A CASE STUDY ON GRID PERFORMANCE MODELING

Baochuan Lu and Amy Apon
Computer Science and Computer Engineering
University of Arkansas
Fayetteville, Arkansas, USA
email: {blu,aapon}@uark.edu

Larry Dowdy and Frank Robinson
Electrical Engineering and Computer Science
Vanderbilt University
Nashville, Tennessee USA
email:{larry.dowdy, frank.j.robinson}@vanderbilt.edu

Doug Hoffman and Denny Brewer
Acxiom Corporation, Conway, USA
email:{doug.hoffman,denny.brewer}@acxiom.com

ABSTRACT
The purpose of this case study is to develop a performance model for an enterprise grid for performance management and capacity planning[1]. The target environment includes grid applications such as health-care and financial services where the data is located primarily within the resources of a worldwide corporation. The approach is to build a discrete event simulation model for a representative work-flow grid. Five work-flow classes, found using a customized k-means clustering algorithm characterize the workload of the grid. Analyzing the gap between the simulation and measurement data validates the model. The case study demonstrates that the simulation model can be used to predict the grid system performance given a workload forecast. The model is also used to evaluate alternative scheduling strategies. The simulation model is flexible and easily incorporates several system details.

KEY WORDS
grid computing, performance modeling, capacity planning, discrete-event simulation, JavaSim, case study

1 Introduction

Large scale science is increasingly being conducted using distributed global computing that is enabled by grid technologies over the Internet. As companies expand their physical boundaries worldwide, it is common for a single corporation to own large scale distributed data resources that are organized in a single, secure, and coordinated fashion using the grid. Performance management of a grid environment is difficult due to its distributed nature and inherent complexity. It is difficult to sustain constant and predictable performance under varying load conditions. Constant and predictable performance of a grid system is often more important than its theoretical peak performance for many practical applications.

Grid benchmarks have been studied in [1] for design trade-off analysis and how applications and grid infrastructure interact. Previous work minimizes performance variability by developing techniques and tools that enable the grid to adapt to changing environmental conditions [2]. A more proactive approach is adopted here. Performance models are used to evaluate grid performance and to conduct capacity planning studies by predicting the performance under various workload forecast scenarios. A simulation model makes it possible to test and evaluate grid scheduling strategies in a repeatable and controlled environment. Related work includes the Performance Prophet [3], which is a performance modeling and prediction tool for distributed programs. It uses a hybrid model, a combination of mathematical models and simulation tools, for automatic performance model generation. The methodology is mainly for performance modeling of distributed programs during early development. In the target grid environment, the building blocks (i.e., workload elements) are not known to capacity planners. The GridSim [4] toolkit provides capabilities for resource modeling and scheduling simulation. It simulates time and space-shared resources with different capabilities and configurations. GridSim is used to improve the effectiveness of resource brokers on the global grid.

Acxiom Corporation has provided system performance files, job accounting logs, measurement data, private communications, and feedback on the modeling results for this case study. The workload characterization portion of this research simplifies the workload model by clustering jobs into job classes. Five job classes are identified and their characteristics are used as input parameters to the trace-driven simulation model. Comparing the results of the simulation against actual measurements validates the model. After model validation various what-if scenarios of altering the configuration of the grid are simulated. Using the simulation results, capacity planners can answer performance questions concerning the current capacity of the system and have more confidence in the future performance of the system. Such questions include:

- What are appropriate Service Level Agreements (SLAs) that the current grid system can achieve?
- What would be the system performance if the work-

[1]This research is funded by a grant from the Acxiom Corporation, Conway, Arkansas, USA and support by the National Science Foundation under Grant No. 0421099.

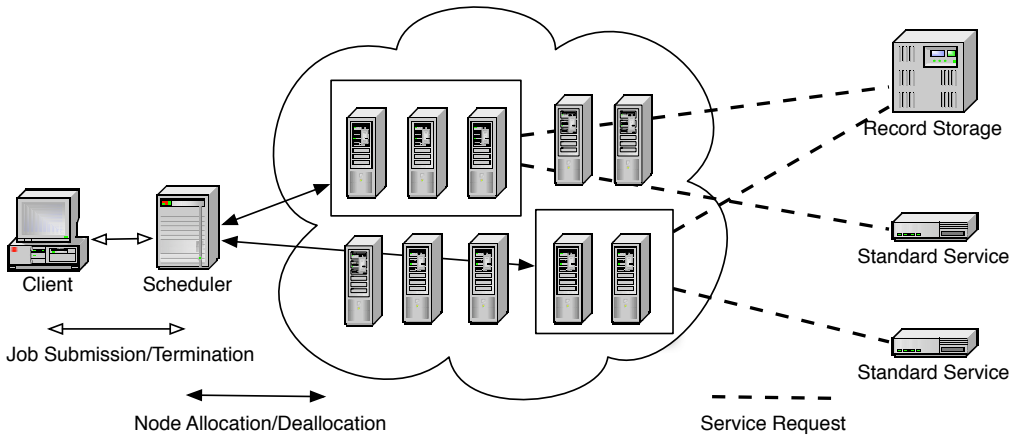

Figure 1. **Work-flow Grid Architecture**

load intensity increased by 50%?

- How many additional grid service nodes are required in order to keep the current SLA if the workload increases by 50%?

- What would be the performance impact if workload class B increased by 50%, but workload class E decreased by 50%?

The simulation model provides valuable information to capacity planners in order to meet future business needs. The model also helps gain a better understanding of the system and the interdependent relationships among the components.

The remainder of the paper is organized as follows. Section 2 gives a description of the target grid environment and characteristics of typical applications. Section 3 describes the performance measurement data and presents the workload characterization. Section 4 outlines the simulation model, its design, implementation, and validation. Section 6 presents the prediction results of several simulation scenarios. Section 7 concludes the paper and outlines future work.

2 Target Grid Environment

The target environment is an enterprise production grid at Acxiom Corporation. In this particular architecture, the physical location of the grid nodes is transparent to the clients and to their grid applications. Applications are run on geographically distributed sites for load balancing and fault tolerance purposes.

The particular grid under study executes work-flow jobs which exhibit a variety of characteristics. A typical work-flow consists of acquiring a large set of records from a database service, performing a series of record manipulations, and creating a new persistent record dataset. Several of the record manipulation steps require standard grid services such as format transformation, record cleanup/consolidation, and shared storage. A work-flow job may be multi-threaded to improve throughput, increase parallelism, and reduce latency. Each job requires an initial number of grid nodes, either specified by the user or determined by the system. The job holds these nodes throughout its execution and releases them back to the free-node pool upon job completion. Whenever a job arrives and enough nodes exist in the free-node pool, the scheduler allocates the required number of nodes to the job. Otherwise, the job queues. When new nodes become available, the scheduler seeks to schedule jobs in the waiting queue according to a specified queueing discipline.

The work-flow grid environment is modeled as a three-tier environment, as shown in Figure 1. The outermost tier handles job submission and job completion; the users submit jobs through a job submission service which monitors the job execution status. In the middle tier, the scheduler queues the jobs and schedules them according to the availability of grid resources (i.e., nodes) and the queueing discipline employed. Node allocation to arriving jobs and node deallocation to departing jobs are treated as grid services. The innermost tier represents the standard service layer where many applications share a variety of services, such as storage, record access, data transformations, persistent archiving, and report generation. Therefore, software/service conflicts occur in this tier. The specific services in this innermost tier are not explicated modeled in the current study. Rather, they are treated as simple black box services and will be modeled in more detail in future studies.

3 Workload Characterization

In order to predict system performance, a representative workload is needed as input to the simulation model. The

workload information on the work-flow grid is available from the job accounting log. For each work-flow job, this log records the submission time, start time, completion time, number of execution threads, number of nodes required, number of records processed, job type, and other job specific attributes. Other accounting logs record node level performance data. This data is available for modeling the innermost tier of the system architecture.

Job characteristics in the accounting log exhibit a wide range of behaviors. They demonstrate diverse arrival times, execution times, nodes required, records processed, and the types of resources used. A preliminary workload characterization study analyzed various jobs attributes to obtain a global view of the workload. Figure 2 shows the number of nodes required by jobs submitted on a typical day in September, 2005. Most of the jobs are submitted

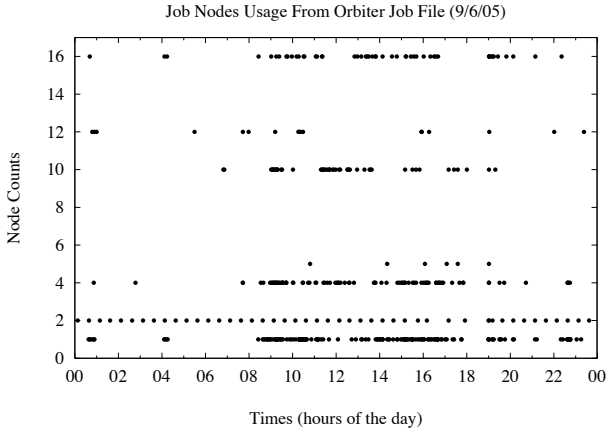

Figure 2. **Number of Nodes Required by Work-flow Jobs**

during daytime hours (8am–8pm), with node requirements ranging from 1 to 16 nodes. The scatter plot of job lengths on a log scale is shown in Figure 3. Most jobs execute in

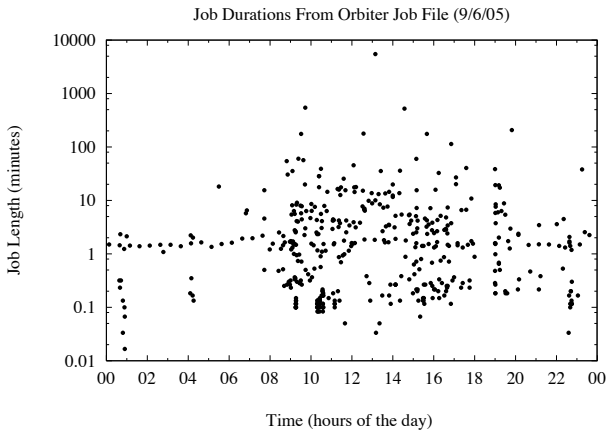

Figure 3. **Job Lengths on Log Scale**

less than 50 minutes. However, there are also a not insignificant number of long, multi-hour jobs that consume significant grid resources that are also submitted during daytime hours. Figure 4 shows the number of active jobs in the system as a function of time. The number of free nodes over

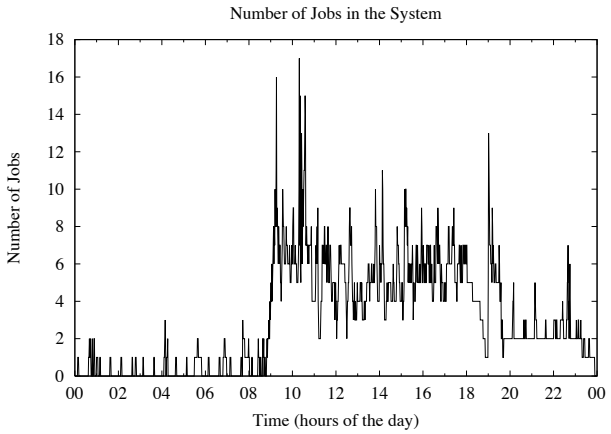

Figure 4. **Number of Jobs in the System Over Time**

time in the grid node pool is shown in Figure 5. Together, these figures provide an overview of the systems workload and its utilization. The workload of the system consists of jobs of different sizes in terms of number of records processed, number of nodes required, and execution time. The workload is too diverse to be modeled effectively with a single job class. Five features from the job accounting log are selected and used to categorize the jobs into homogeneous job classes. The five features include: 1) number of processors, 2) number of records processed, 3) elapsed time, 4) total processor time, and 5) resource type. The k-means clustering technique [5] is used to place the individual jobs into job classes. The number of clusters is determined by plotting a tightness of fit value as a function of the number of clusters and observing the knee of the curve. That is, k-means clustering is used to find k job classes for various values of k. Each clustering also yields a measure of how good the k classes matches the observed data. By looking at various values of k, the one that best describes the data is selected. Since the k-means clustering algorithm is sensitive to the initial starting point values, a detailed analysis is conducted by comparing a total of 14 starting point methods. The 14 methods are categorized into two groups: actual sample data initial starting points and synthetic initial starting points. The best and most robust technique is found to be the one that uses synthetic starting points with scrambled midpoints of the five job features. Details are given in [5]. Using this technique, five job classes are identified as best describing the actual workload. Based on this workload characterization, a workload model is built with five job classes and is used as input for the simulation model. The job classes are independent of each other and each job class has three attributes: arrival rate, service rates, and average number of required nodes. The individual job class arrival rates and service rates are represented by their mean values and their

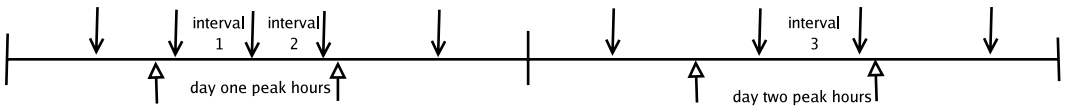

Figure 5. **Peak Hour Average Inter-arrival Time Calculation**

distributions. The arrival rate of a job class can be approximated by its measured system throughput over a relatively long measurement interval [6]. However, since we are primarily interested in heavy loading situations, peak hour arrival rates and service rates are used to parameterize the workload model. The average arrival rate is the inverse of the average inter-arrival time. Figure 5 illustrates the inter-arrival time calculation for peak hours where only intervals 1, 2, and 3 are counted. Each down pointing arrow represents a job arrival. For each job class, arrivals during peak hours from the measurement data are selected and intervals within the peak-hour period are averaged. The resulting arrival rate is then assumed to be negative exponentially distributed, based on actual measurement observations as seen in Figure 7. (However, as described in Section 4, if other distribution functions are found to better describe the workload, they can easily be inserted.) The average service rate is the inverse of the average service time and is directly available from the measured elapsed-time of the peak-hour jobs. To summarize, Table 1 shows the five job classes identified, their inter-arrival rates, service rate, and nodes required.

For simulating various what-if scenarios, an intensity factor is associated with each job class. The base-line average arrival rates shown in Table 1 are multiplied by the intensity factor to yield the average arrival rate used by the simulation. For example, if twice as many job class A arrivals are to be simulated, the intensity factor of job class A is set to 2.

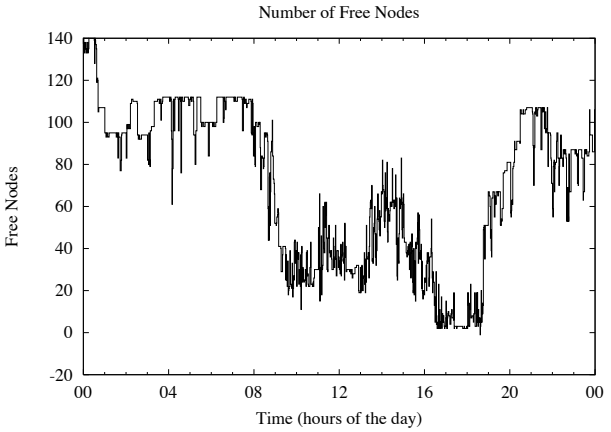

Figure 6. **Free Nodes Over Time**

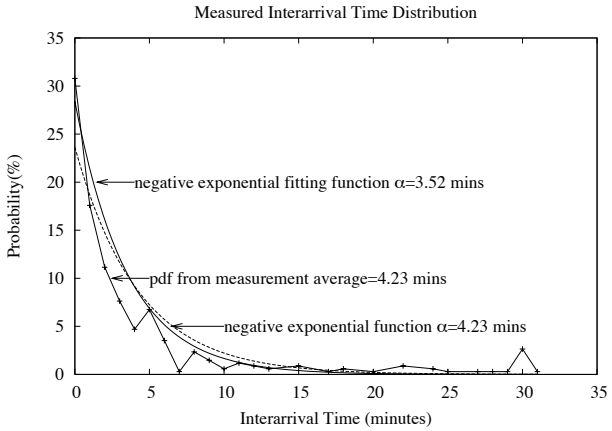

Figure 7. **Interarrial Time Distribution for a Job Class**

Classes	Arrival rate jobs/hour	Service Rate jobs/hour	Nodes Required
class A	23.72	12.63	2.07
class B	5.66	7.03	14.16
class C	8.01	4.83	6.23
class D	1.41	0.792	11.06
class E	0.0907	0.0707	16.0

Table 1. **Five job classes during peak hours (8am - 6pm)**

4 Simulation Model

A dynamic, discrete-time, discrete-state simulation model is used to model the target grid environment. Figure 8 shows a conceptual diagram of the simulator. The model is implemented using a Java simulation package JavaSim [8]. Interested readers are referred to [7] and the JavaSim user manual for detailed description of the simulation model components. In the model, each job represents a work-flow job in the real system. Each job has an arrival time, a required number of nodes, and an execution time. An arrival process representing each particular job class generates each job. These processes create jobs according to the average arrival rate and arrival rate distribution described by the job class. Each job requires a specified number of nodes which is unique for each job class. The scheduler is represented by a separate simulation process that is responsible for scheduling jobs in job queue. Based on the available nodes in the work-flow grid and the scheduling policy, a decision is made on each job. If the system has enough available nodes, the job is started on the required number of nodes. Otherwise, the job will queue. Whenever the system state changes (i.e., upon job termination) , the scheduler seeks to schedule as many jobs as possible. The scheduling discipline is specified as a parameter of the system model in the simulator. First Come First Served with back-fill (FCFS-fillin) was demonstrated to give lower wait and response time than first fit or shortest job first [9]. Two scheduling discipline are simulated and compared in this study, Strict First Come First Served (SFCFS) and FCFS-fillin. Under SFCFS, if job X arrives to the system before job Y, job X is granted its required number of nodes first and begins execution before job Y, regardless of the number of free nodes that are available at job arrival time. Under FCFS-fillin, if job X cannot be initiated due to an insufficient number of free nodes, but job Y requires fewer nodes and can be initiated, then job Y is initiated before job X. Thus, improved node utilization is possible under FCFS-fillin, but starvation is a risk. Each node is also modeled as a separate simulation process. If a node has a job assigned to it, it remains dedicated to the job for an amount of time equal to the service time specification of the job. Otherwise, if a node is not assigned to a job, it idles awaiting allocation by the scheduler process. When a node finishes processing a job, it disassociates itself with the job and releases itself back to the free node pool. During the simulation, system state information and performance statistics are collected. For example, the number of free nodes, the number of jobs in the system, the average queueing delay of each job class, the average response time of each job class, the 95th percentile queueing delay of each job class, the 95th percentile response time of each job class, the overall average queueing delay/response time, and 95th percentile queueing delay/response time are recorded.

An XML schema defines the input and output file formats. The file is used to provide input parameters to the simulator, and to specify the desired output simulation results (i.e., performance metrics). The input parameters to the simulator include parameters to configure the simulator:

- **simulation time:** amount of simulated time
- **node pool size:** number of nodes in the grid
- **queueing discipline:** job scheduling policy, either SFCFS or FCFS-fillin

The input parameters also provide the workload model specifications for each job class (i.e., average arrival rate, arrival rate distribution, average service time, service time distribution, and average number of nodes required). The output metrics include the average and 95th percentiles of queue lengths and response times. In addition to the output metrics for each individual job class, overall output metrics are also reported.

The XML schema not only makes the interface to the simulation model clean, it also provides a flexible input and output description. For example, the number of job classes can be changed easily by editing the file. Individual, or an entire suite of, simulations can be specified in the input XML file. With this XML schema, the simulator is easily linked with other software components. For example a customized XML graphical user interface can easily import the XML schema and extract the desired simulation outputs for display to the user.

5 Model Validation

The simulation program is designed to run using either a trace file as input or statistically generated workload. Comparing the simulated results against actual measurements validates the system simulation model. The event traces used are the job arrival times in the job accounting table. The wall-clock time of job arrivals is mapped to simulation time so that the simulation results can be compared against the measurements.

As shown in Figure 9, the simulated number of executing jobs in the system approximates the measurement data. The simulation closely matches the measurements before 8:00am and after 8:00pm, and reasonably matches the spike at 4:00pm. The discrepancies before and after the 4:00pm spike expose the unmodeled time required by the node allocation/deallocation grid service. The cause and the length of these overhead delays are still under investigation.

6 Prediction Study

The goal of the simulation study is to predict the minimum number of nodes necessary to meet certain performance SLAs, given a specified workload. The parameters for making the predictions are the workload parameters, the queueing discipline, and the SLAs. Two queueing disciplines, SFCFS and FCFS-fillin are studied in detail.

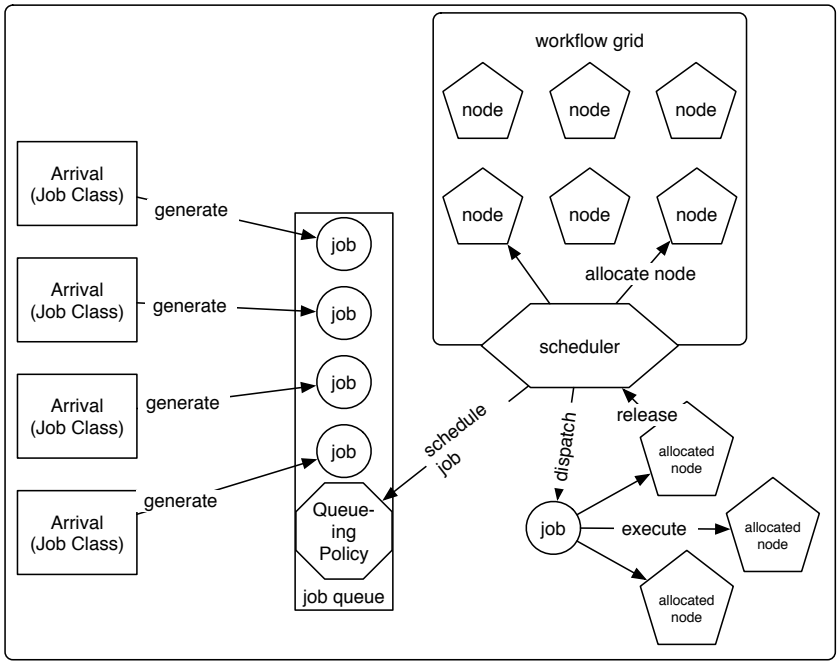

Figure 8. **Simulator Conceptual Diagram**

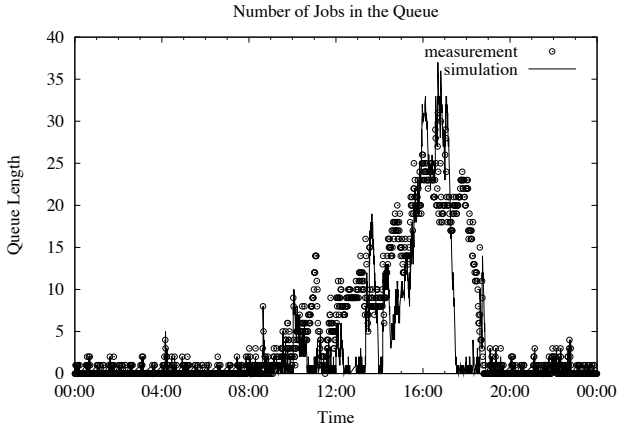

Figure 9. **Queue Length Over Time in Measurement and Simulation**

Simulation experiments are conducted using the five job classes from the workload characterization. First, the simulation is run using the baseline workload parameters. Figure 10 shows the average queueing delay in minutes of the five jobs classes as the node pool size changes from 60 to 130 nodes when SFCFS is employed. With SFCFS, all job classes incur similar average queueing delay because when a job cannot be scheduled, all subsequent arriving jobs must wait. Figure 11 reproduces the results in Figure 10, but under FCFS-fillin. As shown, and as expected, FCFS-fillin improves system performance significantly. Although a slight performance penalty is experienced by Class E jobs (i.e., those jobs requiring 16 nodes), all other job classes benefit from FCFS-fillin.

Second, the effect of increasing the job intensity (i.e., more frequent job arrivals) on queuing delays is tested. Figure 12 shows the average queueing delays of the five jobs classes when their intensity factors increases at the same time from 10% to 100% using FCFS-fillin. The average queueing delays of all job classes increase uniformly and dramatically as their intensities increase. An average queueing delay SLA of 15 minutes is achievable even when all job arrival rates increase by 100%.

However, average queueing delays can be misleading. Many short queueing delays can skew the results by concealing long queueing delays when averages are computed. Since abnormally long queueing delays are significant, the 95th percentile of queueing delays are also studied under varying workloads. Figure 13 shows the 95th percentiles as the job intensities increase under FCFS-fillin. As expected, this graph predicts much longer delays, up to 100

minutes for class B and class E jobs, when the load intensity doubles.

In the above simulations, the node pool size is given as a system parameter. In order to find the minimum pool size required to achieve a given SLA, the simulation is executed iteratively, varying the node pool size. Two types of SLA are considered. One is defined in terms of average queueing delay and the other is defined in terms of 95th percentile queueing delay. Figure 14 shows the number of nodes required to achieve an SLA requirement of 10 minutes for the average queueing delay and to achieve an SLA requirement of 10 minutes for the 95th percentile for all job classes. As indicated, approximately 10 to 20 additional nodes are required to satisfy an SLA for 95% of the jobs instead of for the average job. Figure 15 indicates similar results when the SLA target is reduced to 5 minutes. Such results give the capacity planners a good idea on the size of the node pool required for certain loading conditions.

Individual job class intensities are also varied to study their effects on overall system performance. Figures 16-20 show the average queueing delay of each job classes while individual job class intensities increase. Figures 16 and 18 show that intensity increases on small jobs has minimal effect on larger job classes. The smaller jobs require relatively fewer nodes and require less time to execute as shown in Table 1. They are easily scheduled and finish quickly. Therefore the intensity increase of smaller jobs has little impact on the performance of larger jobs. Thus, smaller jobs can be accommodated without significantly affecting overall system performance. However, as the intensity factor of the larger jobs (i.e., job classes B, D, and E) increases, the performance impact on all jobs is more significant. This is indicated in Figures 17, 19, and 20.

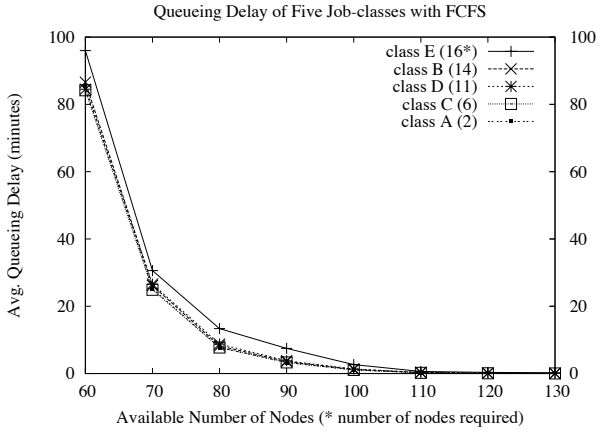

Figure 10. **Average Queueing Delay for 5 Job Classes Using SFCFS**

7 Conclusions and Future Work

In this study, simulation based performance models are constructed for an enterprise grid. The simulation model

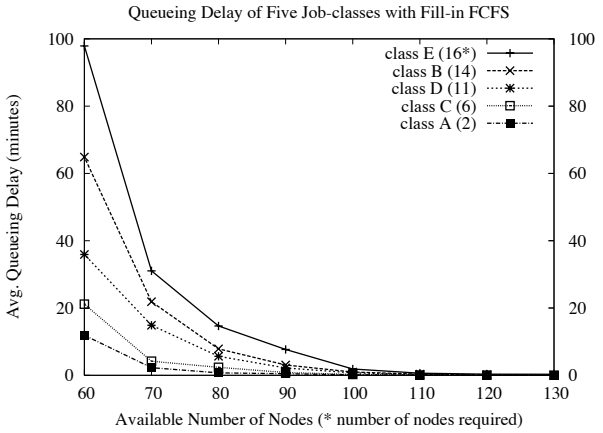

Figure 11. **Average Queueing Delay for 5 Job Classes Using FCFS-fillin**

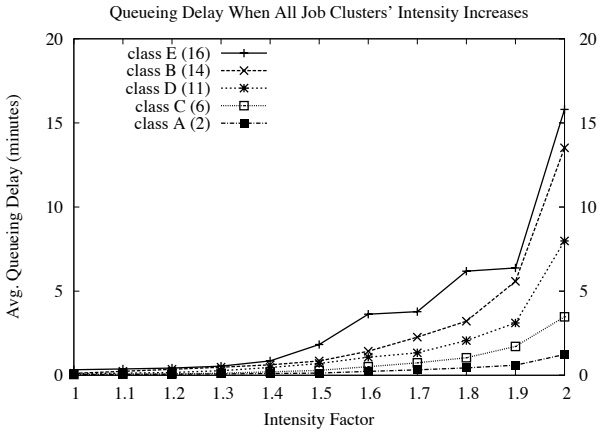

Figure 12. **Queueing Delays When Job Intensities Increase in 10% steps, FCFS-fillin, 120 nodes**:

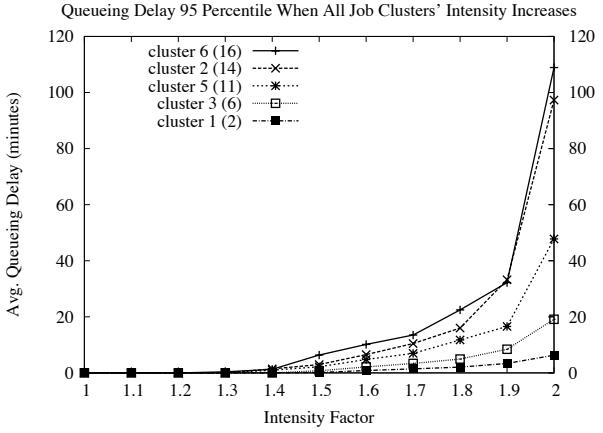

Figure 13. **Queueing Delay 95 Percentile, FCFS-fillin, 120 Nodes**

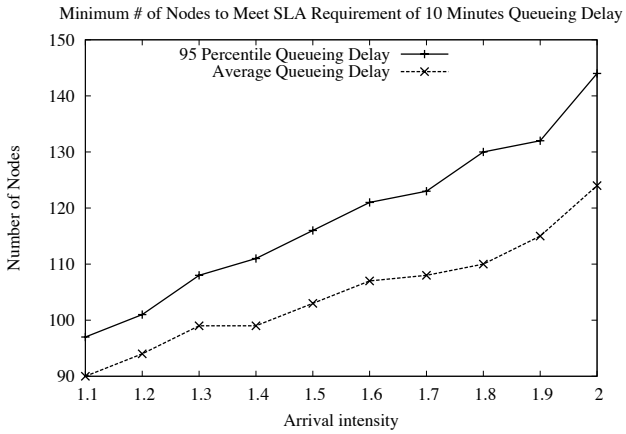

Figure 14. **Minimum Nodes Required to Not Exceed SLA of 10 Minutes**

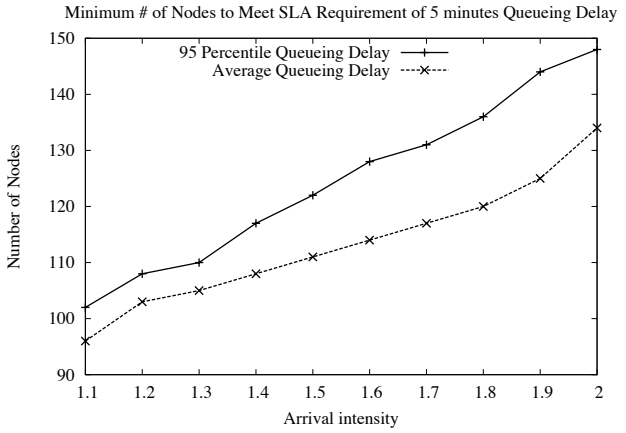

Figure 15. **Minimum Nodes Required to Not Exceed SLA of 5 Minutes**

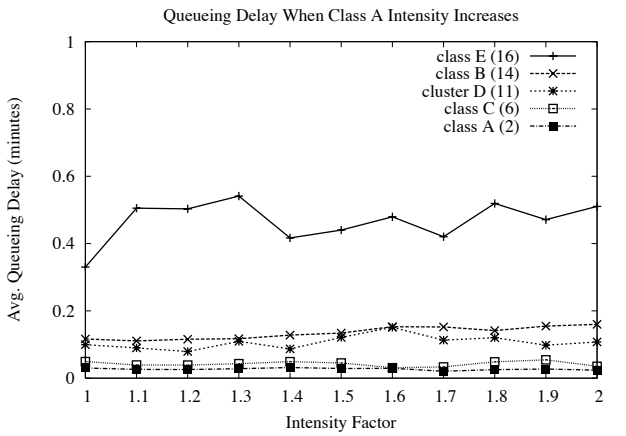

Figure 16. **Average Queueing Delay While Increasing Job Class A Intensity**

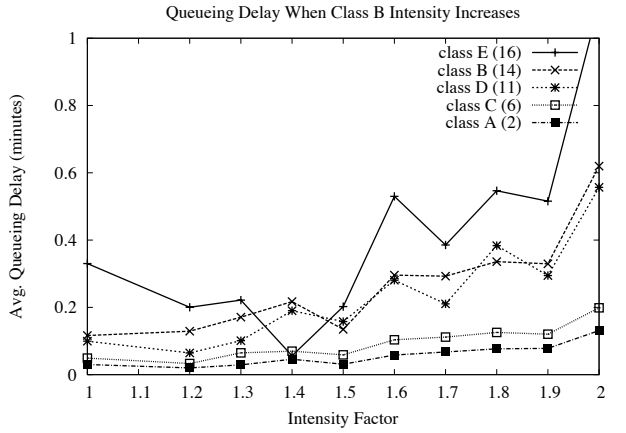

Figure 17. **Average Queueing Delay While Increasing Job Class B Intensity, 120 Nodes**

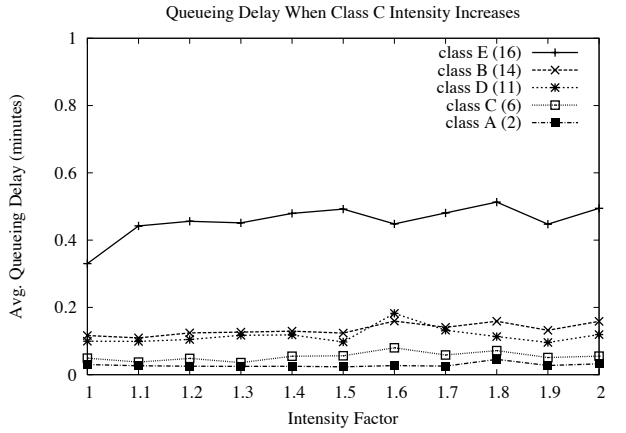

Figure 18. **Average Queueing Delay While Increasing Job Class C Intensity, 120 Nodes**

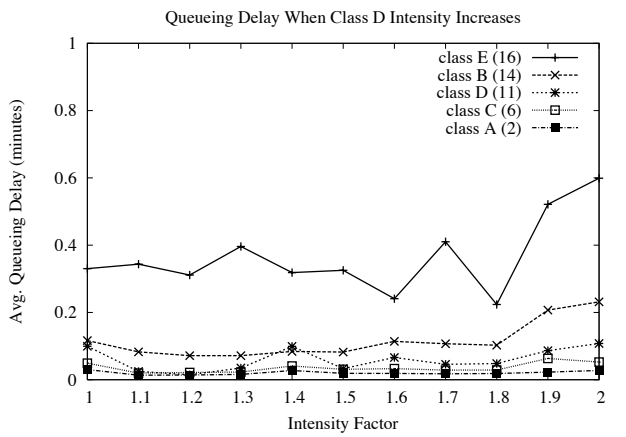

Figure 19. **Average Queueing Delay While Increasing Job Class D Intensity, 120 Nodes**

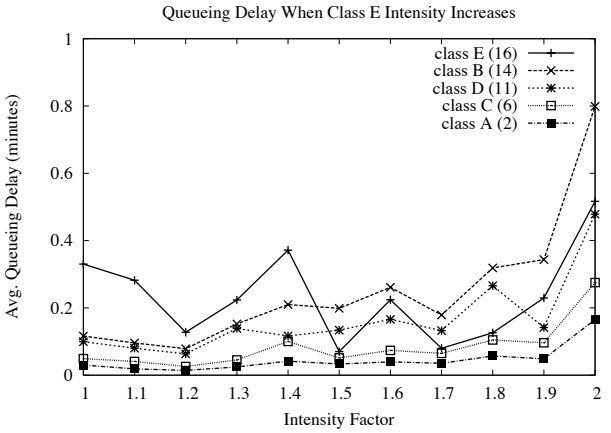

Figure 20. **Average Queueing Delay While Increasing Job Class E Intensity, 120 Nodes**

is validated against measurement data. The system workload is analyzed in detail by examining the job accounting logs spanning several months of data. The jobs are characterized into five workload classes using k-means analysis. Based on this workload characterization, various prediction scenarios are studied via simulation. The simulations indicate that the intensity increase of larger jobs has a significant impact on performance (i.e., queueing delay) of all job classes. This is true even when the larger jobs comprise a relatively small fraction of the overall workload. Alternatively, smaller jobs can pass through the system relatively unaffected, even as their intensity levels and as the number of system nodes change significantly. The simulator can also be used effectively to find the necessary minimum number of grid nodes required to satisfy a specified target SLA. Such tools are useful to system analysts and capacity planners.

Many aspects of the simulation can be expanded. The workload characterization can be improved to better reflect more diverse real world workloads. Node allocation and deallocation delays can be incorporated to make the trace-driven simulation better mimic the real measurement observations. Modeling the shared services aspect of the grid (i.e., the innermost standard service tier) will improve the overall simulation model. Application of the simulation tool to other grid environments is also under investigation.

References

[1] A. Snavely and G. Chun and H. Casanova and R.F. Wijngaart and M.A. Frumkin, Benchmarks for grid computing: a review of ongoing efforts and future directions, *SIGMETRICS Perform. Eval. Rev.*, 30(4), 2003, 27-32.

[2] B. Lu and M. Tinker and A. Apon and D. Hoffman and L. Dowdy, Adaptive Automatic Grid Reconfiguration Using Workload Phase Identification, *Proc. the First Int. Conf. on e-Science and Grid Computing*, Melbourne, Australia, 2005, 172-180.

[3] S. Pllana and T. Fahringer, Performance Prophet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs, *34th Int. Conf. on Parallel Processing Workshops*, Oslo, Norway, 2005, 509-516.

[4] R. Buyya and M.M. Murshed, GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing, *Concurrency and Computation: Practice and Experience*, 14(13-15), 2002, 1175-1220.

[5] F. Robinson and A. Apon and D. Brewer and L. Dowdy and D. Hoffman and B. Lu, Initial Starting Point Analysis for K-Means Clustering: A Case Study, *Proc. 2006 Conf. on Applied Research in Information Technology*, Conway, AR, USA, 2006.

[6] D.A. Menascé and V.A. Almeida and L.W. Dowdy, *Capacity planning and performance modeling: from mainframes to client-server systems*, (New Jersey: Prentice-Hall Inc, 1994).

[7] I. Mitrani, *Simulation Techniques for Discrete Event Systems*, (New York: Cambridge University Press, 1982).

[8] M.C. Little, *JavaSim User's Guide*, (Department of Computing Science, Computing Laboratory, The University of Newcastle upon Tyne, 1999), http://javasim.ncl.ac.uk/manual/javasim.pdf.

[9] H. Shan and L. Oliker and R. Biswas, Job Superscheduler Architecture and Performance in Computational Grid Environments, *Proc. of the 2003 ACM/IEEE SC2003 conf. on High Performance Networking and Computing*, Phoenix, AZ, USA, 2003, 44.