Grid Technology – Vision, Architecture, and Node Capacity Considerations

Dominique A. Heger
Fortuitous Technologies
Austin, TX
dom@fortuitous.com

Phil Carinhas
Fortuitous Technologies
Austin, TX
pac@fortuitous.com

Greg Simco
Nova Southeastern
Fort Lauderdale, FL
greg@nova.edu

Abstract

The basic concept of Grid networking is not new, what *is* relatively new is the current transition of Grid networking from R&D into the commercial arena. The 1st part of this study introduces the vision of Grid networking, and the rather pragmatic evolution that was taken to realize that vision. The report elaborates on the Grid architecture and technologies that surround the Grid paradigm. Further, the report addresses some of the issues surrounding Grid applications. The 2nd part of this study discusses some of the challenges faced by the Grid scheduler and Grid resource managers, and introduces an analytical model that allows quantifying the capacity behavior of the Grid nodes, focusing on improving the reliability and aggregate performance behavior of Grid applications. The model is augmented by a Monte Carlo based probability estimation procedure that focuses on optimizing the communication behavior between a Grid node and the Grid scheduler, respectively.

1 Introduction

In today's economic environment, any organization aims at reducing the time-to-market and any associated cost factors. At the same time, constraints on processing power, and the limitations on existing computing infrastructures hamper the implementation of efficient and effective IT solutions. It is increasingly important today to explore new avenues to utilize the existing IT resources. In many industries, such as financial services, manufacturing, or life sciences, significant improvements to the bottom line have been realized by adopting some form of Grid computing. In a nutshell, the basic idea behind Grid computing is to interconnect and utilize available processor, storage, or memory sub-components of distributed computing systems to solve larger problems more efficiently. The benefits of Grid computing are (1) cost savings, (2) improved business agility by decreasing the time-to-market (delivering actual results), and (3) enhanced collaboration and sharing of resources among departments or institutions [3].

To reiterate, the vision of the Grid movement is to virtualize the computing landscape, focusing on the main goal of creating an actual *utility computing model* that is distributed over a set of resources. In general, a single compute node includes some basic elements such as a processor, some storage (I/O) subsystem, an operating system, and some form of network or interconnect interface. The basic concept of Grid computing is to establish a similar environment, over a distributed area, incorporating heterogeneous elements such as server nodes, storage devices, and network components in a scaleable, wide-area spanning compute infrastructure. The software that coordinates the participating components and elements is analogous to the operating system in a single computer or server environment [1].

2 Virtualization of Computing Resources

Virtualizing the compute resources yields a scaleable (and flexible) pool of processing and data storage components that can be utilized to improve efficiency. Moreover, it aims at generating a competitive advantage by streamlining product development, allowing an organization to focus on their core business. Over time, Grid environments will enable the

creation of virtual organizations and advanced Web services, as partnerships and collaborations become more critical in strengthening each link in the value chain. It has to be pointed out though that nowadays there are many definitions for Grid computing, but the core concept revolves around the "*aggregation of geographically dispersed computing, storage, and network resources, coordinated to deliver improved performance, higher quality of service, better utilization, and easier access to data. It enables virtual, collaborative organizations, sharing applications and data in an open, heterogeneous environment*" [1],[2].

The common denominator for all Grid deployments is the network layer. The shared network fabric is the component that connects all of the resources in a Grid, and hence has a profound impact on the success of a Grid implementation. Therefore, high-speed data networking technologies have been the cornerstone for the evolution of the Grid technology. Distributed Grid systems demand high-speed connectivity and provide very low latency behaviors. High-performance switching elements are paramount in meeting these requirements. Hence, the statement can be made that components such as (bundled) Gigabit Ethernet interfaces, Myrinet connections, or InfiniBand based solutions are considered a necessity to achieve the performance goals. Over time, as the Grid infrastructure evolves from cluster systems to (1) virtualized enterprise data centers to (2) large distributed, wide-area spanning deployments, the underlying network infrastructure has to grow in a cost-effective manner, be scaleable, and meet the performance requirements [5]. Grid computing evolved out of the academic research community and the national defense industry, where researchers have to process vast amounts of data as efficiently as possible (mostly in simulation related projects). Utilizing the original concept of Grid computing, arrays of computational power and storage are constructed from a network of many small (and sometimes widespread) compute nodes. The resulting infrastructure is used to perform large calculation and operation based studies, where the workload generator (application) can be decomposed into independent units of work. This approach allows massive computational projects to achieve results that otherwise could not be completed, even on today's largest super-computers. To re-emphasize, as the concept has evolved, Grid computing has gained some acceptance in the commercial marketplace. Institutions with both, large and small networks have adopted Grid techniques to reduce the aggregate execution time and to enable resource sharing.

3 Types of Grid Environments

There are three basic types of Grid environments discussed in the market today. (1) The compute Grid, (2) the data Grid, and the (3) utility Grid. On the other hand, from an application perspective, there are two types of Grid environments, the compute and the data Grid, respectively. From a topology perspective, the argument being made is that there are additional Grid types, such as the cluster systems, intra-Grids, extra-Grids, and inter-Grids [6]. In reality, clusters, intra-Grids, extra-Grids, and inter-Grids can be better defined as representing stages of the Grid evolution. In each of these stages, it is feasible to support compute Grids, data Grids, or a combination of both types. The majority of the early Grid deployments have focused on enhancing computation, but as data Grids provide easier access to large, shared data sets, data Grids are becoming increasingly important.

A *compute Grid* is essentially a collection of distributed computing resources (within one or multiple locations). The compute resources are aggregated to act as a unified processing resource (virtual super-computer). The process of interconnecting these resources into a unified pool involves the coordination of the usage policies, job scheduling, queuing issues, Grid security, as well as user authentication. The main benefit of a compute Grid is to construct an infrastructure that allows efficient processing of compute-intensive jobs, by utilizing existing resources [3].

A *data Grid* provides distributed, secure access to current data. Data Grids enable managing (and efficiently utilize)

database information from distributed locations. Much like a compute Grid, data Grids also have to rely on software components to insure secure access and enforce usage policies. Data Grids (such as compute Grids) can be deployed within one or across multiple domains. Data Grids eliminate the necessity to move, replicate, or centralize data. Actual data Grids are being used today, primarily serving collaborative research community's [1],[4].

The evolution from compute Grids to data Grids is an important factor in repositioning Grid applications from education and R&D to large enterprises [7]. This transition is an indicator that the market (and the technology itself) is maturing. From a networking perspective, the impact of data Grids includes a tighter integration of storage protocols and high-performance networking. As the middle-ware components for Grid infrastructures mature, coupled with the recent advances in data networking equipment, the statement can be made that the necessary momentum for evolving Grids from local clusters to distributed, wide-area systems is well defined. The first stage of Grid computing revolves around clusters. Based on the true definition of a Grid that includes terms like *distributed and heterogeneous*, it is debatable whether cluster systems should actually be considered as a Grid solution. Semantics aside, nevertheless, clusters are paramount in the evolution of Grid computing. Clusters are often defined by using terms such as distributed or parallel file systems or by a collection of homogeneous server nodes that are aggregated for increased performance. Cluster systems are largely being used in application domains such as simulation-based testing and evaluation. The majority of the newer cluster systems utilize high-speed interconnects such as bundled Gigabit Ethernet, Myrinet, InfiniBand, or any other (sometimes proprietary) high-throughput network interface. Low-latency switching is critical in maintaining application performance across the cluster fabric [4],[6].

To reiterate, cluster systems are critical in the evolution of Grid computing. As in order to move to the next phase (*intra-Grids*), these cluster systems have to be interconnected. By interconnecting individual cluster systems, it is feasible to enable the establishment of enterprise and inter-departmental Grid systems. The creation of intra-Grids puts additional strain on the controlling software layer (middle-ware) and the underlying network infrastructure. The middle-ware has to have a better understanding of the resource allocation, based on the additional complexity of the processor-sharing relationships. Further, latency issues at the network layer become more challenging as well, shifting the focus to even higher throughput capable interconnects. In a next phase, extra-Grids will reflect cluster or intra-Grids that are connected among geographically distributed sites within either a single or among several organizations. The two important distinctions made for an extra-Grid are (1) geographic distribution and (2) inter-enterprise relationships [2],[3],[4]. Because of these relationships, extra-Grids are sometimes referred to as partner Grids. In such a partner Grid, as the data can be shared among different organizations, authentication, policy management, and security issues become even more critical, a fact that has to be addressed by the Grid middle-ware layer. Further, load balancing, topology discovery, network (LAN/WAN) throughput, and application awareness are other factors that have to be considered to provide adequate performance.

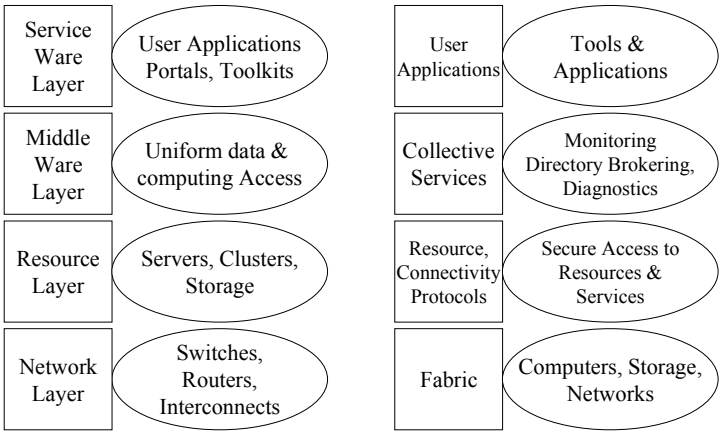
The final stage in the evolution of Grid computing revolves around the *inter-Grid* paradigm. This stage reflects the most powerful phase of Grid evolution, as it embodies two of the primary visions for Grid computing, (1) the utility computing infrastructure and (2) the Grid services providers. Although this is the final stage (which has not yet been reached from a commercial perspective), it has to be pointed out that inter-Grids do exist today in the R&D domain (the TeraGrid as an example). Inter-Grid setups surpass all the other phases in regards to the relative complexity requirements [6]. To illustrate this point, compared to a setup where a couple of institutions participate in an extra-Grid environment, an inter-Grid may service thousands of users. As a result, all of the complex requirements at the middle-ware and the network layer are increased

significantly. Grid setups in this stage are also referred to as utility Grids. At this time, two significant statements about the status quo of Grid implementations can be made. (1) Cluster systems are evolving toward intra-Grid setups in the commercial world. (2) Complex inter-Grids are being designed, tested, and deployed in the R&D domain. These are signs that the Grid momentum is still building, and the market is moving forward. The key to successfully penetrate the market will be to establish early deployments, so that the technology can evolve in parallel with the market requirements.

3.1 Grid Layers

Figure 1: GRID Structures

Descriptions of the Layered Grid Structure



At the base of the Grid architecture, the bottom layer so to speak, is the network, which assures the connectivity for the resources in the Grid (see Figure 1) [2],[4],[6]. On top of the network layer is the resource layer, which incorporates the actual resources that are part of the Grid (mainly computers and storage systems). The middle-ware layer (positioned above the resource layer) provides the tools that enable the various elements (servers, storage, and networks) to participate in a unified Grid environment. The middle-ware layer can be described as the intelligence that brings the various elements in the Grid together. The top layer of the Grid structure represents the application layer, which includes all different user applications (science, engineering, business, financial), as well as portals and development toolkits that are supporting the Grid applications. It is the top layer that the users of the Grid are actually working with.

In most Grid architectures, the application layer also provides the *service-ware* functionality. The service-ware represents a set of general Grid management functions that are utilized to measure the amount of time a user spends on the Grid. Further, the service-ware acts as the billing functionality for the Grid services (assuming a commercial model), and keeps track of who is providing the resources and who can use them. It has to be pointed out that the service-ware is part of the top layer, as it is an entity the user interacts with, whereas the middle-ware components reflect a hidden layer that the user dose not necessarily has to worry about. There are other ways to describe this layered Grid structure. To illustrate, the Grid community uses the term fabric for all the physical infrastructure of the Grid, including the computers and the communication network. Within the middle-ware layer, distinctions can be made between a layer of resource and connectivity protocols, and a higher layer of collective services. Resource and connectivity protocols handle all the Grid specific network transactions (among different computers and any other resources on the Grid). This is accomplished via

communication protocols, which let the resources communicate with each other, enabling the exchange of data as well as authentication protocols, which provide secure mechanisms for verifying the identity of both users and resources [6]. The collective services are also based on protocols, actual information protocols, which obtain information about the structure and state of the resources on the Grid, and management protocols which negotiate access to resources in a uniform way. These services include:

- Updating the directories that represent the available resources
- Providing brokering services (buy and sell resources)
- Monitoring the Grid and diagnosing any potential issues
- Replicating key data so that multiple copies are available at different locations for ease of use
- Providing membership/policy services, keeping track of who is allowed to do what and when

To reiterate, in all the Grid schemes, the top layer represents the applications layer. Applications rely on all the other layers to run on the Grid. To illustrate, assuming a Grid application that analyzes data that is distributed among several independent files. Hence, the application has to:

1. Obtain the necessary authentication credentials to open the files (resource and connectivity protocols)
2. Query an information system and replica catalogue to determine where the copies of the files can be located on the Grid, as well as where the computational resources necessary to conduct the analysis are available, and most conveniently located (collective services)
3. Submit actual requests to the fabric, the appropriate computers, storage systems, and network components to extract the data, initiate computations, and provide the results (resource and connectivity protocols)
4. Monitor the progress of the various computation tasks and data transfer components. Notifying the user when the analysis is completed. Detecting and responding to any potential failure conditions (collective services)

In order to accomplish all the above discussed scenarios, an application written for a single CPU system will have to be *substantially adapted* in order to invoke the proper services, use multiple distributed components simultaneously, and utilize the required protocols [7]. In other words, a Grid requires companies and institutions to heavily invest time and money into getting their applications ready for a Grid environment. Plus, it has to be pointed out that not every application is suitable for running on a Grid.

3.2 Grid Applications

It is paramount to clearly define the type of applications that have the potential to run in a Grid environment, so that the right Grid project can be chosen, realistic expectations can be set, and performance goals can be established and met. Typically, applications that are good candidates for a Grid implementation take many hours (possibly days or weeks) to execute (large problem sizes). In some circumstances, the task is so big that it can not be completed at all even given today's processor capacity. The reason behind the long run time may be due to the application requiring many replicated runs of the same fundamental tasks, such as identical processing on many subgroups of a large data file, or certain types of optimizations or statistical simulations, respectively [3]. Another example of a long-running task may be the one where many independent tasks have to execute against the same large data source (scoring or risk analysis projects). In general, an application should possess one or more of the following characteristics to be considered a good candidate for a Grid implementation.

1. Problem to be solved results in a long execution time
2. Problem to be solved Involves many replicated runs of the same fundamental task
3. Problem to be solved requires processing vast amounts of data

4. The problem to be solved allows the decomposition into multiple execution units and/or data subsets

Many (especially scientific) applications involve repeating the same fundamental task many times against unique subsets of the data pool [1],[3]. While the execution of a single task against a single subset of the data may execute rather quickly, repeating the same task against thousands or even millions of subsets of the data pool impacts the aggregate execution time. These types of problems are massively parallel and the applications that solve them are very well suited to a Grid implementation, as the replicated tasks can be distributed across the Grid and executed in parallel, which greatly reduces the aggregate execution time. Each of the fundamental tasks distributed across the Grid has to have access to all the required (input) data. Sometimes the input data can be small (MB's), and other times the input data may be rather large (GB's). In order to achieve the highest possible efficiency level, the compute nodes have to spend the majority of the time processing compute tasks rather than processing any communication related activities. Compute tasks that require substantial data movement generally do not perform very well in a Grid environment. The data has to be either distributed to the nodes prior to running the application, or the data has to be made available via a shared network storage solution. It has to note that there have been many recent advances in data storage hardware (switches, controllers, disks) and software (advanced parallel file systems for clusters) that provide faster access to data, and hence actively contribute to the success of a Grid implementation.

3.3 Globus Grid Toolkit

Practically all larger (R&D) Grid projects today are built based on protocols and services provided by the Globus Toolkit. The toolkit reflects a software solution, which is being developed by the Globus Alliance, which involves primarily Ian Foster's team at Argonne National Laboratory and Carl Kesselman's team at the University of Southern California in Los Angeles [5]. The toolkit provides a set of software tools to implement the basic services and capabilities required to construct a computational Grid, such as security, resource location, resource management, and communications [8],[9]. Below is a list of the primary elements included in the Globus Toolkit.

- *Globus Resource Allocation Manager (GRAM)*.
- *Monitoring and Discovery Service (MDS)*.
- *Grid Security Infrastructure (GSI)*.
- *Grid Resource Information Service (GRIS).*
- *Grid Index Information Service (GIIS)*.
- *GridFTP, Replica Catalog*, *Replica Management*.

3.4 Benefits of Grid computing

There are several economic and business factors that are contributing to the heightened interest in the development and deployment of Grid computing. Based on the Internet and E-commerce, today's society is inundated with data. As the available data repository grows bigger and wider, the window of opportunity for capturing and translating the available data into information shrinks rapidly. Computing applications in many industries involve processing vast amounts of data and/or performing large numbers of repetitive computation operations that exceed the capabilities of existing platforms. To utilize data analysis techniques to achieve a high level of business intelligence, and improve the decision making process, data has to be analyzed in a much more timely manner. Today's business requirements demand a much larger sample size for analysis to provide the best possible accuracy level achievable. The challenges that IT faces today include budget cuts, server consolidation, hardware provisioning and overall administration, all factors that may further drive the interest in implementing Grid computing. The convergence of recent hardware and software advances has made resource virtualization feasible, hence made it easier to construct and deploy a Grid infrastructure. On the hardware side, advances include networked storage devices and low-cost, modular hardware components (such as blade systems). On

the software side, the advances include improvements in networking, Web services, databases, application servers, and management frameworks [4],[7]. While Grid computing may not be the solution for every application or institution, Grid computing has to be considered as an innovative solution that provides:

1. Scalability of applications, as long-running applications can be decomposed along either execution unit boundaries or data subsets, and hence be executed in parallel, economizing on aggregate execution time.
2. Scalability in number of users, as multiple users can access a virtualized pool of resources to obtain the best possible response time by maximizing the utilization of the compute resources.
3. By implementing Grid computing technology, organizations can optimize the return on investment and lower the cost of ownership.

4 Grid Brokerage Services

The issue addressed in the 2nd part of this study is motivated by the above discussed Grid computing environment, where task-scheduling scenarios are normally performed from Grid nodes on a remote basis. Unlike in a traditional cluster environment, the data utilized by a distributed Grid scheduler can not constantly be updated, as the communication cost would be excessive. Hence, the scheduler has to be capable of making routing choices based on longer-term quantity's [11], [12], [13]. There are several different approaches to task scheduling in a Grid environment. Some of the concepts are derived from the traditional, distributed system environment, and rely on having close to complete knowledge of the resources that are available. The basic idea in such an environment is that the tasks may be directed to the node that has the greatest potential to complete the task first (with a certain degree of probability). If problems emerge, such as a node failure or a task takes much longer to complete than what was predicted, a new schedule can be computed rather easily. To illustrate, Thomas' work on service schedulers [10] aims at improving Grid performance. Spooner's [13] research discusses an approach where local cluster scheduling has been developed for Grid enabled jobs, utilizing a fast genetic algorithm to compute near optimal schedules across highly dependent, distributed tasks. Such an approach is effective on a local level, but can not be directly applied to wider scheduling issues on the Grid. It is generally not feasible to know the status of resources sited remotely from the scheduler, particularly as those resources are subject to local rescheduling.

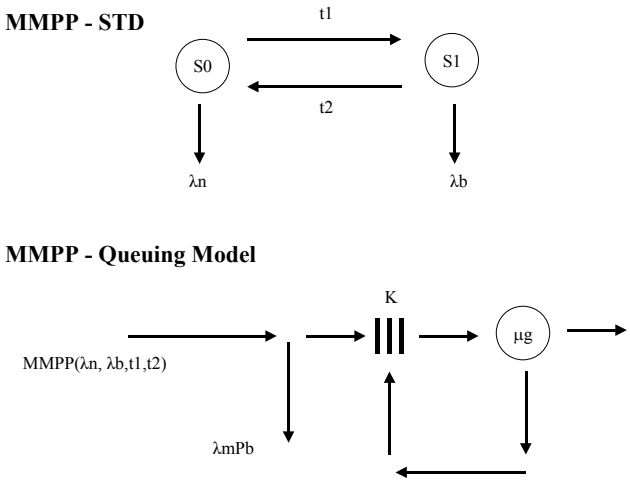
The general problem encountered here is referred to as brokering, incorporating scheduling decisions over resources across multiple domains [12]. A broker operating on the Grid has to be able to select distributed resources, over which the scheduler has no control, and where the information that is available may often be limited or stale [14]. To circumvent these issues, several proposed systems [14], [15], employ resource reservation polices to negotiate a certain service level agreement prior to deployment. This requires direct communication among the scheduler and the resource managers to determine the availability and usage cost of the resources, respectively. Once a good selection has been made, the reservation can be confirmed. However, a certain amount of time has elapsed since the first inquiry, and hence the resource status may already have changed, resulting into a different performance behavior compared to what was expected. Further, a resource manager is normally not considered as a passive entity during that time period either, as the resource manager may offer services to other schedulers. This implies that the system-centric resource manager and the user-centric scheduler are in direct competition. The system therefore reveals an inherently complex interactive behavior, and ultimately will have to be optimized to predominately suit either the system-centric resource manager or the user-centric scheduler.

5 Finite Capacity Issues

The assumption made in this paper is that the nodes in the Grid environment each have a different, finite capacity queue.

As a Grid reflects a heterogeneous environment, some nodes in the Grid have a much higher capacity potential than other nodes. It is further feasible to assume that the scheduler does not know if a node's queue is momentarily saturated, and hence the consequence of sending a task to a node with no capacity is that the task may have to be re-routed. In a Grid environment, minimizing the job loss due to any latency issues can be achieved via specifying a certain node specific threshold that when reached, the node signals the scheduler to send either fewer or no jobs. This approach is similar to the strategy employed in packet switched networks to reduce congestion problems. The result of this strategy is that the queue is less likely to become saturated, consequently the job loss probability may be reduced.

Figure 2: Single Grid Node – Model



Such a strategy may work reasonably well when the system load is relatively light. If all nodes are heavily loaded though, it may not be feasible for the scheduler to re-route some tasks away from the busy nodes, and hence the strategy may not be optimal. Further, such a scenario adds additional states to the scheduler behavior. If each node operates on a single threshold then the operational state space grows, as it will be necessary to track the queue size relative to the threshold. The main issue here is how to determine the optimal value for the queue size thresholds and the scheduler delay for a given set of system parameters. This study addresses the issue of how to quantify the capacity potential on a per Grid node basis, and how to determine the most optimal capacity threshold used to trigger a communication scenario with the scheduler. The argument made is that the distribution of the number of jobs at node *y* with a capacity *K* will tend towards a steady state solution for a M/G/1/Ky queue with a processor-sharing behavior [16],[18].

5.1 Grid Node – Finite Capacity Model

The thesis made is that in a Grid environment, the task distribution (the actual arrival rate into the individual Grid nodes) can not be expressed as a simple steady state average. Hence, this study introduces a queuing model representing a Grid node that encounters a bursty arrival pattern. The arrival pattern of the tasks is assumed to represent a Markov Modulated Poisson Process, and the service discipline of the Grid node reflects processor sharing (PS) [18]. The upper bound on the total number of tasks that can be processed is limited to *K*. The goal of this conceptional model is to determine performance metrics such as the average response time, the throughput, and the blocking probability. It is a fact of life that Grid nodes may become overloaded as the arrival rate exceeds the node's capacity. To cope with such a scenario, overload control can be used, which implies that some requests can be processed while some others may have to be rejected [18].

This allows a single Grid node to achieve reasonable service times for the

accepted tasks. This study resulted into establishing a rather simple abstraction of the above-discussed scenario, a fact that is based on the question that the study tries to answer. A simple model renders a smaller parameter space, and hence is easier to calibrate and validate. In more complicated models, some of the parameters are very difficult to obtain. To reemphasize, the question to be answered drives the complexity of any modeling exercise, and therefore the terms simple and complex reflect the deepness of the problem being analyzed. The thesis made in this study is that a M/M/1/K or M/D/1/K model simulating a First-Come-First-Served (FCFS) service discipline may only predict the Grid node behavior to a certain extend. Conceptually, the argument made is that it is difficult to assume that the service distribution on a Grid node is exponential or deterministic, and that the service discipline reflects FCFS. Hence, this study proposes a model that consists of a processor-sharing (PS) node with a queue size of K [16].

$$\lambda n_{\lambda} := \lambda \cdot 0.66 \qquad (1)$$

$$\lambda b_{\lambda} := \frac{(t1 + t2) \cdot \lambda - \lambda n_{\lambda} \cdot t2}{t1} \qquad (2)$$

$$\lambda m_{\lambda} := \lambda n_{\lambda} \cdot t2 + \lambda b_{\lambda} \cdot t1 \qquad (3)$$

The proposed model augments on research conducted by Cao [19], Nyberg [20], and Scott [21] conducted for Web servers systems. The arrival process into the Grid node is assumed to be a two-state Markov Modulated Poisson Process (MMPP). MMPP abstractions are commonly utilized to represent bursty arrival behavior in communication systems [18]. The service time distribution is arbitrary. The average service time and the maximum number of tasks are parameters that can be determined through a maximum likelihood estimation [1]. A maximum likelihood estimation analysis starts with determining a mathematical expression that is known as the likelihood function of the sample data. Hence, the likelihood of a set of data reflects the probability of obtaining that particular set of data, given the chosen probability distribution model. This expression contains the unknown model parameters. The values of these parameters that maximize the sample likelihood are known as the maximum likelihood estimator (MLE). The maximum likelihood estimation is considered an analytic maximization procedure that applies to every form of censored or multi-censored data, respectively. In the proposed model, the tasks arrive at a Grid node according to the two-state Markov Modulated Poisson Process (MMPP) with parameters λn, λb, *t1*, *t2*, respectively (see Figure 2). An MMPP system reflects a doubly stochastic Poisson process where the rate process is determined by a continuous-time Markov chain [21]. A two-state MMPP (also known as MMPP-2) implies that the Markov chain consists of two different states, *S0* and *S1*. The Markov chain changes state from *S0* to *S1* with intensity *t1*, and transits back with intensity *t2*. When the MMPP is in state *S0*, the arrival process represents a Poisson process with rate λn (Equation 1), and when the MMPP is in state *S1*, the rate λb (Equation 2) is utilized [16]. The service time requirements for the tasks in the queue reflect a general distribution with a mean μg. A task in the queue receives a certain quantum of service and is then suspended until every other runnable task has received a certain quantum of service. After a task has completed the required service, the task leaves the queue.

$$Pb_{\lambda} := \frac{\left(1 - \rho_{\lambda}\right) \cdot \left(\rho_{\lambda}\right)^{K}}{\left[1 - \left(\rho_{\lambda}\right)^{K+1}\right]} \qquad (4)$$

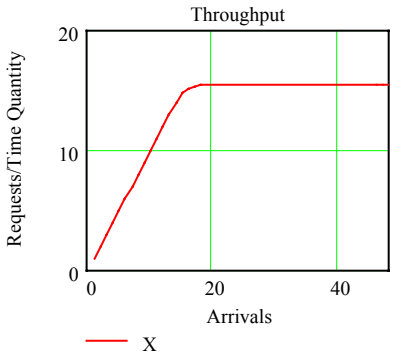
$$X_{\lambda} := \lambda m_{\lambda} \cdot \left(1 - Pb_{\lambda}\right) \qquad (5)$$

$$W_{\lambda} := \frac{\left(\rho_{\lambda}\right)^{K+1} \cdot \left(K \cdot \rho_{\lambda} - K - 1\right) + \rho_{\lambda}}{\lambda m_{\lambda} \cdot \left[1 - \left(\rho_{\lambda}\right)^{K}\right] \cdot \left(1 - \rho_{\lambda}\right)} \qquad (6)$$

The traffic intensity $\rho\lambda$ is expressed as λ times μg. Each node can handle at most *K* requests at a time, but it has to be pointed out that the value for *K* may differ from node to node (based on the resources at hand). The mean arrival rate λm is expressed in this model as depicted in Equation 3. A task will be blocked if the

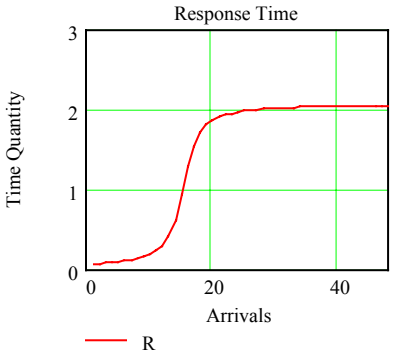
capacity *K* has been reached. *Pb* denotes the blocking probability (Equation 4), whereas the rate of blocked requests is represented by $\lambda m*Pb$. The throughput *X* (Equation 5) reflects the rate of completed requests, and the average response time *W* (Equation 6) depicts the expected sojourn time of a task. The average response time, throughput and blocking probability are performance metrics that can be obtained via simulations [16]. Figure 3, 4, and 5 disclose the response time, throughput, and blocking probability, respectively. In this study, the buffer size *K* was set to 32, whereas the number of tasks arriving at the Grid note was scaled from 1 to 48. The actual arrival pattern into the Grid node was simulated based on the rules outlined in Equations 1, 2, and 3.

Figure 3: Simulated Throughput Behavior



The simulation nicely reveals that under the given workload conditions, as the number of tasks arriving at the Grid node reaches 22, the throughput reaches the asymptotic upper bound imposed by the capacity of the simulated Grid node. On the other hand, as the arrival rate into the node reaches 12 tasks, the response time increases significantly, up to an arrival rate of 28 tasks, where the system starts to oscillate around the capacity potential of the system. With arrivals less than 14, the blocking probability on the node can be considered as being negligible. At the 31-task arrival mark, the blocking probability equals to 50%, whereas at the 48-task level, the blocking probability reaches 67%. To validate the proposed model, an actual empirical study was conducted on a 12-node Linux cluster, utilizing a pathological workload that simulated a parallel, scientific application that operated in 3 distinct processing cycles [17].

Figure 4: Simulated Response Time Behavior

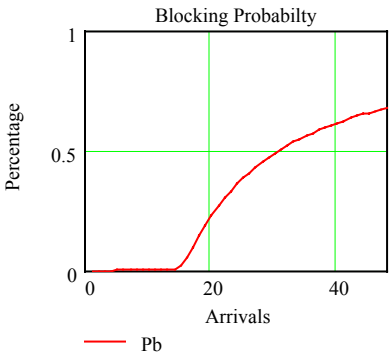


Phase 1 consisted of heavy CPU usage, phase 2 simulated the inter-node communication, whereas phase 3 reflected the IO behavior for checkpoint restart or visualization of the data simulation. During the benchmarks, the workload (number of tasks per node) was increased. The empirical throughput study revealed that the throughput model nicely captured the actual performance behavior of a single node, with an average description error of 9%, and an average prediction error of 14%. The analysis further revealed that the model was very accurate with low arrival counts, and less accurate as the number of tasks arriving at the node increased. Further, in most analyzed scenarios, the Grid node capacity model understated the actual effective performance behavior.

Based on the conducted simulation, the recommendation made is to incorporate the gained knowledge into the Grid scheduling environment, as based on this study, it would be beneficial for the Grid nodes to trigger communication with the Grid managing entities prior to reaching the finite buffer capacity (*K*). In other words, the thesis made is that to improve reliability and aggregate throughput, the Grid nodes have to inform the Grid scheduler to throttle back the number of submitted tasks earlier on, while ample capacity is still available. One major benefit of this approach is that in the case of a node failure, fewer tasks have to be re-routed to other Grid nodes, hence the impact on the aggregate application performance may be minimized. In non-failure situations, the proposed approach

minimizes the probability that application tasks have to be re-routed based on a lack of resources, further enhancing the stability of the Grid environment.

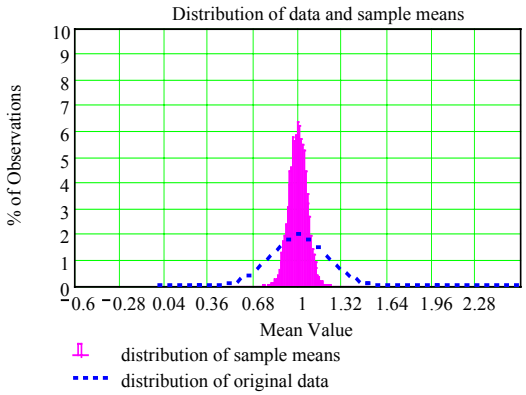
Figure 5: Simulated Blocking Probability



5.2 Probability Estimation

To reduce the number of active messages send from the Grid nodes to the scheduler, this study proposes to utilize a Monte Carlo based probability estimation process that uses historical data collected on the local Grid node to determine if a communication interchange with the scheduler is applicable. A Monte Carlo based method reflects any technique of statistical sampling employed to approximate solutions to quantitative problems. The idea is to empirically determine the probability that the sample mean (based on a sample size *sampsize*), of a normal distribution with parameters μ (mean) and σ (standard deviation) will be in the interval [*a, b*]. In other words, the process consists of first, collecting a larger set of samples from the underlying population (where each sample will contain *sampsize* values). Second, to determine the mean of each *sampsize*, and third, to determine what proportion of the *sampsize's* reflect a mean between 0.9 and 1.1 (while considering 1 as the mean, and a normal distribution behavior). In this approach, the *optimal* per Grid node arrival task value determined by the above-discussed analytical model reflects the yard stick (reference point) into the Monte Carlo analysis. In other words, based on the resources at hand on each Grid node, the analytical model provides the mean for the analysis.

Figure 6: Monte Carlo – Probability Estimation



To illustrate, based on the analytical analysis conducted in this study, it was determined that an arrival capacity of 18 (per time unit) would be considered as efficient and effective for the simulated Grid node. For simplicity reasons, the 18 arrivals are mapped to a mean value of 1 in this study (see Figure 6). Hence, based on the actual arrivals into the Grid node over time (the historical data), the Monte Carlo based approach determines what percentile of the sampled data are in the interval [*a, b*]. The assumption made is that as long as the Grid node operates within this interval, the reliability and performance behavior is considered as being in an optimal state. Assuming a target of 80%, as soon as that value drops below the 80%, the scheduler can be informed to either submit more tasks, or to throttle back on the number of tasks submitted to the Grid node (based on the data distribution). The feature to inform the scheduler that ample capacity is available on the node may not have to be triggered in order to reduce the number of active messages submitted in the Grid environment. Figure 6 outlines the discussed (conceptional) approach. The recommendation made is to determine the epoch for the data collection based on the complexity of the Grid environment.

6 Conclusion

This study introduced some of the key concepts and technologies surrounding a Grid environment. The main focus was on

elaborating on the key challenges faced by institutions to get their application Grid ready. The 2nd part of this study discussed the proposed Grid node capacity model and the Monte Carlo based probability estimation technique that serves two major purposes. First, the approach allows in node failure situations to economize on the number of tasks that have to be re-routed, resulting into improved reliability and performance for the Grid applications. Second, in non-failure situations, the probability that a task has to be rejected by a Grid node is reduced, and at the same time, the Grid node's probability to service the accepted tasks in a timely fashion is increased. Further research on this topic will revolve around further quantifying the communication traffic between the Grid node and the scheduler. As the filed of Grid performance and reliability determination is not yet vastly being incorporated into the traditional systems analysis process, the main intention of this paper was to stimulate some interest in the (commercial) computing and networking community, respectively.

References

1. *Avery, P., Foster, I., Gardner, R., Newman, H. Szalay, "An International Virtual-Data Grid Laboratory for Data Intensive Science", 2001*

2. *Czajkowski, K., Fitzgerald, S., Foster, I. Kesselman, C., "Grid Information Services for Distributed Resource Sharing". IEEE Press, 2001*

3. *Foster, I., "Grid Technologies & Applications: Architecture & Achievements", Argonne National Laboratory, 2002*

4. *Foster, I., Kesselman, C. "A Data Grid Reference Architecture". GriPhyN, 2001*

5. *Foster, I., Kesselman, C. "Globus: A Toolkit-Based Grid Architecture. In The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann, 1999*

6. *Grid Computing and Technologies @ CERN: http://gridcafe.web.cern.ch/gridcafe/GridatCERN/gridatcern.html*

7. *Oracle, "Oracle Grid Computing", Oracle Business While Paper, 2003*

8. *Smith, W., Gunter, D., "Simple LDAP Schemas for Grid Monitoring", NASA Ames Research Center, 2001*

9. *The Globus Alliance: http://www.globus.org/*

10. *Thomas, N.*, Brandely, J., Knottenbelt, W., "Performance of a Semi Blind Service Scheduler, University of Newcastle, 2004

11. *Nitzberg, B., Schopf, J., "Current Activities in the Scheduling and Resource Management Area of the Global Grid Forum", 2002*

12. *Schopf, J., "A General Architecture for Scheduling on the Grid", Argonne National Laboratory, ANL/MCSP1000-1002, 2002.*

13. *Spooner, D., Jarvis, S., Cao, J., Saini, S., Nudd, G., "Local grid scheduling techniques using performance prediction", IEE Proceedings Computers and Digital Techniques, 2003.*

14. *Haji, M., Gourlay, I., Djemame K., Dew, P., "A SNAP-based Community Resource Broker using a Three-Phase Commit Protocol" Performance Study, 2003*

15. *Buyya, R., Abramson D., Giddy, J., "Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid" IEEE Conference on High Performance Computing, 2000.*

16. *Jain, R., "The Art of Computer System Performance Analysis", John Wiley, 1991*

17. *Koenigs, A. "Industrial Strength Parallel Computing", Morgan Kaufmann, San Francisco, 2000*

18. *Gunther, N., "The Practical Performance Analyst", McGraw Hill, 2000*

19. *Cao, J., Andersson, M., Nyberg, C., Kihl, M., "Web Server Performance Modeling Using an M/M/1/K*PS Queue", IEEE, 2003*

20. *[7] Nyberg, C, Cao, J, "On overload control through queue length for web servers", in 16th Nordic Teletraffic Seminar, 2002, Esboo, Finland.*

21. *L. Scott, P. Smyth, "The Markov Modulated Poisson Process and Markov Poisson Cascade with Applications to Web Traffic Modelling", Bayesian Statistics, Oxford University Press,* 2003.