Project Number CS-GXS-0501

Development of Algorithms on the Grid

A Major Qualifying Project Report

Submitted to the Faculty of the

Worcester Polytechnic Institute

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science

By

David Norcott

Kai Rasmussen

April 28, 2005

Approved:

Professor Gábor Sárközy,

Major Advisor

Abstract

A parameter study involves analyzing situations myriad times with varying parameters and is both computationally- and data-intensive; yet there is a great need for these studies in many areas of science and engineering. In response to this need a module for the P-GRADE Portal was developed at MTA-SZTAKI in Budapest, Hungary. This allows researchers in all areas of science to perform and visualize these studies with ease as parallel applications on the P-GRADE Portal.

Acknowledgments

First and foremost we would like to thank Professor Gábor Sárközy for creating such a wonderful experience for us. We have learned so much in the course of this project and enjoyed the beautiful city of Budapest, Hungary; none of this would be possible without his foresight, guidance, and hard work.

We would also like to thank Professor Stanley Selkow for all his assistance in this project both in Worcester and in Budapest.

We would especially like to thank all of our colleagues at MTA-SZTAKI for their help, guidance, and for being so open and generous: particularly Dr. Péter Kacsuk and Robert Lovas for developing this project and providing such a great opportunity; József Patvarczki for all his help, encouragement, and patience; Gábor Hermann, Csaba Németh, and Zoltán Farkas for helping us over countless roadblocks; and Ágnes Janscó and Norbert Podhorszki for helping us in so many ways, many beyond the scope of our project.

And finally a special thanks to everyone who helped us learn the language and culture of Hungary, for without them we would surely be lost: Clarie Szuhai, Dr. Zwack, and of course Ádám Kornafeld, for sure.

Table of Contents

Table of Figures

1 Background

1.1 Parallel Computing

1.1.1 Structure: Supercomputers, Clusters, Grids

Computer applications are demanding more and more resources, as computation problems get more complex. Specifically, scientific applications may need terabytes of storage room and multiple processors. Grid computing aims to fulfill the ever increasing needs of these applications. Ian Foster defines the grid as "coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations."[10]

A Grid infrastructure is a large heterogeneous group of distributed resources connected by a network. The grid is a natural evolution from supercomputers and computer clusters. Supercomputers are highly homogeneous systems that have fast and reliable processors, memory, network and I/O access. Clustered systems have mostly the same make-up of a supercomputer, but it may contain some heterogeneous parts. The Grid is very heterogeneous and may contain supercomputer and cluster parts.

One important tool for Grid architecture is Condor. Condor was developed at the University of Wisconsin in order to create a Grid system where each user was able to contribute as much as they wanted [34]. A main design requirement of Condor was that resource intensive programs would not affect the quality of service for less demanding users.

Condor is a middle level resource manager. A Condor pool is a collection of heterogeneous resources that can be used. The Condor job-manager has load information on all machines in its pool [35]. It uses this resource monitoring to best decide where to run jobs. Condor can also continuously checkpoint programs, and save all information needed to restart the job, in case of a machine failure. In this case, Condor can migrate the job from one resource to another. This gives the user the most reliability possible, as well as uses the available resources as efficiently as possible.

Condor pools can be linked together by the Condor Flocking mechanism to further utilize resources. Jobs can move from one pool to another freely. When one pool is overloaded, it will routinely checkpoint a user's jobs and migrate them to another pool.

Another important Grid system is Globus. Globus is a collection of high-level tools for computational grids [9]. Globus provides similar features as Condor in their middle-ware Metacomputing Toolkit. Instead as acting as a resource manager, Globus's toolkit will contain resource managers that can be used. For example, Condor can be used as a job-manager inside Globus. Because Globus provides a large mixed toolkit behind a well-defined interface, new services can easily be added without changing the underlining grid architecture.

Grid architectures follow a Master/Slave topology. There is a Master Process that breaks work up into smaller jobs and distributes these jobs among its slaves. The slave processes performs the work and return the results back to the master process. The master process is usually a head/access node of a grid or cluster. It is responsible for keeping load information for all of its resources in order to distribute work efficiently. In a more advanced topology, the slave processes are not independent of each other. This is an intercommunication topology. In an Intercommunication Topology, slave processes are aware of other slave processes. This allows them to communicate with each other and share data.

1.1.2 Programming

The traditional paradigm of the programming model assumes that a program will execute instructions sequentially. This is not so in the case of parallel computing; clusters and grids can execute instructions simultaneously across multiple processors. It follows then that parallel programs are inherently more complex than sequential programs and need a new paradigm.

Several new concepts arise when dealing with parallel programs. First is the idea of concurrency, or what actions can occur simultaneously. A parallel program consists of one or more tasks that execute concurrently. These tasks encapsulate a sequential program so they can read and write to local memory, etc. These tasks can perform four additional operations: send messages, receive messages, create tasks, and terminate. Tasks send and receive messages along ports or channels.

The second concept related to parallel programming is location, or on which processors those actions take place. Tasks, resources, and data can be local, meaning they reside on the same processor. They can also be remote, meaning they reside on remote processors. Here message passing is important as tasks are dependent on it to access remote resources.

There are several models for parallel programming all of which stem from the central ideas of concurrency and location. The first is "message passing." Message passing most closely follows the previously explained paradigm and is among the most popular of all parallel programming models although in practice most programs that implement message passing run a set number of tasks on a set number or processors and in general do not dynamically create tasks or allow multiple tasks per processor (because the number of tasks and processors is known before hand). This is called a single program multiple data (SPMD) model because each task executes the same sequential program on a different set of data. The message passing model is the most important model for this project because it closely follows the needs of a parameter study.

Two other parallel programming models worth noting are "data parallelism" and "shared memory" models. The typical use of a data parallel program is to perform the same operation across multiple elements in a data set. A prime example of this is to update all elements in an array in an identical way, such as "divide all elements in this array by two." Data parallel programs are often compiled in such a way that they become SPMD programs. In quite the opposite direction "shared memory" parallel programs execute operations across a shared memory space. This allows all the tasks involved in the program to read and write asynchronously to this shared memory space thereby simplifying the message passing needed but limiting the concept of locality [8]. While both data parallelism and shared memory are interesting models they are not suitable for this project.

The parameter study application created in this project will implement the message passing paradigm. In order to create a message passing parallel program more than just a paradigm is necessary, a standard interface is needed; hence PVM and MPI were developed. MPI stands for Message Passing Interface and it was developed as a standard for designing parallel applications. The first MPI API was specified between 1993 and 1994 by a community of about forty high-performance computing experts. The public domain implementation was written at Argonne National Laboratory and is currently available for virtually all major computer architectures [29]. At the time, parallel processing was an emerging technology and each Massively Parallel Processor (MPP) vendor was creating their own proprietary message-passing API. Therefore it was not possible to write portable applications as each vendor had such different interfaces. MPI was intended to be a standard for MPP which vendors would use to implement parallel programs. Since MPI was created to suit the needs of these vendors the focus became very specific.

The MPI Standard is centered on the needs of vendors. The first aspect considered when creating the MPI standard was performance since that is what vendors need the most. MPI sacrifices some features such as fault tolerance and scalability in favor of performance [11]. On the other hand, the MPI standard is flexible enough that provided they follow the standard, vendors can implement the functions in the way that they most see fit.

The large set of communication routines that MPI contains provides many features. Among these is great control over communication between groups of processors. This is done by the Communicator which is one of the most important concepts introduced by MPI. Communicators are bound to a group of processes and ensure that messages sent between these processes arrive correctly. Beyond that, MPI also has the ability to specify communication topologies and create derived data types that describe messages of non-contiguous data [11]. What this means is that the MPI library transparently does the appropriate data conversion when data is sent between different systems, which highlights another important part of the MPI standard: MPI can run jobs across heterogeneous systems where a mixture of processors of different architectures are clustered together [29].

PVM stands for Parallel Virtual Machine and is built around, obviously enough, the concept of a virtual machine. This allows for a heterogeneous collection of processors to appear logically to the user as a single parallel machine. PVM was started in the summer of 1989 when at the Oak Ridge

National Lab by Vaidy Sunderam, a professor at Emory University, and Al Geist a researcher at Oak Ridge. At the time there was a need for a framework to explore heterogeneous distributed computing. Unlike MPI, PVM strove to stay simple because portability was found to be more important than performance. This is because communication across the internet was slow and PVM was focused more around research and so fault tolerance, scaling, and heterogeneity were important.

As with MPI, PVM is also portable in that an application written on one parallel system can be copied to different architectures, and then compiled and executed without modification. But PVM is not only portable but also interoperable. This means that with PVM the resulting executables can also communicate with each other. MPI does not specifically prohibit interoperation but there is no standard defined for it.

Another area where PVM stands out is in regards to fault tolerance. For many applications run even on a parallel system computing time can reach hours or days. In this case, if a single resource is lost it may cause the entire operation of the application to hang. PVM supports a basic fault tolerance allowing tasks to be notified if another task fails. PVM has also has support for dynamic resource management, so when a resource is lost another can easily be allocated [11].

MPI and PVM are the most notable and most mature methods for message passing in a parallel application but neither is perfect and both have separate strengths and weaknesses. Currently a project is underway to bridge these two technologies at the University of Tennessee and Oak Ridge National Laboratory. The project is called PVMPI and it will allow users to use the virtual machine features of PVM and the message passing features of MPI. PVMPI would have three symbiotic functions: it would use vendor implementations of MPI when available; allow applications to access PVM's virtual machine thereby providing greater fault tolerance and resource control; and it would use PVM's network communication to transfer data between different vendor's MPI implementations allowing them to interoperate within the larger virtual machine [11]. This proves to be an interesting and potentially very useful technology.

1.1.3 Jobs

In a heterogeneous architecture, such as the Grid, reliability cannot be guaranteed. Imagine that a running parallel job has been computing for an hour. In the final moments before print out, one of the final distributed processors becomes over-loaded and crashes. Valuable processing time will have been lost and the job will need to be restarted.

Grid tools, such as Condor, uses a technique called Checkpointing in order to save the state of a process. Checkpointing is saving everything that is needed in order to restart the process at regular intervals so that the result of the program will not be changed [24]. This includes a process's data and stack segments, and information on open files and pending signals.

This Checkpointing is done transparently so that the user will not know that is happening in the background. Later the user can seamlessly create a new process form the collected information and restart the process from the last Checkpointed location. The benefits of Checkpointing are fault-tolerance, as seen in the given example, as well as high throughput computing. The master process will have complete load information of all its resources. It can use Checkpointing for load balancing. The master will migrate processes from overloaded resources onto unused resources.

Migration is moving a process from one resource to another. This is useful when a resource is being overloaded and it can no longer run a specific job. Checkpointing the job, sending the checkpoint file to a new resource and resuming the job from the checkpoint file, performs migration.

Load balancing is assigning jobs to resources in an organized manner. The goal of load balancing is to avoid overloading any one resource and to best use available under loaded resources. In a master-slave topology, the master process will make resource allocation decisions based on load information. This can either been done in a static or in a dynamic fashion.

In static load balancing, the master process allocates a resource for each job before they start. This is best for distributed systems that are homogeneous and when workload is even. When some resources perform faster than others, or when some workloads are shorter than others, some resources remain idle while others finish. This is an inefficient use of resources.

In dynamic load balancing, resources are given tasks when they are idle. This makes most use of available resources. In a Grid architecture, which is heavily heterogeneous, dynamic resource allocation is the most efficient load-balancing algorithm. When faster processors finish their jobs, they do not sit idle for very long. If the master process can assign a job to an idle processor, it does.

Two dynamic algorithms that operate without a master process are the receiver-initiated and the sender-initiated algorithms. A receiver-initiated algorithm is an algorithm under loaded resources which requests more work. In a sender-initialed algorithm, the overloaded resource gives away its work [3]. Assume each resource has load thresholds T-low and T-high where T-low indicates an under loaded resource and T-high for an overloaded resource. In a sender-initiated algorithm, a resource sends out requests to known systems when its workload is greater than the T-high threshold. When a positive acknowledgement is received, the overloaded processor redistributes its work to the under loaded system [7]. A receiver-initiated algorithm works in the same way, but is triggered when a task-load drops below the T-low threshold.

Hummel describes a receiver-initiated algorithm called a work stealer. In this algorithm, each resource has a task load L. At a random time interval, a given resource attempts to balance its load at a probability of 1 / L. On load balancing, the resource chooses another known resource at random and matches loads between them, if their load difference is greater than a set threshold. [14]. This algorithm favors under loaded processors, but allows both senders and receivers to initiate load balancing.

The Symetric [sic] algorithm initiates load balancing when both overloaded and under loaded. This algorithm would have both a T-high threshold and a T-low threshold. When a workload goes above the high threshold, the processor acts as a sender. It will act as a receiver when the workload drops below the low threshold. The proposed PSI Load Balancing Algorithm in a general improvement to the Symetric algorithm [3].

1.1.4 Solving Complex Problems: Workflow

Some problems are so complex that they cannot be modeled by a single job even on a grid system. In order to execute multiple jobs a workflow must be defined. A workflow is a bundle of jobs that comprise a larger application [25]. From the P-GRADE Portal Documentation, "technically a Workflow is a directed graph where each node has a computing resource and a program (job) to be launched on that resource; further the edges of the graph are the 'information pipelines' (streams) which connect the input and output points of the individual jobs" [P-GRADE Portal Documentation].

1.2 The Hungarian Grid

1.2.1 About SZTAKI

MTA SZTAKI is an acronym that stands for "Magyar Tudományos Akadémia, Számítástechnikai és Automatizálási Kutató Intézet" or in English, "Hungarian Sciences Academy, Computer and Automation Research Institute." This institute is based in Budapest, Hungary and has many departments split into two divisions: development and research. The division where the work for this project was done was the Laboratory of Parallel and Distributed Systems (LPDS). LPDS has a major role in the research of cluster and grid technologies in Hungary. LPDS is a member of many important organizations and projects including the European DataGrid Project and the European GridLab Project. The most notable products from LPDS are the Parallel Grid Run-time and Application Development Environment (P-GRADE), the P-GRADE Grid Portal, and the Mercury Grid Monitor [28].

1.2.2 The Grid

The Hungarian Supercomputing GRID (Szupergrid or HUNGIRD) is Hungary's solution for High Performance Computing. It connects supercomputers and machine clusters from separate organizations to create a unified system of resources. These resources are used to meet the ever-increasing demand from research scientists. The HUNGRID project interacts with other Grid projects such as the Hungarian DemoGrid, the European SEE-GRID and the Condor project, giving it a worldwide presence. [31] [5]

The HUNGRID has a physical architecture that is composed of over 300 processors with computing sites all over Hungary. All of these sites are interconnected with a 2.5 Gb/s Internet backbone. HUNGRID is a multi-layered system. The bottom execution layer uses Condor and Globus technology to execute jobs across parallel grid systems. The upper application layer uses SZTAKI's P-GRADE (Parallel Grid Run-time and Development Environment). This program provides tools for easy creation and deployment of parallel programs. [28]

1.2.3 Technologies Used

The majority of this project is in the form of Java Servlets, Portlets, and JavaServer Pages. These technologies run as web services and require Tomcat. Tomcat is part of Apache's Jakarta Project, a collection of open source Java solutions, and is a Servlet container. It uses the official Reference Implementation for the Java Servlet and JavaServer Page technologies and is generally synonymous with Java web applications. For this project we used Tomcat 4.1.31 which implements Servlet 2.3 and JavaServer Pages 1.2. [1]

Servlets are a Java platform technology developed by Sun Microsystems. They are mainly used as interactive web applications and have full access to the entire family of Java APIs as well as access to HTTP-specific calls. Servlets are platform-independent server-side modules. [16]

The JavaServer Page (JSP) technology is an extension of the Servlet technology. Just like Servlets, JSPs are also platform-independent and run on a web server. They enable developers to create dynamic web pages by integrating Java and HTML code. [16]

Portlets are also a Java platform technology but to understand them one must understand what a Java Portal is. From [33] "a portal can be best described as a web site that acts as a 'point of entry' or a gate to a larger system." One of the most notable features of a portal, which is really its essence, is content aggregation. This refers to the ability of a portal to combine content from separate sources into one single view. Portals also allow for personalization, meaning that the content and services it offers can be customized for each user. And finally portals allow a user to sign on once and then access many resources without having to re-authenticate.

Portlets are modular portal components. With the JSR168 specification they became standardized. Portlets are similar to Servlets in many ways, but are fundamentally different. Servlets represent all the content the user needs (usually it takes up the whole viewable window) whereas Portlets represent one part of a larger collection of services and content. In this way Portlets cannot be used to create an entire web page (they are even forbidden from creating certain tags such as 'body' and 'html') and users cannot access them directly through a URL as they could with Servlets. There are many other differences between the two, but what is important to know is that Portlets represent individual services and content and are aggregated into a Portal. [33]

1.2.4 P-GRADE Grid Portal

Programming for Grid systems requires intimate knowledge of many different diverse APIs for programming languages such as MPI and PVM as well as knowledge of grid tools such as Condor and Globus. Also, complex workflows that contain many singular parallel jobs become difficult to achieve. Scientists and other likely users of Grid systems probably will not have the needed programming background to achieve this. The P-Grade portal aims to allow easy creation, execution, and monitoring of these complex workflows.

The P-Grade portal has a graphical programming environment for creating workflows [19]. Workflows are a collection of sequential and parallel jobs that share input and output files. Independent branches of a workflow can be executed simultaneously on several Grid sites [25]. The graphical environment frees the programmer from the complexity of Grid middleware like Globus.

Execution in the P-Grade system hides the heterogeneous Grid, creating one virtual homogenous resource. P-Grade fully integrates with both Condor and Globus Grids and can intercommunicate between them both. The Globus toolkit is used between all layers of P-Grade in order to make the system highly portable between different Grids.

Other tools available through the P-Grade portal are the certificate manager and the visualization/monitoring tools. The Grid certificate manager maintains a list of permissions for using remote Grids. The visualization/monitoring tools allow a user to watch the status of a workflow as it executes. They can see which jobs are being executed, their status, as well as download any log and output files. The PROVE visualization tool will display information about where a job was executed, where it migrated, and which processes it communicated with. This is shown in a space-time diagram.

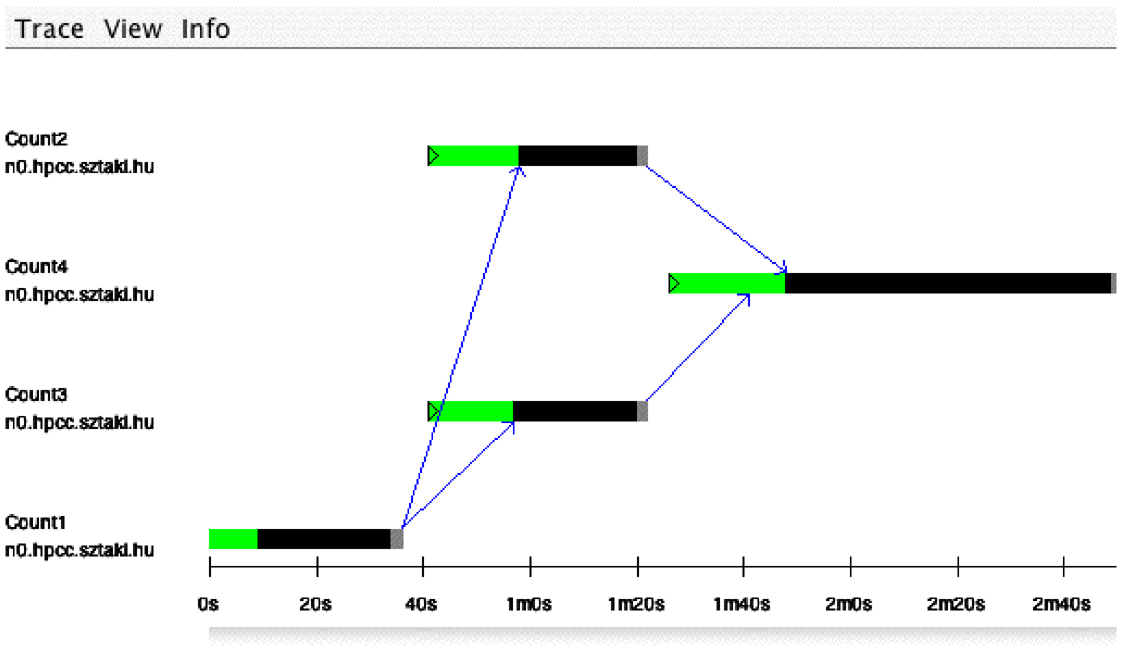


Figure 1 A Space-Time visualization

1.2.5 PROVE Trace Visualization Tool

PROVE is a visualization tool integrated into the P-GRADE program development environment [32]. Its role is to create data visualizations by parsing the trace files that GRM creates. GRM is a monitoring infrastructure that performs trace collection. While currently PROVE relies on the trace files that GRM creates it is possible to run PROVE as a Grid service thereby eliminating its dependency on GRM.

1.2.6 Sample Applications

P-GRADE is more than just an experiment to show what could be done with the power of parallel computing; many applications have already been created and are running with the tools provided by P-GRADE. The most notable of these are in the fields of chemistry, engineering, and meteorology. In general these applications were originally sequential programs that stood to benefit from a parallel environment. P-GRADE was used to create parallel applications for simulating traffic situations, monitoring air pollution, and predicting ultra-short range weather phenomena.

A specific application of P-GRADE is the MadCity traffic simulator. P-GRADE was used on a cluster at the University of Westminster to create a parallel version of MadCity. This simulator consists of two parts: the GRaphical Visualizer (GRV) and the SIMulator (SIM). The GRV helps to design a possible road network while the SIM creates trace files which can be loaded into the GRV. These trace files show the behavior of cars in various intersections. The SIM was implemented on P-GRADE and run as a parallel application; when the simulation starts a file describing the road network is sent to nodes on the cluster and when the simulation ends a trace file is created. A single processor is not suitable for real-time traffic simulations because these situations are so complex. Since it is necessary to run the simulation in a window of only a few minutes, the simulation must be broken down into smaller parts and distributed among many processors. For a situation involving one city, roads are partitioned into segments and each node works on computing the results of different segments. For situations involving many cities, separate clusters can be used to compute the results of a different city and even monitor the behavior of cars between cities. Simulations were run successfully at the University of Westminster with 300,000 cars on one, four, eight, and sixteen nodes. It was proven that the simulation ran fast on a cluster because the amount of computation time spent by each node far outweighed the amount of cycles spent on sending and receiving messages. A larger simulation was run successfully on the Grid which represented multiple cities but the results of that simulation are not yet available [12].

1.3 Parameter Study

1.3.1 What is a Parameter Study?

A parameter study (or "parametric study") is a useful tool designed to assist in a set of similar calculations. More specifically a parameter study is one analysis performed many times on a model where certain properties are varied over a given range. There are many practical applications for parameter studies most notably in chemistry, physics, and engineering. A prime example would be Monte Carlo calculations, which can be found in many disciplines. The "Monte Carlo" method is a very broad way of describing a method of using random numbers in experiments. Monte Carlo applications allow the creation of models for systems which are far to complex to analyze. Fundamentally, to perform such a calculation is to analyze a number of random configurations from a large set and then use the results to describe the entire system [36].

Currently many members of the scientific community around the world are exploiting their parallel systems for use in Monte Carlo applications. Such applications may not be data-intensive but they are very computational-intensive. They usually involve small data sets but myriad computations. A parallel system is very beneficial to anyone wanting to run such calculations because the work can easily be divided among processors [23]. High-profile examples of the Monte Carlo method which may benefit from a grid-based parameter study would include in regards to the financial community: generating stochastic interest rates and assessing the risk and performance of assets and liabilities [13]; in the scientific community: nuclear medicine [22]; and in the engineering community for fault anticipation and analysis [2].

1.3.2 Sample Parameter Study: Computing the Mandelbrot Set

To better understand parameter studies, consider the Julia and Mandelbrot sets. These are two very famous fractals that can be calculated with a simple algorithm. The Julia Set is named for Gaston Julia, a French 20th century mathematician. His work with iterating polynomials and rational functions was very important and groundbreaking in the world of mathematics but he was not given the recognition he deserved until his work was referenced by Benoit Mandelbrot. In 1975 Beniot Mandelbrot, a Polish-born French mathematician, published *Les objets fractals, forme, hasard et dimension*. In this work he coined the word "fractal" and discussed these structures [4].

The Julia and Mandelbrot sets are strikingly alike and are calculated in a very similar fashion. A Julia set is calculated by iterating the function:

$$z_n = z_{n-1}*z_{n-1} + c$$

In order to calculate the set a few parameters must be defined. First is c, a chosen seed. This seed can be a real number or a complex number, but the most interesting results (and those that are usually associated with Julia sets) are found when c is a complex number. The next parameter is z, representing the coordinates of a point in the complex plane. The complex plane can be represented graphically as a set of axes where the x-axis represents the real part of a complex number and the y-axis represents the imaginary part of a complex number.

To clarify, consider the simple case of:

c = 0 + 1i

z_0	$z_1 = z_0 * z_0 + c$	$z_2 = z_1 * z_1 + c$	$z_3 = z_2 * z_2 + c$	$z_4 = z_3 * z_3 + c$	$z_5 = z_4 * z_4 + c$
0 + 0i	0 + 1i	-1 + 1i	0 + 1i	-1 + 1i	0 + 1i
1 + 1i	0 + 3i	-9 + 1i	80 – 17i	6111 – 2719i	A big number

First look at the case were z_0 is a point in the complex plane at the coordinates (0,0). As the function $z_1 = z_0 * z_0 + c$ is iterated it can be seen that the value for z_n simply oscillates between i and -1 + i. Although only 5 iterations are shown here it can be safely assumed that as the number of iterations approaches infinity the value of z_n will remain bounded. It is said that the value at this point converges.

Now consider the second case where z_0 represents a point on the complex plane at (1,1). It is very obvious after 5 iterations that the value of z_n will continue to increase as the number of iterations reaches infinity, so the value at this point does not converge.

Not all situations are this cut and dry though. There may be values of z for which it is very difficult to determine convergence. Due to this, a radius of convergence and a maximum number of iterations need to be defined. The maximum number of iterations is simply the most amount of times the function should be iterated before we are forced to decide if the point converges or not. In the previous example, 5 was chosen as our maximum number of iterations.

The "radius of convergence" is slightly more abstract. It defines the size of a circle centered on point z_0. In the course of iterating the function, if z_n ever found to have left the circle of convergence then it is assumed that z_0 does not converge. On the other hand, if the maximum number of iterations is reached and the value of z_n is still within the circle of convergence then it is assumed that z_0 does converge. If we take the radius of convergence to be 2 in the above example then we see that after 5 iterations the first value of z stays within the circle of convergence but after a mere 2 iterations it is obvious that the second value of z escapes the circle. The rate at which z leaves the circle is called its "escape velocity." [17]

Here are several images of Julia sets with their corresponding c-values. The way to color a Julia set is to choose a separate color for each escape velocity. Here it can be seen that the points that converge are colored black while other points are colored according to their escape velocities with red escaping the fastest [6].

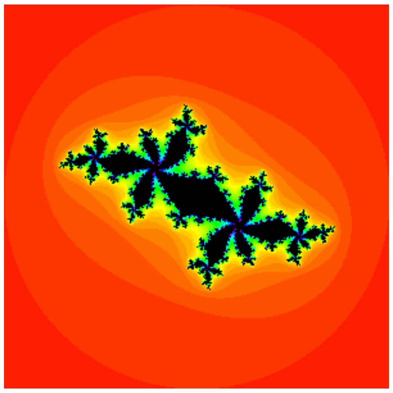


Figure 2: $c = -0.52 + 0.57i$

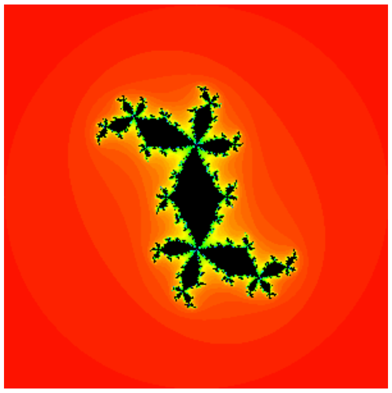


Figure 3: $c = 0.295 + 0.55i$

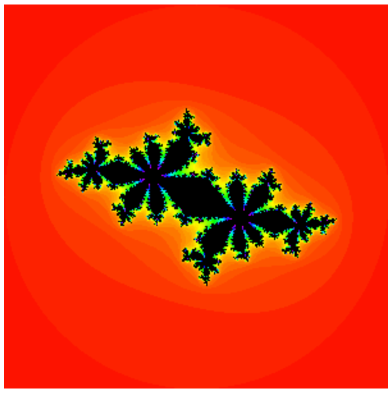


Figure 4: $c = -0.624 + 0.435i$

Now, to compute the Mandelbrot set a very similar method is used. To calculate the Mandelbrot set this formula is employed:

$z_n = z_{n-1} * z_{n-1} + c$

Where $Z_0 = c$

The major difference between the calculating the Mandelbrot set and Julia sets is that in the Mandelbrot set c is the point in question on the complex plane. So with a Julia set the point in question was chosen for z, but with the Mandelbrot set that point is c. Notice how this changes the calculation:

c	$z_0 = c$	$z_1 = z_0*z_0 + c$	$z_2 = z_1*z_1 + c$	$z_3 = z_2*z_2 + c$	$z_4 = z_3*z_3 + c$
0 + 0i	0 + 0i	0 + 0i	0 + 0i	0 + 0i	0 + 0i
1 + 1i	1 + 1i	1 + 3i	-7 + 7i	1 + 99i	-9799 + 199i

The same rules for convergence apply as with a Julia set, so the Mandelbrot set can be graphically represented in the same manner. It is important to note that a Julia set defines a set of points on the complex plane and the fate of each of those points with regards to a certain seed. On the other hand, the Mandelbrot set is actually a set of all Julia sets. All of the following images are of the Mandelbrot set and were created by an application developed by this project.

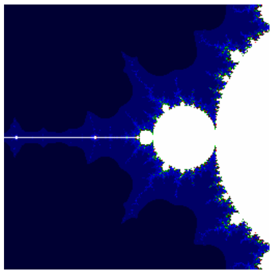


Figure 4: Centered on -1.4 + 0i

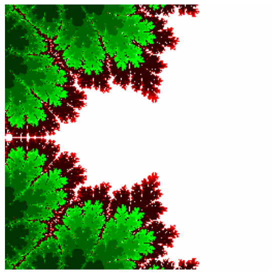


Figure 5: Centered on -1.4 + 0i zoomed in 4 times further

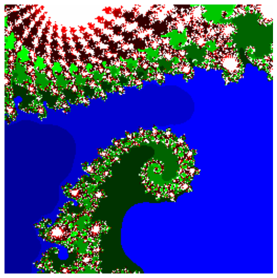


Figure 6: Centered on -.745429 + 0.113010i

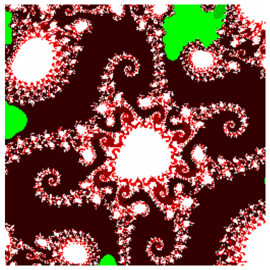


Figure 7: Centered on -.745429 + 0.113010i zoomed in 5 times further

Computing the Mandelbrot set itself is a parameter study because one function is evaluated with its parameter "c" varied over a set of values (the domain of the set). There are a number of other ways the Mandelbrot set could be used to create a parameter study. For example a parameter study could be created that would compute the set at a finer and finer resolution. Such a study would calculate the set for all values in a given range at a specific interval, and then it would calculate a subset of that range with a smaller interval, and then calculate a subset of that range at an even smaller interval, etc. This would be equivalent to "zooming in" on a section of the set. This method was used for the parameter study created for this project.

2 The Problem

2.1 Existing System

2.1.1 Fully Integrated Portal

The Laboratory of Parallel and Distributed Computing at SZTAKI has developed a web-based application called the P-GRADE (Parallel Grid Run-time and Application Development Environment). The P-GRADE web portal allows easy creation and execution of complex parallel workflows in a Grid environment [20]. The portal is a thin-ware application that can be accessed from any web browser.

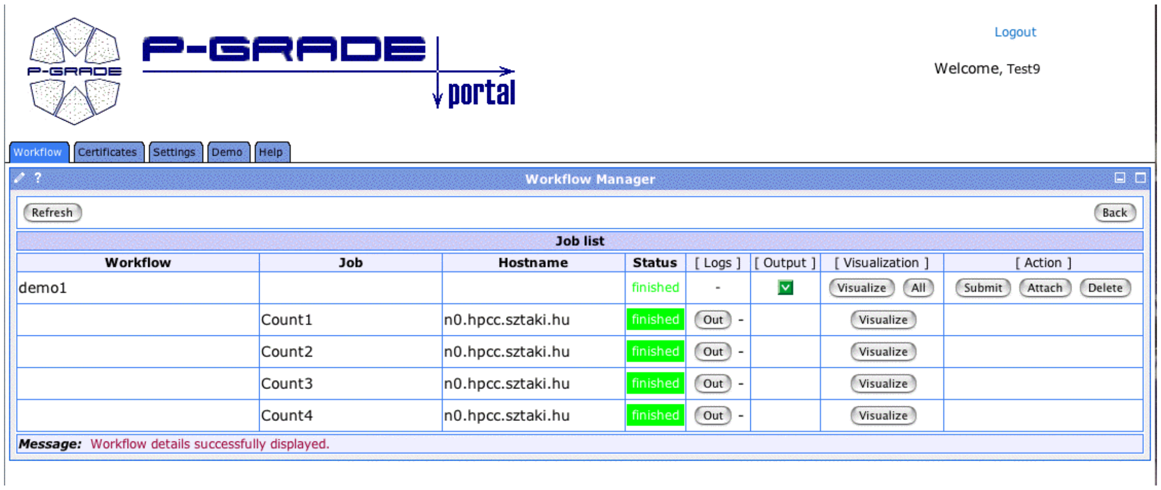
A workflow is a series of interconnected jobs that can be separately executed. A job is an executable application that could contain any number on input files and output files. This paper will call these files input and output ports from here on out. The output ports of one job can be linked as the input port of another job. With these links, a network of jobs can be created. This is a workflow. Independent jobs within the workflow can be executed in parallel. This guarantees that when executed in a Grid system, the workflow is executed in an efficient manner.

There are two main components of the PGRADE portal: the client-side graphical editor and the server-side manager. The client-side application is used for editing, creating and uploading workflows to the Grid. This application is called the Workflow Editor. The server-side application is used as a workflow repository and manager and it is called the Workflow Manager.

2.1.2 Workflow Manager

The Workflow Manager has a number of responsibilities. It must maintain a repository of every user's workflows. From this repository it must allow a user to view the status of all their workflows. Users must be able to get detailed information on specific workflows and specific jobs within a workflow. The Workflow Manager is responsible for visualizing workflow information in a time-space diagram. A very important element of the Workflow Manager is its use of a Servlet. This Servlet maintains communication between the Workflow Manager and the Workflow Editor.

The Workflow Manager keeps all the workflows on the server with the use of a workflow repository. This repository is a combination of files in the server's file system and Java Objects. The file system contains all of the executables and input files are stored and it is the responsibility of the Workflow Editor to upload these files to the server. These Java objects in the repository keep track of information such as which users have which workflows. These objects store information about individual workflows and the jobs they contain, for example which Grid the workflow will execute on and which resource the jobs are assigned to.



Job list

Workflow	Job	Hostname	Status	[Logs]	[Output]	[Visualization]	[Action]
demo1			finished	-	✓	Visualize All	Submit Attach Delete
	Count1	n0.hpcc.sztaki.hu	finished	Out -		Visualize	
	Count2	n0.hpcc.sztaki.hu	finished	Out -		Visualize	
	Count3	n0.hpcc.sztaki.hu	finished	Out -		Visualize	
	Count4	n0.hpcc.sztaki.hu	finished	Out -		Visualize	

Message: Workflow details successfully displayed.

Figure 8 The Workflow Manager

The main purpose of the Workflow Manager is to allow the user to maintain a list of workflows. The user is able to submit, abort and monitor individual workflows and jobs. All this is done through a web interface. In order to submit a workflow (schedule it to be run) the input files and the executable code must be available to the portal. Also, resources must be defined for each job so that the portal knows where to submit the job. These are all done inside the client-side Workflow Editor.

Monitoring a workflow can be down in multiple ways. The WM will show current progress status to the user. This status includes the state of the workflow (either scheduled, running, finished, or error) and information about the outputs. A detailed menu will give similar information about individual jobs. Other information found at the details menu includes the hostname for a running job, a link for a job's log files, its output, and a button for visualization.

Visualization is performed by PROVE, the trace visualization collection tool. It collects information about each job as they run. This information is displayed as a time-space graph. Information provided by the trace file includes where a job executed, how long it executed for and what connections it made to other hosts. A detailed visualization can be made for each job that only displays processing time information.

2.1.3 Workflow Editor

The Workflow Editor is used for editing and creating workflows to be used in the P-GRADE portal. An application using the Grid must be started from the Workflow Manager, which gives the client both the Java executable and the starting parameters for the Editor. If the editor is started from a specific workflow then the WM sends that workflow to the client. Otherwise the user has to open a workflow from the WM's workflow repository, or create a new one.

The Workflow Editor represents a workflow as a visual graph. Each node of the graph is a job. Each job has ports, which can represent input or output files. Output ports can connect to input ports linking jobs. Editing the workflow consists of binding information to each port and job.

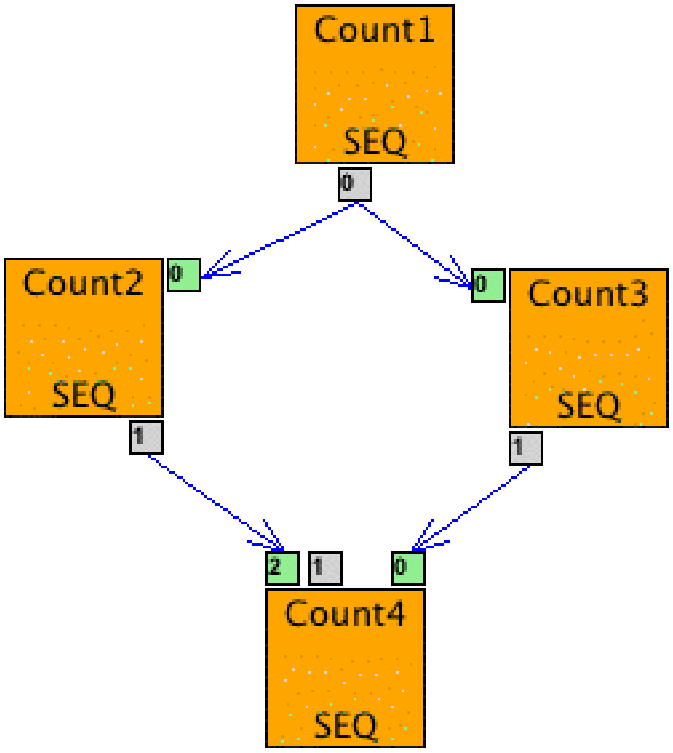


Figure 9 A Workflow from inside the Editor

The port configuration needs to include information about what input file is attached to the port and whether it is an output or an input port. A job configuration includes job type, a path to its executable, the resource where it is to be run, and command line attributes to be executed at runtime. Once all the jobs are configured and linked together a workflow is complete. When the user saves the workflow, it is added to the workflow repository. Any needed files (executables and input files) are uploaded to the portal server.

2.1.4 Need for a Parameter Study

As previously described, a parameter study is a single analysis performed multiple times on a model. At each step in the study the input parameters are varied in a systematic way. This is useful in analyzing situations over a wide range of possibilities. While a parameter study is an important and useful tool in many areas of research it is also very expensive in regards to the amount of computation it requires; therefore it is an obvious candidate for implementation into a grid system. Using the processing power that a grid offers is the only way to run a large parameter study in a reasonable amount of time. Since a parameter study is composed of numerous calculations, which, although they are related on some level, do not rely on each other's results it seems natural that a parameter study could be run on a grid simply by dividing up the calculations and distributing them among available processors. The processors could then return the results of their analysis to one central program which would compile the results into some sort of usable form. And in fact what was just described is exactly the goal of this project.

More specifically a parameter study tool (PStudy) was integrated into P-GRADE. This entailed numerous additions to the portal, most notably front-end changes such as new Portlets for managing, monitoring, and executing parameter studies; a Java WebStart application which provides a graphical tool for creating, updating, and uploading a parameter study; and backend changes such as a Servlet for communicating the operations and results of the parameter study, a repository for parameter studies, and a scheduler for submission to the grid. And finally P-GRADE an extension was created in the form of a Portlet, which allows users to visualize the results of their parameter study both as it is running and when it is finished.

2.1.5 Requirements

2.2 Goal System

2.2.1 Extended Portal for Parameter Studies

This project involved extending the existing portal code to allow Parameter Study jobs. This extension involved creating new Portlets for a Parameter Study Manager and Editor. A Parameter Study job repository was developed based on the workflow repository. A scheduler was developed in order to handle resource management of the individual Parameter Study jobs. And finally, the workflow Servlet was extended for communication between the manager and the editor.

2.2.2 New Portlets

Just as workflows need a Workflow Manager, parameter studies need a Parameter Study Manager. This portlet's main job is to organize a user's Parameter Studies. It provides a list of all of their Parameter Studies to the user as well as useful information about them. This information includes the overall status of the parameter study such as the number of jobs submitted and run and those which have failed or finished, and also detailed information on specific job instances. This information is pertinent to the user because he or she is allowed to submit subsets of the entire parameter study. If at least one study is running, the status of the entire study will be marked as running. If a majority of the jobs have occurred errors, then the status of the entire study will be marked as error.

The Portlet also provides a detailed view which allows the user to look more closely at the parameter study by each parameter key. Here a user can view the same information as above, but also perform actions on specific value ranges. A user can choose a subset of the parameter study in the form of a limited range of one or more parameter and submit it for execution. The user can also abort, resubmit, and download output from subsets of the parameter study. It was necessary when designing the algorithms that worked on subsets to be conscious of the fact that there would be no guarantee that the action chosen by the user would be appropriate for every job in that range. Most likely the user would only expect the chosen action to be performed on jobs for which that action made sense. For example, only jobs that have been submitted and are currently running will be aborted when the abort button is pressed.

2.2.3 Extension to Workflow Editor

In order to best extend the Workflow Editor we simplified the workflow into a single parameter study job. This job represents a full set of jobs, all which share an executable. This job has a required input port that is bound to an input file that contains parameters in variable form. These variables represent values in the set bounded by the parameter study. This port is called the Special Parameter Study Input Port.

To edit a parameter study job, you can either edit its ports or the job itself. The input port configuration is used in order to set up the parameters and their values. Here the user attaches the input file that contains the variable parameters. The user needs to tell the editor how the variables are delimited inside the file. The editor then composes a list of variables that it found in the parameter file. The user can edit each of these keys, attaching values to the variables. These values can either be characters, integers, or real numbers. The user will have further control by supplying the values as a list, as a range, or as a random set.

While editing the job, the user must give the path of the executable that will be run. The user is be able to set options to restrict the number of parallel processors used when executing this job. The user is also be able to choose in which order parameters are changed while executing a range of parameters. In order to simplify the problem, our extension removes the option to monitor the job while running. Tracing these jobs and storing their trace-files could prove to be far too expensive for parameter studies.

2.2.4 Extension of Back End

The back-end will was extended upon in order to handle parameter study jobs. A new Servlet was created in order to handle communication between the Parameter Study Manager and the Parameter Study Editor. Typical communication between the Parameter Study Manager and the Editor involve file transfers. Editing involves the Editor downloading a parameter study file from the server. Editing is done client-side, and when the user is done the file must be transferred back along with any dependent input files and executables. Also the Servlet is given status information of a loaded parameter study back to the editor.

The workflow repository was extended to store parameter study jobs on the server. This repository keeps track of every user's list of parameter studies. Each parameter study has a list of the parameters and their possible values stored and grid information on where the job will be executed. All required files for execution are stored on the server's file system.

Handling the execution of the parameter study is performed differently than for normal jobs. Each parameter study represents a number of individual job executions. The number of possible instances of a parameter study can be quite high. It is difficult and inefficient to assign resources to these executions before they are scheduled. Therefore, the parameter study system has a process that monitors job scheduling. When a job is scheduled it dynamically assigns a resource for it based on a static list. This is a form of load balancing used by our parameter study module to efficiently handle parameter study jobs.

2.2.5 Visualization Portlet

In addition to all these changes that were made to the front end and backend, the P-GRADE Portal required a visualization Portlet. This Portlet is not for the sophisticated visualization of scientific data that the parameter study will return, but merely a structured preview of the study at runtime. Actually, this

Portlet is not limited to just parameter studies but it may also be used to assist other generic workflow applications.

In the case of generic workflow applications the Portlet displays the results of one given workflow as selected by the user. The results are processed on the server-side and are completely transparent to the client. The Portlet displays a preview of the results from each job as they are available in the form of text, xml, or image depending on which is the most relevant for that data.

In the more complicated case of a parameter study application the user may select one or two parameters and their ranges. Just as with a generic workflow application a preview of the data will be displayed as text, xml, or an image as it becomes available. If one parameter is selected then all output files will be displayed arranged in a vertical line organized by parameter value. If the user selects two parameters then he or she will be required to select one permanent output file and that file will be displayed in a two dimensional array organized by parameter. In all cases the user should be able to set the size of visualization blocks as well as fonts used and maximum length of text.

2.2.6 Applications

Finally, an application was developed to showcase the Parameter Study Module that was integrated into the P-GRADE Portal. This application is one that calculates the Mandelbrot set at different coordinates and increasing resolutions, effectively “zooming in” on specific parts of the set. This application was chosen because its results are familiar and developing it is relatively simple. The main purpose of the Mandelbrot application is to determine exactly how the application should interact with the changes and additions made to the P-GRADE Portal.

3 Implementation

Through the front-end system, the user can create, edit and run PStudy Jobs. The current Portal implementation treats job submissions as part of a workflow. These workflows are represented both as Java objects on the Portal's Java Virtual Machine (JVM) as well as files contained in the server's file system. The Java object contains dynamic information such as grid name, workflow status, and information about the workflows individual jobs. Concrete data such as the workflow submit files are stored on the file system. Executables and their respected input and output files are also maintained on the file system.

Our PStudy module extends this workflow implementation. We use a similar Java and file system combination to serve as a repository. We implemented a high-level scheduler that watches job submission. This scheduler handles resource management and load balancing. The new scheduler communicates with the existing low-level Execution Layer as well. We also extended the existing communication between the client and the server with a PStudy Servlet and a PStudy Portlet.

3.1 Back-end

The implementation of the back-end of the PStudy system uses a combination of Java objects and stored files as a repository. Each parameter study is an abstract entity that represents a set of possible concrete jobs. Therefore most of the information stored for a parameter study only needs to be skeleton data. This data will remain generalized for all possible jobs instances of a parameter study until the moment an instance is submitted.

We stored the concrete information for PStudyJobs (instances of parameter studies) in the server's file system. This includes executables and input files that are shared between all instances of a PStudyJob. Output files are also stored in the file system but they must be handled with care. All instances of a PStudyJob share output files, but each of these files will be unique. These files must be stored in unique directory paths. Also, skeleton files are stored in the file structure that contains data that is shared among the entire parameter study.

Java objects are used to store abstract data that changes between PStudyJob instances. Information must be stored in the java objects on how to produce a full file from the skeleton files. This includes a user ID, the PStudyJob's name, the variables (keys) that make up the Special Input File, and the key's values. It is important to note that each instance of a PStudyJob can be uniquely identified by the index of these values. For example, consider a parameter study that has three keys and each key has 3 possible values. All possible combinations of values can be uniquely represented by each the indexes {0,0,0} to {2,2,2}.

3.2 Job Repository

Each user in the system has a unique Job Repository. A singleton Java object that exists in the server's JVM represents this repository. Each Job Repository contains a list of parameter studies available to that user. It also contains interface functions for performing operations on these parameter studies that are used from the front end. Each PStudyJob is an abstract representation of an individual job submission. It contains a user id, grid name of where it will run, and a skeleton input file as well as keys and values for the input file. PStudyJobs are responsible for maintaining a repository of individual concrete job submissions that the user has made. These concrete Job submissions contain a unique index and the jobs current state (either scheduled, running, finished, error, or aborted). For reliability, the contents of the Job Repository are written to disk when it is changed. This allows the repository to be recovered after a system crash.

3.3 High Level PStudy Job Scheduler

In P-GRADE Portal's workflow, jobs can be statically allocated to resources before submission. For a PStudyJob, resource allocation is not a trivial matter. A PStudyJob could represent hundreds of individual runs. To manually allocate resources to each instance would be tedious and time consuming. Also, these jobs will be run on a grid, which makes static allocation inefficient. Faster processors will finish their jobs quickly, and wait idle while slower resources finish. Dynamic allocation is the ideal solution.

The high level scheduler sits between the Job Repository and the low-level Execution Layer and performs dynamic resource allocation for PStudyJob instances. It operates as a daemon thread that constantly monitors the Job Repository. It keeps a list of available resources and will use a fair load-balancing algorithm to allocate resources to submitted jobs.

There are three events that the Scheduler listens for. The scheduler will perform actions when a PStudyJob instance is submitted by the user, aborted by the user, or the Execution Layer reports the termination of a running job (with its termination status).

When a job is submitted, the Scheduler must assign the job a resource by the load balancer. If there are no available resources, then the job must wait until one becomes available. Once assigned to a resource, the scheduler creates a temp directory used for running the job. Copies of the skeleton data from the Job Repository are received and filled with concrete information. The scheduler creates the Special Parameter Input file and workflow file. These files are needed for the Execution Layer for submitting the job. The job then gets submitted to the execution

When a user aborts a job, the scheduler is responsible canceling the job. The Execution Layer gives the scheduler the job’s current status. If the job is running, it must be stopped. The resource attached to the job must be freed and returned to the scheduler if aborted. When the Execution Layer reports the termination of a job, the scheduler must mark that status change in the Job Repository. It must also return the freed resource to the resource pool.

3.4 Load Balancing Algorithm

Our load-balancing algorithm uses receiver-initiated dynamic allocation. This means that resources will receive jobs when under-loaded. The system follows a master-slave topology, where the High Level Scheduler acts as the master and the grid resources are its slaves. The scheduler generates load-status information with status messages sent to available resources. When a resource fails to respond past the max time duration, it is considered over-loaded and unavailable. This time duration is considered the overload threshold for resources.

Available resources are stored in a synchronized queue. This is our resource pool. When resources are taken from the pool, they are taken from the front of the queue. Freed resources are placed at the end of the queue. The scheduler does not have perfect load information about every resource. This is why this 'round robin' method of accessing resources is used. The queue implementation of the resource pool provides a best-effort, fair algorithm without perfect knowledge.

When getting resources from the pool, the Scheduler will test its availability. An unavailable host is either dead or overloaded. The scheduler will place overloaded resources at the end of the queue, to be checked again later.

If the resource is available, it is attached to a submitted job. This process is continued until an available resource is found, or a maximum attempt limit is reached.

3.5 Servlet

Communication between the Job Repository and the client side PStudy Editor is done with the PStudy Servlet. It both serves the purpose of providing the editor PStudies contained within the Repository as well as giving the Job Repository PStudies created and edited by the client. The Servlet was implemented with Java's Servlet technology. It accepts http messages sent by the client and is served by Apache Tomcat.

3.6 System Interactions

The user accesses the PStudy system through the PStudy Portal. This thin-client GUI allows the user to manage their Parameter Studies. All interaction between the Portal and the back-end are through the PStudyList. The interface between the PStudyList and the Job Repository provides the Portal the necessary access to allow the user to manage their parameter studies. Here a user can get the status of jobs, and submit jobs by index or by ranges of indexes. They can also download output from jobs that have run.

The PStudyServlet provides all interaction between the client editor and the Job Repository. This interface allows the user to save PStudies and to download them to be edited. When a PStudy is saved, the editor must send to the Servlet all the information needed to create its entry in the repository. The client must also upload all files required for the study to be able to execute. This includes common input files, the executable, and the skeleton of the special Parameter Study input port.

Once a job is uploaded and saved, the user can submit it through the Portal. The HighLevelScheduler watches for submission from the Job Repository. The Scheduler maintains the communication between the Job Repository and the low-level Execution Layer. The Scheduler's link to the Job Repository is through the individual jobs themselves. The link to the Execution Layer is maintained with Linux shell scripts and shell commands. When a job is submitted, the Scheduler is notified and begins to watch this job. When a resource becomes available, the Scheduler will allocate it to a job. It will then submit it to the Execution Layer. The Execution Layer will communicate run status to the Scheduler by writing status files that the scheduler will read. When a job's status changes, the Scheduler reflects those changes in the Job Repository.

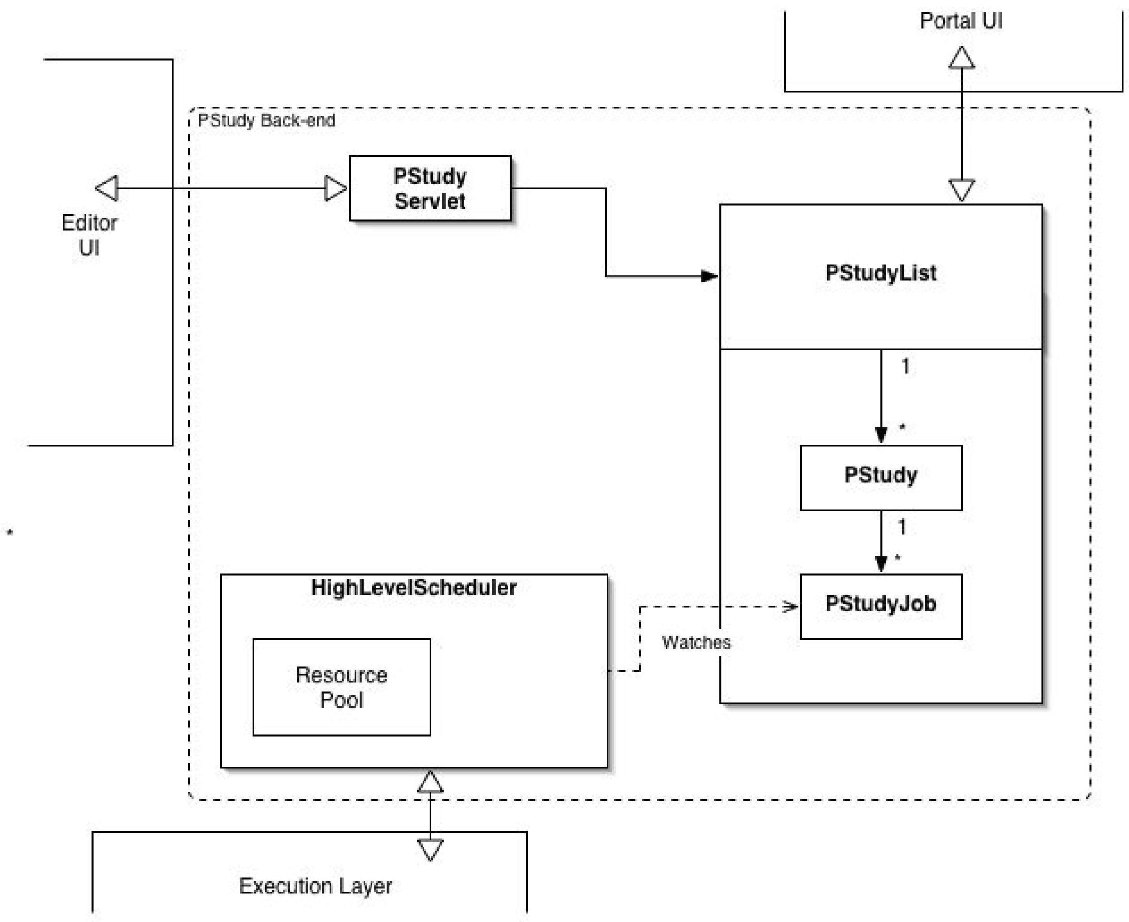


Figure 10 High Level Architecture

While jobs are running, the user will be able to observe these changes in the Portal by getting the status of both PStudies and their individual jobs. They can also abort and resubmit jobs. These actions are, of course, observed and executed by the HighLevelScheduler. The user can visualize the results of finished jobs through the portal as well as download directly the output files. Output files are stored in an output directory that represents the job's index that produced the result. This maintains that the unique output files from jobs can be easily separated. The visualization tab is especially important in parameter studies, as it allows users to quickly compare data, and make parameter changes accordingly.

3.7 Integration – Back- end

The back-end system is the 'focal-point' of the PStudy project. Every layer communicates with the back-end system. Therefore, integration between the back-end and the rest of the project was very important. Integration of this system began at our design phase. We purposely designed interfaces that accessed the back-end that addressed the needs of the front-end systems. These interfaces simplified the actions they represented and could easily be stubbed. This allowed the other sections of the project to fully integrate before the back-end was finished.

The interface that the PStudy Manager uses to access the Job Repository allows easy access to the current state of jobs, but offers no ability to add or edit PStudies. Users from the PStudy editor can upload files to the Job Repository and add new PStudies but cannot submit jobs to run. These well-designed interfaces made integration simple.

The Execution Layer is composed of Linux shell scripts. These scripts were designed to execute jobs structured in a workflow. In order to integrate with this lower level, the back-end creates a temporary working directory to run each job instance. In this temporary directory, the back-end creates an ordinary workflow out of the parameter study. We needed to create status, grid and resource files so that the Execution Layer could prepare Condor and Globus. The

Execution Layer communicates to upper layers by writing to files. Integration to this layer simply meant executing these shell scripts and reading the results from the file system.

3.8 Mandelbrot

In order to begin implementing this project a suitable application for showcasing and testing was necessary. In response to this need the group developed an application which computed the Mandelbrot set. The reader should already be familiar with the way the Mandelbrot set is calculated and how it may function as a parameter study. There were many motivations for choosing this as the study used to prove the usability and effectiveness of the parameter study module, namely because the output should be very familiar and easy to understand. Also, it would be easy to see how modifying the input parameters would affect the output, and finally because computing the Mandelbrot set as a parameter study serves a useful function. When graphing the set it is very difficult to find a point that is interesting when it is enlarged several times. Consider these two cases:

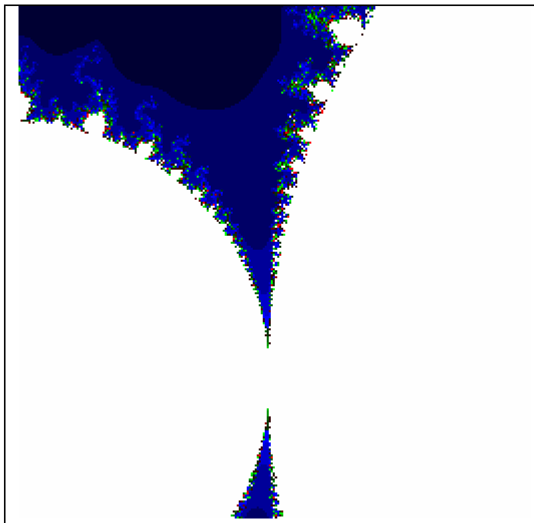


Figure 11: Centered on (-0.745429, 0.113010)

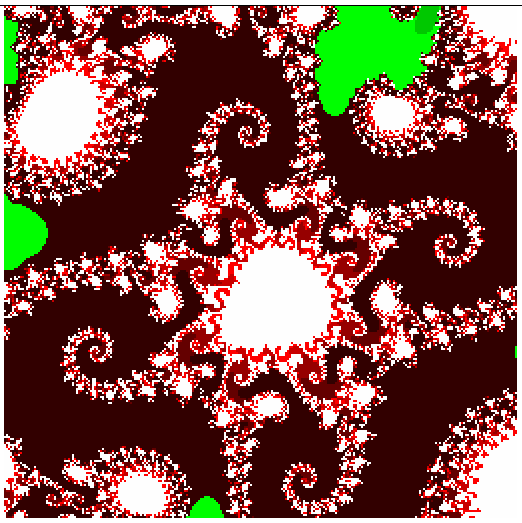


Figure 12: The same point magnified 5 times

And then consider these:

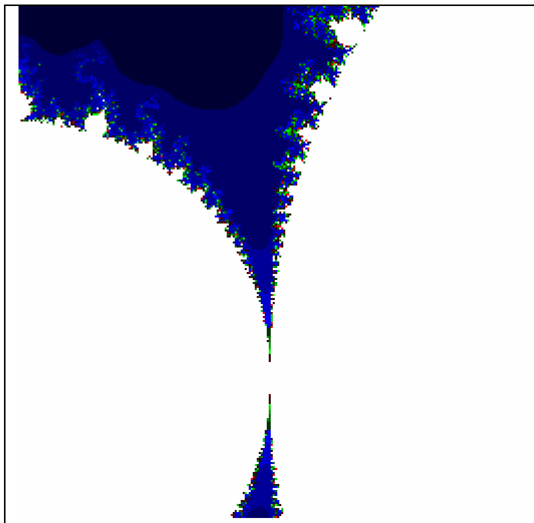


Figure 13: Centered on (-0.746429, 0.113010)

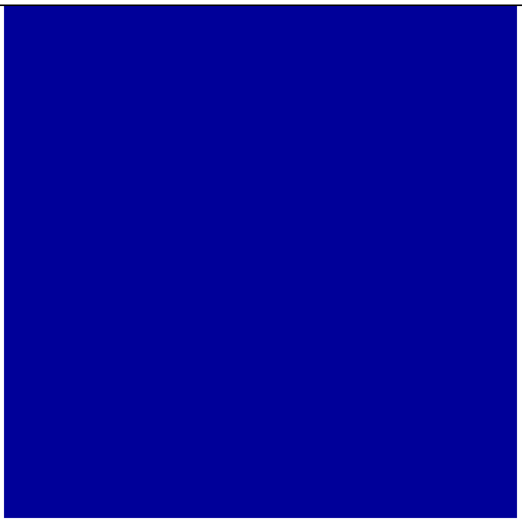


Figure 14: The same point magnified 5 times

The difference is that the first set is centered on (-0.745429, 0.113010) and the second set is centered on (-0.746429, 0.113010), a point which is only

0.001 units away! This illustrates just how difficult it is to find interesting regions of the Mandelbrot set at large magnifications. Creating a parameter study that generates images for every point in a given section of the Mandelbrot set may allow users to explore new areas of the set and generate images that no one has ever seen before.

The program (mandelbrot.c) that generates the images functions as such: first the user supplies an input file that contains four parameters. These parameters are: where to center the view on the x (real) axis, where to center the view on the y (imaginary) axis, desired magnification, and maximum iterations before deciding if a point converges or not. The reason for storing these parameters in an external text file is so that it would be trivial to generate numerous files programmatically for the parameter study. When it is executed, mandelbrot.c generates one bitmap image of the Mandelbrot set corresponding to the supplied parameters. The program also relies on a file called "256header" which contains header information for a 24-bit bitmap of size 256 pixels by 256 pixels. Although the program generates a single image each time it runs it would be simple to modify it so that it creates a series of bitmaps based on a parameter, thus acting as a sequential parameter study. As a matter of fact, the first version of this program did just that and its results were compared to the results of the parallel parameter study for testing purposes.

The Mandelbrot program was written in the C programming language. This was done in the hopes that it would be able to use the MPI library to distribute parts of its execution among resources in the Grid. Although it is trivial to create one small bitmap image on a single processor the idea was that by making the Mandelbrot program into a parallel application it would better model the actual applications that would be run on the Grid. Although this was never implemented in favor of other features, it could be done rather simply. Incidentally, resources and examples on how to create a Mandelbrot parallel application with MPI abound but they were not utilized for this project. There are several reasons that existing versions of a Mandelbrot program were not used. First is because they lacked parametric input from a text file. And second, all of these resources generate the image of the set by either using OpenGL or by using the MPE library which interfaces with X [27]. The desired action of this program was to create bitmap files, not create visualizations on the screen. Bitmaps were chosen because it is simple to create them pixel-by-pixel using C [21].

3.9 Visualization Portlet

The group created two Portlets for the Parameter Study Module for PGRADE Portal. The first was the PStudyPortlet which is an interface for submitting parameter studies. The second, which this team was directly responsible for, was the VisualizationPortlet. The reader should already be familiar with the purpose and overall idea of the VisualizationPortlet. It was created to provide the user with a structured preview of the results of a workflow of parameter study during the execution of the jobs. The purpose of this is to allow the user to abort the workflow or parameter study early if the results are not headed in the right direction.

Both Portlets were created in keeping with the style and structure of other Portlets in the PGRADE Portal. This means that they are accessible from the Portal via a row of tabs along the top of the view port. This tab leads the user to the Portlet which serves him or her JSPs. A JavaBean was also developed for each Portlet in keeping with the structure of other Portlets. This Bean handles many functions and stores state information for use in the JSPs.

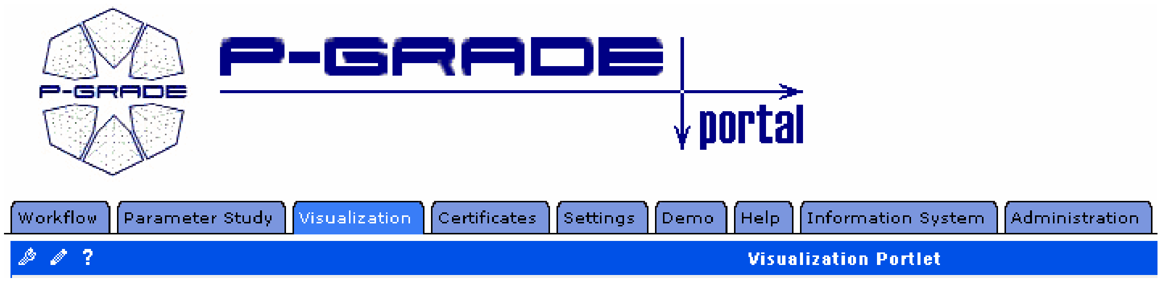
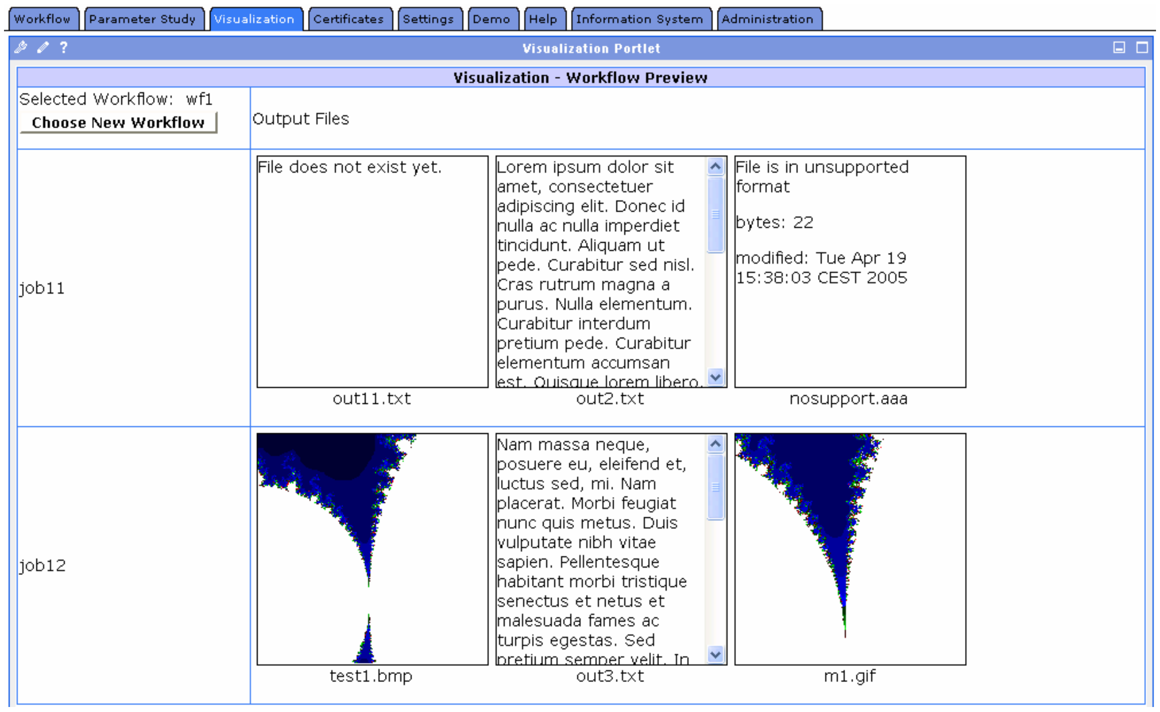


Figure 15: The new Visualization tab in PGRADE Portal

Specifically for the VisualizationPortlet, which is loaded when the user clicks the "Visualization" tab, the user is served with an index page. This page allows him or her to choose from a list of workflows and parameters studies available to that user ID. The user's choice is stored in the new VisualizationBean object. The team decided to create the VisualizationBean as a new class rather than extend the PGradeBean class because most of the functions available in PGradeBean were either obsolete in the VisualizationPortlet or needed to be overridden.

The next action differs depending on if the user chooses a workflow or a parameter study from the list presented on the index page. In the first case where the user chooses a workflow he or she will be presented with the VisualizeWorkflow page. Here the user can see all output for a workflow as each job in the workflow makes available its output files. The jobs are listed along the left side of the screen while the corresponding output files are listed beside the jobs. The list of jobs is taken from a file that the PGRADE Portal creates when a workflow is executed. Other than view the output files there is little the user can do for a standard workflow.



Visualization - Workflow Preview

Selected Workflow: wf1

Output Files

job11

File does not exist yet.
out11.txt

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec id nulla ac nulla imperdiet tincidunt. Aliquam ut pede. Curabitur sed nisl. Cras rutrum magna a purus. Nulla elementum. Curabitur interdum pretium pede. Curabitur elementum accumsan est. Quisque lorem libero.
out2.txt

File is in unsupported format

bytes: 22

modified: Tue Apr 19 15:38:03 CEST 2005
nosupport.aaa

job12

test1.bmp

Nam massa neque, posuere eu, eleifend et, luctus sed, mi. Nam placerat. Morbi feugiat nunc quis metus. Duis vulputate nibh vitae sapien. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed pretium semper velit. In
out3.txt

m1.gif

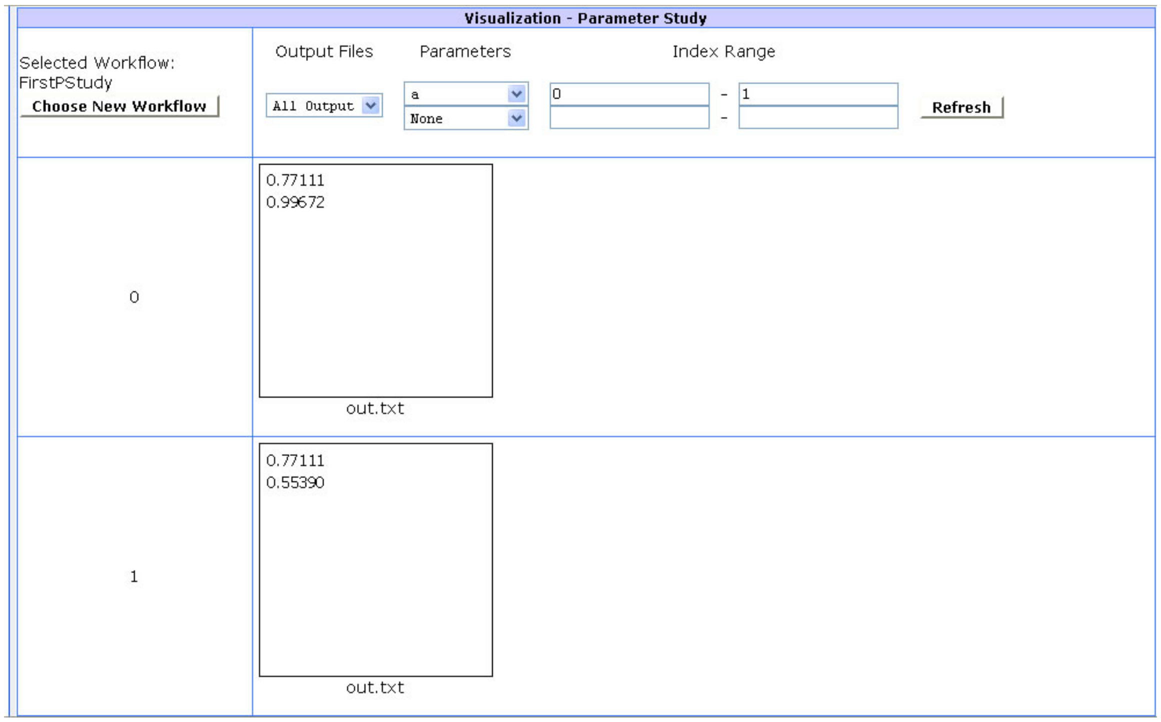
Figure 16: VisualizationPortlet for standard workflows. Jobs are listed on the left and outputs on the right.

In the second case, where the user selects a parameter study from the index page, the VisualizationPortlet will serve him or her with the VisualizePStudy page. This page shows the output files created by the PStudyJob structured by the parameter. Since all of the instances of the parameter study will create files with identical names, the user is allowed to select which files to display on this screen. The user is allowed to select either one or two parameters by which to organize the preview and he or she may also specify an index range for each parameter. The "index range" may best be described by an example.

Since the input to the parameter study may be of any type (integer, decimal, string, etc.), for an example parameter study, parameter "a" may be represented by the array {"egy", "kéttő", "három", "négy"}. In this case if the user specified that he or she wanted to structure the view by parameter "a", indexes 1-2, then that would be translated to the results when parameter "a" is either "kéttő" or "három." This is done because sometimes it is more feasible (as is the case with values represented by strings) to specify a range of indexes than a range of values.

The display on the page differs depending on the amount of information that the user supplies. If the user chooses one parameter to visualize then the values for that parameter are listed vertically along the left of the page starting with the low index that the user specified for that parameter and ending with the high index that the user specified. In the case that the user does not supply sufficient information to display the results (for example, if the low or high indexes are left blank) then the page defaults to using the zero-th index for the first parameter.

The output files corresponding to each index of the given parameter are shown to the right of the index. In the event that there are several other parameters then the output files shown will be those for when the other parameters are at their zero-th index. To find the output files for each parameter the page calls the getOutPaths() method from the PStudyJobList class. This method returns an array of strings where each string represents the directory where the output files for that PStudyJob with the given parameters and values are kept. The page displays all the files in that directory.


Visualization - Parameter Study

Selected Workflow: FirstPStudy

0

```
0.77111
0.99672
```

out.txt

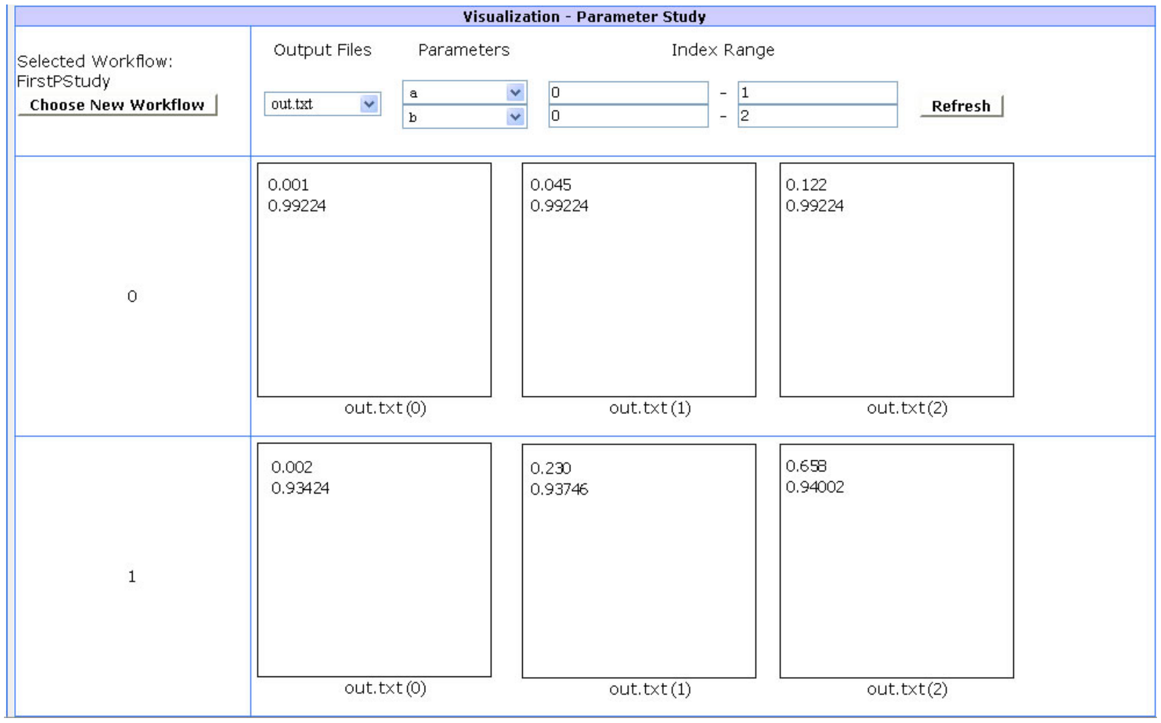
1

```
0.77111
0.55390
```

out.txt

Figure 17: VisualizationPortlet for a parameter study structured by one parameter. The index is on the left and outputs on the right.

If the user selects two parameters to visualize then the structure of the results preview is changed. The indexes for the first parameter are still listed vertically along the left side of the screen as they were in the case of one parameter. To the right of the index are all the output files that correspond to the intersection of the first parameter and the second parameter. Next to the file name is the index of the second parameter that the output file comes from. The reason that the index is written next to the file name as opposed to along the top is because the portal cannot extend horizontally beyond the screen so sometimes the output files wrap onto another line. The numbers prevent the user from confusing the results.



	out.txt (0)	out.txt (1)	out.txt (2)
0	0.001 0.99224	0.045 0.99224	0.122 0.99224
1	0.002 0.93424	0.230 0.93746	0.658 0.94002

Figure 18: VisualizationPortlet for a parameter study. Indexes for the first parameter are on the left while outputs and their second-parameter indexes are on the right.

This list of files is generated by calling the getOutPaths() method from the PStudyJobList class once using the first parameter and once again using the second parameter. The resulting arrays are compared and the first element that appears in both arrays represents the directory where the page will look for the output files. Since that directory can contain many output files the user must choose one to represent on the screen. The default file to use is the first output file in lexicographical order.

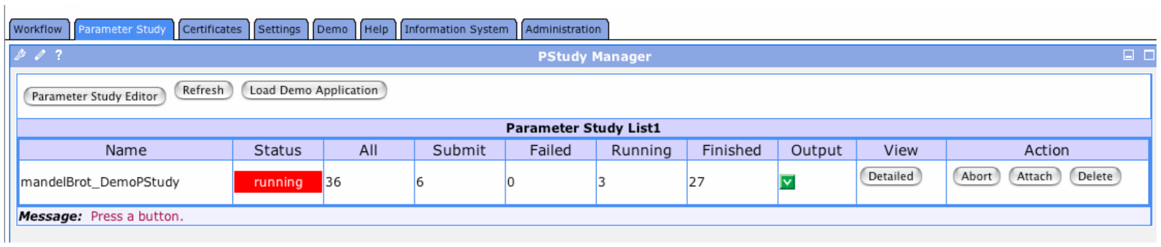
In any case, if the user has chosen to visualize either a workflow or a parameter study, the output files themselves are represented the same way. Each is represented on screen in a block that is 200 pixels tall by 200 pixels wide. Below the block is the name of the file (and parameter number if applicable). If the output file is a .txt or a .xml file then the JSP will open the file and place the contents into the block. If the file is an image file of type .bmp, .gif, .jpg, or .tiff then the JSP will copy the file to a directory local to the Tomcat web server using FileImageInputStream and FileImageOutputStream. An <img> tag will be placed into the block on the resulting webpage. The JSP calls deleteOnExit() which is available to objects of type File in Java. This means that when the virtual machine exits the file will be deleted. The file's name is based on the current timestamp. This is done to avoid collisions because many output files in a workflow or parameter study may share a common name. And finally, for files that are neither images nor text the block will display a message saying that the file is in an unsupported format followed by the size of the file in bytes and the time that it was last modified. If the file does not exist yet because the job has not finished or if there is any other sort of error, the block will display a short error message.

4 Testing

4.1 Back-end

The back-end was built and tested in pieces. First the Job Repository was designed and tested. We used dummy PStudies to ensure that the proper files were being stored, and that all information was retrievable. The interfaces to the Job Repository were tested one method at a time, which dummy methods in place of untested code. Next we designed the Scheduler. The load-balancing algorithm was designed and compared against other resource allocation methods.

When fully integrated, we were able to run parameter studies with known results in order to test our system. Our test applications were a Mandelbrot set generator as well a chemistry application. The results were favorable, as jobs were submitted and the expected output was returned. This demonstrates our system as a working test bed for parameter studies.



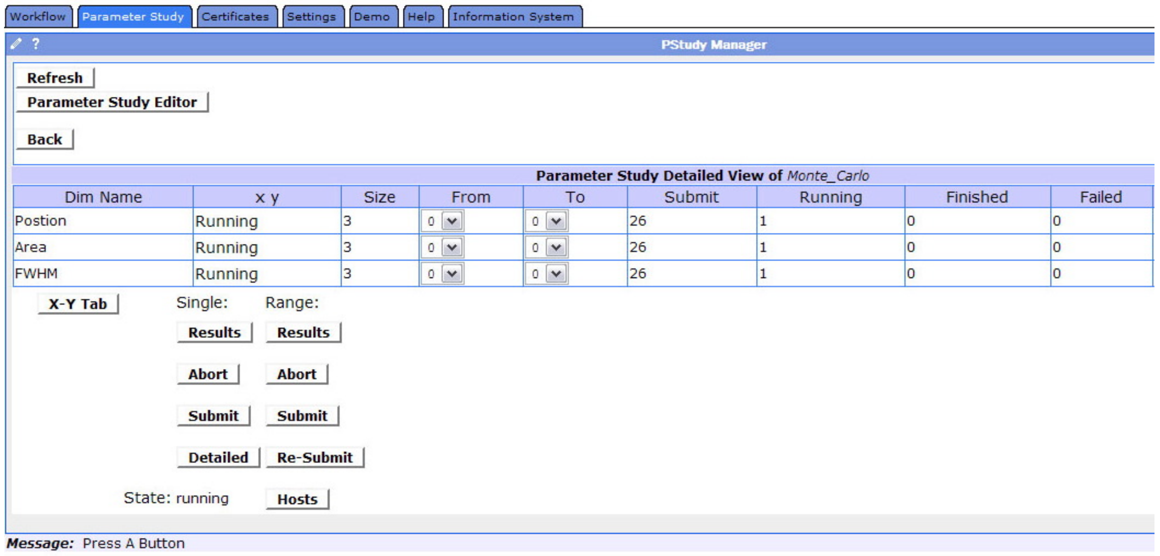
Parameter Study List1

Name	Status	All	Submit	Failed	Running	Finished	Output	View	Action
mandelBrot_DemoPStudy	running	36	6	0	3	27	✓	Detailed	Abort Attach Delete

Message: Press a button.

Figure 19 Running Mandelbrot Study

We compared the results of our load-balancing algorithm against a static allocation method, where resources where assigned to jobs before they were submitted. This led to slower resources becoming overloaded. Faster processors would finish quicker, and wait idle without being assigned a new job. Slower resources would get assigned new jobs before they complete their first. Our load-balancing algorithm ensures that overloaded resources are not assigned new work, as well as fairly distributing new jobs.



Parameter Study Detailed View of *Monte_Carlo*

Dim Name	x y	Size	From	To	Submit	Running	Finished	Failed
Postion	Running	3	0	0	26	1	0	0
Area	Running	3	0	0	26	1	0	0
FWHM	Running	3	0	0	26	1	0	0

State: running

Message: Press A Button

Figure 20 A detailed view of a Running PStudy

4.2 Mandelbrot Application

Before using the Mandelbrot application as a test for the Parameter Study Module it was necessary to verify that the application did in fact generate a correct Mandelbrot set. This was fairly simple with the amount of resources that can be found online. Fractal applets such as the one by David Joyce, a Professor at Clark University [17], are quite numerous and easily showed that the application developed by the team was correct. The only oddity was that bitmaps are rendered starting with the pixel in the lower right corner, meaning that when an application creates a bitmap it must actually write the image data in reverse so it will display correctly.

4.3 Visualization Portlet

In order to test the VisualizationPortlet during development it was necessary to create mock workflows and parameter studies. This was because not all parts of the project had been implemented at the time. So, mock workflows and parameter studies were created along with complete directory structures and output files. Doing this by hand allowed the team to see exactly how functions should create output and exactly what kind of input should be expected. Mock output files were varied between text, image, unknown type, and erroneous files to verify that the Portlet would react accordingly.

4.4 PStudy Application

The final application to test was the Parameter Study Module in its entirety. To do this the team ran the Mandelbrot application as a parameter study and compiled the results. Then the team ran the Mandelbrot application as a sequential program and modified the inputs by hand. These results were then compared with the results generated by PStudy and they were found to be identical.

5 Results

Once all the pieces were working and assembled, the results of the Parameter Study Module were exemplary. Running the Mandelbrot application as a parameter study revealed many areas of the set that could have taken hours of hunting to find. It was easy to create the parameter study and the results were ready quickly. Using the visualization Portlet to view the results made it simple to pinpoint areas that deserved further exploration. It was also very easy to see how changing the parameters would affect the generation of the set.

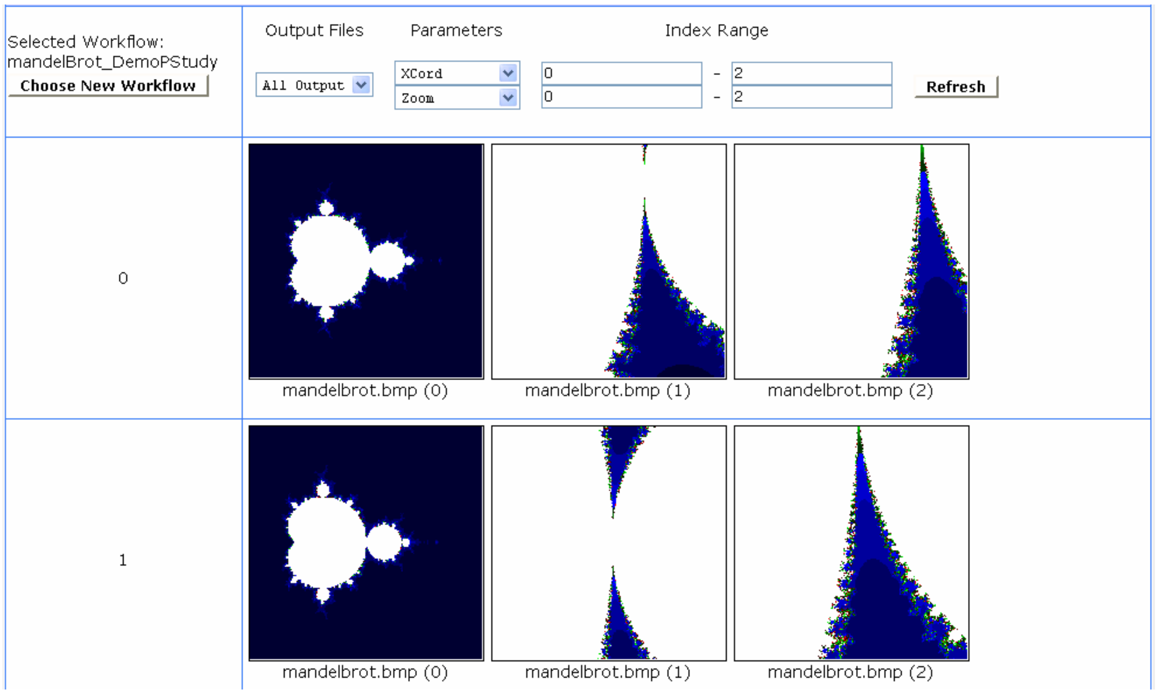


Figure 21: Results of the Mandelbrot application

The best result by far was the ease with which the Mandelbrot application was transformed into a parameter study. By simply loading it into the system and setting the parameters used in the input file we could instantaneously generate results. This showed how easy it would be for other researchers to generate their own parameter studies.

6 Conclusions and Future Works

The Parameter Study Module for the PGRADE Portal was overall a success. The area of parameter studies is one that should be moved into the realm of parallel computing and this represents a great first step. The amount of processing power available on a parallel system would serve a parameter study well and open new doors for research. Our system makes the processes of creating and running parameter studies simple. Creation is done with an easy to use graphical editor. These parameter studies can then be run on a Grid system. Each instance of a PStudy is unique with no dependencies to each other. Therefore, each individual job can be run in parallel. This parallel execution is ideal for efficient completion of a parameter study. The PStudy extension to the P-Grade Portal provides a fair load-balancing solution for dynamic allocation of resources. This provides an efficient use of under-loaded resources. Our algorithm out performs the static resource allocation that was previously implemented. When this is combined with the ease of use of the PGrade Portal, researchers in many fields will find applications that would benefit greatly from the PStudy module.

That is not to say that the Parameter Study Module is in its final state; to the contrary it is just beginning to show its potential. During development, implementation, testing, and analysis the team found many areas where the weaknesses of PStudy may be strengthened and where functionality may be added. Also, there were a few pieces of the original specification that were not fully implemented due to time constraints or system constraints, but these pieces are mostly trivial coding sections that would serve to polish the application.

6.1 Changes to Back-End

Our goal for this project was to provide an extension to an existing framework to allow the use of parameter studies. This underlining framework relies on workflows, which are composed of many jobs. For the simplification of this project, we limited our view of a parameter study as the execution of a single job, with a single variable parameter file. This gave us a useable test bed to prove the parameter study concept. We propose that in the future this work should be extended to allow parameter studies with multiple jobs.

A complete workflow that contains only one job with a parameter study job would be an easy implementation. It would only require that the editor be changed to allow linked jobs but to limit the user to one parameter study job. The current back-end system understands and allows a full workflow, which includes one parameter study. The real challenge would be to implement a workflow with multiple parameter study jobs. This would be a useful extension of our work.

The advantage of parameter studies is studying the difference between two sets of outputs. Our current system has a starting set of tools for studying output. This includes a results tab that visualizes output for a single job, or the comparison at the intersection between key indexes. Future work would include the addition of more tools for examine the output. These tools could include studying the connection between two keys by viewing their outputs side-by-side. These tools could also intelligently find commonality between outputs, and produce data that shows key relationships.

6.2 Additions to the VisualizationPortlet

A few additions could be made to the visualization Portlet that would increase its functionality. Currently little validation is done by the Portlet. The system would benefit by ensuring that on the "Visualize Parameter Study" page users always enter a value in the "high index" fields that is greater than or equal to what he or she has entered in the "low index" fields. Also, the JSPs would be easier to update in the future if they were broken into functions. In order to do this a new taglibrary would have to be defined. Since this is seldom done (if at all) in the rest of the PGRADE Portal there does not seem to be a standard for it which is the reason it was not implemented. The only taglibraries that are used extensively are portletui and portletapi which seem to have more to do with Globus than with PGRADE.

Other than that some work could be done with output file representation and manipulation that would prove to be beneficial to the system. First, instead of copying the image files to a temporary directory it would be better if a thumbnail of the image was transferred. This would reduce the amount of bandwidth used to send the image to the user. This can be done programmatically using functions in Java's java.awt.Toolkit [15]. Also there should be a better way to remove the image files from the temporary directory. Currently they are deleted when the virtual machine exits, but that only happens when Tomcat is restarted. On any stable system Tomcat should not be restarted frequently, so the temporary directory could grow quite large. It may be better if the directory is emptied when the user logs out of the PGRADE Portal or when the VisualizationPortlet object is removed.

6.3 Suggestions for P-GRADE Portal

In the course of working with the existing system the team formulated a few suggestions for improving upon the Portlet as a whole. The first is to rework the usage of the "portletui" taglibrary. In the current implementation, the portletui taglibrary prevails throughout JSPs; in fact it seems to be used more than standard html to specify layout. Unfortunately sometimes portletui seems to cause more layout problems than it promises to solve. For example, it is difficult (if not impossible) to allow an inner block-level element to resize a containing element beyond the horizontal size of the window. What this means is that an element such as a <div> that is 1000 pixels wide will not be able to force the page to scroll horizontally if it is necessary. This causes layout problems when there are numerous output files to display in the VisualizationPortlet. The team feels that it may be better if less emphasis is put on portletui for layout and instead a combination of xhtml and cascading style sheets is used to improve layout and design. Rather, portletui may serve a better purpose if it were extended to provide additional functionality to form elements. For example, common functions such as ensuring that a user has entered only integers into a text-box, could be moved to portletui so they would only have to be written once and could be used anywhere in the Portal. To accomplish this attributes such as 'input-type="integerOnly"' could be added to ui:textfield tags.

References

[1] The Apache Jakarta Project: http://jakarta.apache.org/

[2] S Bavuso. "Aerospace Applications of Weibull and Monte Carlo Simulation with Importance Sampling".

[3] K. Benmohammed-Mahieddine, P.M. Dew. "A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems" *SIGOPS Operating Systems Review Vol. 28 Issue 1*. January 1994 pp. 66-79

[4] "Mandelbrot, Benoit B.." Britannica Concise Encyclopedia. 2005. Encyclopædia Britannica Premium Service
15 Apr. 2005 <http://www.britannica.com/ebc/article?tocId=9371151>.

[5] "DemoGRID Summary"
http://www.caesar.elte.hu/eltenet/projects/demogrid/index.html
[6] Robert L. Devaney. "The Fractal Geometry of the Mandelbrot Set."
http://math.bu.edu/DYSYS/FRACGEOM/FRACGEOM.html

[7] D.L. Eager, E. D. Lazowska, and J. Zahorja, "A Comparison of Reciever-Initiated and Sender-Initiated Adaptive Load Sharing." *Proc of the 1985 ACM SiGMETRICS Conference on Measurement and Modeling of Computer Systems*, August 1985

[8] I. Foster. "Designing and Building Parallel Programs". Addison-Wesley, Inc and Argonne National Laboratory, 1995

[9] I. Foster, C. Kesselman. "The Globus Project: A Status Report" *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop,* pp. 4-18, 1998

[10] I. Foster, C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations." International J. Supercomputer Applications, 15(3), 2001.

[11] G.A. Geist, J.A, Kohl, and P.M. Popadapolous. "PVM and MPI: A Comparison of Features". May 30, 1996

[12] A. Gourgoulis, P. Kacsuk, G. Terstyanszky, S. Winter. "Using Clusters for Traffic Simulation".

[13] T. Herzog, ASA. Ph.D., G. Lord, ASA, Ph.D. "Applications of Monte Carlo Methods to Finance & Insurance". ACTEX Publications, 2003

[14] S. Hummel, J. Schmidt, R.N. Uma, J. Wein. "Load-sharing in Heterogeneous Systems via Weighted Factoring" *SPAA '96: Proceedings of the 8th annual ACM symposim on Parallel algorithms and architectures*, 1996

[15] "Java AWT Toolkit". http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Toolkit.html

[16] Java 2 Platform, Standard Edition, v 1.4.2 API Specification: http://java.sun.com/j2se/1.4.2/docs/api/index.html

[17] D Joyce "Julia and Mandelbrot Sets", 2003

[18] JSR 168 Specification: http://www.jcp.org/en/jsr/detail?id=168

[19] P. Kacsuk, G. Dozsa, J. Kovacs, R. Lovas, N. Podhorszki, Z. Balaton, G. Gombas. "P-Grade: a Grid Programming Environment" *Journal of Grid Computing, 2003*

[20] P. Kacsuk, G. Sipos, Z. Farkas. "World-wide parallel Processing by the P-Grade Grid portal" 2005

[21] S. Kadam. "Create Bitmap Images Using a Text Editor". *DeveloperIQ.* 2004 http://www.developeriq.com/articles/view_article.php?id=137&cid=1

[22] M Ljungberg and M A King. "Monte Carlo Calculations in Nuclear Medicine: Applications in Diagnostic Imaging". IOP Publishing, 1998.

[23] Y. Li and M. Mascagni. "Grid-based Monte Carlo Application". http://www.cs.fsu.edu/~mascagni/papers/RICP2002_3.pdf

[24] M. Litzkow, T. Tannenbaum, J. Basney, M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System" *University of Wisconsin-Madison Computer Sciences Technical Report #1346*, April 1997

[25] R. Lovas, G. Dozsa, P. Kacsuk, N. Padhorszki, D. Drotos. "Workflow Support for Complex Grid Applications: Intergrated and Portal Solutions" *Grid Computing – Second European AcrossGrids Conference,* 2004

[26] The Message Passing Interface (MPI) standard. http://www-unix.mcs.anl.gov/mpi/

[27] MPICH-A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich/

[28] MTA-STAKI Laboratory of Parallel and Distributed Systems Homepage. http://www.lpds.sztaki.hu/

[29] NCSA. Message-Passing Standards: MPI and PVM, 1999[10] I. Foster, C. Kesselman: "The Anatomy of the Grid: Endabling Scalable Virtual Organizations." *International J. Supercomputer Applications,* 15(3), 2001

[30] PVM – Parallel Virtual Machine http://www.csm.ornl.gov/pvm/pvm_home.html

[31] "The See-Grid" http://www.see-grid.org/

[32] G Sipos and P Kacsuk "The PROVE Trace Visualization Tool as a Grid service", 2004

[33] Sun Portlet Wiki: http://wiki.java.net/bin/view/Portlet/JSR168FAQ

[34] D. Thain, T. Tannenbaum, M. Livny, "Condor and the Grid", *Grid Computing: Making The Global Infrastruction a Reality*, John Wiley, 2003

[35] D. Thain, T. Tannenbaum, M. Livny, "Distributed Computing in Practice: The Condor Experience", *Concurrency and Computations: Practice and Experience*, 2004

[36] J. Woller. "The Basics of Monte Carlo Simulations". http://www.chem.unl.edu/zeng/joy/mclab/mcintro.html, Spring 1996.