Grid-enabling Non-computer Resources

Ph.D. Dissertation

Martin Rehr

Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
Submission date: August 31th, 2010

Acknowledgments

The greatest thanks I owe to my advisor, Professor Brian Vinter who arranged my Ph.D. stipend and with his never-ending pipeline of research ideas and positive minded approach to problem solving has been a major inspiration through the project. In addition to his great knowledge, Brian has introduced me to the world of research through his worldwide network of research partners as well as letting me attend several international conferences and Ph.D. summer schools.

In the spring of 2009 Dr. Toni Cortes from the Barcelona Supercomputing Center kindly hosted me for six months despite the fact that he didn't know neither me, Brian or any other from our research team prior to accepting hosting me. I'm deeply thankful to Toni for the enthusiasm he showed to my project and the many hours he spent discussing various aspects and solutions with me. Besides Toni a big thanks goes to his Ph.D. student Jonathan Marti who developed the Oraculo predictor which is used in the last part of the project and with whom I had many joyful discussions regarding the the prediction system. I would also like to thank Ernest Artiaga for teaching me how to generate various representations of huge scientific data sets using his graph generation script. Lastly I'll like to thank all the people I met at BSC for making my stay very enjoyable.

In Copenhagen Jonas Bardino has been a great companion both developing and managing the Minimum intrusion Grid as well as discussing the technical aspects of my research and verifying my papers before submission. My former Ph.D. colleague Rasmus Andersen has been a great travel companion at various conferences and summer schools. He and I have had endless discussions during my research about various technical aspects and he has been a great inspiration when it comes to improving my scientific writing skills.

Last but not least my beautiful girlfriend Vibeke deserves a big hug for supporting me through the project, cheering me at the sour hours where nothing worked the way it was supposed, as well as supporting my sometimes odd working schedule, my late Friday discussion sessions, all the times where I favored work above social activities, and all the travels, specially the six months in Barcelona where she took care of the farm all by her self.

The presented research is funded by "Det frie forskningsrad for natur og univers".

Contents

1 Introduction **1**

1.1 Motivation . . . 1
1.2 Public Resource Computing . . . 2
1.3 Grid for the Public . . . 3
1.4 Non-computer Resources . . . 3
1.5 Challenges We Are Facing . . . 6
1.6 The Grid Concept . . . 7
1.6.1 Accessing Resources . . . 7
1.6.2 Interactive Shell . . . 7
1.6.3 Sensors and Laboratory Equipment . . . 7
1.6.4 Office Roaming Grids . . . 8
1.7 Grid Middlewares . . . 8
1.7.1 First Generation Grids . . . 9
1.7.2 Second Generation Grids . . . 9
1.7.3 Grid Scheduling . . . 11
1.7.4 Grid Scalability . . . 11
1.7.5 Job Fault Tolerance . . . 12
1.7.6 Runtime Environments . . . 13
1.7.7 Virtual Organizations . . . 13
1.7.8 Security: Authentication and Anonymity . . . 15
1.7.9 Grid Storage . . . 15
1.7.10 Connecting Resources to the Grid . . . 16
1.7.11 Choosing a Grid Middleware . . . 17
1.8 Presented Work . . . 17
1.8.1 The One-Click Grid Resource Model . . . 18
1.8.2 The PS3™ Grid Resource Model . . . 18
1.8.3 Application Porting and Tuning on The Cell-BE Processor . . . 19
1.8.4 The Remote Memory Library . . . 19
1.9 Contributions . . . 20
1.10 Publications . . . 20

2 The One-Click Grid Resource Model **22**
2.1 The MiG Scheduler . . . 22
2.2 The One-Click VMware Player Model . . . 24
2.3 The Java One-Click Model . . . 25
2.4 Remote File Access . . . 27
2.4.1 Block Size Estimation . . . 27
2.5 Checkpointing . . . 28
2.5.1 Transparent Checkpointing . . . 28
2.5.2 Semi-transparent Checkpointing . . . 29
2.6 Experiments . . . 29
2.6.1 One-Click as a Concept . . . 29
2.7 One-Click Summary . . . 30

3 The PS3™ Grid Resource Model **32**
3.1 About the PS3™ Game Console . . . 32
3.2 The PS3™ Grid as a Resource . . . 33
3.3 The PS3-LIVECD . . . 34
3.4 The MiG PS3-LIVECD . . . 34
3.4.1 Security . . . 35
3.4.2 Sandboxing . . . 35
3.4.3 File Access . . . 35
3.5 The Execution Environment . . . 35
3.6 Experiments . . . 37
3.6.1 Job Overhead and File Performance . . . 37
3.6.2 Protein Folding . . . 37
3.7 PS3-LIVECD Summary . . . 38

4 Application Porting and Tuning on The Cell-BE Processor **39**
4.1 The Cell-BE Processor . . . 39
4.2 Porting Towards the Cell-BE . . . 41
4.2.1 Task and Memory-parallelization . . . 41
4.2.2 The Nqueens Solution . . . 42
4.3 Register-line Optimizations . . . 43
4.3.1 Recursive vs. Iterative Methods . . . 43
4.3.2 Branch Prediction and Elimination . . . 43
4.3.3 The Nqueens Solution . . . 44
4.4 Data Parallelization . . . 44
4.4.1 The Nqueens Solution . . . 45
4.5 Instruction Parallelization . . . 46
4.5.1 The Nqueens Solution . . . 46
4.6 Nqueens Summary . . . 47

5 The Remote Memory Library **48**
5.1 Memory Management . . . 49
5.2 Kernel-level vs. User-level . . . 49
5.3 The Remote Swap Framework . . . 50
5.4 The Remote Memory Library . . . 50
5.5 Page Eviction . . . 50
5.6 Page Retrieval . . . 51
5.7 Page Blocks . . . 51
5.8 The Memory Server . . . 52
5.9 Implementation Details . . . 53
5.9.1 UDP vs. TCP . . . 53
5.9.2 Nagle's Algorithm . . . 53
5.9.3 The Local Page Table . . . 54
5.10 Memory Allocation . . . 55
5.11 Page Eviction . . . 56
5.12 Page Retrieval . . . 56
5.13 Experiments . . . 60
5.14 Sequential Data Access . . . 60
5.15 Linux Disk Swap . . . 62
5.16 Sequential Data Access with Writes . . . 62
5.17 Scattered Memory Access . . . 63
5.18 Scattered Memory Access with Write . . . 64
5.19 Lattice Boltzmann . . . 64
5.20 Fast Fourier Transform . . . 64
5.21 Barnes-Hut . . . 66
5.22 Experiment Summary . . . 68
5.23 Initial Remote Memory Library Summary . . . 68

6 Prediction Based Page Prefetching **70**
6.1 Oraculo . . . 71
6.2 Combining Oraculo with The Remote Memory Library . . . 73
6.3 Online Prediction Problem . . . 84
6.4 Experiments . . . 84
6.5 Sequential Data Access . . . 85
6.6 Sequential Data Access with Writes . . . 86
6.7 Scattered Memory Access . . . 88
6.8 Scattered Memory Access with Write . . . 89
6.9 Lattice Boltzmann . . . 91
6.10 Fast Fourier Transform . . . 91
6.11 Barnes-Hut . . . 93
6.12 Simulated Network Latency . . . 95

6.13 Experiment Summary 97

7 Future work **99**

8 Conclusion **101**

A Publication 1 **109**

B Publication 2 **124**

C Publication 3 **135**

D Publication 4 **142**

E Publication 5 **151**

F Publication 6 **172**

Chapter 1

Introduction

1.1 Motivation

The scientific modeling community has a seemingly endless need for processing power as new areas of modeling arise steadily and existing models become increasingly fine grained and realistic. To be able to keep up with the growing demand for processing power the Cluster Computing paradigm[SSB+95] was introduced in the early 90's as a cheap alternative to full scale supercomputers due to the falling prices and increasing performance of commodity hardware such as CPU's, memory, storage and network. With the increasing capacity of the network infrastructures connecting the research facilities the idea of interconnecting the supercomputers at different research facilities into one infrastructure arose and was formulated as the Grid Computing paradigm[KF99]. The Grid vision was and still is a computing infrastructure where services are provided through a plug in the wall just like the electrical power grid. However, providing 230 Volt @ 50 Hz* is a simple protocol compared to the overall complexity of the devices capable of communicating by TCP/IP and therefore the current Grid systems are largely stuck with the first generation implementations which merely gathers geographically displaced supercomputers into one computing infrastructure.

While Grid Computing emerged the Berkeley University introduced Public Resource Computing (PRC) in the form of the SETI@home[ACK+02] project, which later turned into the BOINC[And04] framework. The PRC paradigm is a subset of the Grid paradigm focusing on the personal computing devices that are located outside the supercomputer facilities, which has been demonstrated to have a huge computation potential. However PRC computing is not real Grid Computing as the executed applications are tightly coupled to the executing resources opposed to Grid resources that are capable of executing arbitrary applications using arbitrary resources. A number of research projects have shown ways to combine PRC computing and Grid Computing,

*European standard for electricity

the first using BOINC resources within a Grid infrastructure[MBC04] and the latter by using sandboxing techniques[aBV06] to provide a generic execution environments to public resources enrolled in Grid Computing. While this is a step in the right direction the current state of Grid computing is still far from the original paradigm, namely providing services through a plug in the wall. What we are aiming at in this dissertation is bringing the Grid Computing paradigm one step closer to its goal by reaching a broader range of Grid resources in what we call non-computer devices. We define such devices as TCP/IP capable devices, which do not per default have the nature of a general purpose computer. Such devices include, but is not limited to, game consoles, sensor networks, laboratory equipment, telescopes and so forth.

1.2 Public Resource Computing

Public Resource Computing (PRC) is a subset of Grid Computing, and even though many PRC projects denote themselves as Grid Computing projects they are not. Conceptually there are two major differences between PRC computing and Grid Computing:

- Donating resources vs. sharing resources

  PRC systems are based on public resource donation

  Grid systems are based on resource sharing

- Generic vs. Fixed applications

  PRC resources are tightly coupled to a specific application

  Grid resources are capable of executing arbitrary applications

These two differences are tightly coupled, in order to share resources and submit arbitrary jobs to a computing infrastructure, the resources attached to the infrastructure must be capable of executing arbitrary applications from the infrastructures point of view. PRC projects have the advantages of a simpler framework, from the point of view of the framework developers, because they require the donors of the resources to install a client for each project in which they wish to enroll. Furthermore it's required that the target applications are re-written to comply with the PRC framework, which consumes time if applicable at all due to source code restrictions regarding availability and licenses. The BOINC project was introduced in 2004 to avoid installing a client for each project a given resource is enrolled in. This framework provides one resource manager for all BOINC compliant PRC projects with the possibilities to chose between them, but it doesn't solve the fundamental issues regarding portability and flexibility. While the first generation Grids were not suited for PRC computing[And04] the second generation Grids provide a perfect platform for public resources[aBV06] because they deliver the same functionality as the BOINC framework but without the need to modify

the target applications. According to the BOINC web page[†] it takes about three man-months to deploy a BOINC project from an existing application, in addition to that a midrange project server is needed which requires hosting, administration and maintenance. In contrast once the research is familiar with Grid infrastructures there will be a minimal overhead deploying new projects as the applications doesn't need to be ported and there is no investment in additional hardware.

1.3 Grid for the Public

Delivering services through a plug in the wall managed by a Grid infrastructure first of all requires services to deliver. PRC computing has shown that the public is willing to donate their personal computers to science without getting anything in return apart from a scoring list, which shows the amount of computation time delivered by each resource. The philosophy of Grid Computing is sharing resources instead of merely donating them, that is, resource donors get computation credits in return for their donations, which can either be used for future computations or possible sold to users in need of credits. However for this to work there is a bootstrap problem, if there are no services to take advantage of why should resource owners offer their services to a Grid infrastructure. But if nobody offers their resources, how are there going to be any services available? One of the major issues getting the Grid paradigm pushed outside the supercomputing facilities is the administration time and space overhead connecting resources to most of the current Grid middlewares. While this is acceptable in the supercomputing environment with associated administrators hired to customize the supercomputer towards the specific needs of the scientists using them, this is a major limitation when pushing the Grid to the public as a cumbersome and time consuming installation will keep out all but the most hard-core techies from the Grid environment. When pushing the Grid one step further namely to the non-computing devices a minimal Grid client footprint becomes increasingly important as such devices have minimalistic hardware setups compared to supercomputers and personal computers. Therefore a minimalistic Grid client footprint and easy connectivity to the infrastructure is vital getting the public to use Grid as a service infrastructure.

1.4 Non-computer Resources

Combining public resource computing and Grid Computing such that public resources can be utilized in a global Grid infrastructure was a major step towards fulfilling the Grid paradigm: "Providing services through a plug in the wall". However the paradigm

[†]http://boinc.berkeley.edu/trac/wiki/BoincIntro

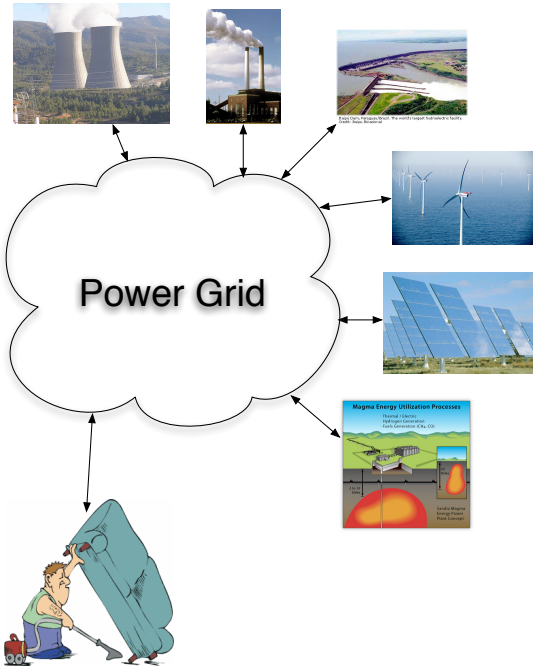


Figure 1.1: The variety of sources generating electricity in a power grid

is still not fulfilled, one of the next steps towards fulfilling the paradigm is reaching further out regarding resources. Like the power Grid utilizes various sources for generating electricity to the end users, figure1.1, computation Grids should utilize all sorts of computational devices to provide the services oriented infrastructure envisioned, figure1.2. However, this is not trivial as every resource class expansion increases the complexity of the overall system due to the heterogeneity of the connected resources. The traditional vector supercomputers are the most homogeneous architecture within the high performance computing (HPC) community. But due to the high prices on those installations and the falling prices and increasing performance of commodity hardware the cluster computer arose, which is merely a collection of homogeneous personal computers with a high-speed network interconnection. Even though the individual machines used for building cluster computers are homogeneous within each cluster setup new programming paradigms had to be developed as the communication went from communicating internally through the memory bus to externally using the network interconnection. For this purpose various systems such as PVM[Sun90] and MPI[For94] were developed. When expanding the resource pool from cluster computers to personal computers (PC) located outside the research facilities heterogeneity arises to a higher level regarding CPU architecture, processor power, memory, processing, disk, operating system, network bandwidth and latency. The group of PCs however are homogeneous vs the class of non-computer resources which includes every TCP/IP interconnected devices.

Figure 1.2: The variety of sources enrolled in a computation grid

The heterogeneity of non-computer devices will show in various ways:

- Processor Architecture

  Most PC's comply with the X86/X86_64 architecture, unlike the non-computer devices

- Memory

  non-computer devices generally have smaller amount of memory than PC's

- Disk

  non-computer devices often have very limited or no disks at all

- Operating system

  Many variations of embedded operating systems exist

The last and most important thing is that non-computer devices are often embedded devices without any possibility for extensions of the hardware. The device hierarchy/heterogeneity is sketched in figure1.3

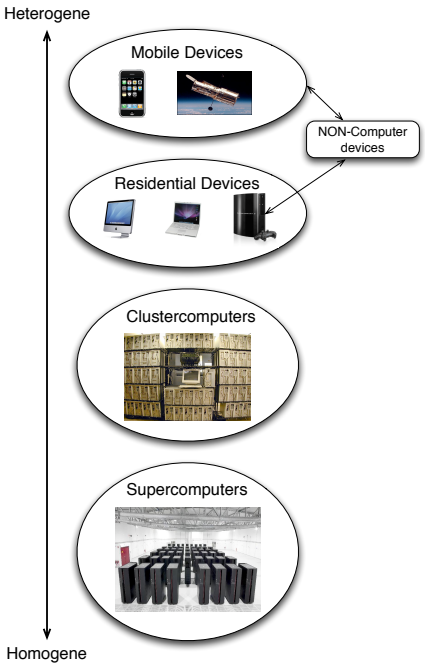


Figure 1.3: The homogeneous/heterogeneous relation of different devices

1.5 Challenges We Are Facing

The Grid paradigm[‡] is a very loose definition, one could define services as merely the ping and echo services and the Grid paradigm would be fulfilled. However in our perspective Grid services should be defined as the sum of all possible services that TCP/IP interfaced devices are capable of delivering. In reality most Grid implementations are still in their cradle in the sense that the resources they are able to handle are limited to supercomputer installations. Only a few systems have expanded to include personal computers located outside the supercomputer facilities in their resource pool, as this requires that the middleware is capable of handling the autonomous and heterogeneous nature of such resources. The goal of this project is to take the Grid resource pool one step further namely to the non-computer resources for computational services. We concentrate on computational services as these are the most hardware bound services and lie within the domain of existing Grid and cluster systems which make them comparable. In any case the frameworks and protocols needed to connect them to a Grid infrastructure will be applicable for all the services the devices are capable of delivering.

[‡]Providing services through a plug in the wall

1.6 The Grid Concept

The Grid computing concept[Fos02] defines a service oriented infrastructure on top of the Internet. The development of Grid middlewares has so far mainly focused on scientific computing, however a global Grid infrastructure should target all Internet capable devices. To achieve this goal several issues have to be considered.

1.6.1 Accessing Resources

As the majority of working Grid resources currently are machines for scientific computing, and the majority of scientific applications comply with a Unix like environment, the common way to access a Grid resource today is through batch jobs written and submitted as shell scripts. However with the goal of connection all capable Internet devices several different execution models have to be considered.

1.6.2 Interactive Shell

The first obvious extension to the default batch execution model is the interactive shell. This allow the Grid user to define which type of shell is needed combined with the information provided for normal batch type of jobs such as architecture, memory consumption, disk usage, runtime environments and the activation time of the shell. The interactive shell is a valuable tool when developing Grid applications, regarding compatibility between the development and execution environment. Beside that, the interactive shell provides the possibility to perform cross-platform computing as the client platform doesn't need to be the same as the shell provided by the Grid middleware.

Interactive Graphical Environments The interactive graphical environment is a natural extension to the interactive shell. This allow the Grid users to request individual graphical applications as well as full graphical desktops on various platforms and thereby provide full cross-platform desktop computing to the Grid end user.

1.6.3 Sensors and Laboratory Equipment

Connecting sensors and laboratory equipment to a Grid infrastructure would let scientists fetch information from those devices through the Grid. The scientists should be able to requests the status of a given sensor by sending a request through the Grid and the sensor should be able to notify the scientists about changes in the sensor environment.

1.6.4 Office Roaming Grids

The Grid infrastructure could also be used in an office environment. The resources here could be both printers and displays, as well as personnel. If one enters a meeting locality with e.g. a display wall[SBA07], it should be possible to acquire that display wall as a resource, by submitting a job to the Grid. This could increase the effectiveness of a meeting, as much time is wasted connecting projectors to individual laptops. Likewise printers could be acquired through the Grid middleware. Last but not least internal support personnel could be a Grid resource, by letting the Grid users acquire personnel through a Grid middleware, This would require some locality parameters of resources and clients in the Grid environment.

1.7 Grid Middlewares

Grid middlewares can be categorized into two types of Grids. global Grids and office Grids. Office Grids are designed to operate on a local network behind a firewall where global Grids are designed to work globally using the entire Internet. Companies and research institutions with sensible data tend to use office Grids to be sure their data and applications never leave the corporate computers. The first working Grid middleware was an office Grid named Condor[BLL91], this office Grid has later been extended with global Grid interconnection such that it can be used as a hybrid Grid. Several other office Grid implementations exists such as Datasynapse[misa], Sun Grid Engine[misc] (SGE) and Entropia[misb].

In this thesis we will concentrate on the global Grids, as the principles used in global Grids apply to office Grids as well. The first working global Grid was Globus[Fos06], this Grid middleware was build using existing software packages with additional Grid protocols for assembling supercomputer installations for simulations into one computation infrastructure. Globus is by many regarded as the defacto standard and many Grid middlewares such as as gLite[gLi] and NorduGRID ARC[EEE+03] are based on the Globus model. However these first generation Grids have limitations when it comes to connecting non-supercomputer resources outside the research facilities regarding usability, connectivity, scalability and administration. To overcome these issues a next generation of Grid middlewares have started to arise such as the XtreemOS[CFJ+08] and the Minimum intrusion Grid[KV05]. Both middlewares are being designed and implemented from scratch using the experience gained using the first generation middlewares to improve the overall functionality of the Grid system.

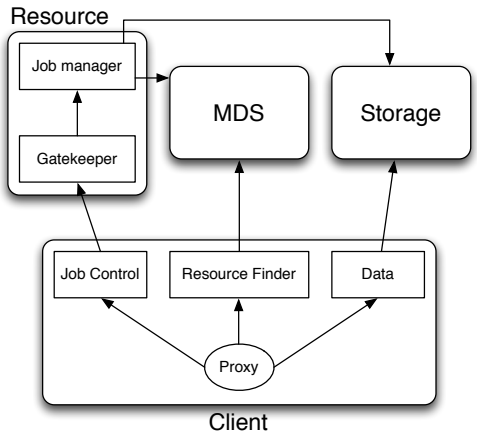


Figure 1.4: The abstract Globus+ model

1.7.1 First Generation Grids

The first generation Grid middlewares, hereafter denoted as Globus+, use a thin Grid infrastructure and a thick Client/Resource infrastructure. Figure 1.4 shows an abstract overview of the Globus+ design. The jobs executed in Globus+ environments are generic applications which means that the applications don't need to be rewritten when used in a Grid context. To submit a job, a user needs to generate a proxy certificate, as the resource accepting the job needs a method to identify the user towards the storage element when input/output files are received/sent to the storage element as well as the result of the job. Proxy certificates have several shortcomings, firstly a proxy certificate can't be revoked once issued, which means that a user with evil purposes can't be excluded from the Grid before an outstanding active proxy certificate expires. The default expiration time is 12 hours, but the user can specify as long as a year for the proxy certificate to expire. Secondly if the proxy certificate expires while a job, submitted with that proxy certificate, is executing, the delivery of the result to the storage element fails, and thereby the computation time is wasted.

1.7.2 Second Generation Grids

Minimum intrusion Grid (MiG) This Grid is is one example of a second generation. which is a stand-alone platform and doesn't inherit code from any earlier Grid middlewares. The philosophy behind the MiG system is to provide a Grid infrastructure that imposes as few requirements on both users and resources as possible to ensure a minimum amount of software to be installed and administrated by users and resource owners. A native resource only needs to create a local MiG user on the system and support inbound SSH[Ven07] and outbound HTTPS[mis02]. Alternatively a Sandboxed

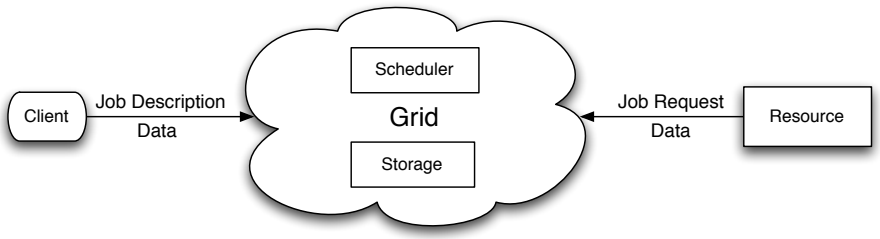


Figure 1.5: The abstract MiG model

resource[aBV06], which is a pull based model, only needs outbound HTTPS. The user is required to have an X.509[HFPS99] certificate which is signed by a source that is trusted by MiG, and a web browser that supports HTTP, HTTPS and X.509 certificates.

Because MiG keeps the Grid system disjoint from both users and resources, as shown in Figure 1.5, the Grid system appears as a centralized black box[Vin05] to both users and resources. This allows all middleware upgrades and troubleshooting to be executed locally within the Grid without any intervention from neither users nor resource administrators. The jobs executed on MiG are, like with the Globus+ Grids, generic applications, which don't need to be rewritten when executed through the Grid system. The MiG system doesn't use proxy certificates, as the resources identify themselves to the Grid system by their public SSH key and users identify themself by their X.509 certificate. An executing job identifies itself to the Grid system by a unique session-id which is created by the Grid system when the job is started and deleted when the job is finished. By using that session-id, the job is allowed to send/receive files to/from the Grid system.

XtreemOS is another example of a second generation Grid system. While it deals with with the same issues regarding first generation Grids as MiG it takes the opposite approach. While MiG imposes minimum intrusion on users and resources, XtreemOS imposes maximum intrusion of both users and resources as the operating system of both user and resource machines are replaces by a Grid operating system namely the XtreemOS. The system uses single sign on by using the standard login and password procedure known from traditional Unix systems, the user receives the credentials needed to transparently use the storage and resources in which he is associated without the need to re-authenticate.

1.7.3 Grid Scheduling

One of the most important tasks the Grid middleware has is distributing jobs among the connected resources in the most efficient way. The first generation of Grids leaves the task of finding a proper resource for a given job to the client submitting the job. The Grid middleware provides the client with a list of connected resources which the client can ask to take the job. All resources in the list are then asked for the waiting time for execution and the shortest one is chosen. This approach has a huge connection overhead and does not scale with respect to the number of nodes in a Grid system. Furthermore it has a built-in race condition if two clients try to submit a job at the same time, the two jobs might end up at the same resource. This means one of the jobs will be stuck waiting in the local queue at the resource to which it was submitted, even though there are other free Grid resources which could have taken the job. Different meta-schedules such as Gridway[HHML05] and Gridbus[dANV+05] have been made as add-ons for the first generation Grid middlewares, but they are not an integrated part of the Grid system.

The second generation Grids are designed with scheduling as a central part of the system. This means that the Grid system has complete track of the jobs executing, regarding which resources are executing them, as well as how long the jobs have been executing. Because the middleware is in full control of job submission jobs can be resubmitted to another resource if the resource executing them fail and they can be killed by the middleware if the upper time limit of the job is exceeded. The global Grid scheduling mechanism is occasionally denoted as a strong scheduler, because some of the first generation Grid systems denote their job placement as scheduling, due to the scheduling taking place at the local resource e.g. the queue system on a cluster.

1.7.4 Grid Scalability

The Grid goal is to provide a service infrastructure on top of the Internet and therefore scalability is a vital issue, because the amount of Internet devices worldwide is huge and growing widely, currently there is more than 1.8 billion users connected to the Internet[§]. As the first generation middlewares have no Grid scheduler the client contacts each connected resource every time it has a job to submit. While this works well for Grids with a small amount of users and resources, it doesn't scale to a full size global Grid with billions of users and resources. To give an example the circumference of the earth is about 40000 km which means that at every surface point there is at most half that distance to the point located furthest away namely 20000 km which means that the average distance between any user and resource in a global Grid is 10000 km, with a speed of light of $2.998 * 10^8$ m/s this gives an average round-trip latency of: $\frac{1.000*10^7 \text{ m}}{2.998*10^8 \text{ m/s}} * 2 = 0.0667$ *seconds* with just a thousand resources this is an average of

[§]http://www.Internetworldstats.com/stats.htm

60 seconds in latency discovering the resource best suited for the job giving the best possible network latency namely the speed of light. In this calculation we have not considered the fact that handling the request also takes time and the fact that the Grid system itself will become one large distributed denial of service attack on the connected resources, when enough users try to submit jobs concurrently.

The second generation of Grids ensures scalability with respect to users and resources by making the middleware resource aware such that the jobs can be submitted to the best fitted resources by the built-in scheduler without contacting each possible resource before submitting the job. The MiG system takes the approach of letting the resources contact the Grid system whenever it's ready to take a job. Upon a job request, the Grid system matches the requesting resource by the jobs currently in the waiting queue and provides the best fitted job[¶]. This means that no overhead connections are made[‖] as well as it's possible for the Grid system to perform load balancing and price auctioning of jobs. Last but not least, it means that the time for submitting a job to the Grid system is constant, as it's accepted and queued by the Grid middleware even if all current resources are occupied. This method makes it possible to keep the resource and client anonymous to each other which might be essential in some environments[MSVR07]. As all communication between user and resources goes through the Grid middleware, the Grid system in itself has to be a distributed system for this to scale.

1.7.5 Job Fault Tolerance

For a Grid system to be useful some level of fault tolerance has to be enforced. This applies both to the middleware functionality and the resources. The Grid middleware should be fault tolerant to a high extent, whereas the resources are more difficult to control from a middleware point of view as it's not always obvious if a job failure is due to an error within the job or within the resource executing it. A Grid job should not fail just because the resource executing it fails. However detecting automatically why a job fails is not trivial, so one has to decide a fault scheme for jobs. The first generation middlewares have no mechanism for handling failing resources due to the missing Grid scheduler, the second generation middlewares have the possibility to resubmit a job if the resource fails. There are two ways to determine if a job or resource fails without contacting the resource from the middleware, the first one is if the resource connects to the Grid middleware with the message that it's ready for a new job, and the Grid middleware knows the resource already has a job, the second is if there is a upper time-limit on a job and it's exceeded without the resource delivering an answer. Which

[¶]The best fitted jobs depends on the configuration of the scheduler

[‖]Provided there is always at least one job in the queue which fits the resource contacting the Grid middleware

decision to make when a job fails is up to the middleware to decide. The MiG approach is to resubmit the job X [**] times before failing the job, and only then notify the user that the job has failed.

1.7.6 Runtime Environments

An important part of a Grid system is dynamic runtime environments which allow the resource owners to specify which software packages are available at a given resource. The Grid user specifies which runtime environments are required by the job submitted to the Grid, and it's then up to the Grid middleware to match the resources with the job and find a resource which is capable of executing the submitted job. As an extension to runtime environments, the MiG group is currently expanding the middleware with zero install[††]. This allows a Grid job to download and install packages needed by the Grid jobs in pure user-space, and thereby the amount of resources capable of executing any given job is increased significantly as the Grid job is no longer dependent of which software packages the resources is offering. The effect is that the resources become gradually more software homogeneous as these packages are made available for the different resource hardware architectures.

1.7.7 Virtual Organizations

To provide a service oriented infrastructure on top of the Internet, where one is able to share all kind of resources, it's essential for any Grid system to be able to organize users and resources into Virtual Organizations[FKT01] (VOs). Virtual Organizations can assure resource owners that the resources they provide for the Grid are only used for the projects which lie within the VOs they accept for the resource. From the user perspective it's assured that the jobs they submit are processed by a resource within the specified VOs. This can be a critical issue if the job contains sensible information, in which the job submitter is not willing or allowed to expose to non-trusted resources or users.

Different Grid systems implement VOs in different ways, the first generation Grid middlewares use Virtual Organization Membership Service VOMS[ACC+03]. This is implemented as a service separate from the Grid middleware itself, and is administrated by the owners of the virtual organizations. It works by keeping a database of user/resource mappings. Each organization has to set up it's own VO server and administrate the user accounts and user/resource mappings on that particular server. When a user wants to use resources within a certain VO the VOMS server is contacted and returns

[**]The value of X is set by the Grid administrators, but it's possible to specify it individually for each job

[††]http://zero-install.sourceforge.net

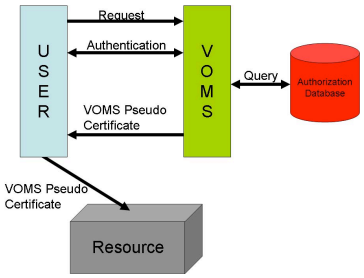


Figure 1.6: The abstract VOMS model

a proxy certificate, if the user is verified as a member of the given VO. This proxy certificate is then used when submitting jobs to the Grid. When a user contacts a Grid resource using a VO proxy certificate, the resource contacts the VOMS servers, which it has been configured to be a part of, and checks whether it's allowed to take the proposed job. If the resource is allowed to take the job it takes it, otherwise it rejects it, and the client has to try another resource. An overview of VOMS is shown in figure 1.6. This method has several shortcomings, firstly the VOMS proxy certificates can't be revoked, which means that the user is allowed to submit jobs to the VO as long as the proxy certificate is active. This is an option the user specifies when generating the certificate. Secondly the VO proxy certificates can't be combined, which means that the user has to keep track of several certificates and manually try to submit his jobs to several VOs if he wishes to execute his job in either VO X or VO Y, which is very inflexible. Thirdly the manual administration of user accounts, is unnecessary as the Grid system already has this information about it's users.

The second generation Grids implements Virtual Organizations as central part of the Grid systems. This is possible due to the global Grid scheduler that has information of which VOs each user and resource are affiliated with. Thereby there is no need for additional authentication and verification when submitting jobs, as the scheduler knows prior to submitting the jobs which resources should be considered for the jobs submitted by any given user. The MiG system defines virtual organizations as Virtual Grids VGrids[KB06]. An overview is shown in figure 1.7.

VGrids eliminate the redundancy and possible extra point of failure of verifying VOs through a third party system (VOMS server), as well as the administrative overhead of managing an extra user database at the VOMS server. Furthermore VGrid resources have significantly lower administration overhead, as VOMS resources need to be configured to communicate with the right VOMS servers to verify which VOs they are part of. This is eliminated in MiG as the user/resource mapping is a part of the MiG scheduler's knowledge of the system, and thereby a resource is never granted a job which is not addressed for one of its specified VGrids. VGrids are administrated by the Grid users and can be created dynamically. Resource owners can specify through the

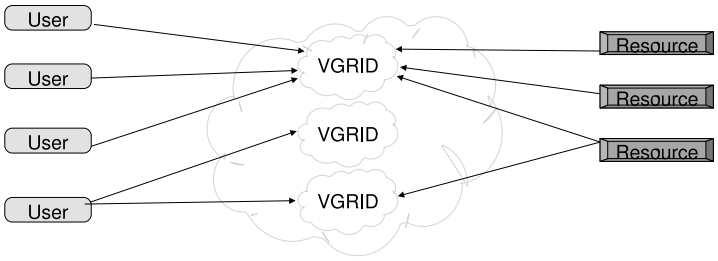


Figure 1.7: The abstract MiG VGrids model

MiG system which VGrids their resource should be a part of. This has to be done with permission from the VGrid owner for mutual agreement about participation. VGrids can have sub-VGrids and thereby a whole hierarchy of VGrids can be designed. A member of a VGrid has access to all sub-VGrids, but not the other way around.

1.7.8 Security: Authentication and Anonymity

The first generation Grid middlewares uses proxy certificates to verify themselves towards the Grid resources. The second generation Grid middlewares uses various approaches to avoid the usage of proxy certificates.

In the MiG system anonymity is a vital design issue and thereby the users and resources should never communicate directly. Users authenticate themselves to the MiG system by a x509 certificate, signed by a CA accepted by the MiG administration group. All communication goes through the MiG system, and thereby users and resources are never in direct contact. Instead of proxy certificates the MiG system uses session-id's for authentication between the resources and the Grid system. Each session-id is unique and active as long as the job assigned to the resource is active, thereby the Grid system can authenticate the resource uniquely by the job session-id.

1.7.9 Grid Storage

To make Grid systems useful one has to have a place to put files, this is referred to as a Grid storage element. The first generation Grid systems use the GridFTP protocol[LT01] which is a Grid extension to the traditional FTP protocol[Gie78]. This means that users have to explicitly consider where to store their files, and make sure to specify the right GridFTP server when describing their Grid jobs.

The second generation Grid system takes a different approach by letting the Grid users access files according to the POSIX standard. This means that the Grid files are accessed by the Grid users and their jobs transparently without explicitly stating where the file is located.

In the MiG system each user has an associated home directory like the standard *NIX operating systems. Files for the different VGrids in which the user is enrolled appears as subdirectories, with the name of the VGrid, within the home directory of the MiG user. The MiG home directory is mounted on the local filesystem using FUSE[iU] and the different storage elements the system uses internally is transparent to the user.

When a user submits a job to the Grid he specifies which inputfiles are needed for the job, all relative to the root of the users home directory. This directory structure is then mapped to the resource and the input files are accessed relative to that structure by the job executables. This combined with the FUSE file system approach enables users to make a test-run of their Grid jobs locally before submitting them to the Grid, thus eliminating the problem with specifying inputfiles correctly which every Grid user struggles with every time a new project is deployed to Grid.

One major shortcoming of the Grid computing model regarding storage is when the input files are huge (in order of GB or TB), but each job only uses a small part of that inputfile. In many of these cases, the time transferring the data from the storage nodes to the computation nodes largely exceeds the computation time of the job. To solve this problem the MiG system has the Transparent Remote File Access[AV05] which is a user level remote file access mechanism capable of remotely accessing only the parts of the files needed by the executing Grid application. This is done at runtime and transparent to the application writer.

1.7.10 Connecting Resources to the Grid

A limiting factor for Grid computing to reach the public is the administrative task of connecting a resource. Most Grid systems require a huge amount of software to be installed on a potential Grid resource and several ports have to be opened in the firewall protecting these resources.

The requirements for connecting a resource to the MiG system on the opposite is just inbound SSH and outbound HTTPS, a Unix user account and cURL[cUR] for the native resources. Still this has shown to be an obstacle for access to the large amount of resources located outside the supercomputer facilities. To gain access to these resources, which mostly use the windows operating system, a sandboxed model executing a Linux execution environment with the appropriate Grid client software inside a Virtual Machine[aBV06]. This model uses a full pull model, requiring only outbound HTTPS.

1.7.11 Choosing a Grid Middleware

As non-computer devices in this thesis are defined as TCP/IP capable devices, a Grid middleware which supports, the broadest range of resource types has to be chosen. The following requirements must be met to cover the broadest range of devices.

- Minimal Software footprint

  Minimal disk consumption by Grid middleware on the resources

- Minimal Memory footprint

  Minimal middleware memory consumption on the resources

- NAT compliance

  Resources are possibly behind a router or firewall

- Fault tolerance

  Resources may join and leave frequently

The office Grids are excluded per default as we want to operate globally. This leaves the choice between the first generation middlewares and the second generation middlewares. The first generation middlewares has a much larger software and memory footprint on the resources than the first generation middlewares, they are not NAT compliant and they do not have any degree of fault tolerance due to the lack of a global Grid scheduler. Based on these observations the choice was between the Minimum intrusion Grid and the XtreemOS which is currently the only two operating second generation Grid middlewares. As the XtreemOS middleware imposes maximum intrusion on both resources and clients due to the replacement of the entire operating system, the Minimum intrusion Grid was chosen as it operates purely in user-space both on the resource and the user side.

1.8 Presented Work

In the last section we introduced different types of Grid middlewares and introduced the requirements put on the middlewares in order to be able to handle non-computer resources. From the Grid middleware perspective the main issues regarding non-computer resources are the heterogeneity both hardware and operating system wise, the minimal hardware setup and the unreliable nature of such devices. The unreliability is due to the fact that computer resources which are not owned by a supercomputing centers tend to join and leave the Grid infrastructure frequently compared to HPC installations, as scientific computing is not the main purpose for non-computer devices. In this section we present the different approaches taken in this dissertation to use non-computer devices for scientific Grid Computing.

1.8.1 The One-Click Grid Resource Model

The initial research done during this dissertation was to make a framework that targets as many of the immediate problems connecting lightweight Grid resources as possible. The considerations taken into account are:

- Minimal Grid software foot-print
- Hardware independence
- OS independence
- NAT firewall independence
- Checkpointing support
- Remote File Access support
- Sandboxing

The model is described in detail in chapter 2 and presents an minimal resource approach requiring merely a Java virtual machine and support for outgoing HTTPS. This framework targets all Java capable devices and works without installing any Grid specific software. By using the Java virtual machine, Sandboxing, Hardware and OS independence is provided to the same extent as the virtual machine. The framework supports checkpointing and remote file access through the Grid infrastructure. The limitations of the framework is that the applications supported by the framework must be written in Java and use special methods for file access and checkpointing. While Java may not be the obvious choice for HPC this framework represents a valuable proof of concept regarding the minimal requirements for connecting resources to a Grid framework and has proved valuable when introducing Grid Computing to new users. The framework is fully functional and is deployed as part of the Minimum intrusion Grid.

1.8.2 The PS3^{TM} Grid Resource Model

With the release of the SONY PS3^{TM} it was announced that it would have native support for the FOLDING@home project. Due to its low price and powerful Cell-BE processor and NVIDIA graphics card it is an attractive non-computer devices for general scientific computing and not just FOLDING@home.

To Grid-enable the PS3™ platform the following requirements need to be met:

- Minimal Grid software footprint
- Sandboxing
- NAT firewall compliance
- Remote File Access support

Grid enabling the PS3™ is presented in chapter3. This model presents the LIVECD which when loaded by the PS3™ boots into a sandboxed Linux environment where no access can be obtained to the local disk. The graphics card can not be utilized for computation by this environment, because it's protected by the PS3™ hypervisor, however the graphics card memory is utilized for temporary storage and remote file access can be obtained through the remote file access library. The framework is fully functional and is deployed as a part of the Minimum intrusion Grid.

1.8.3 Application Porting and Tuning on The Cell-BE Processor

Porting existing sequential scientific applications to the architectures of the new non-computing devices such as PS3™ with its Cell-BE processor is quite a challenge if the full potential of the resource is to be harnessed. Chapter 4 represents the porting of an X86 application to the Cell-BE architecture and encounters the challenges and considerations to make when programming towards the new architectures.

1.8.4 The Remote Memory Library

While the initial research done represents two different models for connecting non-computer devices to Grid infrastructures we have still not found a generic solution to connecting all non-computer devices to Grid infrastructures. To address this problem we decided to take two steps back and look at what we learned from the initial work. One of the greatest challenges within Grid systems and specially the non-computer devices is the heterogeneity of the resources. So how do we solve this problem ?

Sandboxing is a major topic within Grid Computing outside the supercomputing centers as it provides an isolated execution environment which can not compromise the hosting device. Furthermore sandboxes can provide virtualization of the architectures of the executing devices, such that the Grid application programmer doesn't need to worry about the architecture of the devices executing his application. Currently we have transparent virtualization of CPU architectures through the scientific byte code machine[AB08] virtualization of storage through the remote file access library[AV05] but we have no way of virtualizing memory transparently through the Grid infrastructure. To solve this issue we set out to make a User-Level remote swap library for Grid

resources. This is represented in the chapter "Remote Memory" which is the main research subject in this dissertation. The goal is to provide memory through the Grid infrastructure to resources which are lacking physical memory to fulfill the memory requirements of a given job. This closes the virtualization circle for HPC non-computer resources as well as for Public resources in general. The initial framework has been designed, implemented, tested and proved fully functional in a real Grid scenario, as presented in chapter5.

In addition to the initial remote memory framework prediction-based prefetching has been added in collaboration with the researchers at Barcelona Supercomputing Center BSC, who have made an event based predictor for predicting file usage in a Grid framework. It has been adapted to work with the remote memory framework to predict which memory blocks are likely to be used in the near feature, based on the memory reference sequence seen in the past.

1.9 Contributions

The research presented in this dissertation contributes with the design, implementation and experimentation validation of the framework: "The One-Click Grid Resource Model" for Grid enabling all Java capable TCP/IP devices, followed by the design, implementation and experimental validation of the framework: "The PS3™ Grid Resource Model" for Grid enabling the Playstation® 3 game console. These two initial frameworks are followed by defining requirements for porting existing sequential applications to the new vector-based architectures such as the Cell-BE and experimental comparison between the performance of an application ported from the X86 architecture to the Cell-BE architecture. The final contribution is the design, implementation and experimental validation of a Grid memory framework that makes Grid resources memory homogeneous by providing memory transparently to limited resources through the Grid infrastructure using prediction-based memory prefetching.

1.10 Publications

The work presented in this dissertation has resulted in the following published papers:

- [MB10] Martin Rehr and Brian Vinter, *The User-level Remote Swap Library*, Accepted for The 12th IEEE International Conference on High Performance Computing and Communications 2010, Melbourne, Australia, To Appear
- [RV09] Martin Rehr and Brian Vinter, *Application Porting and Tuning on the Cell-BE Processor*, 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing 2008, Trondheim, Norway

- [RB08] Martin Rehr and Brian Vinter, *The PS3 Grid-Resource Model*, The 2008 International Conference on Grid Computing and Applications, Las Vegas, USA

- [aBV07] Martin Rehr and Brian Vinter, *The One-Click Grid-Resource Model*, High Performance Computing Conference 2007, Houston, USA

Additional the following book chapters have been published:

- [ARV09] Rasmus Andersen, Martin Rehr, and Brian Vinter, *Cycle-Scavenging in Grid Computing*, Grid Technology and Applications: Recent Developments

- [VAR+09] Brian Vinter, Rasmus Andersen, Martin Rehr, et al., *Towards a Robust and Reliable Grid Middleware*, Grid Technology and Applications: Recent Developments

Chapter 2

The One-Click Grid Resource Model

The "One-Click Grid Resource Model" started out as a proof of concept. Opposed to the several hundred of megabytes required to get a Globus+ resource running, what would be the minimum requirements for a MiG resource. Would it be possible to create and run a Grid resource just by visiting a URL in a browser? For this to work several requirements has to be met by the Grid middleware. Firstly the Grid middleware must support a full pull model*, since most machines targeted by this model will be located behind a NATed firewall. Secondly the Grid middleware must support an automatic creation and re-authentication of the connected resources, to avoid human administration. The full length paper published on the the One-Click Grid resource model can be found in appendix A.

2.1 The MiG Scheduler

The MiG scheduler plays an important role for the research described in this chapter, this section will give an overview of how the MiG scheduler operates. When a resource is ready to execute a job it sends a job request to the Grid system. The Grid system has a queue of jobs submitted by the Grid users, and upon a job request the scheduler traverses the queue to find the job which matches the resource who send the job request. The requesting resource and the jobs in queue are matched by the following parameters

- Architecture
- CPU count
- Node count
- Maximum CPU time

*All communication is initiated by the resource

- Maximum Memory usage
- Maximum disk usage
- Runtime environments
- VGrid settings

All jobs are filtered against the above criteria, whereafter the job to be executed is scheduled by one of the following schemes

- First fit

  Find the first job that can be run on the resource

- Best fit

  Find the job which has the highest score regarding the match between job and resource

- Fair fit

  This is the "Best fit" Scheduler modified to include job age

- Max throughput

  Select matching job with lowest CPU time

- Random

  Find the jobs that fit the resource and choose one of them at random

- FIFO

  Like "First fit", but takes job arrival time into account

When the right job has been found for the requesting resource, a job description file is generated by the Grid system and sent to the resource, which then retrieves the appropriate input files, the executables, and starts executing the job. The scheduler marks the job as executing and gives it a start time stamp. Thereby the Grid middleware has the possibility to check if executing jobs exceeds their maximum execution time. If that happens the job is re-queued to wait for another resource to pick it up. The job can be re-queued X times before the job is marked as failed[†]. This ensures some degree of fault tolerance, which offers the possibility for resources to join and leave the Grid without discarding the jobs running on the leaving resources. If 100% fault tolerance

[†]The value of X is set by the Grid administrators, but it's possible to specified it individually for each job

has to be guaranteed, one has to sacrifice the possibility of failing jobs which continuously exceeds their upper time limit. This is due to the fact that it's not possible to detect whether the exceedance of the upper time limit is due to a faulty job, a job that executes too long compared to the given upper time limit, or a job which constantly is assigned a resource which leaves the Grid system before it's finished.

2.2 The One-Click VMware Player Model

The first version of the The One-Click Grid resource model uses the Java Applet[javb] and VMware Player[vmw] as a virtual machine with a Linux Sandboxed Grid environment as execution environment. This works by a browser initiating a connection to the MiG system through HTTPS. If the resource hasn't been connected before, the MiG server creates a new resource configuration file and generates a 32 char long random key, which is given back to the resource used for further identification. When the resource has been created, or recognized, the MiG system responses with the HTML description of the MiG Java Applet. When the Applet is loaded into the browser, the files needed for the MiG-VMplayer image are downloaded and the VMplayer is started with the downloaded image. When the browser leaves the MiG resource site, etc. when its closed, refreshed, or the forward/back button is pushed, the execution of the VMplayer is stopped and the files downloaded are removed, leaving the local resource in the same state as before the MiG server was accessed. Alternatively one could leave the VMplayer open, when leaving the MiG resource site, and make it up to the user to stop the VMplayer manually. The advantage of leaving the VMplayer open is that the currently running image, and Grid job, will not be killed, if the user accidentally refreshes, closes, or hits the forward/back button of the browser. However, closing the VMplayer and cleaning up all the files gives the most transparent behavior, as the user will expect the execution to stop whenever he leaves the "One-Click" site. Whether the MiG image files should be left on the resource, to avoid downloading the files every time the resource is started can be considered, but a method for time stamping the images, which makes it possible to detect whether the local image is up-to-date with the server image should be introduced.

This model has the advantage of providing a full Sandboxed Linux execution environment for the Grid system through a virtual machine, which means that existing Linux Grid jobs can be processed by this framework without any modification. This model has several drawbacks, firstly it depends on third party software, The VMware Player, to be installed as it's not a part of a standard desktop. Secondly it breaks the Java Applet Security model[java] as the Linux image has to be downloaded and saved locally to the disk of the host system and the Java Applet has to start a native process, the VMware Player. This can be circumvented by packing the Applet into a jar file and signing it by the Java jar-signer tool. Still, when the jar file is signed, the default

browser setting is to alert the user that the default security setting is violated, and ask if the user permits this. This has two downsides, firstly it's never good promotion for a web-based system, that the first thing the user is met with is a security alert telling him that the system is about to compromise the security of his system. Secondly if the user is convinced to say yes, the Applet has write permissions to the hard-drive and execution permissions of arbitrary code on the system. This can result in bad behavior either by accident, bugs in the framework, or if the Applet is compromised by an evil third part. The default Java Applet security model prevents the Applet to download code from any other machine than the one where the Applet was loaded, but as this is URL based, it can be compromised by an evil attender by altering the DNS information and thereby download arbitrary code for execution.

2.3 The Java One-Click Model

To address the security issues introduced by the VMware Player model, and the fact that the installation of third party software is not minimum intrusion, a different approach to the One-Click model was taken.

The primary goal was to create a model which does not violate the Applet Security Model, ASM, enabling the possibility of using the Applet as a trusted sandboxed execution environment for unknown Java bytecode (The Grid jobs). The Java virtual machine allows us to load and execute Java bytecode located on a remote server directly without writing it to local disk. Thereby it's possible to load and execute remote Java bytecode within a Java Applet without violating the ASM, as long as the Applet and the bytecode are located at the same server.

The Java Applet technology makes it is possible to turn a web browser into a MiG sandbox without installing any additional software. This is done automatically when the user accesses "MiG One-Click"[‡], which loads an Applet into the web browser. This Applet works as a Grid resource script and is responsible for requesting pending jobs, retrieving and executing granted jobs, and delivering the results of the executed jobs to the MiG server.

To make the Applet work as a resource script, several issues must be addressed. First of all ASM disallows local disk access. Because of this both executables and input/output files must be accessed directly at the Grid storage. Secondly, only executables that are located at the same server as the initial Applet are permitted to be loaded dynamically. Thirdly, text output of the Applet is written to the web browsers Java console and is not accessible by the Grid middleware.

When the Applet is granted a job by the MiG server, it retrieves a specification of the job which specifies executables and input/output files. The Applet then loads the

[‡]The URL accessed to activate the web browser as a sandboxed MiG Java resource is called "MiG One-Click", as it requires one click to activate it.

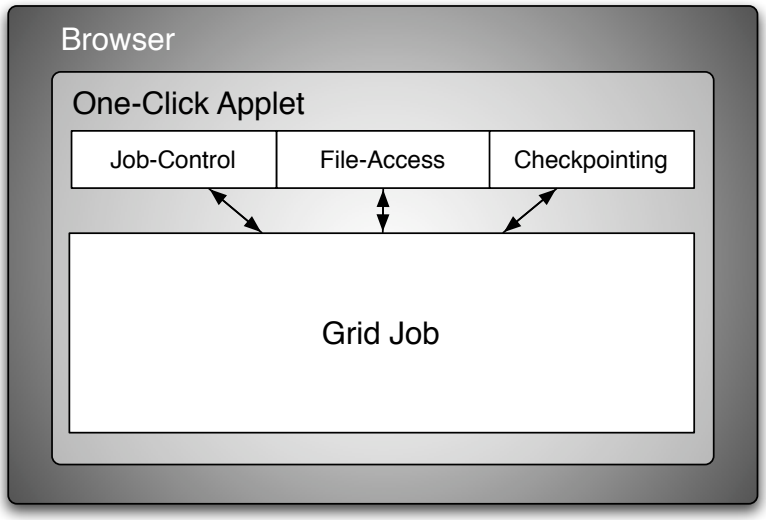


Figure 2.1: The structure of a One-Click job

executable from the Grid, this is made possible by the MiG server which sets up a URL from the same site as the resource Applet was originally loaded which points to the location of the executables. This allows unknown executables to be loaded and comply with the ASM restrictions on loading executables. Figure 2.1 shows the structure of a One-Click job. Executables that are targeted for the MiG One-Click model must comply with a special MiG One-Click framework, which defines special methods for writing stdout and stderr of the application to the MiG system[§]. Normally the stdout and stderr of the executing job is piped to a file in the MiG system, but a Java Applet, by default, writes the stdout and stderr to the web browsers Java console. It has not been possible to intercept this native output path. Input and output files that are specified in the job description must be accessed directly at the Grid storage unit since the ASM rules prohibits local file access. To address this issue the MiG One-Click framework provides file access methods that transparently provide remote access to the needed files. Note that the MiG system requires input files and executables to be uploaded to the MiG server before job submission which ensures that the files are available at the Grid storage unit.

In addition to the browser Applet, a Java console version of the MiG resource has been developed to enable the possibility of retrieving and executing MiG One-Click jobs as a background process. This requires only a Java virtual machine. To obtain the desired security model, a customized Java security policy is used, which provides the same restrictions as the ASM.

[§]The result of a MiG job is the stdout/stderr and the return code of the application that is executed.

2.4 Remote File Access

The One-Click executing framework that was introduced above also provides transparent remote file access to the jobs that are executed. The MiG storage server supports partial reads and writes, through HTTPS, of any file that is associated with a job. When the resource Applet accesses files that are associated with a job, a local buffer is used to store the parts of the file that are being accessed. If a file position which points outside the local buffer is accessed, the MiG server is contacted through HTTPS, and the buffer is written to the MiG server if the file is opened in write mode. The next block of data is then fetched from the server and stored into the buffer and finally the operation returns to the user application. The size of the buffer is dynamically adjusted to utilize the previously observed bandwidth optimally.

2.4.1 Block Size Estimation

To achieve the optimal bandwidth for remote file access it is necessary to find the optimal block size for transfers to and from the server. In this case the optimal block size is a trade off between latency and bandwidth. We want to transfer as large a block as possible without excessive latency increment since the chance of transferring data that will not be used increases with the block size.

We define the optimal block size bs_{opt} as the largest block where a doubling of the block size does not double the time to transfer it. This can be expressed the following way:

$$t(x) * 2 > t(x * 2) \qquad \forall x < bs_{opt} \tag{2.4.1}$$

$$t(x) * 2 < t(x * 2) \qquad \forall x > bs_{opt} \tag{2.4.2}$$

$$t(x) \ = time\ to\ transfer\ block\ of\ size\ x$$

We do not want block sizes below bs_{opt} as the time t used to transfer a block of size x is less than doubled when the block size is doubled. On the other hand we don't want 'too large' block sizes as we do not know if the retrieved data is going to be used or discarded due to a seek operation beyond the end of the local buffer.

As the One-Click resources can be placed at any sort of connection, and the bandwidth of the connection thus may differ greatly from one resource to another, it is not possible to use a fixed block size and reach a good ratio between bandwidth and latency at an arbitrary type of connection.

The simplest approach would be to use a fixed bs_{opt} based on empirical tests on the most common connections.

A less trivial, but still simple, approach would be to measure the time it takes to connect to the server and then choose a block size which ensures the transfer time of

that block to be a factor of x larger than the time to connect, to make sure that the connection overhead does not exceed the time of the actual data transfer.

The chosen approach is to estimate bs_{opt} from the time spent transferring block $x-1$ with the time of transferring block x, starting with an initial small[¶] block size bs_0 and then doubling the block size until a predefined cutoff ratio CR is reached. After each data transfer the bandwidth bw_x is calculated and compared to the bandwidth of the previous transfer bw_{x-1}. If the ratio is larger than the predefined CR:

$$\frac{bw_x}{bw_{x-1}} > CR \tag{2.4.3}$$

then the block size is doubled:

$$bs_{x+1} = bs_x * 2 \tag{2.4.4}$$

As the block size is doubled in each step the theoretical CR to achieve bs_{opt} should be 2, since there is no incentive to increase block size once the latency grows linearly with the size of the data that is transferred. However in reality, one needs to get a CR below 2 to achieve bs_{opt}. This is due to the fact that all used block sizes are powers of 2, and one cannot rely on the optimal block size to match a power of 2.

Therefore to make sure to get a block size above bs_{opt} you need a lower CR. Empirical tests showed that a CR about 1.65 yields good results, see section 2.6

Additional extensions include adapting to the frequency of random seeks in the estimation of the CR. A large amount of random seeks to data placed outside the range of the current buffer will cause new blocks to be retrieved in each seek. Therefore the block size should be lowered in those cases to minimize the latency of each seek.

2.5 Checkpointing

One-Click resources will join and leave the Grid dynamically, which means that jobs with large running time have a high probability of being terminated before they finish their execution. To avoid wasting already spent CPU-cycles a checkpointing mechanism is build into the Applet framework. Two types of checkpointing have been considered for inclusion, transparent checkpointing and semi-transparent checkpointing.

2.5.1 Transparent Checkpointing

All to the author known transparent checkpoint mechanisms provided to work with Java, require the JVM to be replacement or access to the /proc file system on Linux/Unix operating system variants, as the default JVM does not support storing program counter

[¶]An initial small block size gives a good result as many file accesses applies to small text files such as configuration files.

and stack frame. Since our goal is to use a web browser with the Java Applet as a Grid resource neither of those solutions are satisfactory, since both the replacement of the JVM and access to the /proc file system violates the Java Applet security model. Furthermore most One-Click resources will be running the Windows operating system which do not support the /proc file system.

2.5.2 Semi-transparent Checkpointing

Since transparent checkpointing is not applicable to the One-Click model, we went on to investigate what we call semi-transparent checkpointing. Semi-transparent checkpointing covers that the One-Click framework provides a checkpoint method for doing the actual checkpoint, but the application programmer is still responsible for calling the checkpoint method when the application is in a checkpoint safe state.

The checkpoint method stores the running Java object on the MiG server through HTTPS. Since it can only store the object state, and not stack information and program counters, the programmer is responsible for calling the checkpoint method at a point in the application, where the current state of the execution may be restored from the object state only. To restart a previously checkpointed job, the resource Applet framework first discovers that a checkpoint exists and then loads the stored object.

To ensure file consistency as part of the checkpoint, the framework also supports checkpointing of modified files, which is done automatically without involving the application writer. Open files are checkpointed if the job object includes a reference to the file.

2.6 Experiments

To test the One-Click model we established a controlled test scenario. Eight identical Pentium 4, 2.4 GHz machines with 512 MB ram were used for tests.

2.6.1 One-Click as a Concept

The test application used, is an exhaustive algorithm for folding proteins written in Java. This was changed to comply with the Applet framework.

A protein sequence of length 26 was folded on one machine, which resulted in a total execution time of 2 hours, 45 minutes and 33 seconds. The search space of the protein was then divided into 50 different subspaces using standard divide and conqueror techniques. The 50 different search spaces were submitted as jobs to the Grid, which provides an average of 6 jobs per execution machine and 2 extra jobs to prevent balanced execution. The search spaces on their own also provide unbalanced execution

as the valid protein configurations vary from one search space to another and thus results in unbalanced execution times. The experiment was made without checkpointing the application. The execution of the 50 jobs completed in 29 minutes and 8 seconds, a speedup of 5.7 for 8 machines. While this result would be considered bad in a cluster context it is quite useful in a Grid environment.

To test the total overhead of the model, a set of 1000 empty jobs was submitted to the Grid with only one One-Click execution resource connected. The 1000 jobs completed in 19935 seconds, which translates to an overhead of approximately 20 seconds per job.

2.7 One-Click Summary

The One-Click resource model presents a framework for connecting all Java capable TCP/IP devices into a Grid infrastructure without the need for the resource administrator to install any additional software beside the Java Virtual Machine. If the device has a web-browser supporting Applet, the resource is started by accessing a URL which downloads a Java Applet and sets up the Grid resource framework needed to retrieve jobs, execute them, and deliver the results. This can also be done from a console by manually downloading the resource jar file and security policy for the MiG server.

The contribution of this model is to provide an easy way of connecting Java resources to the Grid environment, as well as stretching how little effort is needed to connect a potential resource to the Grid, if one has the right framework. This model has a great educational value when teaching Grid systems, as students can easily generate their own VGrids for testing purposes, connect several One-Click resources and write simple Grid programs with a minimal effort.

At the cost of easy usage the "One-Click" model has several limitations, these are stretched below

- Applications must be written in Java
- Applications must comply with the One-Click framework
- Applications must apply to ASM
- The total memory usage is limited to 64 MB including the Grid framework
- Special methods must be used to catch output
- Special methods must be used for file access

These shortcomings are a compromise between security and flexibility, as the One-Click framework is designed to work inside a Java Applet, and we don't want to break with

ASM, there is no way around them. This approach is chosen as we want to guarantee that the host system is not compromised by the Grid middleware or jobs[‖]

[‖]One can never guaranty 100% security, but if this system breaks security, it's the security of ASM that is broken, and thereby all machines using Applets is at risk, and the assumption is that this will be quickly discovered and fixed by the Java vendors

Chapter 3

The PS3^{TM} Grid Resource Model

The release of the Playstation® 3 (PS3^{TM}) introduced a whole new standard for game consoles with its Cell Broadband Engine[CRDI05], Cell BE processor. This processor is a heterogeneous multicore processor with a theoretical peak performance of 153,6 GFLOPS in single precision and 10.98 GFLOPS in double precision*. According to Sony more than 9 million PS3^{TM} was sold worldwide[son] at the end of 2007. If one could combine them all in a Grid infrastructure this would sum up to a theoretical peak performance of 1382.4 peta-FLOPS in single precision and 98.82 peta-FLOPS in double precision. The main core of the Cell BE is an IBM 64 bit power processor, PPC64, which is the core running the operating system. By using a PPC64 core as primary core, the Cell processor can be used out of the box, due to the fact that operating systems for the PPC64 architecture are already available. The full length paper published on the the PS3^{TM} Grid resource model can be found in appendix C.

3.1 About the PS3^{TM} Game Console

Contrary to other game consoles, the PS3^{TM} officially supports alternative operating systems besides the default Sony Game OS. Even though other game consoles can be modified to boot alternative operating systems, this requires either an exploit of the default system or a replacement of the BIOS. Replacing the BIOS is intrusion on the highest level, expensive at a large volume and not usable beyond the academic perimeter. Security exploits are most likely to be patched within the next firmware update, which makes this solution unusable in any scenario. Beside the difficulties in modifying other game consoles towards our purposes, the processors used by the game consoles currently on the market, except for the PS3^{TM}, are not of any interest for scientific computing.

*The PS3^{TM} Cell has 6 SPE's available for applications. Each SPE is running at 3.2 GHz and capable of performing 25.6 GFLOPS in single precision and 1.83 GFLOPS in double precision.

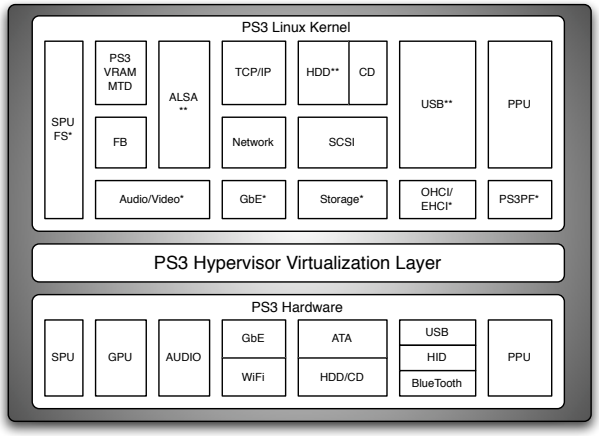


Figure 3.1: An overview of the PS3^{TM} Hypervisor structure for the PS3-LIVECD

The fact that the PS3^{TM} is low priced from a HPC point of view, equipped with a high performance vector processor, and supports alternative operating systems, makes it interesting as a Grid resource. Third party operating systems work on top of the Sony GameOS, which acts as a hypervisor for the guest operating system. See figure 3.1. The hypervisor controls which hardware components are accessible from the guest operating system. Unfortunately the GPU is not accessible by guest operating systems[†], which is a pity, as it in itself is a powerful vector computational unit with a theoretical peak performance of 1.8 tera-FLOPS in single precession. However 252 MB of the 256 MB GDDR3 ram located on the graphics card can be accessed through the hypervisor, The hypervisor reserves 32 MB of main memory and 1 of the 7 SPE's available in the PS3^{TM} version of the Cell processor[‡]. This leaves 6 SPE's and 224 MB of main memory for guest operating systems.

3.2 The PS3^{TM} Grid as a Resource

The PS3^{TM} supports alternative operating systems, making the transformation into a Grid resource rather trivial, as a suitable Linux distribution and an appropriate Grid client are the only requirements. However if you target a large amount of PS3's this

[†]It is not clear whether it's to prevent games to be played outside Sony GameOS, due to DRM issues, or due to the exposure of the GPU's register-level information

[‡]The Cell processor consists of 8 SPE's, but in the PS3^{TM} one is removed for yield purposes, if one is defective it is removed, if none is defective a good one is removed to assure that all PS3's have exactly 6 SPE's available for applications, to preserve architectural consistency

becomes cumbersome. Furthermore if the PS3's located beyond the academic perimeter are to be reached, minimal administrational work form the donor of the PS3™ is a vital requirement. The FOLDING@HOME[BEJ+09] project has managed to make an agreement with Sony to place their client as an embedded part of the PS3's Sony GameOS, the BOINC projects are not Grid computing, but public resource computing, which differs by the fact that their projects targets specific projects written towards using a special framework, Grid computing supports a native execution environment for all applications, providing no need for rewriting the application towards a Grid context.

The approach presented here minimizes the workload required transforming a PS3™ into an powerful Grid resource by using a LIVECD. Using this CD, the PS3™ is booted directly into a Grid enabled Linux system. The LIVECD comes in two versions, a sandboxed version which isolated the Grid execution environment from existing systems installed, and a dedicated version where the LIVECD has unlimited[§] access to the PS3's hardware.

3.3 The PS3-LIVECD

Several requirements must be met by the Grid middleware to support the described LIVECD. First of all the Grid middleware must support resources which can only be accessed through a pull based model, which means that all communication is initiated by the resource, i.e. the PS3-LIVECD. This is required because the PS3's targeted by the LIVECD are most likely located behind a NATed router. Secondly, the Grid middleware needs a scheduling model where resources are able to request specific types of jobs, e.g. a resource can specify that only jobs which are targeted the PS3™ hardware model can be executed. Thirdly the Grid middleware should support resubmission of timed out jobs, as PS3's used for other purposes than being a Grid resource, e.g. gaming, join and leave the Grid system frequently, and thereby the Grid middleware should be able to handle this event, without failing the Grid jobs.

3.4 The MiG PS3-LIVECD

The idea behind the LIVECD is to boot the PS3™ by inserting a CD, containing the Linux operating system and the appropriate Grid clients. Upon boot, the PS3™ connects to the Grid and requests Grid jobs without any human interference. Several issues must be dealt with. First of all the PS3™ must not be harmed by flaws in the Grid middleware nor exploits through the middleware, Secondly the Grid jobs may not harm the PS3™ neither by intention nor by faulty jobs. This is especially true for sandboxed version where an exploit may cause exposure of personal data.

[§]Besides the limits introduced by the hypervisor

3.4.1 Security

To keep faulty Grid middleware and jobs from harming the PS3™, both versions of the LIVECD use the operating system as a security layer. The Grid client software and the executed Grid jobs are both executed as a dedicated user, who does not have administrational rights of the operating system. The MiG system logs all relations between jobs and resources, thus providing the possibility to track down any job.

3.4.2 Sandboxing

The sandboxed version of the LIVECD operates in a sandboxed environment to protect the donated PS3™ from faulty middleware and jobs. This is done by excluding the device driver for the PS3™ HDD controller from the Linux kernel used, and keeping the execution environment in memory instead. Furthermore, the support for loadable kernel modules is excluded, which prevents Grid jobs from loading modules into the kernel, even if the OS is compromised and root access is achieved.

3.4.3 File Access

Enabling file access to the Grid client and jobs without having access to the PS3's harddrive is done by using the graphics card's VRAM as a block device. As main memory is a limited resource[¶], using the VRAM as a block device is therefore a great advantage compared to the alternative of using a ram disk which would decrease the amount of main memory available for the Grid jobs. However the total amount of VRAM is 252 MB and therefore Grid jobs requiring input/output files larger than 252 MB are forced to use a remote file access framework[AV05].

3.5 The Execution Environment

The PS3-LIVECD is based on the Gentoo Linux[Lin] PPC64 distribution with a customized kernel[ext] capable of communicating with the PS3™ hypervisor. Gentoo catalyst[Cat] was used as build environment, this provides the possibility of configuring exactly which packages to include on the LIVECD, as well as providing the possibility to apply a custom made kernel and initrd script.The kernel was modified in different ways, firstly loadable modules support was disabled to prevent potential evil jobs, which manages to compromise the OS security, from modifying the kernel modules. Secondly the frame- buffer driver has been modified to make the VRAM appear as a memory technology device, MTD, which means that the VRAM can be used as a

[¶]The PS3™ only has 224 MB of main memory for the OS and applications

block device. The modification of the frame-buffer driver also included freeing 18 MB of main memory occupied by the frame-buffer used in the default kernel[‖].

The modified kernel ended up consuming 7176 kB of the total 229376 kB main memory for code and internal data structures, leaving 222200 kB for the Grid client and jobs. Upon boot the modified initrd script detects the block device to be used as root file system[**] and formats the detected device with the ext2 filesystem, reserving 2580 kB for the superuser, leaving 251355 kB for the Grid client and jobs[††]. When the block device has been formatted, the initrd script sets up the root file system by coping writable directories and files from the CD to the root file system. Read-only directories, files, and binaries are left on the CD and linked symbolically to the root filesystem keeping as much of the root filesystem free for Grid jobs as possible. The result is that the root file system only consumes 1.6 MB of the total space provided by the used block device.

When the Linux system is booted the LIVECD initiates the communication with MiG through HTTPS. This is done by sending a unique key identifying the PS3™ to the MiG system, if this is the first time the resource connects to the Grid a new profile is created dynamically. The response to the initial request is the Grid resource client scripts, these are generated dynamically upon the request. By using this method its guaranteed that the resource always has the newest version of the Grid resource client scripts, disabling the need for downloading a new CD upon a Grid middleware update. When the Grid resource client script is executed the request of Grid jobs is initiated through HTTPS.Within that request a unique resource identifier is provided, giving the MiG scheduler the necessary information about the resource such as architecture, memory, disk space and a upper time-limit. Based on these parameters the MiG scheduler finds a job suited for the PS3™ and places it in a job folder on the MiG system. From this location the PS3™ is able to retrieve the job consisting of job description files, input-files, and executables. The location of these files is returned within the result of the job request, and is a HTTPS URL including a 32 character random string generated upon the job request and deleted when the job terminates. At job completion the result is delivered to the MiG system which verifies that it's the correct resource (by the unique resource key) which delivers the result of the job. If it's a false deliver[‡‡] the result is discarded, otherwise its accepted. And the PS3™ resource requests a new job

[‖]As the hypervisor isolates the GPU from the operating system, the display is operated by having the frame-buffer writing the data to be displayed to an array in main memory, which is then copied to the GPU by the hypervisor

[**]The sandboxed version uses VRAM, the dedicated version uses the real harddrive provided through the hypervisor

[††]This is true for the sandboxed version, the dedicated version uses the total disk space available, which is specified through the Sony Game OS

[‡‡] The resource keys doesn't match, the time limit has been violated, or another resource is executing the job, due to a rescheduling

when the result of the previous one has been delivered.

3.6 Experiments

Testing the PS3™ Grid Resource model was done establishing a controlled test scenario consisting of 8 PS3's in their own VGrid. The experiments performed included of a model overhead check, a file system benchmark using VRAM as a block device, and application performance, using a protein folding application.

3.6.1 Job Overhead and File Performance

The total overhead of the model was tested by submitting 1000 empty jobs to the Grid with only one PS3™ connected. The 1000 jobs completed in 12366 seconds, which translates to an overhead of approximately 13 seconds per job. The performance of the VRAM used as a block device was tested by writing a 96 MB file sequentially. This was achieved in 1.5 seconds, resulting in a bandwidth of 64 MB/s. Reading the written file was achieved in 9.6 seconds, resulting in a bandwidth of 10 MB/s. This shows that writing to the VRAM is a factor of approximately 6.5 faster than reading from the VRAM, which was an expected result as the nature of VRAM is write from main memory to VRAM, not the other way around.

3.6.2 Protein Folding

Protein folding is a compute intensive algorithm for folding proteins. It requires a small input and generates a small output, and is embarrassing parallel which makes it very suitable for Grid computing. In this experiment, a protein of length 27 was folded on one PS3™ resulting in a total execution time of 57 minutes and 16 seconds. The search space was then divided into 17 different subspaces using standard divide and conquer techniques. The 17 different search spaces were then submitted as jobs to the Grid, which adds up to 4 jobs for each of the 4 nodes used in the experiment plus one extra job to ensure unbalanced execution. Equivalently, the 17 jobs were distributed among 8 nodes, yielding 2 jobs per node plus one extra job. The execution finished in 18 minutes and 50 seconds using 4 nodes giving a speedup of 3.04. The 8 node setup finished the execution in 10 minutes and 56 seconds giving a speedup of 5.23, this is shown in figure 3.2. These results are considered quite useful in a Grid setup. It should be noted that the unbalanced execution scheme provided to simulate that Grid executions is most likely unbalanced, results in 7 of the 8 nodes, in the 8 node setup, to idle while the last node is executing the last job.

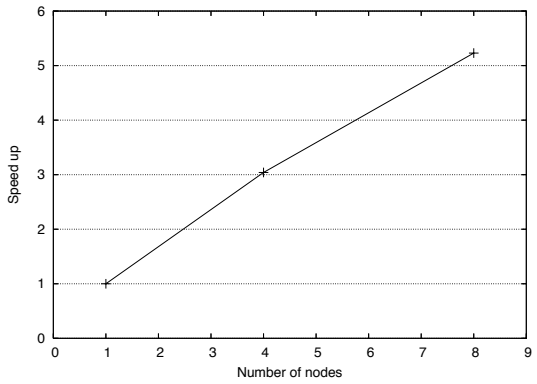


Figure 3.2: The speedup achieved using the PS3-LIVECD for protein folding with 4 and 8 nodes

3.7 PS3-LIVECD Summary

The PS3-LIVECD framework presented here provides a method for connection all PS3's to a Grid infrastructure, with minimal administrational work both on the Grid administration side and on the resource owner side. This is essential to reach the huge amount of PS3's located outside the academic perimeter. Two versions of the PS3-LIVECD have been presented, one which operates in a sandboxed environment, and one where the Grid environment has full access to the PS3's hardware. The first version is targeted resource owners outside the academic perimeter, where the latter are targeted resources dedicated to scientific computing through the Grid middleware. The sandboxed version is isolated from the harddrive of the system, by removing the HDD driver from the Linux kernel and removing support for loadable module support from the kernel, thereby the harddrive is totally isolated from the Grid execution environment and can't be compromised. This is essential as the Grid middleware then can guarantee the safety of the resource even though unknown binaries, the Grid jobs, are executed.

The shortcoming of this model compared with the FOLDING@HOME client provided inside the Sony GameOS, is that the PS3^{TM} needs to be rebooted with the CD/DVD in the drive, and thereby is a little more intrusive than just starting an application within the Sony GameOS. The advantage of our model is that it provides a fully functional Linux Grid execution environment, capable of executing any scientific application written for the Cell BE processor.

Chapter 4

Application Porting and Tuning on The Cell-BE Processor

To verify that non-computer resource contributes the the Grid environment in the traditional HPC context, an application has been ported from the X86 architecture to the PS3™ architecture*.

In this section a step by step porting of an nqueen[nqu48][Dij72] application written by Takaken[Tak] is covered. This Takaken solution to the nqueens problem is a small piece of code based on bitmasks and recursion and is highly optimized towards the X86 architecture. The full length paper published on the this topic can be found in appendix B.

4.1 The Cell-BE Processor

The Cell processor located in the PS3™ is a heterogeneous multi core processor consisting of 9 cores. The Primary core is an IBM 64 bit power processor (PPC64) with 2 hardware threads. This core is the link between the operating system and the 8 powerful working cores, called the SPE's for Synergistic Processing Element. The power processor is called the PPE for Power Processing Element, all cores are connected by an Element Interconnect Bus (EIB). Figure 4.1 shows an overview of the Cell architecture. The cores are connected by an Element Interconnect Bus (EIB) capable of transferring up to 204 GB/s at 3.2 GHz. Each SPE is dual pipe-lined, has a 128x128 bit register file and 256 kB of on-chip memory called the local store. Data is transferred asynchronously between main memory and the local store through DMA calls handled by a dedicated Memory Flow Controller (MFC). An overview of the SPE is shown in figure 4.2. By using the PPE as primary core, the Cell processor can execute PPC64 applications out of the box, however these will only use the PPE core. To use the SPE

*This is the Cell BE architecture with only 6 available SPE's

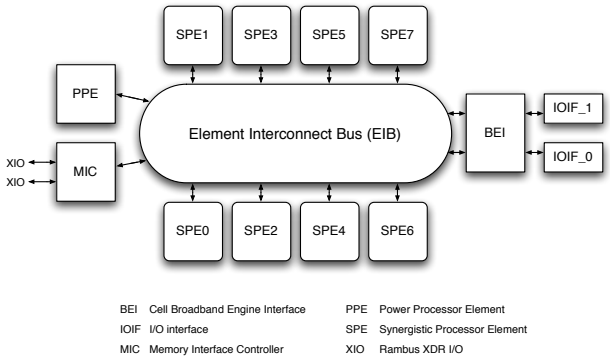


Figure 4.1: An overview of the Cell architecture

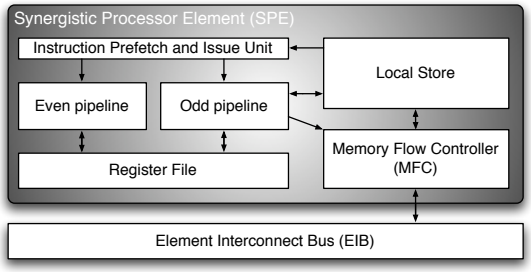


Figure 4.2: An overview of the SPE cores

cores it's necessary to develop code specifically for the SPE's, which includes setting up a memory communications scheme using DMA through the MFC. For a in-depth description of the Cell BE architecture, please look at the Master Thesis by Mohammad Jowkar[Jow07]

4.2 Porting Towards the Cell-BE

To make the Cell BE perform at a high level, one has to consider several levels of parallelization, including task-, memory-, data and instruction-level parallelization. The application ported is a divide and conquer algorithm for finding how many ways to place N queens safely at an NxN chess board according to the common chess rules.

4.2.1 Task and Memory-parallelization

As the PS3™ architecture can be viewed as a 6 node[†], the SPE's, cluster[SSB+95] on a chip with a front end, the PPE, splitting an application into smaller tasks is done by the same principles as when parallelizing towards a cluster computer. However due to the limited amount of local store, 256 kB, available at the SPE's for both code and data, one has to consider the size of each task. This means that an application which would be best parallelized by a bag of task model in a cluster setup, might be best parallelized by a pipelined setup using the Cell processor.

Task-parallelization The first step of porting an application to the Cell is to parallelize it task wise, using the PPE as a task manager and the SPE's as computation units. This is done by analyzing the application, picking the right method for parallelization including analyzing if data and code for each task fits into the 256 kB local store of the SPE's, and if not, which method one wishes to use to make it fit. The two possibilities here are either to split the data-set for each task into smaller data-sets or to use a pipelined setup, where the code is split among 2,4 or 8 SPE's each performing some piece of the computation.

Memory-parallelization Each SPE has its own MFC controller running in its own thread, meaning that main memory can be accessed through DMA asynchronously regarding computation. This gives the possibility of effective memory latency hiding, as the MFC writes data directly from the EIB to the local store without involving the computational unit. Thereby the data for iteration i+1 can be retrieved while computing iteration i, this is known as double buffering.

[†]The PS3™ has only 6 SPE's available for applications

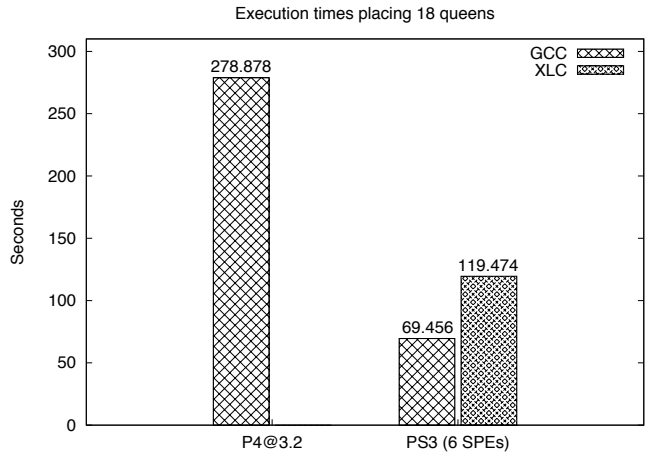


Figure 4.3: Execution times for the task parallelized nqueens application

Furthermore each MFC is capable of issuing 16 simultaneous DMA transfers giving a high level of possible multi-buffering[‡]. When porting an application, a communication scheme between the PPE and the SPE using DMA transfers through the MFC has to be chosen. Most applications will use at least double buffering to hide memory latency, and some applications need to use multi-buffering to keep the computational unit busy, this all depends of the computational intensiveness of the task.

4.2.2 The Nqueens Solution

In the nqueens example, code and data fit into the local store of each SPE, and therefore a bag-of-task model is used, where each SPE requests a task from the PPE, gets the input data, computes the result, delivers the result to the PPE and requests a new task. As the input and output data for the nqueens application is quite small and the compute intensive part of the application is quite large, double buffering is sufficient for keeping the SPE's busy, hiding the memory latency efficiently. This first step reduced the execution time of placing 18 queens on an 18x18 chess board from 278.878 seconds on a Pentium 4 running at 3,2 GHz to 69.456 seconds when executed on a Playstation® 3 [§] giving a speedup of 4. The application has been compiled with both the GCC compiler and IBM's XLC compiler, the result is shown in figure 4.3. It's seen that in this case the GCC compiler produces code which is significantly faster than the code produced by the XLC compiler.

[‡]Where data for iteration i+1,i+2, ..., i+n is retrieved in iteration i

[§]Note the PS3™ only has 6 SPE's

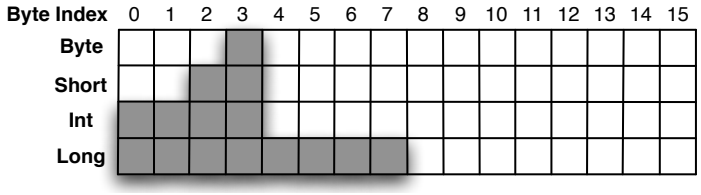


Figure 4.4: The preferred slot in a register line for the different data types

4.3 Register-line Optimizations

The Cell BE SPE's are SIMD[Sun00] vector cores each operating on a 128 bit register-line, which can be divided into 2x64 bit longs, 4x32 bit int's, 8x16 shorts or 16x16 bytes. This results in scalar operations to be mapped down to an atomic sequence of register-line operations, as the scalar has to be put in its preferred slot of the register-line see figure 4.4. The compute intensive part of the code should be fitted to use 128 register-line operations instead of scalars, as this will eliminate the rotate and shuffle instructions needed to get the scalar into the correct preferred slot.

4.3.1 Recursive vs. Iterative Methods

If the application to be ported is of a recursive nature, one might consider to transform the compute intensive part into an iterative method, as the limited local store of 256 kB is exhausted quite quickly if deep recursions are reached. Furthermore, the use of recursive methods might slow down execution if the data used in the compute intensive part is fitted into the 2 kB register file, as local parameters are pushed to the stack upon a function call.

4.3.2 Branch Prediction and Elimination

The Cell processor has no hardware branch predictor, but the instruction set contains a branch hinting instruction. However if the programmer has no clue on what branch is to be taken, and the piece of code within each branch is fairly small, branch elimination can be done by calculating both results and selecting the right result based on the branch condition[IBM07]. This has two advantages, it eliminates branch misses and it operate directly on register-lines whereas normal branch operations operates on scalars. Using these two methods the overall penalty of branching can be reduced quite impressively.

Figure 4.5: Execution time comparisons between the different optimizations

4.3.3 The Nqueens Solution

The compute intensive part of the presented nqueen application was transformed from a recursive algorithm, to an iterative algorithm using the described register-line optimizations and branch elimination techniques. This resulted in an execution time reduction from 69.456 seconds to 42.486 seconds, giving a speedup of 1.63 compared to the task parallelized code, and a speedup of 6.55 compared with the Pentium 4 execution running at 3.2 GHz. Furthermore the performance of the iterative scalar version of the algorithm was measured, all versions was compiled with both the GCC and XLC, the results are shown in figure 4.5. It's seen that the recursive scalar code overall performs a little better than the scaler iterative code, but that the XLC compiler produces faster binaries when used on the iterative code. Finally it's seen that the XLC compiler produces 15% faster binaries from the register-line code compared to GCC. These results show that the GCC produces faster binaries in the common case[¶] whereas XLC produces faster binaries when the application is tuned towards the Cell SPE's architecture.

4.4 Data Parallelization

As mentioned, the Cell BE SPE cores are SIMD vector cores, which offer data parallelization as an optimization parameter. Each SPE core is capable of performing 4 integer operations, 8 short operations or 16 byte operations per instruction. If the data has an integer SIMD nature, for example integer matrix multiplication, one can effectively vectorize by loop-unrolling, doing four integer operations in each loop shortening the total loop length by a factor of four. Otherwise if the application has a divide and

[¶]Applications written for traditional single core architectures

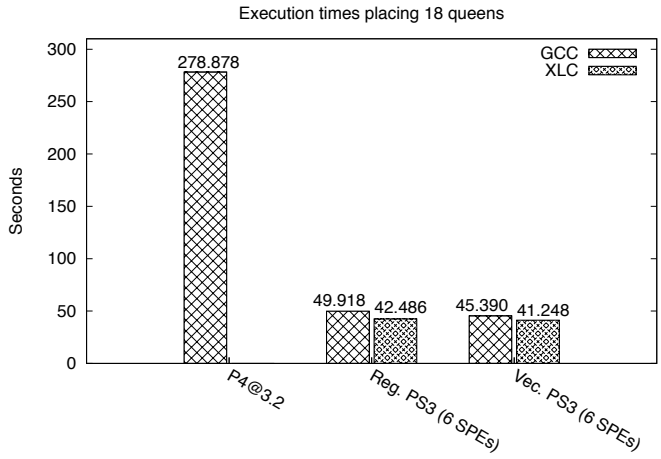


Figure 4.6: Execution times for the vectorized nqueens application

conquer nature, one can vectorize by doing four branches simultaneously. The performance gain of branch vectorization depends heavily on the balance of the four branches, as the amount of leaves traversed is the worst case of the four branches.

4.4.1 The Nqueens Solution

The compute intensive part of the presented nqueens application is based on a divide and conquer algorithm. This algorithm was vectorized by investigating four branches simultaneously. The execution time of the vectorized code placing 18 queens was reduced from 42.486 seconds to 41.248 seconds, which is considered an insignificant speedup, the result is shown in figure 4.6. An analysis of the vectorized code showed that the average depth reached in the search tree increased by 0.3 due to unbalanced branch vectorization. The presented nqueens application has a search space of N!, this means that solving the problem for more than 14 queens eliminates the advantage of branch vectorization. Performance measurements show that when placing 20 queens, the branch vectorized code gets slower than the register-line optimized code, figure 4.7. This is due to additional operations needed to test when all four branches within the vector have met their termination criteria. To make the branch vectorized code perform better, one could try to balance the branches placed within each vector. This has not been done with the presented nqueens application, as it's believed by the author that the amount of processing power needed to balance the branches within the vectors is equal, or exceeds, the processing power needed to solve the problem itself.

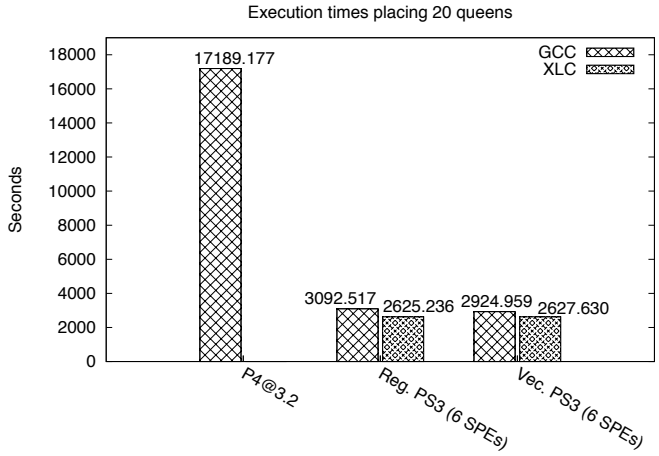


Figure 4.7: Execution times for the register-line and branch-vectorized nqueens

4.5 Instruction Parallelization

As the Cell BE is dual pipelined, one pipeline for computation and one for management, it's possible to perform load/store instructions from/to the local store while a computation instructions is being performed, this is specially usable within loops over data-sets, where it's clear which data are needed next. This is one of the optimization tasks which the compiler is capable of doing.

4.5.1 The Nqueens Solution

To measure how effectively the compiler interleaves the computation and data management instructions, the iterative register-line version[‖] of the nqueens application compiled with the IBM XLC compiler has been profiled by using the Cell-BE system simulator[nqu]. The result of this was that approximately 13.9% of all instructions are instruction parallelized[**]. However the profiling revealed that 8.5% of all clock-cycles are used waiting on load/stores between the register-file and the local store. This indicates there is still room for improvement, which can be done either by eliminating local store access by using registers if possible, or doing instruction level parallelizing by hand at assembler level. This has not been looked further into.

[‖]The iterative register-line version was the one performing best

[**]Called dual-instructions in Cell-BE terminology

4.6 Nqueens Summary

When porting an application from a traditional single core application, i.e. the X86, to the Cell BE architecture, at least task and memory-parallelization should be chosen as the PPE of the Cell in itself performs poorly, see figure 4.8. The core computational

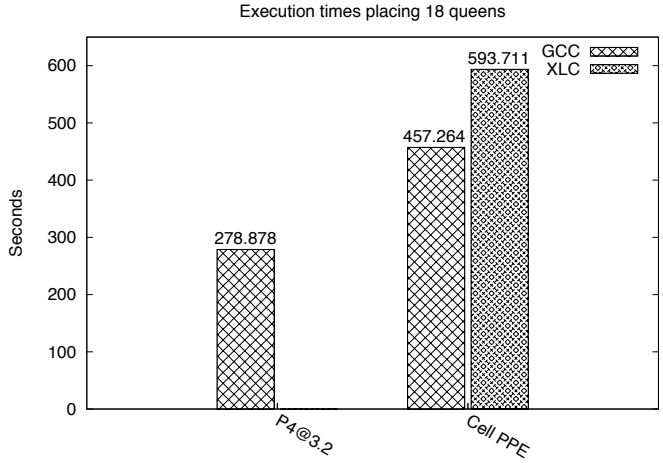


Figure 4.8: Execution times for the PPE nqueens application

part of the task and memory-level parallelized application can be compiled directly on the SPE's and thereby requires no rewriting. As mentioned, the speedup gained using the task and memory-level parallelization is 4 on 6 SPE's. Moving from the tasks-and memory-level parallelized version to the register-line version with branch elimination improved the speedup from 4 to 6.5 using 6 SPE's, but this required a rewrite of the compute intensive part using 203 lines of code as opposed to the 75 lines of code used in the original recursive version. The vectorized version resulted in 348 lines of code, but didn't perform at all, due to unbalanced branch-vectors. However, if one has a well balanced branch-vector or data is SIMD applicable, one can expect a vectorized version to perform well, but it's not possible to reach beyond a speedup of 128/X, where X is the size of each data entry in the vector, compared to a register-line version.

Chapter 5

The Remote Memory Library

In the perfect world a computing device would hold sufficient physical memory to allow all running processes and their data in memory. Naturally this is not the case and we thus need mechanisms to administrate the available physical memory to help optimize utilization of the hardware. This task is traditionally performed by the OS memory manager, which has knowledge of the memory in use by the active processes running at the system. However when moving from a homogeneous system with a single operating system to a global Grid environment with billions of autonomous machines and operating systems there is no longer any global memory manager and therefore the memory is managed autonomously be each individual machine connected to a Grid infrastructure. Our goal is to change that by introducing a Grid memory manager that is responsible for the memory used by the Grid jobs executed on the connected resources. Instead of replacing the memory managers of the connected resources we aim at interposing a Grid memory layer at user-level between the operating system and the Grid application at the local resources. This means that we utilize the local operating system memory manager and merely monitor the amount of memory the Grid application uses while executing the Grid jobs. Thereby it's possible to control the amount of physical memory used by the executing Grid application and aid the resource with memory through the Grid infrastructure when needed. By interposing the memory library at user-level it's possible to deploy the memory library at the resources at runtime along with the Grid jobs without interference from the resource owner. This is vital as modifications to the operating system of the donated resources is both an administrative burden on the resource owners and a security risk. The full length paper published on the The Remote Memory Library can be found in appendix D.

5.1 Memory Management

All modern computer architectures have a memory management unit and use an operating system, which provides a full virtual address space to each process. This gives the processes an impression of having exclusive access to a system where the upper limit of memory is bound only by the hardware architecture. While providing a full virtual address space for each process is a powerful abstraction, it also places a large responsibility on the OS to manage the available physical memory. Traditionally this is achieved by swapping memory to disk when the system runs short on physical memory. If the system exhausts the available disk space for swapping, the OS has to decide what to do. Typically the policy is killing a process to free up memory.

5.2 Kernel-level vs. User-level

The OS memory manager is responsible for swapping page-frames between physical memory and secondary memory, typically a hard-disk. In Linux the swap device is implemented as a generic block device, which makes it easy to change swap-target implementation as one can plug these directly into the kernel just by emulating the behavior of a block device. This approach has the advantage that the memory manager is left unmodified and works independently of which underlying media is actually used as storage for the swapped out memory. The downside to this approach is that it requires administrator privileges to deploy such a block device into the kernel, and implicitly that the administrator will have to trust the code that is added to the memory manager since it will run with kernel privileges.

While the kernel-level approach is easy to implement due to the cleaner interface with the OS, the user-level model is more flexible in a Grid context since it overrides the local memory manager and thus provides a homogeneous swapping mechanism to the Grid resources without requiring administration privileges or implicit trust to the swapping module. This enables the Grid infrastructure to fully control the amount of physical memory that the executing application is allowed to use on the resource *. The user-level model however has several drawbacks, primarily that you have to implement a new memory manager to work on top of the native memory manager. However reusing most of the OS memory manager and overloading only a few functionalities such as allocation of memory, freeing memory and swapping pages in and out of physical memory may be sufficient. Another drawback of the user-level approach is that it invokes frequent switching between user- and kernel-level, which results in an execution time overhead compared to the native model where everything is done in kernel space. In addition an overhead in memory consumption is imposed because the user-level library must maintain a set of internal data structures to keep track of the state and

*Naturally it's not possible to utilize more physical memory than present at the resource

location of pages that are used by the application. Last but not least a user-level library doesn't have access to the hardware supported status bits of the process page-table, which has the effect that the widely used LRU eviction algorithm is not applicable [†]

Despite the overhead of the user-level approach compared to the kernel-level approach, we choose to make our remote swap library run at user-level. This is chosen because it imposes fewer requirements for deployment in Grid environments, as well as the opportunity to change the page replacement algorithms with algorithms that are more suited towards a Grid environment.

5.3 The Remote Swap Framework

The Remote Swap framework consists of two components namely a memory server and a Remote Memory Library (RML). This memory library is interposed in user-level between the OS and the user process on the executing Grid resource to provide a transparent user-level swap mechanism. It's responsible for allocating memory, freeing memory and evicting pages to the remote memory server, as well as bringing pages back into physical memory, when they are once again needed.

5.4 The Remote Memory Library

The goal of making RML transparent to both the user application and the underlying OS implies that RML should support all the memory routines that are available to the application writer in the original environment. In this initial version of RML, Linux heap memory is the target for remote swapping, making the libc routines *malloc, calloc, realloc* and *free* the ones supported in RML. Rather than re-writing and maintaining these routines, they are merely overloaded with the purpose of maintaining a local page-table in RML to keep track of the pages in use by the user process. The actual memory allocation is done by calling the generic memory routines in libc from within the overloaded routines. This is illustrated in figure 5.1.

5.5 Page Eviction

When the user process reaches its physical memory limit pages need to be evicted. The target pages are found using the second chance FIFO evict strategy rather than the LRU strategy, which is used by most OSs, because LRU is not feasible in a user-level environment. When the pages to be evicted are found, they are protected in read mode to ensure consistency by preventing any other active user threads from modifying them

[†]LRU uses the hardware page referenced bit, which is not accessible at user-level

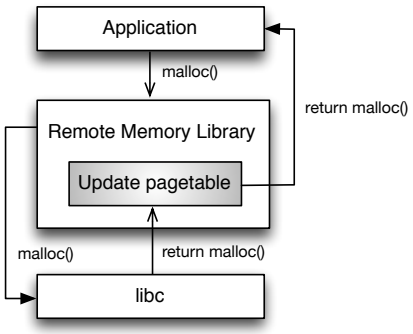


Figure 5.1: The flow of a malloc call from the user application

during eviction. If the chosen pages were modified while resident in memory, they are sent to the remote memory server through the network. The physical memory used for the evicted pages are then released and the virtual pages are protected in order to detect when the user process tries to access them.

5.6 Page Retrieval

When the user process tries to access a protected page, the kernel will send, an access violation signal to the user process. This signal is intercepted by RML, which checks the state of the violated page in its local page-table. If the page has previously been swapped out, a page request is sent to the remote server in order to retrieve the page. The server responds with the page data, which is placed into the correct page slot and control is given back to the user process which can continue execution. This is shown in figure 5.2. If RML doesn't have any information about the violated page the access violation signal is forwarded to the user process which then has to handle the signal.

5.7 Page Blocks

The memory manager in modern OSs arrange memory into an abstraction called page-frames, which is blocks of contiguous bytes. In the same manner RML arranges blocks of contiguous pages into what we call page blocks in order to swap out and retrieve several contiguous pages in a single evict or retrieve operation. This is beneficial if the executing application has a sequential memory access pattern across page blocks, that is the byte access pattern within a page block can be scattered as long as it doesn't access bytes outside the page block or its adjacent neighbors. Not all scientific applications have a strictly sequential memory access pattern, but even then they may still take

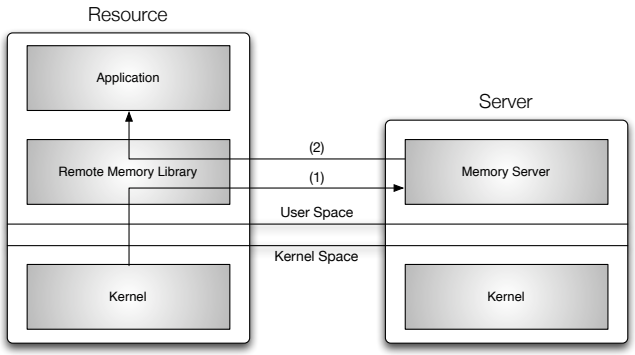


Figure 5.2: The flow of swapping in a page from the remote server. (1) The kernel catches a page access violation which is sent to the user process but intercepted by RML and transformed into a server page request. (2) The server responds with the page needed which is transferred to the client and mapped into the right position in memory and control is given back to the user process

.

advantage of using blocks of pages when evicting or retrieving pages. This is clarified in the experiments section. The optimal number of pages to block into one page block is dependent on the network latency between the client and the memory server, and the memory access pattern of the executing application. In this first version of RML the page block size (PBS) is provided to the framework before execution. Adaptive adjustment towards the needs of the executing algorithm is subject for future work.

5.8 The Memory Server

The memory server is a user-level process communicating with the clients through a TCP socket. Two kinds of services are offered by the memory server: Page send and page retrieve. When the client asks for a service the index of the page to send/retrieve is sent along with the request. The server stores the pages in memory and uses a hash table with the page index as key and the memory address where the page is stored locally at the memory server as value. When a page is received at the server the page index is looked up in the hash table to check if memory has previously been allocated for the specific page due to an earlier swap-out. If the page has an entry in the hash table the memory associated with it is reused, otherwise memory is allocated for the new page and the key/value pair is inserted into the hash table.

Upon a page retrieve request the server makes a lookup in the hash table, if the page index is present in the hash table the server responds with the data associated with the page index otherwise an error message is sent to the client.

5.9 Implementation Details

The current implementation of RML is bound to the Linux kernel, but is portable to any page based operating system that supports virtual memory and functions to manipulate the virtual page tables such as mmap, mremap, munmap and mprotect or equivalents. In addition to the page table manipulation functions, support for overloading the default signal handler is required in order to detect page access violations at user-level. RML is loaded in between the system level libraries such as libc and the user application by using the LD_PRELOAD environment variable. This way the library can be a part of any Grid job without involving the resource owner, as it's delivered along with the Grid application and loaded as a part of the Grid job. When the library is initialized malloc, calloc, realloc and free is overloaded and thereby every call to these functions passes through RML, which means that the user application doesn't need to use customized memory functions in order to use the framework, this is what provides the transparency to user applications.

5.9.1 UDP vs. TCP

The pages transferred between the client and the server can either be sent using UDP or TCP. Using UDP might give better performance as it has no transmission control protocol overhead. However reliability is required in order to be sure that pages are properly sent between the client and the server. The ordering might be omitted if it's just guaranteed that a whole page is sent in one chunk, because we don't care if page X arrives before Y if the time span is short. Looking at the Linux operating system revealed that the default maximum transmission unit (MTU) of the network interfaces is 1500 bytes, with a default page size of 4096, this means that the data corresponding to one page is not guaranteed to arrive in the correct order. Therefore using UDP would require an implementation of reliability and ordering mechanisms as a part of the framework. Because these features are incorporated in TCP and are what differentiates UDP from TCP the latter was chosen as the communication protocol for the framework.

5.9.2 Nagle's Algorithm

Congestion control in TCP/IP internetworks introduced by J. Nagle[Nag84] is incorporated in most TCP stacks of modern operating systems with the purpose to eliminate network flooding when sending small packages. A TCP/IPv4 package has 40 bytes of header which means sending eg. 1 byte at a time will impose a huge overhead and possible flooding of the network if a large amount of small messages are sent. Nagle's algorithm solves this issue by buffer messages until enough data is available to fill the maximum TCP package payload or a timeout occurs. However, this strategy performs poorly in a real-time system like the RML, where no network traffic is present until a

page fault arrives and a small page request header of 16 bytes is sent to the server in order to receive the faulting page data. The client page request of 16 bytes is hardly enough to fill the maximum TCP package payload on any modern network. Thereby each page fault request is delayed by the timeout of Nagle's algorithm unless page faults occur frequently enough to be piggy-backed on the ACKs sent to the server when page data is received. Since this behavior is not feasible in RML, Nagle's algorithm is disabled by configuring the TCP socket used for communication between the client and the server with the TCP_NODELAY option.

5.9.3 The Local Page Table

RML uses a local page table in order to keep track of the pages managed by RML. The initial version only supports 32 bit applications, because the page table is implemented as an array using page indexes[‡] to lookup information. In a 64 bit environment a new data structure is needed as an array for 64 bit addressing even when using page indexes, will consume all available memory if possible at all. In this case a more advanced data structure such as hash tables or other *key* $\rightarrow$ *value* based data structures has to be considered. In order to keep track of the pages administrated by RML each page is associated with a local page state and a previous and next pointer to maintain a page activated order used when evicting pages. Last but not least an allocation counter is kept in order to keep track of the number of active allocations associated with each page. This is used when freeing memory to detect when a page becomes inactive. Pages managed by RML can be in the following states:

Unused Pages that are not yet managed by RML

System Pages that are never evicted, these includes the page table itself as well as the buffers used for retrieving pages from the remote server.

Allocated Pages which are a part of an allocation but have not yet been referenced, this means that no real memory is mapped to the virtual pages. These pages are protected in *no-access* mode, that is the first time the executing application tries to reference a page in state *Allocated* a page access violation will occur, this is trapped by RML and the page is re-protected in mode *write* where after the page state is changed to dirty.

Read Pages that have been swapped in as a result of a read access violation is protected in mode *read*. If the executing application tries to modify a page in mode *read* an access violation will occur, this is trapped by RML and the page is re-protected in mode *write* where after the page state is changed to dirty.

[‡]Page address divided by the page size

Second chance When searching for pages to evict, pages that are not evicted are marked as *second chance* and protected in *no-access* mode in order to detect if they are accessed between evicts. If accessed between evicts an access violation is trapped by RML and the *second chance* flag is removed.

Out Pages that are no longer present in memory have their virtual page slot mapped in *no-access* mode meaning that RML will detect an access-violation signal if the page is accessed by the application. If the page is in mode *out* the page will be retrieved from the server and mapped into the right position in memory before execution is returned to the application which continues.

Dirty Pages that are up for eviction are transmitted to the memory server and then unmapped from memory. The data of pages that have previously been evicted is already present at the remote server and therefore only data of pages that have been modified since the last evict is send to the server.

Padding Pages that are a part of a page block but not a part of the allocations managed by RML. That is, a page block consists of several contiguous pages, but it's not guaranteed that all of those pages are active and controlled by RML. These pages are marked as padded which means that they are ignored by RML when modifying page blocks.

5.10 Memory Allocation

In order to avoid re-implementing the existing memory allocation routines, RML merely uses the original malloc, calloc and realloc routines to allocate memory. When allocating a chunk of memory, the address returned by the original allocator along with the chunk size is translated into page indexes. These indexes are used as keys for a local page table containing the page state information. The first and the last page are typically used by the original allocator to store internal information about the allocated memory and thereby these are mapped to real memory, the pages in between are merely marked as allocated both at operating system level and in the local RML page table. These pages are access protected by RML using the mprotect routine in order to detect when they are activated. The framework then returns the allocated memory address to the running application which continues its execution. When one of the newly allocated access protected pages is accessed by the running application, an access violation signal is thrown by the kernel. This signal is caught by RML and the page is looked up in the local page table. If the page is marked as allocated, but not yet used, the page is re-protected in mode write, the page table is updated, and execution is returned to the running application.

5.11 Page Eviction

The user-level approach makes it possible to regulate the real memory usage of a single application without considering other active applications on the system. When a page becomes active, either by activating a new page or swapping in a previously used page, it's checked if the upper active page limit has been reached. If the limit is reached pages are chosen for eviction using the second chance FIFO algorithm and thereafter protected in *read* mode, to prevent other active threads from modifying the pages during eviction. Modified pages are then sent to the remote server along with a header containing the page indexes. Finally the page states are changed to *swapped-out* and the real memory is freed by mmap'ing the evicted pages in protection *none*. This atomically frees the real memory and maps the virtual pages in *no access* mode. The evicted pages are now resident at the remote memory server until they are once again accessed from the running application.

5.12 Page Retrieval

When pages are swapped out they are protected in *no-access* mode, this means that whenever the executing application accesses such a page, an access violation signal is sent by the kernel and trapped by RML. The page causing the access violation is looked up in the local page table and if the page is found to be resident at the memory server, a page request is sent. The server will respond with the requested page data, which is received in a local buffer reserved for swap-in page data. When the data is fully received, the receive buffer is protected in mode *read* to detect future modifications, used to determine if the page should be written back to the server upon its next swap-out. When the receive buffer is protected the page is put into the right position in memory by using the *mremap* routine. This routine atomically updates the kernel virtual page table keeping other threads from accessing the page until the page is in a write safe state[§]. Finally the page is marked in RML as *swapped-in*, the page retrieve buffer is re-allocated using mmap for future swap-in's and the execution is returned to the thread which accessed the swapped-out page.

Remapping pages One of the major challenges creating a user-level swap library is putting page data atomically into the right virtual memory slot when pages are swapped in, that is the executing threads should not be able to access the swapped in memory pages before the received data is fully in place. This is not trivial as the virtual page slot needs to be in *write* mode in order to place the received data into the right slot, however

[§]Without the remap routine one would need to make the target page writeable, in order to copy data from the buffer page to the target page slot. This would allow other user threads to modify the page before it's completely in place.

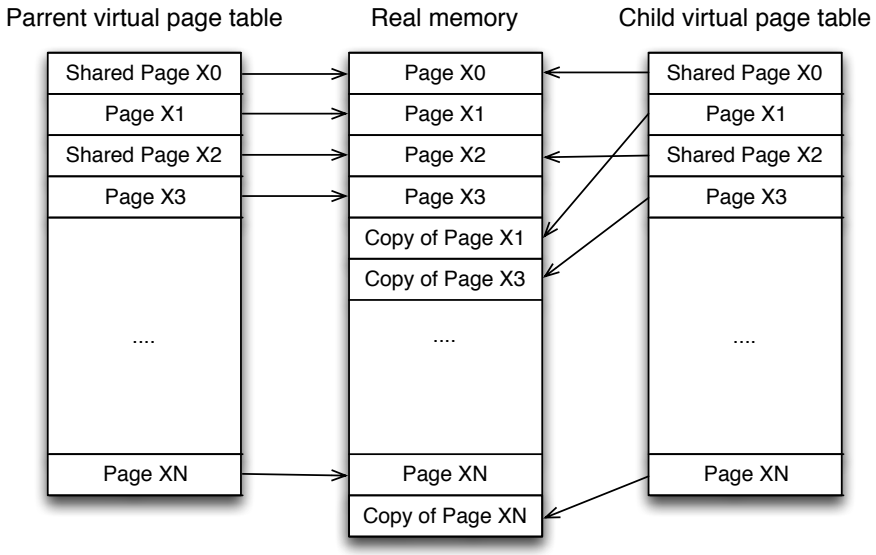


Figure 5.3: The memory layout of parent and child just after a fork

this will allow other user-level threads to access the page while RML is putting the data into place. We therefore need a mechanism for protecting the swapped in pages from executing user-level threads while the RML thread receives the page data and places it in the correct memory slot. This was first solved by using a fork/join model, where pages managed by RML is mapped in *shared* mode, which ensures that only the virtual page table is copied during a fork and not the actual page data. This means that the data of a shared page X is accessible both from parent and child, but with different access restrictions through their individual virtual page tables. The memory layout of parent and child just after a fork is shown in figure 5.3 where it's seen how the shared virtual pages of parent and child maps to the same real memory pages.

When the page data received from the server needs to be put into the right location a child is forked and the child's virtual target page slot is re-protected in mode *write* in order to copy the received data to it. As the child has its own virtual page table this will not affect the parent virtual page table and therefore the data can be safely copied to the right location by the child without any execution threads being able to read or write to it while being copied. This is shown in figure 5.4

The fork/join model were fully operational but turned out to have several issues regarding performance and scalability. The poor scalability was caused by the fact that when one maps a memory page as *shared*, a /dev/zero file-descriptor is attached to it. In a 32 bit Linux system with a page size of 4096 bytes, where user processes are allowed to use 3 GB of application memory this will result in 1048576 file descriptors, which consumes kernel memory and is far above the default amount of file descriptors allowed

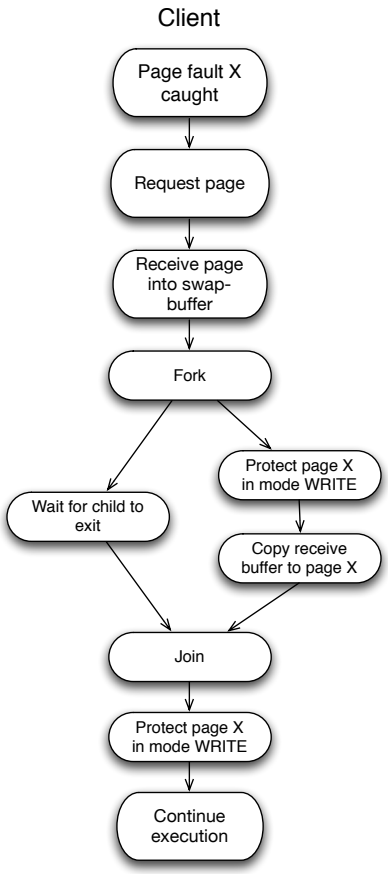


Figure 5.4: The flow of the fork/join page fault scheme

	fork/join	mremap
Execution time	1395.634 seconds	4.110 seconds
Speedup	-	340

Table 5.1: Initializing and reading an 128 MB array with a 64 MB real memory limit

by the kernel. While the fork/join setup is applicable in a 32 bit system by tuning kernel parameters, when moving to 64 bit architectures the number of shared pages which each result in a file descriptor increases beyond what's applicable. Therefore we set out to find another solution and discovered that the *mremap* routine was what we needed. This routine is capable of atomically moving a page from one virtual page slot into another without touching the page data. What it actually does is to update the virtual page table of the user process such that the data remains in it physical location merely updating the entries in the virtual page table. This method has several advantages, firstly it eliminates the file descriptor problem as pages no longer need to be mapped in mode *shared*, secondly it eliminates the fork/join scenario and thereby the overhead of fork and joining child processes. Last but not least data doesn't need to be copied from the receive buffer and into the right memory slot which saves time. Finally as it's merely the virtual page address that is modified, the right protection flags can be set on the page before it's remapped and thereby the page is in the right state as soon as the *mremap* routine returns.

Validating the performance gain obtained when moving from the fork/join model to the mremap model we set up an experiment initializing an array of 128 MB data by writing a byte to every index of the array and then reading it sequentially byte for byte afterwards. With a page size of 4096 bytes this results in 32768 pages being evicted and retrieved once during the execution when RML has a limit of 64 MB real memory usage. The execution time of this experiment dropped from 23 minutes and 15 seconds using the fork/join model to 4 seconds using the mremap model, which is a considerable speedup of 340. The result is shown in table 5.1

5.13 Experiments

To validate the Remote swap library and document its performance, experiments were done in an isolated execution environment. This consisted of a Playstation® 3 execution node and an dual quad core Intel Xeon running at 1.60 GHz with 8 GB RAM as memory server. These machines were interconnected through a 1 Gb/s switch and controlled by the Minimum intrusion Grid[KV05]. The Playstation® 3 was chosen as execution device because it represents a unextendible hardware device with a powerful processor but a limited amount of memory, namely 224 MB for the OS and user applications. As a reference machine without swap we used a Cell-BE QS22 blade. The reason for using an isolated high-bandwidth network with only one node, rather than a full Grid setup with a slower network and a number of nodes, is to validate the performance of the transparent user-level model under optimal conditions. If the model doesn't perform well under these conditions it will never perform in a real Grid system.

The RML framework has been tested with special designed highly I/O bound sequential memory access and scattered memory access applications, as well as real scientific applications. Each of the applications were tested with different page block sizes to evaluate how this influences the performance. The results of the experiments are covered in the following sections.

5.14 Sequential Data Access

The sequential data access tests were done by allocating N bytes of memory using malloc and then initializing the memory to make sure it was mapped to physical memory. Then a timer was started and the time spent traversing the memory start-to-end in 10 iterations (reading each byte to produce a checksum) was measured. Finally the performance was compared to the execution without swap on the Cell-BE QS22. This test is highly I/O bound as the only computation done by the executing machine is one integer addition per byte that is read. The performance of this execution with page block sizes 1 and 64 is shown in figure 5.5. The experiment shows that RML outperforms swap to disk significantly, as the memory consumption increases. Furthermore, the performance increases with the page block size until it starts to converge at 64 pages per block, which is were the bandwidth of the network is saturated. This is shown in Figure 5.6. We found the steep raise in disk swap execution time when the memory consumption reaches 1024 MB peculiar, as we are only performing sequential reads while measuring time and would expect the disk and its caches to be able to prefetch the pages needed. Thereby we would expect the execution time to raise no more than linearly with respect to the memory consumed and perform better than a user-level remote swap library as this algorithm is highly I/O bound. To investigate this further we settled out to take a closer look at the Linux swapping scheme.

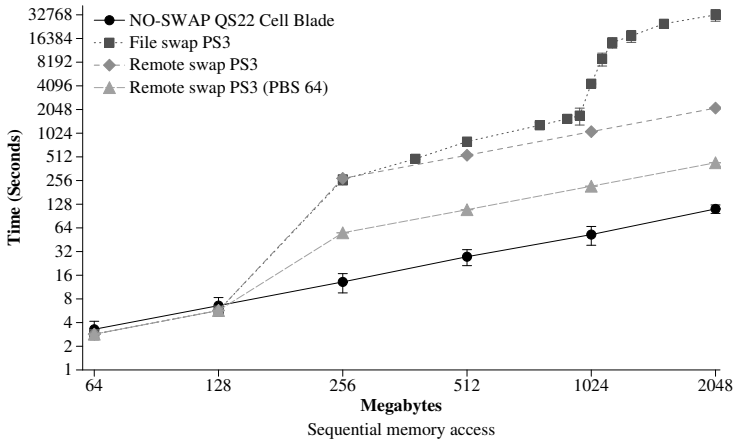


Figure 5.5: Sequential memory access performance

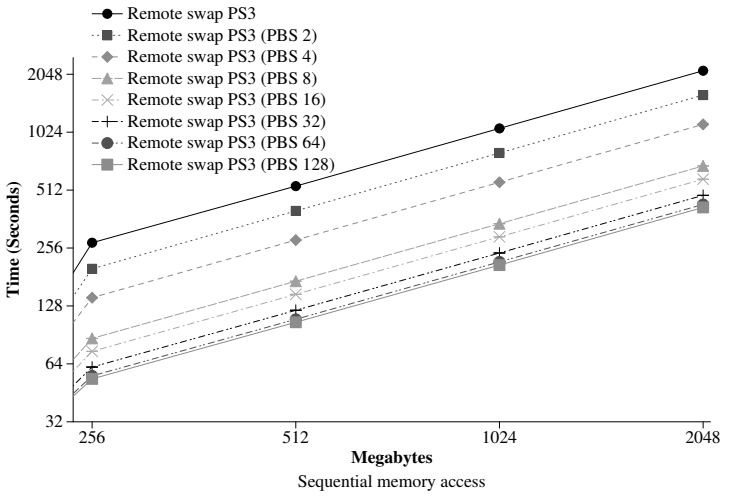


Figure 5.6: Sequential remote memory performance with increasing page block sizes

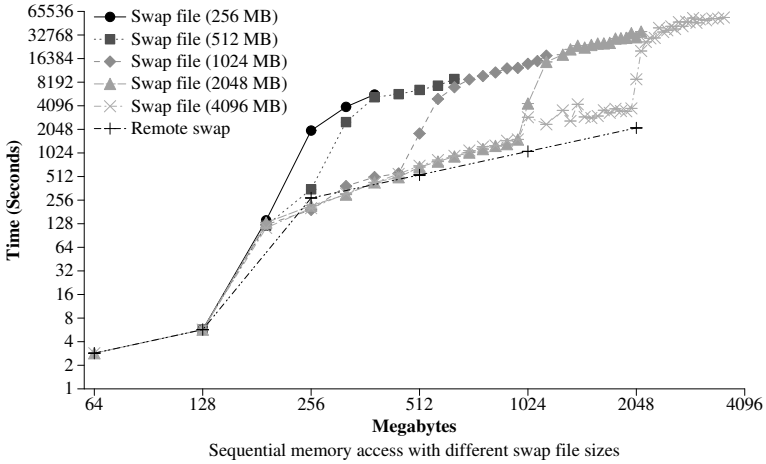


Figure 5.7: Linux file swap performance

5.15 Linux Disk Swap

In this experiment we executed the sequential memory access program described in the last section using different swap file sizes to see if the execution time was affected by the size of the swap device. All swap files were made using dd with a block size of 4096 bytes. The result of the executions is shown in figure 5.7. This experiment shows that disk swap performance is highly dependent on the swap file size, which was an unexpected result. We discovered the reason for this by looking into the Linux kernel source code where we found that when half of the disk swap space is filled the memory manager starts to remove swapped-in pages from the swap device to prevent it from running out of space. The effect is that all swapped-in pages are marked as dirty and thereby needs to be re-written to disk when they are once again swapped out. This is what causes the steep raise in execution time, when the swap device becomes half full, as the sequential experiment doesn't perform any writes in the 10 iterations that are used for time measurement.

In the rest of the experiments we continued to use a 2 GB swap file to show how the artifact of the Linux kernels half-full dirty mark strategy will be expressed within the scientific applications used in the experiments.

5.16 Sequential Data Access with Writes

In this test we used the sequential test described in the previous section and performed a write to each page accessed after its data had been read. The result is shown in figure 5.8. This experiment shows that even though a write was done to every page, it didn't eliminate the half-full Linux swap artifact, but the gap is smaller compared to

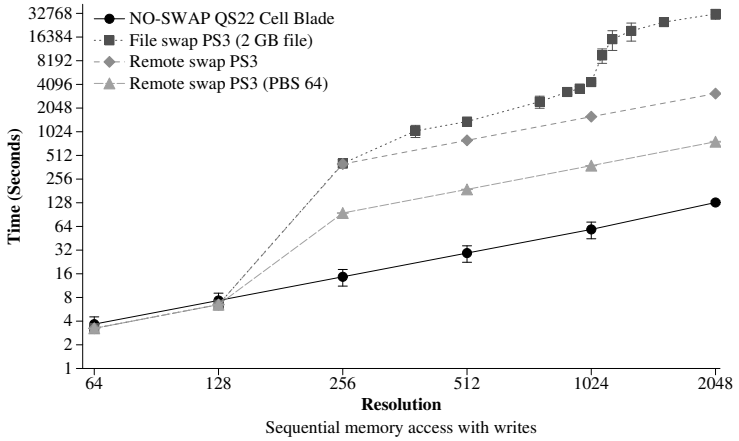


Figure 5.8: Sequential memory access with writes performance

the execution without writes. This is due to the fact that the Linux kernel in addition to marking the page dirty also removes it from the page cache and the swap device, which consumes time compared to the alternative of leaving the page on disk and then just writing it back out when needed. Beside the effect of the half-full artifact, it is observed that the execution time increases when using disk swap compared to remote swapping. This is caused by the mechanical structure of a disk, which causes the average seek time to rise as the amount of swapped out pages increases. As in the experiment without writes, the performance increases with the page block size until it converges at 64 pages per block, when the bandwidth of the network is saturated.

5.17 Scattered Memory Access

This test was designed similarly to the sequential access test, but instead of reading the pages in a sequential manner the data is read one page at a time, starting with the first page followed by the last page and then the second page followed by the second last page, this pattern is used until all pages are read. The result is shown in figure 5.9. This test shows that the execution time when swapping to disk is increasing relatively more than the execution time when swapping to the remote location. That is caused by the fact that disk swap can no longer take advantage of block prefetch combined with larger seek times when searching for the page to swap in, even before the half-full artifact arises. As with the sequential tests the performance increases with the size of the page blocks until the network is saturated at a block size of 64 pages.

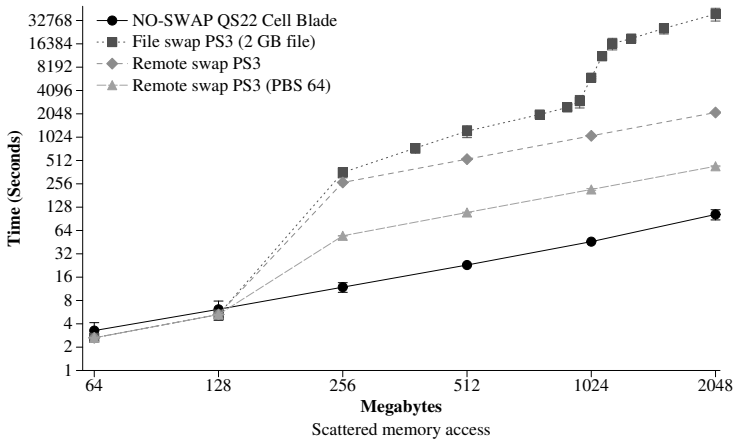


Figure 5.9: Scattered memory access performance

5.18 Scattered Memory Access with Write

This test is like the scattered access test described above, but with one write to each page like in the sequential write test. The result of this test is shown in figure 5.10. As with the sequential write test, the gap between Linux swap to disk and RML decreases compared to the execution without writes and the half-full artifact of the Linux swap is still present. The performance increases with the size of the page blocks until the network is saturated at a block size of 64 pages.

5.19 Lattice Boltzmann

OpenLB[slBc] is a free library for lattice Boltzmann simulations. In this experiment we used the forcedPoiseuille2d example provided in the package with different resolution sizes. The results of this test is shown in figure 5.11, which displays that swapping to the remote location outperforms disk swap with a factor of 6 and is 11 times slower than no-swap execution with a resolution of 2048 and a page block size of 4, which proved to be the optimal page block size for this application (figure 5.12). The half-full artifact is clearly visible in this test.

5.20 Fast Fourier Transform

Fftw[FJ05] is a free implementation of discrete Fourier transformation. In this test we use Fftw to transform a vector of random complex numbers to their corresponding Fourier values and back to the original values. The result of this test is shown in figure

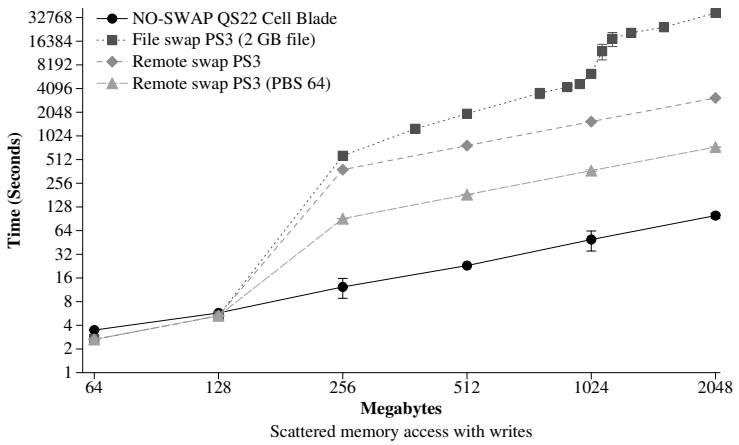


Figure 5.10: Scattered memory access with writes performance

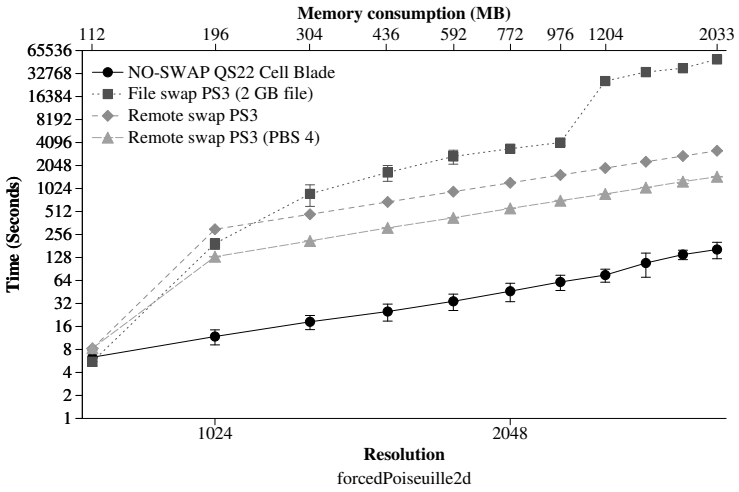


Figure 5.11: Lattice Boltzmann performance

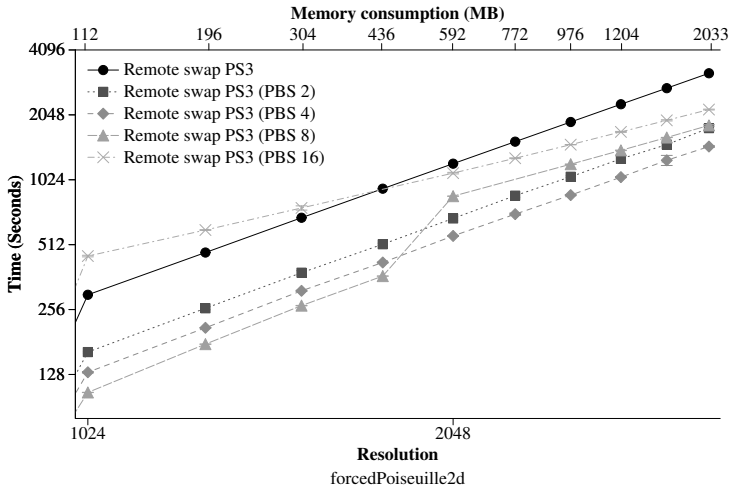


Figure 5.12: Lattice Boltzmann performance with increasing page block sizes

5.13, which displays that swapping to the remote location outperforms swapping to disk with a factor of 21 using a vector of 16777216 complex numbers and a page block size of 2. The slowdown compared to native executing at this instance is 4. Furthermore it shows that a page block size of 1 is the optimal when we reach a vector size of 67108864 complex numbers.

5.21 Barnes-Hut

The Barnes-Hut[BH86] algorithm is an O(nlogn) algorithm for performing N-Body force simulations. In this experiment we have used the code that is provided by one of the authors J. Barnes (the code can be downloaded from his ftp site[Bar]). The experiments were performed with the tree-body 6 test data, provided in the original source package, with a variable number of bodies and a random seed of 12345. The results is shown in figure 5.14, which displays that remote swap outperforms swap to disk by a factor of 6 and is a factor of 4 slower than native execution in real memory, when using 4194304 bodies and a page blocks size of 8, which is the optimal page block size (figure 5.15) for this application. It should be noted that the difference between swapping to disk and swapping to a remote memory location increases as the number of bodies grows and that the half-full artifact is present in this experiment.

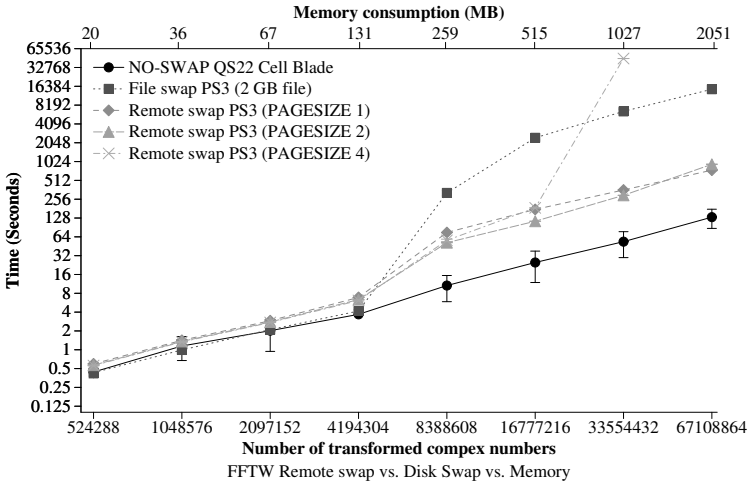


Figure 5.13: FFTW performance

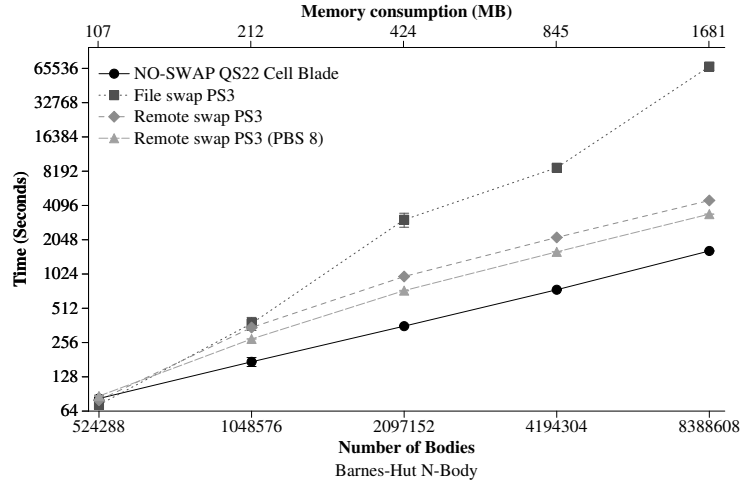


Figure 5.14: Barnes-Hut performance

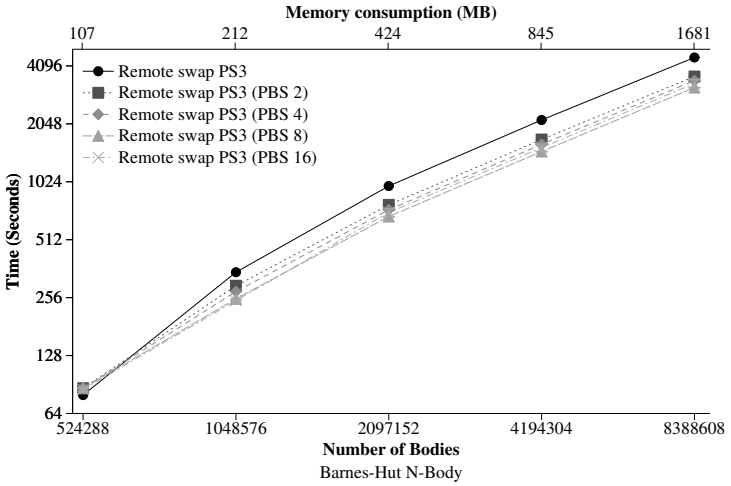


Figure 5.15: Barnes-Hut performance with increasing page block sizes

5.22 Experiment Summary

The experiments show that swapping to a remote memory server outperforms swapping to disk on the Playstation® 3 platform, both in the tests that are designed specially towards the framework, as well as in the generic scientific applications, which have not been modified in order to work with the framework. The speedup varies from 6 to 21 in the tested scientific applications and the slowdown varies from 4 to 11 compared to native executions. We have also shown that Linux disk swap has serious issues when it comes to scientific applications. The conservative strategy of removing pages from the swap device, whenever it becomes half-full, is poorly suited for scientific applications, due to their nature of initializing data and performing several iterations on the same data set, which means that the same pages are accessed several times without any renewal. Furthermore devices that are dedicated to scientific applications should favor the scientific application over other processes running at the system regarding CPU and memory.

5.23 Initial Remote Memory Library Summary

The initial Remote Memory Library presents a method for providing remote swap to global Grid infrastructures. Opposed to previous presented models, we present a fully transparent user-level library, which can be submitted along with the Grid jobs eliminating the need to modify neither the OS of the executing Grid resource nor the Grid application to execute. Furthermore the user-level approach makes it possible to throttle the real memory usage of the running job, through the Grid middleware, and thereby

increase the pool of resources capable of fulfilling the memory requirements of a given job. Last but not least the user-level approach ensures that only pages that are used by the Grid application are subject for eviction. The disadvantages of using the transparent user-level approach is the time overhead of passing signals, page mappings and page protections between kernel- and user-level, as well as the space overhead of keeping a local process page table within the framework as one can't access the page data-structures of the kernel from user-level.

Experiments show that the framework is operational and despite the introduced overhead still outperforms swapping to disk by a factor of 6 in the worst case. In the best case the framework outperforms swapping to disk with a factor of up to 21.

Chapter 6

Prediction Based Page Prefetching

The remote memory library introduced in the last chapter uses on-demand paging, which is the paging scheme used by most memory managers when retrieving pages from secondary memory. The remote memory library targets paging in Grid environments where the capacity of the network is the limiting factor regarding how fast pages can be swapped in. The capacity of computer networks grows constantly regarding bandwidth, but with respect to latency the upper limit is constant, namely the speed of light. As this upper limit can't be moved, on demand paging through a network will always be limited by the round-trip time of a page request and answer between the client and the memory server. With this in mind we set out to find out if we could take advantage of the increasing bandwidth of networks by applying a prefetching scheme to the remote memory library, such that instead of sending one page upon every page fault we can send several pages. Obviously sending the right pages is crucial as pages which are not needed will still consume bandwidth, memory and processing power at both the client and the server side. The question is, how do we decide which pages are needed in the near future. The novel solution is to send the pages X+1, X+2, ..., X+N when page X fails. This performs well for applications that have a purely sequential memory access pattern, but will perform poorly with applications which have a non-sequential memory access pattern. This was shown in the experiments of the initial remote memory library. In this chapter we introduce page prefetching using event-based prediction. That is based on the page fault pattern of the executing application what pages are likely to be used in the near future. While this is not well suited for desktop applications it's greatly suited for scientific applications as these often traverse the same data set over and over. Furthermore scientific applications that are well suited for Grid computing are often executed several times using different parameters which can prove to be an advantage if the page access pattern of one execution is saved within the Grid infrastructure for future executions.

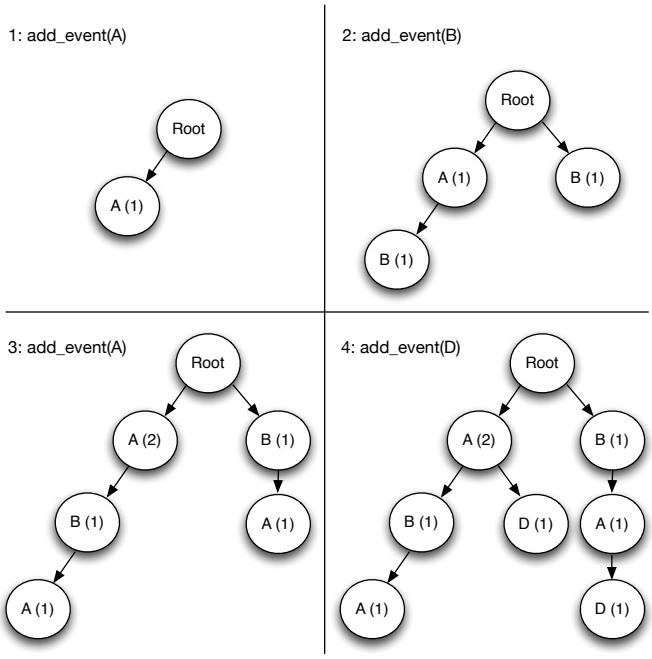


Figure 6.1: Structure of the Oraculo Trie adding the sequence A, B, A, D

6.1 Oraculo

The prediction mechanism chosen for the remote memory library is named Oraculo and is designed for file predictions using Tries[KL96][KL01] optimized for Grid environments. Oraculo is developed at Barcelona Supercomputer Center where I spent six months during which the initial integration of the remote memory library and Oraculo was done. Oraculo is an event-based prediction that predicts a series of events based on the previously recorded events. Oraculo provides two services which is 1) Adding events, 2) Getting an event sequence based on an event and the last sequence of events. The Trie used in Oraculo has a root to which the disjoint set of all events are attached as nodes. Each successor of the root is itself a root of what is denoted as a Trie partition.

Adding events The Trie is build from left to right and every time a new event is added a new partition is created, if no partition representing the event is present, at the partition root level. If a partition representing the event is present a counter within the node is increased to indicate the number of times the event has occurred. In addition to the new partition presenting the new event all previous partitions represents the previous event sequence and therefore the node is added to those as well. Figure 6.1 shows the process of adding the event sequence “A”, “B”, “A”, “D”. Firstly “A” is added as a new partition of the root, when the event “B” arrives it’s added as a new partition as well as added to

the original partition “A”, when the second “A” event arrives the first “A” node counter is updated as well as the last added node “B” is appended with a new node “A”. Lastly “D” arrives which is added to the original partition after the “A” which had just updated it’s counter from 1 to 2, it’s added to the “B” partition and finally a new partition “D” is created.

Pruning the Trie If all events are disjoint the first partition will consist of all the events E added to the Trie, the second will consist of E-1 and so forth until the E’ed partition which will consist of exactly the last event. Thereby a lot of redundant data is represented in the Trie. To reduce the memory consumption and the time traversing the Trie, the Trie depth can be reduced without loosing sequence information, but merely the number of times the sequence has occurred. Oraculo uses a Trie depth of three, which has proven sufficient for making good predictions. This, however can be fine tuned toward special needs as well as pruning the amount of partitions in the Trie and the amount of nodes within each partition is supported. Pruning the number of nodes in each partition is done when a new event is added to a partition with no room for additional nodes. Then all the node values are divided by two and the nodes with values below 1 is deleted, if no nodes have values below 1 the new node is not added, but the newly computed node values are kept and eventually nodes will be removed from the partition when new events arrive. Three schemes for pruning the partitions is available. 1) FIFO, removes the oldest partition, 2) LRU removes the least recently used partition. 3) LRU-USAGE selects the 30% least recently used partitions and removes the one with the lowest aggregated node values.

Predicting events When predicting future events, Oraculo is given an event sequence as input that is matched against the Trie by using each suffix of the provided input in the matching process. For example given the sequence “B”, “A” first B, A is matched against the Trie and then A is matched against the Trie. The result of the prediction will be “B”, “D”, this is sketched in figure 6.2. When the Trie grows from representing a few sequences, as shown in our example, and to a Trie that represents millions of different sequences a mechanism for eliminating all but the relevant predictions is needed. Oraculo keeps a counter at each node representing the number of times the event represented by the node is found in the current path of events. By introducing a probability threshold paths containing nodes below a given probability threshold are not taken into account when the prediction set is generated.

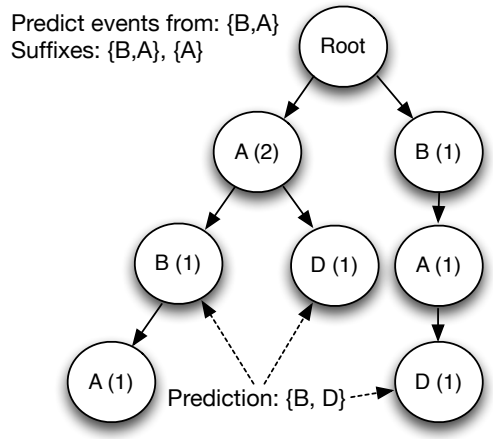


Figure 6.2: Oraculo predicting events based on the input sequence B, A

6.2 Combining Oraculo with The Remote Memory Library

The main purpose of the Remote Memory Library is to aid memory-limited Grid resources with additional memory through the Grid framework. Extending the remote memory framework with page-based prediction can be done either by performing the prediction at client or server side. The page prediction itself requires both memory and processing power and therefore it was chosen to perform the prediction server side as our focus is using the Grid infrastructure to get the highest possible performance out of the connected resources.

Prediction The free dictionary[Dic] defines the term 'prediction' as: "Something foretold or predicted; a prophecy", if one look up 'prophecy' it's defined as: "An inspired utterance of a prophet, viewed as a revelation of divine will". From this definition a prediction is eventually going to fail when used in computer environments because these don't comply with prophets or divine wills. This means that when trying to predict the pages that an application will be needing in the future it is bound to fail at some point. Based on that observation the overall design philosophy behind the memory prediction framework presented in this chapter is: "Always handle real page faults first" This means that if a real page fault occurs while handling predicted pages the real page fault is always handled immediately.

Server incorporation Oraculo has been incorporated with the memory server such that the page indexes requested by the client are added to Oraculo as events. When a

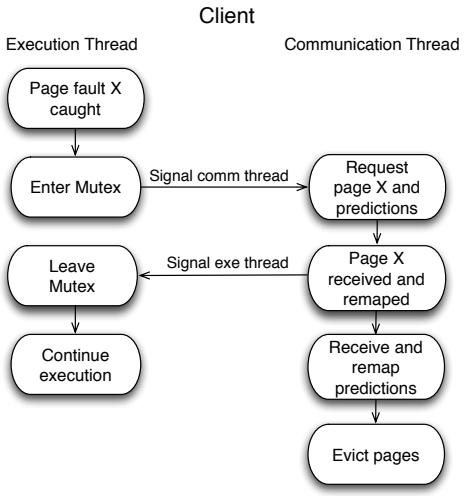


Figure 6.3: Receiving a faulting page and the corresponding predictions is handled in a separate communication thread

page request has been served and the page data is sent to the client, Oraculo is asked to make a prediction based on the faulting page index. The prediction returned by Oraculo is a list of page indexes which is sent to the client one at a time in a header followed by the corresponding page data.

Client incorporation At the client side the library has been modified to handle incoming predicted pages asynchronously to the computation threads. The communication between the execution thread and communication thread is done by synchronizing barriers. When the communication thread is idle it holds a barrier waiting for a page fault to occur, the barrier is reached when the execution thread faults and signals the communication thread to receive the faulting page. The communication thread then sends a page request message to the server, which responds with the page, adds the page fault event to Oraculo and makes a prediction that is sent to the client one page at a time. When the client communication thread receives the faulting page, it signals the faulting execution thread that the page has arrived. The execution thread then continues while the communication thread receives the predicted pages from the server. This is shown in figure 6.3.

Figure 6.3 is a simplified overview of the client design as the asynchronous page receiving requires several combinations of the presented synchronization mechanisms in order to ensure page consistency when having multiple threads causing multiple page faults either to the same page, to pages which are a part of a prediction, or to pages which are not present at the client nor in the pipeline of pages to receive. In addition to the barriers described above the synchronization mechanisms used in the original

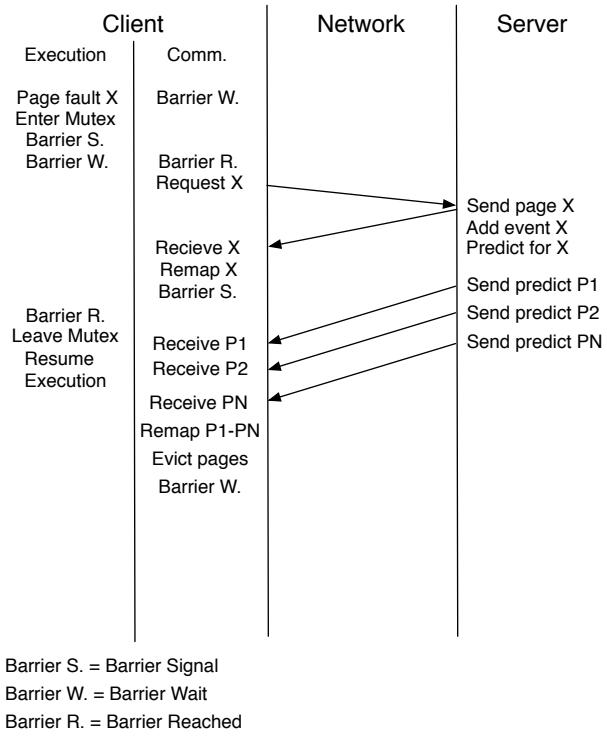


Figure 6.4: Barriers are used to synchronize between the execution thread and the communication thread

URSL framework is used. That is a global mutex for handling simultaneous page faults caused by multiple execution threads and the *mremap* routine for atomically remapping received pages into the right memory slot. A detailed execution flow using the synchronization methods described above is shown in figure 6.4 This figure shows how the execution thread accesses a page that is swapped out resulting in an access violation signal to be caught by URSL. The execution thread enters the page fault mutex to ensure that only one page fault is handled at a time, and meets the communication thread in a barrier. The communication thread then requests the page at the memory server that responds with the page data which is received and remapped by the communication thread, where after the faulting execution thread is met in a *page ready* barrier meaning that that the faulting page has arrived. The execution thread leaves the page fault mutex and continues execution while the communication thread receives and remaps the predicted pages until all are received. Finally pages are evicted in order to free up memory.

Predicted pages Predicted pages arrive at the client with a header that describes the page position in memory and page data which is received into a buffer reserved for

predictions. A page header is sent along with each page instead of just sending the complete list of predicted pages in advance. This is done to be able to interleave the data of real page faults with the predicted pages and therefore it's not possible to generate a page receive order in advance. To avoid wasting network bandwidth, the server keeps track of which pages are active at the client, and ignores these pages if they occur in the predicted page list. In addition to saving bandwidth, this also saves processing power at the client which would otherwise have to check the received predictions in order to avoid overwriting pages that are active in memory. When the complete list of predicted pages is received into the client receive buffer the pages are remapped into the right position in memory using the *mremap* routine. In an optimal scenario where every predicted page is needed in the near future, and the communication thread is capable of receiving and remapping predicted pages before they are needed, not only do we get the latency hiding of receiving the pages asynchronously to execution, we also get the advantage of not receiving page faults as the pages are already in place when they are accessed. Thus eliminating the kernel-level overhead of trapping the signal and the user-level overhead of catching and processing the signal. Unfortunately the world is not perfect and therefore scenarios will appear where page faults arrive while the communication thread is busy handling predictions initiated by an earlier page fault.

Page faults while predicting When a page fault occurs while the communication thread is receiving predicted pages from the server, an interrupt flag is raised by the execution thread in order to tell the communication thread that a new page fault has occurred. Thereafter the execution thread enters a waiting barrier until the communication thread signals that the page causing the fault has been retrieved and remapped. The communication thread checks the page fault interrupt flag between each retrieval of predicted page data. If the interrupt flag has been raised the communication thread handles the interrupting page fault and then continues the retrieval of predicted pages. The communication thread needs to take several scenarios into account when a predict interrupting page fault occurs, these are listed below:

- Faulting page already received and remapped
- Faulting page already received, but not yet remapped
- Faulting page in predicted set, and sent by server, but not yet received by client
- Faulting page in predicted set, but not yet sent from server
- Faulting page is not in the set of predicted pages

Page already received and remapped This is the simplest of the page fault interrupt scenarios as the faulting page was predicted by the server and thereafter received and

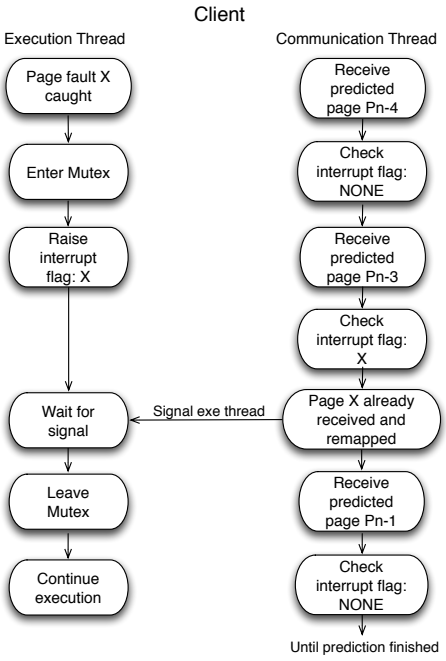


Figure 6.5: A page fault arrives while predicting, but the faulting page has already been received and remapped as a part of the active prediction

remapped by the client immediately after the page fault occurred in the execution thread. In this case the communication doesn't need to contact the memory server but merely need to signal the execution thread that the page is swapped in and the execution thread can continue with its computation. This is shown in figure 6.5

Page already received, but not yet remapped This situation is similar to the above, except that the page has only been received and not yet remapped by the communication thread. Therefore the communication needs to remap the page into the right virtual memory slot before signaling the main thread that the page is swapped in and ready to use.

Page in predicted set, and sent by server, but not yet received by client In the two previous scenarios the faulting page was already received by the client as a part of an active prediction and thereby the communication thread didn't need to contact the memory server in order to handle the page fault. On the contrary this scenario requires the communication thread to contact the memory server, as the only information the client has about the page is that it's located at the memory server. The server receives the predict interrupting page request and finds that the page has already been sent to the

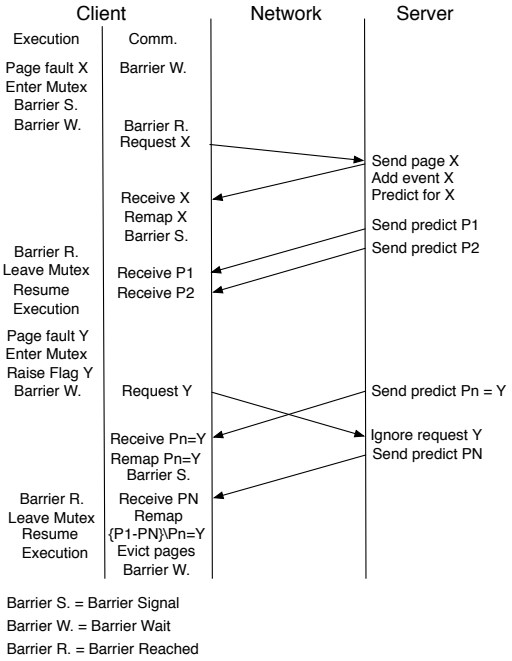


Figure 6.6: A page fault arrives while predicting, a request is sent to the server which finds that the page has already been sent to the client

client as a part of the predictions currently being processed. Thereby the server merely ignores the request and continues sending the predicted pages it was processing when the interrupt page fault arrived. The client will receive the faulting page in the sequence of predicted pages unaware whether the server predicted it or responded to the issued page request. When the faulting page is received it's remapped into the right virtual memory slot and the a signal is sent to the faulting execution thread that the page has arrived and execution can continue. This is shown in figure 6.6

Page in predicted set, but not yet sent from server From the perspective of the communication thread this is the same scenario as the one described above, the page is not yet received from the server and therefore a page request is sent. The server receives the request and finds that the page is in the list of predicted pages but has not yet been sent to the client. The server then sends the page immediately and removes it from the list of predicted pages.

Page is not in the set of predicted pages This scenario is handled like the above, except that the page is not in the predicted set and is therefore not removed from the predicted list after it has been sent. When the server receives the page request the

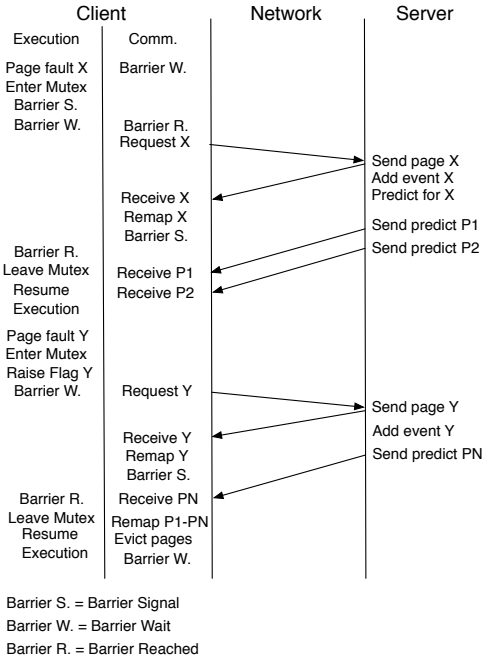


Figure 6.7: A page fault arrives while predicting, a request is sent to the server which finds that the page fault is not in the predicted set and interleave the requested page in between the predictions

page data is sent in between the predicted pages such that the page request is handled immediately. This is shown in figure 6.7

Evicting pages In order to enforce the upper memory limit of the executing application, pages need to be evicted when allocated memory is activated or pages are retrieved from the memory server. Ideally eviction can be done asynchronously to the execution thread but this requires the time gap between page faults to be large enough to hold the time spent receiving predictions and evicting pages, see figure 6.8. When predicted pages are remapped into the correct virtual page slot the state of the page in the local page table is modified, likewise the state of evicted pages is changed after eviction. These actions are performed by the communication thread that in order to modify the page table needs to enter the page table mutex. The execution threads queue up for the page table mutex if it's busy when they need to enter it, this is not possible for the communication thread as it will result in a deadlock. This is shown in figure 6.9. If the mutex is available the communication thread remaps the predicted pages and evicts pages to the server, otherwise an evict starvation flag is raised and the remap/eviction action is postponed until a page out fault occurs, as the mutex will then be acquired

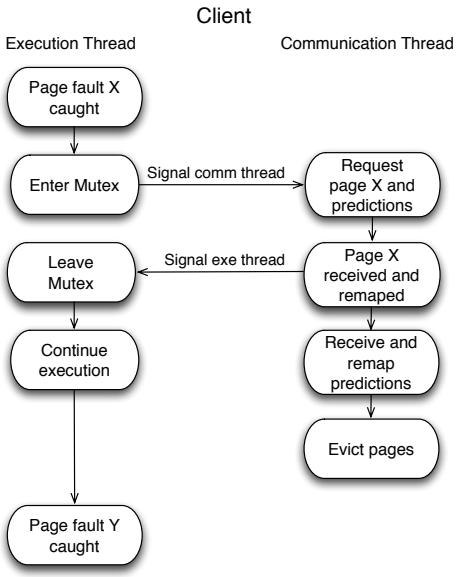


Figure 6.8: The execution time after page fault X is large enough for the predictions and eviction to complete before the page fault Y arrives

by the faulting execution thread. This is shown in figure 6.10. While this solution is deadlock safe with respect to remapping predicted pages and evicting pages it also introduces a possible bottleneck. This is because the mutex is not solely entered as a result of page faults caused by accessing swapped out pages. The access violation could be caused by writing to a clean page that is in mode *read* or a newly allocated page which have not been referenced yet, this could result in the communication thread idling until the next page fault caused by access to a swapped out page occurs, even though the starvation flag is raised. However this is a necessary cost as exclusive access to the local page table is required whenever the page states are changed. Several more fine grained mutex strategies exist, such as letting the communication thread continuously try to obtain the page table mutex when the starvation flag is raised. This, however will impose a significant overhead compared to the used strategy of waiting until the next swapped out page fault occurs. Another strategy could be using page-based mutexes instead of one global mutex for all pages. This, however would impose a significant space and time overhead as mutexes consume memory and entering them impose synchronization overheads.

Communication thread overview With all the URSL communication thread details covered, a complete overview can be found in figure 6.11.

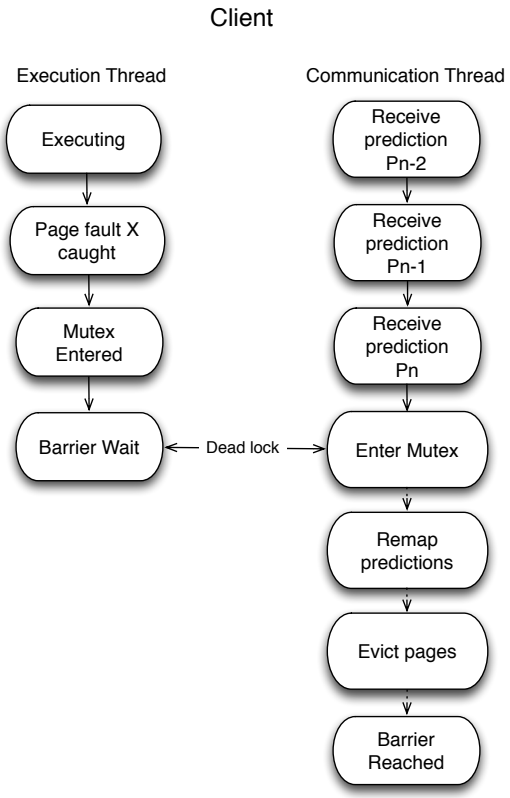


Figure 6.9: A page fault arrives after the last predicted page has arrived at the communication thread, but before the remap is done. The execution thread enters the mutex and waits for the page ready barrier to be reached. If the communication thread is put in the queue for entering the mutex a deadlock will occur as the execution thread holds the mutex and waits for a signal from the communication thread, which will not arrive before the communication thread enters the mutex

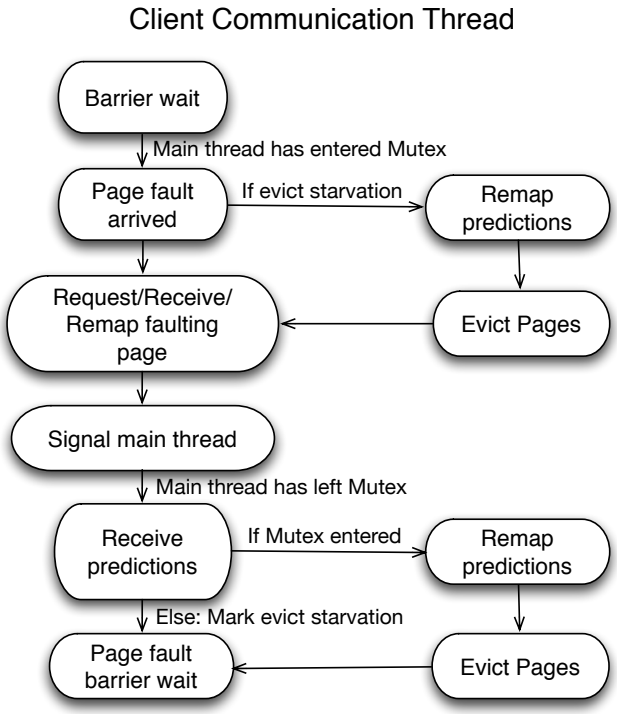


Figure 6.10: When the predicted page data is received from the server, the mutex is entered if possible, and the predicted pages are remapped whereafter pages are evicted to the remote memory server. If the mutex can not be entered, a starvation flag is raised and the page remapping and eviction is performed when the next page out fault arrives, as the faulting execution thread will then enter the page table mutex

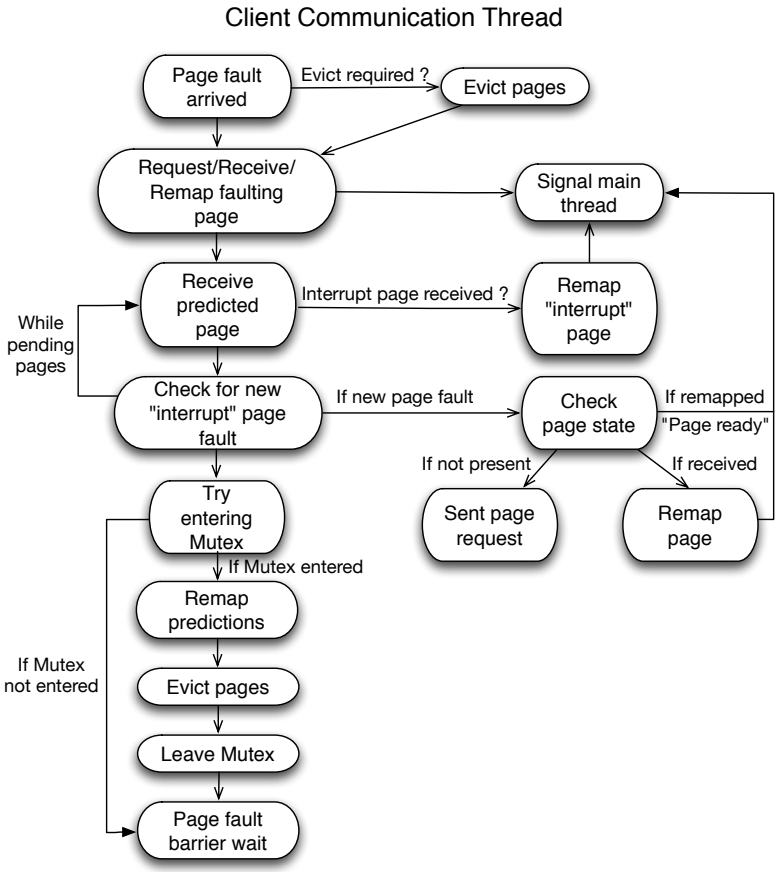


Figure 6.11: The client communication thread

6.3 Online Prediction Problem

Oraculo uses event-based prediction, we base our events on the request of faulting pages that arrives at the memory server. To get the most accurate prediction it would be preferable to base the events on the memory access pattern of the executing application. This, however is not feasible due to the enormous amount of events that would be generated. In order to make accurate and efficient predictions a relatively small number of representative events should be used. Furthermore the fact that the page referenced bit is not available from user-level means that in order to record the memory access pattern of the executing application every page would have to be protected in *no-access* mode meaning that every page access would result in a page fault to be handled by the framework. This would degrade performance to an extent where the framework would be useless.

Because we don't have any information about the referenced pages it's not possible to make feedback to the remote memory server about which of the predicted pages were actually used, and which were just discarded during an evicting without ever being used. The result is that a successful prediction can actually destroy the event sequence of future events. Take for example the page access sequence "a, b, c, d, e, f" where "a, b, c, d, e" is recorded by Oraculo. Lets say page "a" faults and is requested at the memory server, Oraculo then returns "a" and along with "a" the predictions "b, c, d, e". The predicted pages are received and remapped before they are accessed and thereby the next page fault that occurs is when accessing page "f". This event is added to Oraculo and eventually if "b, c, d, e," is predicted every time page "a" faults the probability of this prediction is degraded, as the new event sequence when faulting "a" will be, "a, f" as "f" is the page faulting after "a". This means that a successful prediction actually messes up the previous measured events, this can not be solved as long as there is no feedback to the predictor from the client.

6.4 Experiments

Like the initial URSL experiments, the experiments performed using predictions were made in an isolated execution environment using a Playstation® 3 and a memory server interconnected through a 1 Gb/s switch. The memory server, however, was upgraded between the two experiments to a duel core Intel Xeon running at 1.88 GHz with 32 GB of memory. The Playstation® 3 has 224 MB memory of total memory for the OS and applications and it's observed that the executing application is using approximately 175 MB of real memory when swapping to disk. In order to avoid that the machine used for the experiments runs out of real memory, the real memory limit in URSL is set to 128 MB. URSL in itself uses 16 MB for the local page table and 4 MB for the buffer used to store predicted pages before they are remapped into their correct virtual memory slots.

This is in total 148 MB of real memory, which should be a sufficient limit in respect of the total 175 MB memory in order to avoid that the OS kills the experiment due to an out of memory error. In the performed experiments the Oraculo system is configured not to use partition pruning and node pruning, as the current implementation of Oraculo is made in Java and the pruning of the Trie will result in the Java garbage collector to be initialized arbitrarily. Because it can not be controlled by the system designer this may cause two similar executions to provide ambiguous results. The Trie depth however is enforced to be no deeper than three as the time adding events and searching for them will otherwise increase beyond the reasonable. Furthermore Trie depth pruning doesn't need to involve the Garbage collector as the memory of a removed node is reused for the arriving event that caused the pruning. In a production system one could port Oraculo to C or C++ and then free up memory of the pruned partitions whenever this is suitable with respect to the execution flow between the remote memory server and the URSL client. To compare the performance of URSL with and without prediction the initial experiments were repeated using the new framework with predictions, the result of the experiments is described in the following sections.

6.5 Sequential Data Access

Similar to the sequential access experiment presented in the URSL chapter 5, the execution was performed by allocating N bytes of memory using malloc and then initializing the memory to make sure it was mapped to physical memory. After memory initialization a timer was started and the time spent traversing the memory start-to-end in 10 iterations reading each byte to produce a checksum was measured. This test is highly I/O bound as the only computation done by the executing machine is one integer addition per byte that is read. The result of this experiment using prediction is shown in figure 6.12. In this experiment the prefetching version using Oraculo outperforms the original URSL framework by a factor of 2.5 when using a page block size of one, but when the page block size of the original framework is raised to 64, where the network bandwidth is saturated, the framework using the Oraculo predictor is outperformed by a factor of 2.2. Using naive prefetching by increasing the page block size till the network bandwidth is saturated results in optimal prefetching when the running application is purely sequential, because every prefetched byte is used by the running application. In addition the naive block prefetching doesn't consume processing power, but most importantly the pages prefetched are used by the application without causing any page access violations. Finally the simple URSL framework without predictions uses a single thread model where the Oraculo prediction model uses a communication thread to receive predicted pages from the memory server. This in itself consumes time due to context switching and synchronizing barriers which need to be met in order for the communication thread to communicate with the faulting execution thread. To see how close

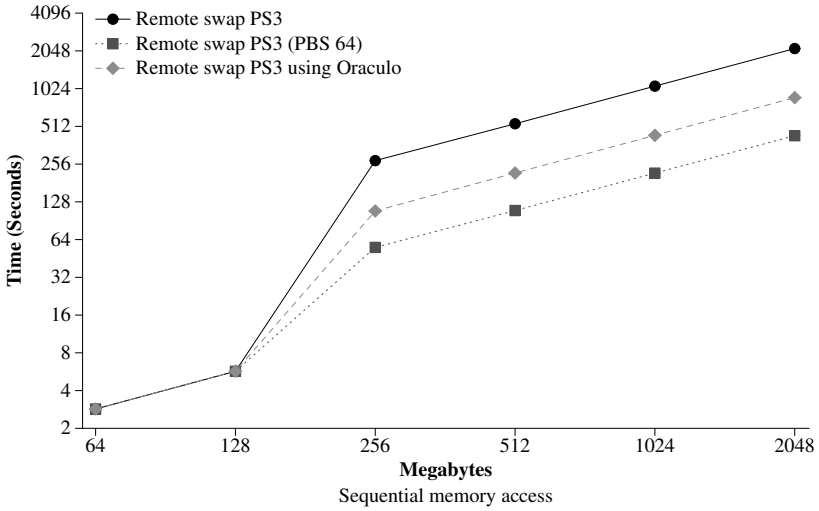


Figure 6.12: Performance of sequential memory access using prediction

the Oraculo prediction scheme could get to the naive block prefetching with a highly I/O bound application we repeated the experiment using a learning iteration in order to feed Oraculo with the page fault sequence before measuring the time of executing the 10 iterations. Thereby Oraculo has knowledge of the page sequence before the time measurement begins. The result of this experiment is shown in figure 6.13. Using the learning iteration lowered the gap between using Oraculo prefetching and optimal naive block prefetching, but still the optimal naive block prefetching is faster by a factor of 1.4, due to the overheads of catching page faults using a communication thread and actually performing the predictions.

6.6 Sequential Data Access with Writes

In this test we used the sequential test described in the previous section and performed a write to each page accessed after its data had been read. The result is shown in figure 6.14. This experiment reveals that when performing a write to every page read, the naive prefetching using a page block size of 64 still performs better than the Oraculo prediction based solution. However the performance gap between the optimal naive and the prediction based solution is smaller than in the last experiment where no pages were written back to the server, namely a factor of 1.6. In this test each page read is modified which means they are written back to the server when evicted from real memory. The prediction based solution have two advantages over the original version, namely that pages are evicted asynchronously to execution if possible, however with the amount of computation performed in this experiment the most likely course is that

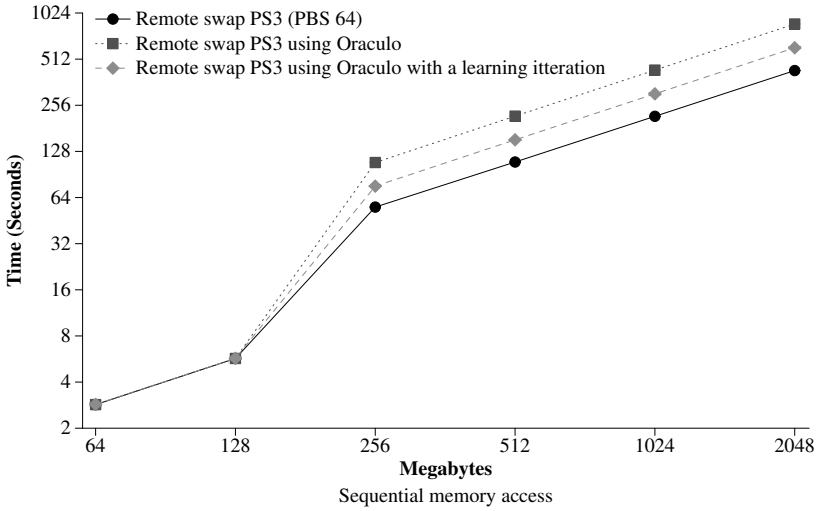


Figure 6.13: Performance of sequential memory access using prediction with a learning iteration

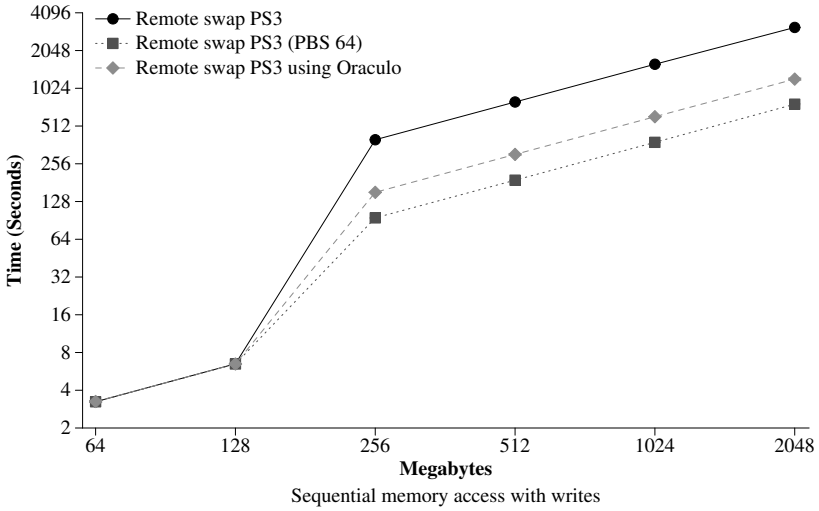


Figure 6.14: Performance of sequential memory access with writes using prediction

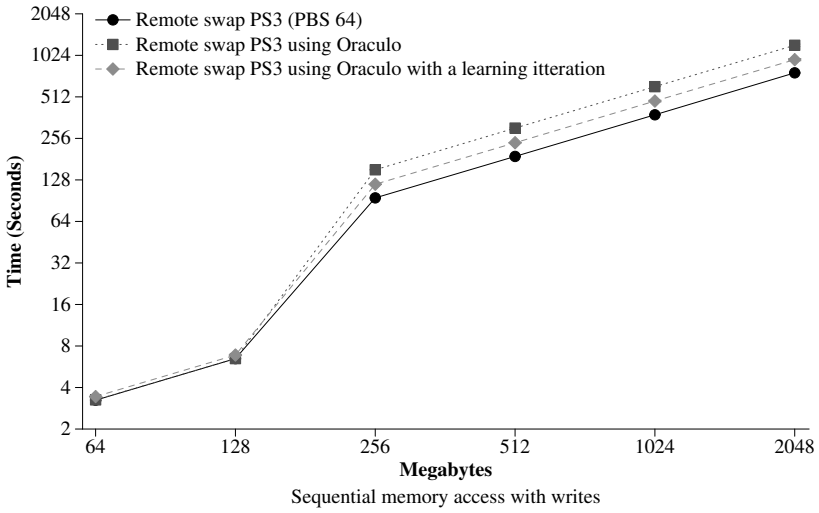


Figure 6.15: Performance of sequential memory access with writes and a learning iteration using prediction

evictions are bulked, meaning that when the predicted pages have been received they are remapped and first then pages are evicted. This means that if for example 128 pages were received as a part of a page fault and the following predict, then 128 pages need to be evicted in order to free up memory for the arriving pages. Because we need to evict 128 pages at once they can be bulked into one evict and thereby only one write is issued for all of the 128 pages. Like with the sequential experiment we remade the experiment using a learning iteration before measuring time over the 10 iterations. The result of this experiment is shown in figure 6.15. Like the sequential experiment without writes the execution with a learning iteration decreases the gap between the naive page block prefetcher and the Oraculo based prediction, but still it doesn't outperform the optimal naive page block prefetcher which is faster by a factor of 1.2.

6.7 Scattered Memory Access

This test was designed similarly to the sequential access test, but instead of reading the pages in a sequential manner the data is read one page at a time, starting with the first page followed by the last page and then the second page followed by the second last page, this pattern is used until all pages are read. The result is shown in figure 6.16. Like the experiment using sequential access, the prefetching version using Oraculo outperforms the original URSL framework by a factor of 2.4 when using a page block size of one, but when the original framework uses the optimal naive prefetching with

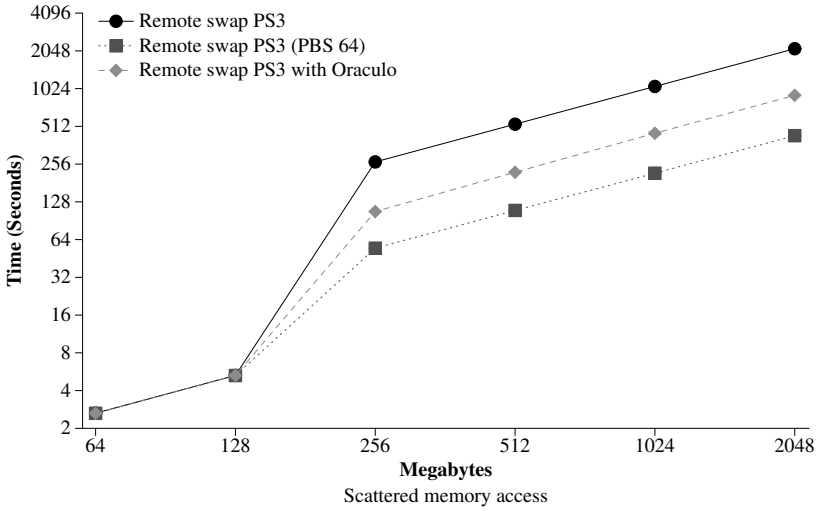


Figure 6.16: Scattered memory access performance using prediction

a page block size of 64 the Oraculo version is outperformed by a factor of 2.1. The introduction of a learning iteration shows the same pattern as with the sequential data access and the prediction based prefetching is outperformed by the optimal naive page block based prefetching by a factor of 1.5. The result is shown in figure 6.17. Actually the prediction based prefetching performs a bit worse compared to the sequential version, which indicates that the predictor is not making the correct predictions even with a learning iteration. This indicates that the online prediction problem has an effect in this experiment where pages are accessed in an uneven way.

6.8 Scattered Memory Access with Write

This experiment is performed like the above scattered experiment, but with a write to each page like in the *sequential with write experiment*. The result is shown in figure 6.18. Like in the *sequential with write experiment* the performance of the Oraculo predicted prefetching performs better than in the scattered experiments without write, but it still doesn't outperform the initial naive page block prefetching which is faster by a factor of 1.7. The same is the case if we use a learning iteration, performance gets better, but doesn't outperform the naive page block prefetching which is faster by a factor of 1.3. This is shown in figure 6.19.

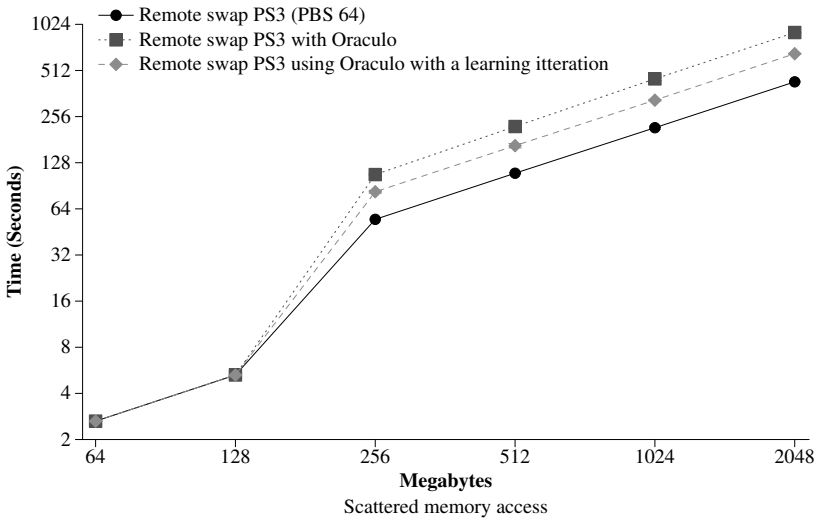


Figure 6.17: Performance of scattered memory access using prediction with a learning iteration

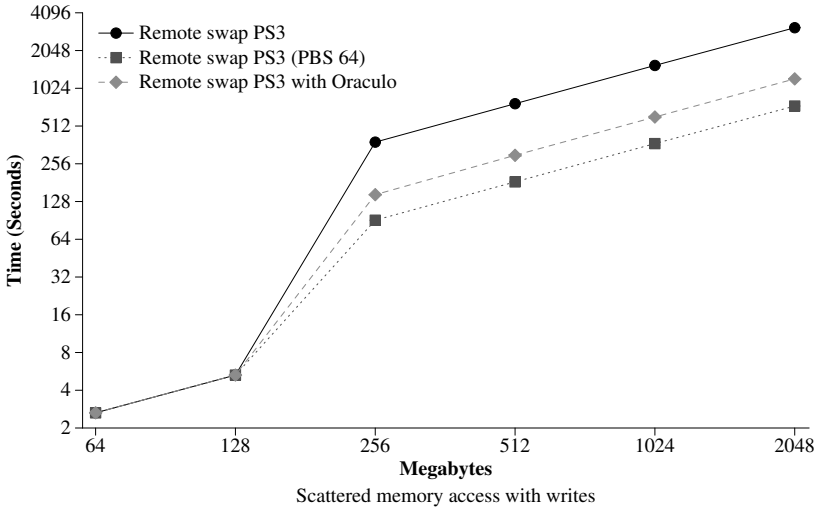


Figure 6.18: Performance of scattered memory access with writes using prediction

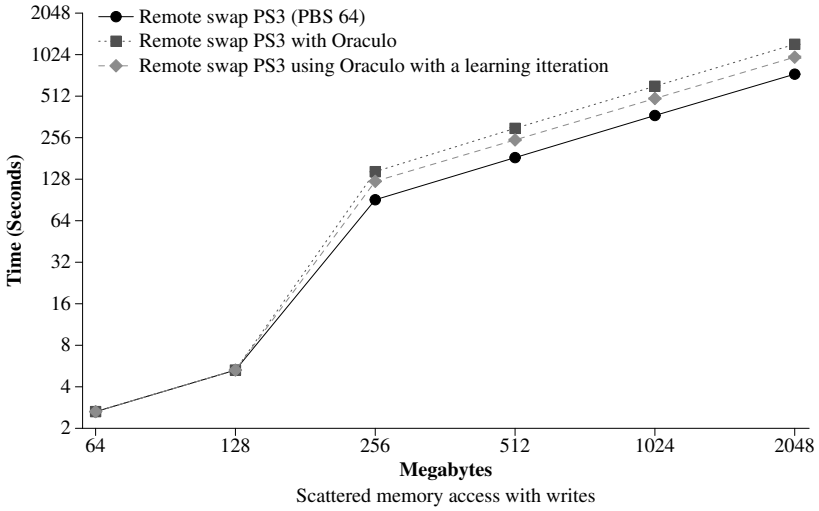


Figure 6.19: Performance of scattered memory access with writes using prediction and a learning iteration

6.9 Lattice Boltzmann

The Lattice Boltzmann experiment performed in chapter 5 was repeated using the Oraculo prediction prefetching, the result of this test is shown in figure 6.20. This experiment reveals that the Oraculo based prediction is a factor of 1.3 faster than the optimal naive prefetching using a page block size of 4. Lattice Boltzmann is an iterative application and therefore a page fault pattern can easily be obtained using a learning iteration. The experiment was repeated using a learning which showed no significant speed up. This result is shown in figure 6.21. The reason why the learning iteration didn't provide any significant speed-up is because the application have a non-sequential memory access pattern that causes the pages to be evicted and retrieved several times during one iteration. Thereby the predictor is capable of providing correct predictions before the first iteration of the application is done. A more fine grained definition of the learning iteration than merely using what the application defines as the first iteration is needed if a learning iteration is to be useful in this scenario.

6.10 Fast Fourier Transform

The fftw experiment performed in chapter 5 was repeated using the Oraculo prediction prefetching, the result of this test is shown in figure 6.22. This test reveals that the Oraculo prediction based prefetching outperforms the naive page-based prediction though only slightly namely by a factor of 1.2. Because fftw is not an iterative ap-

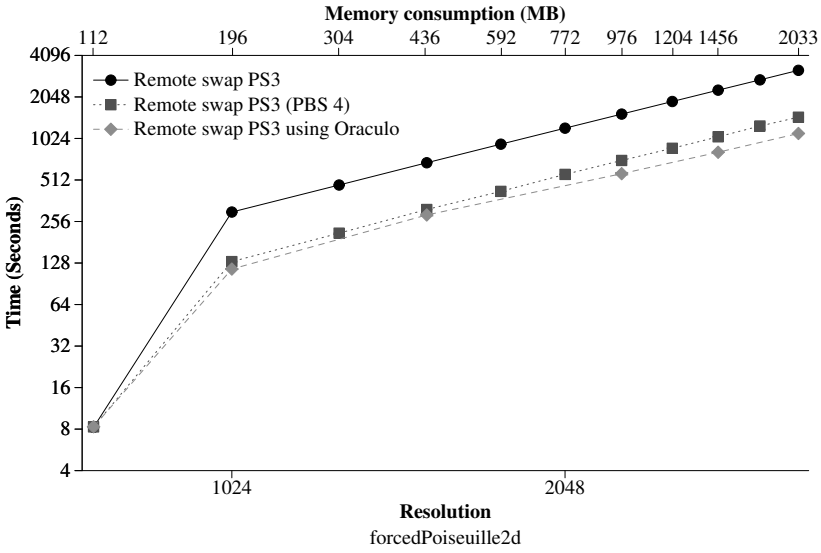


Figure 6.20: Lattice Boltzmann performance using prediction

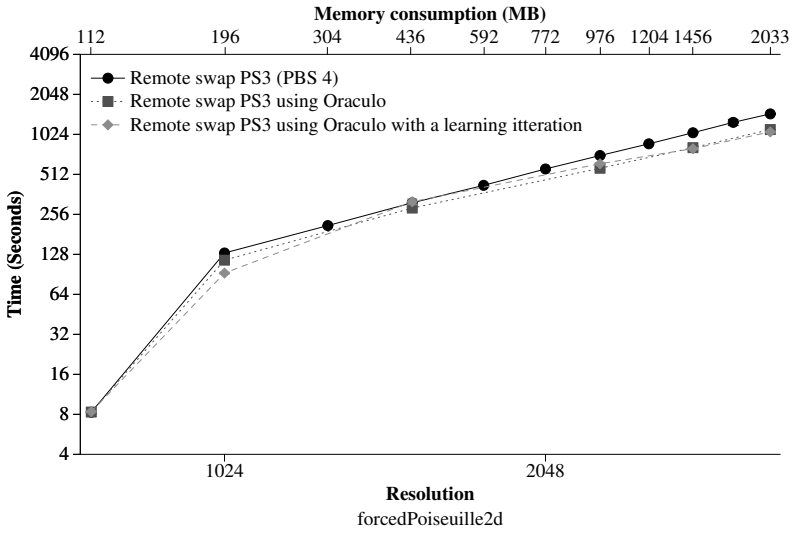


Figure 6.21: Performance of Lattice Boltzmann using prediction and a learning iteration

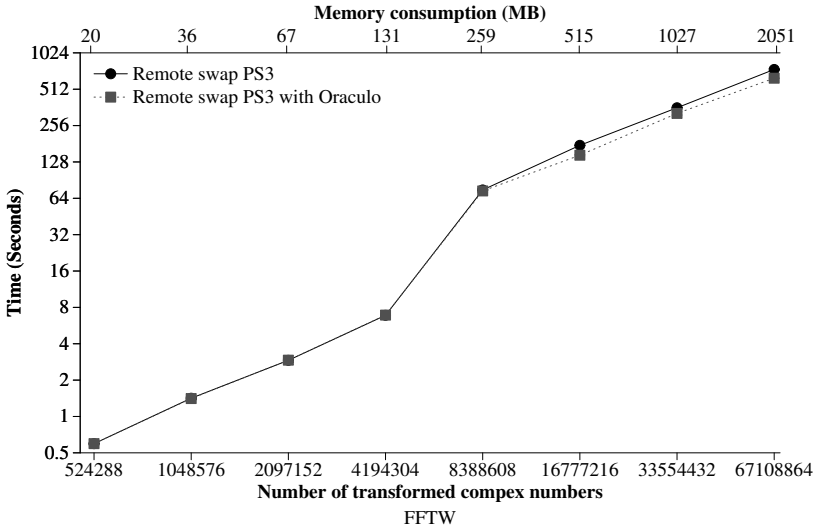


Figure 6.22: Fftw performance using prediction

plication Oraculo is having a hard time learning the event sequence of faulting pages. Therefore we tried to modify the type of events added to Oraculo such that instead of using the absolute page indexes as events we used the delta indexes. That is the distance $\delta = X_n - X_{n-1}$ Where X_{n-1} is the page index of the last faulting page and X_n is the page index of the current page fault. Thereby we get an memory access sequence based on the variety of pages instead of the absolute values. Because a lot more similar events will happen e.g. the delta distance of 1 is much more likely than the absolute page index X the probability threshold used by Oraculo was lowered from 0.5 to 0.01 as initial tests showed that this value was the optimal regarding the hit rate of the received predicted pages. The result of the fftw experiment using delta prediction with a probability threshold of 0.01 is shown in figure 6.23. This experiment shows that using predictions based on the delta value instead of the absolute page indexes halves the execution time of fftw which means that the Oraculo prediction based version outperforms the naive page block prefetching method by a factor 2.

6.11 Barnes-Hut

The Barnes-hut experiment is performed like in chapter 5, and the result is shown in figure 6.24. It's seen that the naive page block prefetching algorithm outperforms Oraculo prediction by a factor of 1.3 when using a page blocks size of 8 in the naive prediction. Delta prediction was tried with this experiment, but didn't improve performance. The Barnes-Hut application has long execution time and therefore only one step of the sim-

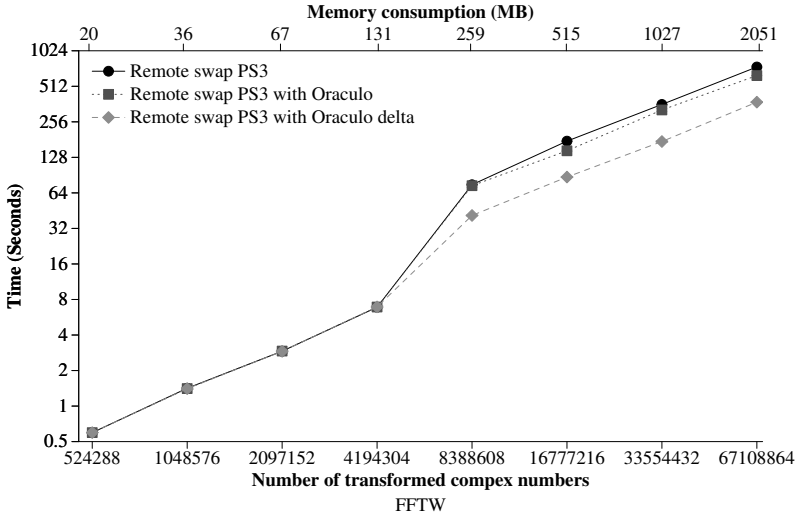


Figure 6.23: Fftw performance using delta prediction

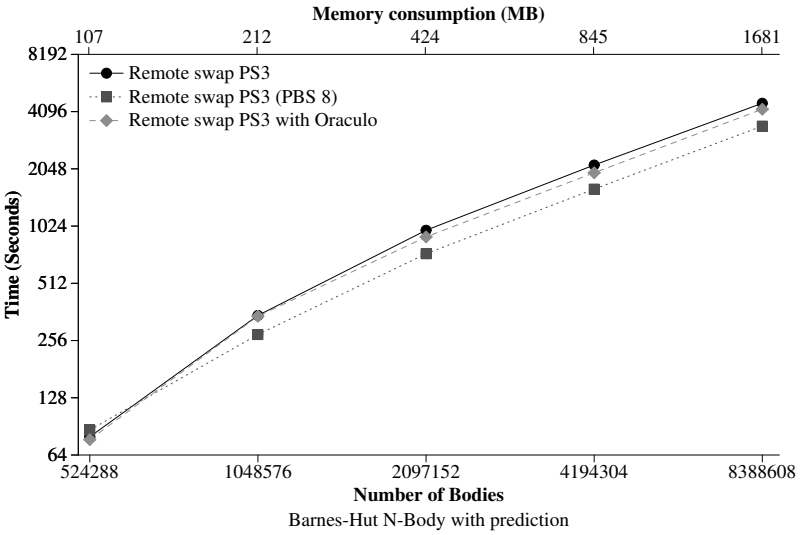


Figure 6.24: Barnes-Hut performance with prediction

ulations was performed in this experiment. Further analyses of the Barnes-Hut memory access pattern is needed to fine tune the prediction based prefetcher towards this application, which is a subject for further investigation.

6.12 Simulated Network Latency

All the above experiments were performed in a low latency high bandwidth network in order to see how the presented solutions perform at their optimal conditions. However in order to use the presented framework in a real Grid environment latency has to be taken into account. Therefore support for simulated higher latencies was implemented into the server part of the framework. When the server receives a request for a page, it sleeps an number of milliseconds (*t_latency*) to simulate round-trip latency before sending the data of the faulting page to the client. Predictions are thereafter sent without any latency, as they are written to the network as successors to the faulting page, and thereby no extra latency is required. If an interrupt page fault arrives at the server while sending predictions, the *t_latency* milliseconds are simulated by starting a timer continuing sending predictions until the *t_latency* milliseconds have passed by. Then the interrupt page fault is severed whereafter the server continues to send the remaining predicted pages. This is done to simulate a realistic scenario where predicted pages are sent to the client while the interrupt page request is on its way through the network. If the transmission of a prediction finishes after an interrupt page has arrived, but before the *t_latency* milliseconds have passed, the remaining milliseconds is slept before handling the request. Additional latency is not put on page evictions as evicted pages are sent without blocking from the client and thereby will be received at the server before the next page fault request arrives. Thereby the evict latency is implicitly applied by the simulated latency associated with the successive page fault.

To verify the effect of higher latencies, the above experiments were repeated using the server with simulated latency set to 40 ms which corresponds to the latency between my home ADSL line and the university network where the memory server is located. The sequential memory access used as the first experiment was executed with these new settings. This is an I/O bound test that is easy to predict due to the fact it's purely sequential. The result of this experiment is shown in figure 6.25. This experiment shows that the original framework with the naive page block prefetching outperforms the Oraculo based prediction without learning by a factor of 6, this is due to the amount of page faults which occurs within the first iteration before Oraculo learns the page sequence. For example in the execution using 2048 MB of memory 524288 page faults occur in each iteration which each have a round-trip latency of 40 ms. This gives 20972 seconds of accumulated latency just waiting for responses on page requests. In the naive page block prefetching strategy using 64 pages per request the accumulated latency in each iteration is 327 seconds as the amount of page faults is lowered by a

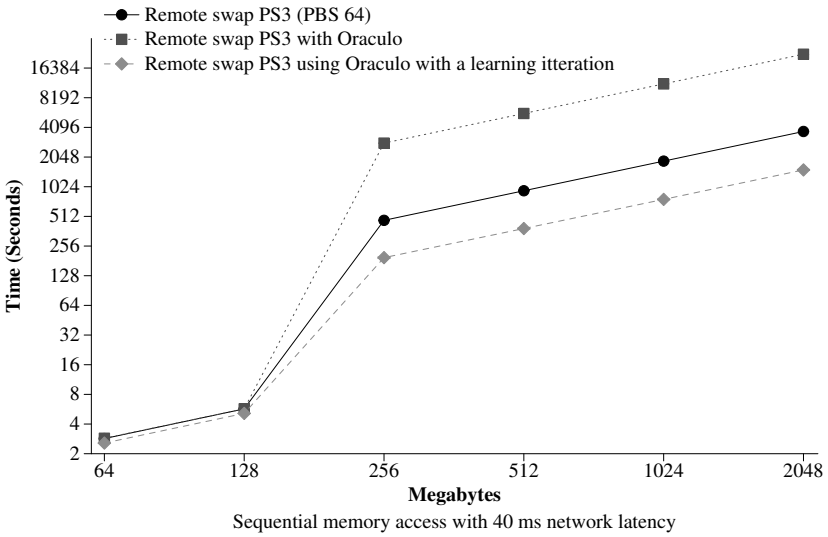


Figure 6.25: Performance of sequential memory access using prediction

factor corresponding to the amount of pages sent per request. In order to clarify the performance obtained if the predictor has knowledge of the page sequence in advance we used a learning iteration before starting measuring execution time. This resulted in the predictor outperforming the naive prefetching method by a factor of 2.5 due to the asynchronous nature of the version using Oraculo prediction. That is, the predicted pages are sent while the execution thread performs it's work and while interrupting page faults occur. The result is that the latency of predicted pages are hidden as they are streamed to the client while the application performs execution whereas the naive method pays the latency for every page fault. This, however, depends highly on the success of the predictor and the success of using a learning iteration. In effect the sequential, the scattered and the Lattice Boltzmann experiments are very successful with high latencies and a learning iteration. Performance of the fftw and Barnes-Hit experiments in the simulated latency environment is not presented, as they didn't terminate within the experiment time frame. The reason is that they are not iterative and thereby a learning iteration is not easily obtained. Effectively providing meaningful page fault learning sequences for such applications are subject for further research. The Lattice Boltzmann experiment in a high latency, high bandwidth environment on the other hand was very successful with a speed-up of 10 at the most memory consuming experiment and the gap between the naive page block prefetching and the prediction based prefetching is growing as the memory consumption grows. The result of the Lattice Boltzmann experiment using simulated latencies is shown in figure 6.26.

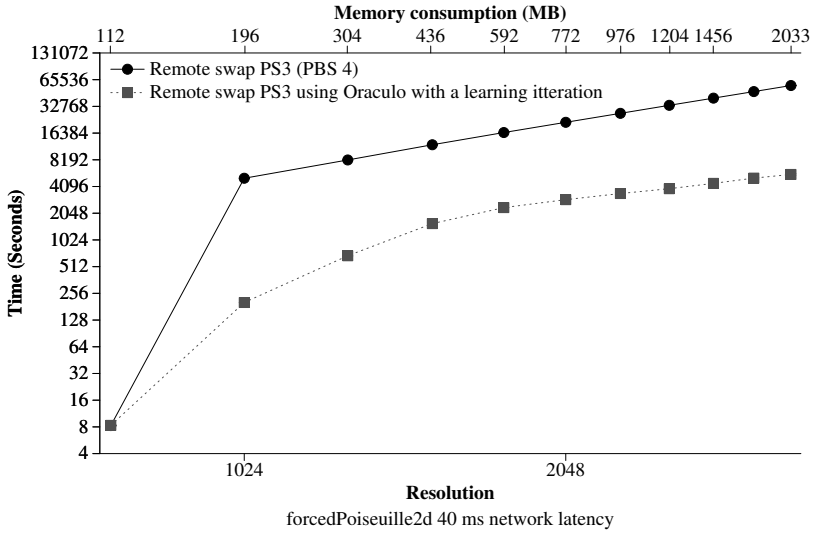


Figure 6.26: Performance of scattered memory access with writes using prediction and a learning iteration

6.13 Experiment Summary

The experiments made in chapter 5 were re-executed using the presented prediction based prefetching and the performance was compared to the performance obtained using the optimal page block sizes for each experiment in the original URSL framework. In a low latency, high bandwidth network setup the naive page block prefetching method outperforms the prediction-based framework when performing the highly I/O bound experiments, namely sequential, sequential with write, scattered and scattered with write. When it comes to the real scientific applications, the Lattice Boltzmann experiment using the prediction based framework outperforms the naive page block prefetching scheme with a factor of 1.3. Lattice Boltzmann is an iterative scientific application traversing the data set over and over meaning that the page fault sequence can be easily obtained, however in each iteration the memory pages are received and evicted several times and therefore a learning iteration didn't provide any significant speedup in the low latency, high bandwidth scenario as pages are already correctly predicted and prefetched in an early state of the iteration. A more fine grained learning iteration is to be defined in order to make a learning iteration useful in this scenario.

The fftw application is non-iterative and therefore there is no novel way of providing a page fault sequence in advance, instead we tried to use the delta value of page faults instead of real values. Using absolute page indexes the naive page block prefetcher is outperformed by a factor of 1.2, and when using delta indexes for predictions the performance is increased by a factor of two outperforming the naive page block prefetcher

with a factor of 2. The Barnes-Hut experiments didn't prove any significant speed-up compared to the naive page block prefetcher, to improve this a closer study of this applications page access pattern is required.

The Oraculo predictor was on par with the naive page block prefetcher and the delta prediction provided no significant improvement.

Moving the experiments from a low latency, high bandwidth network to a high latency, high bandwidth network was done by introducing simulated latencies to the framework. The sequential memory access experiment were re-executed introducing a 40 ms simulated round trip latency. This experiment shows that without a learning iteration the naive page block prefetcher outperforms the prediction based prefetching scheme due to the amount of page faults occurring in the first iteration until the predictor learns the page fault sequence. When using a learning iteration before starting time measurements the prediction based prefetcher outperform the naive page block prefetching scheme with a factor of 2.5 The sequential with write, scattered memory access and scattered memory access with writes were not measured but expected to show similar results as the previous experiments with these applications showed that they behaved similar to each other. The Lattice Boltzmann experiments showed really good results using the simulated high latency network when the predictor was provided with an initial page fault sequence, namely by outperforming the naive page block prefetcher with a factor of 10. The fftw and Barnes-Hut were not applicable in the simulated high-latency scenario and due to their non-iterative nature a proper learning execution method was not found.

The results show that using a page-based prediction system performs very well in a high latency high bandwidth environment compared to the naive page block prefetching mechanism when it comes to iterative applications with an initial page fault sequence obtained in advance. In the experiments performed the prediction based prefetching method didn't perform well with the non-iterative applications, specially not in the high latency scenario where the experiments were dropped due to the long lasting executions in each experiment. This is regarded to be a matter of defining new ways of gathering events for the page predictor or a matter of pre-recording the page fault sequence of these executions and saving them for further use. A method for effectively making learning executions for arbitrary applications and providing them effectively to the execution resource through the Grid environment is subject for further research.

Chapter 7

Future work

The research presented in this dissertation approaches Grid-enabling non-Computer resources for scientific computing. During the first part of the project we found that memory shortage was one of the major issues when connecting non-Computer resources to Grid infrastructures and therefore the last part of the project is concentrated on providing memory to Grid resources through the Grid infrastructure. The presented research covers the design, implementation and analysis of a frame work that is still in it's early state and therefore there are several possibilities of improvement. In this chapter we will sum up the open ends this dissertation leaves and suggestions for further progress within the topic.

With respect to prediction based prefetching, the sequence of events used for predicting future events is a subject for further research. In this dissertation the page fault indexes or their delta's are used, but it might be that there are more efficient ways of representing page fault events regarding predicting future events. Another research topic which is left open is finding a general way of obtaining learning sequences of the executing applications through the Grid framework automatically. That is the event sequence is recorded and adjusted by the Grid middleware in order to provide an optimal starting sequence which can be provided along with the Grid job when starting a new execution. The high latency, high bandwidth experiments revealed that a good starting event sequence is crucial for deployment in real Grid environments regarding performance.

In addition to the event sequence optimizations a future topic is making the memory servers an incorporated part of the Grid infrastructure, so that when a Grid job is submitted to the Grid infrastructure, the scheduler decides which resource should execute the job based on the requirements of the job. If it turns out that the most suitable resource is one which doesn't have sufficient memory to execute the application, the Grid infrastructure could use the remote memory framework to provide the additional memory to the resource. In order to minimize latency a memory server close to the resource and with enough free memory should be used. In addition multiple memory server could

be connected in a raid-0 or raid-1 fashion in order to boost either bandwidth, reliability or both using raid-10. Furthermore support for automatic checkpointing could be embedded into the framework. The time transferring memory and execution states to a remote location is often a show-stopper when it comes to checkpointing, but when using a remote memory framework the majority of the memory in use by a running application is already located at the a memory server in the Grid framework. Performing a checkpoint will then require much less effort than in the usual case where the whole memory footprint is located on the client at the time of a checkpoint.

Finally an interesting project could be to incorporate the remote memory framework into a virtual machine such as VirtualBox[Wat08]. By embedding the framework in the virtual machine neither the guest operating system, nor the guest applications will ever be aware of the fact that memory is transferred to a remote destination. This might prove to perform better than the current system as the overhead of catching and handling page faults at user-level might be reduced if not removed depending on the design and implementation chosen.

Chapter 8

Conclusion

Non-Computer resources are a broad term for a variety of different computer devices. In the presented work we focused on Grid enabling non-computer devices for scientific applications. Firstly "The One-Click Grid Resource Model" was presented which demonstrated a framework for connecting all Java capable Internet devices, with a minimum amount of work both from the Grid administrators and the resource owners. This model requires existing Java applications to be modified to use the framework, however the modifications are minimal, as it's only a question of calling a different Java main method, and using a customized set of operations for stdout/stderr and file-access. While Java is not the obvious choice for scientific computing, this framework represents a valuable proof of concept regarding the minimal requirements for connecting resources to a Grid framework and has proved valuable when introducing Grid computing to new users. The framework is fully functional and is deployed as part of the Minimum intrusion Grid. The "One-Click resource model" resulted in a paper published and presented at the HPCC07[aBV07] conference in Houston, Texas in September 2007.

Secondly the "PS3™ Grid Resource-model" was presented, which introduces a way to connect Playstation® 3's into a Grid infrastructure. The PS3™ represents a powerful computational resource as each machine has a theoretical peek performance of 153,6 GFLOPS in single precision. With more than 9 millions sold this sums up to a quite powerful resource even if only a fraction of them are gathered. The result of this research is a LIVECD which boots the PS3™ into a sandboxed Linux Grid execution environment. It cannot harm any existing operating system or data, as the execution environment doesn't have access to the HDD controller of the PS3™. The shortcoming of this model is that file access is limited to using 252 MB of the GPU's VRAM as block-device, and the write speed to the VRAM is limited to 10 MB/s which is quite slow. The "PS3™ Grid Resource-model" resulted in a paper published and presented at the GCA08[RB08] conference in Las Vegas, Nevada in July 2008.

To show that non-computer resources represent a real value to the Grid environment, an X86 application was ported to the PS3™ architecture. This was done to verify the

power of the PS3™ as well as addressing some of the issues to be taken into account when choosing to use a non-computer resource for computation. This work resulted in the paper ”Application Porting and Tuning on the Cell-BE Processor” presented at the PARA08[RV09] conference in Trondheim, Norway in May 2008.

The initial research showed that one shortcoming of non-Computer devices is the amount of memory available at those devices. As non-Computer devices are often non-extendible when it comes to hardware more physical memory can not simply be added. Furthermore the amount of general purpose disk present at the devices might severely be limited and un-utilizable for swap. To address these issues we presented the The Remote Memory Library which provides memory-limited Grid resources with memory through the Grid infrastructure. The Remote Memory Library presents a method for providing remote swap to global Grid infrastructures. We present a fully transparent user-level library, which can be submitted along with the Grid jobs eliminating the need to modify neither the OS of the executing Grid resource nor the Grid application to execute. Furthermore the user-level approach makes it possible to throttle the real memory usage of the running job, through the Grid middleware, and thereby increase the pool of resources capable of fulfilling the memory requirements of a given job. Last but not least the user-level approach ensures that only pages that are used by the Grid application are subject for eviction. The disadvantages of using the transparent user-level approach is the time overhead of passing signals, page mappings and page protections between kernel- and user-level, as well as the space overhead of keeping a local process page table within the framework as one can't access the page data-structures of the kernel from user-level.

While the widely used on-demand paging scheme performs well in a low latency high bandwidth network, it comes short when moving to a real high latency Grid environment. To address this we added asynchronous event based prediction to the framework in order to use the growing bandwidth of computer networks for latency hiding. Experiments showed that the asynchronous prediction based page prefetching scheme performed well in high latency, high bandwidth environments using iterative scientific applications. The non-iterative applications didn't show usable performance in the high latency, high bandwidth network, this however is regarded as a matter of gathering useful initial event sequences for these kinds of applications which is a subject for further research.

Bibliography

[AB08] Rasmus Andersen and Brian Vinter, *The scientific byte code virtual machine*, in Arabnia [Ara08], pp. 175–181.

[aBV06] Rasmus Andersen and Brian Vinter, *Harvesting idle windows cpu cycles for grid computing*, GCA (Hamid R. Arabnia, ed.), CSREA Press, 2006, pp. 121–126.

[aBV07] Martin Rehr and Brian Vinter, *The one-click grid-resource model*, HPCC (Ronald H. Perrott and Barbara M. Chapman and Jaspal Subhlok and Rodrigo Fernandes de Mello and Laurence Tianruo Yang, ed.), Lecture Notes in Computer Science, vol. 4782, Springer, 2007, pp. 296–308.

[ACC+03] Roberto Alfieri, Roberto Cecchini, Vincenzo Ciaschini, Luca dell'Agnello, Ákos Frohner, Alberto Gianoli, Károly Lörentey, and Fabio Spataro, *VOMS, an authorization system for virtual organizations*, European Across Grids Conference (F. Fernández Rivera, Marian Bubak, A. Gómez Tato, and Ramon Doallo, eds.), Lecture Notes in Computer Science, vol. 2970, Springer, 2003, pp. 33–40.

[ACK+02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer, *Seti@home: an experiment in public-resource computing*, Commun. ACM **45** (2002), no. 11, 56–61.

[And04] David P. Anderson, *Boinc: A system for public-resource computing and storage*, GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Washington, DC, USA), IEEE Computer Society, 2004, pp. 4–10.

[Ara08] Hamid R. Arabnia (ed.), *Proceedings of the 2008 international conference on grid computing & applications, gca 2008, las vegas, nevada, usa, july 14-17, 2008*, CSREA Press, 2008.

[ARV09] Rasmus Andersen, Martin Rehr, and Brian Vinter, *Cycle-scavenging in grid computing*, Grid Technology and Applications: Recent Develop-

ments (H.R: Arabina G.A. Gravvanis, J.P. Morrison and D.A. Power, eds.), Nova Science Publishers Inc., Hauppauge, NY, USA, 2009, p. Chapter 5.

[AV05] Rasmus Andersen and Brian Vinter, *Transparent remote file access in the minimum intrusion grid*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 311–318.

[Bar] Joshua Edward Barnes, *ftp://ftp.ifa.hawaii.edu/pub/barnes/treecode*.

[BEJ+09] Adam L. Beberg, Daniel L. Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S. Pande, *Folding@home: Lessons from eight years of volunteer distributed computing*, IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing (Washington, DC, USA), IEEE Computer Society, 2009, pp. 1–8.

[BH86] Josh Barnes and Piet Hut, *A hierarchical O(N log N) force-calculation algorithm*, Nature **324** (1986), no. 6096, 446–449.

[BLL91] Allan Bricker, Michael Litzkow, and Miron Livny, *Condor technical summary*, Tech. report, May 06 1991.

[Cat] Gentoo Catalyst, `http://www.gentoo.org/proj/en/releng/catalyst`.

[CFJ+08] Toni Cortes, Carsten Franke, Yvon Jégou, Thilo K̃ielmann, Domenico Laforenz, Brian Matthews, Christine M̃orin, Luis Pablo Prieto, and Alexander Reinefeld, *Xtreemos: a Vision for a Grid Operating System*, Technical report 4, XtreemOS European Integrated Projet, May 2008.

[CRDI05] Thomas Chen, Ram Raghavan, Jason Dale, and Eiji Iwata, *Cell broadband engine architecture and its first implementation*, IBM developerWorks (2005), `http://www.ibm.com/developerworks/power/library/pa-cellperf`.

[cUR] cURL, `http://curl.haxx.se/`.

[dANV+05] Marcos Dias de Assuno, Krishna Nadiminti, Srikumar Venugopal, Tianchi Ma, and Rajkumar Buyya, *An integration of global and enterprise grid computing: Gridbus broker and xgrid perspective.*, GCC (Hai Zhuge and Geoffrey Fox, eds.), Lecture Notes in Computer Science, vol. 3795, Springer, 2005, pp. 406–417.

[Dic] The Free Dictionary, *http://www.tfd.com*.

[Dij72] Edsger W. Dijkstra, *Chapter i: Notes on structured programming*, 72–82.

[EEE+03] Paula Eerola, Tord Ekelöf, Mattias Ellert, John Renner Hansen, Aleksandr Konstantinov, Balázs Kónya, Jakob Langgaard Nielsen, Farid Ould-Saada, Oxana Smirnova, and Anders Wäänänen, *The nordugrid architecture and tools*, CoRR **physics/0306002** (2003).

[ext] PS3 Linux extensions, `ftp://ftp.uk.linux.org/pub/linux/Sony-PS3`.

[FJ05] Matteo Frigo and Steven G. Johnson, *The design and implementation of FFTW3*, Proceedings of the IEEE **93** (2005), no. 2, 216–231, Special issue on 'Program Generation, Optimization, and Platform Adaptation".

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke, *The anatomy of the Grid: Enabling scalable virtual organization*, The International Journal of High Performance Computing Applications **15** (2001), no. 3, 200–222.

[For94] Message Passing Interface Forum, *Mpi: A message-passing interface standard*, 1994.

[Fos02] Ian Foster, *The grid: A new infrastructure for 21st century science*, Physics Today **55(2)** (2002), 42–47.

[Fos06] Ian T. Foster, *Globus toolkit version 4: Software for service-oriented systems*, J. Comput. Sci. Technol. **21** (2006), no. 4, 513–520.

[Gie78] Michel Gien, *A file transfer protocol (ftp).*, Computer Networks **2** (1978), 312–319.

[gLi] gLite, `www.glite.org`.

[HFPS99] R. Housley, W. Ford, W. Polk, and D Solo, *Internt x.509 public key infrastrucure certificate and crl profile*, RFC 2459 Edition, January 1999, http://www.ietf.org/rfc/rfc2459.txt.

[HHML05] Jose Herrera, Eduardo Huedo, Rubn S. Montero, and Ignacio Martn Llorente, *Porting of scientific applications to grid computing on gridway.*, Scientific Programming **13** (2005), no. 4, 317–331.

[IBM07] *Cell broadband engine programming handbook*, 2007, `http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9F820A5FFA3ECE8C8725716A0062585F/$file/CBE_Handbook_v1.1_24APR2007_pub.pdf`, pp. 891–703.

[iU] Filesystem in Userspace, http://fuse.sourceforge.net/.

[java] *Java applet security*, http://java.sun.com/security/javaone97-whitepaper.html.

[javb] *Java applets*, http://Java.sun.com/applets.

[Jow07] Mohammad Jowkar, *Exploring the Potential of the Cell Processor for High Performance Computing*, Master's thesis, University of Copenhagen, Denmark, August 2007.

[KB06] Henrik Hoey Karlsen and Brian Vinter, *Vgrids as an implementation of virtual organizations in grid computing*, WETICE, IEEE Computer Society, 2006, pp. 175–180.

[KF99] Carl Kesselman and Ian Foster, *The grid: Blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers, San Francisco, CA., 1999.

[KL96] Thomas M. Kroeger and Darrell D. E. Long, *Predicting file system actions from prior events*, ATEC '96: Proceedings of the 1996 annual conference on USENIX Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 1996, pp. 26–26.

[KL01] Tom M. Kroeger and Darrell D. E. Long, *Design and implementation of a predictive file prefetching algorithm*, Proceedings of the General Track: 2002 USENIX Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 2001, pp. 105–118.

[KV05] Henrik Hoey Karlsen and Brian Vinter, *Minimum intrusion grid - the simple model*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 305–310.

[Lin] Gentoo Linux, http://www.gentoo.org.

[LT01] L. Liming and S. Tuecke, *GridFTP protocol ANL category: ? march 2001 expires: August 2001 page 1 of 21 gridFTP: Protocol extensions to FTP for the grid*, February 27 2001.

[MB10] Martin Rehr and Brian Vinter, *The User-level Remote Swap Library*, HPCC, 2010, To Appear.

[MBC04] D S Myers, A L Bazinet, and M P Cummings, *Expanding the reach of grid computing: Combining globus- and boinc-based systems*, Journal of Parallel and Distributed Computing (2004).

[misa] *Dataynapse*, `http://www.datasynapse.com`.

[misb] *Entropia (SGE)*, `www.entropia.com`.

[misc] *Sun Grid Engine (SGE)*, `http://gridengine.sunsource.net/`.

[mis02] *Network working group E. rescorla request for comments: 2818 RTFM, inc. category: Informational may 2000 HTTP over TLS*, March 27 2002.

[MSVR07] Yassene Mohammed, Ulrich Sax, Fred Viezens, and Otto Rienhoff, *Shortcomings of current grid middlewares regarding privacy in healthgrids.*, Stud Health Technol Inform **126** (2007), 322–9.

[Nag84] J. Nagle, *Congestion Control in IP/TCP Internetworks*, RFC 896, January 1984.

[nqu] *IBM Full-System Simulator for the Cell Broadband Engine Processor*, `http://www.alphaworks.ibm.com/tech/cellsystemsim`.

[nqu48] Berliner Schachgesellschaft **3** (1848), 363.

[RB08] Martin Rehr and Brian Vinter, *The PS3 Grid-Resource Model*, in Arabnia [Ara08], pp. 90–95.

[RV09] Martin Rehr and Brian Vinter, *Application Porting and Tuning on the Cell-BE Processor*, Proceedings of the PARA 08, 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing, Lecture Notes in Computer Science, Springer, December 2009, accepted for publication.

[SBA07] Daniel Stødle, John Markus Bjørndalen, and Otto J. Anshus, *A system for hybrid vision- and sound-based interaction with distal and proximal targets on wall-sized, high-resolution tiled displays*, ICCV-HCI (Michael S. Lew, Nicu Sebe, Thomas S. Huang, and Erwin M. Bakker, eds.), Lecture Notes in Computer Science, vol. 4796, Springer, 2007, pp. 59–68.

[slBc] Open source lattice Boltzmann code, *http://www.openlb.org*.

[son] *Sony sales*, `http://www.sony.net/SonyInfo/IR/financial/fr/07q4_eleki.pdf`.

[SSB+95] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer, *BEOWULF: A parallel workstation for scientific computation*, Proceedings of the 24th International Conference on Parallel Processing (Oconomowoc, WI), 1995, pp. I:11–14.

[Sun90] V. S. Sunderam, *Pvm: A framework for parallel distributed computing*, Concurrency: Practice and Experience **2** (1990), 315–339.

[Sun00] Michael Sung, *SIMD parallel processing*, `http://www.ai.mit.edu/projects/aries/papers/writeups/darkman-writeup.pdf`.

[Tak] Takaken, `http://www.ic-net.or.jp/home/takaken/e/queen/index.html`.

[VAR+09] Brian Vinter, Rasmus Andersen, Martin Rehr, Jonas Bardino, and Henrik Karlsen, *Towards a robust and reliable grid middleware*, Grid Technology and Applications: Recent Developments (H.R: Arabina G.A. Gravvanis, J.P. Morrison and D.A. Power, eds.), Nova Science Publishers Inc., Hauppauge, NY, USA, 2009, p. Chapter 10.

[Ven07] Girish Venkatachalam, *The openssh protocol under the hood*, Linux J. **2007** (2007), no. 156, 6.

[Vin05] Brian Vinter, *The Architecture of the Minimum intrusion Grid (MiG)*, Communicating Process Architectures 2005, sep 2005.

[vmw] *VMware Player*, `http://www.vmware.com/products/player`.

[Wat08] Jon Watson, *Virtualbox: bits and bytes masquerading as machines*, Linux J. **2008** (2008), no. 166, 1.

Appendix A

Publication 1

Proceedings of High Performance Computing and Communications, Third International Conference, HPCC 2007, Houston, USA, September 26-28, 2007
ISBN: 978-3-540-75443-5, pp. 296-308
Martin Rehr, Brian Vinter: The One-Click Grid-Resource Model

The One-Click Grid-resource model

Martin Rehr and Brian Vinter

Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
{rehr, vinter}@diku.dk

Abstract. This paper introduces the One-Click Grid resource, which allows any computer with a Java enabled web browser to safely provide resources to Grid without any software installation. This represents a vast increase of the number of potential Grid resources that may be made available to help public interest research. While the model does make restrictions towards the application writer, the technology provides a real Grid model and supports arbitrary binaries, remote file access and semi-transparent checkpointing. Performance numbers show that the model is usable even with browsers that are connected to the Internet through relatively weak links, i.e. 512 kb/s upload speeds. The resulting system is in use today, and freely available to any research project.

1 Introduction

Grid Computing and Public Resource Computing, PRC, provide increasingly interesting means of obtaining computational resources. Grid Computing is mostly used for connecting university supercomputers, while PRC is predominantly used by research projects for harvesting PC based idle CPU-cycles for a small number of research projects. However, even though the two fields appear closely related, little effort has been made to combine them to a system that offers the flexibility of Grid computing with the resource richness of the PRC model.

1.1 Motivation

Harvesting 'free' cycles through PRC is of great interest since a modern PC is powerful and highly underutilized, and as such cycle harvesting provides a huge calculation potential if one combines millions of them in a computing Grid[1].

Most known Grid systems such as ARC[2] which is based on the Globus toolkit[3] and Condor[4] are unsuitable for PRC computing, as they work under the underlying assumption that the resources are available at anytime, which PRC resources by their very nature are not.

To extend the PRC concept to actual Grid computing, security and installation of software on the donated resource are vital issues. All, to the authors known, PRC projects requires the donor to install software on the resource that should contribute, which alone eliminates users from donating resources from

computers that they do have administrative rights on. The software installation also opens for possible exploits and requires the donor to perform updates on that software. This is not desirable and may reduce the amount of donated resources.

Ensuring the safety of a donated resource while it executes a Grid job in a PRC context is an all important topic since all free resources will vanish if the model proves harmful to the hosts. Contrary to standard PRC tasks, a Grid job may take any form and include the execution of any binary. Thus it is necessary to take precautions to ensure that the execution of Grid jobs cannot harm donated resources neither from intention nor by accident.

This paper addresses some of the problems that need to be solved in order to combine PRC computing and Grid Computing. Our goal is to design a Grid PRC secure sandbox model, where Grid jobs are executed in a secure environment and no Grid or application specific software is needed on the donated resource. Furthermore we ensure that resources may be in a typical PRC context, i.e. located behind a Network Address Translation router and a firewall and thus that the model may be used without modifications to firewalls or the routers.

1.2 Related Work

BOINC[5] is a middleware system providing a framework, which has proved the concept of PRC, and is widely used by scientific research projects such as SETI@HOME[6] and FOLDING@HOME[7]. MiG-SSS[8] is a Screen Saver Science model built to combine PRC with Minimum intrusion Grid, MiG[9][10]. Our work differs from BOINC by aiming at a full Grid model, and differs from both BOINC and MiG-SSS by aiming at no Grid specific software installation on the client.

2 Web browsers and Java

To reach our stated goal of no Grid specific software installation and no modification of the donated machines firewall settings, we are forced to use software which is an integrated part of a common Internet connected resource.

We found that amongst the most common software packages for any PC type platform there is a Java enabled web browser. The web browser provide a common way of securely communicating with the Internet, which is allowed by almost all firewall configurations of the resources we target.[1] The web browser itself provides us with a communication protocol, but it does not by itself, provide a safe execution environment, however all of the most common graphics enabled web browsers have support for Java applets[11], that are capable of executing Java byte-code located on a remote server.

[1] Resources located behind firewalls that do not support outgoing HTTPS is considered out of range for this PRC, however it is not unseen that outbound HTTPS is blocked.

The Java applet security model[12], ASM, prevents the Java byte-code executed in the applet from harming the host machine and thereby provides the desired sandbox effect for us to trust the execution of unknown binaries on donated resources.

The choice of web browsers and Java applets as the execution framework, results in some restrictions on the type of jobs that may be executed in this environment:

- Applications must be written in Java
- Applications must apply to ASM
- The total memory usage is limited to 64 MB including the Grid framework
- Special methods must be used to catch output
- Special methods must be used for file access

By accepting the limitations described above, a web browser may become a Grid resource simply by entering a specific URL. This triggers the load and execution of an applet which acts as our Grid gateway and enables retrieving and executing a Java byte-code based Grid job. The details of this process is described next.

3 The Applet Grid Resource

Several changes to the Grid middleware are needed to allow Java applets to act as Grid resources. First of all the Grid middleware must support resources which can only be accessed through a pull based model, which means that all communication is initiated by the resource, i.e. the applet. This is required because the ASM rules prevents the applet from initiating listening sockets, and to meet our requirement of functioning behind a firewall with no Grid specific port modifications. Secondly, the Grid middleware needs a scheduling model where resources are able to request specific type of jobs, e.g. a resource can specify that only jobs which are tagged to comply to the ASM can be executed.

In this work the Minimum intrusion Grid[9], MiG, is used as the Grid middleware. The MiG system is presented next, before presenting how the Applet Grid resource and MiG work together.

3.1 Minimum intrusion Grid

MiG is a stand alone Grid platform, which does not inherit code from any earlier Grid middlewares. The philosophy behind the MiG system is to provide a Grid infrastructure that imposes as few requirements on both users and resources as possible. The overall goal is to ensure that a user is only required to have a X.509 certificate which is signed by a source that is trusted by MiG, and a web browser that supports HTTP, HTTPS and X.509 certificates. A fully functional resource only needs to create a local MiG user on the system and to support inbound SSH. A sandboxed resource, which can be used for PRC, only needs outbound HTTPS[8].

Because MiG keeps the Grid system disjoint from both users and resources, as shown in Figure 1, the Grid system appears as a centralized black box[9] to both users and resources. This allows all middleware upgrades and trouble shooting to be executed locally within the Grid without any intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system that, though it appears as a centralized system in reality is distributed. The basic functionality in MiG starts by a user submitting a job

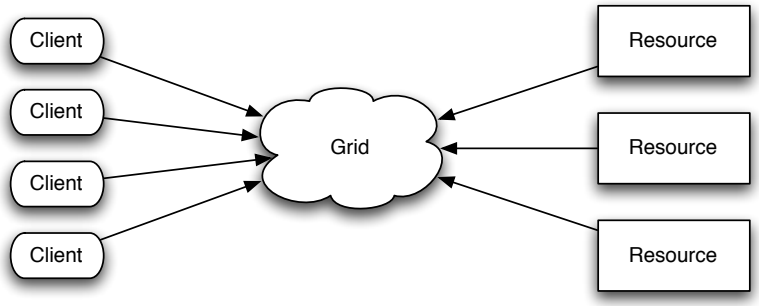


Fig. 1: The abstract MiG model

to MiG and a resource sending a request for a job to execute. The resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can then inform the user of the job completion. Since the user and the resource are never in direct constant, MiG provides full anonymity for both users and resources, any complaints will have to be made to the MiG system that will then look at the logs that show the relationship between user and resource.

3.1.1 Scheduling The centralized black box design of MiG makes it capable of strong scheduling, which implies full control of the jobs being executed and the resource executing them. Each job has an upper execution time limit, and when the execution time exceeds this time limit the job is rescheduled to another resource. This makes the MiG system very well suited to host PRC resources, as they by nature are very dynamic and frequently join and leave the Grid without notifying the Grid middleware.

3.2 The MiG Applet Resource

As explained above, all that is required for a PRC resource to join MiG is a sandbox and support for outgoing HTTPS. However, the previous solution[8] requires installation of non standard software to activate and execute the sandbox.

The Java applet technology makes it is possible to turn a web browser into a MiG sandbox without installing any additional software. This is done automatically when the user accesses "MiG One-Click"[2], which loads an applet into the web browser. This applet functions as a Grid resource script and is responsible for requesting pending jobs, retrieving and executing granted jobs, and delivering the results of the executed jobs to the MiG server.

To make the applet work as a resource script, several issues must be addressed. First of all ASM disallows local disk access. Because of this both executables and input/output files must be accessed directly at the Grid storage. Secondly only executables that are located at the same server as the initial applet are permitted to be loaded dynamically. Thirdly text output of the applet is written to the web browser's Java console and not accessible by the Grid middleware.

When the applet is granted a job by the MiG server, it retrieves a specification of the job which specifies executables and input/output files. The applet then loads the executable from the Grid, this is made possible by the MiG server which sets up an URL from the same site as the resource applet was originally loaded which points to the location of the executables. This allows unknown executables to be loaded and comply with the ASM restrictions on loading executables. Figure 2 shows the structure of an One-Click job. Executables that are

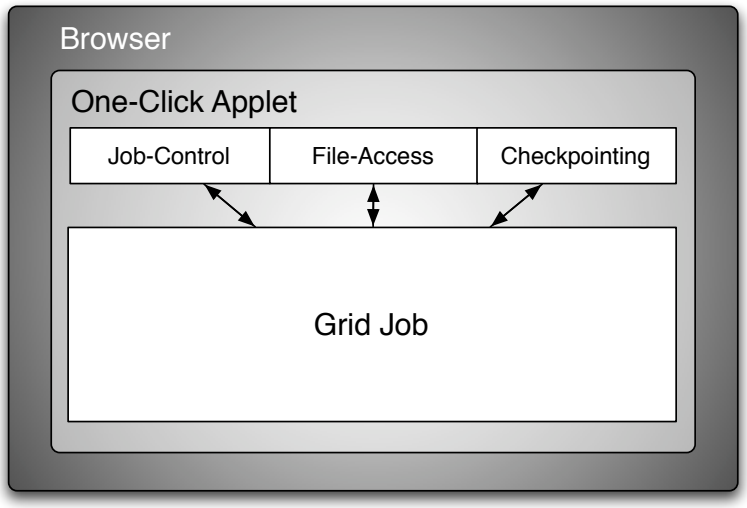


Fig. 2: The structure of an One-Click job

[2] The URL accessed to activate the web browser as a sandboxed MiG Java resource is called "MiG One-Click", as it requires one click to activate it.

targeted for the MiG One-Click model must comply with a special MiG One-Click framework, which defines special methods for writing stdout and stderr of the application to the MiG system[3]. Normally the stdout and stderr of the executing job is piped to a file in the MiG system, but a Java applet, by default, writes the stdout and stderr to the web browsers Java console. We have not been able to intercept this native output path. Input and output files that are specified in the job description must be accessed directly at the Grid storage unit since the ASM rules prohibits local file access. To address this issue the MiG One-Click framework provides file access methods that transparently provide remote access to the needed files. Note that the MiG system requires input files and executables to be uploaded to the MiG server before job submission which ensures that the files are available at the Grid storage unit.

In addition to the browser applet a Java console version of the MiG resource has been developed, to enable the possibility of retrieving and executing MiG One-Click jobs as a background process. This requires only a Java virtual machine. To obtain the desired security model, a customized Java security policy is used, which provides the same restrictions as the ASM.

3.3 Remote File Access

The One-Click executing framework that was introduced above also provides transparent remote file access to the jobs that are executed. The MiG storage server supports partial reads and writes, through HTTPS, of any file that is associated with a job. When the resource applet accesses files that are associated with a job, a local buffer is used to store the parts of the file that are being accessed. If a file position which points outside the local buffer is accessed, the MiG server is contacted through HTTPS, and the buffer is written to the MiG server if the file is opened in write mode. The next block of data is then fetched from the server and stored into the buffer and finally the operation returns to the user application. The size of the buffer is dynamically adjusted to utilize the previously observed bandwidth optimally.

3.3.1 Block size estimation To achieve the optimal bandwidth for remote file access it is necessary to find the optimal block size for transfers to and from the server. In this case the optimal block size is a trade off between latency and bandwidth. We want to transfer as large a block as possible without excessive latency increment since the chance of transferring data that will not be used increases with the block size.

We define the optimal block size bs_{opt} as the largest block where a doubling of the block size does not double the time to transfer it. This can be expressed the following way:

$$t(x) * 2 > t(x * 2) \qquad \forall x < bs_{opt} \tag{3.1}$$

$$t(x) * 2 < t(x * 2) \qquad \forall x > bs_{opt} \tag{3.2}$$

[3] The result of a MiG job is the stdout/stderr and the return code of the application that is executed.

$$t(x) \ = time\ to\ transfer\ block\ of\ size\ x$$

We do not want block sizes below bs_{opt} as the time t used to transfer a block of size x is less than doubled when the block size is doubled. On the other hand we don't want 'too large' block sizes as we do not know if the retrieved data is going to be used or discarded due to a seek operation beyond the end of the local buffer.

As the One-Click resources can be placed at any sort of connection, and the bandwidth of the connection thus may differ greatly from one resource to another, it is not possible to use a fixed block size and reach a good ratio between bandwidth and latency at an arbitrary type of connection.

The simplest approach would be to use a fixed bs_{opt} based on empirical tests on the most common connections.

A less trivial, but still simple, approach would be to measure the time it takes to connect to the server and then choose a block size which ensures the transfer time of that block to be a factor of x larger than the time to connect, to make sure that the connection overhead does not exceed the time of the actual data transfer.

The chosen approach is to estimate bs_{opt} from the time spent transferring block $x-1$ with the time of transferring block x, starting with an initial small[4] block size bs_0 and then doubling the block size until a predefined cutoff ratio CR is reached. After each data transfer the bandwidth bw_x is calculated and compared to the bandwidth of the previous transfer bw_{x-1}. If the ratio is larger than the predefined CR:

$$\frac{bw_x}{bw_{x-1}} > CR \tag{3.3}$$

then the block size is doubled:

$$bs_{x+1} = bs_x * 2 \tag{3.4}$$

As the block size is doubled in each step the theoretical CR to achieve bs_{opt} should be 2, since there is no incentive to increase block size once the latency grows linearly with the size of the data that is transferred. However in reality, one need to get a CR below 2 to achieve bs_{opt}. This is due to the fact that all used block sizes are powers of 2, and one cannot rely on the optimal block size to match a power of 2.

Therefore to make sure to get a block size above bs_{opt} you need a lower CR. Empirical tests showed that a CR about 1.65 yields good results, see section 5.2

Additional extensions include adapting to the frequency of random seeks in the estimation of the CR. A large amount of random seeks to data placed outside the range of the current buffer will cause new blocks to be retrieved in each seek. Therefore the block size should be lowered in those cases to minimize the latency of each seek.

[4] An initial small block size gives a good result as many file accesses applies to small text files such as configuration files.

4 Checkpointing

PRC resources will join and leave the Grid dynamically, which means that jobs with large running time have a high probability of being terminated before they finish their execution. To avoid wasting already spent CPU-cycles a checkpointing mechanism is build into the applet framework. Two types of checkpointing have been considered for inclusion, transparent checkpointing and semi-transparent checkpointing.

4.1 Transparent Checkpointing

All to the authors known transparent checkpoint mechanisms provided to work with Java, require the JVM to be replacement or access to the /proc file system on Linux/Unix operating system variants, as the default JVM does not support storing program counter and stack frame. Since our goal is to use a web browser with the Java applet as a Grid resource neither of those solutions are satisfactory, since both the replacement of the JVM and access to the /proc file system violates the Java applet security model. Furthermore most PRC resource will be running the Windows operating system which do not support the /proc file system.

4.2 Semi-transparent Checkpointing

Since transparent checkpointing is not applicable to the One-Click model, we went on to investigate what we call semi-transparent checkpointing. Semi-transparent checkpointing covers that the One-Click framework provides a checkpoint method for doing the actual checkpoint, but the application programmer is still responsible for calling the checkpoint method when the application is in a checkpoint safe state.

The checkpoint method stores the running Java object on the MiG server through HTTPS. Since it can only store the object state, and not stack information and program counters, the programmer is responsible for calling the checkpoint method at a point in the application, where the current state of the execution may be restored from the object state only. To restart a previously checkpointed job, the resource applet framework first discovers that a checkpoint exists and then loads the stored object.

To ensure file consistency as part of the checkpoint, the framework also supports checkpointing of modified files, which is done automatically without involving the application writer. Open files are checkpointed if the job object includes a reference to the file.

5 Experiments

To test the One-Click model we established a controlled test scenario. Eight identical Pentium 4, 2.4 GHz machines with 512 MB ram were used for tests.

5.1 One-Click as concept

The test application used, is an exhaustive algorithm for folding proteins written in Java. This was changed to comply with the applet framework.

A protein sequence of length 26 was folded on one machine, which resulted in a total execution time of 2 hours, 45 minutes and 33 seconds. The search space of the protein was then divided into 50 different subspaces using standard divide and conqueror techniques. The 50 different search spaces were submitted as jobs to the Grid, which provides an average of 6 jobs per execution machine and 2 extra jobs to prevent balanced execution. The search spaces on their own also provide unbalanced execution as the valid protein configurations vary from one search space to another and thus results in unbalanced execution times. The experiment was made without checkpointing the application. The execution of the 50 jobs completed in 29 minutes and 8 seconds, a speedup of 5.7 for 8 machines. While this result would be considered bad in a cluster context it is quite useful in a Grid environment.

To test the total overhead of the model, a set of 1000 empty jobs was submitted to the Grid with only one One-Click execution resource connected. The 1000 jobs completed in 19935 seconds, which translates to an overhead of approximately 20 seconds per job.

5.2 File access

To achieve the best bandwidth cutoff ratio CR several experiments has been made. In the experiments a 16 MB file was read 100 times by the One-Click resource on a 20 Mb/s broadband Internet connection. All experiments start with an initial block size of 2048 (2^{11}) bytes. The first experiment was run with a CR of 0, which means that the block size is doubled in every transfer. The result is shown in figure 3.

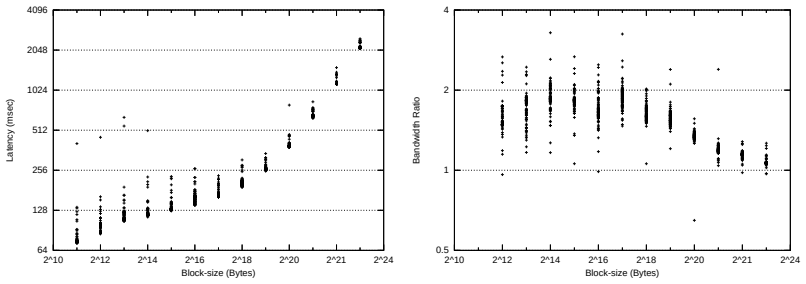


Fig. 3: The upper figure shows the latency as a function of the block size, the lower figure shows the bandwidth ratio $\frac{bw_x}{bw_{x-1}}$ as a function of the block size. Between block size 2^{18} and 2^{20} the latency starts to raise and the bandwidth ratio starts to fall. This is where the cutoff is chosen to avoid excessive raise in latency

The figure shows how that the latency starts to raise dramatically between block size 2^{18} and 2^{20} and the bandwidth to latency ratio starts to fall at those block sizes. The bandwidth to latency ratio between block size 2^{18} and 2^{20} lies in the interval from 1.25 to 1.75. Based on these observations we performed the same test with a CR of 1.5. The result is shown in figure 4.

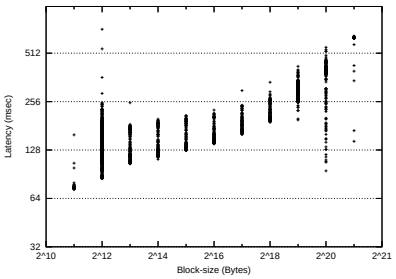


Fig. 4: The latency as a function of the block size with CR 1.5

This shows that a CR of 1.5 is too low as block sizes of 2^{21} occur and we want the block sizes to be between 2^{18} and 2^{20} to limit the maximum latency. Therefore the CR must be between 1.5 and 1.75. The test was then run with CR 1.55, 1.60, 1.65, 1.70 and 1.75. The result is shown in figure 5.

We observe that a CR of 1.75 is too high, as only a few block sizes of 2^{19} occur and no block sizes of 2^{20} occurs. A CR of 1.55 results in a few block sizes of 2^{21} which is above the block sizes we want. 1.60 represents the block sizes we want and block size 2^{20} is well represented. A CR of 1.65 represents block size 2^{19} well and a few block sizes of 2^{20} is reached as well, and a CR 1.70 represents block size 2^{20} but no block sizes of 2^{21} are represented. We choose a CR of 1.65 as block size 2^{20} is considered the braking point where the latency starts to grow excessively, therefore we do not want it to be to well represented, but we want it to be represented, which is exactly the case at a CR of 1.65.

To verify the previous finding that the CR value should be 1.65 a test application, which traverses a 16 MB file of random 32 bit integers was developed. First the application was tested against the framework, where fixed block sizes were used, and then the application was tested against the framework, where the dynamic block sizes with a CR of 1.65 were used. The results are shown i figure 6.

The experiment shows, as expected, that the execution time decreases as the block sizes increase in the experiments with static block sizes. The execution time in the experiments with the dynamic block sizes all reside around 256 seconds[5] which are satisfactory, as this shows that compared to largest static buffer size of

[5] With the exception of 3 runs, which are classified as outliers

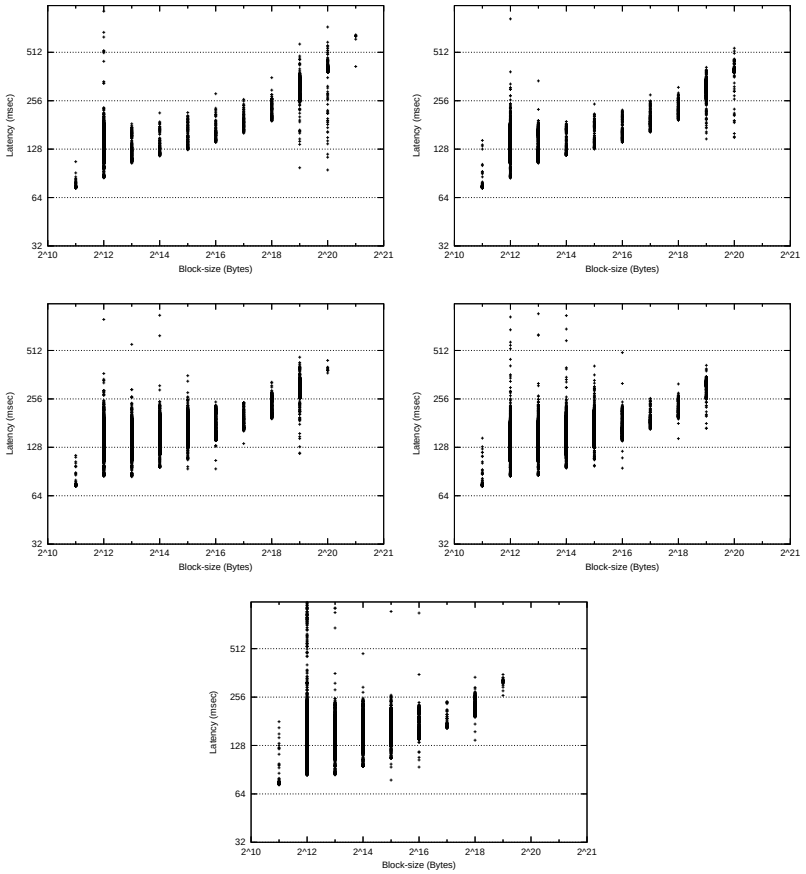


Fig. 5: The latency as a function of the block size with CR 1.55, 1.60, 1.65, 1.70, 1.75

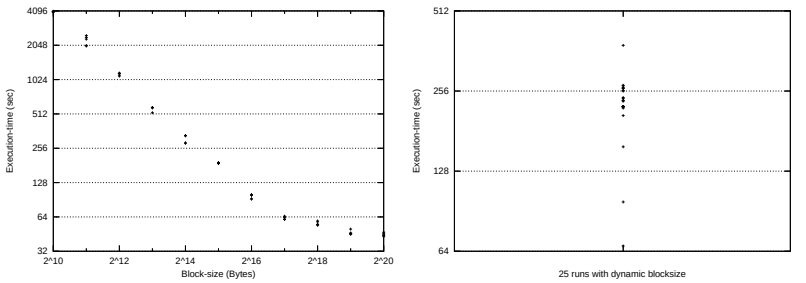


Fig. 6: The execution time as a function of the block size and the execution time with dynamic block sizes and a CR of 1.65

interest[6], the execution time loss using a dynamic buffer size is at most a factor of four. The reader should note that this type of application is the worst case for dynamic buffer sizing as all the data are read sequentially. If the integers were read in random order, the dynamic buffer size execution would perform much better.

5.3 Checkpointing

The next obvious performance issue is to test the overhead of performing a checkpoint operation within a process. This was tested by submitting jobs that allocate heap memory in the range from 0 kB to 8192 kB. Each job first allocates X kB, where X is in the order power of 2, and does 10 checkpoints, which saves the entire heap space. The performance was first tested on a 20 Mb/s broadband Internet connection. The test was then repeated using a more modest 2048/512 kb/s broadband Internet connection. The result of these tests is shown in figure 7. In the first test, using the 20 Mb/s connection, the checkpoint time is constant

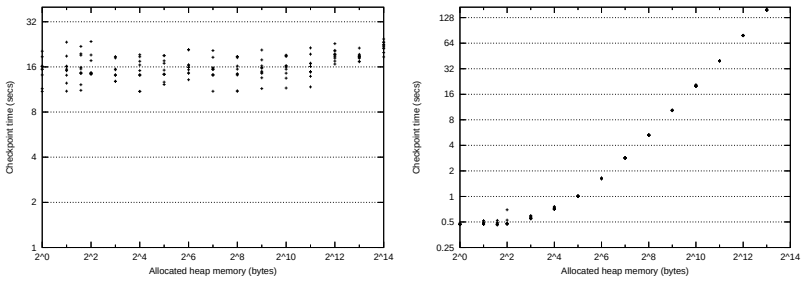


Fig. 7: The time spend checkpointing on a 20 Mb/s and a 2048/412 kb/s Broadband Internet

as the memory size grows. We can conclude from this, that the overhead of serializing the Java object is dominating compared to the actual network transfer time. The opposite is the case when we examine the results of the 2048/512 kb/s connection. Here we see that the time spent grows linearly with the size of the allocated memory, from which we may conclude that on a 512 kb/s connection the bandwidth is, not surprisingly, the limiting factor.

[6] The largest static buffer of interest is 2^{17} as this is where the time gained by doubling the buffer, levels out.

6 Conclusion

In this work we have demonstrated a way to combine Grid computing with PRC, the One-Click framework, without the need to install any PRC client software on the donating resource.

The use of Java applets provides a secure sandboxed executing environment that prevents the executing Grid jobs from harming the donated machine. The disadvantage of this approach is that all jobs must be written in Java and in addition comply with the presented framework, including the Java Applet Security Model. However the modifications that are needed to port an existing Java application are limited to using special methods for stdout and stderr, applying to the Java applet security model, and using the One-Click framework for remote file access. The One-Click framework also includes the means to provide semi-transparent checkpointing of the applications at runtime.

The Minimum intrusion Grid supports the required pull-job model for retrieving and executing Grid jobs on a resource located behind a firewall without the need to open any ingoing ports. By using the One-Click approach any computer with a web browser that can execute Java applets can become a Grid resource for PRC simply by entering the MiG One-Click URL. Once the user of the donated computer wishes to stop the execution, the browser is simply closed down or pointed to another URL, and the execution stops. The MiG system detects this event, by a timeout, and resubmits the job to another resource, where the job is resumed from the latest checkpoint that was made.

Experiments have been performed to find the optimal block size for the remote file transfer that the framework includes. The experiments show that doubling the block size in each transfer gives the optimal tradeoff between bandwidth and latency as long as the CR is below 1.65.

The experiments also show that the dynamic block sizes approach increases the execution time by of factor of four compared to the execution time reached with the largest static block size in a worst case scenario.

The building checkpointing mechanism has an overhead of 15 seconds per checkpoint on a 2.4GHz P4 and the One-Click framework overall is causing approximately 20 seconds of overhead to each execution, compared to local execution. Despite of this a considerable speedup is reached in the presented protein experiment.

References

1. I. Foster, The Grid: A New Infrastructure for 21st Century Science, Physics Today, 55(2):42-47,(2002)
2. NorduGrid, (http://www.nordugrid.org)
3. Globus Toolkit, (http://www.globus.org/toolkit)
4. Condor Project, (http://www.cs.wisc.edu/condor)
5. Berkeley Open Infrastructure for Network Computing, (http://boinc.berkeley.edu)
6. SETI@home, (http://setiathome.berkeley.edu)

7. Folding@Home, (http://folding.stanford.edu)
8. Rasmus Andersen, Brian Vinter; Harvesting Idle Windows CPU Cycles for Grid Computing, Proceedings of (2006) International Conference on Grid Computing & Applications (GCA'06/ISBN #:1-60132-014-0/CSREA), Editor: Hamid R. Arabnia, pp.: 121-126, Las Vegas, USA, (2006)
9. B. Vinter 'The architecture of the Minimum intrusion Grid: MiG" In Communicating Process Architectures: pp. 189-201 Broenink J, Roebbers H, Sunter J, Welch P, Wood D (eds.) IOS Press, (2005)
10. Henrik Hoey Karlsen, Brian Vinter, "Minimum intrusion Grid - The Simple Model," wetice, pp. 305-310, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05), (2005)
11. Java Applets, (http://Java.sun.com/applets)
12. Java Applet Security, (http://Java.sun.com/sfaq)

Appendix B

Publication 2

Proceedings of the PARA 08, 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing, Trondheim, Norway, May 13-16, 2008
Accepted for publication December 2009
Martin Rehr, Brian Vinter: Application Porting and Tuning on the Cell-BE Processor

Application porting and tuning for the CELL-BE processor

Martin Rehr and Brian Vinter

Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
{rehr, vinter}@diku.dk

Abstract. The Cell is a heterogeneous multi-core processor, consisting of 9 cores with a peak performance in excess of 100 giga-operations per second. To make the Cell processor provide more that a fraction of that performance, a very high level of parallelism is needed as well as a number of basic, but very important code optimizations. This paper covers the process of porting an existing recursive application to the Cell processor, and the steps needed to achieve high performance. Optimization methods such as the four levels of parallelism supported by the Cell, task-, memory-, data-, and instruction-level parallelization are covered as well as branch elimination. The final performance numbers show that the Cell processor outperforms traditional processor architectures quite impressingly if an application is ported properly.

1 Introduction

Intel cancelled the Pentium 4, 4 GHz in 2004 due to heating issues, we still haven't seen it and never will. Moore's law[6] is still going strong, which means the number of transistors per square-inch doubles approximately every two years. Using the increasing amount of transistors for increasing cache-sizes and deeper pipelines are no longer improving performance significantly. This has led to the design of multi-core processors, some of which are just extensions of the old processor design making it easy to apply existing applications to them, but making it hard to gain any significant performance increase. However Sony, IBM and Toshiba have developed the Cell Broadband Engine (Cell BE[4]) which is a powerful heterogeneous multi-core chip capable of doing 204 GFLOPS single precision and 14 GFLOPS double precision, using a whole new architecture. To get near this kind of performance one has to carefully tune the applications towards this architecture.

1.1 The Cell BE

The Cell processor is a heterogeneous multi core processor consisting of 9 cores, The Primary core is an IBM 64 bit power processor (PPC64) with 2 hardware threads. This core is the link between the operating system and the 8 powerful

working cores, called the SPE's for Synergistic Processing Element. The power processor is called the PPE for Power Processing Element, all cores are connected by an Element Interconnect Bus (EIB). Figure 1 shows an overview of the Cell architecture. The cores are connected by an Element Interconnect Bus (EIB)

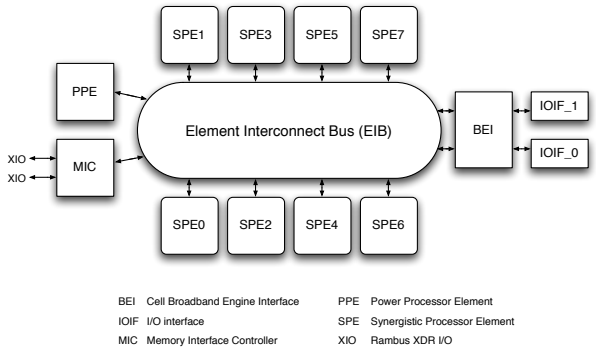


Fig. 1. An overview of the Cell architecture

capable of transferring up to 204 GB/s at 3.2 GHz. Each SPE is dual pipelined, has a 128x128 bit register file and 256 kB of on-chip memory called the local store. Data is transferred asynchronously between main memory and the local store through DMA calls handled by a dedicated Memory Flow Controller (MFC). An overview of the SPE is shown in figure 2. By using the PPE as

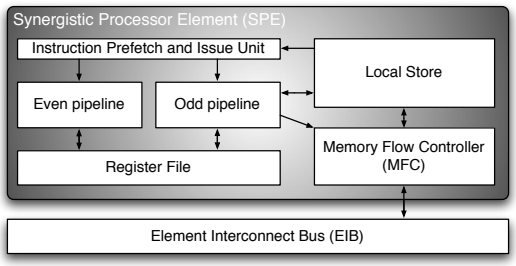


Fig. 2. An overview of the SPE cores

primary core, the Cell processor can be used out of the box, due to the fact that existing operating systems support the PPC64 architecture. Thereby it's possible to boot a PPC64 operating system on the Cell processor, and execute PPC64 applications, however these will only use the PPE core. To use the SPE

cores it's necessary to develop code specifically for the SPE's, which includes setting up a memory communications scheme using DMA through the MFC.

2 Porting towards the Cell

To make the Cell BE perform at a high level one has to consider several levels of parallelization, including task-, memory-, data-and instruction-level parallelization. As an experiment an nqueen[2][5] application written by Takaken[9] has been ported from the X86 architecture to the Cell architecture. The nqueen problem is solved by using a divide and conquer algorithm for finding how many ways to place N queens safely at an NxN chess board according to the common chess rules.

2.1 Task-and memory-parallelization

As the Cell BE architecture can be viewed as an 8 node (the SPE's) cluster[7] on a chip with a front end (The PPE) splitting an application into smaller tasks is done by the same principles as when parallelizing towards a cluster computer. However due to the limited amount of local store (256 kB) available at the SPE's for both code and data, one has to consider the size of each task. This means that an application which would be best parallelized by a bag of task model in a cluster setup, might be best parallelized by a pipelined setup using the Cell processor.

2.1.1 Task-parallelization The first step of porting an application to the Cell is to parallelize it task wise, using the PPE as a task manager and the SPE's as computation units. This is done by analyzing the application, picking the right method for parallelization including analyzing if data and code for each task fits into the 256 kB local store of the SPE's, and if not, which method one wishes to use to make it fit. The two possibilities here are either to split the data-set for each task into smaller data-sets or to use a pipelined setup, where the code is split among 2,4 or 8 SPE's each performing some piece of the computation.

2.1.2 Memory-parallelization Each SPE has its own MFC controller running in its own thread, meaning that main memory can be accessed through DMA asynchronously regarding computation. This gives the possibility of effective memory latency hiding, as the MFC writes data directly from the EIB to the local store without involving the computational unit. Thereby the data for iteration i+1 can be retrieved while computing iteration i, this is known as double buffering.

Furthermore each MFC is capable of issuing 16 simultaneous DMA transfers giving a high level of possible multi-buffering[1]. When porting an application,

[1] Where data for iteration i+1,i+2, ..., i+n is retrieved in iteration i

a communication scheme between the PPE and the SPE using DMA transfers through the MFC has to be chosen. Most applications will use at least double buffering to hide memory latency, and some applications need to use multi-buffering to keep the computational unit busy, this all depends of the computational intensiveness of the task.

2.1.3 The nqueens solution In the nqueens example, code and data fit into the local store of each SPE, and therefore a bag-of-task model is used, where each SPE requests a task from the PPE, gets the input data, computes the result, delivers the result to the PPE and requests a new task. As the input and output data for the nqueens application is quite small and the compute intensive part of the application is quite large, double buffering is sufficient for keeping the SPE's busy, hiding the memory latency efficiently. This first step reduced the execution time of placing 18 queens on an 18x18 chess board from 278.878 seconds on a Pentium 4 running at 3,2 GHz to 69.456 seconds when executed on a Playstation 3[2] giving a speedup of 4. The application has been compiled with both the GCC compiler and IBM's XLC compiler, the result is shown in figure 3. It's seen that in this case the GCC compiler produces code which is

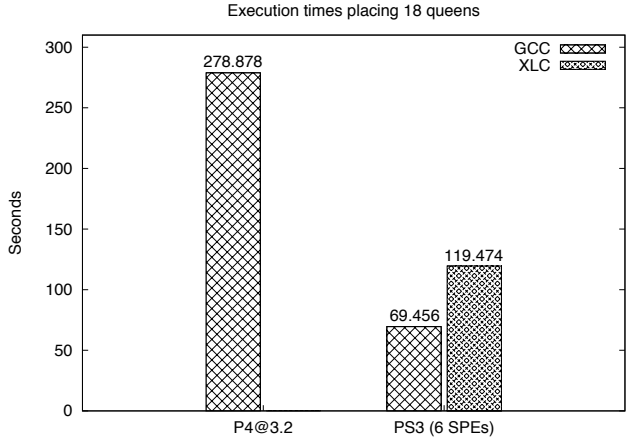


Fig. 3. Execution times for the task parallelized nqueens application

significantly faster than the code produced by the XLC compiler.

[2] Note the PS3 only has 6 SPE's

2.2 Register-line optimizations

The Cell BE SPE's are SIMD[8] vector cores each operating on a 128 bit register-line, which can be divided into 2x64 bit longs, 4x32 bit int's, 8x16 shorts or 16x16 bytes. This results in scalar operations to be mapped down to an atomic sequence of register-line operations, as the scalar has to be put in its preferred slot of the register-line see figure 4. The compute intensive part of the code should be fitted

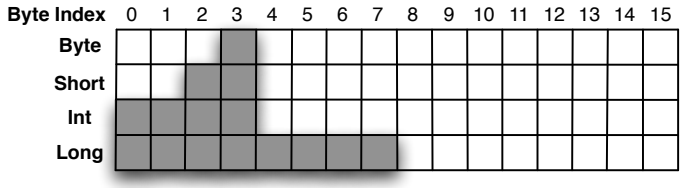


Fig. 4. The preferred slot in a register line for the different data types

to use 128 register-line operations instead of scalars, as this will eliminate the rotate and shuffle instructions needed to get the scalar into the correct preferred slot.

2.2.1 Recursive vs. iterative methods If the application to be ported is of a recursive nature, one might consider to transform the compute intensive part into an iterative method, as the limited local store of 256 kB is exhausted quite quickly if deep recursions are reached. Furthermore, the use of recursive methods might slow down execution if the data used in the compute intensive part is fitted into the 2 kB register file, as local parameters are pushed to the stack upon a function call.

2.2.2 Branch prediction and elimination The Cell processor has no hardware branch predictor, but the instruction set contains a branch hinting instruction. However if the programmer has no clue on what branch is to be taken, and the piece of code within each branch is fairly small, branch elimination can be done by calculating both results and selecting the right result based on the branch condition[3]. This has two advantages, it eliminates branch misses and it operate directly on register-lines whereas normal branch operations operates on scalars. Using these two methods the overall penalty of branching can be reduced quite impressively.

2.2.3 The nqueens solution The compute intensive part of the presented nqueen application was transformed from a recursive algorithm, to an iterative algorithm using the described register-line optimizations and branch elimination techniques. This resulted in an execution time reduction from 69.456 seconds

to 42.486 seconds, giving a speedup of 1.63 compared to the task parallelized code, and a speedup of 6.55 compared with the Pentium 4 execution running at 3.2 GHz. Furthermore the performance of the iterative scalar version of the algorithm was measured, all versions was compiled with both the GCC and XLC, the results are shown in figure 5. It's seen that the recursive scalar code

Fig. 5. Execution time comparisons between the different optimizations

overall performs a little better than the scaler iterative code, but that the XLC compiler produces faster binaries when used on the iterative code. Finally it's seen that the XLC compiler produces 15% faster binaries from the register-line code compared to GCC. These results show that the GCC produces faster binaries in the common case[3] whereas XLC produces faster binaries when the application is tuned towards the Cell SPE's architecture.

2.3 Data parallelization

As mentioned, the Cell BE SPE cores are SIMD vector cores, which offer data parallelization as an optimization parameter. Each SPE core is capable of performing 4 integer operations, 8 short operations or 16 byte operations per instruction. If the data has an integer SIMD nature, for example integer matrix multiplication, one can effectively vectorize by loop-unrolling, doing four integer operations in each loop shortening the total loop length by a factor of four. Otherwise if the application has a divide and conquer nature, one can vectorize by doing four branches simultaneously. The performance gain of branch vectorization depends heavily on the balance of the four branches, as the amount of leaves traversed is the worst case of the four branches.

[3] Applications written for traditional single core architectures

2.3.1 The nqueens solution The compute intensive part of the presented nqueens application is based on a divide and conquer algorithm. This algorithm was vectorized by investigating four branches simultaneously. The execution time of the vectorized code placing 18 queens was reduced from 42.486 seconds to 41.248 seconds, which is considered an insignificant speedup, the result is shown in figure 6. An analysis of the vectorized code showed that the average depth

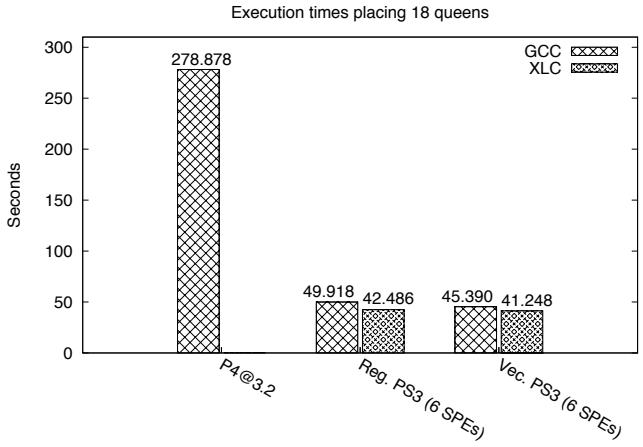


Fig. 6. Execution times for the vectorized nqueens application

reached in the search tree increased by 0.3 due to unbalanced branch vectorization. The presented nqueens application has a search space of N!, this means that solving the problem for more than 14 queens eliminates the advantage of branch vectorization. Performance measurements show that when placing 20 queens, the branch vectorized code gets slower than the register-line optimized code, figure 7. This is due to additional operations needed to test when all four branches within the vector have met their termination criteria. To make the branch vectorized code perform better, one could try to balance the branches placed within each vector. This has not been done with the presented nqueens application, as it's believed by the authors that the amount of processing power needed to balance the branches within the vectors is equal, or exceeds, the processing power needed to solve the problem itself.

2.4 Instruction parallelization

As the Cell BE is dual pipelined, one pipeline for computation and one for management, it's possible to perform load/store instructions from/to the local store while a computation instructions is being performed, this is specially usable within loops over data-sets, where it's clear which data are needed next. This is one of the optimization tasks which the compiler is capable of doing.

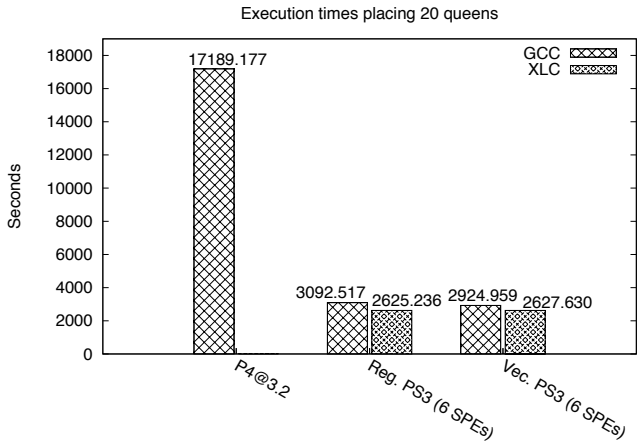


Fig. 7. Execution times for the register-line and branch-vectorized nqueens

2.4.1 The nqueens solution To measure how effectively the compiler interleaves the computation and data management instructions, the iterative register-line version[4] of the nqueens application compiled with the IBM XLC compiler has been profiled by using the Cell-BE system simulator[1]. The result of this was that approximately 13.9% of all instructions are instruction parallelized[5]. However the profiling revealed that 8.5% of all clock-cycles are used waiting on load/stores between the register-file and the local store. This indicates there is still room for improvement, which can be done either by eliminating local store access by using registers if possible, or doing instruction level parallelizing by hand at assembler level. This has not been looked further into.

2.5 Summary

When porting an application from a traditional single core application, i.e. the X86, to the Cell BE architecture, at least task-and memory-parallelization should be chosen as the PPE of the Cell in itself performs poorly, see figure 8. The core computational part of the task-and memory-level parallelized application can be compiled directly on the SPE's and thereby requires no rewriting. As mentioned, the speedup gained using the task and memory-level parallelization is 4 on 6 SPE's. Moving from the tasks-and memory-level parallelized version to the register-line version with branch elimination improved the speedup from 4 to 6.5 using 6 SPE's, but this required a rewrite of the compute intensive part using 203 lines of code opposed to the 75 lines of code used in the original recursive version. The vectorized version resulted in 348 lines of code, but didn't perform at all, due to unbalanced branch-vectors, however if one has a well balanced

[4] The iterative register-line version was the one performing best
[5] Called dual-instructions in Cell-BE terminology

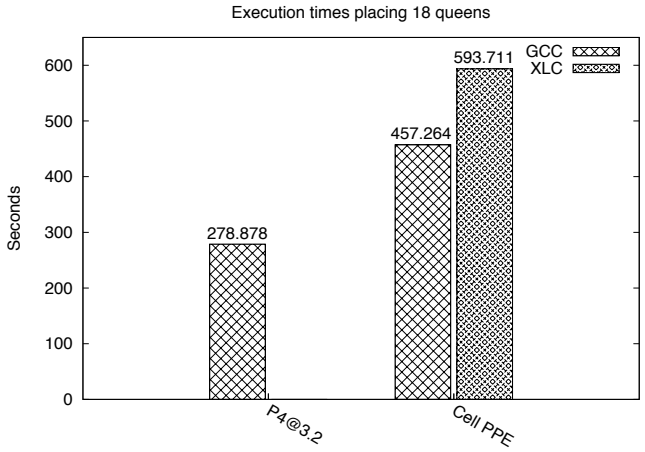


Fig. 8. Execution times for the PPE nqueens application

branch-vector or data is SIMD applicable one can expect a vectorized version to perform well, but it's not possible to reach beyond a speedup of 128/X, where X is the size of each data entry in the vector, compared to a register-line version.

3 Conclusion

The basic principles of porting an X86 application towards the Cell BE architecture has been presented in this paper as well as tuning methods including memory-, data-, and instruction-level parallelization. The porting of the nqueens test application shows that the execution time for placing 18 queens on an 18x18 chess board was reduced from 278.878 seconds on a Pentium 4 running at 3.2 GHz to 69.457 seconds, yielding a speedup of 4 just by applying standard task- and memory-parallelization techniques known from cluster computing. This was achieved without rewriting the compute intensive part towards the Cell BE architecture. Rewriting the compute intensive part towards the Cell-BE architecture reduced the execution time of placing 18 queens to 42.486 seconds, yielding a speedup of 6.5 compared to a Pentium 4 running at 3.2 GHz. The vectorized version of the nqueens application didn't yield any speedup as the average depth reached in the search tree increased by a factor of 0.3 neglecting the performance gain reached by investigating the four branches simultaneously. To gain speedup by vectorizing, the vector-branches need to be balanced, or the data needs to have a SIMD nature.

References

1. IBM Full-System Simulator for the Cell Broadband Engine Processor. http://www.alphaworks.ibm.com/tech/cellsystemsim.
2. *Berliner Schachgesellschaft*, 3:363, 1848.
3. Cell broadband engine programming handbook. pages 891–703, 2007. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9F820A5FFA3ECE8C8725716A0062585F/$file/CBE_Handbook_v1.1_24APR2007_pub.pdf.
4. Thomas Chen, Ram Raghavan, Jason Dale, and Eiji Iwata. Cell broadband engine architecture and its first implementation. *IBM developerWorks*, 2005. http://www.ibm.com/developerworks/power/library/pa-cellperf.
5. Edsger W. Dijkstra. Chapter i: Notes on structured programming. pages 72–82, 1972.
6. Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38,8, 1965. ftp://download.intel.com/research/silicon/moorespaper.pdf.
7. T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I:11–14, Oconomowoc, WI, 1995.
8. Michael Sung. SIMD parallel processing. 2000. http://www.ai.mit.edu/projects/aries/papers/writeups/darkman-writeup.pdf.
9. Takaken. http://www.ic-net.or.jp/home/takaken/e/queen/index.html.

Appendix C

Publication 3

Proceedings of the 2008 International Conference on Grid Computing & Applications,
GCA 2008, Las Vegas, Nevada, USA, July 14-17, 2008
ISBN: 1-60132-068-X, pp. 90-95
Martin Rehr, Brian Vinter: The PS3^{TM} Grid-Resource Model

The PS3® Grid-resource model

Martin Rehr and Brian Vinter
eScience center, University of Copenhagen, Copenhagen, Denmark

Abstract—*This paper introduces the PS3® Grid-resource model, which allows any Internet connected Playstation 3 to become a Grid Node without any software installation. The PS3® is an interesting Grid resource as each of the over 5 millions sold world wide contains a powerful heterogeneous multi core vector processor well suited for scientific computing. The PS3® Grid node provides a native Linux execution environment for scientific applications. Performance numbers show that the model is usable when the input and output data sets are small. The resulting system is in use today, and freely available to any research project.*

Keywords: Grid, Playstation 3, MiG

1. Introduction

The need for computation power is growing daily as an increasing number of scientific areas use computer modeling as a basis for their research. This evolution has led to a whole new research area called eScience. The increasing need of scientific computational power has been known for years and several attempts have been made to satisfy the growing demand. In the 90's the systems evolved from vector based supercomputers to cluster computers which is build of commodity hardware leading to a significant price reduction. In the late 90's a concept called Grid computing[7] was developed, which describes the idea of combining the different cluster installations into one powerful computation unit.

A huge computation potential beyond the scope of cluster computers is represented by machines located outside the academic perimeter. While traditional commodity machines are usually PC's based on the X86 architecture a whole new target has turned up with the development and release of the Sony Playstation 3 (PS3®). The heart of the PS3® is the Cell processor, The Cell Broadband Engine Architecture (Cell BE)[4] is a new microprocessor architecture developed in a joint venture between Sony, Toshiba and IBM, known as STI. Each company has their own purpose for the Cell processor. Toshiba uses it as a controller for their flat panel televisions, Sony uses it for the PS3®, and IBM uses it for their High Performance Computing (HPC) blades. The development of the Cell started out in the year 2000 and involved around 400 engineers for more than four years and consumed close to half a billion dollars. The result is a powerful heterogeneous multi core vector processor well suited for gaming and High Performance Computing (HPC)[8].

1.1. Motivation

The theoretical peak performance of the Cell processor in the PS3® is 153,6 GFLOPS in single precision and 10.98 GFLOPS in double precision[4][1]. According to the press more than 5 million PS3's have been sold worldwide at October 2007. This gives a theoretical peak performance of more than 768.0 peta-FLOPS in single precision and 54.9 peta-FLOPS in double precision, if one could combine them all in a Grid infrastructure. This paper describes two scenarios for transforming the PS3® into a Grid resource, firstly the Native Grid Node (NGN) where full control is obtained of the PS3®. Secondly the Sandboxed Grid Node (SGN) where several issues have to be considered to protect the PS3® from faulty code, as the machine is used for other purposes than Grid computing.

Folding@Home[6] is a scientific distributed application for folding proteins. The application has been embedded into the Sony GameOS of the PS3®, and is limited to protein folding. This makes it Public Resource Computing as opposed to our model which aims at Grid computing, providing a complete Linux execution environment aimed at all types of scientific applications.

2. The Playstation 3

The PS3® is interesting in a Grid context due to the powerful Cell BE processor and the fact that the game console has official support for other operating systems than the default Sony GameOS.

2.1. The Cell BE

The Cell processor is a heterogeneous multi core processor consisting of 9 cores, The Primary core is an IBM 64 bit power processor (PPC64) with 2 hardware threads. This core is the link between the operating system and the 8 powerful working cores, called the SPE's for Synergistic Processing Element. The power processor is called the PPE for Power Processing Element, figure 1 shows an overview of the Cell architecture. The cores are connected by an Element Interconnect Bus (EIB) capable of transferring up to 204 GB/s at 3.2 GHz. Each SPE

1. The PS3® Cell has 6 SPE's available for applications. Each SPE is running at 3.2 GHz and capable of performing 25.6 GFLOPS in single precision and 1.83 GFLOPS in double precision.

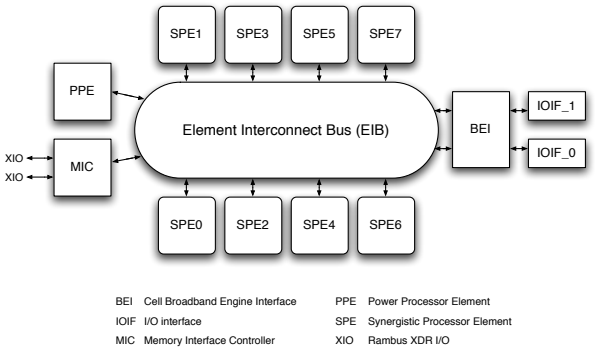


Figure 1. An overview of the Cell architecture

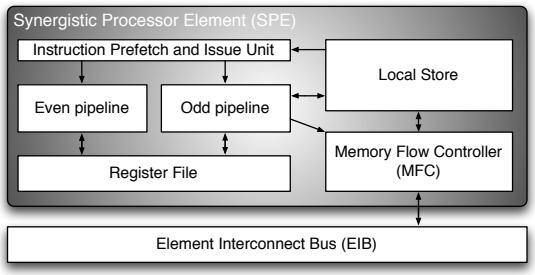


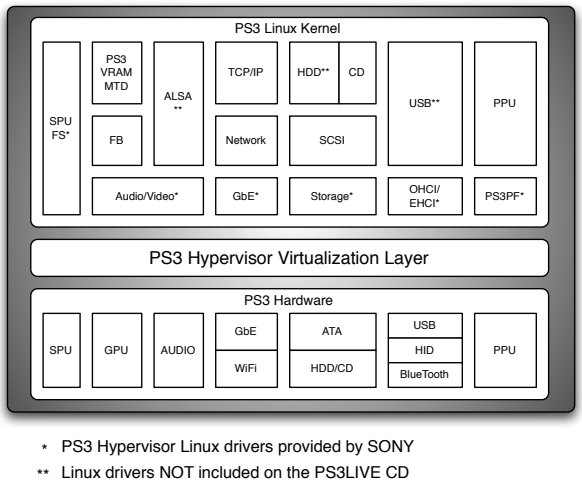
Figure 2. An overview of the SPE

is dual pipelined, has a 128x128 bit register file and 256 kB of on-chip memory called the local store. Data is transfered asynchronously between main memory and the local store through DMA calls handled by a dedicated Memory Flow Controller (MFC). An overview of the SPE is shown in figure 2.

By using the PPE as primary core, the Cell processor can be used out of the box, due to the fact that many existing operating systems support the PPC64 architecture. Thereby it's possible to boot a PPC64 operating system on the Cell processor, and execute PPC64 applications, however these will only use the PPE core. To use the SPE cores it's necessary to develop code specially for the SPE's, which includes setting up a memory communications scheme using DMA through the MFC.

2.2. The game console

Contrary to other game consoles, the PS3® officially supports alternative operating systems besides the default Sony Game OS. Even though other game consoles can be modified to boot alternative operating systems, this requires either an exploit of the default system or a replacement of the BIOS. Replacing the BIOS is intrusion at the highest level, expensive at a large volume and not usable beyond the academic perimeter. Security exploits are most likely to be patched within the next firmware update, which makes this solution unusable in any scenario. Beside the difficulties modifying other game consoles towards our purposes, the processors used by the game consoles currently on the market, except for the PS3®, are not of any interest for scientific computing.



* PS3 Hypervisor Linux drivers provided by SONY
** Linux drivers NOT included on the PS3LIVE CD

Figure 3. An overview of the PS3® Hypervisor structure for the Grid-resource model

The fact that the PS3® is low priced from a HPC point of view, equipped with a high performance vector processor, and supports alternative operating systems, makes it interesting both as an NGN node and an SGN node. All sold PS3's can be transformed to a powerful Grid resource with a little effort from the owner of the console. Third party operating systems work on top of the Sony GameOS, which acts as a hypervisor for the guest operating system. See figure 3. The hypervisor controls which hardware components are accessible from the guest operating system. Unfortunately the GPU is not accessible by guest operating systems[2], which is a pity, as it in itself is a powerful vector computation unit with a theoretical peak performance of 1.8 tera-FLOPS in single precession. However 252 MB of the 256 MB GDDR3 ram located on the graphics card can be accessed through the hypervisor, The hypervisor reserves 32 MB of main memory and 1 of the 7 SPE's available in the PS3® version of the Cell processor[3]. This leaves 6 SPE's and 224 MB of main memory for guest operating systems. Lastly a hypervisor model always introduces a certain amount of performance decrease, as the guest operating system does not have direct access to the hardware.

3. The PS3® Grid resource

The PS3® supports alternative operating systems, making the transformation into a Grid resource rather trivial, as a suitable Linux distribution and an appropriate Grid client are the only requirements. However if you target a large amount of PS3's this becomes cumbersome. Furthermore if the PS3's

2. It is not clear whether it's to prevent games to be played outside Sony GameOS, due to DRM issues or due to the exposure of the GPU's register-level information

3. The Cell processor consists of 8 SPE's, but in the PS3® one is removed for yield purposes, if one is defective it is removed, if none is defective a good one is removed to assure that all PS3's have exactly 6 SPE's available for applications, to preserve architectural consistency

located beyond the academic perimeter are to be reached, minimal administrational work form the donator of the PS3® is a vital requirement. Our approach minimizes the workload required transforming a PS3® into a powerful Grid resource by using a LIVECD. Using this CD, the PS3® is booted directly into a Grid enabled Linux system. The NGN version of the LIVECD is targeted at PS3's used as dedicated Grid nodes, and uses all the available hardware of the PS3®, whereas the SGN version uses the machine without making any change[4] to it, and is targeted at PS3's used as entertainment devices as well as Grid nodes.

3.1. The PS3-LIVECD

Several requirements must be met by the Grid middleware to support the described LIVECD. First of all the Grid middleware must support resources which can only be accessed through a pull based model, which means that all communication is initiated by the resource, i.e. the PS3-LIVECD. This is required because the PS3's targeted by the LIVECD are most likely located behind a NAT router. Secondly, the Grid middleware needs a scheduling model where resources are able to request specific types of jobs, e.g. a resource can specify that only jobs which are targeted the PS3® hardware model can be executed.

In this work the Minimum intrusion Grid[11], MiG, is used as the Grid middleware. The MiG system is presented next, before presenting how the PS3-LIVECD and MiG work together.

3.2. Minimum intrusion Grid

MiG is a stand-alone Grid platform, which does not inherit code from any earlier Grid middlewares. The philosophy behind the MiG system is to provide a Grid infrastructure that imposes as few requirements on both users and resources as possible. The overall goal is to ensure that a user is only required to have a X.509 certificate which is signed by a source that is trusted by MiG, and a web browser that supports HTTP, HTTPS and X.509 certificates. A fully functional resource only needs to create a local MiG user on the system and to support inbound SSH. A sandboxed resource, the pull based model, only needs outbound HTTPS[1].

Because MiG keeps the Grid system disjoint from both users and resources, as shown in Figure 4, the Grid system appears as a centralized black box[11] to both users and resources. This allows all middleware upgrades and trouble shooting to be executed locally within the Grid without any intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system that, though it appears as a centralized system in reality is distributed. The basic functionality in MiG starts by a user submitting a job to MiG and a resource sending a request for a job to execute. The resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that

4. One has to install a boot loader to be able to boot from CD's

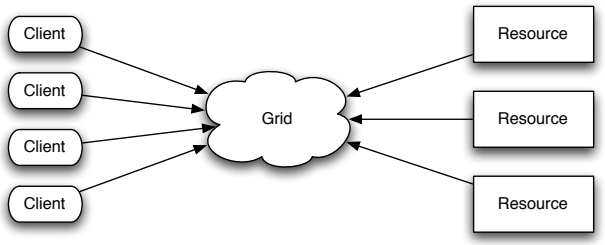


Figure 4. The abstract MiG model

can then inform the user of the job completion. Since the user and the resource are never in direct contact, MiG provides full anonymity for both users and resources, any complaints will have to be made to the MiG system that will then look at the logs that show the relationship between users and resources.

3.2.1. Scheduling. The centralized black box design of MiG makes it capable of strong scheduling, which implies full control of the jobs being executed and the resource executing them. Each job has an upper execution time limit, and when the execution time exceeds this time limit the job is rescheduled to another resource. This makes the MiG system very well suited to host SGN resources, as they by nature are very dynamic and frequently join and leave the Grid without notifying the Grid middleware.

4. The MiG PS3-LIVECD

The idea behind the LIVECD is booting the PS3® by inserting a CD, containing the Linux operating system and the appropriate Grid clients. Upon boot, the PS3® connects to the Grid and requests Grid jobs without any human interference. Several issues must be dealt with. First of all the PS3® must not be harmed by flaws in the Grid middleware nor exploits through the middleware, Secondly the Grid jobs may not harm the PS3® neither by intention nor by faulty jobs. This is especially true for SGN resources where an exploit may cause exposure of personal data.

4.1. Security

To keep faulty Grid middleware and jobs from harming the PS3®, both the NGN and SGN model use the operating system as a security layer. The Grid client software and the executed Grid jobs are both executed as a dedicated user, who does not have administrational rights of the operating system. The MiG system logs all relations between jobs and resources, thus providing the possibility to track down any job.

4.2. Sandboxing

The SGN version of the LIVECD operates in a sandboxed environment to protect the donated PS3® from faulty middleware and jobs. This is done by excluding the device driver for the PS3® HDD controller from the Linux kernel

used, and keeping the execution environment in memory instead. Furthermore, the support for loadable kernel modules is excluded, which prevents Grid jobs from loading modules into the kernel, even if the OS is compromised and root access is achieved.

4.3. File access

Enabling file access to the Grid client and jobs without having access to the PS3's hard drive is done by using the graphics card's VRAM as a block device. Main memory is a limited resource[5], therefore using the VRAM as a block device is a great advantage compared to the alternative of using a ram disk, which would decrease the amount of main memory available for the Grid jobs. However the total amount of VRAM is 252 MB and therefore Grid jobs requiring input/output files larger than 252 MB are forced to use a remote file access framework[2].

4.4. Memory management

The PS3® has 6 SPE cores and a PPE core all capable of accessing the main memory at the same time, through their MFC controllers. This results in a potential bottleneck in the TLB, as it in the worst case ends up thrashing, which is a known problem in multi core processor architectures. TLB thrashing can be eliminated by adjusting the page size to fit the TLB, which means that all pages have an entry in the TLB. This is called huge pages, as the page size grows significantly. The use of huge pages has several drawbacks, one of them is swapping. Swapping in/out a huge page results in a longer execution halt as a larger amount of data has to be moved between main memory and the hard drive.

The Linux operating system implements huge pages as a memory mapped file, this results in a static memory division of traditional pages and huge pages, using different memory allocators. The operating system and standard shared libraries use the traditional pages which means the memory footprint of the operating system and the shared libraries has to be estimated in order to allocate the right amount of memory for the huge pages. Opposite to a cluster setup where the execution environment and applications are customized to the specific cluster, this can't be achieved in a Grid context[6]. Therefore a generic way of addressing the memory is needed. Furthermore future SPE programming libraries will most likely use the default memory allocator. This and the fact that no performance measurement clarifying the actual gain of using huge pages could be found, led to the decision to skip huge pages for the PS3-LIVECD.

At last it's believed by the authors that the actual applications which could gain a performance increase by using huge pages is rather insignificant, as the the majority of applications will be able to hide the TLB misses by using double- or multi buffering, as memory transfers through the MFC are asynchronous.

5. The PS3® only has 224 MB of main memory for the OS and applications

6. Specially in MiG where the user and resources are anonymous to each other

5. The execution environment

The PS3-LIVECD is based on the Gentoo Linux[9] PPC64 distribution with a customized kernel[5] capable of communicating with the PS3® hypervisor. Gentoo catalyst[3] was used as build environment, this provides the possibility of configuring exactly which packages to include on the LIVECD, as well as providing the possibility to apply a custom made kernel and initrd script. The kernel was modified in different ways, firstly loadable modules support was disabled to prevent potential evil jobs, which manages to compromise the OS security, from modifying the kernel modules. Secondly the frame-buffer driver has been modified to make the VRAM appear as a memory technology device, MTD, which means that the VRAM can be used as a block device. The modification of the frame-buffer driver also included freeing 18 MB of main memory occupied by the frame-buffer used in the default kernel[7].

The modified kernel ended up consuming 7176 kB of the total 229376 kB main memory for code and internal data structures, leaving 222200 kB for the Grid client and jobs. Upon boot the modified initrd script detects the block device to be used as root file system[8] and formats the detected device with the ext2 filesystem, reserving 2580 kB for the superuser, leaving 251355 kB for the Grid client and jobs[9]. When the block device has been formatted, the initrd script sets up the root file system by coping writable directories and files from the CD to the root file system. Read-only directories, files, and binaries are left on the CD and linked symbolically to the root filesystem keeping as much of the root filesystem free for Grid jobs as possible. The result is that the root file system only consumes 1.6 MB of the total space provided by the used block device.

When the Linux system is booted the LIVECD initiates the communication with MiG through HTTPS. This is done by sending a unique key identifying the PS3® to the MiG system, if this is the first time the resource connects to the Grid a new profile is created dynamically. The response to the initial request is the Grid resource client scripts, these are generated dynamically upon the request. By using this method it's guaranteed that the resource always has the newest version of the Grid resource client scripts, disabling the need for downloading a new CD upon a Grid middleware update. When the Grid resource client script is executed the request of Grid jobs is initiated through HTTPS. Within that request a unique resource identifier is provided, giving the MiG scheduler the necessary information about the resource such as architecture, memory, disc space and an upper time-limit. Based on these parameters the MiG scheduler finds a job suited for the PS3® and places it in a job folder on the MiG system. From this location the PS3® is able to retrieve the job consisting of

7. As the hypervisor isolates the GPU from the operating system, the display is operated by having the frame-buffer writing the data to be displayed to an array in main memory, which is then copied to the GPU by the hypervisor

8. The SGN version uses VRAM, DGN version uses the real hard drive provided through the hypervisor

9. This is true for the SGN version, the NGN version uses the total disc space available, which is specified through the Sony Game OS

job description files, input-files, and executables. The location of these files is returned within the result of the job request, and is a HTTPS URL including a 32 character random string generated upon the job request and deleted when the job terminates. At job completion the result is delivered to the MiG system which verifies that it's the correct resource (by the unique resource key) which delivers the result of the job. If it's a false deliver[10] the result is discarded, otherwise it's accepted. And the PS3® resource requests a new job when the result of the previous one has been delivered.

6. Experiments

Testing the PS3® Grid-resource model was done establishing a controlled test scenario consisting of a MiG Grid server and 8 PS3's. The experiments performed included a model overhead check, a file system benchmark using VRAM as a block device, and application performance, using a protein folding and a ray tracing application.

6.1. Job overhead and file performance

The total overhead of the model was tested by submitting 1000 empty jobs to the Grid with only one PS3® connected. The 1000 jobs completed in 12366 seconds, which translates to an overhead of approximately 13 seconds per job. The performance of the VRAM used as a block device was tested by writing a 96 MB file sequentially. This was achieved in 1.5 seconds, resulting in a bandwidth of 64 MB/s. Reading the written file was achieved in 9.6 seconds, resulting in a bandwidth of 10 MB/s. This shows that writing to the VRAM is a factor of approximately 6.5 faster than reading from the VRAM, which was an expected result as the nature of VRAM is write from main memory to VRAM, not the other way around.

6.2. Protein folding

Protein folding is a compute intensive algorithm for folding proteins. It requires a small input and generates a small output, and is embarrassing parallel which makes it very suitable for Grid computing. In this experiment, a protein of length 27 was folded on one PS3® resulting in a total execution time of 57 minutes and 16 seconds. The search space was then divided into 17 different subspaces using standard divide and conquer techniques. The 17 different search spaces were then submitted as jobs to the Grid, which adds up to 4 jobs for each of the 4 nodes used in the experiment plus one extra job to ensure unbalanced execution. Equivalently, the 17 jobs were distributed among 8 nodes, yielding 2 jobs per node plus one extra job. The execution finished in 18 minutes and 50 seconds using 4 nodes giving a speedup of 3.04. The 8 node setup finished the execution in 10 minutes and 56 seconds giving a speedup of 5.23, this is shown in figure 5. These results are considered quite useful in a Grid setup as opposed to a cluster setup where this would be considered bad.

10. The resource keys doesn't match, the time limit has been violated, or another resource is executing the job, due to a rescheduling

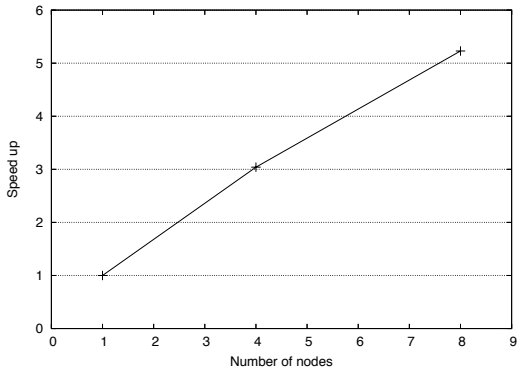


Figure 5. The speedup achieved using the PS3-LIVECD for protein folding with 4 and 8 nodes

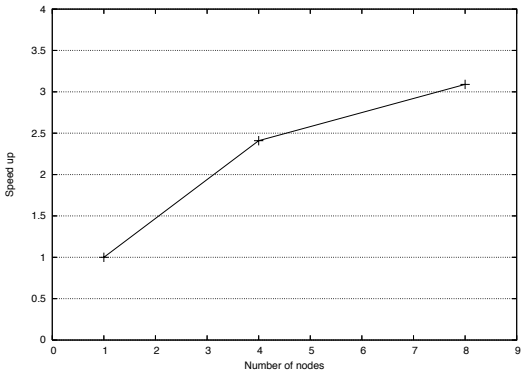


Figure 6. The speedup achieved using the PS3-LIVECD for ray tracing with 4 and 8 nodes

6.3. Ray tracing

Ray tracing is compute intensive, requires a small amount of input and generates a large amount of output. This experiment uses a Ray tracing code written by Eric Rollings[10], this has been modified from a real time ray tracer to a ray tracer which writes the rendered frames to files in a resolution of 1920x1080 (Full HD). The final images are jpeg compressed to reduce the size of the output. A total of 5000 frames were rendered in 78 minutes and 6 seconds on a single PS3®, the search space was then divided into 25 equally large subspaces. These were submitted as jobs to the Grid resulting in a total of 25 jobs, which adds up to 6 jobs per node plus one extra in the 4 node setup, and 3 jobs per node plus one extra in the 8 node setup. The execution time using 4 nodes was 32 minutes and 23 seconds giving a speedup of 2.41 and the execution time using 8 nodes was 25 minutes and 12 seconds giving a speedup of 3.09, this is sketched in figure 6. While the speedup achieved with 4 nodes is quite useful in a Grid context, the speedup gained using 8 nodes is quite disappointing. The authors

believe this is due to network congestion when the rendered frames are sent to the MiG storage upon job termination.

7. Conclusion

In this work we have demonstrated a way to use the Sony Playstation 3 as a Grid computing device, without the need to install any client software on the PS3®. The use of the Linux operating system provides a native execution environment suitable for the majority of scientific applications. The advantage of this is that existing Cell applications can be executed without any modifications. A sandboxed version of the execution environment has been presented which denies access to the hard drive of the PS3®. The advantage of this is that donated PS3's can't be compromised by faulty or evil jobs, the disadvantage is the lack of file access, which is solved by using the VRAM of the PS3 as block device.

The Minimum intrusion Grid supports the required pull-job model for retrieving and executing Grid jobs on a resource located behind a firewall without the need to open any incoming ports. By using the PS3-LIVECD approach any PS3® connected to the Internet can become a Grid resource by booting it with the LIVECD. When a Grid connected PS3® is shut down the MiG system will detect this event, by a timeout, and resubmit the job to another resource.

Experiments show that the ray tracing application doesn't scale well, due to the large amount of output data resulting in network congestion problems. Opposite to this, a considerable speedup is reached when folding proteins despite of the model overhead of 13 seconds applied to each job.

References

[1] Rasmus Andersen and Brian Vinter. Harvesting idle windows cpu cycles for grid computing. In Hamid R. Arabnia, editor, *GCA*, pages 121–126. CSREA Press, 2006.

[2] Rasmus Andersen and Brian Vinter. Transparent remote file access in the minimum intrusion grid. In *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 311–318, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Gentoo Catalyst. http://www.gentoo.org/proj/en/releng/catalyst.

[4] Thomas Chen, Ram Raghavan, Jason Dale, and Eiji Iwata. Cell broadband engine architecture and its first implementation. *IBM developerWorks*, 2005. http://www.ibm.com/developerworks/power/library/pa-cellperf.

[5] PS3 Linux extensions. ftp://ftp.uk.linux.org/pub/linux/Sony-PS3.

[6] Folding@home. http://folding.stanford.edu.

[7] Ian Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, 2002.

[8] Mohammad Jowkar. Exploring the Potential of the Cell Processor for High Performance Computing. Master's thesis, University of Copenhagen, Denmark, August 2007.

[9] Gentoo Linux. http://www.gentoo.org.

[10] Eric Rollings. Ray tracer. http://eric_rollins.home.mindspring.com/ray/ray.html.

[11] Brian Vinter. The Architecture of the Minimum intrusion Grid (MiG). In *Communicating Process Architectures 2005*, sep 2005.

Appendix D

Publication 4

IEEE HPCC-10, the 12th edition of the highly successful International Conference on High Performance and Communications (HPCC), Melbourne, Victoria, Australia, September 1-3, 2010. Proceedings to appear
Martin Rehr, Brian Vinter: The User-level Remote Swap Library

The User-level Remote Swap Library

Martin Rehr and Brian Vinter
eScience center, University of Copenhagen, Copenhagen, Denmark

***Abstract*—*This paper introduces the User-level Remote Swap Library, URSL, which enables memory-homogeneous Grid execution. This is obtained by letting the Grid infrastructure supply a remote memory bank to the connected resources. Insufficient memory at the contribution computer resource in a Grid is a major limitation when users seek to get high throughput in a Grid infrastructure, simply because a number of the available computers do not offer enough memory for the user job to run. URSL bypasses the lack of memory by offering the required memory as part of the Grid infrastructure in the form of a user-level swapping resource. A set of dedicated memory servers provide the additional memory that the computing resources then access remotely through the Grid infrastructure. Utilization of the extra memory is obtained without modifications to neither the operating system of the host resource nor the executed applications. The paper includes experiments that are made in an isolated Grid framework, and shows that the framework is fully operational, transparent and outperforms swapping to local disk in both the synthetic micro-benchmarks and in real-life scientific applications.*

Keywords: Remote Swap, Grid

I. Introduction

The scientific modeling community has a seemingly endless need for processing power as new areas of modeling arise steadily and existing models become increasingly fine grained and realistic. To be able to keep up with the growing demand for processing power the Grid[13] paradigm was introduced in the late 90's with the purpose of providing an infrastructure that combines super-computer installations located at different research institutions into one high performance infrastructure which in turn enables sharing the computer resources among the researchers across organizations. In the same period Berkeley University introduced Public Resource Computing (PRC) in the form of the SETI@home[4] project, which later turned into the BOINC[3] framework. The PRC paradigm differs from the Grid computing paradigm by focusing on the large set of computing devices that are located outside the super-computer facilities, which has been demonstrated to have a huge computation potential. At the time of writing the most successful PRC project FOLDING@home[8] has a total of 325638 active CPU's that contribute a theoretical total of 4432 TFLOPS[1], which potentially makes it the second fastest super-computer installation in the world right now[2]. A number of research projects have shown ways to combine PRC computing and PRC computing, the first using BOINC resources within a Grid infrastructure[16] and the latter by harvesting generic windows resources[1] and Playstation 3 resources[17] in a Grid environment. This work provides a possibility of utilizing the processing power that was previously only applicable to PRC computing in a Grid infrastructure, because the resources would not offer enough memory for the Grid jobs. While PRC resources offer a huge boost to the theoretical processing power that is available in a Grid infrastructure it also introduces a number of new challenges. Using millions of PRC resources is necessarily more heterogeneous than a few super-computer installations, which impacts the size of the set of resources that are capable of executing any given job and thus the overall utilization of the Grid system. There are four levels of homogeneity, which needs to be addressed: Hardware-, OS-, disk- and memory-homogeneity. Hardware- and OS-homogeneity may be addressed by wrapping the Grid execution environment into a virtual machine image, disk-homogeneity may be applied by using a remote file library[2], but so far there is no mechanism for providing memory homogeneity in a Grid framework. We propose the Remote User-Level swap framework in order to enable memory homogeneity to Grid computing and thus increase the total system utilization[3] This framework provides Grid resources with a mechanism for seamlessly accessing memory that is located in the Grid infrastructure, without modifying neither the resource nor the job that is executed.

A. Related Work

Several research groups have looked into the idea of swapping to a remote machine rather than a local disk. D. Comer and J. Griffioen[10] present a model where a dedicated memory server is used to store pages that are swapped out by the clients. E. Markatos and G. Dramitinos[14] present a reliable remote memory pager that makes use of free memory in the nodes of a cluster. E. Anderson and J. Neefe[5] present a user-level remote memory pager that targets Network Of Workstations (NOW's). R.T. Mills, C. Yue, A. Stathopoulos and D.S. Nikolopoulos[15] presents a user-level remote memory system for scientific applications that uses local disk and dedicated memory servers. R. Chu et. al.[9] present remote memory paging using NOW's in a Grid environment.

The kernel-level models presented in most of the previous work are poorly suited for global Grid environments, because these models require modifications to the operating system of

[1]According to their web-site:http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats

[2]According to the Top-500 list of the worlds fastest super-computer installations at November 2009 (http://www.top500.org/list/2009/11/100). The Top 500 list is based on the Linpack benchmark suite, which is not applicable to a PRC environment, therefore the comparison is theoretical.

[3]Providing the throughput is bound by memory constraints

the executing host resources. While this is acceptable in an isolated homogeneous cluster environment it's difficult if not impossible to deploy in global Grid environments that have thousands of executing hosts and host administrators. With this in mind we chose to extend the existing solutions by combining the idea of dedicated memory servers that was presented by D. Comer and J. Griffioen and the user-level idea presented by E. Anderson and J. Neefe integrated with the Grid approach taken by Chu et al. But opposed to the previous user-level solutions, which requires the use of non-standard memory routines, our solution works transparently to the application and thereby transparently to the application writer and ultimately the Grid user. This leads to a solution that differs from the previous work by providing a Remote Memory framework for global Grid environments with resources ranging from the traditional cluster computer installations all the way to PRC devices. That is, a solution which is deployed through the Grid infrastructure and is fully transparent to both the execution hosts and to the application writer.

II. The Grid infrastructure

The Grid computing paradigm is a natural extension to cluster computing, that aims at assembling a global network of computation resources into one shared, although not parallel, infrastructure. Where the devices used in cluster computers are homogeneous, the devices used in a Grid system are heterogeneous, in all aspects of the hardware configuration, the resource location, and the administrative domain they run within. In this work we aim at improving the usability of Grid resources by providing them with more memory than they offer locally. By letting the Grid infrastructure provide a memory service to supplement local memory, the Grid may take advantage of resources, which would otherwise be idle because no jobs fit the memory they offer. The end result is that the total utilization of the Grid system is increased as a broader set of resources are capable of executing the pending Grid jobs.

A. Grid Resource Memory

In the perfect world a computing device would hold sufficient physical memory to allow all running processes and their data in memory. Naturally this is not the case and we thus need mechanisms to administrate the available physical memory to help optimize utilization of the hardware. This is usually done through the OS memory manager, which has global knowledge of the memory usage of all running processes in the system. However, changes to the operating system is a limiting factor in the adaptation of any system, thus in this work we aim at deploying a user-level swap library, which is placed between the OS and the application and intercepts memory needs before they reach the OS memory manager. Using this approach we introduce a Grid-enabled memory manager that runs non-intrusively at any Grid resource.

1) Memory management: All modern computer architectures have a memory management unit and use an operating system, which provides a full virtual address space to each process. This gives the processes an impression of having exclusive access to a system where the upper limit of memory is bound only by the hardware architecture. While providing a full virtual address space for each process is a powerful abstraction, it also places a large responsibility on the OS to manage the available physical memory. Traditionally this is achieved by swapping memory to disk when the system runs short on physical memory. If the system exhausts the available disk space for swapping, the OS has to decide what to do. Typically the policy is killing a process to free up memory.

2) Kernel-level vs. user-level: The OS memory manager is responsible for swapping page-frames between physical memory and secondary memory, typically a hard-disk. In Linux the swap device is implemented as a generic block device, which makes it easy to change swap-target implementation as one can plug these directly into the kernel just by emulating the behavior of a block device. This approach has the advantage that the memory manager is left unmodified and works independently of which underlying media is actually used as storage for the swapped out memory. The downside to this approach is that it requires administrator privileges to deploy such a block device into the kernel, and implicitly that the administrator will have to trust the code that is added to the memory manager since it will run with kernel privileges.

While the kernel-level approach is easy to implement due to the cleaner interface with the OS, the user-level model is more flexible in a Grid context since it overrides the local memory manager and thus provides a homogeneous swapping mechanism to the Grid resources without requiring administration privileges or implicit trust to the swapping module. This enables the Grid infrastructure to fully control the amount of physical memory that the executing application is allowed to use on the resource [4]. The user-level model however has several drawbacks, primarily that you have to implement a new memory manager to work on top of the native memory manager. However reusing most of the OS memory manager and overloading only a few functionalities such as allocation of memory, freeing memory and swapping pages in and out of physical memory may be sufficient. Another drawback of the user-level approach is that it invokes frequent switching between user- and kernel-level, which results in an execution time overhead compared to the native model where everything is done in kernel space. In addition an overhead in memory consumption is imposed because the user-level library must maintain a set of internal data structures to keep track of the state and location of pages that are used by the application. Last but not least a user-level library doesn't have access to the hardware supported status bits of the process page-table, which has the effect that the widely used LRU eviction algorithm is not applicable [5]

Despite the overhead of the user-level approach compared to

[4]Naturally it's not possible to utilize more physical memory than present at the resource

[5]LRU uses the hardware page referenced bit, which is not accessible at user-level. A workaround can be made, but introduces a significant overhead due to switching between user- and kernel-level

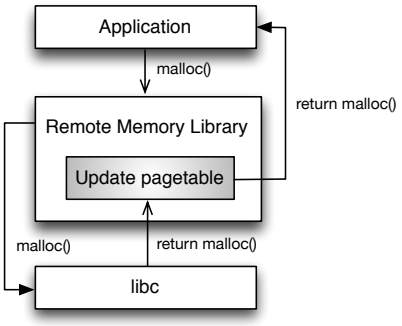


Fig. 1. The flow of a malloc call from the user application

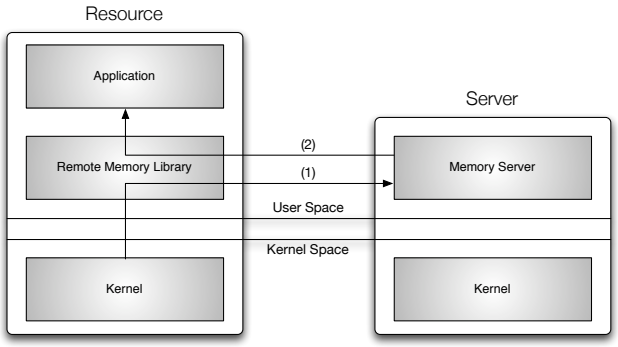


Fig. 2. The flow of swapping in a page from the remote server. (1) The kernel catches a page access violation which is sent to the user process but intercepted by RML and transformed into a server page request. (2) The server responds with the page needed which is transferred to the client and mapped into the right position in memory and control is given back to the user process

the kernel-level approach, we choose to make our remote swap library run at user-level. This is chosen because it imposes fewer requirements for deployment in Grid environments, as well as the opportunity to change the page replacement algorithms with algorithms that are more suited towards a Grid environment.

III. The Remote Swap framework

The Remote Swap framework consists of two components namely a memory server and a Remote Memory Library (RML). This memory library is interposed in user-level between the OS and the user process on the executing Grid resource to provide a transparent user-level swap mechanism. It's responsible for allocating memory, freeing memory and evicting pages to the remote memory server, as well as bringing pages back into physical memory, when they are once again needed.

A. The Remote Memory Library

The goal of making RML transparent to both the user application and the underlying OS implies that RML should support all the memory routines that are available to the application writer in the original environment. In this initial version of RML, Linux heap memory is the target for remote swapping, making the libc routines *malloc, calloc, realloc* and *free* the ones supported in RML. Rather than re-writing and maintaining these routines, they are merely overloaded with the purpose of maintaining a local page-table in RML to keep track of the pages in use by the user process. The actual memory allocation is done by calling the generic memory routines in libc from within the overloaded routines. This is illustrated in figure 1.

1) Page eviction: When the user process reaches its physical memory limit pages need to be evicted. The target pages are found using the second chance FIFO evict strategy rather than the LRU strategy, which is used by most OSs, because LRU is not feasible in a user-level environment. When the pages to be evicted are found, they are protected in read mode to ensure consistency by preventing any other active user threads from modifying them during eviction. If the chosen pages were modified while resident in memory, they are sent to the remote memory server through the network. The physical memory used for the evicted pages are then released and the virtual pages are protected in order to detect when the user process tries to access them.

2) Page retrieval: When the user process tries to access a protected page, the kernel will send, an access violation signal to the user process. This signal is intercepted by RML, which checks the state of the violated page in its local page-table. If the page has previously been swapped out, a page request is sent to the remote server in order to retrieve the page. The server responds with the page data, which is placed into the correct page slot and control is given back to the user process which can continue execution. This is shown in figure 2. If RML doesn't have any information about the violated page the access violation signal is forwarded to the user process which then has to handle the signal.

3) Page blocks: The memory manager in modern OSs arrange memory into an abstraction called page-frames, which is blocks of contiguous bytes. In the same manner RML arranges blocks of contiguous pages into what we call page blocks in order to swap out and retrieve several contiguous pages in a single evict or retrieve operation. This is beneficial if the executing application has a sequential memory access pattern across page blocks, that is the byte access pattern within a page block can be scattered as long as it doesn't access bytes outside the page block or its adjacent neighbors. Not all scientific applications have a strictly sequential memory access pattern, but even then they may still take advantage of using blocks of pages when evicting or retrieving pages. This is clarified in the experiments section. The optimal number of pages to block into one page block is dependent on the network latency between the client and the memory server, and the memory access pattern of the executing application. In this first version of RML the page block size is provided to the framework before execution. Adaptive adjustment towards the needs of the executing algorithm is subject for future work.

B. The Memory Server

The memory server is a user-level process communicating with the clients through a TCP socket. Two kinds of services are offered by the memory server: Page send and page retrieve. When the client asks for a service the index of the page to send/retrieve is sent along with the request. The server stores the pages in memory and uses a hash table with the page index as key and the memory address where the page is stored locally at the memory server as value. When a page is received at the server the page index is looked up in the hash table to check if memory has previously been allocated for the specific page due to an earlier swap-out. If the page has an entry in the hash table the memory associated with it is reused, otherwise memory is allocated for the new page and the key/value pair is inserted into the hash table.

Upon a page retrieve request the server makes a lookup in the hash table, if the page index is present in the hash table the server responds with the data associated with the page index otherwise an error message is sent to the client.

IV. Implementation details

The current implementation of URSL is bound to the Linux kernel, but is portable to any page based operating system that supports virtual memory and functions to manipulate the virtual page tables such as mmap, mremap, munmap and mprotect or equivalents. In addition to the page table manipulation functions, support for overloading the default signal handler is required in order to detect page access violations at user-level. URSL is loaded in between the system level libraries such as libc and the user application by using the LD_PRELOAD environment variable. This way the library can be a part of any Grid job without involving the resource owner, as it's delivered along with the Grid application and loaded as a part of the Grid job. When the library is initialized malloc, calloc, realloc and free is overloaded and thereby every call to these functions passes through URSL, which means that the user application doesn't need to use customized memory functions in order to use the framework, this is what provides the transparency to user applications.

A. Memory allocation

In order to avoid re-implementing the existing memory allocation routines, URSL merely uses the original malloc, calloc and realloc routines to allocate memory. When allocating a chunk of memory, the address returned by the original allocator along with the chunk size is translated into page indexes. These indexes are used as keys for a local page table containing the page state information. The first and the last page are typically used by the original allocator to store internal information about the allocated memory and thereby these are mapped to real memory, the pages in between are merely marked as allocated both at operating system level and in the local URSL page table. These pages are access protected by URSL using the mprotect routine in order to detect when they are activated. The framework then returns the allocated memory address to the running application which continues its execution. When one of the newly allocated access protected pages is accessed by the running application, an access violation signal is thrown by the kernel. This signal is caught by URSL and the page is looked up in the local page table. If the page is marked as allocated, but not yet used, the page is re-protected in mode write, the page table is updated, and execution is returned to the running application.

B. Page eviction

The user-level approach makes it possible to regulate the real memory usage of a single application without considering other active applications on the system. When a page becomes active, either by activating a new page or swapping in a previously used page, it's checked if the upper active page limit has been reached. If the limit is reached pages are chosen for eviction using the second chance FIFO algorithm and thereafter protected in *read* mode, to prevent other active threads from modifying the pages during eviction. Modified pages are then sent to the remote server along with a header containing the page indexes. Finally the page states are changed to *swapped-out* and the real memory is freed by mmap'ing the evicted pages in protection *none*. This atomically frees the real memory and maps the virtual pages in *no access* mode. The evicted pages are now resident at the remote memory server until they are once again accessed from the running application.

C. Page retrieval

When pages are swapped out they are protected in *no-access* mode, this means that whenever the executing application accesses such a page, an access violation signal is sent by the kernel and trapped by URSL. The page causing the access violation is looked up in the local page table and if the page is found to be resident at the memory server, a page request is sent. The server will respond with the requested page data, which is received in a local buffer reserved for swap-in page data. When the data is fully received, the receive buffer is protected in mode *read* to detect future modifications, used to determine if the page should be written back to the server upon its next swap-out. When the receive buffer is protected the page is put into the right position in memory by using the mremap routine. This routine atomically updates the kernel virtual page table keeping other threads from accessing the page until the page is in a write safe state[6]. Finally the page is marked in URSL as *swapped-in*, the page retrieve buffer is re-allocated using mmap for future swap-in's and the execution is returned to the thread which accessed the swapped-out page.

[6]Without the remap routine one would need to make the target page writeable, in order to copy data from the buffer page to the target page slot. This would allow other user threads to modify the page before it's completely in place.

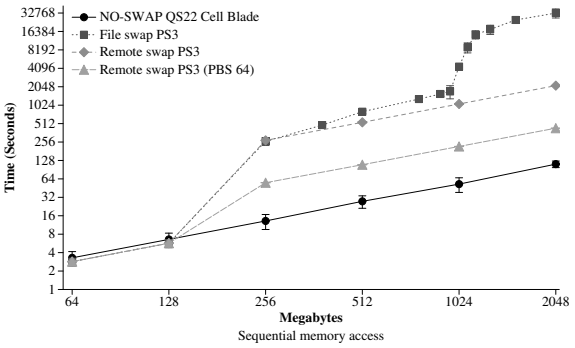


Fig. 3. Sequential memory access performance

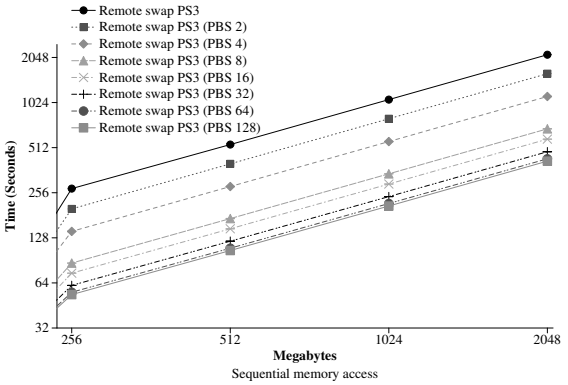


Fig. 4. Sequential remote memory performance with increasing page block sizes

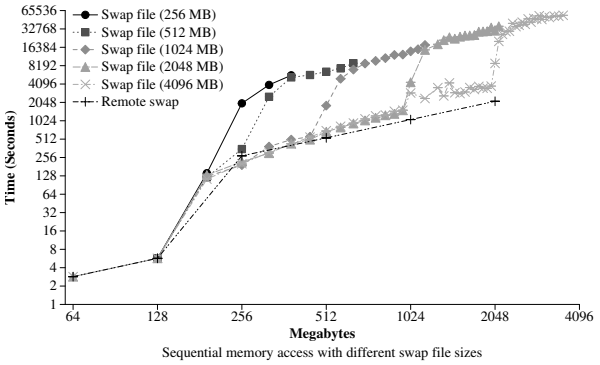


Fig. 5. Linux file swap performance

V. Experiments

To validate the Remote swap library and document its performance, experiments were done in an isolated execution environment. This consisted of a Playstation 3 execution node and an dual quad core Intel Xeon running at 1.60 GHz with 8 GB RAM as memory server. These machines were interconnected through a 1 Gb/s switch and controlled by the Minimum intrusion Grid[12]. The Playstation 3 was chosen as execution device because it represents a unextendible hardware device with a powerful processor but a limited amount of memory, namely 224 MB for the OS and user applications. As a reference machine without swap we used a Cell-BE QS22 blade. The reason for using an isolated high-bandwidth network with only one node, rather than a full Grid setup with a slower network and a number of nodes, is to validate the performance of the transparent user-level model under optimal conditions. If the model doesn't perform well under these conditions it will never perform in a real Grid system.

The URSL framework has been tested with special designed highly I/O bound sequential memory access and scattered memory access applications, as well as real scientific applications. Each of the applications were tested with different page block sizes to evaluate how this influences the performance. The results of the experiments are covered in the following subsections.

A. Sequential data access

The sequential data access tests were done by allocating N bytes of memory using malloc and then initializing the memory to make sure it was mapped to physical memory. Then a timer was started and the time spent traversing the memory start-to-end in 10 iterations (reading each byte to produce a checksum) was measured. Finally the performance was compared to the execution without swap on the Cell-BE QS22. This test is highly I/O bound as the only computation done by the executing machine is one integer addition per byte that is read. The performance of this execution with page block sizes 1 and 64 is shown in figure 3. The experiment shows that URSL outperforms swap to disk significantly, as the memory consumption increases. Furthermore, the performance increases with the page block size until it starts to converge at 64 pages per block, which is were the bandwidth of the network is saturated. This is shown in Figure 4. We found the steep raise in disk swap execution time when the memory consumption reaches 1024 MB peculiar, as we are only performing sequential reads while measuring time and would expect the disk and its caches to be able to prefetch the pages needed. Thereby we would expect the execution time to raise no more than linearly with respect to the memory consumed and perform better than a user-level remote swap library as this algorithm is highly I/O bound. To investigate this further we settled out to take a closer look at the Linux swapping scheme.

1) Linux disk swap: In this experiment we executed the sequential memory access program described in the last section using different swap file sizes to see if the execution time was affected by the size of the swap device. All swap files were made using dd with a block size of 4096 bytes. The result of the executions is shown in figure 5. This experiment shows that disk swap performance is highly dependent on the swap file size, which was an unexpected result. We discovered the reason for this by looking into the Linux kernel source code where we found that when half of the disk swap space is filled the memory manager starts to remove swapped-in pages from the swap device to prevent it from running out of space. The effect is that all swapped-in pages are marked as dirty and

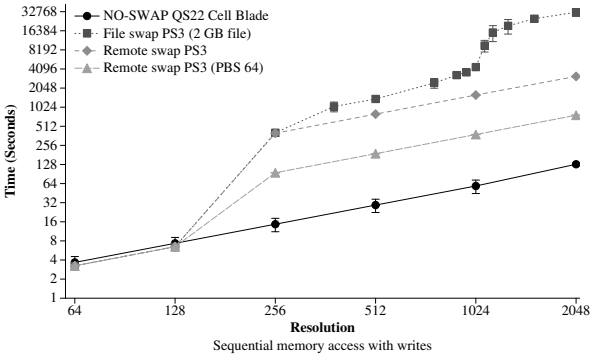


Fig. 6. Sequential memory access with writes performance

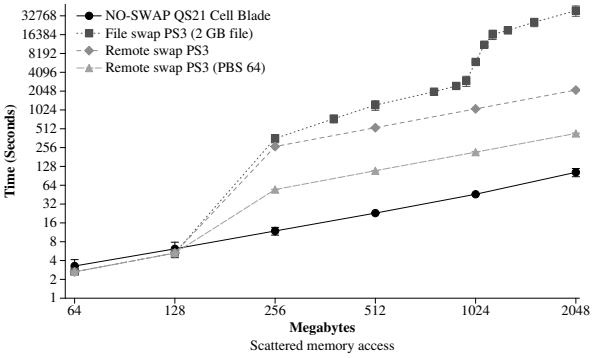


Fig. 7. Scattered memory access performance

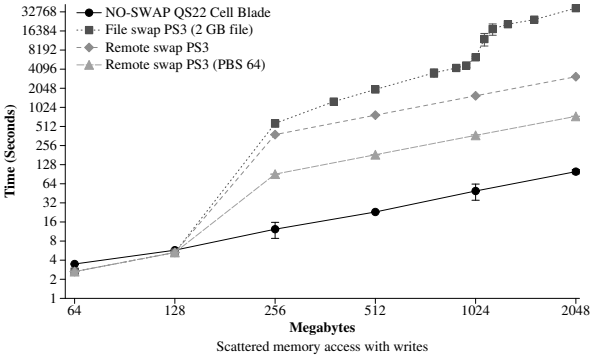


Fig. 8. Scattered memory access with writes performance

thereby needs to be re-written to disk when they are once again swapped out. This is what causes the steep raise in execution time, when the swap device becomes half full, as the sequential experiment doesn't perform any writes in the 10 iterations that are used for time measurement.

In the rest of the experiments we continued to use a 2 GB swap file to show how the artifact of the Linux kernels half-full dirty mark strategy will be expressed within the scientific applications used in the experiments.

B. Sequential data access with writes

In this test we used the sequential test described in the previous section and performed a write to each page accessed after its data had been read. The result is shown in figure 6. This experiment shows that even though a write was done to every page, it didn't eliminate the half-full Linux swap artifact, but the gap is smaller compared to the execution without writes. This is due to the fact that the Linux kernel in addition to marking the page dirty also removes it from the page cache and the swap device, which consumes time compared to the alternative of leaving the page on disk and then just writing it back out when needed. Beside the effect of the half-full artifact, it is observed that the execution time increases when using disk swap compared to remote swapping. This is caused by the mechanical structure of a disk, which causes the average seek time to rise as the amount of swapped out pages increases. As in the experiment without writes, the performance increases with the page block size until it converges at 64 pages per block, when the bandwidth of the network is saturated.

C. Scattered memory access

This test was designed similarly to the sequential access test, but instead of reading the pages in a sequential manner the data is read one page at a time, starting with the first page followed by the last page and then the second page followed by the second last page, this pattern is used until all pages are read. The result is shown in figure 7. This test shows that the execution time when swapping to disk is increasing relatively more than the execution time when swapping to the remote location. That is caused by the fact that disk swap can no longer take advantage of block prefetch combined with larger seek times when searching for the page to swap in, even before the half-full artifact arises. As with the sequential tests the performance increases with the size of the page blocks until the network is saturated at a block size of 64 pages.

D. Scattered memory access with write

This test is like the scattered access test described above, but with one write to each page like in the sequential write test. The result of this test is shown in figure 8. As with the sequential write test, the gap between Linux swap to disk and URSL decreases compared to the execution without writes and the half-full artifact of the Linux swap is still present. The performance increases with the size of the page blocks until the network is saturated at a block size of 64 pages.

E. Lattice Boltzmann

OpenLB[18] is a free library for lattice Boltzmann simulations. In this experiment we used the forcedPoiseuille2d example provided in the package with different resolution sizes. The results of this test is shown in figure 9, which displays that swapping to the remote location outperforms disk swap with a factor of 6 and is 11 times slower than no-swap execution with a resolution of 2048 and a page block size of 4, which proved to be the optimal page block size for this application (figure 10). The half-full artifact is clearly visible in this test.

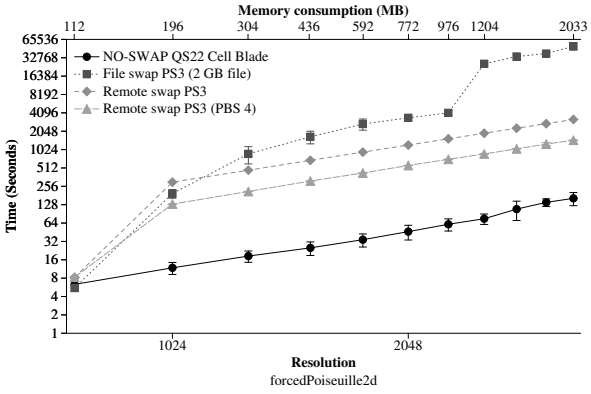


Fig. 9. Lattice Boltzmann performance

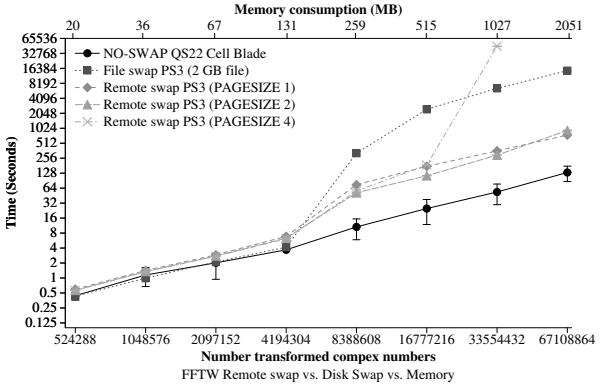


Fig. 11. FFTW performance

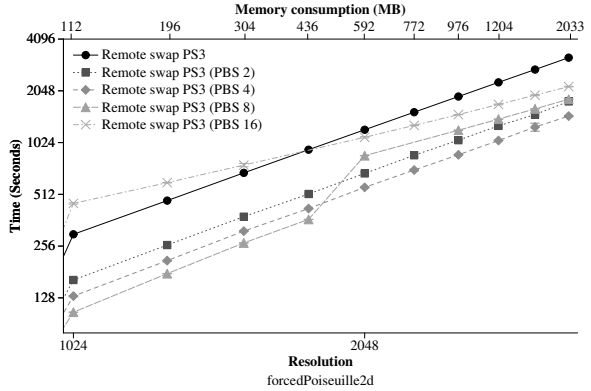


Fig. 10. Lattice Boltzmann performance with increasing page block sizes

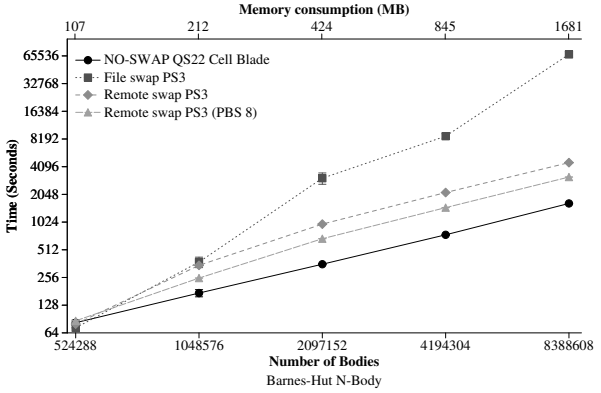


Fig. 12. Barnes-Hut performance

F. Fast Fourier Transform

Fftw[11] is a free implementation of discrete Fourier transformation. In this test we use Fftw to transform a vector of random complex numbers to their corresponding Fourier values and back to the original values. The result of this test is shown in figure 11, which displays that swapping to the remote location outperforms swapping to disk with a factor of 21 using a vector of 16777216 complex numbers and a page block size of 2. The slowdown compared to native executing at this instance is 4. Furthermore it shows that a page block size of 1 is the optimal when we reach a vector size of 67108864 complex numbers.

G. Barnes-Hut

The Barnes-Hut[6] algorithm is an O(nlogn) algorithm for performing N-Body force simulations. In this experiment we have used the code that is provided by one of the authors J. Barnes (the code can be downloaded from his ftp site[7]). The experiments were performed with the tree-body 6 test data, provided in the original source package, with a variable number of bodies and a random seed of 12345. The results is shown in figure 12, which displays that remote swap outperforms swap to disk by a factor of 6 and is a factor of 4 slower than native execution in real memory, when using 4194304 bodies and a page blocks size of 8, which is the optimal page block size (figure 13) for this application. It should be noted that the difference between swapping to disk and swapping to a remote memory location increases as the number of bodies grows and that the half-full artifact is present in this experiment.

VI. Experiment summary

The experiments show that swapping to a remote memory server outperforms swapping to disk on the Playstation 3 platform, both in the tests that are designed specially towards the framework, as well as in the generic scientific applications, which have not been modified in order to work with the framework. The speedup varies from 6 to 21 in the tested scientific applications and the slowdown varies from 4 to 11 compared to native executions. We have also shown that Linux disk swap has serious issues when it comes to scientific applications. The conservative strategy of removing pages from the swap device, whenever it becomes half-full, is poorly suited

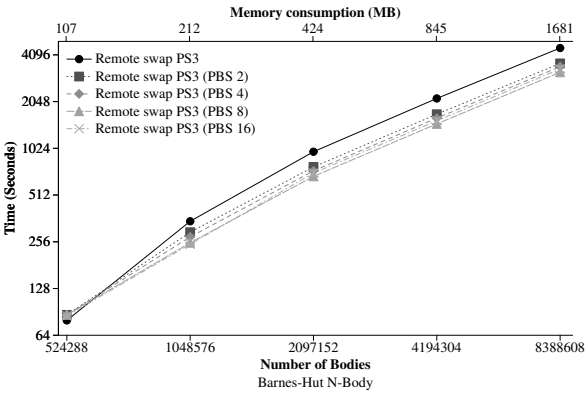


Fig. 13. Barnes-Hut performance with increasing page block sizes

for scientific applications, due to their nature of initializing data and performing several iterations on the same data set, which means that the same pages are accessed several times without any renewal. Furthermore devices that are dedicated to scientific applications should favor the scientific application over other processes running at the system regarding CPU and memory.

VII. Conclusion

In this work we have presented a method for providing remote swap to global Grid infrastructures. Opposed to previous presented models, we present a fully transparent user-level library, which can be submitted along with the Grid jobs eliminating the need to modify neither the OS of the executing Grid resource nor the Grid application to execute. Furthermore the user-level approach makes it possible to throttle the real memory usage of the running job, through the Grid middleware, and thereby increase the pool of resources capable of fulfilling the memory requirements of a given job. Last but not least the user-level approach ensures that only pages that are used by the Grid application are subject for eviction. The disadvantages of using the transparent user-level approach is the time overhead of passing signals, page mappings and page protections between kernel- and user-level, as well as the space overhead of keeping a local process page table within the framework as one can't access the page data-structures of the kernel from user-level.

Experiments show that the framework is operational and despite the introduced overhead still outperforms swapping to disk by a factor of 6 in the worst case. In the best case the framework outperforms swapping to disk with a factor of up to 21.

References

[1] Rasmus Andersen and Brian Vinter. Harvesting idle windows cpu cycles for grid computing. In Hamid R. Arabnia, editor, *GCA*, pages 121–126. CSREA Press, 2006.

[2] Rasmus Andersen and Brian Vinter. Transparent remote file access in the minimum intrusion grid. In *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 311–318, Washington, DC, USA, 2005. IEEE Computer Society.

[3] David P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[4] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[5] Eric A. Anderson and Jeanna M. Neefe. An exploration of network ram. Technical report, Berkeley, CA, USA, 1994.

[6] Josh Barnes and Piet Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324(6096):446–449, December 1986.

[7] Joshua Edward Barnes. ftp://ftp.ifa.hawaii.edu/pub/barnes/treecode.

[8] Adam L. Beberg, Daniel L. Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S. Pande. Folding@home: Lessons from eight years of volunteer distributed computing. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[9] Rui Chu, Nong Xiao, Yongzhen Zhuang, Yunhao Liu, and Xicheng Lu. A distributed paging ram grid system for wide-area memory sharing. In *IPDPS*. IEEE, 2006.

[10] Douglas Comer and James Griffioen. A new design for distributed systems: The remote memory model. In USENIX, editor, *Proceedings of the Summer 1990 USENIX Conference: June 11–15, 1990, Anaheim, California, USA*, pages 127–136, Berkeley, CA, USA, Summer 1990. USENIX.

[11] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on 'Program Generation, Optimization, and Platform Adaptation".

[12] Henrik Hoey Karlsen and Brian Vinter. Minimum intrusion grid - the simple model. In *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 305–310, Washington, DC, USA, 2005. IEEE Computer Society.

[13] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[14] Evangelos P. Markatos and George Dramitinos. Implementation of a reliable remote memory pager. In *USENIX Annual Technical Conference*, pages 177–190, 1996.

[15] Richard Tran Mills, Chuan Yue, Andreas Stathopoulos, and Dimitrios S. Nikolopoulos. Runtime and programming support for memory adaptation in scientific applications via local disk and remote memory. *J. Grid Comput.*, 5(2):213–234, 2007.

[16] D S Myers, A L Bazinet, and M P Cummings. Expanding the reach of grid computing: Combining globus- and boinc-based systems. *Journal of Parallel and Distributed Computing*, 2004.

[17] Martin Rehr and Brian Vinter. The ps3 grid-resource model. In Hamid R. Arabnia, editor, *GCA*, pages 90–95. CSREA Press, 2008.

[18] Open source lattice Boltzmann code. http://www.openlb.org.

Appendix E

Publication 5

Recent Developments in Grid Technology and Applications. G.A. Gravvanis, J.P. Morrison, H.R. Arabnia and D.A. Power, Editors, Chapter 5, ISBN: 978-1-60692-768-7
Rasmus Andersen, Martin Rehr, Brian Vinter: Cycle-Scavenging in Grid Computing

Chapter 1

CYCLE-SCAVENGING IN GRID COMPUTING

Rasmus Andersen, Martin Rehr, and Brian Vinter*
University of Copenhagen, Department of Computer Science

PACS 05.45-a, 52.35.Mw, 96.50.Fm. **Keywords:** .

Key Words: cycle scavenging, Grid Computing, Public Resource Computing, Minimum intrusion Grid, Virtual Machines **A**MS Subject Classification: 53D, 37C, 65P.

Abstract

Grid Computing and Public Resource Computing systems each provide means of obtaining and utilizing distributed computational resources. In this chapter we explore the benefits and potential of combining the two fields into a system that offers the flexibility of Grid Computing and the resource richness of the cycle-scavenging PRC systems.

Important aspects of a combined effort include host system security, application security, platform independence, and resource management and control.

Sandboxing technology accommodates these requirements, and two different models offered by the Minimum intrusion Grid are presented. Both of them use virtual machines to sandbox Grid Computing jobs on idle personal desktop computers and have proved to close the gap between Grid Computing and Public Resource Computing.

*E-mail address: vinter@diku.dk

1 Introduction

Cycle scavenging, or Screen Saver Science, is an increasingly popular computing paradigm used within many fields of science that seek to tap the enormous amount of unused processing power from the millions of computers connected to the Internet. The paradigm is best known from the many successful Public Resource Computing, PRC, projects, such as SETI@Home, where the idle time cycles are used for a dedicated scientific application.

Grid Computing [6] and PRC each provide increasingly interesting means of obtaining computational resources. Grid Computing is mostly used for connecting university supercomputers, while PRC is predominantly used by research projects for harvesting PC-based idle CPU cycles for a small number of research projects. However, even though the two fields appear closely related, little effort has been made to combine them into a system that offers the flexibility of Grid computing with the resource richness of the PRC model.

One of the Grid promises is to make it possible to share and effectively use distributed resources on an unprecedented scale. Specifically, this includes harnessing the unused capacity of idle desktop PCs. Harnessing 'free' cycles through PRC is of great interest since a modern PC is powerful and highly underutilized, and as such cycle harvesting provides huge calculation potential if one combines millions of them in a computing Grid. A lot of research has been done to Grid-enable idle resources, yet no widely accepted system to effectively scavenge idle cycles, in particular idle Windows cycles, has been found. Most known Grid systems such as ARC which is based on the Globus toolkit and Condor [3] are unsuitable for PRC computing, as they work under the underlying assumption that the resources are available indefinitely, while PRC resources are transient by nature.

Ensuring the safety of a donated resource while it executes a Grid job in a PRC context is an all important topic since all free resources will vanish if the model proves harmful to the hosts. Contrary to standard PRC tasks, a Grid job may take any form and include the execution of any binary. Therefore it is necessary to take precautions to ensure that the execution of Grid jobs cannot harm donated resources neither intentionally nor by accident.

Extending the PRC concept to actual Grid Computing, security and installation of software on the host resource are equally important issues to a successful solution. All PRC projects known to the authors require the donor to install software on the resource that should contribute, which alone eliminates users from donating resources from computers that they do not have administrative rights on. The software installation also opens up possible exploits and requires the donor to perform updates on that software. This is not desirable and may reduce the amount of donated resources.

Our goal is to design Grid PRC sandbox models that with a minimal effort from the resource owner can execute Grid jobs in a secure environment. This chapter addresses the problems that need to be solved in order to combine PRC and Grid Computing by scavenging idle desktops for general scientific use. First of all, under the restrictions mentioned above, a method to gain access to the CPU cycles on the idle resource must be found. In this aspect, security is a major issue. Ideally, a resource owner should neither install any software nor execute any foreign applications that, intentionally or not, could compromise his system. Secondly, the resource, possibly hidden by a Network Address Translation router and a firewall, must be attached to the Grid without forcing modification to routers or firewalls. Thirdly, it must be ensured that a given resource has installed the correct soft-

ware base that a given Grid job requires. Finally, we introduce extra features to improve the model; for instance, a method to predict the idle time period of a resource in advance. Using this method, a job with a time deadline is submitted to a resource that is predicted to be available in the specified time frame.

The approach taken here is to use virtual machines to provide a sandbox that by default includes everything needed to execute Grid jobs and completely separates them from the resource host system. This separation ensures that, on the one hand, a grid job cannot compromise the host system, and on the other hand, the grid job is protected from other users of the system. Two different models have been implemented: Firstly the The MiG-SSS, which is based on system virtual machines that execute an entire linux guest operating system specifically designed as a Grid resource. Secondly, the One-Click MiG Applet Resource, which is based on Java applet technology and enables users with a standard Java-enabled web browser to donate their idle time PCs with a single click on a website, thereby starting a Java applet capable of executing Grid jobs written for this framework.

1.1 Related Work

BOINC [2] is a software platform that allows many different distributed computing projects to utilize idle volunteered computer resources. Many Public Resource Computing systems use BOINC, and research groups can with little effort create new projects. A project involves a set of applications that will be run in a BOINC client on a user's resource. As such, BOINC could be used for this project by running the proposed virtual machine as the project application.

A few projects have been found to combine Screen Saver Science with Grid computing, for instance the Entropia Virtual Machine [4], which is a commercial product, and the survey [5] that is merely an extensive introduction to the approach of using virtual machines for Grid computing.

2 Sandboxing in a Grid Context

Although virtualization was introduced several decades ago, the concept is now more popular than ever and has revived in a multitude of computer system aspects that benefit from properties such as platform independence and increased security. One of those applications is Grid computing, where the ultimate goal of combining and utilizing distributed, heterogeneous resources as one big virtual supercomputer necessitates these properties. Regarding utilization of the public's computer resources for grid computing, virtualization, in the sense of virtual machines, is a necessity for fully leveraging the true potential of Grid computing. Without virtual machines, experience shows that people are, with good reason, reluctant to put their resources on a grid since they have to not only install and manage a software code base, but also allow native execution of unknown and untrusted programs. All these issues can be eliminated by introducing virtual machines.

eScience, modelling scientific problems using computers, has driven the development of Grid technology, and as the simulations get more and more accurate, the amount of data and needed compute power increase equivalently. Many research projects have already made the transition to Grid platforms to accommodate the immense requirements for data

and computational processing. Using this technology, researchers gain access to many networked computers at the cost of a highly heterogeneous computing platform. Obviously, maintaining application versions for each resource type is tedious and troublesome, and results in a deploy-port-redeploy cycle. Further, different hardware and software setups on computational resources complicate the application development drastically. One never knows to which resource a job is submitted in a grid, and while it is possible to assist each job with a detailed list of hardware and software requirements, researchers are better left off with a virtual workspace environment that abstracts a real execution environment.

Hence, a virtual execution environment spanning the heterogeneous resource platform is essential in order to fully leverage the grid potential. From the view of applications, this would render a resource access uniform and thus allow the much easier ”compile once run anywhere” strategy; researchers can write their applications, compile them for the virtual machine and have them executed anywhere in the Grid.

Due to the renewed popularity of virtualization over the last few years, virtual machines are being developed for numerous purposes and therefore exist in many designs, each of them in many variants with individual characteristics. Despite the variety of designs, the underlying technology encompasses a number of properties beneficial for Grid Computing:

Platform Independence: In a grid context, where it is intrinsic to move around application code as freely as application data, it is highly profitable to enable applications to be executed anywhere in the grid. Virtual machines bridge the architectural boundaries of computational elements in a grid by raising the level of abstraction of a computer system, thus providing a uniform way for applications to interact with the system. Given a common virtual workspace environment, grid users are provided with a compile-once-run-anywhere solution.

Furthermore, a running virtual machine is not tied to a specific physical resource; it can be suspended, migrated to another resource and resumed from where it was suspended.

Host Security: To fully leverage the computational power of a grid platform, security is just as important as application portability. Today, most grid systems enforce security by means of user and resource authentication, a secure communication channel between them, and authorization in various forms. However, once access and authorization is granted, securing the host system from the application is left to the operating system.

Ideally, rather than handling the problems after system damage has occurred, harmful - intentional or not - grid applications should not be able to compromise a grid resource in the first place.

Virtual machines provide stronger security mechanisms than conventional operating systems, in that a malicious process running in an instance of a virtual machine is only capable of destroying the environment in which it runs, i.e. the virtual machine.

Application Security: Conversely to disallowing host system damage, other processes, local or running in other virtualized environments, should not be able to compromise the integrity of the processes in the virtual machine.

System resources, for instance the CPU and memory, of a virtual machine are always mapped to underlying physical resources by the virtualization software. The real resources

are then multiplexed between any number of virtualized systems, giving the impression to each of the systems that they have exclusive access to a dedicated physical resource. Thus, grid jobs running in a virtual machine are isolated from other simultaneous grid jobs running in other virtual machines on the same host as well as possible local users of the resources.

Resource Management and Control: Virtual machines enable increased flexibility for resource management and control in terms of resource usage and site administration. First of all, the middleware code necessary for interacting with the Grid can be incorporated in the virtual machine, thus relieving the resource owner from installing and managing the grid software. Secondly, usage of physical resources like memory, disk, and CPU usage of a process is easily controlled with a virtual machine.

Performance: As a virtual machine architecture interposes a software layer between the traditional hardware and software layers, in which a possibly different instruction set is implemented and translated to the underlying native instruction set, performance is typically lost during the translation phase. Despite of recent advances in new virtualization and translation techniques, and the introduction of hardware-assisted capabilities, virtual machines usually introduce performance overhead and the goal remains achieving near-native performance only. The impact depends on system characteristics and the applications intended to run in the machine.

To summarize, virtual machines are an appealing technology when combining Grid Computing and PRC because they solve the conflict between the grid users at the one end of the system and resource providers at the other end. Grid users want exclusive access to as many resources as possible, as much control as possible, secure execution of their applications, and they want to use certain software and hardware setups. At the other end, introducing virtual machines on resources enables resource owners to service several users at once, to isolate each application execution from other users of the system and from the host system itself, to provide a uniform execution environment, and managed code is easily incorporated in the virtual machine.

3 Enabling the sandboxes for the Grid

The main problem with scavenging personal computers is that the vast majority are hidden behind a NAT router, i.e. they do not have global IP address and are therefore not reachable from the Internet. Hence, to enable the sandbox for Grid Computing, care must be taken to circumvent the missing inbound Internet access. Naturally, this issue is highly dependent on the Grid middleware in question. In this work, the sandboxes are designed for the Minimum intrusion Grid, MiG, which is presented next, before the details of how to tailor the two sandboxes for MiG are explained.

3.1 Minimum intrusion Grid

MiG [8] [7] is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely new platform for Grid computing. The philosophy behind the MiG is

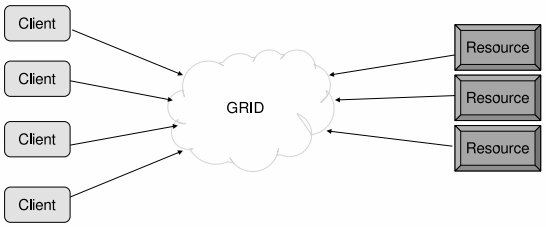


Figure 1: The abstract MiG model

to provide a Grid infrastructure that imposes as few requirements on users and resources as possible.

The idea is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A resource only needs to create an MiG user on the system and to support inbound ssh and outbound HTTPS. Initially, the resource must register to the MiG system using a certificate.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 1, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system that, although it appears as centralized system, in reality is a distributed system itself.

The basic functionality in MiG starts with users submitting jobs to MiG and resources sending requests for jobs. A resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can inform the user of the job completion. Thus, MiG provides full anonymity; users and resources interact only with MiG, never with each other.

3.1.1 Scheduling

The centralized black box design of MiG makes it capable of strong scheduling, which implies full control of the jobs being executed and the resource executing them. Each job has an upper execution time limit, and when the execution time exceeds this time limit the job is rescheduled to another resource. This makes the MiG system very well suited to host PRC resources, as they by nature are very dynamic and frequently join and leave the Grid without notifying the Grid middleware.

3.2 Cycle-Scavenging using System Virtual Machines

In MiG-SSS, the screen saver model based on system virtual machine, the basic idea is to let resource owners install a sandbox to provide a secure execution environment in which the Grid job is completely isolated from the host machine and vice versa. Such a sandbox can take the shape of a virtual machine, which is exactly the approach that we have taken in this work. Two techniques can be used to provide a virtual machine: Emulation and Virtualization[5]. Emulation provides the functionality of the target processor completely in software, which makes it a very secure approach. Also, the ability to emulate one processor type on any other processor type makes it ideal for this scenario. However, the method of interpreting the entire guest operating system, rather than running it on the native hardware, results in a significant performance drawback. When emulating a PC architecture on a PC, a compatibility layer that enables the target code to be run directly on the host processor, can reduce the performance penalty. On the other hand, virtualization partitions hardware in multiple contexts, thus enabling running multiple operating systems on the same hardware resources simultaneously. Several virtualization approaches exist[6]:

- Bare-metal Architecture
- Para-virtualization
- Full Virtualization, also known as Transparent Virtualization, or Hosted Architecture

The Bare-metal Architecture approach runs the guest operating system in Ring 0, the most privileged protection level in x86 architectures. Running multiple operating systems in the same protection level could potentially result in one of the systems compromising the other. Clearly, this approach is not acceptable for resource owners. The Para-virtualization approach needs to modify the host system by interposing a hypervisor between the operating system and the hardware. The hypervisor then takes on the Ring 0 and the operating system must be explicitly ported to run in Ring 1. These modifications to the host operating system exclude this approach. The Full Virtualization approach has performance drawbacks, but is the most secure and thus chosen. As shown in Figure 2, the virtual machine allows a guest operating system to run as an application in the host operating system. The virtual machine emulates the underlying hardware, thus creating a secure sandbox that allows an application written for one operating system, e.g. Linux, to be executed in another, e.g. Windows.

The performance penalties are mitigated by kernel support that enables it to run most of the target application code directly on the host processor, thus achieving near native speed. Regarding security, the virtual machine is a user space process that cannot do any harm to the host system as long as the permanent storage is protected properly. If the virtual machine is destroyed by a malicious application, the host system is not affected, and the virtual machine can start afresh. The following such solutions exist for the Windows platform:

- VirtualPC
- VMWare
- VirtualBox

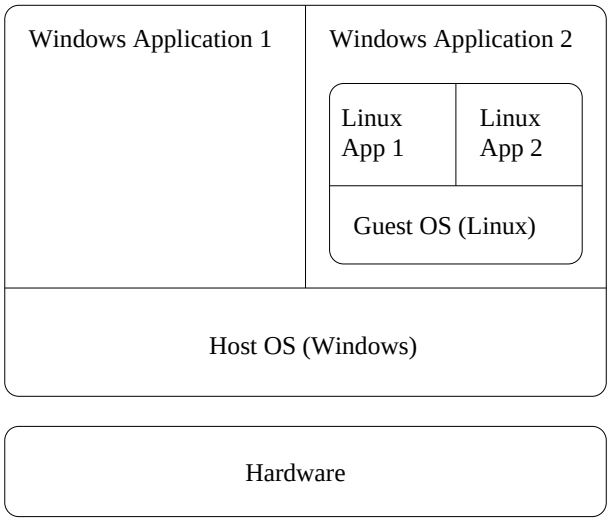


Figure 2: Full Virtualization

- Qemu + Qemu Accelerator Module

Having downloaded and installed one of the virtual machines, a resource owner only needs to install a screen saver that starts the virtual machine upon activation and a tailor-made Linux image that is capable of running the Grid resource software automatically. In this manner, when the resource goes into screen saver mode, the virtual machine is activated and the Linux guest operating system is booted. The details of how to Grid-enable the hosted Linux system are explained next.

3.2.1 The MiG Linux Guest OS

As explained above, all that is required for a resource to join MiG, is to create a grid user account and support for incoming SSH and outgoing HTTPS. So basically, the MiG Linux image can be built using any Linux distribution that runs an x86 system. Since the virtual machine provides a standardized virtualized set of hardware, compatibility amongst the wide range of different hardware setups on the resources will not be an issue. The main concerns with respect to the distribution is the size and the start-up time. Both issues matter only for practical reasons, the size should be minimized to avoid an excessively large download, and, naturally, the start-up time should be minimized as much as possible. In order to circumvent the missing inbound Internet access on resources that use NAT, it must be ensured that all communication is initiated by the resource. In MiG, this was easily integrated by small changes that only apply for sandboxes. In addition, it allows for directing jobs that users point out as Public Resource Computing jobs directly to a free sandbox. The generic MiG Linux Image consists of a kernel and a Ram-disk that altogether take up less than 3 MB. An online generator modifies the generic image by giving it a unique resource name and a session id needed for requesting a job. Further, the resource owner can choose the size of a hard disk image file to provide as storage for the sandbox. Thus, certified resource owners can have a complete image built with a unique key allowing the sandbox

to automatically request and execute Grid jobs. In the standard MiG model, the identity of a resource requesting a job is verified by keeping the public SSH key of the resource in the MiG system and copying all job files to the resource over SSH. The sandbox model however, is modified to use a pull model on the resource where all files are transferred using HTTPS. Hence, a firewall in front of a resource only needs to be open for HTTP and HTTPS to allow the resource to run Grid jobs.

3.2.2 Runtime Environments

Once the basic sandbox is in place, it is possible to execute user applications within the virtual Linux machine. Many applications can be passed as executables from the Grid job and these need no further components to execute. Other, commonly used, applications may benefit from a preinstalled runtime environment, such as they are found on ordinary Grid resources.

Installing runtime environments in the sandboxed environment could be done as on a conventional resource, which however would require the PC owners to personally maintain the sandboxed Linux distribution; this model is obviously not desirable. Alternatively the sandbox image could be distributed with the initial Linux image, but this would greatly increase the size of the distribution image and in addition be a very static model.

The chosen solution allows individual research groups to maintain runtime environments for the sandboxed resources and at the same time allows the individual PC owners to control which runtime environments are downloaded and at which time. The runtime environments are kept in individual virtual disk-partitions, in the form of a single file. The PC owner can download individual runtime environments from the VGrids, MiG's notion of a virtual organization, that maintain the runtime environments and when a job that uses a runtime environment is received by the sandboxed resource, the virtual Linux machine will mount the file system that contains the runtime environment. This way each runtime environment is kept isolated from the rest of the system, and can easily be built and maintained by the research groups that need them to be available for their executions.

3.3 Cycle-Scavenging using Java Applets

As explained above, all that is required for a PRC resource to join MiG is a sandbox and support for outgoing HTTPS. The previous solution presented above requires installation of non standard software to activate and execute the sandbox. In this model, "MiG One-Click"[1] the work imposed on the resource donor is taken to the extreme: No software install is needed.

To reach our stated goal of no Grid specific software installation and no modification of the donated machines firewall settings, we are forced to use software which is an integrated part of a common Internet connected resource.

We found that amongst the most common software packages for any PC type platform there is a Java enabled web browser. The web browser provides a common way of securely

[1]The URL accessed to activate the web browser as a sandboxed MiG Java resource is called "MiG One-Click", as it requires one click to activate it.

communicating with the Internet, which is allowed by almost all firewall configurations of the resources we target.[2]

The web browser provides us with a communication protocol, but it does not by itself provide a safe execution environment, however all of the most common graphics enabled web browsers have support for Java applets that are capable of executing Java byte-code located on a remote server.

The Java applet security model, ASM, prevents the Java byte-code executed in the applet from harming the host machine and thereby provides the desired sandbox effect for us to trust the execution of unknown binaries on donated resources.

The choice of web browsers and Java applets as the execution framework results in some restrictions on the type of jobs that may be executed in this environment:

- Applications must be written in Java
- Applications must apply to ASM
- The total memory usage is limited to 64 MB including the Grid framework
- Special methods must be used to catch output
- Special methods must be used for file access

By accepting the limitations described above, a web browser can be turned into a Grid resource simply by entering a specific URL. This triggers the load and execution of an applet which acts as our Grid gateway and enables retrieving and executing a Java byte-code based Grid job. The details of this process are described next.

3.3.1 The Applet Grid Resource

Several changes to the Grid middleware are needed to allow Java applets to act as Grid resources. First of all the Grid middleware must support resources which can only be accessed through a pull based model, which means that all communication is initiated by the resource, i.e. the applet. This is required because the ASM rules prevents the applet from initiating listening sockets, and to meet our requirement of functioning behind a firewall with no Grid specific port modifications. Secondly, the Grid middleware needs a scheduling model where resources are able to request specific type of jobs, e.g. a resource can specify that only jobs which are tagged to comply to the ASM can be executed.

The Java applet technology makes it is possible to turn a web browser into a MiG sandbox without installing any additional software. This is done automatically when the user accesses the MiG One-Click web page, which loads an applet into the web browser. This applet functions as a Grid resource script and is responsible for requesting pending jobs, retrieving and executing granted jobs, and delivering the results of the executed jobs to the MiG server.

To make the applet work as a resource script, several issues must be addressed. First of all ASM disallows local disk access. Because of this both executables and input/output files

[2] Resources located behind firewalls that do not support outgoing HTTPS is considered out of range for this PRC, however it is not unseen that outbound HTTPS is blocked.

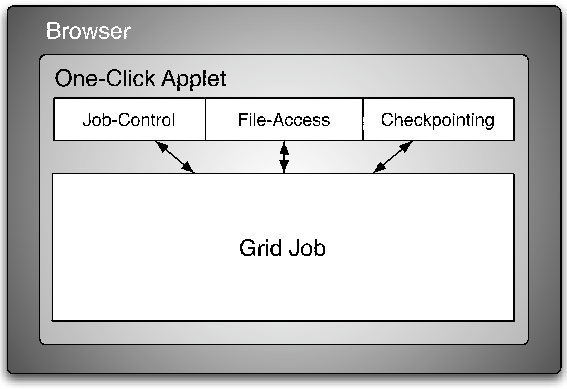


Figure 3: The structure of a One-Click job

must be accessed directly at the Grid storage. Secondly only executables that are located at the same server as the initial applet are permitted to be loaded dynamically. Thirdly text output of the applet is written to the web browser's Java console and not accessible by the Grid middleware.

When the applet is granted a job by the MiG server, it retrieves a specification of the job which specifies executables and input/output files. The applet then loads the executable from the Grid, this is made possible by the MiG server which sets up an URL from the same site as the resource applet was originally loaded which points to the location of the executables. This allows unknown executables to be loaded and comply with the ASM restrictions on loading executables. Figure 3 shows the structure of a One-Click job.

Executable jobs that are targeted for the MiG One-Click model must comply with a special MiG One-Click framework, which defines special methods for writing `stdout` and `stderr` of the application to the MiG system[3]. Normally the stdout and stderr of the executing job is piped to a file in the MiG system, but a Java applet, by default, writes the stdout and stderr to the web browsers Java console. We have not been able to intercept this native output path. Input and output files that are specified in the job description must be accessed directly at the Grid storage unit since the ASM rules prohibits local file access. To address this issue the MiG One-Click framework provides file access methods that transparently provide remote access to the needed files. Note that the MiG system requires input files and executables to be uploaded to the MiG server before job submission which ensures that the files are available at the Grid storage unit.

In addition to the browser applet a Java console version of the MiG resource has been developed, to enable the possibility of retrieving and executing MiG One-Click jobs as a background process. This requires only a Java virtual machine. To obtain the desired security model, a customized Java security policy is used, which provides the same restrictions as the ASM.

[3]The result of a MiG job is the stdout/stderr and the return code of the application that is executed.

3.3.2 Checkpointing

PRC resources will join and leave the Grid dynamically, which means that jobs with large running time have a high probability of being terminated before they finish their execution. To avoid wasting already spent CPU-cycles a checkpointing mechanism is build into the applet framework. Two types of checkpointing have been considered for inclusion, transparent checkpointing and semi-transparent checkpointing.

Transparent Checkpointing All to the authors known transparent checkpoint mechanisms provided to work with Java, require the JVM to be replacement or access to the /proc file system on Linux/Unix operating system variants, as the default JVM does not support storing program counter and stack frame. Since our goal is to use a web browser with the Java applet as a Grid resource neither of those solutions are satisfactory, since both the replacement of the JVM and access to the /proc file system violates the Java applet security model. Furthermore most PRC resource will be running the Windows operating system which do not support the /proc file system.

Semi-transparent Checkpointing Since transparent checkpointing is not applicable to the One-Click model, we went on to investigate what we call semi-transparent checkpointing. Semi-transparent checkpointing covers that the One-Click framework provides a checkpoint method for doing the actual checkpoint, but the application programmer is still responsible for calling the checkpoint method when the application is in a checkpoint safe state.

The checkpoint method stores the running Java object on the MiG server through HTTPS. Since it can only store the object state, and not stack information and program counters, the programmer is responsible for calling the checkpoint method at a point in the application, where the current state of the execution may be restored from the object state only. To restart a previously checkpointed job, the resource applet framework first discovers that a checkpoint exists and then loads the stored object.

3.4 MiG Features

To improve and simplify the sandboxes further, MiG contains two components that apply: Strong scheduling and remote file access.

3.4.1 Scheduling

Contrary to the majority of the existing Grid middlewares, where several levels of scheduling results in jobs being submitted to a resource where another level of scheduling takes place, MiG makes the scheduling for fairness much simpler as the local scheduling comes before the Grid scheduling. Thus, a single job is never left waiting a long time for CPU cycles once it has been submitted to a resource.

Existing Screen Saver Science systems all target problems that have many, usually millions, of independent tasks that often run for tens or hundreds of hours. Once a task has been assigned to a computer it will be processed while the computer is in screen saver mode. Processing is suspended if the screensaver is suspended and similarly resumed again along

with the screensaver. An artefact of this model is that one never knows when the result of a given task is ready, and it is very hard to determine if the task has been lost or if it is simply only allowed to proceed very slowly.

For applications such as computational chemistry this model is very poorly suited. The number of tasks is usually in the tens or hundreds and it is often the case that analyses of the results can only start once the result of every task is in.

This is easily addressed by putting an upper time limit on each job, and if the time limit is exceeded, the job is resubmitted to another resource. However, in order to schedule a job with a deadline to a screen saver resource, we need to know how long the resource is available, i.e. how much time it takes before the screen saver is deactivated. To predict the available time slot of a screen saver resource, we use exponential average on an hourly basis, which has proved to converge against the actual resource idle time quite fast.

3.4.2 Remote File Access

One difficulty that users report when using Grid is file access, since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element and result files must also be downloaded explicitly. The MiG model introduces home catalogs for all Grid users, and all file references are relative to this home catalog. This eliminates all naming problems, since MiG provides one simple access entry to a user's home catalog. Furthermore, using the MiG Remote File Access library [1], applications running on a Linux resource - as the ones used in the MiG-SSS model - can, transparently and without recompiling or relinking applications, access application input and output files remotely, thus only downloading needed data and only uploading modified data.

The same ideas have been implemented in the One-Click Model which also provides transparent remote file access to the jobs that are executed. The MiG storage server supports partial reads and writes, through HTTPS, of any file that is associated with a job. When the resource applet accesses files that are associated with a job, a local buffer is used to store the parts of the file that are being accessed. If a file position which points outside the local buffer is accessed, the MiG server is contacted through HTTPS, and the buffer is written to the MiG server if the file is opened in write mode. The next block of data is then fetched from the server and stored into the buffer and finally the operation returns to the user application. The size of the buffer is dynamically adjusted to utilize the previously observed bandwidth optimally.

3.4.3 Block size estimation

To achieve the optimal bandwidth for remote file access it is necessary to find the optimal block size for transfers to and from the server. In this case the optimal block size is a trade off between latency and bandwidth. We want to transfer as large a block as possible without excessive latency penalty since the chance of transferring data that will not be used increases with the block size.

We define the optimal block size bs_{opt} as the largest block where a doubling of the block size does not double the time to transfer it. This can be expressed the following way:

$$t(x) * 2 > t(x * 2) \qquad \forall x < bs_{opt} \tag{1}$$

$$t(x) * 2 < t(x * 2) \qquad \forall x > bs_{opt} \tag{2}$$

$$t(x) \ = time\, to\, transfer\, block\, of\, size\, x$$

We do not want block sizes below bs_{opt} as the time t used to transfer a block of size x is less than doubled when the block size is doubled. On the other hand we don't want 'too large' block sizes as we do not know if the retrieved data is going to be used or discarded due to a seek operation beyond the end of the local buffer.

As the One-Click resources can be placed at any sort of connection, and the bandwidth of the connection thus may differ greatly from one resource to another, it is not possible to use a fixed block size and reach a good ratio between bandwidth and latency at an arbitrary type of connection.

The simplest approach would be to use a fixed bs_{opt} based on empirical tests on the most common connections.

A less trivial, but still simple, approach would be to measure the time it takes to connect to the server and then choose a block size which ensures the transfer time of that block to be a factor of x larger than the time to connect, to make sure that the connection overhead does not exceed the time of the actual data transfer.

The chosen approach is to estimate bs_{opt} from the time spent transferring block $x-1$ with the time of transferring block x, starting with an initial small[4] block size bs_0 and then doubling the block size until a predefined cutoff ratio CR is reached. After each data transfer the bandwidth bw_x is calculated and compared to the bandwidth of the previous transfer bw_{x-1}. If the ratio is larger than the predefined CR:

$$\frac{bw_x}{bw_{x-1}} > CR \tag{3}$$

then the block size is doubled:

$$bs_{x+1} = bs_x * 2 \tag{4}$$

As the block size is doubled in each step the theoretical CR to achieve bs_{opt} should be 2, since there is no incentive to increase block size once the latency grows linearly with the size of the data that is transferred.

However in reality, one need to get a CR below 2 to achieve bs_{opt}. This is due to the fact that all used block sizes are powers of 2, and one cannot rely on the optimal block size to match a power of 2.

Therefore to make sure to get a block size above bs_{opt} you need a lower CR. Empirical tests showed that a CR about 1.65 yields good results, see section 4.2

Additional extensions include adapting to the frequency of random seeks in the estimation of the CR. A large amount of random seeks to data placed outside the range of the current buffer will cause new blocks to be retrieved in each seek. Therefore the block size should be lowered in those cases to minimize the latency of each seek.

[4] An initial small block size gives a good result as many file accesses applies to small text files such as configuration files.

4 Experiments

To test the One-Click model we established a controlled test scenario. Eight identical Pentium 4, 2.4 GHz machines with 512 MB ram were used for tests.

4.1 One-Click as concept

The test application used is an exhaustive algorithm for folding proteins written in Java. This was changed to comply with the applet framework.

A protein sequence of length 26 was folded on one machine, which resulted in a total execution time of 2 hours, 45 minutes and 33 seconds. The search space of the protein was then divided into 50 different subspaces using standard divide and conqueror techniques. The 50 different search spaces were submitted as jobs to the Grid, which provides an average of 6 jobs per execution machine and 2 extra jobs to prevent balanced execution. The search spaces on their own also provide unbalanced execution as the valid protein configurations vary from one search space to another and thus results in unbalanced execution times. The experiment was made without checkpointing the application. The execution of the 50 jobs completed in 29 minutes and 8 seconds, a speedup of 5.7 for 8 machines. While this result would be considered bad in a cluster context it is quite useful in a Grid environment.

To test the total overhead of the model, a set of 1000 empty jobs was submitted to the Grid with only one One-Click execution resource connected. The 1000 jobs completed in 19935 seconds, which translates to an overhead of approximately 20 seconds per job.

4.2 File access

To achieve the best bandwidth cutoff ratio CR several experiments has been made. In the experiments a 16 MB file was read 100 times by the One-Click resource on a 20 Mb/s broadband Internet connection. All experiments start with an initial block size of 2048 (2^{11}) bytes. The first experiment was run with a CR of 0, which means that the block size is doubled in every transfer. The result is shown in figure 4.

The figure shows how the latency starts to raise dramatically between block size 2^{18} and 2^{20} and the bandwidth to latency ratio starts to fall at those block sizes. The bandwidth to latency ratio between block size 2^{18} and 2^{20} lies in the interval from 1.25 to 1.75. Based on these observations we performed the same test with a CR of 1.5. The result is shown in figure 5.

This shows that a CR of 1.5 is too low as block sizes of 2^{21} occur and we want the block sizes to be between 2^{18} and 2^{20} to limit the maximum latency. Therefore the CR must be between 1.5 and 1.75. The test was then run with CR 1.55, 1.60, 1.65, 1.70 and 1.75. The result is shown in figure 6.

We observe that a CR of 1.75 is too high, as only a few block sizes of 2^{19} occur and no block sizes of 2^{20} occurs. A CR of 1.55 results in a few block sizes of 2^{21} which is above the block sizes we want. 1.60 represents the block sizes we want and block size 2^{20} is well represented. A CR of 1.65 represents block size 2^{19} well and a few block sizes of 2^{20} is reached as well, and a CR 1.70 represents block size 2^{20} but no block sizes of 2^{21} are represented. We choose a CR of 1.65 as block size 2^{20} is considered the braking

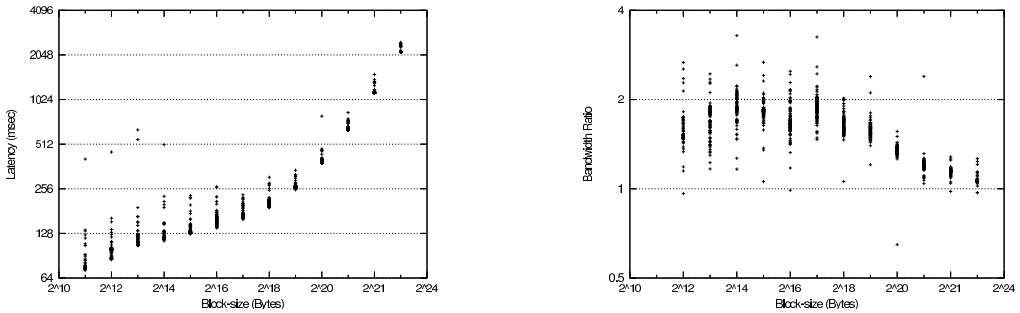


Figure 4: The left figure shows the latency as a function of the block size, the right figure shows the bandwidth ratio $\frac{bw_x}{bw_{x-1}}$ as a function of the block size. Between block size 2^{18} and 2^{20} the latency starts to raise and the bandwidth ratio starts to fall. This is where the cutoff is chosen to avoid excessive raise in latency

point where the latency starts to grow excessively, therefore we do not want it to be to well represented, but we want it to be represented, which is exactly the case at a CR of 1.65.

To verify the previous finding that the CR value should be 1.65 a test application, which traverses a 16 MB file of random 32 bit integers was developed. First the application was tested against the framework, where fixed block sizes were used, and then the application was tested against the framework, where the dynamic block sizes with a CR of 1.65 were used. The results are shown i figure 7.

The experiment shows, as expected, that the execution time decreases as the block sizes increase in the experiments with static block sizes. The execution time in the experiments with the dynamic block sizes all reside around 256 seconds[5] which are satisfactory, as this shows that compared to largest static buffer size of interest[6], the execution time loss using a dynamic buffer size is at most a factor of four. The reader should note that this type of application is the worst case for dynamic buffer sizing as all the data are read sequentially.

[5]With the exception of 3 runs, which are classified as outliers

[6]The largest static buffer of interest is 2^{17} as this is where the time gained by doubling the buffer, levels out.

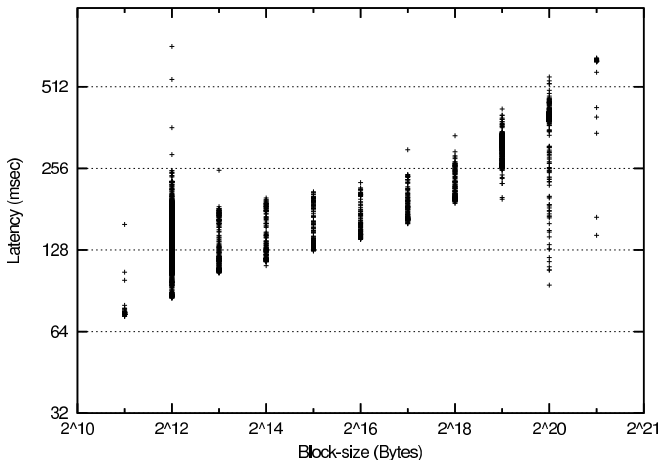


Figure 5: The latency as a function of the block size with CR 1.5

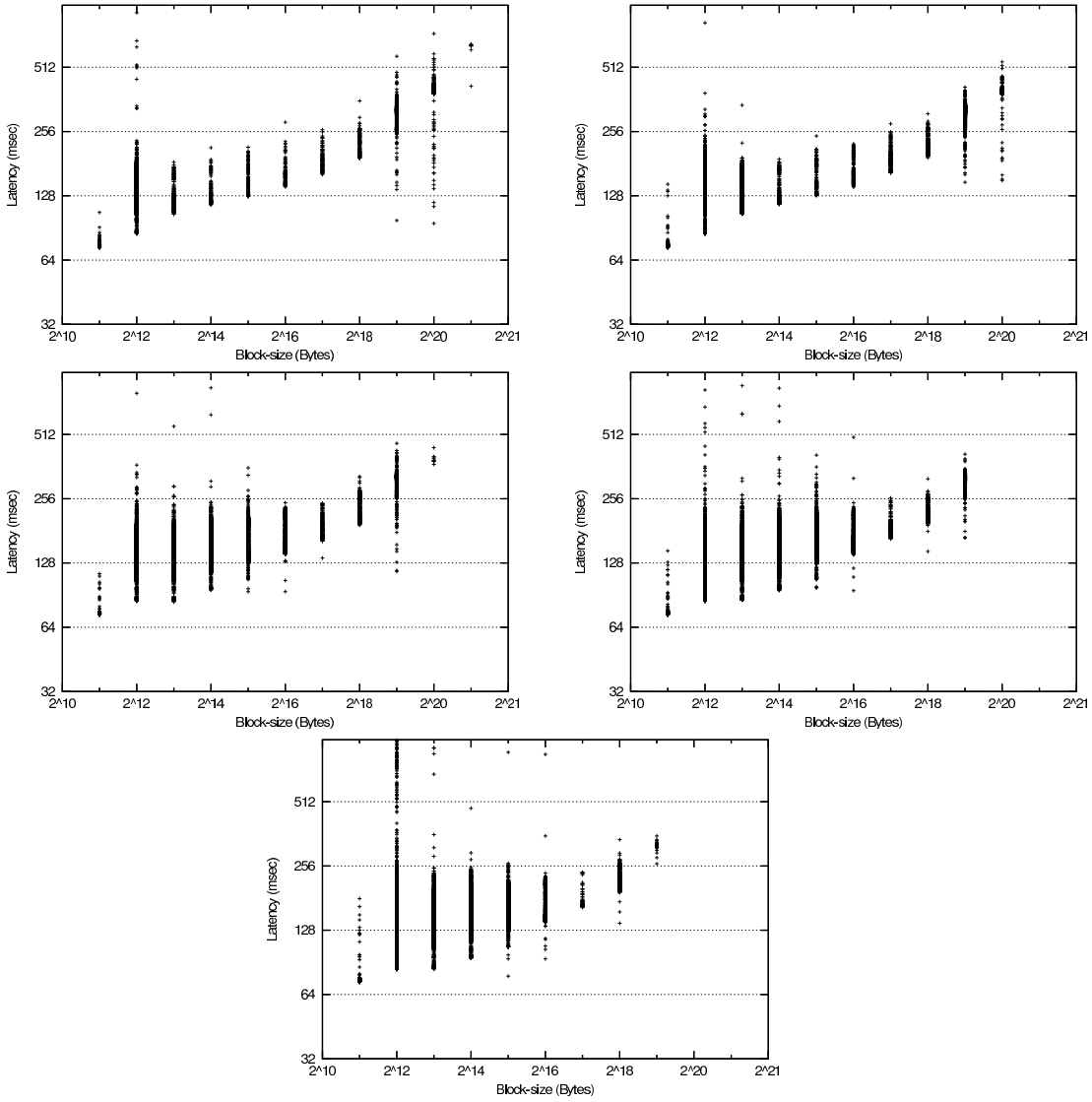


Figure 6: The latency as a function of the block size with CR 1.55, 1.60, 1.65, 1.70, 1.75

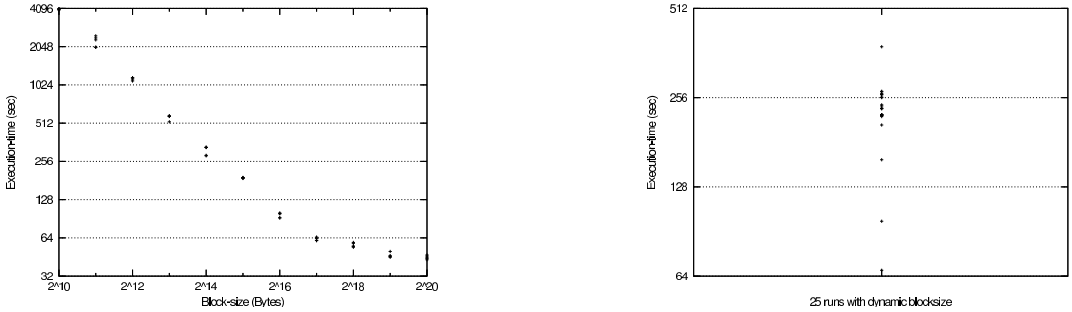


Figure 7: The execution time as a function of the block size and the execution time with dynamic block sizes and a CR of 1.65

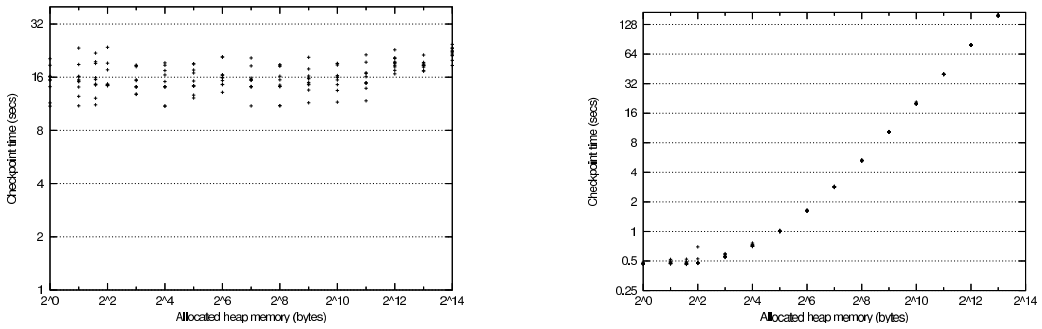


Figure 8: The time spent checkpointing on a 20 Mb/s and a 2048/412 kb/s Broadband Internet

If the integers were read in random order, the dynamic buffer size execution would perform much better.

4.3 Checkpointing

The next obvious performance issue is to test the overhead of performing a checkpoint operation within a process. This was tested by submitting jobs that allocate heap memory in the range from 0 kB to 8192 kB. Each job first allocates X kB, where X is in the order power of 2, and does 10 checkpoints, which saves the entire heap space. The performance was first tested on a 20 Mb/s broadband Internet connection.

The test was then repeated using a more modest 2048/512 kb/s broadband Internet connection. The result of these tests is shown in figure 8.

In the first test, using the 20 Mb/s connection, the checkpoint time is constant as the memory size grows. We can conclude from this, that the overhead of serializing the Java object is dominating compared to the actual network transfer time. The opposite is the case when we examine the results of the 2048/512 kb/s connection. Here we see that the time spent grows linearly with the size of the allocated memory, from which we may conclude that on a 512 kb/s connection the bandwidth is, not surprisingly, the limiting factor.

The experiments also show that the dynamic block sizes approach increases the execution time by of factor of four compared to the execution time reached with the largest static block size in a worst case scenario.

The building checkpointing mechanism has an overhead of 15 seconds per checkpoint on a 2.4GHz P4 and the One-Click framework overall is causing approximately 20 seconds of overhead to each execution, compared to local execution. Despite of this, a considerable speedup is reached in the presented protein experiment.

5 Conclusion

This work has shown how to eliminate the factors that have previously impeded the fusion of Public Resource Computing and Grid Computing to effectively utilize idle CPU cycles from desktop machines for any kind of Grid job.

The prohibiting factors include NAT-hidden resources, means to utilize Windows desktops, the workload required by a non-expert resource owner to install and manage all resource software, and the security issues involved with installing a large software base on the resource.

Using sandboxing technology we have successfully eliminated all of these limitations. Two models exist, the MiG-SSS and the One-Click model. In the MiG-SSS model, resource donors download a bundle consisting of a screen saver, a virtual machine, and a special MiG Linux image in order to share their idle resources. The MiG One-Click model lowers the workload imposed on the resource owner to only requiring a visit on a web page.

The MiG system has proved flexible enough to easily deal with computers behind network address translators, and mobile processes and automatic resubmission of jobs solve the problem with resources that are cut off the network or leave the screen saver mode.

Using the MiG-SSS approach, a desktop computer volunteers as a Grid resource upon screen saver activation, and as soon as the screen saver is deactivated, the executing job either stops or migrates. Thus, the resource owner is completely unaffected by the Grid job. The users submitting jobs to the public resources gain access to a virtualized Linux environment where standard Linux applications can run unmodified.

In the One-Click framework, resource owners can contribute without installing any client software at all. By using Java Applet technology, the resource owner simply points a Java-enabled browser to the MiG One-Click URL which will load an applet acting as a Grid resouce. Once the user of the donated computer wishes to stop the execution, the browser is simply closed down or pointed to another URL, and the execution stops. The MiG system eventually detects this event, by a timeout, and resubmits the job to another resource, where the job is resumed from the latest checkpoint that was made.

The use of Java applets provides a secure sandboxed executing environment that prevents the executing Grid jobs from harming the donated machine. The disadvantage of this approach is that all jobs must be written in Java and in addition comply with the presented framework, including the Java Applet Security Model. However the modifications that are needed to port an existing Java application are limited to using special methods for stdout and stderr, applying to the Java applet security model, and using the One-Click framework for remote file access. The One-Click framework also includes means to provide semi-transparent checkpointing of the applications at runtime.

Experiments have been performed to find the optimal block size for the remote file transfer that the framework includes. The experiments show that doubling the block size in each transfer gives the optimal tradeoff between bandwidth and latency as long as the CR is below 1.65.

References

[1] Rasmus Andersen and Brian Vinter, *Transparent remote file access in the minimum intrusion grid*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 311–318.

[2] David P. Anderson, *Boinc: A system for public-resource computing and storage*, GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Washington, DC, USA), IEEE Computer Society, 2004, pp. 4–10.

[3] Allan Bricker, Michael Litzkow, and Miron Livny, *Condor Technical Summary, Version 4.1b*, 1992.

[4] Brad Calder, Andrew A. Chien, Ju Wang, and Don Yang, *The entropia virtual machine for desktop grids*, VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments (New York, NY, USA), ACM, 2005, pp. 186–196.

[5] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes, *A case for grid computing on virtual machines*, ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems (Washington, DC, USA), IEEE Computer Society, 2003.

[6] Ian Foster and Carl Kesselman, *The grid: blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, November 1998.

[7] Henrik Hoey Karlsen and Brian Vinter, *Minimum intrusion grid - the simple model*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 305–310.

[8] Brian Vinter, *The Architecture of the Minimum intrusion Grid (MiG)*, Communicating Process Architectures 2005, sep 2005, pp. –.

Appendix F

Publication 6

Recent Developments in Grid Technology and Applications. G.A. Gravvanis, J.P. Morrison, H.R. Arabnia and D.A. Power, Editors, Chapter 10, ISBN: 978-1-60692-768-7
Brian Vinter, Rasmus Andersen, Martin Rehr, Jonas Bardino, and Henrik Karlsen: Towards a Robust and Reliable Grid Middleware

In: ISBN:
Editor: © 2007 Nova Science Publishers, Inc.

Chapter

TOWARDS A ROBUST AND RELIABLE GRID MIDDLEWARE

Brian Vinter, Rasmus Andersen, Martin Rehr, Jonas Bardino and Henrik Karlsen
University of Copenhagen, Department of Computer Science

Abstract

This chapter describes the philosophy behind a new Grid model, the Minimum intrusion Grid, MiG. The idea behind MiG is to introduce a 'fat' Grid infrastructure which allows much 'slimmer' Grid installations on both the user and resource side. Components that differentiate MiG from Globus and similar models include a zero-size Grid code base that is required to be installed on client or resource computers, mandatory payment and pricing of Grid services, end-to-end anonymity between consumers and producers and approximating the perception of a PC by means of Grid computing. In addition MiG provides a new, simpler approach to Virtual Organizations and a seamless integration of the Public Resource Computing model into Grid.

1. The idea behind the Grid

In the mid 1990's when Grid first was introduced, the idea of getting computing resources from a socket in the wall helped create a giant hype around Grid-computing. Since then one could argue that, as money has been poured into Grid research, the ambition level of the same research has dropped proportionally; to the point where Grid today often appears to be merely web-services in another wrapping! Grid research and Grid systems seem to be severely limited by initial choices made towards location, naming and security aspects in Grid. Most, or even all, of these were not obviously wrong, or were not at all wrong at the time they were made, but in the end we have ended up with a Grid model that is a long way from providing computing resources from a socket in the wall, particularly for commercial enterprises and in particular for private users.

Grid computing is just around the top of the hype-curve, and while large demonstrations of Grid middleware exist, including Globus toolkit[8] and NorduGrid ARC[9], the tendency in Grid middleware these days is towards a less powerful model, Grid services, than what was

available previously. This reduction in sophistication is driven by a desire to provide more stable and manageable Grid systems. While striving for stability and manageability is obviously right, doing so at the cost of features and flexibility is not so obviously correct.

The Minimum intrusion Grid, MiG, is a project that aims 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 a Grid infrastructure where the requirements on users and resources alike, to join Grid, are as small as possible – thus the minimum intrusion part. While striving for minimum intrusion, MiG still seeks to provide a feature rich and dependable Grid solution.

2. Grid Middleware

The driving idea behind the Minimum intrusion Grid project is to develop a Grid middleware that allows users and resources to install and maintain a minimum amount of software to join the Grid. MiG will seek to allow very dynamic scheduling and scale to a vast number of processors. As such MiG will close the gap between the existing Grid systems and popular "Screen Saver Science" systems, like SETI@Home.

2.1 Philosophy behind MiG

"The Minimum intrusion Grid", this really is the philosophy - we want to develop a Grid middleware that makes as few requirements as possible. The working idea is to ensure that a user needs only a signed x509 certificate, trusted by Grid, and a web-browser capable of secure HTTP, HTTPS. A resource on the other hand must also hold a trusted x509 certificate and in addition create a user – the Grid user – who can use secure shell, ssh, to enter the resource and once logged on can open HTTPS connections to the outside. The requirements then become:

	User	**Resource**
Must have certificate	Yes	Yes
Must have outbound HTTPS	Yes	Yes
Must have inbound SSH	No	Yes[1]

Table 1. Requirements for using MiG

3. What's wrong with the classic Grid systems?

While there are many Grid middleware systems available most of them are based on, or descendents of, the Globus toolkit. Thus the description below addresses what the author believe to be shortcomings in the Globus toolkit, and not all issues may be relevant to all Grid systems.

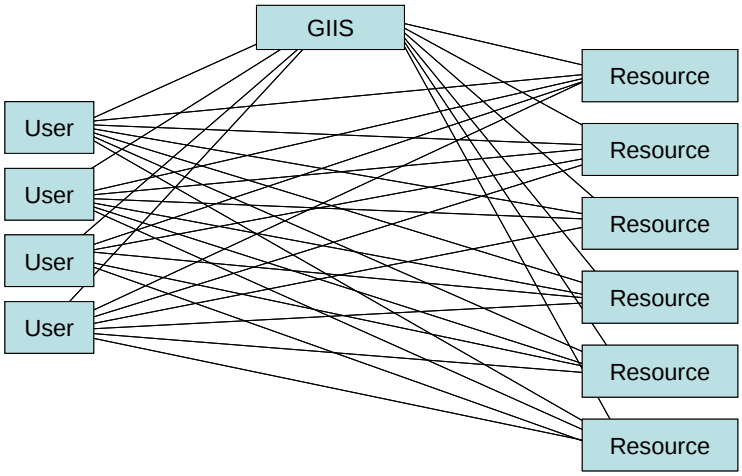
[1] Some resource models in MiG, known as sandboxes, do not even require inbound ssh.

3.1 Single point of failure

Contrary to popular claim, all existing Grid middlewares hold a central component that, if it fails, requires the user to manually choose an alternative. While the single point of failure may not truly be a single point, but comply with some level of redundancy, none of the components scale with the size of the Grid. Thus even for a Grid of infinite size a finite number of failures are required to make the overall Grid fail

3.2 Lack of scheduling

The classic Grid systems perform a job-to-resource mapping. However, an actual scheduling with a metric of success is not available. Work is underway in this in the community scheduler[16] but for this scheduler to work, the resources need to be exclusively signed over to Grid, i.e. a machine can not be accessed both through Grid and a local submission system.



Scheduling layout of the Globus Grid model

3.3 Poor scalability

The time taken to perform the job-to-resource mapping in the current systems scales linearly with the number of sites that are connected. This is already proving to be a problem in NorduGrid, which is one of the largest known Grids, though only 36 sites are connected. Imagining tens of thousands of connected sites is not likely. In the Grid service model scalability issues are more or less eliminated by absence of a single system view from a user perspective and thus forces the user to visit every site on the Grid before making a decision on where to place a new job.

3.4 No means of implementing privacy

The job submission API at the users machine communicates directly with all the potential sites, thus all sites know the full identity of all jobs on the Grid. This means that the grid user does not only reviles his or her full identity and intentions to the resource that end up executing the job, which in itself is bad enough, full disclosure is given to all participating resources in the Grid. Thus if one wish to make a map of ‘who-does-what’ on the Grid all that is needed is to place a resource on the Grid and log all incoming requests. There is not even a need to accept any jobs at all, simply to receive requests and the reject them.

3.5 No means of utilizing ‘cycle-scavenging’

Cycle-scavenging, or Screen Saver Science, utilizes spare CPU cycles when a machine is otherwise idle. This requires an estimate on how long the machine will be available and all classic Grid systems just assume that a free resource will be available indefinitely. Cycle scavenging in Grid has been partly demonstrated in NorduGrid by connecting a network of workstations running Condor, to NorduGrid, but Grid itself has no means of screen-saver science.

3.6 Requires a very large installation on each resource and on the user site

The middleware that must be installed on a resource to run NorduGrid, which is probably the most robust of the well known Grid middlewares, is more than 367 MB including hundreds of components. All of which must be maintained locally. This means that donating resources to Grid is associated with significant costs for maintenance; this naturally limits the willingness to donate resources.

3.7 Firewall dependency

To use the existing middlewares special communication ports to the resource must be opened in any firewall that protects a resource. This is an obvious limitation for growing Grid since many system administrators are reluctant towards such port-openings. One project that seeks to address this problem is the centralized-gateway-machine project under the Nordic Data Grid Facility[17] that receives jobs and submits them to the actual resource using SSH.

3.8 Highly bloated middleware

The existing middleware solutions provide a very large set of functions that are placed on each site, making the software very large and increasing the number of bugs, thus the need for maintenance, significantly.

3.9 Complex implementation using multiple languages and packages

Many current Grid middlewares have reused a large amount of existing solutions, for data-transfer, authentication, authorization, queuing, etc. These existing solutions are written in various languages and thus the Grid middleware uses more than 6 programming languages and several shell types, in effect raising the cost of maintaining the package further. The many languages and shells also limit portability to other platforms. The Globus project have acknowledged this as a problem and have switched entirely to Java for implementing their middleware.

4 Design Criteria for the Minimum intrusion Grid

All MiG component must design and implement a functional Grid system with a minimal interface between the Grid, the users, and the resources. The successful MiG middleware implementation holds the following properties.

4.1 Non-intrusive

Resources and users must be able to join Grid with a minimum of effort and with a minimum software installation. The set of requirements that must be met to join Grid must also be minimal. “Minimal” in this context should be interpreted rigidly, meaning that if any component or functionality in MiG can be removed from the resource or user end, this must be done, even if adding the component at the resource or user end would be easier.

4.2 Scalable

MiG must be able to contain tens of thousands, even millions, of resources and users without the size of the system impacts performance. Even individual PCs should be able to join as resources. For a distributed system, such as MiG, to be truly scalable it is necessary that the performance of the system is not reduced as the number of associated computers grows.

4.3 Autonomous

MiG should be able to perform an update of the Grid without changing the software on the user or resource end. Thus compatibility problems that arise from using different software versions should be eliminated by design. To obtain this feature it is necessary to derive a simple and well defined protocol for interaction with the Grid middleware. Communication within the Grid can be arbitrarily complex though since an autonomous Grid architecture allows the Grid middleware to be upgraded without collaboration from users and resources.

4.4 Anonymous

Users and resources should not see the identity of each other if anonymity is desired. This is a highly desirable feature for industrial users that are concerned with revealing their intentions to competing companies. A highly speculative, example could be two pharmaceutical companies A and B. Company A may have spare resources on a computational cluster for genome comparisons, while B may be lacking such resources. In a non-anonymous Grid model, company B will be reluctant to use the resources at company A since A may be able to derive the ideas of B from the comparisons they are making. However, in a Grid that supports anonymous users, A will not know which company is running which comparisons which makes the information far less valuable. In fact many comparisons will be likely to be part of research projects that map genomes and will thus reveal nothing but information that is already publicly available.

4.5 Fault tolerance

Failing machines or processes within the Grid should not stop users or resources from using the Grid. While obvious, the lack of fault tolerance is apparent in most Grid middlewares today. The consequences of lacking fault tolerance range from fatal to annoying. Crashes are fatal when a crashed component effectively stops users from running on Grid, i.e. a hierarchy of Meta Directory Servers.

If a resource that runs users' processes crash it becomes costly for the users that are waiting for the results of the now lost jobs. Finally crashes are merely annoying when a crashed component simply does not reply and thus slows down the users interactions with the Grid because of timeouts.

4.6 Firewall compliant

MiG must be able to run on machines behind firewalls, without requiring new ports to be opened in the firewall. While this requirement is quite simple to both motivate and state, actually coping within the restraints of this point may prove highly difficult.

4.7 Strong scheduling

MiG provides real scheduling, not merely job-placement, but it needs to do so without requiring exclusive ownership of the connected resources. Multi-node scheduling should be possible as should user-defined scheduling for dynamic subtasking. In effect MiG also supports meta-computing[2].

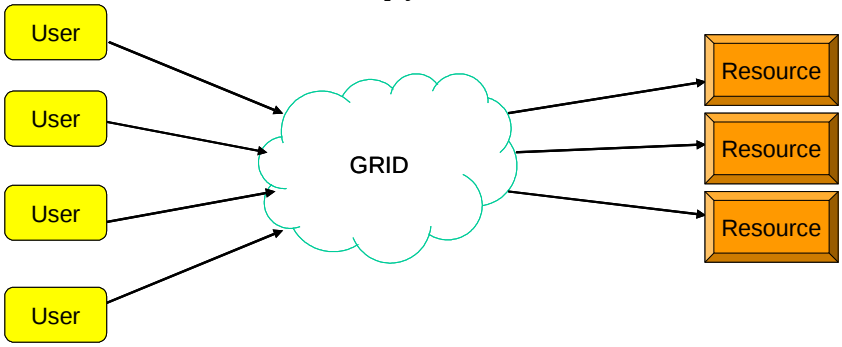
[2] Metacomputing is a concept that precedes Grid computing. The purpose of metacomputing is to create a large virtual computer for executing a single application.

4.8 Cooperative support

In order to improve the meta-computing qualities, MiG provides access to shared user-defined data-structures. Through these data-structures a MiG based Grid system can support collaborating applications and thus improve the usability of Grid.

5 The abstract MiG model

The principal idea behind MiG is to provide a Grid system with an overall architecture that mimics a classic, and proven, model – the Client-Server approach. In the Client-Server approach the user sends his or her job to the Grid and receives the result. The resources, on the other hand, send a request and receive a job. After completing the job the resource sends the result to the Grid which can forward the reply to the user.



The abstract MiG model

The Grid system should be disjoint from both the users and the resources, thus the Grid appears as a centralized black-box to both users and resources.

This model allow us to remain in full control of the Grid, thus upgrades and trouble shooting can be performed locally within Grid, rather than relying on collaboration from a large number of system administrators. In addition, moving all the functionality into a physical Grid system, lowers the entry level that is required for both users and resources to join, thus increasing the chances that more users and resources do join the Grid.

In MiG, storage is also an integrated component and users will have their own ‘home directory’ on MiG, which can be easily accessed and referenced directly in job-descriptions so that all issues with storage-elements and replica catalogues is entirely eliminated.

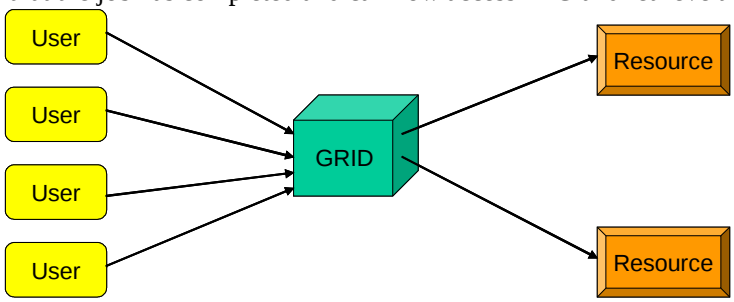
For a user to join, all that is required is an x509 certificate which is signed by a certificate authority that is trusted by MiG. Accessing files, submitting jobs and retrieving results can the all be done through a web-browser that supports certificate based HTTPS. As a result the user need not install any software to access Grid and if the certificate is carried on a personal storage device, e.g. a USB key, a user can access Grid from any internet enabled machine.

The requirements for resources to join MiG should also be an x509 certificate, but in addition the resource must create a Grid account in which Grid jobs are run. Initially MiG requires that this user can SSH into the account, some resources are run in a sandboxed pull mode instead but these are not described in this chapter..

6 The simple MiG model

In a simple version of the MiG model there's only a single node acting as the Grid. Clients and resources then communicate indirectly through that Grid-node. The interface between the user and Grid should be as simple as possible. The exact protocol remains a topic for investigation but, if possible, it will be desirable to use only the HTTP protocol or a similar widely used, and trusted, protocol. Towards the resources the protocol should be equally simple, but in this case, as we also desire that no dedicated Grid service is running on the resource, one obvious possibility is to use the widely supported SSH protocol.

When submitting a job, the user sends it to the Grid machine which stores the job in a queue. At some point a resource requests a job and the scheduler chooses a job to match the resources that are offered. Once the job is completed the results are sent back to MiG. The user is informed that the job has completed and can now access MiG and retrieve the results.



The simple MiG model

6.1 Considering the simple model

The simple model of course, is quite error-prone as the single Grid machine becomes both a single point of failure and a bottleneck which is not acceptable. The obvious solution is to add more Grid machines which can act as backup for each other.

7 The full MiG model

The obvious flaw in using the client-server model is that achieving robustness is inherently hard in a centralized server system where potential faults include:

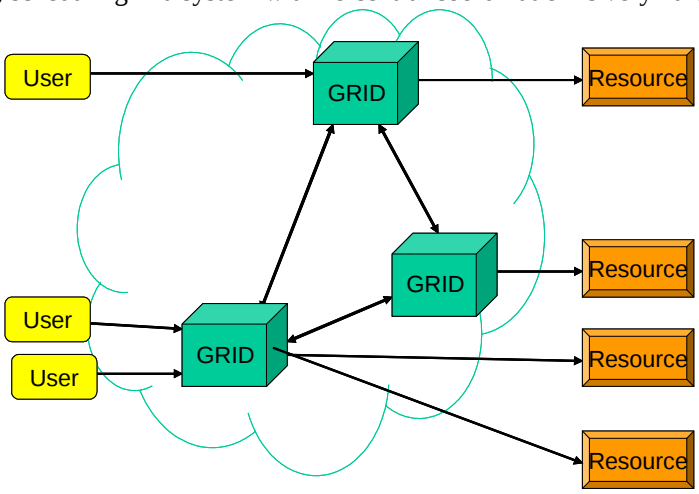
- Crashed processes
- Crashed computers
- Segmented networks
- Scalability issues

To correctly function in the presence of errors, including the above, error redundancy is needed. The desired level of redundancy is a subject to further investigations, but should probably be made dynamic to map the requirements of different systems. To address the performance issues Grid itself must be distributed so that users can contact a local Grid server. Thus workload will be distributed through the physical distribution of users.

Once a job arrives at a Grid server the server must ensure that the job is “deposited” at a number of other servers, according to the current replication rate. The user should not receive an acknowledgement of submission before the job has been correctly received and stored at the required number of servers.

Once a resource has completed a job the resource is expected to deliver the result. If, however, the client has not provided a location for placing the result, the resource can still insist on uploading the results. To facilitate this, the Grid should also host storage to hold results and user input-files, if a resource cannot be allocated at the time the client submits his job.

To facilitate payment for resources and storage a banking system should be implemented. To allow inter-organization resource exchange, the banking system should support multiple banks. Dynamic price-negotiation for the execution of a job is a very attractive component that is currently a research topic. Supporting price-negotiations in a system such as MiG where no central knowledge is available is an unsolved problem that must be addressed in the project. Likewise, scheduling in a system with no central coordination is very hard.



The full MiG model

7.1 Considering the full model

One topic for further investigations is: how do we schedule on multiple Grid servers? In principle we would prefer complete fairness, so that the order in which jobs are executed is not dependent on where they are submitted, i.e. to which MiG node. Such a full coordination between all nodes in MiG for each job-submission is not realistic since it will limit scalability, thus a model that allows scalability while introducing some level of load-balancing and fairness will have to be invented.

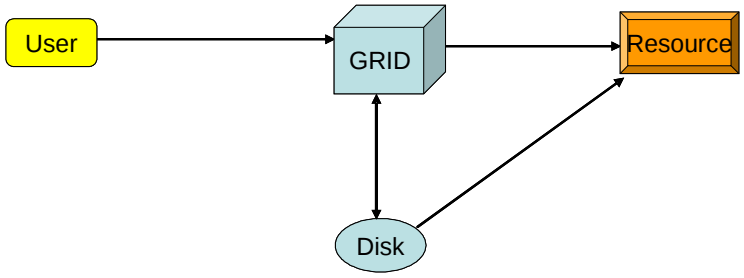
8 MiG Components

8.1 Storage in MiG

One difficulty that users report when using Grid is file access. Since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element, result files must be downloaded equally explicitly. On the other hand it is a well known fact that the expenses associated with a professional backup strategy often prohibit smaller companies from implementing such programs, and relies on individual users to do the backup - a strategy that naturally results in a large loss of valuables annually. Some interesting statistics include:

- 80% of all data is held on PCs (Source, IDC)
- 70% of companies go out of business after a major data loss (Source, DTI)
- 32% of data loss is due to user error (Source, Gartner Group)
- 10% of laptops are stolen annually (Source, Gartner Group)
- 15% of laptops suffer hardware failure annually (Source, Gartner Group)

By using the Grid, we do not just gain access to a series of computational resources, but also to a large amount of storage. Exploitation of this storage is already known about from peer-to-peer systems, but under “well-ordered" conditions it can be used for true Hierarchal Storage Management, HSM. When working with HSM the individual PC or notebook only has a working copy of the data which is then synchronized with a real dataset located on Grid. By introducing a Grid based HSM system, MiG offers solutions to two important issues at one time; firstly Grid jobs can now refer directly to the dataset in the home-catalog thus eliminating the need for explicit up- and down-loads of files between the PC and Grid. Second, and for many smaller companies much more importantly, we can offer a professionally driven storage-system with professional backup solutions, either conventional backup systems or, more likely, simple replica based backup - the latter is more likely because disks are becoming rapidly less expensive and keeping all data in three copies is easily cheaper than a conventional backup-system and the man-power to run it. A Grid based HSM system also allows small companies to outsource the service while medium and large companies can chose to either outsource or implement a Grid HSM in-house. Thus by introducing Grid based HSM, Grid can offer real value to companies that are not limited by computational power and these companies will thus be "Grid integrated" when Grid becomes the de-facto IT infrastructure.



MiG Storage support

8.2 Scheduling

Scheduling in Grid is currently done at submission-time and usually a scheduled task is submitted to a system where another level of scheduling takes place. In effect the scheduling of a job provides neither fairness for users nor optimal utilization of the resources that are connected to the Grid, and the current scheduling should probably just be called job-placement. Furthermore, the current model has a built in race-condition since the scheduling inquires all resources and submits to the one with the lowest time-to-execute. If two or more jobs are submitted at the same time they will submit to the same resource, but only one will get the expected timeslot. The MiG model makes scheduling for fairness much simpler as the local scheduling comes before the Grid scheduling in the proposed model.

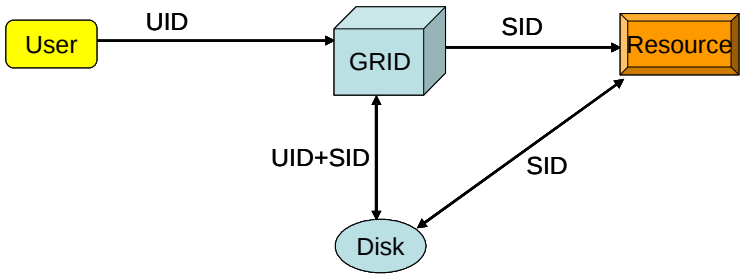
Scheduling for the best possible resource utilization is much harder and of much more value. The problem becomes one that may be described as: given the arrival of an available resource, and an existing set of waiting jobs, which job is chosen for the newly arrived resource so that the global utilization will be as high as possible?

The above is in the common case where jobs are more frequent than resources, in the rare case that resources are more abundant than jobs, the same problem is valid on the arrival of a job.

When scheduling a job, future arrivals of resources are generally not known, i.e., we are dealing with an on-line scheduling problem. On-line scheduling is an active research area, initiated as early as 1966 and continued in hundreds of papers, see and for a survey. This problem, however, differs from all these on-line scheduling problems investigated previously in that the resources, not the jobs, arrive over time in the common case. The problem also has some similarity with on-line variable-sized bin packing , but again with a twist that has not been considered before; the bins, not the items to be packed, arrive on-line.

8.3 Security and Secrecy

In Grid, security is inherently important, and the MiG system is designed be at least as secure as the alternative systems. The simple protocols and minimal software based on the resources make this goal easy to achieve, but still the mechanisms for security must be investigated. Secrecy is much harder and is currently not addressed in Grid. Privacy will mean much towards achieving secrecy but other issues are also interesting topics of research. I.e. if a data file is considered valuable, e.g. a genomic data sequence, how can we hold the contents of that file secret to the owner of the resource? In other words, can MiG provide means of accessing encrypted files without asking the users to add decryption support to his application?

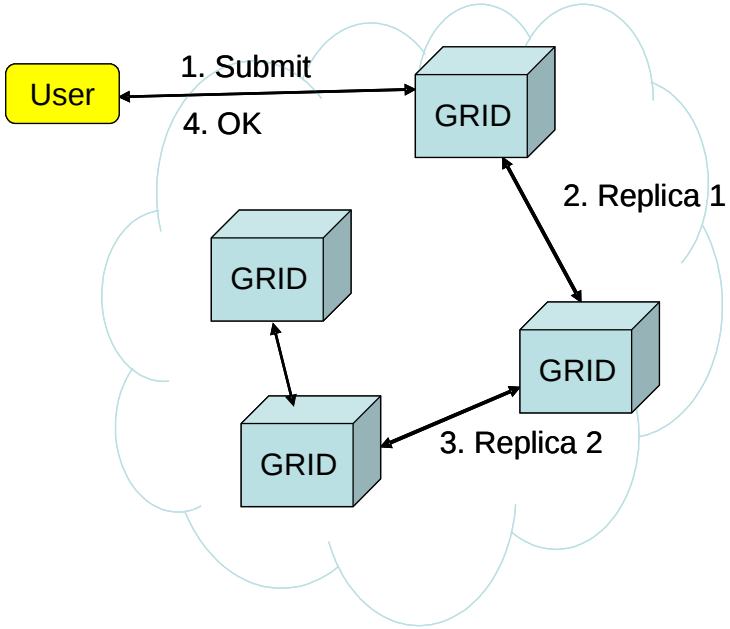


Anonymity and security model

8.4 Fault-tolerance

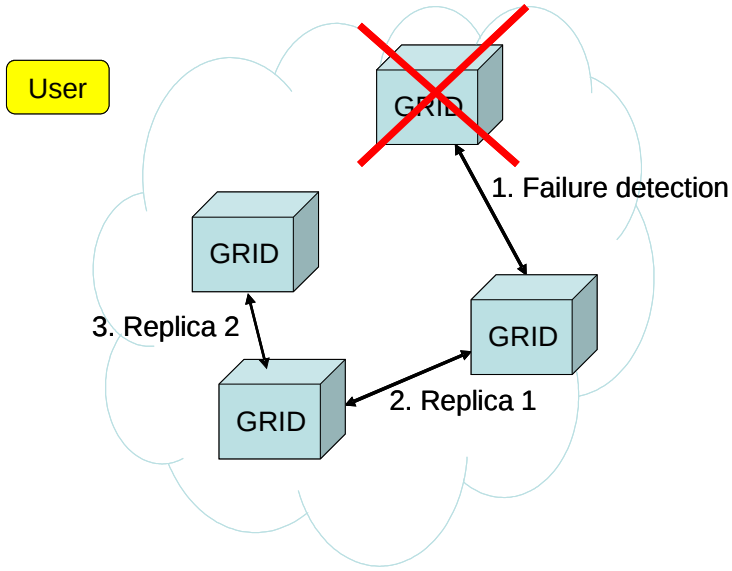
In a full Grid system errors can occur at many levels and failures must be tolerated on MiG nodes, resources, network connections and user jobs. Any single instance of these errors must be transparent to the user. More complex errors of course, or combinations of the simple errors, cannot fully be hidden from the users, i.e. if a user is on a network that is segmented from the remaining internet we can do nothing to address this.

Achieving fault tolerance in a system such as MiG is merely a question of never loosing information when a failure occurs, e.g. keeping redundant replicas of all information. shows how a submitted job is replicated when it is submitted.



Replicating a new job

Recovering from a failure is then a simple matter of detecting the failure and restoring the required number of replica's as shown in where the number of replicas is three.



Recovering from a failure

8.5 Load balancing and economics

Load balancing in distributed systems is an interesting and well investigated issue. However load balancing for, potentially, millions of resources while maintaining a well defined measure of fairness is still an unsolved issue. However adding economics to the equation actually makes this easier. Since MiG should support a market oriented economy, where the price for executing a job is based on demand and supply, this introduces a simple notion of fairness which is that resources should optimize their income while users should minimize their expenses.

In case there are more jobs than resources, which is the common case, the next job to execute is the job that is willing to pay most for the available resource. In case two or more jobs bid the same for the resource the oldest of the bidders is chosen.

In the rare case that there are more resources offering their services than there are jobs asking for a resource, the next available job is sent to the resource that will sell its resources cheapest. In case more resources bid at the same price, the one that have been waiting the longest wins the bid.

8.6 Shared data-structures for MiG

When people with little knowledge of Grid computing are first introduced to Grid, they often mistake it for meta-computing and expect the Grid to behave as one large parallel processor and not a large network of resources. This misunderstanding is quite natural, since such a Grid computing model would be highly desirable for some applications, of course most parallel applications cannot make use of such an unbalanced virtual parallel processor.

However, to support the applications that can make use of Grid as a meta-computing system, to address this MiG provides support for shared data-structures which are hosted on Grid.

Users who wish to utilize MiG for meta-computing may currently use one of four approaches

1. Shared files
2. MiGSpace communication
3. SQL database
4. CSP

The use of shared files are often seen in applications that are executed on a LAN and is quite simple, though not very powerful. Because, MiG works with a Grid based home-directory and runtime access to the files in this catalog, rather than a pure copy-semantics as commonly seen with Grid.

MiGSpace communication is in the tradition of Linda. Similar to the Linda tuple space, tuples provide the granularity for shared entities in MiGSpace. Single variable based granularity is ineffective in high latency environments. Communication latency is of significance to the task grain size in distributed shared memory systems. When communication latency increases, the grain size must be coarser to achieve good performance. Document based granularity like that found in xSpace is overly specialized. In contrast to JavaSpaces, tuples in MigSpace are flexible ordered collections of typed elements, similar to those found in Linda. Arrays, sets and matrices remain relatively common data structures in scientific computing, even though object oriented designs and languages have become more popular. MiGSpace supports matrices and arrays with its simple tuple approach. If desired is very easy to develop a tuple-to-object bridge. The following is a formal definition of tuples in MigSpace:

A tuple consists of finite collections of ordered typed elements. Each element can be an actual or a formal. The following notation for a tuple is used:

$<P_1, P_2, P_3, ...P_j>$,

where Pi is an element.

A tuple consists of only actuals. Actuals have a type and a value. The

following is an example of a tuple with three elements in which all are actuals:

$<1_{int}, "John"_{string}, 2_{int}>$

Applications may this write actuals to the MiGSpace and read by using formals. Since parallel programming with tuple-spaces is a well established paradigm, a large set of algorithms that are designed for use with tuple-spaces. MiGSpace extends the classic tuple-space model by introducing the option of reading and writing entire sub-spaces, this is introduced to help hide latency over wide area networks.

Spaces in MiGSpaces are implemented as files and thus inherit their full security model from the file-level security.

The SQL interface is quite straight forward, applications may access a MiG hosted SQL database through a MiG enabled ODBC interface that seamlessly wraps all SQL queries in the MiG security mechanisms and forwards the request to the MiG server-side SQL engine. Likewise the reply is wrapped in the security layer and returned to the ODBC layer at the execution host where the ODBC library translates the reply into standard ODBC format.

CSP, Communication Sequential Processes, is a well established model for designing and implementing concurrent applications[ref]. In CSP the communication mechanism is synchronous channels and in MiG-CSP these channels have been extended to a Grid

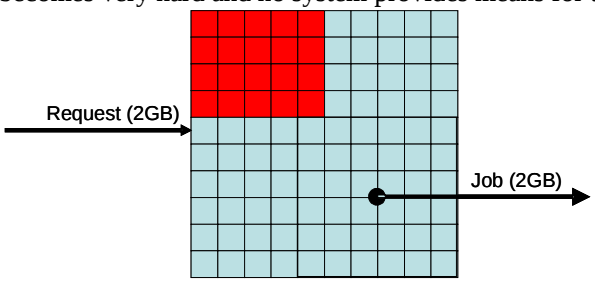
environment. The channels too are implemented as files and thus inherit the entire security model from the MiG file modes.

8.7 Accounting/Price-negotiations

Grid becomes really interesting once users can purchase resources on Grid, thus transforming Grid from a resource sharing tool into a market place. To support this vision, MiG does not only do accounting but also support a job bourse, where the price for a task can be dynamically negotiated between a job and a set of resources. Such dynamic price-setting is also a known subject, but combining it with load-balancing and fairness in a truly distributed system has not been investigated.

8.8 User defined scheduling

An advanced extension of the online-scheduling problem is the subtasking problem, where a job may be divided into many subjobs. If the subtasks have a natural granularity the task is trivial and known solutions exist, including functioning systems, such as SETI@Home. If, on the other hand, a subtask can be selected that solves the largest possible problem on the given resource, the problem becomes very hard and no system provides means for this today.



Dynamic sub-scheduling

When comparing with on-line bin packing, this variant of the problem has one further twist to it; the size of an item (a subtask) may depend on which other items are packed in the same bin, since the data needed by different subtasks may overlap.

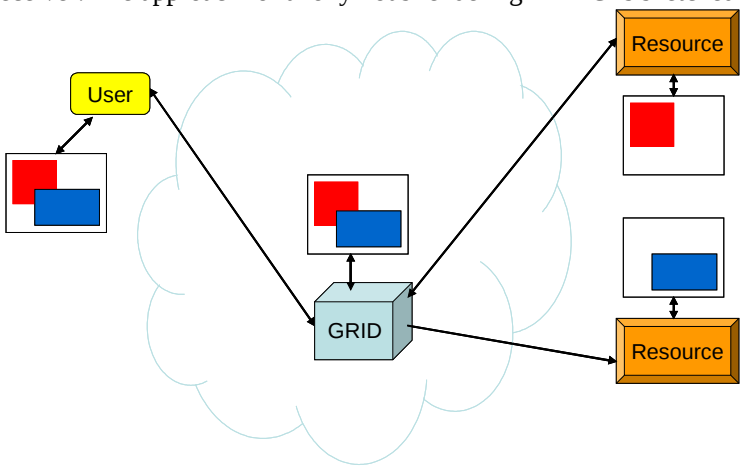
MiG has developed a model where a job can be accompanied with a function for efficient sub-tasking. The demonstration application for this will be a new version of the Grid BLAST application, which is used in Bio-Science for genome comparisons. The efficiency of BLAST depends on two parameters; input-bandwidth and available memory. We currently developing a dynamic subtasking algorithm that creates subjobs fitted for resources as they become available.

8.9 Graphics rendering on Grid

Currently Grid is used exclusively for batch job processing. However for Grid to truly meet the original goal of “computing from a plug in the wall”, graphics and interactivity is needed.

In this respect MiG makes things more complex than the existing middlewares since MiG insists on maintaining anonymity, e.g. we insist that a process can render output to a screen-buffer that it cannot know the address of.

The solution to this problem is similar to the storage model. A ‘per-user’ frame-buffer is hosted in the MiG infrastructure, and resources can render to this, anonymous, region. Users on the other hand can choose to import this buffer into their own frame-buffer and thus observe the output from their processes without the hosts of these processes knowing the identity of the receiver. The approach for anonymous rendering in MiG is sketched in .



Anonymous graphics rendering in MiG

9 Virtual Organisations

Facilitating the organization and work of Virtual Organizations is amongst the premiere advantages of Grid computing. In [4] it is stated that “the real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.” Resource sharing in this context is not only about files, but access to all kinds of resources like computational power, external data, software and specialized hardware. When sharing resources in loosely organized collaborations, such as virtual organizations, one need the ability to apply a number of rules and conditions which define the policy of the individual collaborations. Thus, seen at the utmost abstraction level, a virtual organization is a set of well defined individuals who share a well defined set of resources. While essential to Grid computing, Virtual Organizations are defined purely as concept in Grid computing and not strictly defined by a protocol or similar. The most widespread implementation of Virtual Organizations is the Virtual Organization Membership Service, VOMS, which essentially works by allowing the user to request a proxy-certificate from a given Virtual Organization, using that proxy-certificate the user may then continue to submit the desired job to a resource that accepts the VO proxy-certificate.

9.1 VOMS

VOMS work by allowing the user to request a proxy certificate that verifies VO membership from any VOMS server. If the VOMS server has information to the end that the requesting users is in fact member of the desired VO it returns the proxy-certificate. Using that proxy-certificate the user may then continue to submit the desired job to a resource that accepts the VO proxy-certificate. The process is shown in figure 2.

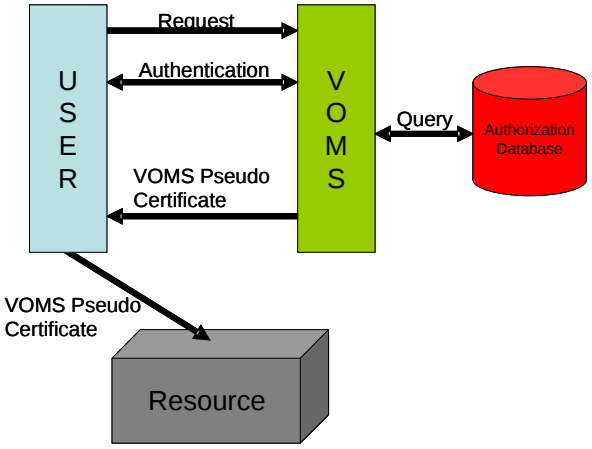


Figure 2. VO membership certificate in VOMS

When presented with a proxy-certificate the resource verifies that the user is member of an accepted VO and continues the authorization process. The resource may maintain a list locally with banned users and deny a user access even though VO membership has been confirmed by the VOMS server through the proxy-certificate.

9.2 CAS

Community Authorization Service, CAS, which is part of the Globus toolkit[8][7], seeks to provide a more fine grained access control than simply membership of a VO. CAS works by introducing a new abstraction level in the system called roles. Roles are similar to sub-groups in a VO except that roles are not only associated with a set of resources but also with the operations that may be performed on the resources, i.e. a specific role may only provide read privileges to a data-set but not write privileges. The process that implements this mechanism is identical to the overall VOMS process as shown in figure 2.

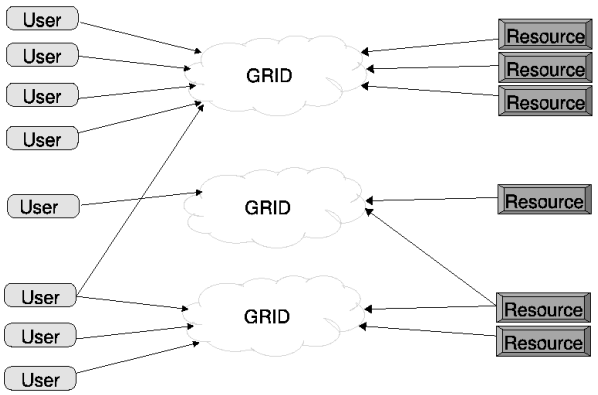
9.3. GridShib

GridShib[5] is a project which seeks to replace ordinary VOMS and CAS systems with a Shibboleth[1] based authorization model. One of the driving motivations for Grid-Shib is the same as one of the primary motivations for MiG, namely the need for user privacy.

9.4 VGrids

The MiG design has made it easy to obtain a lot of the Grid features that was previously very hard to implement. The development of MiG in general has greatly benefited from the knowledge and mistakes learned by the first middleware that appeared. The VOMS approach of proxy certificates is cumbersome and represents some concerns on manageability and security, as an example it is not possible to revoke a proxy-certificate. Solutions to these issues was discussed for some time in the MiG team. We seek a model that supports the anonymity required by MiG, and which does not introduce proxy-certificates, a concept which is entirely eliminated from the MiG design and which should not be re-introduced in order to support VOs. At the same time we also need to keep the anonymity between user and resource and provide the strong-scheduling capabilities found in MiG. Another important observation from real-world Grids is that many large resources are hard to connect to Grid since they run on user group quota allocations and often use a fair-share scheduling mechanism. In order to support access to a resource from Grid by two independent user groups, complex submission handling or even multiple Grid entry-points must often be introduced, both of which increase the complexity of managing resources towards Grid and thus decreases motivation to join a Grid system. We believe it is imperative to support the natural regulation mechanisms in local sites, and we also believe that local administration of Grid related options should be kept at a minimum, according to the project name, minimum intrusion grid. The proposed solution is an entirely different approach to virtual organizations. In its nature Grid seeks to allow a set of users to share a set of resources, while VOs seek to control which users of a Grid share which resources, in essence a VO becomes a subset of a Grid. With this in mind we choose Grids as our basic mechanism and treats a VO as Grid-whithin-a-Grid, or Virtual Grids, VGrids (figure 3). A VGrid appears to a user almost as an ordinary Grid, it has users, compute-resources and data-resources. As MiG seeks to hide much of the Grid complexity to the user, all data-resources a user has access to are presented in the form of a unified file-system, and VGrid data-resources appear as subdirectories in the users home-directory. Resources never see the identity of the users due to the anonymity feature of MiG. It therefore makes no sense for resources to maintain a local list of banned users as in traditional middleware. If a resource joins a VGrid it grants access to all users within it.

In fact, today there are no resources in the ordinary MiG Grid any longer, all resources are located in one or more VGrids.



VGrids are integrated by design.

The set of allowed users in a VGrid consist of two types, owners and members. All valid MiG users are allowed to create new VGrids and can then include any other user they know of as either co-owners of the VGrid or ordinary members of the VGrid. The user who creates it automatically becomes an owner of the new VGrid. Owners can add and remove other owners and members, but a VGrid must always have at least a single owner. Besides having the authorization to manage other owners and members an owner also has the privileges as regular members, that is to access the resources in the VGrid and the files belonging to the VGrid.

The authorization structure is hierarchical and as such similar to the structure of VO's. If you have owner or member rights of a VGrid you automatically have the same rights on all sub VGrids. This means that an owner of VGrid V0 is also an owner of V0/V1 and V0/V1/V2 but not the other way round, i.e. some VGrids may have owners that are not even members of the parent VGrid if this is desired the MiG servers and the client never communicate directly with the resources, everything goes through the central MiG servers. This means that the input files needed to execute a job must first be uploaded to the server from where the resource can retrieve the file before executing the job. After a job has been executed the outputfiles are uploaded by the resource to the server and the user can download the file or use it in new job submissions or simply leave the file at the server where it may be safely stored.

To support file sharing between members of a VGrid, a directory is created on the server where all members are allowed to read and write. It looks just like any other directory in the members home directories, but the files within it are readable and writable by all VGrid members.

One of the observed complications with the VOMS model is the problem of having two user groups with each their allocation on a large resource that both accesses their allocation through Grid. In the VGrid model this becomes extremely easy, and both allocation quotas and fair share scheduling is supported by default and without requiring any administration on the local resource. Upon creating a VGrid the owners of that VGrid can add resources to it. As in the original MiG model a resource is simply an ordinary user account that allow incoming ssh and outgoing https. The administrator of a resource allocation simply creates an account on the resource, as he would do with a new member of his research team, and registers that account with the VGrid. From that point the VGrid can use the resource, but locally at the resource the VGrid use appears simply as the use of an ordinary, untrusted, user.

VGrids have a number of additional advantages and features compared to the VOMS approach. One is custom optimized job submission When a VGrid is created two web-page references are automatically generated within the MiG namespace. A public page that can be accessed by all Internet users where the owners can promote and publish information about the project, and a private page which is only accessible by members of the VGrid. Within the private page a custom job submission page may be created, using HTML. This page can be optimized to the special purpose of the VGrid, often in the form of an application portal to Grid execution by the VGrid owner using a set of predefined names for the HTML controls in a HTML form which has the MiG server as target. The MiG server generates a job description file based on the values of the HTML controls when it receives the form and submits it to the execution queue of the VGrid on behalf of the user. All usual HTML controls

are thereby available for the VGrid owners when they create their specialized submit page as well as Cascading Style Sheets (CSS), a technology often used in conjunction with HTML to create a set of web pages with a consistent look.

Another special component is the VGrid Monitor and Statistics. A Grid monitor with a snapshot of the current state of the Grid and a web page with various statistics e.g. the total number of jobs the Grid has processed are two features that have been a part of most Grid middlewares for some time. Besides an overall monitor and statistics page for the complete Grid, the same information is available for the separate VGrids. The default is that the VGrid monitor and statistics pages are only accessible by members of the VGrid, but the owners can change this to make the information public available.

VGrids also allow for easy information sharing. A WIKI is a special kind of website where users can easily add and edit pages and content. It is a very simply way for a group of people to collaborate and together create and maintain information. Perhaps the most known WIKI is the wikipedia [3] where Internet users together have created an encyclopedia that at the time of this writing has more than 950.000 articles and more than 900.000 registered users that maintain the information. This is big-scale collaboration! The fundamental technology behind a WIKI is HTTP and HTML, technologies as MiG is build upon and compatible with. It is possible to install a WIKI back-end on the MiG server and let the owners of MiG VGrids make WIKI functionality available on VGrid pages that can be accessible to the public or to members only.

11 Conclusions

The purpose of this paper is to motivate the work on a new Grid middleware, the Minimum intrusion Grid, MiG. MiG is motivated in a set of claimed weaknesses of the existing Grid middleware distributions, and a desire to develop a model for Grid computing that is truly minimum intrusion.

The proposed model will provide all the features known in today's Grid systems, and a few more, while lowering the requirements for a user to simply having an X.509 certificate, and for a resource to have a certificate and create a Grid-user who can access the resource through SSH.

While MiG is still in its very initial stage, users can already submit jobs and retrieve their results, while maintaining complete anonymity from the resource that executes the job.

References

[1] R. L. Graham, Bounds for Certain Multiprocessing Anomalies, Bell Systems Technical Journal, vol 45, 1563--1581, 1966

[2] Y. Azar, On-Line Load Balancing, Online Algorithms: The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[3] J. Sgall, On-Line Scheduling, Online Algorithms: The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[4] J. Csirik, An On-Line Algorithm for Variable-Sized Bin Packing, Acta Informatica, 26, pp 697--709, 1989.

[5] J. Csirik and G. Woeginger, On-Line Packing and Covering Problems, Online Algorithms:

The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[6] L. Epstein and L. M. Favrholdt, On-Line Maximizing the Number of Items Packed in Variable-Sized Bins, Eighth Annual International Computing and Combinatorics Conference (to appear), 2002

[7] I. Foster. The Grid: A New Infrastructure for 21st Century Science. Physics Today, 55(2):42-47, 2002.

[8] I. Foster, C. Kesselman. The Globus Project: A Status Report. Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998.

[9] P. Eerola et al. "Building a Production Grid in Scandinavia". IEEE Internet Computing, 2003, vol.7, issue 4, pp.27-35.

[10] R. Fielding et al, RFC2616 Hypertext Transfer Protocol -- HTTP/1.1, http://www.rfc.net/rfc2616.html, The Internet Society, 1999 .

[11] T. Ylonen, SSH - Secure login connections over the internet, Proceedings of the 6th Security Symposium, p 37, 1996.

[12] S. F. Altschul et al., Basic local alignment search tool, J. Mol. Biol. 215:403-10, 1990.

[13] G. Barish and K. Obraczka, World Wide Web Caching: Trends and Techniques, IEEE Communications Magazine Internet Technology Series, May 2000.

[14] Minimum intrusion Grid - The Simple Model, Henrik H Karlsen and Brian Vinter, in proc. of ETNGRID 2005 (to appear)

[15] Transparent Remote File Access in the Minimum Intrusion Grid, Rasmus Andersen and Brian Vinter, in proc. of ETNGRID 2005 (to appear)

[16] The Community Scheduler Framework, http://csf.metascheduler.org, 2005.

[17] The Nordic Data Grid Facility, NDGF, www.ndgf.org, 2003.

[18] Data Clinic, http://www.dataclinic.co.uk/data-backup.htm