5-2011

ANALYZING SUPERCOMPUTER UTILIZATION UNDER QUEUING WITH A PRIORITY FORMULA AND A STRICT BACKFILL POLICY

Michael David Vanderlan
mvanderl@utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Michael David Vanderlan entitled "ANALYZING SUPERCOMPUTER UTILIZATION UNDER QUEUING WITH A PRIORITY FORMULA AND A STRICT BACKFILL POLICY." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Industrial Engineering.

Joseph H. Wilck, Major Professor

We have read this thesis and recommend its acceptance:

Xueping Li, Rapinder Sawhney

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting a thesis written by Michael David Vanderlan entitled “Analyzing Supercomputer Utilization under Queuing with a Priority Formula and a Strict Backfill Policy.” I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Industrial Engineering.

Joseph H. Wilck, IV, Major Professor

We have read this thesis
and recommend its acceptance:

Xueping Li

Rapinder Sawhney

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

ANALYZING SUPERCOMPUTER UTILIZATION UNDER QUEUING WITH A PRIORITY FORMULA AND A STRICT BACKFILL POLICY

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Michael David Vanderlan
May 2011

ACKNOWLEDGEMENTS

The author would like to thank Mark Fahey and the Oak Ridge National Laboratory for allowing the author to investigate the research problem addressed in this thesis. The author would also like to thank Dr. Joseph H. Wilck, IV for guidance and the structuring of the research and the countless hours of questions and revisions. The author would like to take this opportunity to thank Dr. Xueping Li for his guidance along the way. None of this would have been possible without the course on simulation and access to Arena software. The author would like to again thank Dr. Mark Fahey for his insight of the subject matter and providing the raw data. The author would also like to thank the Department of Industrial and Information Engineering for their continued support in courses and the software licenses that keep the students on the leading edge of technology and applications. The author would like to thank Jonathan R. Celso for his input and knowledge.

The author would also like to thank family and friends for sticking with him through his education, especially his wife for the financial support and time spent without him.

ABSTRACT

Supercomputers have become increasingly important in recent years due to the growing amount of data available and the increasing demand for quicker results in the scientific community. Since supercomputers carry a high cost to build and maintain, efficiency becomes more important to the owners, administrators, and users of these supercomputers. One important factor in determining the efficiency of a supercomputer is the scheduling of jobs that are submitted by users of the system. Previous work has dealt with optimizing the schedule on the system's end while the users are blinded from the process. The work presented in this thesis investigates a scheduling system that is implemented at the Oak Ridge National Laboratory (ORNL) supercomputer Kraken with a backfilling policy and attempts to outline the optimal methods from the user's point of view in the scheduling system, along with using a simulation approach to optimize the priority formula. Normally the user has no idea which scheduling algorithms are used, but the users at ORNL not only know how the scheduling works but they can also view the current activity of the system. This gives an advantage to the users who are willing to benefit from this knowledge by utilizing some elementary game theory to optimize their strategies. The results will show a benefit to both the users, since they will be able to process their jobs sooner, and the system, since it will better utilized with little expense to the administrators, through competition.

Queuing models and simulation have been well studied in almost all relevant aspects of the modern world. Higher efficiency is the goal of many researchers in several different fields; the supercomputer queues are no different. Efficient use of the resources makes the system administrator pleased while benefiting the users with more timely results. Studying these queuing models through simulation should help all parties involved by increasing utilization. The simulation will be validated and the utilization improvement will be measured and

reported. User defined formulas will be developed for future users to help maximize utilization and minimize wait times.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I
INTRODUCTION

This thesis discusses methods of utilizing supercomputer cores on Kraken at the Oak Ridge National Laboratory (ORNL) through backfilling as well as an analysis of a time refund policy that ORNL has implemented recently in the event that the supercomputer crashes while performing one or more jobs. Specifically, the author has examined strategies that users can utilize in order to minimize the time they may have to wait in a queue in order to have their jobs completed as quickly as possible. The supercomputer has a weekly scheduled downtime for maintenance every Wednesday morning. The scheduled maintenance gives the operators two distinct approaches based on the utilization currently within the system. As soon as the supercomputer begins to accept jobs after its routine maintenance, the user should request the maximum amount of cores in order to increase his priority. But when time approaches the scheduled maintenance period, then the user should take the opportunity to backfill, which allows a job that was submitted later than another job gets to run first because of availability of cores, by asking for no more than the amount of available cores at a given point in time in order to maximize efficiency of the supercomputer nodes.

Queuing models have been around since the early 1950's (Kendall, 1953), although the scope and depth of models have changed over the years to include almost anything that is a wait and service model. The basic design aspects of queuing models are all the same. A customer, unit, job, car, or widget enters a system in what is generally called the "arrival". From there, the queue defines how the system will handle all arrivals through what is considered the "queuing discipline" (Bose, 2002). Once the item is in for service then the service times are determined by the system, and once the job is finished it exits the system. To increase efficiency of these systems, queuing models have been developed for most scenarios so that the queuing disciplines can be optimized for that specific system based on a deterministic set of data. The common queuing

disciplines that the everyday human would be involved in are the First-In First-Out (FIFO) design which is the basis for almost all queues seen in a normal daily routine. A FIFO queue can be seen at supermarkets for each line individual line (not taking into account parallel servers), fast food restaurant drive-thrus, and almost any service line. Research in the area of queue wait time fairness was described in a paper by Sabin, Kochhar, and Sadayappan in which they gave weighting factors to prevent an extreme delay of one job but allowed jobs to get skipped to improve the overall time (Sabin, Kochhar, and Sadayappan, 2004).

Other common, but less widely used models are the Last-In First-Out (LIFO), processor sharing, and priority models. LIFO is used in manufacturing and any process that has a bin. For example, a manufacturer has a bucket containing a particular part that is refilled when it gets low and the next needed part is taken from the top. Processor sharing is common in high demand environments such as supermarkets or bank tellers. The first in may not be the first out depending on service time, but everyone in this system should experience similar delays. Priority models will be the primary model discussed, because they are the primary model used in a computer queuing system. An example of a priority based model is a hospital emergency room, a principle known in the field as "triage". Certain patients will get a higher priority based on their specific needs. If everyone had the same needs then the priority queue reduces to a FIFO queue. Priority queues can be used efficiently because they can set aside the users' needs to better serve the queue as a whole and have a better utilization of resources. The priority queue algorithm discussed in this paper has a distinct advantage in that some users will not recognize delay and other users will not be as concerned about possible delays. This allows for the adjustment and optimization of this priority queue so that it will optimize the usage of the system, therefore resulting in satisfied system administrators.

Business Plan/Justification

There are many ideas and reasons for addressing the utilization within these supercomputer systems. With supercomputer power and financing on the rise, this is a financial obligation as much as it is a performance measure. With the grant of $65 million from the National Science Foundation (NSF) to ORNL, it is easy to see that the 700 million CPU hours per year on Kraken do not come at a cheap price (HPCwire.com, 2008). Not only are there costs of computation to consider, but the problems being solved by Kraken are important to today's research areas. Everything from government security issues to climate models to fusion energy calculations are being performed on this system (HPCwire.com, 2008).

The primary goal for this thesis is to improve utilization of the CPU hours on the Kraken system. Kraken, being the first academic petaflop computer (UTK Web News, 2010), is one of the largest computers that many people have access to. This helps Kraken achieve maximum utilization but may also hurt the utilization based on the differences in the job structures that are submitted to the system. To find the importance of utilization, one does not have to look much further than the importance of some of the problems being solved on Kraken. With the Department of Energy or the Office of Homeland Security, it is easy to see that getting an answer back quickly can be important and necessary. The other reason to focus on utilization is the cost described earlier. Node hours are not cheap and anytime a core is idle it can be considered a wasted node hour.

Utilization will be examined in two ways, first from the user point of view and second from the system point of view. The user metrics are slightly different than the system utilization metrics but will serve the same function. Users of Kraken, if they choose, can see the system state and queue at any given time. This view of the system will allow the users to formulate their jobs' key attributes to

maximize their chances of getting their jobs run faster. Not only does this help the users who understand their benefits but in the long run should benefit the system because of the users understanding of how to tighten up the jobs in the system. The approaches discussed in this thesis should allow users to better understand and formulate their job submissions better. From the system perspective it is all about formulating the right priority function. The priority function is the driver behind how jobs get into the system and how they are stacked in the queue. This thesis will address the priority formula through simulation in *Arena* and the software add-on *OptQuest*, which will optimize the variables in the formula through replications and design of experiment (DOE).

An understanding of the system dynamics and controls should allow the users to achieve faster results with little effort and also increase system utilization. This thesis will show a possible instant increase in utilization of 0.092% which equals almost 16,000 node hours. This increase could be higher if the users deploy a more optimal strategy with submitting their jobs. Overall, these increases should produce more jobs finished in less time and this will show more value to the $65 million spent on the development of the system.

The basis behind this work can be used in many different areas as an effective method for analyzing priority within a system. Any priority system which uses a specific algorithm and has both capacity and time constraints could follow a similar solution approach. Common systems would include manufacturing systems, airline priorities, and other computer systems. All of these systems have a capacity constraint and have time dependant customers or jobs. Using the same approach presented here in the thesis it would show how users can optimize their strategies and may provide a method for the system administrators to adjust their priority algorithms for better utilization.

The rest of this thesis is organized as follows: a literature review will be discussed in Chapter 2. Chapter 3 will explore how jobs are given priority before and after they have been submitted, along with a payoff and job check pointing analysis. Chapter 4 will address the simulation model that is used to optimize the system. Chapter 5 will analyze the results. Chapter 6 of this thesis discusses conclusions and future research involving this problem. Chapter 7 will be a summary business plan.

CHAPTER II
LITERATURE REVIEW

Previous literature reveals that there has been some research related to backfilling, but there are limited specifics that are relevant to the situation at ORNL. Most of the literature addresses the operators' queuing system in an attempt to organize the jobs for maximum efficiency under a predetermined set of rules. For the purpose of this paper, the operators have already established the set rules that the user must take advantage of in order to minimize wait time. A review of literature on different queue structures will help us identify an adequate approach to the problem.

Queuing has been studied for the better part of a century now. Little proved what is now Little's Law in 1961, in which he proves the long run equilibrium of a system can be modeled with $L = \lambda W$ (Little, 1961), where L is the expected number of units in the system, W is the expected time spent by a unit in the system, and $1/\lambda$ is the expected time between two consecutive arrivals. This equation is used to model queuing systems so that the system can be modeled without having to find all necessary information. For instance, if it is easier for the engineer to find the number of units in the system, then it is to measure the expected time in system.

Later in the 1960's, scientists began to study more specific models of the Kendall-Lee notation system. In 1966, Schrage and Miller examined what was one of the first priority formula developments (Schrage and Miller, 1966). Schrage and Miller gave priority to the job in queue that had the least remaining process time (i.e. the smallest job). They also used pre-emption to queue the highest priority job should it not fit into the process as it currently was. They appear to be one of the first to realize the benefits of sorting the queued jobs for the advantage of reducing system wait times.

In 2005, Ernemann, Krogmann, Lepping, and Yahyapour published a paper that listed the scheduling methods for the top 50 supercomputers from November 2003 (Ernemann, Krogmann, Lepping, and Yahyapour, 2005). From the list they surveyed the top 50 machine owners to see what scheduling algorithms they used on their machines along with several other utilization important parameters. This survey showed the extremes of the supercomputer world of the time. Some of the systems were for government use only, therefore they had no priority formula or backfilling allowed and provided very little information while other more public computers shared that they had a scheduling priority and did use a backfilling technique.

Smith, Foster, and Taylor addressed the issue of advanced reservations on systems (Smith, Foster, and Taylor, 2000). They proposed that some jobs need several parallel systems available at the same time. To do this they would have to have some advanced reservation period on all of the machines. These reservations cause delays on other jobs that are queued depending on the size and durations of the reservations. The authors made some lofty assumptions by stating, “the best performance is achieved when we assume that applications can be terminated and restarted, backfilling is performed and relatively accurate run-time predictions are used” (Smith, Foster, and Taylor, 2000). Two of these are not going to work for the case presented in this thesis. Jobs on Kraken can not be terminated and restarted nor do the users provide accurate run-time predictions. Margo, Yoshimoto, Kovatch, and Andrews give a much more unbiased few of the impact of reservation scheduling (Margo, Yoshimoto, Kovatch, and Andrews, 2007). They show the impacts of several case studies which show that utilization and wait time are both negatively affected but “that these effects can be mitigated with appropriate policies” (Margo, Yoshimoto, Kovatch, and Andrews, 2007).

The game theory approach presented in this thesis will show users how the can increase their outcome by understanding other users positions. The system administrators would like to see this because in the long run an optimal equilibrium might be attainable. In 1985, a paper was published that took the queuing model and showed how equilibrium could be achieved by using bribery (Lui, 1985). This paper shows how in the long run a queue in which the server can use bribery will reach the optimal state for all users such as a Nash equilibrium. Chun and Culler also had the users' interests in mind by creating user-centric performance metrics for their analysis of cluster batch schedulers (Chun, and Culler, 2002). Chun and Culler used their metrics to rate the users' value in using different market-based cluster batch schedulers. Both of these papers addressed the issue of the users' value and how the system might use these values to their advantage.

Nurmi, Brevik, and Wolski developed software that predicted the queue time for jobs so that the user could have a better estimation of their delay. QBETS (Queue Bounds Estimation from Time Series), would give users an accurate estimation of the queue so that they would not be slowed by having to wait aimlessly for their jobs to run or have their job finished when they weren't ready to come back to the results.

Tsafrir uses a modeling approach to explain how users' inaccurate estimates of runtime eliminate opportunities for backfilling (Tsafrir, 2010). Active competitive ORNL users do not have to be concerned about these inaccuracies because since the users can see how many cores are available to them at a given time, they will eventually develop a competitive advantage with accurate estimates. This is not always the case though, and many users of the system simply do not care enough or are too conservative of their estimates, and therefore request unnecessarily large amounts of time. For these examples, Tsafrir is correct but

in the case of ORNL this might make backfilling easier if the time window is larger.

Snell, Clement, and Jackson discuss the backfill computation problem by allowing the administrator to manipulate some of the job requirements to prevent a large blockage of jobs that are waiting to enter the system (Snell, Clement, and Jackson, 2002). Efficiency has been improved under this freedom to manipulate jobs, but the paper fails to address a scheduling equation for efficient supercomputer usage. Furthermore, these methods might not be as beneficial if the user has hundreds or thousands of jobs to examine, which is where the priority formula and scheduling rules would benefit from the competition involving users, therefore using several minds to produce an advantage to the system.

Ward, Mahood, and West approach the scheduling equation, which usually considers a strict rule on backfilling, where jobs that can be run without delaying the highest priority job in the queue can be run immediately upon submission (Ward, Mahood, and West, 2002). Backfilled jobs have to be able to run fast enough so that they do not delay the entrance of the next top priority job. Ward, Mahood, and West introduce a new factor to the scheduling equation referred to as a "weight of allowable delay". They argue that average wait time can be reduced with only a slight allowable delay of major jobs from backfilled jobs. This is the most recent research to the priority formula and game theory strategies delivered in this paper. Instead of approaching the relaxation of the backfill policy, the competition from the users should optimize the usage.

Tsafrir, Etsion, and Feitelson approach scheduling by using backfilling and a first in, first out (FIFO) queue (Tsafrir, Etsion, and Feitelson, 2007). But they use a system-estimated runtime instead of a user-estimated runtime in an attempt to have better accuracy. They show that inaccurate estimates do reduce efficiency, and their system of system-estimated time is more beneficial. However, they are

taking away the competitive advantage that is used in this thesis. In contrast, the user estimated times in the system will be shown to be inaccurate, therefore causing some utilization loss.

These papers have shown that the user should have the ability to maximize their results based on given priority formulas therefore creating a competitive approach to scheduling with the competitive advantage going to the most accurate and opportunistic users. This competition should increase efficiency by utilizing users' needs and desires to have their results as fast as possible.

CHAPTER III
USING THE SYSTEM EFFECTIVELY THROUGH COMPETITION

Priority Formula

When a job is submitted to a supercomputer at ORNL, it is assigned a priority score based on the following priority function, which may be revised as ORNL learns more about user behaviors:

$$P(t) = S\left[Q \times q(t) + X\left(1 + q(t)/r\right)\right] + R\left(C \times c\right) + Z \times z \qquad (1)$$

where:

- *P(t)* = Total priority at time *t*
- *S* = Service weight, which is equal to 1 at ORNL
- *Q* = Queue time weight, which is equal to 5 at ORNL
- *q(t)* = Queue time of job at time *t* in minutes
- *X* = Expansion factor weight, which is equal to 100 at ORNL
- *r* = Requested wall-time limit in minutes, where the maximum is 24 hours and the minimum is 1 minute
- *R* = Resource weight, which is equal to 1 at ORNL
- *C* = Processor count weight, which is equal to 1 at ORNL
- *c* = Number of cores requested, which is a number between 1 and approximately 98,000
- *Z* = QOS (Quality of Service) weight, which is equal to 100 at ORNL
- *z* = Job QOS priority, which is equal to 0 at ORNL.

After substituting some of the variables in the priority function with values predetermined by ORNL, the priority function becomes the following:

$$P(t) = \left(5 + 100/r\right)q(t) + c + 100 \qquad (2)$$

A reasonable assumption made by the author is to model *q(t)* as a linear function with a zero intercept and a slope of 1. In other words, let *q(t)* = *t* unless otherwise noted. Thus, the priority function that will be discussed in this paper is equation (3), which can be seen below:

$$P(t) = \left(5 + 100/r\right) \cdot t + c + 100 \tag{3}$$

This priority function can be seen in one of two ways. Before the job gets submitted to the supercomputer, the priority function is a payoff to the user since they can request the amount of runtime and cores. The reader should note that if the user requests more cores, they will get a higher priority score before submission. Also, if the user requests less runtime, they will get a higher priority score after submission since *t* = 0 at the time of submission. Also, the higher the priority a user achieves for their job, the greater the chance their job has of getting backfilled into the supercomputer.

Requesting More Cores

Suppose that two jobs are being considered for submission with the same amount of runtime requested, *r*. However, one job requests *c* cores, and the other job requests for ($c + \Delta c$) cores where $\Delta c > 0$. If both jobs were to be submitted at the same time, the job that requests more cores will have a higher priority. So, how long does the job with fewer requested cores need to sit in the queue in order to have the same priority as the job with more requested cores? It can be shown that the time needed in the queue, t^*, for the job with fewer requested cores is given by the equation:

$$t^* = \frac{\Delta c}{(5 + 100/r)} \tag{4}$$

For an illustration of the above concept, consider Figure 1, a graph of the two priority functions where one asks for c cores at the very beginning and the other asks for ($c + \Delta c$) cores after t^* passes.

At ORNL, users are able to view how many cores are available at any given time, so they can use this information to implement the optimal strategy of asking for the maximum amount of cores.

To illustrate the current effect of requesting more cores we can look at previous data to show how, in Figure 2, requesting more cores can effect the wait time of the job.

This shows how requesting more cores will increase your wait time as a general rule. There are three issues that are occurring on Figure 2 that can be explained for a better representation. First, the second category of requested CPU's (24-48 CPU's), has a higher average than the next higher category. This is due to the same set of users submitting large numbers of smaller jobs and then having to wait because each user can only have five jobs running at a single given time unless the machine is idle and the queue has been emptied except for their jobs. Second is the significantly smaller wait time average for the large to full size jobs. This is explainable because of the knowledge of the system that previous users have. Since there is a weekly maintenance period, the users know that the only good chance they have of getting a full size job to run is just after this down time. Therefore, users submit their full size jobs just before this restart time. Third is the median difference from the mean. The mean is susceptible to outliers while the median is not. This just shows that there are extremely large wait times for all the categories and that no category is free from the possibility of long wait times.

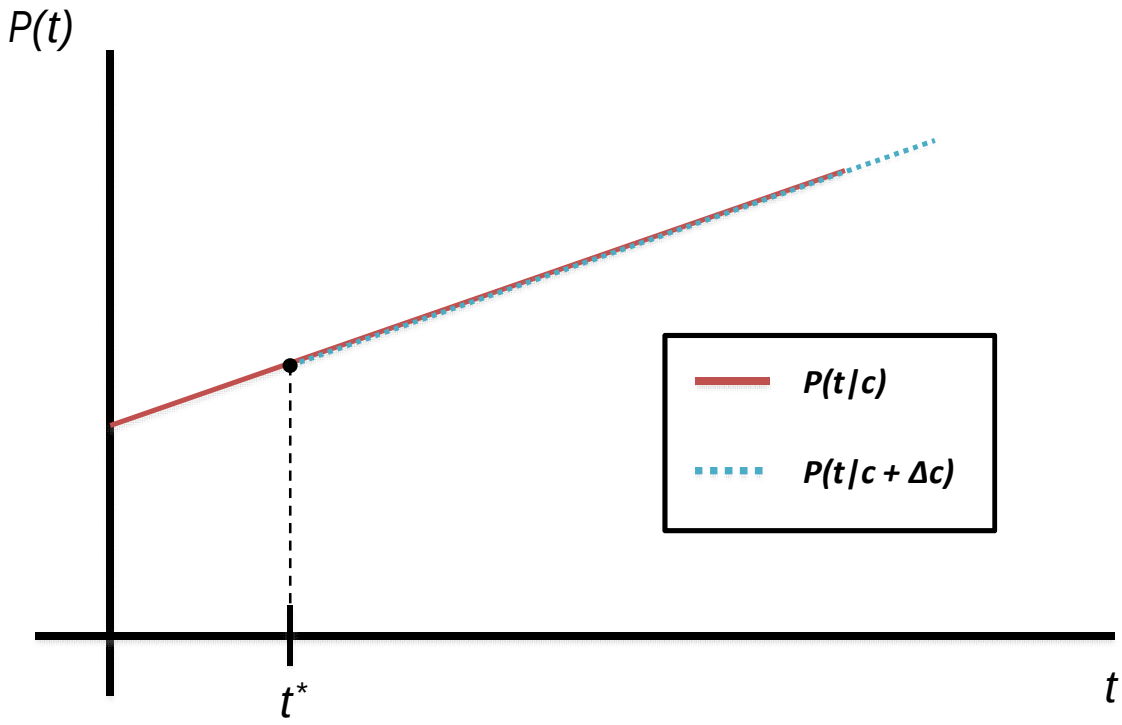

Figure 1: An illustration of the benefit when requesting more cores

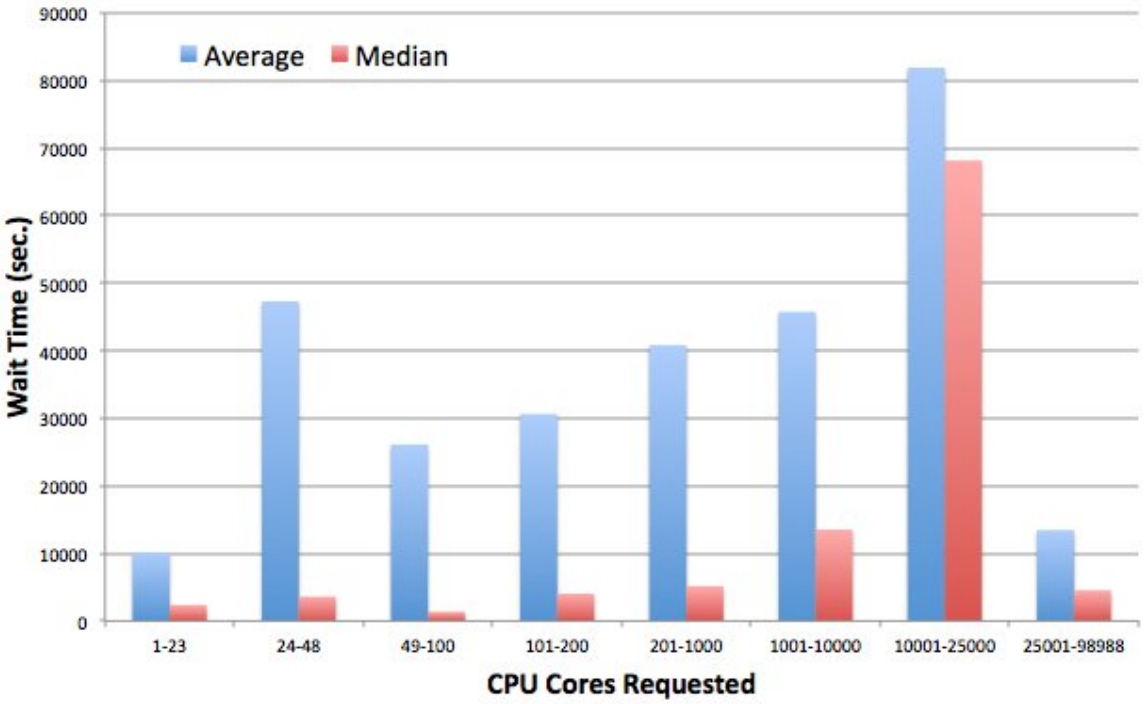

Figure 2: A graph showing the effects of CPU's requested against the average and median wait times.

Requesting Less Runtime

Let's now consider the benefit of asking for less runtime. Suppose that two jobs request the same number of cores, c, with one job requesting r runtime and the other requesting $(r - \Delta r)$ runtime, where $\Delta r > 0$. After submission, the job that asks for less runtime will always have a higher priority. But what if the inferior job were to be submitted first at time $t_o > 0$? How much time will elapse, t^*, before the superior job will have equal priority to the inferior job? It can be shown that

$$t^* = \left(\frac{(5r + 100)(r - \Delta r)}{100 \Delta r} \right) t_o \qquad (5)$$

For an illustration of the above concept, consider Figure 3, a graph of the two priority functions where one asks for r runtime at t_o and the other asks for $(r - \Delta r)$ runtime after t^* passes.

To illustrate the current effects of requesting longer runtimes, Figure 4 shows the effects of requesting longer runtimes against the wait time of the job.

Figure 4 shows similar trends to Figure 2. Users requesting really short jobs tend to have many jobs that wait because of the 5 jobs per user rule. Also, the average wait times again are susceptible to outliers while the median is not.

Along with wait time as it is relates to requested time, user requested time error is an issue worth noting. Tsafrir discussed in two papers (Tsafrir, Etsion, and Feitelson, 2007 and Tsafrir, 2010) the importance of accuracy of user estimated runtime and the effects that the errors had on the performance. While it has been shown to be beneficial to request less run time as it relates to wait time, the users of Kraken do not seem to recognize this benefit yet or have reservations

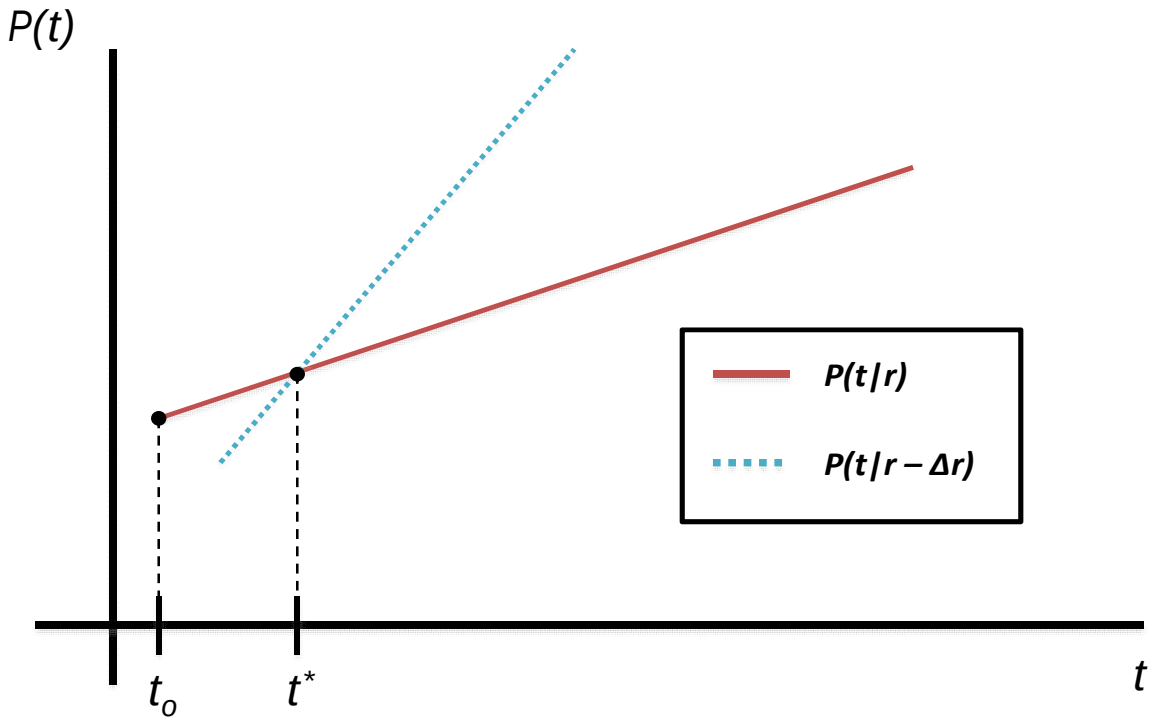

Figure 3: An illustration of the benefit when requesting less runtime

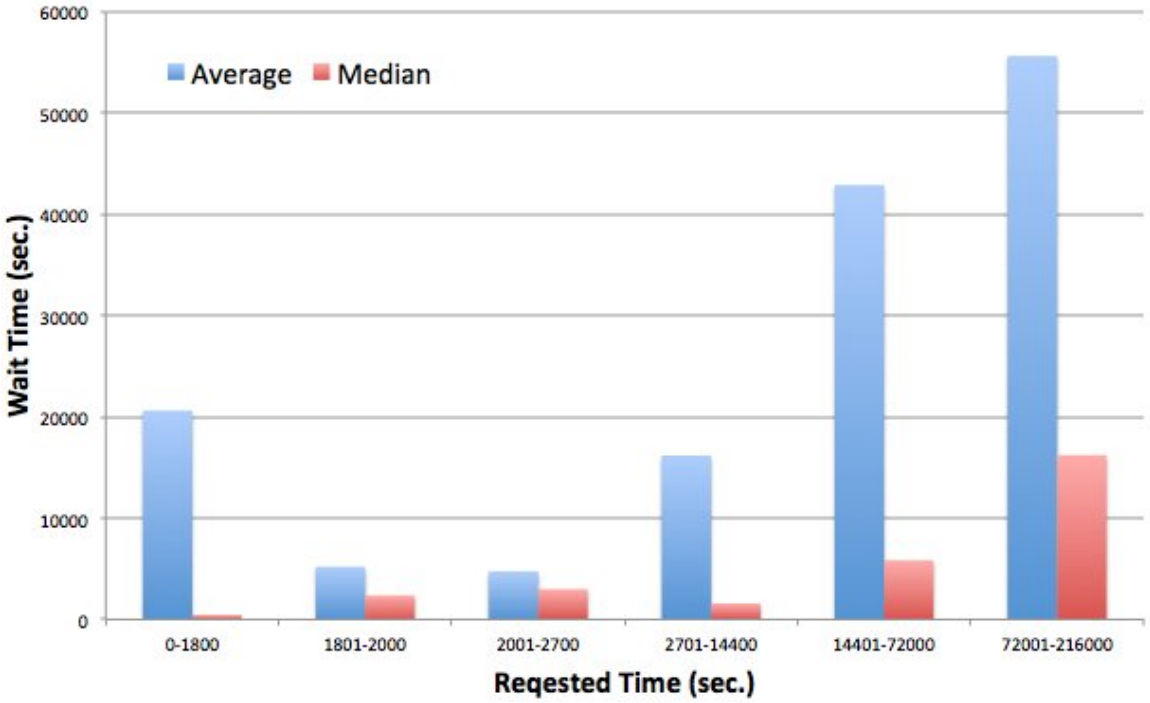

Figure 4: A graph showing the effects of requested time against the average and median wait times.

about becoming less conservative on their requested time estimates. Figure 5 shows how inaccurate these estimates have been in the past.

Figure 5 gives the percent error in requested time as it relates to actual runtimes. A near 100 percent error would be that a user required 1/100 of the time requested for their job to finish. Figure 5 shows that this error does get less as the jobs get larger but even the largest jobs on average request almost twice the time needed to run their job. This may have caused them a substantial delay because the system has to find room for their job based on the requested time without knowing how little time the job will actually require.

Now we shall shift the focus of our discussion over to a policy that ORNL has used in order to reward users who checkpoint their code in case of the event that the system crashes while running the user's code.

Investigation of Refund Policy

Assume that the system will crash at some point in time, and we have a job that takes t_{end} > 6 node-hours to complete. We are considering this set of jobs because jobs that crash on or before 6 node-hours get the time before crashing refunded to them, and it's practically considered a rerunning of the job. However, if a job crashes after the 6-node-hour mark, then they do not get back all of the time that was spent before the crash, just 6 node-hours. With that said, let's consider time in three discrete parts, which are the following:

- From the beginning of the job until and including the 6-node-hour mark.
- The 6-node-hour mark until and including the end of the job.
- After the end of the job.

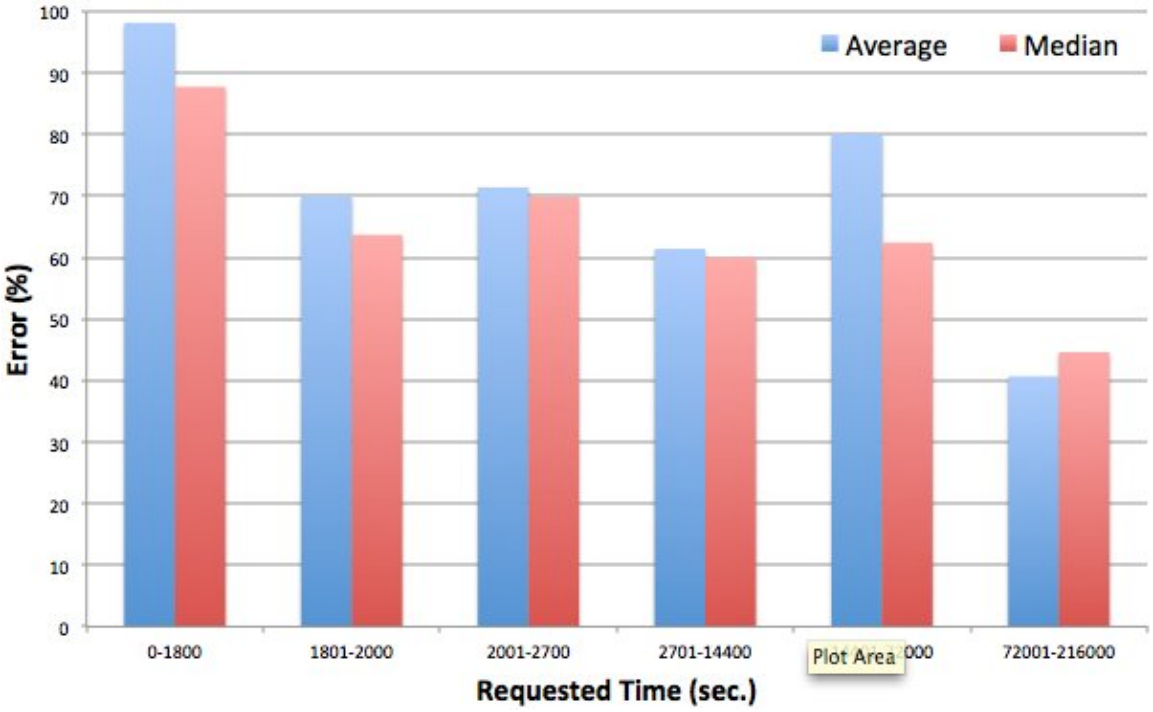

Figure 5: A graph showing the error of user estimated runtime.

Let τ_1 represent the time the system crashes, if it occurred, in the first time period. In other words, $\tau_1 \in [0, 6]$. Following suit, let $\tau_2 \in (6, t_{end})$ and $\tau_3 \in [t_{end}, +\infty)$ be the times at which the system fails, if it occurred, in the other two periods of time that were described above. Now let's consider what happens if there is no checkpointing allowed in the code.

No Checkpoints

If the failure were to occur at τ_1, then the user would initially lose τ_1 for not checkpointing but gain it back because of the policy. Therefore, the net loss would simply be t_{end}. Now what if the failure were to occur at τ_2? Then the user would initially lose τ_2, gain 6 node-hours back because of the policy, and then lose t_{end} for running the code again. Therefore, the net loss is equal to $t_{end} + (\tau_2 - 6)$. If the failure were to occur after the job had completed running, then the user simply loses t_{end}. If we let τ , the time at which the system fails, be described by a Weibull distribution with parameters β and δ and a probability density function of :

$$f(t|\beta,\delta) = \frac{\beta}{\delta}\left(\frac{t}{\delta}\right)^{\beta-1} e^{-(t/\delta)^{\beta}}, \qquad (6)$$

then the expected loss takes the following form:

$$\begin{aligned} &P(\tau \le 6)[t_{end}] + P(6 < \tau < t_{end})[t_{end} + (\tau_2 - 6)] + P(\tau \ge t_{end})[t_{end}] \\ &= [1 - P(6 < \tau < t_{end})][t_{end}] + P(6 < \tau < t_{end})[t_{end} + (\tau_2 - 6)] \end{aligned} \qquad (7)$$

Checkpoints at the 6 Node-Hour Mark

Since the user gets all of their time back if the system crashes before the 6-node-hour mark, then it's risky for the user if the checkpoint ends before the 6-node-

hour mark. Therefore, if someone were to checkpoint once in their code, it would end at the 6-node-hour mark. However, checkpointing takes time, so the total time to run the job is $t_{end} + t_{check}$, where t_{check} is the time needed to set the checkpoint. If $\tau = \tau_1$, then nothing changes from the previous problem. We get our time back and rerun the code for a net loss of $t_{end} + t_{check}$. If $\tau = \tau'_2 \in (6, t_{end} + t_{check})$, then we get 6 node-hours back like we normally would, but only lose $t_{end} + t_{check} - 6$ on the second running of the code. Therefore our net loss is $\tau'_2 - 6 + t_{end} + t_{check} - 6 = \tau'_2 + t_{end} + t_{check} - 12$. Again, if the failure were to occur after the job finishes, then we simply lose $t_{end} + t_{check}$. Thus, our expected loss is now

$$\left[1 - P\left(6 < \tau < t_{end} + t_{check}\right)\right]\left[t_{end} + t_{check}\right] + P\left(6 < \tau < t_{end} + t_{check}\right)\left[\tau'_2 + t_{end} + t_{check} - 12\right] \qquad (8)$$

Checkpoints After the 6 Node-Hour Mark

Here's when things get interesting. Let's suppose that we finish the checkpoint at τ_c, which occurs after the 6-node-hour mark but before the end of the job. Assume that the time it takes to checkpoint is the same as before, t_{check}.

If $\tau'_2 < \tau_c$, then the user loses everything and has to start over again with a net loss of $\tau'_2 - 6 + t_{end} + t_{check}$. However, if the crash were to occur between τ_c and $t_{end} + t_{check}$, then the net loss would simply be $\tau'_2 - 6 + t_{end} + t_{check} - \tau_c$ $= \tau'_2 + t_{end} + t_{check} - (6 + \tau_c)$, which is a greater payout than the far right term of the previous expected loss since $6 < \tau_c < t_{end} + t_{check}$. From the above conclusions our expected loss is the following:

$$\begin{aligned}&\left[1 - P\left(6 < \tau < t_{end} + t_{check}\right)\right]\left[t_{end} + t_{check}\right] + P\left(6 < \tau < \tau_c\right)\left[\tau'_2 + t_{end} + t_{check} - 6\right] \\ &+ P\left(\tau_c \le \tau < t_{end} + t_{check}\right)\left[\tau'_2 + t_{end} + t_{check} - \left(6 + \tau_c\right)\right]\end{aligned} \qquad (9)$$

CHAPTER IV
SIMULATING THE SYSTEM

From the queuing perspective of an administrator, it can not get much better than the queue described in this paper. The administrator not only has access to view all the jobs in the system, but they can also manipulate the priority formula at any time. The priority queue that will be described is the queue of jobs scheduled on Kraken at ORNL. Kraken became the 1st academic Peta-flop system in October 2009 and is currently 8th on the world supercomputer rankings (UTK news, 2010). The priority queue formula has been implemented with a strict backfill policy that will not allow blocking. Backfilling allows smaller jobs to jump over larger ones if they do not slow down the top job in the priority queue. Backfill policy is explained in great detail in the following papers: Tsafir, 2010; Tsafir, Etsion, and Feitelson, 2007; Ward, Mahood, and West, 2002; Snell, Clement, and Jackson, 2002. The current priority formula as discussed earlier is:

$$P(t) = S * \left[Q * q(t) + X * \left(1 + \frac{q(t)}{r} \right) \right] + R * (C * c) + Z * z \qquad (10)$$

The *r* and *c* are the two variables that are entered into the system by the user and cannot be changed after the request is submitted. The value of *q(t)* is a continuously increasing time value that begins with the submission of the job. The weight values *S, Q, X, R, C,* and *Z* are all manipulated by the systems' administrators in an attempt to maximize Kraken utilization and optimize the backfill of jobs. These weighted values will be the primary investigation of this paper because they have been arbitrarily assigned by the system administrators. Although, there obviously was some care taken in balancing the equation values as to not overvalue one criterion over another.

From February 2010 through August 2010 this system, with the current strict backfill policies, ranged between 82-96% utilization. It is the hope of the author that some small re-arrangement of the weight values will narrow this range,

reduce average wait times, and increase utilization over time. The simulation design should allow the user to run a previous data set and test the results of possible priority formula changes, comparing wait times to job size, and overall average wait times.

Theoretical Models

The model that will be developed for this application has the benefit of already being used on the Kraken system. This allows the author to attain an actual, 13 month data set of almost 573,000 jobs that were submitted into the system. Dr. Mark Fahey of the National Institute of Computational Sciences (NICS) and Joint Faculty with the University of Tennessee's Industrial and Information Engineering Department has been kind enough to provide this data set that will be used to verify the model. The data and model will be validated by comparing the actual benchmarking that has been done on the system and comparing the graphs of system usage. From this data set the author will extract the time that the job enters a system and the numbers that will be assigned to that specific job, such as the requested wall-time, requested cores, and for validation, the actual run time of the specific job.

To demonstrate this current system and how adjustment within queues might change the utilization, examples are shown in Figure 6, 7 and 8.

Figure 6 shows just how a queue works that does not allow backfilling, therefore jobs can only run simultaneously if they both fit in order as shown with jobs 4 and 5. This method would be used only if computations within a system can not be accomplished or fairness is a top priority.

Most current scheduling programs view the job queue as a set of two dimensional boxes with the x-axis being requested time, in which a job is either

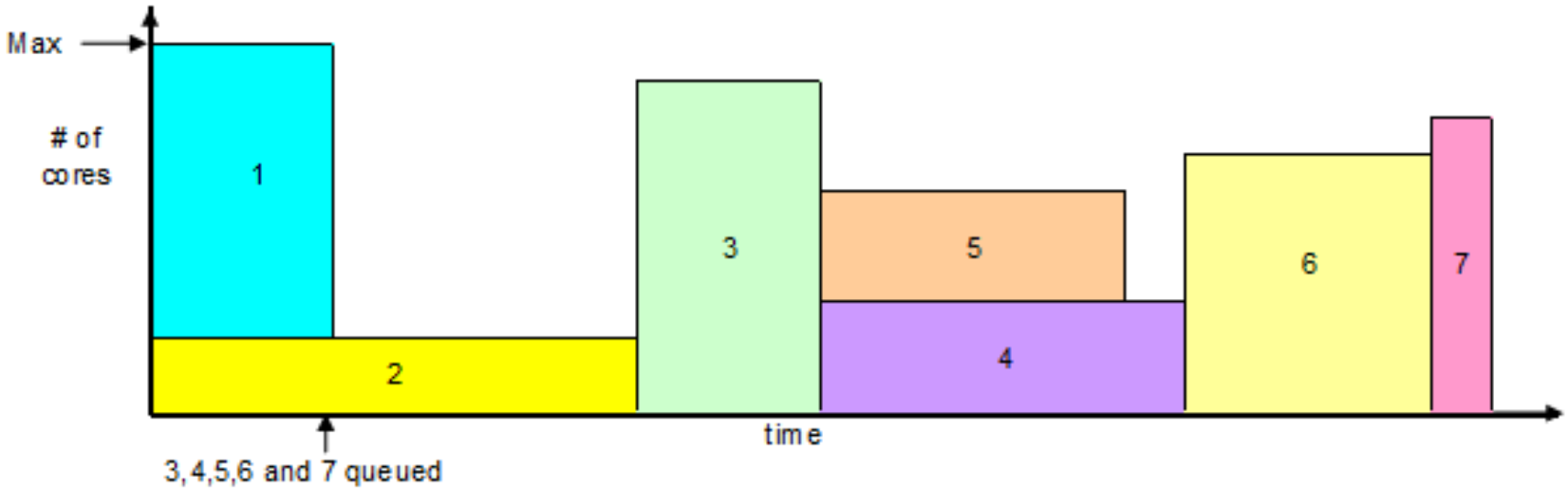

Figure 6: This shows a queue in which there is no backfilling.

short or long, and the y-axis being CPUs requested with the maximum being 98,976 CPUs on Kraken and jobs are considered narrow or wide. In current backfilling systems both the x and y axis' are considered equal value to the jobs waiting to be backfilled. The job that gets backfilled is the first job that fits into the available gap on a FIFO basis as shown in Figure 7.

From the actual priority formula we can analyze how this is different on the Kraken system. The priority formula uses a weighting system that can move jobs around in the queue not just on the wait time but also on the jobs requested time and cores. The formula can be seen as follows in the simplified form:

$$P(t) = \left[5*t + 100*\left(1 + \frac{t}{r}\right)\right] + (1*c) \tag{11}$$

The variables are:

- *P(t)* = total priority at time t
- *Q* = queue time weight = assigned start at 5
- *t* = queue time of job at time *t* in minutes
- *X* = expansion factor weight = assigned start at 100
- *r* = requested time limit in minutes
- *C* = processor count weight = assigned start at 1
- *c* = number of cores requested

So the weighted values of 5, 100, and 1 are the current values that are used in the priority formula. These values adjust the priority of all jobs in the queue based on the time waiting (*t*), a ratio of time waiting to the requested wall-time (*r*), and the request number of cores (*c*). This means that neither the largest job that fits gets in nor that the first job that fits based on arrivals gets in necessarily but rather the first job that fits that has the current highest priority gets in. This is

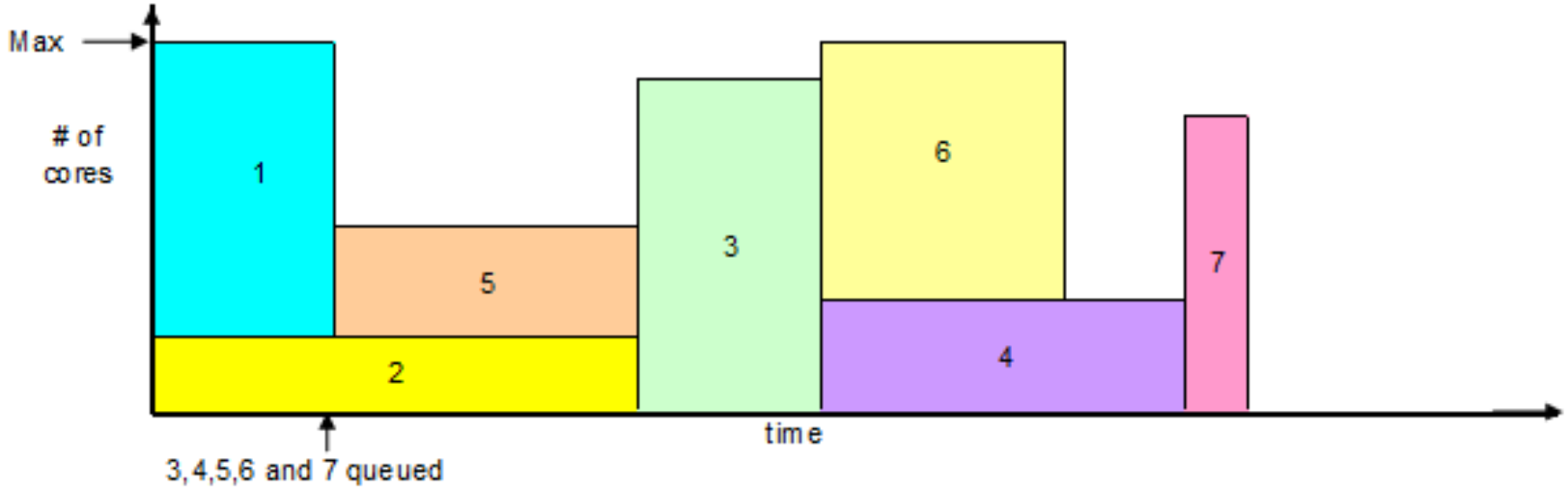

Figure 7: This shows a queue with strict backfilling on a FIFO basis.

shown in Figure 8 using the data in the accompanying Table 1. These somewhat arbitrary variables will be the optimization focus of this simulation model.
To explain Figure 8 using Table 1, when jobs 3 thru 7 are queued at time 3 their priority scores are calculated and job 3 is preempted for time 8 and this allows backfilling if it does not delay job 3. The next highest priority score is job 7 and it fits both time and cores constraints. Job 6 has the next highest score and fits both constraints as well. After job 3 the next two jobs are reordered at time 11 based on their priority scores even though they will both run at the same time.

Simulation Configuration

Based on the data from the data files and using the *Input Analyzer* function on the Arena software it was evident that the only way to produce an actual model was to use the real data set. The arrivals, requested cores, and requested wall-time could not be modeled accurately enough through Arena to produce an accurate representation of the data. Therefore replications of this problem should not be needed because the same data set will be used each time to stay consistent. Fundamentally this will eliminate the need for hypothesis testing since all answers should be exact based on the entered jobs. This means that the model should be an exact replication of the queuing system during the time period in which it is drawn from, October 3, 2009 to November 30, 2010.

Arrivals of all jobs are taken from the data file and inputted into the model. With each arrival the attributes specific to each job are attached to each specific entity. This is shown in Figure 9.

After this step the job enters into the decision tree. The first step is to check availability of the cores on the Kraken system. If the cores needed are available, then the job is supplied to the system, but if there are not enough available cores, then the job enters the priority queue. Once in the queue the second step is to

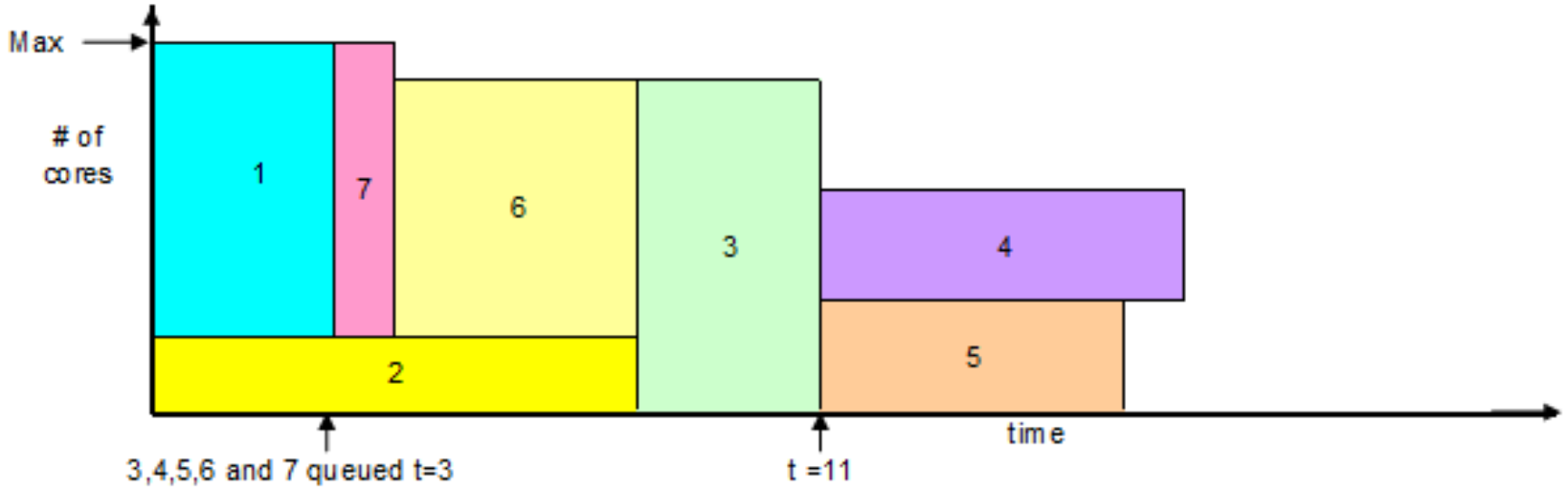

Figure 8: This shows a queue with strict backfilling on a priority formula basis.

Table 1: Priority calculation table for job ordering.

Job Numer	Time Requested	Wait Time	Cores Requested	Priority Score t=3	Priority Score t=11
3	3	n/a	9	109	
4	6	8	3	103	276.3333333
5	5	8	3	103	303
6	4	n/a	7	107	
7	1	n/a	8	108	

Priority Formula = 5t + 100*(1 + t/r) + c

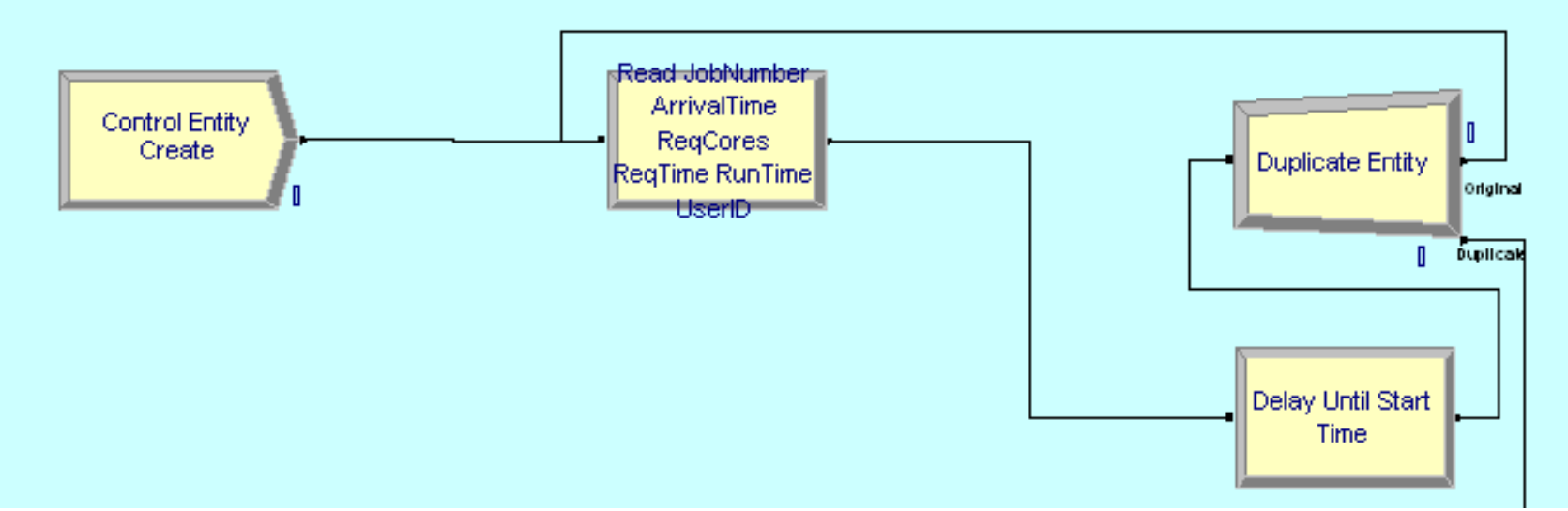

Figure 9: System creation of exact replica of jobs on the Kraken system for a supplied period of time.

wait for computer cores to become available. Once cores become available the next step is to calculate an attribute for each job in the queue, the *P(t)* value. Once this is assigned the wait queue re-orders the jobs and releases them based on priority for entry into the Kraken system. Each time a job leaves the system, meaning cores are available, the waiting queue is recycled to calculate new *P(t)* values and search the queue for the highest priority job that will fit into the available cores. This is modeled in Figure 10.

The actual job runtimes are taken from the data file to create an accurate representation of the system although in real life these values are not known when the job enters the system. This value is just used to create the accurate model because if they were actually known before running the system then this would make a huge advantage in scheduling by allowing the scheduler to fit the jobs using a 2-D optimization algorithm. The actual processing within the system is shown if Figure 11. The process “Supercomputer” is backed by the exact resources within the system, 98,976 cores. If a job takes the maximum amount of time it is terminated and released unfinished.

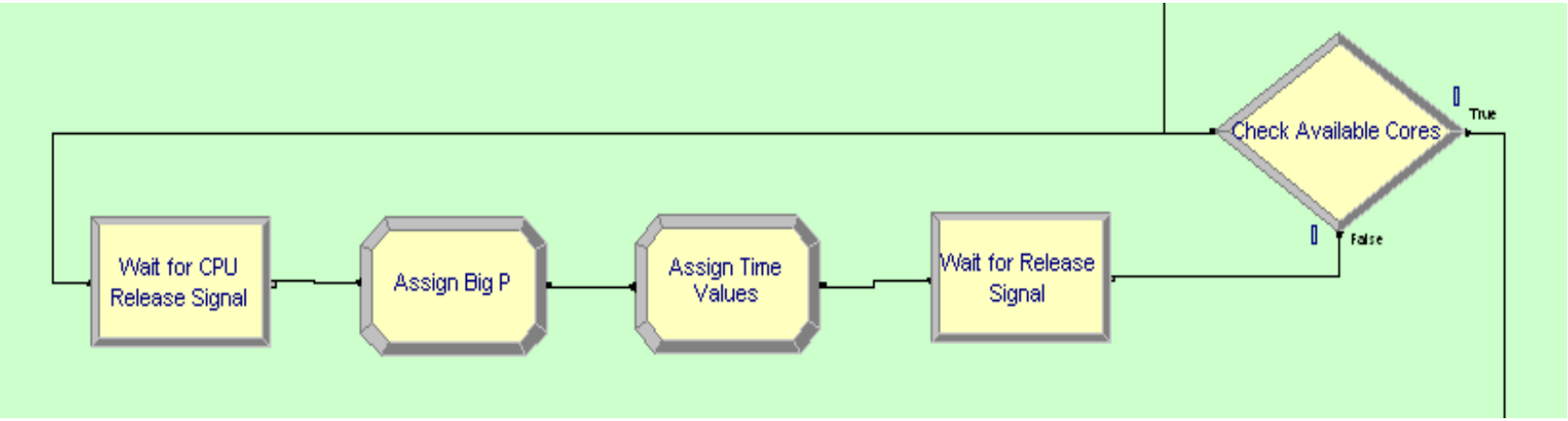

Figure 10: Decision tree and recycling wait queue.

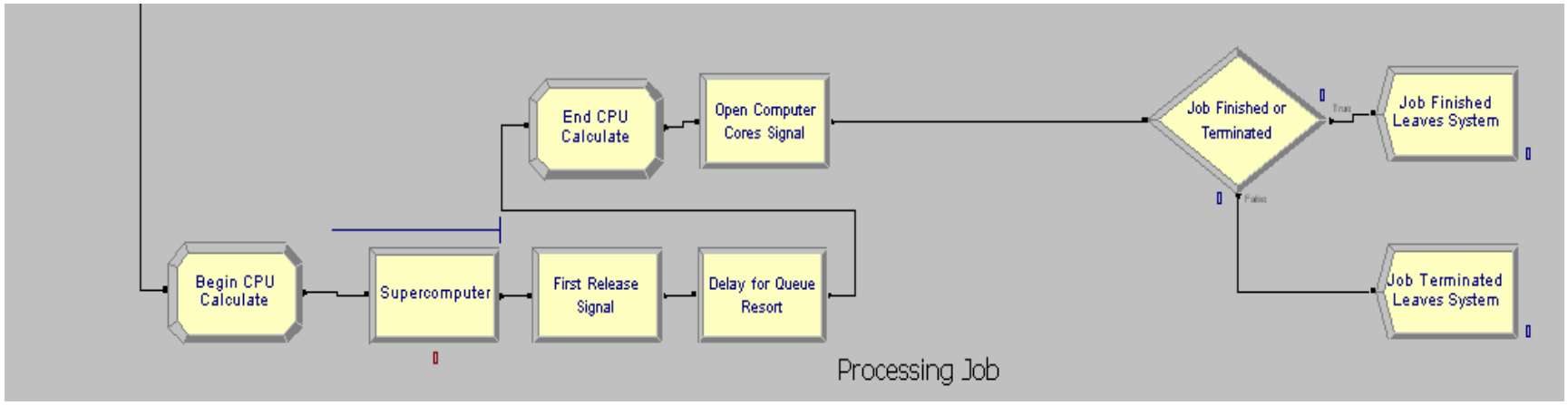

Figure 11: The processing of the jobs on the supercomputer.

CHAPTER V
SIMULATION RESULTS

The simulation results have been interesting. Currently, the system processes the correct amount of jobs for the time periods that is being represented, approximately 8,100 jobs/week. The number that is being verified and in theory is the most important is the utilization. Kraken administrators are currently claiming an 87-96% monthly utilization during the time period used for the model and the model shows 87.66% for the time period of October 13, 2010 through October 20, 2010. To scale down the data set the author has cut the data down to the last 47.5 days of data from October 13, 2010 to November 30, 2010. This data set contains 100,000 jobs. To validate the data set two graphs were developed, both representing CPU Core usage of the first 18,315 jobs which cover the first 336 hours or 2 weeks of the new data set. The graphs shown in Figure 12 and 13 show some remarkable similarities and are used to validate that the data being used is being accurately represented in the model. There are a few minor differences, the first being that the actual data has jobs that were running before it, while the simulation begins with an empty CPU and that first job. The other visible difference is the actual data has a CPU maintenance in which the system jobs are cleared and the usage spikes to zero.

These figures show that the model is close to accurate but the author will discuss later how some future areas of research are currently affecting the outcomes. With the model as is, *OptQuest*, an Arena optimization software package was used to vary the three main parameters, *C, Q and X*, of the priority Formula 11. A large scale was used first to serve as a DOE approach and to make sure all the variables had interaction effects. From there the scaling of the variables was made smaller but retained the balance of the formula. The optimization results are shown in Table 2. From the results an increase of 0.092% CPU utilization can be obtained from the new priority formula values. This may not seem like much but a 0.1% improvement means that: 16,000 additional core hours are

used. That is a large amount of computational power that is now being utilized and there is still room for improvement.

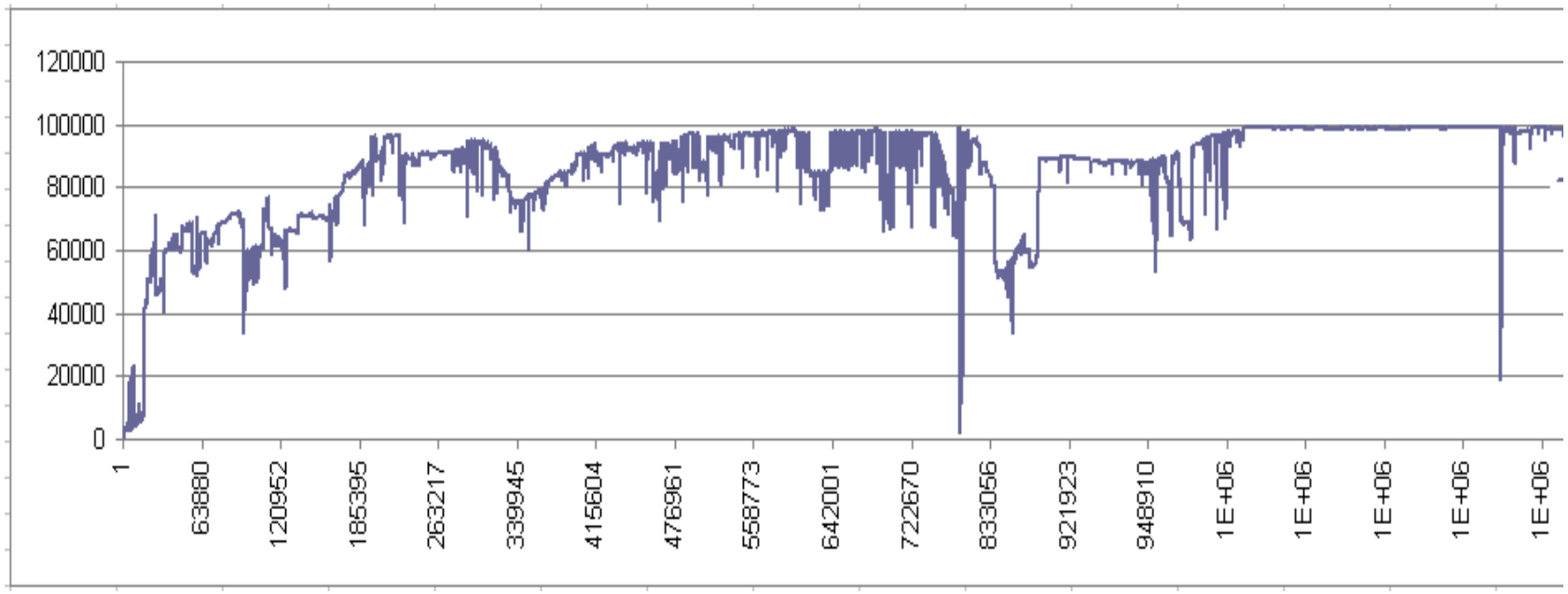

Figure 12: Graph of real CPU usage data from the data file with visibly slower beginning usage due to previous jobs and visible maintenance spike around 80,000 wall clock seconds.

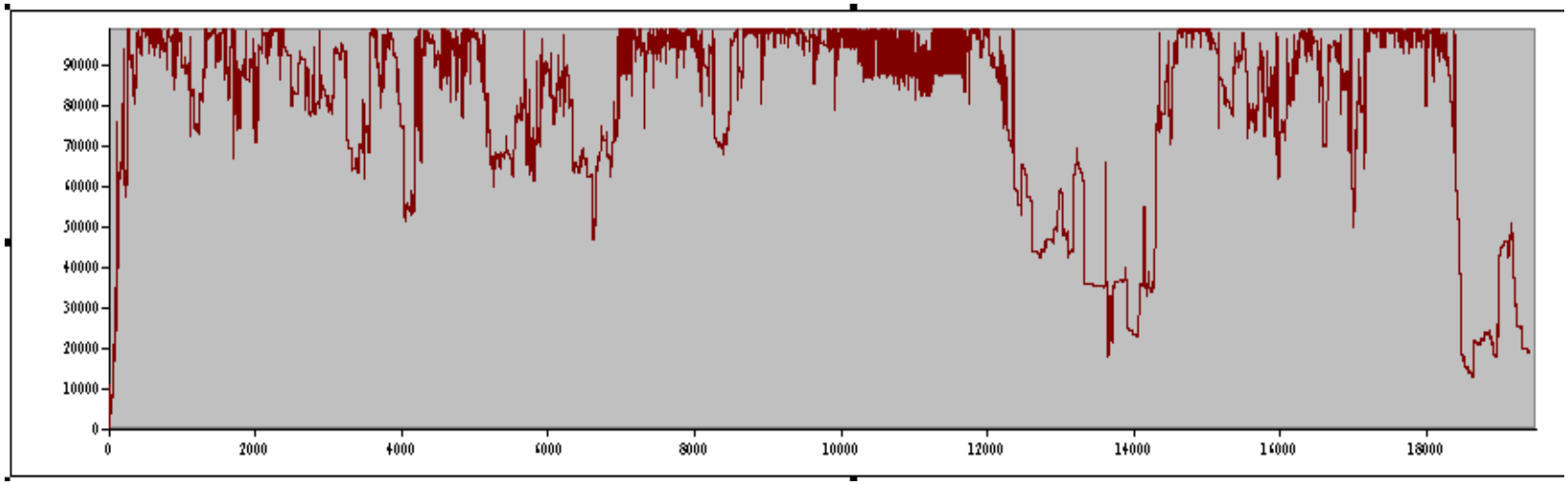

Figure 13: Graph of simulated CPU usage from the Arena simulation model with peaks and valleys similar to the real data.

Table 2: Optimization results of priority function variables.

Best Solutions Found Using OptQuest				
		Function Variables		
Simulation Run	Utilization	Big C	Big Q	Big X
75	0.87752	0	7	120
109	0.87752	0	6	100
134	0.87752	0	3	50
135	0.87752	0	9	150
159	0.877449	5	9	150
61	0.877402	3	7	120
78	0.8774	4	7	130
136	0.877356	4	3	50
151	0.877352	5	8	130
137	0.877352	4	4	70

CHAPTER VI
CONCLUSIONS AND FURTHER RESEARCH

This thesis has demonstrated how competition can be used to improve the efficiency of supercomputers. The users at ORNL will benefit from requesting the maximum amount of available cores when submitting a job as well as requesting a lower runtime, which has been shown in this paper though an examination of the priority function that is used at ORNL. Furthermore, the 6-hour refund policy has been examined at the surface level, and it has been shown that the usage of checkpoints before or after the 6-node-hour-mark depends on the distribution of when the supercomputer has a tendency to crash. There are some questions for future research that the authors would like to explore. Sequence dependent jobs and optimal look ahead strategies can be examined in the context of the current system at ORNL. A further analysis of checkpointing under the 6-node-hour refund policy needs to be explored as well.

Using the model developed, the utilization of the system has been determined and verified to the previous charted data. The use of Arena's optimization tool, *OptQuest*, has shown how the weighting variable in the priority formula can be manipulated to improve efficiency by moving jobs around in the queue to create a better fit in the two dimensional system. Using *OptQuest*, the most efficient priority formula was a tie between 4 of 500 different simulations tested. Each of the top four results had a *C* value of zero: this effectively means that requested cores should not be a part of the priority formula. Therefore the recommended formula would appear as shown in Formula 12. Of course, any of the top four variable sets would provide the same results since the order of the 8,000 plus jobs are probably the same and yield the same utilization.

$$P(t) = \left[7 * t + 120 * \left(1 + \frac{t}{r} \right) \right] \tag{12}$$

This removal of the requested cores can be explained by the amount of different jobs within the system. With an average work in progress (WIP) of 229 jobs during the one week period in the simulation, it is easy to see that finding a job to fit the leftover cores is not a hard job. Getting the time values worked out seems to be the biggest optimization challenge.

Future investigation for research would be to restrict the users' number of jobs in the system. The current system at ORNL only allows a single user to run five jobs simultaneously at any given point unless the system queue is empty and there is adequate space in the processors. This is a future modeling question to tackle to make a more efficient model.

From this research and models it is apparent that more research can be done in the area of backfill scheduling policies. Although there have been many research papers published in this area, it is apparent that until every system can use a single policy development formula to get the optimal result that this topic will still be around. The author hopes that these adjustments in weight variables will result in better utilization but continued monitoring of the system will have to be done, especially since the jobs that are entered into the system will vary so greatly. The author also hopes that this model will be a useful tool for any priority formula scheduling administrator, in that he can manipulate the priority formula for existing data to see what the possible effects of the changes may be.

CHAPTER VII
SUMMARY OF BUSINESS PLAN

By looking at the cost of the Kraken computer system and the importance of the user projects it is evident that utilization of this resource is an important problem. Utilization is the primary metric based on CPU node hours used.

User approaches for submitting their jobs and getting them run quicker:

Requesting more cores:

It can be shown that the time needed in the queue, t^*, for the job with fewer requested cores is equal to

$$t^* = \frac{\Delta c}{(5 + 100/r)} \tag{4}$$

Requesting less runtime:

How much time will elapse, t^*, before the superior job will have equal priority to the inferior job? It can be shown that

$$t^* = \left(\frac{(5r + 100)(r - \Delta r)}{100 \Delta r} \right) t_o \tag{5}$$

Checkpointing criteria:

Checkpointing depends on the development of a distribution of failure rates but the formulas were established based on those.

System approach of the best priority formula:

Therefore the recommended formula would appear as shown in Formula 12:

$$P(t) = \left[7 * t + 120 * \left(1 + \frac{t}{r} \right) \right] \tag{12}$$

These approaches and formulas show that an instant utilization boost of 16,000 CPU hours is possible and with the average job using approximately 1,100 node hours, this means almost 15 more jobs will be finished in a week. Also, giving the users the appropriate skill and formulations will make their jobs and the system run even more efficiently.

LIST OF REFERENCES

[1] Little, John D. C., “A Proof for the Queuing Formula: L=λ W”, *Operations Research*, Vol. 9, No. 3, pp. 383-387, May-Jun., 1961.

[2] Schrage, Linus E. and Miller, Louis W., “The Queue M/G/1 with the Shortest Remaining Processing Time Discipline”, *Operations Research*, Vol. 14, No. 4, pp. 670-684, Jul.-Aug., 1966.

[3] Ernemann, Carsten; Krogmann, Martin; Lepping, Joachim; and Yahyapour, Ramin., “Scheduling on the Top 50 Machines”, *JSSPP*, LNCS 3277, Springer-Verlag Berlin Heidelberg(publisher), pp. 17-46, 2005.

[4] Warren Smith, Ian Foster, and Valerie Taylor. “Scheduling with Advanced Reservations”, *Parallel and Distributed Processing Symposium, IDDPS proceedings,* 14th International, pp. 127-132, 2000.

[5] Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch and Phil Andrews. “Impact of Reservations on Production Job Scheduling” *JSSPP Proceedings of the 13th International conference on Job scheduling strategies for parallel processing*. Springer-Verlag Berlin Heidelberg(publisher), 2007.

[6] Brent N. Chun and David E. Culler. “User-centric Performance Analysis of Market-based Cluster Batch Schedulers”, *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 20-30, 2002.

[7] Daniel Nurmi, John Brevik, and Rich Wolski. “QBETS: Queue Bounds Estimation from Time Series”, *Lecture Notes in Computer Science*, Vol. 4942, pp. 76-101, 2008.

[8] Francis T. Lui, “An Equilibrium Queuing Model of Bribery” *The Journal of Political Economy*, Vol. 93, No. 4, pp. 760-781, Aug. 1985.

[9] Tsafrir, Dan. "Using Inaccurate Estimates Accurately". E. Frachtenberg and U. Schwiegelshohn (Eds.): JSSPP 2010, LNCS 6253, pp. 208–221, 2010.

[10] Snell, Q., Clement, M., and Jackson, D.B. "Preemption Based Backfill". D.G. Feitelson et al. (Eds.): JSSPP 2002, LNCS 2537, pp. 24-37, 2002.

[11] William A. Ward, Jr., Carrie L. Mahood, and John E. West. "Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy". D.G. Feitelson et al. (Eds.): JSSPP 2002, LNCS 2537, pp. 88-102, 2002.

[12] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates"

IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 6, Pg. 789-804, June 2007

[13] Gerald Sabin, Garima Kochhar, and P. Sadayappan. (2004). "Job Fairness in Non-Preemptive Job Scheduling", ICPP 2004, pp 186-194, vol. 1.

[14] Barry G. Lawson and Evgenia Smirni. (2002). "Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems", Lecture Notes in Computer Science, volume 2537, pp 72-87, 2002.

[15] Li, Dr. Xueping. Personal Conversation. University of Tennessee, Knoxville. November 2010.

[16] W. David Kelton, Randall P. Sadowski, and David T. Sturrock. *Simulation with Arena, 4e,* McGraw Hill (2006)

[17] S.J. Bose, (2002) "Chapter 1 - An Introduction to Queueing System", Kluwer/Plenum Publishers, 2002.

[18] UTK Web News, 2010 (http://www.utk.edu/tntoday/2010/11/12/supercomputer-top500-rankings-released-kraken-top-10/)

[19] David G. Kendall, "Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain", *The Annals of Mathematical Statistics*, Vol. 24, No. 3, pp. 338-354, Sep., 1953

[20] HPCwire.com, "NICS Unleashes 'Kraken' Supercomputer", April 4, 2008. (http://www.hpcwire.com/features/17910874.html?page=1).

[21] Jonathan R. Celso, Michael Vanderlan, and Joseph H. Wilck IV, "Analyzing the Utilization of the Oak Ridge National Laboratory Supercomputer", *not published*, 2010.

APPENDIX

A1: A table of sample data from the Kraken data set

JobNumber	SubmitTime	WaitTime	RunTime	AllocatedCPU	CPU_TimeUsed	UsedMemory	ReqNumOfCPU	ReqTime	ReqMemory	Status	UserID	GroupID	NumOfApps	QueueNumber	PartitionNumber	PrecedJobNumber	ThinkTime
1	13334	11422	518	49176	0	2	1	43200	-1	1	1	1	1	5	1	0	0
2	13652	64	0	12	0	0	1	600	-1	1	2	2	1	1	1	0	0
3	13833	58	410	12000	0	6	1	600	-1	1	2	2	1	4	1	0	0
4	17191	10	2	98976	0	0	1	600	-1	1	2	2	1	5	1	0	0
5	17277	33	17	98976	0	0	1	600	-1	1	2	2	1	5	1	0	0
6	17655	9	609	98976	0	4	1	600	-1	1	2	2	1	5	1	0	0
7	18415	7	143	98976	0	4	1	600	-1	1	2	2	1	5	1	0	0
8	18633	45	640	98976	0	4	1	600	-1	1	2	2	1	5	1	0	0
9	19240	169	115	98976	0	0	1	14400	-1	1	2	2	1	5	1	0	0
10	32506	9	4024	7032	0	9	1	86400	-1	1	3	3	3	3	1	0	0
11	32818	9	1093	5856	0	15	1	86400	-1	1	3	3	3	3	1	0	0
12	33384	9	13232	1008	0	71	1	86400	-1	1	4	4	1	3	1	0	0
13	33721	6	12902	2016	0	36	1	43200	-1	1	4	4	1	3	1	0	0
14	33903	26	39749	5856	0	11	1	86400	-1	1	3	3	3	3	1	0	0
15	35080	9	244	960	0	75	1	39600	-1	1	5	5	2	3	1	0	0
16	35424	10	41792	960	0	75	1	64800	-1	1	5	5	2	3	1	0	0
17	35456	9	3487	72	0	984	1	3600	-1	1	6	6	8	1	1	0	0
18	35679	41555	40306	960	0	75	1	64800	-1	1	5	5	2	3	1	0	0
19	36192	9	2756	48	0	1475	1	3600	-1	1	6	6	8	1	1	0	0
20	36273	81275	41862	960	0	75	1	64800	-1	1	5	5	2	3	1	0	0
21	36716	12	18013	960	0	75	1	18000	-1	1	5	5	2	3	1	0	0
22	36725	17	4363	7032	0	9	1	86400	-1	1	3	3	3	3	1	0	0
23	37611	121809	42161	960	0	75	1	64800	-1	1	5	5	2	3	1	0	0
24	37666	163928	43220	960	0	95	1	64800	-1	1	5	5	2	3	1	0	0
25	37711	207184	31104	960	0	0	1	64800	-1	1	5	5	2	3	1	0	0
26	37752	277448	30	960	0	64	1	64800	-1	1	5	5	2	3	1	0	0
27	38528	8476	56663	48	0	1098	1	172800	-1	1	7	7	2	2	1	0	0
28	39017	8	371	12	0	7682	1	900	-1	1	8	2	1	1	1	0	0
29	39807	7250	90711	96	0	550	1	172800	-1	1	7	7	2	2	1	0	0
30	40583	13	628	81120	0	4	1	600	-1	1	9	8	1	5	1	0	0
31	40944	338	2069	180	0	294	1	16200	-1	1	10	9	1	1	1	0	0
32	41060	219	67966	7032	0	10	1	86400	-1	1	3	3	3	3	1	0	0
33	41309	11	622	12000	0	4	1	600	-1	1	9	8	1	4	1	0	0
34	41393	8	84	12000	0	7	1	600	-1	1	9	8	1	4	1	0	0

VITA

Michael Vanderlan is a graduate student earning his Masters of Science in Industrial Engineering at the University of Tennessee, Knoxville. Before coming back to the University of Tennessee he attained his Bachelors of Science in Business Administration in 2005 from the University of Tennessee. After three years of various positions with local companies he decided to enroll back in the university in the college of engineering. He is excited to be continuing his education at the University of Tennessee with his current enrollment in the graduate program in Industrial Engineering with a Research Assistantship under Dr. Joseph H. Wilck, IV. His email address is mvanderlan@gmail.com.