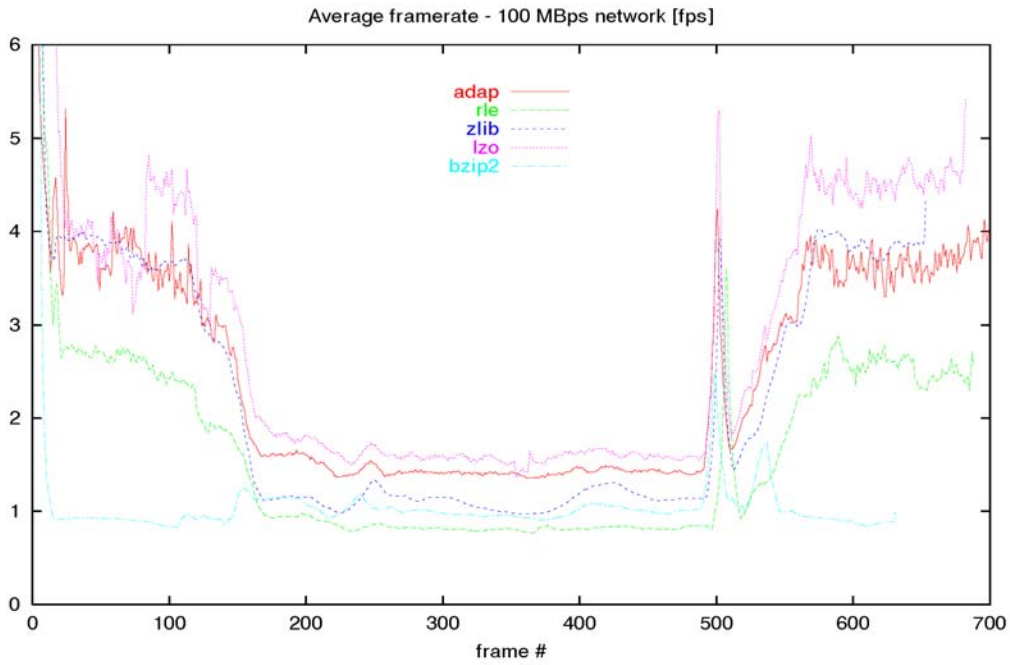


Figure.9.25 Average frame rate - 100 MBps network

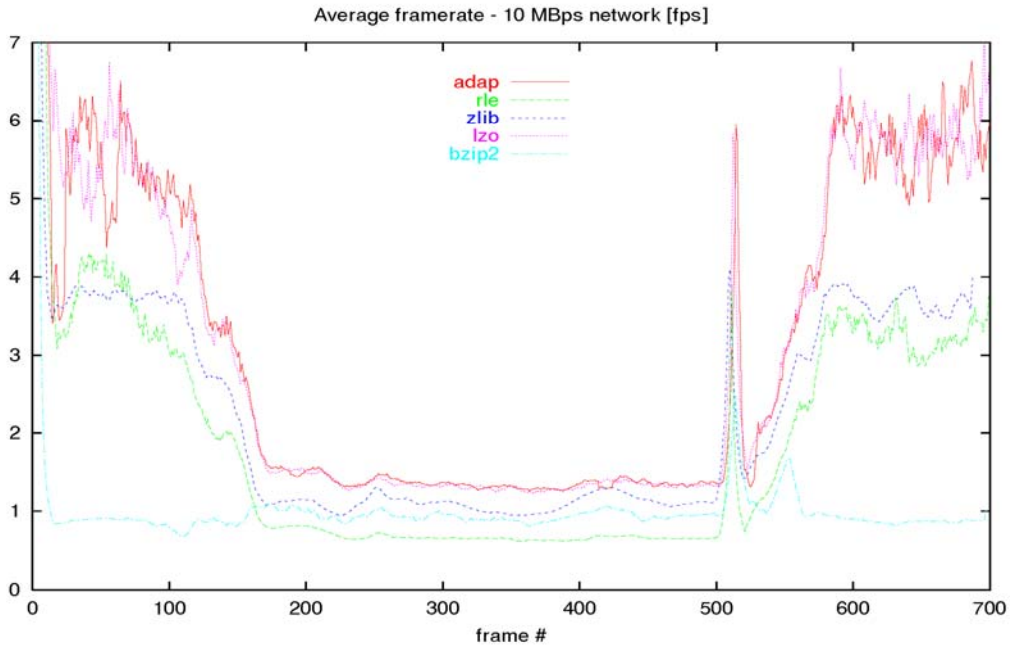


Figure.9.26 Average frame rate - 100 MBps network.

Conclusions. As the amount and size of scientific data continues to increase, the demand for high-resolution imaging will also increase. Remote visualization is one solution for making accessible remote data sets to users with low capability desktop machines. Use of image compression techniques permits remote visualization of larger resolution images or over lower bandwidth networks. There are different compression methods for different kind of images. Some of the methods give very good compression rates but only for a certain types of images, while for other types of images the compression is poor. The selection of the best compression algorithm is still a matter of experience. One other problem is that in most cases a better compression method also requires much more computational power. There is a tradeoff between the size of the compressed image and the amount of computation used in order to obtain the optimal frame rate using remote visualization.

In this experiment we presented an adaptive algorithm based on reinforcement learning for choosing one of the available compression methods in order to maximize the frame rate. Our experiments show that such an algorithm can work in a dynamic and uncertain environment, consisting of a visualization server, a visualization client, and a network for transferring the compressed images between the server and the client. One of the problems we experience with the current algorithm is that, in certain situations, one of the compression methods, which are evaluated by the adaptive algorithm, is giving really poor frame rates. This affects the interactive responsiveness of the application. One possible improvement of the algorithm would be to use a different selection algorithm for evaluating the next possible method, by making actions, which give small rewards to be less likely to occur. In this way, compression methods, which give poor frame rates, will be less probable to be selected in the future.

The modules for the compression methods and the adaptive algorithm are available for download, both as source code and binary at the following web site: http://www.idi.ntnu.no/~zoran/vizserver.

10 Conclusions and Future Work

Continued exponential technology improvements, new collaborative modalities enabled by the quasi-ubiquitous Internet, and the demands of increasingly complex problems have, over recent decades, fuelled a revolution in the practice of science and engineering. Today's science is as much based on large-scale numerical simulation, data analysis, and collaboration as it is on the efforts of individual experimentalists and theorists. Licklider's vision of man-machine symbiosis and a global communication network came from the scientific community. Today, this community still leads the way, as early attempts in Grid computing evolve to the more sophisticated and ubiquitous *virtual organization* concept, in which Grid middleware enables "coordinated resource sharing and problem solving in dynamic, multi-institutional organizations". The scientific community recognizes that following a decade of pioneering work in computational science, data technologies, supercomputing, and networking linked with Grid technologies, "computational and data management infrastructure has become a global phenomenon that is poised to evolve as a key enabler for science and society" (Foster and Kesselman, 2004).

The new modes of inquiry outlined here constitute an ambitious vision for the future of science and engineering. The realization of this vision will require long-term investments of financial resources by governments and of intellectual resources by those who must build and apply the necessary global information infrastructure. We should not underestimate the difficulty of the technical challenges that must be overcome before we can fully realize the vision of a robust middleware infrastructure capable of supporting true virtual organizations. We hope that we have emphasized just how critically important the realization of this goal is for the future of science and engineering.

With the rapid and simultaneous advances in software and computer technology, especially commodity computing, supercomputing and grid computing, every scientist and engineer will have on his or her desk an advanced simulation kit of tools that will make analysis, product development, and design more optimal and cost effective. Through the availability of increasingly powerful computers with increasing amounts of internal and external memory, it is possible to investigate incredibly complex dynamics by means of ever more realistic simulations. However, this brings with it vast amounts of data. To analyze these data it is imperative to have software tools, which can visualize these multi-dimensional data sets. Comparing this with experiment and theory it becomes clear that visualization of scientific data is useful yet difficult. For complicated, time-dependent simulations, the running of the simulation may involve the calculation of many time steps, which requires a substantial amount of CPU time, and memory resources are still limited, one cannot save the results of every time step. Hence, it will be necessary to visualize and store the results selectively in real time so that we do

not have to recompute the dynamics if we want to see the same scene again. Real time means that the selected time step will be visualized as soon as it has been calculated. Scientific breakthroughs depend on insight. In our collective experience, better visualization of a problem leads to a better understanding of the underlying science, and often to an appreciation of something profoundly new and unexpected. Advanced capabilities for visualization may prove to be as critical as the existence of the supercomputers themselves for scientists and engineers, and also for specialists in other domains. Better visualization tools would enhance human productivity and improve efficiency in several areas of science, industry, business, medicine and government. The most exciting potential of widespread availability of visualization is the insight gained and the mistakes caught by spotting visual anomalies while computing. Visualization will put the scientist into the computing loop and change the way the science is done (McCormick, 1988)

Visualization as a human activity precedes computing by hundreds of years, possibly thousands if we include cave paintings as examples of Human's attempts to convey mental imagery to his fellows. Visualization specifically in the service of science has a rather shorter but distinguished history of its own, with graphs and models produced by hand all having been used to explain observations, make predictions, and understand theories. The current era of visualization, however, is different in its pace and spread, and both can be attributed to the modern invention of the computer. Today, we are bombarded with visual imagery, no news report is considered complete without flying in graphs of statistics: the weather report can be seen to animate rain-drop by rain-drop, our banks send as plots of our incomings, and outgoings in an attempt to persuade us to manage our finances more responsibly (Wright, 2007).

Moreover, everyone can now produce their own computer graphics, with easy-to-use software integrated into word-processors that makes charts and plots an obligatory element of any report or proposal. More specialist packages in turn offer complex techniques for higher dimensional data. These used to be the domain of experts, but without the expert on hand to advise on their usage we run the risk of using the computer to make clever rubbish. Visualization has thus become ubiquitous. As a tool it is powerful but not infallible. Scientists are becoming familiar with desktop programs capable of presenting interactive models molecules and microbiological. The field of bioinformatics and the field of cheminformatics make a heavy use of these visualization engines for interpreting lab data and for training purposes. Medical imaging is a huge application domain for Scientific Visualization with an emphasis on enhancing imaging results graphically, e.g. using pseudo-coloring or overlaying of plots. Real-time visualization can serve to simultaneously image analysis results within or beside an analyzed (e.g. segmented) scan. Data visualization techniques are now commonly used to provide business intelligence. Performance metrics and key performance indicators are displayed on an interactive digital dashboard. Business

executives use these software applications to monitor the status of business results and activities.

All users of Scientific Computing and Visualization have an interest in better hardware, software and integrated systems, and much of what has being developed was shared by a number of scientific and engineering disciplines up to a point, but with very large costs that were accessible only to large research facilities (e.g. SGI visualization servers and large PC clusters). Then the gaming industry has made a breakthrough under the pressure of the gamers, who did require more and more graphical power, by developing very high performance graphics cards, at very low costs (commodity hardware). The visualization community has shifted to using these low-priced resources for their visualization tasks and progressively more PC-based visualization results have been obtained. However, gaming graphics hardware is not well suited for Scientific Visualization, leading to a fundamental rethinking of how high-end systems are built as designers attempt to apply to large scale (interactive) rendering the clustered computing techniques that have revolutionized HPC.

The remarkable performance figures of the major volunteer computing projects, such as SETI@home, self-credited with more than 65 TFlops, as of September 2005, clearly demonstrate the usefulness of harvesting cycles over the internet. The attractiveness of exploiting desktop grid systems is further reinforced by the fact that costs are highly distributed: every volunteer supports his or her resources (hardware, power costs and internet connections) while the benefited entity provides management infrastructures, namely network bandwidth, servers and management services, receiving in exchange a massive and otherwise unaffordable computing power. Fortunately, the usefulness of desktop grid computing is not limited to major high throughput public computing projects. Many institutions, ranging from academics to enterprises, hold vast number of desktop machines and could benefit from exploiting the idle cycles of their local machines.

Also, the availability of several desktop grid platforms have smoothened the setup, management and exploitation of desktop grid systems. Indeed, the potential gains of harvesting idle resources have fostered the development of desktop grid middleware. Currently, several platforms exist ranging from academic projects such as BOINC, XtremWeb, MiG, and Alchemi, to commercial solutions like Unicore, United Devices and OfficeGrid. This plethora of desktop grid and volunteer computing platforms has contributed to the explosion of new desktop grids and related projects, not only over the internet but also at an institutional level, like in the case of a university campus. The typical and most appropriate application for desktop grid is comprised of independent tasks (with no communication between tasks) with a high computation to communication ratio. The execution of the application is orchestrated by a central scheduler node, which distributes the tasks amongst the worker nodes and awaits workers' results. It is important to note that an application only finishes when all tasks have been completed.

The main difference in the usage of institutional desktop grids relatively to public ones lies in the dimension of the application that can be tackled. In fact, while public projects usually embrace massive applications made up of an enormous number of tasks, institutional desktop grids (much more limited in resources) are better matched for small size applications. So, whereas in public volunteer projects importance is on the number of tasks carried out per time unit (*throughput*), users of institutional desktop grids are normally more interested in a fast execution of their applications, seeking fast *turnaround time*.

Because of the huge number of PCs in the world, desktop grid and volunteer computing can (and do) supply more computing power to science than does any other type of computing. This computing power enables scientific research that could not be done otherwise. This advantage will increase over time, because the laws of economics dictate that consumer electronics (PCs and game consoles) will advance faster than more specialized products, and that there will simply be more of them. Volunteer computing power cannot be bought; it must be earned. A research project that has limited funding but large public appeal (such as SETI@home) can get huge computing power. In contrast, traditional supercomputers are extremely expensive, and are available only for applications that can afford them (for example, nuclear weapon design and espionage). Desktop grid and volunteer computing encourage public interest in science, and provides the public with voice in determining the directions of scientific research.

Desktop grid and volunteer computing are not to evolve outside the Grid, but connected intimately with it, inside it. Though there are some notable differences. First, within the Grid, each organization can act as either producer or consumer of resources (hence the analogy with the electrical power grid, in which electric companies can buy and sell power to/from other companies, according to fluctuating demand). Second, the organizations are mutually accountable. If one organization misbehaves, the others can respond by suing them or refusing to share resources with them. This is different from volunteer computing or desktop grid computing in some sort of institutions, like universities, where is practically impossible to track down each user of a resource at some point in time. On the other hand, desktop grid computing, which uses desktop PCs within a more formal organization, is superficially similar to volunteer computing, but because it has accountability and lacks anonymity, it is significantly different.

Internet standards made possible the Web, which enabled the near-global access to and sharing of content. Open Grid protocols hold the promise of fostering unprecedented integration of technologies, applications, files, and just about any other IT resource, enabling global sharing of these resources beyond what has been possible with the Web. The same protocols will also virtualize those resources, shielding users from their complexity and allowing them to focus on what they wish to do, rather than how the technology can get it done. Likewise, they will permit management tools to range over that vast

heterogeneous infrastructure, rendering it tractable and delivering a quality of service consistent with mass adoption. Finally, having standardized the infrastructure, open Grid protocols (like OGSA) will permit the delivery of computing services when and where needed, on-demand. In enabling all these capabilities, Grid computing is establishing the necessary conditions for IT to approach mass adoption, or what can be called a post-technology era (Foster and Kesselman, 2004). Nevertheless, Desktop Grid computing is still under heavy conceptualization, research and development. There are still many aspects to clarify and solve: security issues, scheduling, volatile environment, sabotage-tolerance, integration with Grid, decentralization etc.

The core idea of the work presented in this thesis has been to provide a desktop grid computing framework and to prove its viability by testing it in some Scientific Computing and Visualization experiments. We presented here QADPZ, an open source system for desktop grid computing, which enables users from a local network or even Internet to share their resources. It is a multi-platform, heterogeneous system, where different computing resources from inside an organization can be used. It can also be used for volunteer computing, where the communication infrastructure is the Internet. QADPZ supports the following native operating systems: Linux, Windows, MacOS and Unix variants, as opposed to some other similar systems that usually are limited to only one (Unix or Windows). Consequently, that kind of limitation restricts very much the usability of desktop grid computing in real life situations.

QADPZ provides a flexible object-oriented software framework that makes it easy for programmers to develop various applications, and for researchers to address issues such as adaptive parallelism, fault-tolerance, and scalability. The system supports also the execution of legacy applications, which for different reasons could not be rewritten, and that makes it also suitable for other domains as business. It also supports either low-level programming languages as C and C++ or high-level language applications, like for example Lisp, Python, and Java, providing the necessary mechanisms to use such applications in a computation. Therefore users with various backgrounds can benefit from using QADPZ. The flexible, object oriented structure and the modularity of the system allows improvements and further extensions to other programming languages to be made easily.

We have developed a general-purpose runtime and an API to support new kind of high performance computing applications, and therefore to benefit from the advantages offered by desktop grid computing. We show how distributed computing grid extends beyond the master-worker paradigm, typical for such systems, and provide QADPZ with an extended API which supports in addition lightweight tasks creation and parallel computing, using the message passing paradigm (MPI). The API directly supports the C/C++ programming language. QADPZ supports parallel programs running on the desktop grid, by providing and API in the C/C++ language, which implements a subset of the MPI standard. This extends the range of applications that can be

used in the system to already existing MPI based applications, like for example parallel numerical solvers, from computational science, or parallel visualization algorithms. Another restriction of existing systems, especially middleware based, is that each resource provider needs to install a runtime module with administrator privileges. This poses some issues regarding data integrity and accessibility on providers' computers. The QADPZ system tries to prevail this by allowing the middleware module to run as a non-privileged user, even with restricted access, to the local system.

QADPZ provides for low-level optimizations, such as on-the-fly compression and encryption for communication. The user can pick out from different algorithms, depending on the application, improving both the communication overhead imposed by large data transfers and keeping privacy of the data. The system goes further, by providing an experimental, adaptive compression algorithm, which can transparently choose different algorithms to improve the application. QADPZ support two different protocols (UDP and TCP/IP) in order to improve the efficiency of communication.

Free availability of the source code allows its flexible installations and modifications based on the individual needs of research projects and institutions. In addition to being a very powerful tool for computationally-intensive research, the open-sourceness makes QADPZ a flexible educational platform for numerous small-size student projects in the areas of operating systems, distributed systems, mobile agents, parallel algorithms, and others. More, free software is a natural choice for modern research, as well, because it encourages effectively integration, cooperation and boosting of new ideas. We offered the QADPZ system as open source from the beginning, at a time when very few such solution were free, with all the positive implications of this for research and other computationally intensive applications.

Beside the extended master-worker conceptual model (which makes contributions in several directions - pull vs. push work-units, pipelining of work-units, more work-units sent at a time, adaptive number of workers, adaptive time-out interval for work-units, multithreading, resource estimation and monitoring, scheduling) and the QADPZ desktop grid system, this thesis make contributions in form of a hierarchical taxonomy of the main existing desktop grids, and of an adaptive compression algorithm for remote visualization. We have also been trying to demonstrate that the use of desktop grid computing should not be limited to only master-worker type of application, but can be used also for more fine-grained parallel applications, in the field of Scientific Computing and Visualization, by performing some experiments in those domains. The system is currently used for research tasks in the areas of large-scale Scientific Visualization, evolutionary computation, simulation of complex neural network models, and other computationally intensive applications. It is worth to mention that to the present, the QADPZ has over a thousand downloads, from users who use it for their tasks, as it can be seen in the appendix.

The work on QADPZ system and its conceptual model has been done in collaboration with my colleague Pavel Petrovič from NTNU-IDI, who has had major contributions mostly on the slave side and on job descriptions in XML (XML parser included), and on a number of specific requirements, coming from his research interests. Some of the work he has done on the slave side has been later rewritten for better modeling or efficiency purposes. The job description and the parser have remained the same as he has developed. The final master worker model that has been implemented in QADPZ meets his requirements.

Some of the results of this thesis have already been published (they are listed in the references) and some are in course of publication. Thus, contributions that are already published concern: the QADPZ system (Constantinescu and Petrovic, 2002) and (Constantinescu *et al.*, 2002), QADPZ proven to be useful in Scientific Computing – example of using it to solve the Navier Stokes equation for fluid dynamics (Constantinescu, 2003), QADPZ as an autonomic distributed computing system (Constantinescu, 2003), and the hierarchical taxonomy of desktop grid systems built from users' perspective (Constantinescu and Vladoiu, 2008). The paper on QADPZ's autonomicity has been highly cited since it has been published and considered as pioneering this approach in desktop grids, as it can be seen in the appendix. The results on QADPZ, as a viable desktop grid/volunteer computing open solution, which can also use parallel computing techniques using the MPI layer - this is a novel approach in desktop grid, on the improved master worker model, on the adaptive compression algorithm for remote visualization, on master virtualization, on QADPZ testing in some experimental scientific visualizations, and on QADPZ development journey are in course of publication.

It is worth to reconsider here briefly the Desktop Grid requirements (which have been presented in section 4.7.2.2) that were not yet synthesized at QADPZ's development time, and to try to match them against the QADPZ requirements that have been implemented in the system. With respect to these requirements we may say that QADPZ was expected to manage available resources efficiently (*efficiency*), to enforce program security (*security*), to be easy maintainable, flexible, and extensible (*scalable, manageable*), to provide for job management (*open/easy to integrate applications*), to offer proper resource and job management (*manageable, unobtrusive*), to handle multiple-projects (*multiple-project participation*) and to support parallel programming (*communicative*). What QADPZ misses is full robustness and data security that are in our future work plans. The robustness is accomplished simplistically in QADPZ: if one job fails due to one or more of task failures, for various reasons, the job is started over. But QADPZ had some other rewarding requirements such as providing performance measurements, on-line/off-line support for batch and interactive applications, personalization and simplicity. Moreover, QADPZ has pioneered autonomic features requirements for desktop grids.

Further on we present some future work ideas that aim to improve both the conceptual model and the QADPZ system. First, at this time the system

does not support job checkpointing and does not handle restart of master computer. Adding these features has high priority. In the current version of the system, each job needs a different client process, although we are working on extending the client functionality to allow single instance of client to optionally connect to multiple masters and handle multiple jobs. Future development of the system will include improved support for user data security. Computation results data can be encrypted and/or signed so that the user of the system can be sure the received data is correct. We are considering allowing optional data integrity in the future versions of QADPZ. This is especially useful if the system is used in an open environment, for example over the Internet. For faster performance, slave libraries will be cached at slave computers – in the current version, they are downloaded before each task is started. Slave computers will provide a flexible data storage available to other computers in QADPZ. The scheduling algorithm of the master needs improvements. We plan to support more hardware platforms and operating systems.

Our current implementation does not support collective communication, only the MPI_COMM_WORLD communicator. However, a complete library of the collective communication routines can be written entirely using the point-to-point communication functions and a few auxiliary functions. QADPZ's implementation is limited to a small subset of the MPI. It contains only the most used functions, and is intended only for testing purposes and evaluation of the parallel communication. More complete implementation based on existing libraries is possible, but it was outside the scope of this thesis. The MPI functions implemented provide sufficient features for our parallel experiments. The current user interface to the system is based on C++. Possible extensions of the system would be different interfaces for other languages, e.g. Java, Perl, Tcl or Python. This can easily be done, since the message exchanges between different components of the system are based on an open XML specification.

The current implementation of the system is made considering only one central master node. This can be an inconvenience in certain situations, where computers located in different networks are used together. The master node can also be subject to failures, software or hardware. A more decentralized approach is needed in this case. However, our high-level communication protocol between the entities, especially between the client and master, allows a master to act as a client to another master, thus making possible to create a distributed master, consisting of independent master nodes, which communicate with each other i.e. some sort of *virtual master*. Ideas from peer-to-peer computing will be used for implementing such a decentralized approach.

Future desktop grid infrastructure must be decentralized, robust, highly available, and scalable, while efficiently mapping application instances to available resources in the system. However, current desktop grid computing platforms are typically based on a client-server architecture, which has inherent shortcomings with respect to robustness, reliability and scalability. Fortunately, these problems can be addressed through the capabilities promised by new

techniques and approaches in P2P systems. By employing P2P services, our system could allow users to submit jobs to be run in the system and to run jobs submitted by other users on any resources available in the system, essentially allowing a group of users to form an ad-hoc set of shared resources.

Another future work idea is to add a set of transparent profiling tools for evaluating the performance of the different components. This is an important issue, especially when running parallel applications. Dynamic balancing of the workload can be used. We plan to introduce more autonomic features in the system. Other possible extensions of the system are currently considered, for example interconnection with a grid computing environment.

We invite the interested developers in the open-source community to join our development team and we appreciate any kind of feedback.

Viewed from a historical perspective, Information Technology has clearly been in the developmental stage of its evolution. IT began with mainframes and supercomputers sheltered in the "glass house." Expensive and complex, these early systems yielded results only to highly trained specialists steeped in the mysteries of programming. With the advent of personal computers and local area networks, millions of people began to use the technology, and since the emergence of network computing and the Internet, hundreds of millions more have come to use it. Information Technology, like electricity and automobiles before it, is last approaching its own post-technology phase - a time when the application will be dominant and the technology will gradually sink into the background of our lives and be integrated into society. The signs are all there. One of the major heralds of this new phase is the increasing commoditization of information technologies. Microprocessors, storage, DRAMs, bandwidth, and all sorts of other information technologies, year in and year out, are improving by 50, 60, even 70%, becoming much less expensive, with much more power packed into a smaller unit. Obviously, powerful technologies that are less expensive and smaller are more easily hidden in the environment. Commodity IT, therefore, is potentially ubiquitous, like the little electric motors found throughout our homes and in our cars.

Another indication that IT is headed toward mass adoption is the never-ending and incredible increase in the power of systems. That same inexpensive commodity technology is being aggregated into larger and more powerful computers. Soon blades-servers on inch-thick cards will let us cluster systems by the thousands. In the not-so-distant future, we will see systems with tens of thousands, eventually even hundreds of thousands, of blades or similar small components, all collaborating, all solving unimaginably sophisticated problems, all supporting hundreds of millions of users. In sum, technologies are becoming commoditized to such a degree that we can afford to have billions of them in the environment, while systems are growing so incredibly potent that we can build a commensurately powerful and connected infrastructure to support them.

Properly handled, this rich layer promises a brilliant future. What will it look like? For one, it will be thoroughly integrated: systems, business processes, organizations, people - everything required for a smoothly functioning whole - will be in close dynamic communication. In addition, the infrastructure will reach much advanced levels of efficiency, with all the enterprise's resources fully employed rather than the current spotty, uneven, piecemeal application of these costly assets. The quality of services will be vastly improved, as the infrastructure becomes more autonomous and has capabilities as self-configuring, sell-optimizing, self-healing, and self-protecting. Finally, much greater degrees of flexibility will emerge, leaving people free to make technology choices based on their needs rather than on some architectural issues.

However, the planets are aligning: open standards are becoming more prevalent. Grids are evolving in the research community and making their way swiftly into many other areas of life. It is only a matter of time before information technology achieves the kind of productive anonymity that electricity did when standards made it ubiquitous and routine. Arthur Clarke may have been right. As we become capable of doing more and more with our advanced technologies and as we hide those technologies and their complexities from users, the result will indeed seem like magic. Making that magic convincing is one of the most complex and exciting challenges facing our community, as we move IT into its post-technology phase.

Selective Bibliography

ALCHEMI (2004) Alchemi Plug&Play Desktop Grid - http://www.alchemi.net/.

ANDERSON, D. P. et al., 2002, SETI@home: an experiment in public-resource computing. Commun. ACM 45, 11 (Nov. 2002), pp. 56-61.

ANSHUS O., ELSTER A., VINTER B., 2003, Cluster Computing as a Teaching Tool, in *Proc. of Parallel Computing 2003 (ParCo 2003)*, Dresden, Germany

BAKER, C. W., 2000, *Scientific Visualization: the new eyes of science*, Brookfield, Conn., Millbrook Press.

BAYANIHAN (2006) Bayanihan Computing Group - http://bayanihancomputing.net/.

BEDERSON, B., SHNEIDERMAN, B., 2003, *The craft of information visualization: readings and reflections*, Boston, Morgan Kaufmann.

BERMAN, F., FOX, G., HEY, A. J. G., 2003, *Grid computing: making the global infrastructure a reality*, New York, J. Wiley.

BOINC (2006) BOINC - open source software for volunteer computing and grid computing- http://boinc.berkeley.edu/.

BONNEAU, G.-P., ERTL, T., NIELSON, G. M., 2006, *Scientific Visualization: the visual extraction of knowledge from data*, Berlin, Springer-Verlag.

BROWNE, J. C. et al., 2004, General parallel computations on desktop grid and P2P systems. *Proc. of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems, Houston, Texas, ACM Int. Conference Proceeding Series*, Vol. 81.

CASSENS, J., CONSTANTINESCU , Z., 2003, Free Software: An Adequate Form of Software for Research and Education in Informatics? *LinuxTag 2003 Conf.*, Karlsruhe, Germany.

CASSENS, J., CONSTANTINESCU, Z., 2003, It's Magic: SourceMage GNU/Linux as a High Performance Cluster OS, *LinuxTag 2003 Conf.*, Karlsruhe, Germany.

CHINAGRID (2007) ChinaGRID - http://www.chinagrid.edu.cn/.

CHOI, S. et al., C., 2007 Characterizing and Classifying Desktop Grid, *7th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007),* pp. 743-748.

COMB-E-CHEM (2005) http://www.it-innovation.soton.ac.uk/projects/comb-e-chem/.

CONSTANTINESCU, Z., 2000, Levels of Detail: An Overview. *The Journal of LANA*, No 5.

CONSTANTINESCU, Z., 2003, Towards an autonomic distributed computing environment, in *Proc. of 14th Database and Expert Systems Applications Workshops*, September 2003, Prague, Czech Republic

CONSTANTINESCU, Z., HOLMEN, J., PETROVIC, P., 2003, Using Distributed Computing in Computational Fluid Dynamics, *ParCFD 2003 Conference*, Moscow, Russia.

CONSTANTINESCU, Z. & PETROVIC, P., 2002, Q2ADPZ* an open source, multi-platform system for distributed computing, *ACM Crossroads*, Vol. 9, pp. 13-20.

CONSTANTINESCU, Z., PETROVIC, P., PEDERSEN, A., 2002, Q2ADPZ* An Open system for distributed computing, *NordU2002 Conference*, Helsinki, Finland.

CONSTANTINESCU Z., VLADOIU M., 2008, Desktop Grid Experiments for Computational Science and Engineering, *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing*, Trondheim, Norway

CONSTANTINESCU Z., VLADOIU M., 2008, A Taxonomy for Desktop Grids from Users' Perspective, in *Proc. of Int. Conf. on Parallel and Distributed Computing, World Congress on Engineering (WCE 2008),* London, UK

CONSTANTINESCU Z., VLADOIU M., 2008, An Extended Master Worker Model for a Desktop Grid Computing Platform (QADPZ), submitted to *3rd Int. Conf. on Software and Data Technologies (ICSOFT 2008)*, Porto, Portugal

CONSTANTINESCU Z., VLADOIU M., 2008, The Development Journey of QADPZ - a Desktop Grid Computing Platform, submitted to *3rd Int. Conf. on Software and Data Technologies (ICSOFT 2008)*, Porto, Portugal

CUMMINGS, M. P. (2007) Grid Computing - http://serine.umiacs.umd.edu/research/grid.php.

cURL (2007) cURL groks URLs - http://curl.haxx.se/.

DAVID, P. A., et al., 2002, SETI@home: an experiment in public-resource computing, *Communications of ACM*, 45, pp. 56-61.

DISTRIBUTED.NET (2004) distributed.net project - http://distributed.net/.

DISTRIBUTEDCOMPUTING.INFO (2007) - http://distributedcomputing.info.

DOMINGUES, P., MARQUES, P., SILVA, L., 2005, Resource usage of Windows computer laboratories, in MARQUES, P., Ed., *Int. Conf. on Parallel Processing Workshops (ICPP 2005),* Leiria, Portugal

DOMINGUES, P., SILVA, J. G., SILVA, L., 2006, Sharing checkpoints to improve turnaround time in desktop grid computing, in SILVA, J. G., Ed., *20th Int. Conf. on Advanced Information Networking and Applications (AINA 2006)*, Viena, Austria.

DOMINGUES, P., SOUSA, B., SILVA, L. M., 2007, Sabotage-tolerance and trust management in desktop grid computing, *Future Generation Computing Systems,* 23, pp. 904-912.

ELSTER A.C., 2002, High-Performance Computing: Past, Present and Future, *PARA 2002*, Espoo, Finland.

EGEE (2008) Enabling Grids for E-SciencE http://www.eu-egee.org/

eSCIENCE (2007) eScience - http://www.rcuk.ac.uk/escience/default.htm.

FORSSELL, L. K., COHEN, S. D., 1995, Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows, *Transactions on Visualization and Computer Graphics*, 1, pp. 133-141.

FOSTER, I., KESSELMAN, C., 1999, *The grid: blueprint for a new computing infrastructure*, San Francisco, Morgan Kaufmann Publishers.

FOSTER, I., KESSELMAN, C., 2004, *The grid: blueprint for a new computing infrastructure*, Boston, Morgan Kaufmann Publishers.

FP7 (2007) Framework Programme 7 - http://cordis.europa.eu/fp7/home_en.html.

FRIEDHOFF, R. M., PEERCY, M. S., 2000, *Visual Comp*., Scientific American Library, NY.

FUNKHOUSER, T. A., SÉQUIN, C. H., 1993, Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments, *Proc. SIGGRAPH '93*, Anaheim, California, USA

GARG, V. K., 1996, *Principles of distributed systems*, Boston, Kluwer Academic Pub.

GARG,V. K., 2002, *Elements of distributed computing*, New York, Wiley-Interscience.

gLite (2008), LightWeight Middleware for Grid Computing - http://glite.web.cern.ch/glite/

GLOBUS (2007) The Globus Alliance - http://www.globus.org/.

GOLUB, G. H., 1997, *Proceedings of the Workshop on Scientific Computing,* Hong Kong, March, 1997, New York, Springer.

GOLUB, G. H., ORTEGA, J. M., 1993, *Scientific Computing: an introduction with parallel computing*, Boston, Academic Press.

GROPP, W. et al., 1996, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, 22, pp. 789-828.

GROPP, W. D., LUSK, E., 1996, *User's Guide for MPICH, a Portable Implementation of MPI*, Mathematics and Computer Science Division, Argonne National Laboratory.

HAMSCHER, V. et al., 2000, Evaluation of job-scheduling strategies for grid computing, in *Proc. of the 1st IEEE/ACM Int. Workshop on Grid Computing (Grid 2000*), LNCS 1971, Springer-Verlag, Bangalore, India, pp. 191-202.

HANSEN, C. D., JOHNSON, C. R., 2005, *The visualization handbook*, Boston, Elsevier-Butterworth Heinemann.

HEAP, D. G., 2003, *Taurus - A Taxonomy of Actual Utilization of Real UNIX and Windows Servers,* IBM White Paper.

HEATH, M. T., 1997, *Scientific Computing: an introductory survey*, NY, McGraw-Hill.

HEATH, M.T., 2002, *Scientific Computing: an introductory survey*, Boston, McGraw-Hill.

KRAUTER K., BUYYA R., MAHESWARAN M., 2002, A taxonomy and survey of grid resource management systems for distributed computing, in *Software – Practice and Experience*, 2002, 32, pp. 135–164

IBM, 2001, *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IMB Manifesto on Autonomic Computing.

JOHN, P. M., JAMES, J. K., DAVID, A. P., 2001, WebCom: A Web Based Volunteer Computer, *Journal of Supercomputing*, 18, pp. 47-61.

JPGRID (2007) JPGRID - http://www.jpgrid.org/.

JUHASZ Z., KACSUK P., KRANZLMULLER D., 2004, *Distributed and Parallel Systems: Cluster and Grid Computing*, New York, Springer

JUOZAPAVICIUS A., BLAKE R.E., 1999, Indices and Data Structures in Information Systems, *Informatica, Lithuanian Academy of Science* 10(1), pp. 71-88

JUOZAPAVICIUS A., MAZEIKA D., The Lithuanian National Grid Initiative: LitGrid - http://www.ebaltics.com/01004600

KARNIADAKIS, G., KIRBY, R. M., 2003, *Parallel Scientific Computing in C++ and MPI: a seamless approach to parallel algorithms and their implementation*, New York, Cambridge University Press.

KAUFMAN, A., NIELSON, G. M., *Visualization '92: proceedings*, October, 1992, Boston, Massachusetts, IEEE Computer Society Press.

KEPHART, J. O., CHESS, D. M., 2003, The vision of autonomic comp., *Computer*, 36, 41-50.

KONDO, D. et al., 2004 Characterizing and evaluating desktop grids - An empirical study, in *Proc. of 18th IEEE/ACM International Parallel & Distributed Processing Symposium (IPDPS2004),* SanteFe, New Mexico.

LANGTANGEN, H. P., 2003, *Computational partial differential equations: numerical methods and Diffpack programming*, Berlin, New York, Springer.

LANGTANGEN, H. P., BRUASET, A. M. & QUAK, E., 2000, *Advances in software tools for Scientific Computing*, Berlin, New York, Springer.

LANGTANGEN, H. P., TVEITO, A., 2003, *Advanced topics in computational partial differential equations: numerical methods and Diffpack programming*, Berlin, Springer.

LCG (2008) Worldwide LHC Computing Grid - http://lcg.web.cern.ch/LCG/

LEOPOLD, C., 2001, *Parallel and distributed computing: a survey of models, paradigms, and approaches*, New York, Wiley.

LHC (2007) LHC - the Large Hadron Collider - http://lhc.web.cern.ch/lhc/.

LITZKOW, M. L., MUTKA M. W., 1988, Condor - A Hunter of Idle Workstations, *Proc. of the 8th International Conference of Distributed Computing Systems (ICDCS1988).*

MAGNENAT-THALMANN, N., THALMANN, D., 1991, *New trends in animation and visualization*, Chichester, New York, Wiley.

MAŽEIKA D., JUOZAPAVIČIUS A., Grid Computing Infrastructure, Problems and Perspectives in Lithuania, *Informacines technologijos 2006 Konferencijos pranesimu medziaga*, Kaunas, Lithuania.

MCCONNELL, J. J., 2002, *Computer Graphics Companion*, Nature Publishing Group.

MCCORMICK, B. H., 1988, Visualization in Scientific Comp., *SIGBIO Newsletter*, 10, 15-21.

MCCORMICK, B. H., DEFANTI, T. A., BROWN, M. D., 1987, Visualization in Scientific Computing - A Synopsis, *IEEE Computer Graphics and Applications*, 21, pp. 61-70.

MERZKIRCH, W., 1987, *Flow visualization*, Orlando, Academic Press.

MIG (2008) Minimum Intrusion Grid - http://mig-1.imada.sdu.dk/MiG/index.html

MORRISON, J. P., et al., 2004, Webcom-G: grid enabled metacomputing, *Neural, Parallel Scientific Computing,* 12, 3 (Sep. 2004), pp. 419-438.

MUSTAFEE, N., TAYLOR, S. J. E., 2006, Using a desktop grid to support simulation modelling, in TAYLOR, S. J. E., Ed., in *Proc. of 28th International Conference on Information Technology Interfaces (ITI 2006),* Dubrovnik, Croatia.

myGRID (2007) myGrid - http://www.mygrid.org.uk/

NAGEL, R. N. (2006), Scientific Visualization versus Information Visualization - http://www.hpc2n.umu.se/para06/papers/paper_213.pdf

NATVIG T., ELSTER A. C., Automatic and Transparent Optimization of an Application's MPI Communication, *PARA'06*, Umeå, Sweden, June 2006.

NIELSON, G. M., 1991, Visualization in Scientific and Engineering Computing, *Computer*, 21, pp. 58-66.

NIELSON, G. M., BERGERON, D., 1993, Visualization '93: proceedings, San Jose, California, IEEE Computer Society Press.

NIELSON, G. M., HAGEN, H. & MÜLLER, H., 1997, Scientific Visualization: overviews, methodologies, and techniques, Los Alamitos, California, IEEE Computer Society.

NIELSON, G. M., ROSENBLUM, L. J., *Visualization '91: proceedings*, San Diego, California, Los Alamitos, California, IEEE Computer Society Press.

NorduGrid (2008) Grid Solution for Wide Area Computing and Data Handling http://www.nordugrid.org/

Office Grid (2008), Office Grid - http://www.meshtechnologies.com

OPENSSL (2007) OpenSSL - http://www.openssl.org/.

OXFORD, 2002, Oxford English Dictionary, Oxford University Press.

PANG, A., 1995, A Syllabus for Scientific Visualization, in THOMAS, D. A., Ed., *Scientific Visualization in Mathematics and Science Teaching*, Charlottesville, Association for the Advancement of Computing in Education.

PATRIKALAKIS, N. M., 1991, *Scientific Visualization of physical phenomena*, Tokyo, New York, Springer-Verlag.

PETROVIC, P., 2007, Incremental Evolutionary Methods for Automatic Programming of Robot Controllers, *PhD Thesis*, Norwegian University of Science and Technology, Trondheim, Norway.

QADPZ (2007) QADPZ - Quite Advanced Distributed Parallel Zystem - http://qadpz.sourceforge.net.

ROSENVINGE E. R., ELSTER A. C., BANINO C., 2004, Exp. with Scheduling Strategies for Data-Parallel MPI Applications on Clusters, *PARA 2004*, Lyngby, Denmark.

RSA (2005) RSA - http://www.rsa.com/.

SARMENTA, L. F. G., 2001, Sabotage-tolerance mechanisms for volunteer computing systems, in *Proc. of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid 2001 (CCGrid 2001),* Brisbane, Australia.

SARMENTA, L.F.G., 2001, Volunteer computing, *Ph.D. thesis*, MIT, Cambridge, USA.

SAGEPUB (2008) A practical evaluation taxonomy - www.sagepub.com/upm-data/5047_Chen_Chapter_3.pdf

SETI@HOME (2008) SETI@home - http://setiathome.ssl.berkeley.edu/.

SMITS, A. J., LIM, T. T., 2000, *Flow visualization: techniques and examples*, River Edge, NJ, Imperial College Press, World Scientific Publishers.

STERLING, T., BECKER, D. J., Salmon, J., Savarese, D. F., 1999, *How to Build a Beowulf. A Guide to Implementation and Application of PC Clusters*, The MIT Press.

SUNDERAM, V. S., 1990, PVM: a framework for parallel distributed computing, *Concurrency: Practice and Experience*, 2, pp. 315-339.

SUTTON, R. S., BARTO, A. G., 1998, *Reinforcement learning: an introduction*, Cambridge, Mass., MIT Press.

SZAJDA, D., LAWSON, B., OWEN, J., 2003, Hardening functions for large scale distributed computations, in LAWSON, B., Ed., *Proceedings of Symposium on Security and Privacy*, 2003, Oakland, California, USA.

THALMANN, D., 1990, *Scientific Visualization and graphics simulation*, Chichester England, New York, Wiley.

TOP500 (2007) Top500 Supercomputer sites - http://www.top500.org/.

TUFTE, E.R., 1997, *Visual explanations images and quantities, evidence and narrative*, Cheshire, Conn., Graphics Press.

TUFTE, E. R., 2001, *The visual display of quantitative information*, Cheshire, Connecticut, Graphics Press.

UNICORE (2008) Uniform Interface to Computing Resources - http://www.unicore.eu/

UTNES, T., BRORS, B., 1993, Numerical modelling of 3-D circulation in restricted waters, *Applied Mathematics Modeling*, 17, pp. 522-535.

VAHID, G., LIONEL, C.B., YVAN, L., 2006, Traffic-aware stress testing of distributed systems based on UML models, *Proceeding of the 28th ACM International Conference on Software Engineering*, Shanghai, China.

VENUGOPAL, S., BUYYA, R., RAMAMOHANARAO, K., 2006, A taxonomy of data grids for distributed data sharing, management and processing, *ACM Computing Surveys* 38, 1 (Mar.), pp. 1-53.

VINTER B. et al., 2004, A Comparison of Three MPI Implementations, in *Proc. of Communicating Process Architectures (CPA 2004)*, Oxford, UK

VizServer, (2004), SGI OpenGL Vizserver www.sgi.com/products/software/vizserver/

WALDROP M. M., 2001, *The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal*, Viking Adult, New York

WARE, C., 2004, *Information visualization: perception for design*, San Francisco, California, Morgan Kaufman.

WEBSTER, 1998, Merriam-Webster's Collegiate Dictionary, Merriam-Webster, Inc.

WEICKERT, J., HAGEN, H., 2006, *Visualization and processing of tensor fields*, Berlin, Springer-Verlag.

WEISKOPF, D., 2006, *GPU-based interactive visualization techniques*, New York, Springer.

WRIGHT, H., 2007, *Introduction to Scientific Visualization*, New York, Springer.

XtremWeb (2005) XtremWeb - http://www.lri.fr/~fedak/XtremWeb/.

YANG, W.-J., 2001, *Handbook of flow visualization*, New York, Taylor & Francis.

YEO, C., BUYYA, R., 2006, A taxonomy of market-based resource management systems for utility-driven cluster computing, *Software: Practice and Experience* 36, 13, 1381-1419.

YU, J., BUYYA, R., 2005, A taxonomy of scientific workflow systems for grid computing, *SIGMOD Record, Sp. Issue on Scientific Workflows* 34, 3, pp. 44-49.

ZOMAYA, A. Y., 1996, *Parallel and Distributed Computing Handbook*, New York, McGraw-Hill.

Appendix 1. Feedback and reactions to QADPZ

Within this appendix, the raw feedback and reactions to the system are listed. These can be categorized into four main categories: feedback and support requests from users who use QADPZ for their research and development tasks, forum discussions, citations in papers, and working assignments, based on QADPZ features, for students from some universities. Each item is preceded by a word, which indicates in which category it falls: feedback, forum, citation or assignment.

1. Assignment:
http://diplab.snu.ac.kr/courses/2007f/dip/Homework/assignment_3.pdf
4541.662A Distributed Information Processing Handout 4 (2007 Fall)
Mobile Embedded Software Lab, CSE, SNU - Assignment 3
Add encryption and decryption steps that use the following cryptographic algorithms, to the client program that you have written as part of your Assignment 2 work.
Miscellaneous Links: QADPZ Documentation by the QADPZ Team at the Norwegian University of Science and Technology
RSA: http://qadpz.idi.ntnu.no/doxy/html/RSAcrypter_8cpp-source.html

2. Assignment:
http://www.eecg.toronto.edu/~ashvin/courses/ece1746/2003/project-suggestions.html
ECE 1746, Fall 2003, Project Suggestions
Implement a fault-tolerant large-scale distributed computation. Implement a large-scale distributed, perhaps scientific, algorithm of your choice. You can use an infrastructure such as QADPZ - Quite Advanced Distributed Parallel Zystem. Test the fault-tolerant behavior of your application, e.g., does the algorithm degrade gracefully if one node crashes. Modify your algorithm so that it is fault-tolerant in the face of node failure. One method of evaluation could be in terms of progress made by the application after node crashes, e.g., does the application make progress proportional to the number of surviving nodes in the system.

3. Feedback:
http://ibalita.msuiit.edu.ph/modules.php?name=Forums&file=viewtopic&p=11976
Since we don't have a cluster here, where students interested in distributed computing, can have access to, I am planning to build one for IIT, something similar to SETI, where IDLE CPUs of ordinary desktop PCs can be utilized [CPUs in our offices]. I'm currently experimenting with QADPZ (http://qadpz.sourceforge.net/). Pag ok na, I'll ask for the admin's approval to install the clients in the different offices.

4. Citation:
http://www.comp.leeds.ac.uk/kwb/publication_repository/2005/cgf_006.pdf
http://www-compsci.swan.ac.uk/~csmark/PDFS/visualsupercomputing.pdf
"Visual Supercomputing: Technologies, Applications and Challenges"
A noticeable amount of research effort in autonomic computing has been placed on the self-management of system infrastructure and business services. Examples of this include self-configuration in patching management [135] and Grid service composition

[136], self-optimization in power management [137], business objectives management [138], and network resource management [139], and self-healing in online service management [140] and distributed software systems [141]. Efforts have also been made to broaden the scope of autonomic computing, addressing a wide range of related research issues, such as economic models [142], physiological models [143], interaction law [144], preference specification [145], ontology [146,147], human-computer interaction [148], and so forth. Though the development of generic software environments for autonomic applications is still in its infancy, several attempts were made, which include projects such as QADPZ [149], AUTONOMIA [150] and Almaden Optimal-Grid [151]. QADPZ [149] provides an open source framework for managing heterogeneous distributed computation in a network of desktop computers using autonomic principles. In QADPZ, the system complexity is hidden in the middleware layer, facilitating self-knowledge, self-configuration, self-optimization and self-healing.

5. Citation:
http://arxiv.org/pdf/cs/0607061
On Some Peculiarities of Dynamic Switch between Component Implementations in an Autonomic Computing System
The success of an autonomic system behavior is essentially determined by ability to detect or predict overall performance that is actually the ground for management of autonomic components, in particular, for activation of an appropriate component implementation. For this, establishing of mathematical abstractions and models giving criteria governing the sequence of switches between component implementations is an important point of autonomic computing [2-5].

6. Citation:
http://www.scientificjournals.org/journals2007/articles/1198.pdf
There is no full fledged autonomic system either in the business domain or in the research domain that the author is aware of [83]. Most of the autonomic systems so far are actually prototypes or provide a limited amount of required functionality [58, 106] of an autonomic system. The most important aspect that is missing in all these systems is that the authors do not actually describe how to write programs in such systems or how to utilize such a system in a simpler fashion. They either introduce new metaphors or provide a completely new approach to autonomic computing that adds additional complexity and a steep learning curve to the programmer. The goal of this research is to make the resultant system simple to use, by making the underlying autonomic framework transparent. None of the following systems match this goal.
QADPZ [19] provides an open source framework that allows the management and use of the computational power of idle computers in the network using autonomic principles. QADPZ is implemented in C++ and uses MPI as its communication protocol, which restricts this system to a certain class of architectures. It also deploys a masterslave pattern for task distribution, which actually does not follow the autonomic system architecture and it does not take any measure to overcome a single point of failure, e.g. the master node. The clients and the slaves (which do the actual work on behalf of the client) talk to each other by the use of a shared disk space, which is certainly a performance bottleneck and requires costly synchronization.

7. Citation:
http://www.thevantagepoint.com/resources/articles/Mining%20Conference%20Proceedings%20for%20Corporate%20Technology%20KnowledgeManagement.pdf
Mining Conference Proceedings for Corporate Technology Knowledge Management
Let's explore Figure 4 further. The term "autonomic computing" appears in two factor groups. Autonomic computing occurred first in 2003 in 7 abstracts and then in 4 abstracts in 2004. In a 2003 paper, Constantinescu states "Systems which are autonomic, capable of managing themselves are required" in "Towards an Autonomic Distributed Computing System." In a 2003 paper, Sterritt et al. claim autonomic computing aims to (i) increase reliability by designing systems to be self-protecting and self-healing; and (ii) increase autonomy and performance by enabling systems to adapt to changing circumstances, using self-configuring and self-optimizing mechanisms. This field, autonomic computing, appears to fit the definition of an emerging area of research.

8. Assignment:
http://buscatextual.cnpq.br/buscatextual/visualizacv.jsp?id=K4792170D6
TONIN, Neilor Avelino; GIORDANI, Luis Otávio; ADÁRIO, A. M. S..
Participação em banca de Carlos Alberto Ferrari.
Um Estudo Sobre o Sistema de Computação Distribuída Q2ADPZ. 2004.
Trabalho de Conclusão de Curso (Graduação em Informática) –
Universidade Regional Integrada do Alto Uruguai e das Missões.

9. Citation:
http://staff.science.uva.nl/~emeij/publications/OSIR2006_Edgar_Meij.pdf
http://www.emse.fr/OSIR06/2006-osir-p25-meij.pdf
Deploying Lucene on the Grid, Edgar Meij and M. de Rijke. In: Proceedings SIGIR 2006 workshop on Open Source Information Retrieval (OSIR2006), 2006 [PDF]
A grid enables the integrated use of resources, which are typically owned by multiple organizations and/or individuals and is in fact a system consisting of distributed, but connected resources [12]. It also encompasses software and/or hardware that provides and manages logically seamless access to those resources [13, 24]. Grids can be roughly classified in two categories: institutional grids (IG's) and global computing or P2P (GCP) systems [3, 23, 11]. GCP systems typically harvest the computing power provided by individual computers, using otherwise unused bandwidth and computing cycles in order to run very large and distributed applications [22, 15]. Some examples include SETI@home [38], LookSmart's Grub [28] (a voluntary initiave to crawl the Internet in a distributed fashion), and Zeta- Grid [40]. ZetaGrid is an attempt to verify Riemann's Hypothesisusing grid technology, with a reported peak performance rate of around 7000 GFLOPS. There are also (open source) packages such as XtremWeb [11], and Q2ADPZ [30] which allow to setup, deploy and run GCP projects. BOINC (the Berkeley Open Infrastructure for Network Computing) is another open source platform for public-resource distributed computing [3] and currently the enabling system for SETI@home, LHC@home, Einstein@home, Climateprediction. net, and many more.

10. Feedback:
http://grulic.org.ar/lurker/message/20040709.151501.e675f4be.es.html
[translation by babelfish]
Fecha: 2004-07-09 18:15 +300
A: sw, List of mail of the User group of Linux in Cordoba
Asunto: Re: [GRULIC] to cluster with mayusculas

Message mentioned by sw@,
>
> Pregunta....
> Hay people with desire to begin to see that she leaves, to see if it is possible?
> I suppose that if....pero that I can say to them?
> Single I am filosofando, and in fact single not much on nothing.
>
Surely that is people with desire to prove something...
Something but or less asi is QÀDPZ (Clears Advanced Distributed Parallel Zystem)
Segun its presentation...
[...]
A time ago I was proving it a pair of hours in the Intranet of my company and walks enough good...
Basicamente consists of a group of three binary ones: Masters, slave and client.
The thing is thus:
- a PC is defined Masters that is the one who receives the orders and it delegates and it controls the processes.
- they settle the enslaved soft in the PC with capacity of idle processing.
- they settle the soft client in the PC that “throws” processes to to cluster.
In individual it installs the Masters in a Network Hat 7,3 that I have of server the Dpto. of development and in 3 PC installs the Slave and Client.
I made a simple rutinita of mathematical calculos with numeros (several cascade curls for) that took the time from beginning and end of calculos and when it finished reduced End to me - Beginning
Corri in a PC normally and soon in that PC with the soft client. The time gave something me smaller, but nonmemory whatever...
Despues I did not have but time for tests...
If to somebody it interests to him to prove with something but great, I fall in love...
Greetings
Paschal Guillermo
Infrastructure IT
To integrate Solutions
It jumps 548 - It plants Discharge
Jesus Maria - Cordoba - X5220BGB
Tel/Fax: (03525) 421224
to www.integrarsoluciones.com.ar

11. Feedback:
Date: Fri, 28 Apr 2006 21:41:13 +0530
From: “premkumar srinivasan” <prem.srini@gmail.com>
To: zoran@idi.ntnu.no
Subject: Q2ADPZ
References: <3a3029120604280904p3de0b65fuad79cc6b1976425a@mail.gmail.com>
Hi Zoran,
I reached http://qadpz.idi.ntnu.no/paper-Crossroads/qadpz.html#qadpz while searching for available open-source distributed computing softwares.
I have downloaded q2adpz from http://sourceforge.net/projects/qadpz. While trying to setup in windows, I couldn’t open two .dsp files (cli_flic and slv_flic). Can I please know, from where can I download uncorrupted .dsp files? Also, where can we find makefile with respect to building q2adz in windows environment. Can I also know what are the other good open source softwares available for distributed computing in windows environment? A little advice from you, can help me, to dwelve into the amazing area of distributed computing.
Thanks Zoran!
--Prem.

From: Pavel Petrovic <Pavel.Petrovic@idi.ntnu.no>
Message-Id: <200605011604.k41G4w3k019849@furu.idi.ntnu.no>
Subject: Re: Q2ADPZ
To: premkumar srinivasan <prem.srini@gmail.com>
Date: Mon, 1 May 2006 18:04:22 +0200 (MEST)
Hi,
> Hi Pavel,
> Thanks for sending me the latest CVS snap-shot of q2adpz.
> I could download liblzo, libzlib, ssl libraries.
> But I am facing the following problems:
> 1) Where can I find flic.h in the code-base? Also, the make system requires
> flic_lib.lib. Where can I get this?
Good question. flic sample is maintained by Zoran. It relates to his research of visualization of stream data with the help of clusters. flic is FastLIC (Fast LIC), unfortunatelly, I do not have sources of FLIC. The search on the net tells me that FLIC is a past project of ZIP, but from their page http://www.zib.de/Visual/projects/ it seems that the project is no longer maintained. You can just ignore the FLIC sample unless you get more info from Zoran.
> 2) Where can I get crypto.lib and curl.lib? > libcurl-7.15.3’s libcurl.lib isn’t matching, it seems. I will try with some other version of curl.
I am using the libcurl3-dev 7.14.0-2ubuntu. Are you getting some compilation errors with the newer version? For the start, it is better to not use libcrypto (set HAVE_OPENSSL = 0 in Makefile.base).
Q^2ADPZ is an experimental research software, it is not [yet] a complete product.
Let me know if you have further questions.
Pavel.

12. Feedback:
Subject: Question Regarding Using QADPZ w/ MPI (i.e. qadpz_mpirun)
Date: Mon, 8 May 2006 14:16:51 -0700
Message-ID:
<8AB7DFF4B7187C43B0C93FA2D55E5B8C05AF8383@xcgca210.northgrum.com>
From: "Carl, Andrew" <a.carl@ngc.com>
To: <zoran@idi.ntnu.no>
Cc: "Carl, Andrew" <a.carl@ngc.com>
Mr. Constantinescu,

I am attempting to understand the implementation of the MPI w/QADPZ. Which version did you use in your testing, and is there any documentation available? I have contacted Mr. Petrovic, but he stated that you were the author of the MPI related upgrades associated w/qadpz_mpirun.
Thanks,

Andy Carl

13. Feedback:
Date: Fri, 26 Oct 2007 16:53:27 +0200
To: zoranc@users.sourceforge.net
From: "Marcus Dapp Survey-Admin (sg)" <swpat-floss@gess.ethz.ch>
Subject: Software patents and the 'qadpz' project - A scientific survey (sg)
Message-ID: <8ff4bab879dea18739390188a343041d@www.swpat-floss.ethz.ch>
Dear zoranc!
There is considerable debating in the Free/Libre/Open Source Software communities about software patents; but what do we really know? What are your own experiences with software patents in the qadpz project? We are cordially inviting you to participate in our global scientific survey on software patents and FLOSS projects.
Participation is by invitation only. Only a sample of project leaders/key developers of active SF projects (August 2006) have been invited. So, it is important that your project is represented as well. Please see the survey page for our privacy policy.
=> Start from here: http://www.swpat-floss.ethz.ch/lv/index.php?sid=6&token=1566224118
As your participation is really important, we include everybody in a lottery who completes the questionnaire. The prizes are nice, we think:
1st—A green 'XO' (OLPC) laptop, sponsored by Google's Open Source Program Office[2]
2nd—A free 'Neo1973' mobile phone, sponsored by OpenMoko/FIC[3]
3rd—Be surprised. We aim for similar 'coolness' as the other prizes ;-)
There are only multiple choice questions, so answering will be straightforward. We hope you find coming up with answers as exciting as we found coming up with questions.
=> Start from here:
http://www.swpatfloss.ethz.ch/lv/index.php?sid=6&token=1566224118
Thanks for helping us by submitting your response ideally within the next days. If you face technical problems, please email Marcus at swpat-floss@gess.ethz.ch with the subject line: 'bug-report-sg'.

This is a joint project of the Center for Comparative and International Studies (CIS), the Chair for Strategic Management and Innovation (SMI), and the Chair for Law&Economics at ETH Zurich, Switzerland[1].
Thank you very much for your interest, time and invaluable contribution!
Professor Thomas Bernauer, http://www.cis.ethz.ch
Professor Georg von Krogh, http://www.smi.ethz.ch
Professor Gérard Hertig, http://www.hertig.ethz.ch
Marcus M. Dapp, PhD candidate
Marcus M. Dapp | WEC C 19 | ETH-Zentrum | CH-8092 Zurich | Switzerland

14. Citation:
http://www.cs.montana.edu/techreports/2007/MohammadFuad.pdf
AN AUTONOMIC SOFTWARE ARCHITECTURE FOR DISTRIBUTED APPLICATIONS
PhD by Mohammad Muztaba Fuad
QADPZ [19] provides an open source framework that allows the management and use of the computational power of idle computers in the network using autonomic principles. QADPZ is implemented in C++ and uses MPI as its communication protocol, which restricts this system to a certain class of architectures. It also deploys a masterslave pattern for task distribution, which actually does not follow the autonomic system architecture and it does not take any measure to overcome a single point of failure, e.g.
the master node. The clients and the slaves (which do the actual work on behalf of the client) talk to each other by the use of a shared disk space, which is certainly a performance bottleneck and requires costly synchronization.

15. Citation:
http://http://www.emse.fr/OSIR06/2006-osir-CONTENT.pdf

Grids can be roughly classi-fied in two categories: institutional grids (IG's) and global computing or P2P (GCP) systems [3, 23, 11]. GCP systems typically harvest the computing power pro-vided by individual computers, using otherwise unused band-width and computing cycles in order to run very large and distributed applications [22, 15]. [...] There are also (open source) packages such as XtremWeb [11], and Q2ADPZ [30] which allow to setup, deploy and run GCP projects.

16. Forum:
http://tech.groups.yahoo.com/group/neat/message/2780
Re: NEAT Supervising NEAT
Ken,
First, I am counting attempting to achieve 80% benefit for 20% effort. And second, I intend to use qadpz to achieve a discrete form of parallel processing.
The various experiments and their associated ".ne" parameter files reveal a limited range for the various parameters. That being the case, could you make any suggestions as to the max ranges which have been found to be stable and "play nicely" with neat (i.e. not blow-up neat), based upon your experiences?
Thanks,
Andy

http://tech.groups.yahoo.com/group/neat/message/2772
Re: Parallel NEAT

Sidhant,
You might take a look at QADPZ at the following link:
http://qadpz.sourceforge.net/
, and incorporating your driver into the client "qadpz_run" source code. The master & client can be incorporated onto the same machine if required.
By the way, I enjoyed reading your report in the files section!
AFC

>
> Hullo Joe
>
> That sounds like something I have been looking for. A client-server mechanism is what I am thinking of at the moment, and not a multithreaded version of NEAT. You are talking of something that can be executed on a cluster, right??
> It would be very nice if you could share some of your code.
Thanks a lot..
> Sidhant
>

17. Forum:
http://groups.google.com/group/microsoft.public.de.vc/browse_frm/thread/8a9837d8513f9bce/21db1ad9d55bebc3?tvc=1&q=qadpz#21db1ad9d55bebc3
Newsgroups: microsoft.public.de.vc
From: "Lars Stegelitz" <lars.stegel...@t-online.de>
Date: Thu, 4 Dec 2003 20:52:07 +0100
Local: Thurs, Dec 4 2003 9:52 pm
Subject: Re: Wie CPU Speed herausfinden?

Sebastian Schwaiger wrote:
> Ich weiß, dass es nicht so schwer sein sollte, aber Google gibt zu CPU
> Speed algorithm nichts brauchbares her.
http://qadpz.idi.ntnu.no/doxy/html/cputicker_8cpp-source.html
sieht vielversprechend aus
MfG
Lars Stegelitz

Newsgroups: microsoft.public.de.vc
From: Hans J. Ude <hajue....@arcor.de>
Date: Fri, 05 Dec 2003 14:33:55 +0100
Local: Fri, Dec 5 2003 3:33 pm
Subject: Re: Wie CPU Speed herausfinden?
"Lars Stegelitz" <lars.stegel...@t-online.de> schrieb:

>Sebastian Schwaiger wrote:
>> Ich weiß, dass es nicht so schwer sein sollte, aber Google gibt zu CPU
>> Speed algorithm nichts brauchbares her.
>http://qadpz.idi.ntnu.no/doxy/html/cputicker_8cpp-source.html
>sieht vielversprechend aus

Sieht nicht nur vielversprechend aus, sondern hält das auch. Hat mir
schon sehr gute Dienste beim Profiling geleistet. Die Klasse kann
wesentlich mehr als nur die CPU Geschwindigkeit messen.
Hajü

18. Forum:
http://groups.google.com/group/fr.comp.os.unix/browse_thread/thread/c73eaa477f77f430/c512157d95c7f962?lnk=st&q=qadpz#c512157d95c7f962
Newsgroups: fr.comp.os.unix
From: William Wu <b...@no.spam>
Date: Fri, 10 Jan 2003 23:12:02 +0100
Local: Sat, Jan 11 2003 12:12 am
Subject: Re: b64encode.sh est-il dispo sous hpux?

On Fri, 10 Jan 2003 23:11:34 +0100, farid wrote:
> Bonjour à tous,
> je voulais savoir si le script b64encode.sh est censé être dispo sous hp
> ux, ou alors quelqu'un peut-il mele fournir.
tout dépends si le shell et les programmes sont aussi sur hppux, non ou bien je dis une conn*r*e ?
sinon j'ai bien trouvé ça ce qui me semble pas spécialement pour une plateforme particulière je sais pas si c'est ce que tu cherche ... tu n'as qu'à jeter un coup d'oeil :
http://qadpz.idi.ntnu.no/doxy/html/b64encode_8cpp-source.html
William.
comment ça mon mail marche pas ???
william.wu chez free.fr
Newsgroups: fr.comp.os.unix
From: « farid » <lfa...@free.fr>
Date: Sat, 11 Jan 2003 12:05:48 +0100
Local: Sat, Jan 11 2003 1:05 pm
Subject: Re: b64encode.sh est-il dispo sous hpux?
C'est exactement ce que je cherchais,
merci beaucoup William.
http://www.gridresources.info/
QADPZ - Quite Advanced Distributed Parallel Zystem
http://qadpz.sourceforge.net/
http://www.erlang.org/pipermail/erlang-questions/2003-July/009383.html
Distributing computations
Vlad Dumitrescu <>

Hi,
From: “Luke Gorrie” <>
> I’ve never done any of this stuff, but have been doing some reading
> and looking for an excuse to :-). You’re not getting any solid info
> out of me, but maybe some inspiring/entertaining/distracting links :-)
I don’t expect a solution, but just as you say - inspiration!
> It seems the main trick is to design an algorithm that can run in parallel.
Yes, that’s one thing that has to be tailored after the specific problem at hand.
> Then it seems a popular package today is Parallel Virtual Machine
> (PVM), http://www.csm.ornl.gov/pvm/pvm_home.html.
I was thinking about using Erlang as back-end :-)
I found some references at http://www.aspenleaf.com/distributed/distrib-devel.html, and I think ideas from Q2ADPZ (at http://qadpz.idi.ntnu.no) could be reused with relative ease. The fact is, ERTS does already a lot of the things that such a beast should do, and better - probably except only the security aspects. And, hey!, it’s also a good oportunity to use UBF, both -A and -B! :-) Thanks for the input. Regards,
Vlad

19. Citation:
http://www-compsci.swan.ac.uk/~csmark/PDFS/visualsupercomputing.pdf
Visual Supercomputing: Technologies, Applications and Challenges
COMPUTER GRAPHICS forum
Volume 24 (2005), number 2 pp. 217–245

20. Forum:
http://curl.haxx.se/mail/lib-2004-04/0366.html
Re: effects of removing curl_formparse?
From: Tor Arntsen <tor_at_spacetec.no>
Date: 2004-04-30
On Apr 30, 10:20, Daniel Stenberg wrote:
>Hi
>curl_formparse() been deprecated and adviced not to be used since 21 August 2001.
>Can anyone mention anything or anyone that would be affected if we removed it
>completely?
I asked google.. didn’t seem to be much out there (but a lot of references
to updates becuse curl_formparse() has been deprecated):
http://qadpz.idi.ntnu.no/doxy/html/GetURL_8cpp-source.html
http://www.seismo.unr.edu/ftp/pub/updates/bankert/php-4.0.4pl1/ext/curl/curl.c
The strange thing is that when I first searched I got more than 21 pages of references, a moment later only 13, then just 10.. is curl_formparse() being purged all over the place out there? :-) (there’s probably a more google-technical explanation for this I guess)
-Tor

21. Forum:
http://www.broadbandreports.com/forum/remark,8785738
franconia

join:2001-07-04
Alexandria, VA
Opinions on best DC development platform?

I am interested in developing a DC system to analyze a large dataset of atmospheric measurements. Since the data and research area are rather mundane, I don't foresee a lot of internet community interest in this project. Nonetheless, capturing the unused cycles on our LAN would be great.

Through Google I have run across a few DC backends, such as BOINC, FIDA, and QADPZ. All these seem to run from Linux or BSD server systems. Does anybody who developed a DC project have any software recommendations for me? Any familiarity with the DC platforms listed above or others?
to forum · permalink · 2003-12-14 16:50:24 · (locked)

22. Citation:
http://www.iit.edu/~mummsat/wsrf/SATISH_K_%20MUMMADI.pdf
from CV
COMPUTER SKILLS:
Distributed Computing: Using Parabon Frontier SDK for Java applications, QADPZ toolkit.

23. Forum:
http://hp.parallel.ru/parBB/viewtopic.php?p=4168
Ищу библиотеку для использования idle процессорного времени офиса (везде WinXP).
Интересует возможность выполнять jobs, при этом должно использоваться только idle время, компы могут перезагружаться, т.е. никто ничем не обязан. Нужна система автоматического апдейта библиотек на агентах.
MPICH поэтому и не подходит.
Я нашел несколько библиотек:
http://www.alchemi.net/index.html - похоже, то что надо
http://qadpz.sourceforge.net/
http://ngrid.sourceforge.net/index.html
http://mygrid.sourceforge.net/

Кто нибудь имеет опыт работы с ними, или может что-то посоветовать?

24. Feedback:

Subject: QUADPZ
From: Manel Soria Guerrero <manel@labtie.mmt.upc.es>
To: Zoran Constantinescu-Fulop <zoran@idi.ntnu.no>
Message-Id: <1054202508.2900.5303.camel@congre.cttc.org>
Date: 29 May 2003 12:01:49 +0200

Zoran,

I've been talking about QADPZ with Ramiro, our system administrator.
You can reach him at raq@labtie.mmt.upc.es
I'll try to write a summary of our conversation :)
-He had problems with the binary and with the CVS versions of the code so he downloaded and compiled the last stable version.
-He needs time to do more tests, but he likes the design of the code, the definition of the needs with XML, the security, etc.
-It is our opinion that it would be a good idea to focus first on sequential jobs and when they are closed, go for the parallel executions. In our case, parallel executions are complex, they need lots of resources (RAM, disk, network), good load balance, etc and we would prefer to run them on the cluster.
-We wonder if it would be possible for the end users to control the executions.
This is, the executions need a long time to be completed and it is normal that the (research) programs fail to converge, so the executions must be stopped and resubmitted with slightly different parameters. The codes write in one or several text files a summary of how is the execution going, and from time to time, a rather large binary file.
The users should be able to get this information as soon as it is generated and kill the codes if necessary.

Hope this helps and I'll write more when/if I have more information. Please contact Ramiro if you are interested, maybe the problem with the last version is already fixed.

Best regards,
Manel

25. Feedback:
Subject: Re: QUADPZ
From: Ramiro Alba Queipo <raq@labtie.mmt.upc.es>
To: Zoran Constantinescu-Fulop <zoran@idi.ntnu.no>
Message-Id: <1054235109.5902.12.camel@mundo.cttc.org>
Date: 29 May 2003 21:05:09 +0200
On Thu, 2003-05-29 at 18:04, Zoran Constantinescu-Fulop wrote:
> Hi Ramiro,
> Manel told me to contact you about QADPZ. My name is Zoran and I am one of the
> developers. We are planing to make a new release with some of the new features we
> added. You had some problems with the binary and CVS versions... could you tell me
> more about > these problems? so that we can fix them for the new release :)
Where about libstdc++ (binary version) and something related with
MPI (not finding a source file). Anyway I must say that I did not try
very hard as I prefered going into installation step I know about
functionality. I will be more specific after trying better the
functionality from the user's view.
> > I agree that we should start with the sequential type of jobs and see how it works.
The parallel code should be easier to test afterwards.
> > Regarding user control of the executions: after a job/task is
> started, the owner (i.e. user) can then control it, like for

> example send a special control-message or stop the task. For
> the data, input/output files are downloaded/uploaded from a
> web server (or any other type of server, like for example ftp).
> All input files are downloaded before the task starts, while
> the output files are uploaded after it finishes. We could change
> a bit the code to do the upload of some intermediary output
> files more often. For example every 5 minutes, or even catch
> the “fclose()” type system library calls and do an aditional
> upload there :) The later should be quite easy to do in Linux,
> though i’m not sure how to do it in Windows :)
>
I would need to make some aditional tests so as I can know a bit more, so as I can ask you some questions. I will be a bit busy the next three days, but I promise to contact you next week.

See you
Ramiro
> Cheers,
> --zoran

26. Feedback:
Subject: Re: QADPZ
From: Ramiro Alba Queipo <raq@labtie.mmt.upc.es>
To: Zoran Constantinescu-Fulop <zoran@idi.ntnu.no>
Message-Id: <1054297658.9201.1.camel@mundo.cttc.org>
Date: 30 May 2003 14:27:38 +0200

On Thu, 2003-05-29 at 21:31, Zoran Constantinescu-Fulop wrote:
> >> problems [...] > > Where about libstdc++ (binary version) [...]
> I compiled now a binary version which doesn’t need anymore a specific libstdc++. :)
> That was my intention also first time, but probably I’ve put the wrong archive on the
> web... :(There is also an SSL-enabled version on the web site:
> http://qadpz.idi.ntnu.no/download/bin/
Now it seems to be running. I will keep you informed. Thanks Zoran.
See you

27. Feedback
From: Manel Soria Guerrero <manel@labtie.mmt.upc.es>
To: Zoran Constantinescu-Fulop <zoran@idi.ntnu.no>
Message-Id: <1054541358.2899.5369.camel@congre.cttc.org>
Date: 02 Jun 2003 10:09:18 +0200
> > It would be really nice to be able to control a number of files (maybe
> > specified with that XML thing ??), as if they were on the local machine.
> > > i’m not sure i understant exactly what you mean...
When the programs run on the local machine, to see how are the programs

going, from time to time we usually:
-print the tail of one or more control files, that contain lots of numbers (ascii)
-gnuplot an ascii file
-use a visualization code to see a large binary file
It would be nice if QADPZ could allow to do that on some remote files just as if they were at the local machine. Maybe a possibility could be to specify the files that must be controlled, using the same XML file that contains the description of the job requirements (if I understood correctly).

28. Feedback:
Subject: QADPZ: Got the first successful execution
From: Ramiro Alba Queipo <raq@labtie.mmt.upc.es>
To: zoran@idi.ntnu.no
Message-Id: <1054750909.2849.12.camel@mundo.cttc.org>
Date: 04 Jun 2003 20:21:50 +0200

Hi Zoran:
After some minor problems, I have got the first successful execution using the simple.cpp example. I now can understand much better the way qadpz works and I also already have some questions/suggestions, but I prefer to go on trying with our programs so as to test real cases and then talk to you. I also compiled the last CVS version, but with no MPI (deactivated). (In any case no matter at this moment), and played with slv_app.cpp.
May be only two questions question: How can I send a job to execution and forget (not waiting)? Can a slave decide that a job must be stopped? How?
See you,
Ramiro

29.Feedback:
Date: Wed, 6 Aug 2003 02:45:26 -0700 (PDT)
From: Devesh Singhal <deveshsinghal2003@yahoo.com>
Subject: Problem : in implementing QADPZ utility
To: zoranc@acm.org
Hello Mr. Zoran Constantinescu,
Sir,myself is Devesh Singhal and I have recently downloaded your 0.8beta version of QADPZ utility. But there are some problems arrriving during execution,I am mentiong here - [1]. As you have mentioned in article 5.3.4 of Q2ADPZ User & Developer Manual , point 1 -> It is quite understandable and I have copied src/wscript/*.cgi (i.e. all .cgi files) to /var/www/html/wscript
point 2 -> Statement " Set the location of these scripts in client.cfg and slave.cfg files before installing them accordingly."
But to which variables in client.cfg and slave.cfg, these scripts must be assigned.
[2]. I have renamed the Library libslv-app.so to libslv.so in this utility. But when I execute ./qadpz_run at bin directory, an error occurs stating that error in loading libslv.so to a new file (file has an rbitrary name) It creates this new file at destination directory and return download Ok, but new file remains empty(size 0 bytes).

Sir I am here attaching files for more acccurate information-
- output of ./qadpz_run command and status of master and slave.
- XML file simple2.xml.
- file slave.log.

Please respond it Sir.
Thanking you Sir.
Devesh Singhal

30. Feedback:
From: "Deraldo" <deraldo@veloxmail.com.br>
To: <zoran@idi.ntnu.no>
Subject: Qadpz
Date: Mon, 8 Sep 2003 18:29:24 -0300
Message-ID: <000001c37650$46a8e180$64c8a8c0@dedaserver>
Zoran!
Im trying to use the qadpz into a windows environment. We cant get it! Using the linux env, we can connect the slave and the client. But this last one stops after requiring some reserved slaves. Could you help us?
thanks in advance!

31. Feedback:
Date: Sun, 21 Sep 2003 21:00:46 +0200 (MEST)
From: Zoran Constantinescu <zoran@idi.ntnu.no>
To: =?ISO-8859-1?Q?Leif_Snorre_Sch=F8yen_Boasson?=
<Leif.Snorre.Schoyen.Boasson@idi.ntnu.no>
cc: Zoran Constantinescu-Fulop <zoran@idi.ntnu.no>
Subject: psim, mpi, compression
Message-ID: <Pine.GSO.4.51.0309212026130.17828@dionysus.idi.ntnu.no>
MIME-Version: 1.0
Hi,
As I told you, I was hacking a bit around with MPI and compression. I made a few tests with your program and here are some results...

grid 128x128 (one message size is 128 kBytes)
particles 1048576
time steps 100
#nodes 8

MPI version	simul.time
MPICH	53.7 sec
MPI-QADPZ no compress	28.6 sec
MPI-QADPZ with LZO	17.6 sec
MPI-QADPZ with ZLIB	39.1 sec
MPI-QADPZ with BZIP2	87.2 sec

As you can see, it's quite promissing... =D> :-) LZO, ZLIB, BZIP2 are different compression algorithms. I ran each simulation two times, to be sure about the results ;). See below the ouput of one of each simulation.

The small hack is made as part of the tiny MPI library on top of QADPZ, the distributed computer project I'm working on (http://qadpz.sourceforge.net).

If you want, I can show you how to play with it, so that you can make more tests with it.

Cheers,
--zoran

MPICH

No. of time-steps: 100
Part_rho took : 7.02368 seconds
Push_v took : 4.77408 seconds
Push_loc took : 1.16180 seconds
Solve took : 2.33222 seconds
Field_grid took : 0.07043 seconds
Simulation took : 53.47985 seconds
MPI_Allreduce took : 37.41524 seconds
Total simulation : 53.72704 seconds

MPI-QADPZ no compression

No. of time-steps: 100
Part_rho took : 4.41704 seconds
Push_v took : 6.04893 seconds
Push_loc took : 1.17400 seconds
Solve took : 2.42634 seconds
Field_grid took : 0.06443 seconds
Simulation took : 28.59725 seconds
MPI_Allreduce took : 10.00685 seconds
Total simulation : 28.60410 seconds

MPI-QADPZ with LZO

No. of time-steps: 100
Part_rho took : 4.44002 seconds
Push_v took : 5.16125 seconds
Push_loc took : 0.93403 seconds
Solve took : 2.50315 seconds
Field_grid took : 0.06425 seconds
Simulation took : 17.55874 seconds
MPI_Allreduce took : 4.26227 seconds
Total simulation : 17.56558 seconds

MPI-QADPZ with ZLIB

No. of time-steps: 100
Part_rho took : 4.52847 seconds
Push_v took : 6.56303 seconds
Push_loc took : 1.27463 seconds
Solve took : 2.67737 seconds
Field_grid took : 0.06879 seconds
Simulation took : 39.10445 seconds
MPI_Allreduce took : 23.68118 seconds
Total simulation : 39.11132 seconds

MPI-QADPZ with BZIP2

Part_rho took : 5.24602 seconds
Push_v took : 8.42995 seconds
Push_loc took : 1.43487 seconds
Solve took : 3.39705 seconds
Field_grid took : 0.07894 seconds
Simulation took : 87.14938 seconds
MPI_Allreduce took : 67.71850 seconds
Total simulation : 87.15621 seconds

32. Feedback"
Date: Tue, 30 Sep 2003 19:32:18 +0200 (MEST)
From: Cyril Banino <Cyril.Banino@idi.ntnu.no>
To: Zoran Constantinescu <zoran@idi.ntnu.no>
cc: Cyril Banino <Cyril.Banino@idi.ntnu.no>
Subject: Re: Q2ADPZ
> one idea, for example, would be to use QADPZ as a scheduler for ClustIS and, in
> addition to the 37 nodes we have now, we could add some more office-PCs with
> computing power available.
> we could talk more about this if you want...
> cheers,
> --zoran
I'd like that. Let's take about it one day when you have time.
See you,
Cyril.

33. Feedback:
Subject: Request for info on QADPZ systems
To: zoran@idi.ntnu.no
Message-ID: <OF526BB527.14345D7A-ON65256DCD.0013AAD1@interliant.com>
From: avijayakumar@frost.com
Date: Tue, 28 Oct 2003 10:09:53 +0530

Dear Mr.Zoran Constantinescu,
I would like to thank you for your kind and quick response. We had a long festival weekend here in India,so I was not able to reply you immediately.As I mentioned earlier,I am working on a research service which focusses on the technological developments in the area of distributed systems.With respect to this I have few questions in my mind,answers to which will give me insight into your work and about the topic

1.Can you give a thorough description of your work which will be understood by a person without any technical expertise.
2.What is the driving factor for your research?
3.What are the competing technologies and what are their deficiencies which has been addressed in your systems?
4.What are the major challenges faced during your research in evolving to the marketplace?How is it addressed?
5.How are the security issues dealt in such open source distributed systems?
6.What are the current and potential applications for the technology? When do you think the potential applications will become commercial? Can you list those applications you expect to have most impact and also comment on their degree of expected impact?
7.Is the technology available for licensing? Are you interested in partnering to further develop the technology or applications? Are there patents on the technology? Can you provide me with the patent titles and numbers?
8. In your opinion which do you think are the emerging technologies in the area of distributed systems playing a key role in the market?

If you think there are any relevant documents related to this study which you can share with us, kindly attach the same along with this mail.

Thank you for your time and cooperation.I look forward to hear from you soon.
Regards, Amreetha

Ms.Amreetha Vijayakumar
Research Analyst-Technical Insights, Frost & Sullivan
Chennai,India.
www.ti.frost.com

34. Feedback:
Ph : +91-44-24314263/5/6/7 Ext-299
Fax :+91-44-24314264
email : avijayakumar@frost.com

http://www.undergroundnews.com/forum/ubbthreads.php?ubb=showflat&Number=2392
Re: distributed computing sinetific sinetific Offline
nobody
Registered: 03/02/02
Posts: 815

Loc: Ann Arbor
http://qadpz.sourceforge.net/
Platforms supported are Linux, Unix, Win32 and MacOS X.

Seems to be what you are looking for.
you specify master and slave computers the master sends the computing out to the slaves.

I've actually beent thinking of trying something like this myself since i have a few computers that dont really do a lot with their CPU cycles.

On Tue, 21 Oct 2003 avijayakumar@frost.com wrote:

> Hello Mr.Zoran Constantinescu,
>
> I read with interest about your research work on Quite Advanced Distributed Parallel Zystem. I am an analyst with the Technical Insights division of Frost and Sullivan (www.ti.frost.com). We publish several subscription services on topics such as sensors, IT,microelectronics, and Advanced materials that are read by researchers,engineers and executives at top companies worldwide. We specialize in new developments with commercial promise. I am currently working on a research service which focuses on Distributed systems,its applications and the core technologies associated with it which are evolving into the market from the research labs.
>
> For this I would like to incorporate your latest developments in this field. I wondered if you mind taking the time to answer a few questions for us. Please let me the know if its appropriate sending them over to you.
> I look forward to hearing from you soon.Thank you for your time and cooperation.
>
> With Best Regards,
> Amreetha

35. Assigment:
http://www.idi.ntnu.no/~zoran/Hydro2/Velo10d-lic-anim.html
data /wrk_c4/hdb2/BACKUP/clustis/zoran/data/qadpz-log/1056737676

Test case: 3cyl
1. flow in a channel around 3 cylinders (2D, 26600 elems, 13567 nodes, 400 time steps)
2. LIC animation (small 25 MBytes)
3. LIC animation (large 125 MBytes)

36. Forum:
http://www.beowulfwindows-reserves.us/checkpoint-restart.htm

Simulation was done using the CPM Navier-Stokes solver developed at SINTEF. The results were obtained by running a distributed computing simulation using 8 desktop lab-computers (not a dedicated cluster!). The QADPZ distributed computing (desktop

grid computing) system was used, developed as part of the CSE project. A lightweight MPI library on top of QADPZ was used for communication purposes.

37. Forum:
http://ml.tietew.jp/cppll/cppll_novice/thread_articles/446

Subject: [cppll_novice:0447] Re: openssl で RSA public key を読み込むには
From: Hisao Tsutsumi <tsutsumi@pf.highway.ne.jp>
Date: Wed, 30 Jun 2004 00:52:18 +0900
X-Mailer: Microsoft Outlook Express 6.00.2800.1409
Message-Id: <00f301c45df1$0f1ac290$e29768db@BIGBOY>
References: 446

堤です。

私自身はまったく分かっていませんので間違っていたらゴメンナサイ。m(_ _)m

PEM_read_RSAPublicKey ではなく、PEM_read_RSA_PUBKEY を使ってみては如何でしょう？

#google で PEM_read_RSAPublicKey を検索したら
#http://qadpz.idi.ntnu.no/doxy/html/cryptest_8cpp-source.html
#を見つけました。
#このソースでは PEM_read_RSAPublicKey ではなく、
#PEM_read_RSA_PUBKEY を使うようにしているみたいです。
#……といってこれが正しいかどうかは分からないですけど。

では、では。

--
Hisao Tsutsumi <tsutsumi@pf.highway.ne.jp>

38. Forum:
http://www.mail-archive.com/expert@linux-mandrake.com/msg70236.html
[expert] Distributed computing package
Ezequiel Martín Cámara
Fri, 13 Jun 2003 05:06:37 -0700
What about integrating some distributed computing system into Mandrake?
There are a couple of open-source systems that generalize over the setup:
http://boinc.ssl.berkeley.edu
http://qadpz.sourceforge.net
I -and, I guess, many other users- would be happy to give my idle computing power -and I have several Mandrake machines running most of the time- to Mandrake in exchange of, say, Club membership, Mandrake packages. (Or cash)

Would it be very hard for Mandrake to sell all those petaflops commercially? I've been Goggling around and I've found .15$/hour for a new Compac (http://www.tech-report.com/onearticle.x/4467) and 7500$/year for a 400Mhz PII

(http://www.mithral.com/pressroom/archive/2000-11-SciAm.html).

I mean, all of us want Mandrake (the company) to survive financially, but many aren't ready to actually pay them. This would be a way to give back that would not actually cost a penny to users(at least, for those users to whom the company/Daddy pay the electric bill

Even if the cash cow is not feasible -and I can't think why not- it would be nice to have some OS distributed computing effort integrated on Mandrake. That would mean *so* much computing power...
-- Ezequiel Martín Cámara
http://www.geocities.com/ezequielmartin
http://www.radicalparty.org

39. Citation:
http://java.icmc.usp.br/dilvan/papers/2004-Webmedia/TanakaFinal.pdf
Um Sistema de Controle para Web Farms
Webmedia 2004

Para tal, é preciso que o projeto e a implementação dos sistemas de computação, software, armazenamento e suporte exibam alguns fundamentos básicos, tais como [7]: autonomia, flexibilidade, acessibilidade e transparência. A autonomia pode ainda ter as seguintes propriedades [2][12]: auto-configuração, auto-otimização, auto-tratamento e auto-proteção.
[12] Z. Constantinescu, “Towards an Autonomic Distributed Computing System”, Proc. of the 14th Inter. Workshop on Database and Expert Systems Applications, IEEE Computer Society, 2003, pp. 694-698.

40. Citation

www.netlab.hut.fi/opetus/s384030/k06/papers/SecuredRemoteTrackingOfCritical.pdf
Secured Remote Tracking Of Critical Autonomic Computing Applications 2004
“…but also because of the need to integrate multiple heterogeneous environments, and to extend beyond company boundaries into the Internet [1]”
[l] Zoran Constantinescu. ‘Towards an Autnnomic Distributed Computing System, ”
Proceedings of rhe 14th International Workhop on Database and Expert Systems Applications (DEXA ‘OS), 2003

41. Citation:
http://www.scientificjournals.org/journals2007/articles/1198.pdf
An Autonomic Software Architecture for Distributed Applications 2007

QADPZ [19] provides an open source framework that allows the management and use of the computational power of idle computers in the network using autonomic principles. QADPZ is implemented in C++ and uses MPI as its communication

protocol, which restricts this system to a certain class of architectures. It also deploys a masterslave pattern for task distribution, which actually does not follow the autonomic system architecture and it does not take any measure to overcome a single point of failure, e.g. the master node. The clients and the slaves (which do the actual work on behalf of the client) talk to each other by the use of a shared disk space, which is certainly a performance bottleneck and requires costly synchronization.
19. Constantinescu Z., "Towards an Autonomic Distributed Computing System", *14th International Workshop on Database and Expert Systems Applications*, pp. 699-703, 2003.

42. Citation:
http://www.comp.leeds.ac.uk/kwb/publication_repository/2005/cgf_006.pdf
Visual Supercomputing: Technologies, Applications and Challenges
Computer Graphics Forum 2005
Though the development of generic software environments for autonomic applications is still in its infancy, several attempts were made, which include projects such as QADPZ [149], AUTONOMIA [150] and Almaden Optimal- Grid [151]. QADPZ [149] provides an open source framework for managing heterogeneous distributed computation in a network of desktop computers using autonomic principles. In QADPZ, the system complexity is hidden in the middleware layer, facilitating self-knowledge, self-configuration, self-optimization and self-healing.
149. Z. Constantinescu. Towards an autonomic distributed computing environment. Proc. 14th Int.Workshop on Database and Expert Systems Applications, pp. 699–703, 2003.

43. Citation
http://arxiv.org/pdf/cs/0607061
On Some Peculiarities of Dynamic Switch between Component Implementations in an Autonomic Computing System
The success of an autonomic system behavior is essentially determined by ability to detect or predict overall performance that is actually the ground for management of autonomic components, in particular, for activation of an appropriate component implementation. For this, establishing of mathematical abstractions and models giving criteria governing the sequence of switches between component implementations is an important point of autonomic computing [2-5].
5. Z. Constantinescu, Towards an Autonomic Distributed Computing System, *Workshop on "Autonomic Computing Systems", ACS'2003,* September 1-5, Prague, Chech Republic (2003).