ASCII Queuing Systems: Overview and Comparisons

Scott H. Clearwater
P.O. Box 620513
Woodside, CA 94062
clearway@ix.netcom.com

Stephen D. Kleban
Sandia National Laboratories
Albuquerque, NM 87545
sdkleba@sandia.gov

Abstract

This paper describes research in and a performance comparison of the Accelerated Strategic Computing Initiative (ASCI) queuing algorithms using a newly developed simulator. The goal of this research is to develop models of the queuing systems used at the Sandia, Los Alamos, and Livermore National Laboratories (SNL, LANL, LLNL) and to study the strengths and weaknesses of the queuing algorithms. We present results of a number of simulation runs using actual job log data as well as generated job data where we analyze various performance metrics relating to overall facility utilization, responsiveness at the individual job level, and predictability in terms of run time. We find that the algorithms generally perform fairly closely to each other and are able to identify some manifestations of queuing policy in the performance metrics.

1: Introduction

The ASCII series of supercomputers at the national laboratories were built for specific purposes. Consequently the job queuing system that evolved at each lab bears a bias to that lab's particular needs and sociology. The machine fulfills its primary mission by satisfying its most important users, namely those that run big jobs. This state of affairs is due in part to the customized needs of bleeding edge scientific computing. These systems are so new and become obsolete so fast that the queuing systems and accounting systems are rarely studied in depth. Also, queue parameter settings are a moving target and change as often as monthly to meet the ongoing needs of the users. Nevertheless, future utilization of these machines will involve more complex scheduling of various computational resources within a lab and across labs, and thus there is a need to understand the implications of queuing policy in some detail.

We present an analysis of the queuing systems NQS (Network Queuing System)[2, 3], LSF (Load Sharing Facility)[4, 5], and DPCS (Distributed Production Control System)[6, 7] at the Tri-lab's (SNL, LANL, and LLNL) ASCII machines (Red, BlueMountain, White) using a newly-developed simulator called BIRMinator (Big Iron Resource Management simulator)[8]. We use actual job log data from each of the Tri-labs to drive the simulator as well as statistical generation of jobs to more fully study the parameter space.

2: Comparison

Table 1 summarizes the salient features of the ASCII job queues. NQS is the closest to a pure FIFO queue. DPCS has the most operator intervention for job ordering at least on White, which at the time of this paper was still not a production machine. DPCS uses an LSF-type mechanism for prioritization, but is more complicated and subject to manual overriding. Also, when system-level checkpointing is employed, a job is not restarted with a priority queue position, but must be manually resubmitted by the user. To employ a political analogy, NQS at SNL is "Senatorial" scheme (each user no matter how politically powerful or weak has an equal job limit), while the Fair Share schemes of LANL and LLNL are "Representational" (proportional to political clout).

In all cases of the above queuing systems, the user has no direct control over where the job goes into the queue. The burden of machine resource allocation is thus placed on the queuing algorithm. This means that there is a large premium on the algorithm being correct over a wide variation of operating conditions. Also, the algorithm must be reconfigured to adapt to novel situations of hardware and user environments. A synopsis of the stages of the three algorithms is shown in Table 2. The process starts with a user submitting a job, followed by a decision about which job to dispatch to the machine and to "age" the jobs (increase their priority based on time in queue) still waiting in the queue.
There are a number of ways that BIRMinator and its ancillary analysis program, BIRMalyzer, can be used to study supercomputers. The BIRMinator workflow is shown in Fig. 1. One of the features of BIRMinator is that it allows several mechanisms to generate jobs. Specifically, a BIRMinator user must define which of the following possible mechanisms is the source of jobs:

1) A “rerun” of log data where the job parameters (submission time, number of processors, run time) found in an actual job log are presented exactly the same way to the simulated queues. The reason for this mode is to validate the simulator.

2) A “replay” of the jobs where the actual number of processors and run time (defined as the “job size”) are used in random order from the log with a stochastically determined submission time. The reason for this mode is to be able to test the performance under a wider variety of job submission time distributions.

3) A “generation” of the jobs where the submission time, number of processors and run time are all generated randomly from given distributions, e.g., based on parameters from actual log data or other sources.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>NQS (Sandia)</th>
<th>LSF (Los Alamos)</th>
<th>DPCS (Livermore)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num Procs</td>
<td>9632</td>
<td>6144</td>
<td>8192</td>
</tr>
<tr>
<td>Gflops/Proc</td>
<td>0.247</td>
<td>0.262</td>
<td>0.603</td>
</tr>
<tr>
<td>User Job Quota</td>
<td>≤2 in queue or running/person</td>
<td>based on shares of user, group</td>
<td>based on shares of user, group</td>
</tr>
<tr>
<td>Partition/Configuration</td>
<td>BIG (~6764 procs), SMALL (~3382) BIG runs 2 weeks, then rewired for SMALL for 2 weeks</td>
<td>big(4864 procs) small(1280 procs) each partition is dedicated to big or small jobs</td>
<td>no partitions</td>
</tr>
<tr>
<td>Queues</td>
<td>big, “non-qual”, day</td>
<td>large, small</td>
<td>Single queue</td>
</tr>
<tr>
<td>max run time</td>
<td>24 hours</td>
<td>6 hours</td>
<td>24 hours</td>
</tr>
<tr>
<td>Backfill</td>
<td>larger jobs first, then smaller</td>
<td>based on shares</td>
<td>based on shares</td>
</tr>
<tr>
<td>Variability</td>
<td>Queue in place for ~3 months at a time so users can learn it</td>
<td>Queues and shares changed monthly for political reasons</td>
<td>Queue definitions and shares changed frequently for political reasons</td>
</tr>
<tr>
<td>Prioritization</td>
<td>queue + jobsize + &quot;aging, larger jobs have higher initial priority</td>
<td>shares, aging</td>
<td>political, aging</td>
</tr>
<tr>
<td>Time (&quot;aging&quot;) effects</td>
<td>job priority increased by 0.05/hr job is in queue</td>
<td>Half-life of 48 hours to recover priority</td>
<td>Half-life of 1 week to recover priority</td>
</tr>
<tr>
<td>Limiting factors</td>
<td>present and queued usage</td>
<td>past, present, and queued usage</td>
<td>past, present, and queued usage</td>
</tr>
<tr>
<td>Philosophy</td>
<td>Simple algorithms allow gameable strategies. Larger groups get more time because of 2 jobs/user limit.</td>
<td>Funding of projects dictates shares, therefore usage. Object to make non-gameable.</td>
<td>Political pressures from projects dictates share allocations.</td>
</tr>
</tbody>
</table>

Table 1: Salient Features of ASCI Queuing Algorithms.

3: BIRMinator: Big Iron Resource Management Simulator

To study the various ASCI queues we developed the BIRMinator simulator[8]. BIRMinator is written entirely in Java and uses a command line interface with plain text input files to define the facility, machine, queue, groups, and user configurations. It should be pointed out that BIRMinator was developed over a period of about 4 months with less than 10k lines of code, while a real queuing system such as DPCS—the most complex of the queuing systems we studied—took ten years of development and has over 300k lines of code. Thus, necessarily some of the complexity and nuances of the queuing algorithms has not been captured in BIRMinator.

The task of creating BIRMinator was made more difficult by the fact that the actual queuing algorithms are not well documented, although extensive documentation does exist that approximates the algorithms—or what they were in the past. Consequently, a significant amount of energy was spent in eliciting the actual algorithms from the respective queue experts.
The analysis of BIRMinator results involves cross-checking each queue type against data from each facility and the corresponding machine. For example, using job data from SNL, BIRMinator was run Red under NQS, LSF, and DPCS (see Fig. 1) and similarly for job data from BlueMountain, and job data from White (actually BluePacific as explained below). However, due to space limitations, we describe only one of the analyses, using data from BluePacific. Nevertheless, it is instructive to see how tightly coupled the various queuing algorithms are to their respective roles because it reflects on how a coordinated tri-lab effort like DRM will fare[9].

All the simulations using the BluePacific data were run over a simulated time period of 8 days which results in several thousand jobs. The simulations were started from a cold start with no jobs in any queue and are later filled as users submit jobs. Fig. 2 shows the workflow of BIRMinator. The point of the experiments was to try to isolate the queuing algorithm from its machine and see how it would respond to jobs from another machine.

We defined a number of performance metrics to study the performance of the various queuing algorithms as shown in Table 3. In addition to performance-based measures (usage, runtime efficiency), we also study prediction-based measures (nthSubmitted – nthRun and QueueTravel). The predictability of a queuing system, from the user's point of view, could in fact be called a form of "scheduling risk." Riskier systems are likely to be viewed with more skepticism by users. We should note that one might expect that the mean of the nthSubmitted-nthRun distribution should always be centered at zero, but it is the spread of the data that is of interest.

Another measure of predictability has to do with the travel a job takes enroute to being actually run. Thus in queue travel, we look at the position of a job in the queue each time the queue is reordered due to aging. The \( q_i \) is the position of the job in the queue at the \( i \)th update. A job may have only one entry, while others may have none if they start running before the jobs in the queue are aged. Note that this value has a minimum value of zero, which corresponds to a job that stays in its original position until it is run on the machine.

The cumulative probability distribution versus the logarithm of the cumulative job sizes (procs*runtime) for BlueMountain is shown in Fig. 3. To exemplify how the total jobsize of the machine is dominated by large jobs,
Fig. 1: Experimental comparisons of queuing algorithms.

Fig. 2: Workflow of BIRMinator for this project.

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage = ( \frac{CPU \text{ sec used}}{CPU \text{ sec possible}} )</td>
<td>Overall utilization of the machine, facility level metric.</td>
</tr>
<tr>
<td>runtime efficiency = ( \frac{runtime}{waittime + runtime} )</td>
<td>Job level efficiency. Close to 1 shows that queue is allocating jobs quickly.</td>
</tr>
<tr>
<td>nthSubmitted – nthRun &lt; 0 job moves ahead in queue nthSubmitted – nthRun = 0 job runs in same order as submitted (= FIFO) nthSubmitted – nthRun &gt; 0 job moves back in queue</td>
<td>Predictability measure of ordering of submission versus ordering of running. Narrow distribution means more predictable in terms of run time.</td>
</tr>
<tr>
<td>( QueueTravel = \sum</td>
<td>q_i - q_{i-1}</td>
</tr>
</tbody>
</table>

Table 3: Performance Metrics
for BlueMountain the smallest 90% of the jobs account for only 4% of the total job size of the machine and the smallest 80% of the jobs account for just 0.3% of the total job size of the machine. Put another way, the largest 10% of the jobs account for 96% of the job size and the largest 20% of the jobs account for over 99%. In terms of number of users, out of the 77 used in this study, the top 5 users accounted for 2/3 of the job size.

Given the extremely skewed, or heavy-tailed job-size distribution, we should study how these large jobs behave in terms of wait-time for the various algorithms[10]. This result will indicate how the algorithms are responding to their biggest users and is thus an indicator of the responsiveness of the queuing algorithm to its most important users. Additionally, given the highly skewed nature of the distributions of processors and runtimes, the meaning of average values is misleading. For small-time users, the averages appear unrealistically large and to the power users the averages are unrealistically small. Consequently, the mean values represent only a tiny minority of jobs. To capture the scale of the small-time users, the median is a more representative measure. To capture the power users, the 95%-ile of the distributions is used, that is, the 95%-ile in terms of job size distribution. To capture the small user we use median values. Since the power users account for most of actual job size on the machines, it is these latter distributions that are most important in terms of evaluating the performance of the queuing algorithms for these users. Nevertheless, the small-time users have many more voices than the power users, even if their voices are not as loud.

![Fig. 3: Probability versus Log_{10}(jobsize).](image)

To verify that our queuing algorithms were correct, we compared simulated reruns of the log data wherein not only were the actual processor and runtimes used, but also the submission times (as opposed to stochastically generated submission times) versus the actual log data. The results were not identical with those from the log data. The reasons for this are several, namely; 1) the simulator starts off “cold” with no existing jobs running, whereas the log data comes from a snapshot from a running machine, i.e., we did not have a snapshot state of the machine, 2) we did not have the exact snapshot of the state of the users, namely, we did not have their actual shares, but had to infer them from the log data, 3) real jobs are queued based on estimated run time, but typically run in less than their scheduled time, and 4) there are hand submitted jobs that run immediately and outside any algorithmic control. Recall that BIRMinator assumes that scheduled time is the run time, which leads to a difference in the way the queues are ordered. Consequently, we are not able to exactly replicate the log data. The actual distributions of wait times show a reasonable agreements with the rerun data with about 40% of the rerun jobs in exact agreement with the actual log data and about 60% within 5 minutes of the actual log data.

The BlueMountain data is the only data in which the machine is partitioned to accommodate small and large jobs. To test the effect of this partition we ran the simulator at a rate 1.5 times the raw rate to add extra stress on the queuing system. The results are shown in Table 4. Removing the partition led to an overall improvement in all the areas measured, but not necessarily with respect to the objectives of the facility management. NQS tended to run the larger and longer jobs, but with the cost of an average longer wait time.

Fig. 4 shows the wait time distribution for the stochastic replay results of Table 4. Note that the LSF(no partition) is very close to NQS and DPCS and quite different from LSF(partition). This indicates that the partitioning effect done for BlueMountain has a significant effect on the wait time distribution of the jobs.

![Fig. 4: Probability versus Log_{10}(wait-time(sec)) for stochastic replay rate = 1.5.](image)
Table 4: BlueMountain Replay rate = 1.5

emphasizing large jobs as shown by its higher numbers in average procs, run-time, and job-size (procs*run-time).

5: Discussion

We have analyzed real and simulated job data from the Tri-labs three queuing systems run on each machine using the BIRMinator simulator. The queuing algorithms perform fairly closely to each other with the major differences occurring in waiting times metrics and queue travel distributions. For the most part, the queues are fairly close to each other in performance and as the load on the machines is increased, they tend to respond in the same way, i.e., with higher usage but longer wait times. The major source of discrepancy between the log data and simulated data can come from several sources, as discussed in the previous sections. Unfortunately, these discrepancies manifest themselves most strongly in the wait time of the jobs, which is one of the key values for determining the efficacy of the queuing algorithm for users. To resolve the remaining discrepancy between the log data and the rerun simulations would require many more months of work to understand deeply all the fine points of the various queue algorithms as they are in practice. Further work may also reveal subtleties that go toward a more sophisticated understanding of the strengths and tradeoffs of the queuing strategies. Finally, in BIRMinator all jobs use all their scheduled time, as opposed to overestimating job size as is typically the case. We hope to improve our modeling of these effects in subsequent versions of BIRMinator.

Table 5 shows how the queuing algorithms ordered from best to worst performed (ordered from best to worst) based on the averages of the results of section 4 (LSF refers to LSF with a partition). Items enclosed by parentheses are within 10% of the best algorithm for the efficiency measures, half a standard deviation
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>(DPCS, NQS), LSF</td>
<td>(NQS, DPCS, LSF)</td>
<td>(NQS, DPCS, LSF)</td>
<td>NQS, DPCS, LSF</td>
</tr>
<tr>
<td>Run time Eff.(all)</td>
<td>(LSF, DPCS), NQS</td>
<td>(DPCS, NQS), LSF</td>
<td>(LSF, DPCS), NQS</td>
<td>DPCS, LSF, NQS</td>
</tr>
<tr>
<td>Run timeEff.(95%-ile)</td>
<td>(DPCS, NQS), LSF</td>
<td>DPCS, NQS, LSF</td>
<td>DPCS, NQS, LSF</td>
<td>DPCS, NQS, LSF</td>
</tr>
<tr>
<td>Min. spread of nthSubmitted-thRun(95%-ile)</td>
<td>(DPCS, NQS), LSF</td>
<td>DPCS, NQS, LSF</td>
<td>DPCS, LSF, NQS</td>
<td>DPCS, NQS, LSF</td>
</tr>
<tr>
<td>Min. Q.Trav.(all)</td>
<td>NQS, DPCS, LSF</td>
<td>(NQS, DPCS), LSF</td>
<td>DPCS, NQS, LSF</td>
<td>NQS, DPCS, LSF</td>
</tr>
<tr>
<td>Min.Q.Trav.(95%-ile)</td>
<td>NQS, DPCS, LSF</td>
<td>(NQS, DPCS), LSF</td>
<td>DPCS, NQS, LSF</td>
<td>(NQS, DPCS), LSF</td>
</tr>
<tr>
<td>Overall Order</td>
<td>(NQS, DPCS), LSF</td>
<td>(DPCS, NQS), LSF</td>
<td>DPCS, NQS, LSF</td>
<td>DPCS, NQS, LSF</td>
</tr>
</tbody>
</table>

Table 5: Ordered List of Best to Worst Queuing Algorithms for Specific Performance Metrics.

The queue travel metric is biased by the fact that the queue travel is only updated when the priorities of the jobs in the queue are updated, which varies for each machine and in some cases is dynamically determined by every submission to the queue or job finishing on the machine. Thus, an algorithm that samples the queue much more frequently may have jobs that move around more simply because of that greater sampling rate. Still, it must be remembered that our goal was to see how the “native” queues performed, not how they would perform if they were completely normalized to each other. Normalizing these results across the different queuing algorithms would provide another view of this metric and might lead to different conclusions. The results shown in Table 5 for the queue travel hold for the overall distributions as well as the 5% largest job sizes.

While most of our metrics are related to waiting time, queue activity, and predictability, it is important to note that waiting time, while important to the individual user is not a good predictor of the overall usage of the facility. In fact, waiting time and usage are complementary, i.e., improving one of them leads to a worsening in the other. These differences in performance have implications for the future when the tri-labs move to co-allocation of resource queuing schemes, both within each lab, and across the labs. BIRMinator can be modified to study these more complicated issues as well as more sophisticated queuing algorithms such as those involving market-based queuing algorithms such as those involving market-based ideas[13, 14].

6: Future Work

We have shown how a new supercomputer simulator, BIRMinator, can be used to study various queuing algorithms at the Tri-labs. At present, however, BIRMinator, and the Tri-lab queuing algorithms themselves, are limited to scheduling or queuing based on...
CPU needs in terms of number of processors and run time. As mentioned earlier, there is a need to understand how these algorithms will scale as the Tri-labs move to a more cooperative model involving sharing of resources and incorporating co-allocation of other resources such as network bandwidth and disk space[11, 12]. To study these upcoming issues, it would indeed be useful to have a simulation to study how existing or proposed scheduling systems will perform. This suggests a course of action for future work with BIRMINATOR that would include adding multiple resource scheduling. With this capability we could study, for example, how badly does performance degrade when multiple resources such as CPU, disk, and bandwidth are required, but no co-allocation scheme is employed? How much better can an advanced reservation scheme improve performance?

An intriguing possibility for future queuing algorithms is to introduce market mechanisms into job scheduling. This is quite different from the current state of affairs where the queuing algorithm determines job priority, leaving the user with having to justify to the facility management a manually submitted high priority job. Such a market-based mechanism has been proposed in the First-In Dutch Auction (FIDA) as discussed in [13]. Basically, it provides a user with the means for determining the priority of each job he or she submits. The priority of jobs in the queue is then determined by the willingness of the users to pay for the resources out of their allotment or shares. The “First In” aspect of FIDA means that small users will still move forward in the queue and not be forever pre-empted by larger users who can move ahead through the Dutch Auction.

References


Acknowledgements

This paper would not have been possible without the expertise of many people who provided us with detailed knowledge of algorithms, data formats, as well as providing the job log data. The authors gratefully acknowledge the assistance of Charles Hales, Michael Hannah, Steve Humphreys, Moe Jette, Wilbur Johnson, Tom Klingner, Amy Pezzoni, Randall Rheinheimer, Bob Wood, and Andy Yoo. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.