VARQ: Implementing Probabilistic Advanced Reservations for Batch-scheduled Parallel Machines
UCSB Technical Report 2007-09

Daniel Nurmi, Rich Wolski and John Brevik
Computer Science Department
University of California, Santa Barbara
Santa Barbara, California 93106

Abstract

In high-performance computing (HPC) settings, in which multiprocessor machines are shared among users with potentially competing resource demands, processors are allocated to user workload using space sharing. Typically, users interact with a given machine by submitting their jobs to a centralized batch scheduler that implements a site-specific, and often partially hidden, policy designed to maximize machine utilization while providing tolerable turn-around times. To these users, the functioning of the batch scheduler and the policies it implements are both critical operating system components since they control how each job is serviced. In practice, while most HPC systems experience good utilization levels, the amount of time experienced by individual jobs waiting to begin execution has been shown to be highly variable and difficult to predict, leading to user confusion and/or frustration.

One method for dealing with this uncertainty that has been proposed is to allow users who are willing to plan ahead to make "advanced reservations" for processor resources. To date, however, few if any HPC centers provide an advanced reservation capability to their general user populations for fear (supported by previous research) that diminished machine utilization will occur if and when advanced reservations are introduced.

In this work, we describe **VARQ***, a new method for job scheduling that provides users with probabilistic "virtual" advanced reservations using only existing best effort batch schedulers and policies. VARQ functions as an overlay, submitting jobs that are indistinguishable from the normal (i.e. non-reservation) workload serviced by a scheduler. We describe the statistical methods we use to implement VARQ, detail an empirical evaluation of its effectiveness in a number of HPC settings, and explore the potential future impact of VARQ should it become widely used. Without requiring HPC sites to support advanced reservations, we find that VARQ can implement a reservation capability probabilistically and that the effects of this probabilistic approach are unlikely to negatively affect resource utilization.*

1. Introduction

Most high-performance computing (HPC) centers serving the scientific and engineering research communities use space-sharing to manage the allocation of compute resources to user programs. These systems today are frequently implemented as centralized batch schedulers to which users submit jobs (text scripts each specifying a program to be run and its resource requirements) for eventual execution on some partition of a target machine. When a job begins executing, it is given exclusive access to the processors in its partition (*i.e.*, the machine resources are space-shared) and preemption is typically not supported. Thus, each job must wait in a queue until programs that have been scheduled before it release sufficient resources to permit it to run.

Queue-scheduling priority is rarely, if ever, first-in-first-out (FIFO). Instead, system administrators control scheduling priority through a policy interface (typically a scheduler configuration file) specifying how jobs waiting for execution should be chosen when sufficient resources within the machine become available. While minimizing user turnaround times and scheduling fairness are concerns, the scheduling policies must balance these requirements against the need to maximize machine utilization and dynamically changing administrative demands such as an important user who suddenly requires a substantial fraction of the machine to meet an unforeseen deadline. Because of this tension, the exact scheduling policy in place at any given time is rarely revealed to the general user community, nor are users always informed when policy changes are instituted, particularly when they are made to favor a particular user or group of users (however necessary the changes might be).

For these reasons, but also because large-scale parallel programs have highly variable execution times [4, 16] and because HPC resources in research settings tend to be severely overcommitted, user jobs often experience highly variable queuing delays [2, 6, 7, 27]. At present, for example, jobs submitted at the National Science Foundation (NSF) HPC centers may experience several orders or magnitude variation in queuing delay, when delay is measured in seconds [1]. Predicting the time an individual job will wait

[1] This level of variability is not particular to the NSF centers, and in fact has been greater historically across machines no longer in service that have been managed by a number of different organizations. For evidence, see Feitelson's workload archive at `http:`

in a given queue when it is submitted has been the subject of previous research [6, 7, 27] but reliable and accurate predictions of the delays experienced by individual jobs remain elusive. Thus users with specific deadlines find it difficult or impossible to plan effectively to meet those deadlines.

One possible approach to solving the problem of unpredictable queuing delay is to allow users to make *advanced reservations* [17, 26, 28] for resources. While most open-source and commercial batch schedulers provide support for user reservations, to date this capability is not offered to the general user population by any of the HPC computing centers of which we are currently aware. While there are a number of reasons why advanced reservations are not available (*e.g.*, it is not clear what users should be charged for a reservation, what the priorities for making a reservation should be, etc.) the primary concern appears to be the possible loss of machine utilization. Unlike a busy restaurant that can cover the cost of an unused or under-used reservation through higher prices to all customers, the HPC centers pay for their resources almost entirely "up front" and then account for the capital expense as utilization over the lifetime of the machine. Thus lost utilization can be viewed as lost revenue that cannot be recovered. Specially privileged users may still make reservations for particularly important deadlines, but these reservations are negotiated with site administrators beforehand and are not available to the general user community.

In this work, we present a new statistical method that implements advanced reservations probabilistically as an overlay atop existing best-effort (i.e. non-reservable) batch-queue systems in production HPC settings. Our approach builds upon recent work in predicting bounds on queuing delay using fast, on-line time-series techniques [29]. In this paper, we use these results to build a virtual reservation capability for regular (e.g. non-privileged) users that does not require the cooperation of the target batch scheduler. That is, site administrators are not required to implement a local reservation capability; rather, they see jobs managed by our system as part of the normal workload. Users experience the cost of a virtual reservation as an additional charge to their accounts (typically funded in units of allowed occupancy time) automatically, without a change to local accounting systems. One drawback of our approach is that the exact cost for a specific reservation is difficult to predict precisely. The system attempts to minimize this cost, however, and it does provide conservative worst-case estimates. Finally, users are able to specify explicitly an acceptable failure probability for each virtual reservation.

In this paper, we detail the implementation of virtual advanced reservations and evaluate its effectiveness empirically using several shared production HPC facilities currently dedicated to science and engineering research. We also analyze the cost, in terms of additional charges to our occupancy allocations, incurred during our experiments. Finally, we use a trace-based, faster-than-real-time simulator to explore the possible effects of virtual advanced reservations should our system become a popular infrastructure component.

In so doing, this paper makes the following contributions.

- We propose a statistical approach to implementing advanced reservations in production science and engineering HPC settings that does not require site administrators to implement hard reservations.
- We analyze the effectiveness of this approach using both a working implementation targeting "live" HPC systems running in production mode and its potential impact using a new trace-based simulation capability.
- We find that virtual advanced reservations are surprisingly effective at the present time on the HPC machines we tested and that their impact is unlikely to affect current HPC operational settings negatively.
- We describe the statistical conditions that must exist at the sites for these results to be general in future, and argue that they are likely to exist for the near and medium term.

These contributions are important and relevant to research in operating systems for three reasons. First, from the perspective of the site operators, the job scheduler is a critical operating system component. While it may not be implemented with direct kernel support, the scheduler essential for system operation in production mode and, as a result, is treated as having the same operational value as other components more obviously part of the HPC operating system (e.g. parallel file systems). Secondly, the job scheduler is the scheduler with which users interact in HPC settings. For example, while the individual nodes of an HPC cluster may be running Linux and the Linux process scheduler, that fact is generally invisible to users submitting parallel programs written using MPI [13] – a common scenario. Finally, as grid computing evolves [1, 14] new distributed workflow applications [8, 23] that require advanced reservations to meet specific processing deadlines in response to unforeseen events (*e.g.*, natural disasters) are being developed. As part of a grid middleware or overlay, the development of our system could play an important role.

The remainder of this paper is organized as follows. In the next section (Section 2) we describe how batch systems are currently managed in the science and engineering HPC community and explore some previous related work in this field. Continuing in Section 3, we describe the statistical approaches and methods we have developed to make virtual advanced reservations. Section 4 describes the experiments we have performed to evaluate the efficacy and generality of our technique and their results. Finally we conclude in Section 5.

2. Background and Related Work

Our contribution of a **V**irtual **A**dvanced **R**eservation system for **Q**ueues (VARQ) is designed to function in an administrative environment that is typical of science and engineering computing centers serving users with potentially competing resource demands. In this section, we describe the general characteristics of these HPC settings and discuss other research projects that are germane to our effort. VARQ's function depends critically on QBETS (**Q**ueue **B**ounds **E**stimation from **T**ime **S**eries) – an on-line tool for making real-time bounds predictions of queue delay [29]. While developing and deploying QBETS in a number of University, National Science Foundation (NSF), and open Department of Energy (DOE) centers, we observed several features common to the

//www.cs.huji.ac.il/labs/parallel/workload.

way these systems are managed that, in part, make the success of VARQ possible. We provide a brief summary of these features as background to the remainder of this paper.

2.1 Users, Accounting, and Priority

In most of these settings, each user (represented by a unique per-site user identifier) is associated with one or more accounts each funded with an allocation of per-node occupancy time. When a user submits a job to the local batch scheduler, she must specify which account to charge when the program is eventually allocated machine resources for execution (on some systems, the scheduler may choose a default for the user if none is specified). There is no charge made to the account while the job is waiting in queue, nor is any form of refund or compensation granted for jobs that wait for overly long periods. It is only the execution occupancy time that is decremented from the account once the program begins executing. If the account is exhausted, the currently executing jobs charging the account are terminated. Note that in most settings the "aspect ratio" of a parallel job submitted by a user is specifically not considered in the accounting subsystem. That is, a 1-node job executing for 100 hours decrements the user's account by the same quantity that a 100 node job executing for 1 hour does: 100 node-hours.

Typically, site administrators configure their batch systems to employ a simple fundamental scheduling policy based on techniques such as first-come-first-serve (FCFS), and then perform further tuning, taking into consideration specific job and/or user priority goals that are unique to each site. A nice overview of current parallel job scheduling techniques is provided in [12] and a more comprehensive survey for past methods is provided in [11]. In particular, most sites currently use some form of backfilling [18, 19, 21] to maintain utilization in the presence of large resource requests without introducing starvation. Utilization, in this context, is measured in terms of node occupancy. Once a user's job is allocated a set of nodes, the system does not (and almost certainly cannot) determine whether the work done by the program is useful – only that an account should be charged for the occupancy.

If large node counts are needed by a job, the machine must "drain" until a sufficient number of nodes become available (recall from the previous section that pre-emption and/or checkpointing is typically not available). To avoid the potential loss of utilization that results during a drain, each job can specify a maximum execution time, past which it is willing to be terminated. A backfilling scheduler will use these execution limits to schedule jobs from farther down in the queue onto draining nodes such that they do not prolong the waiting time of a job causing the drain that is ahead of them.

In general, the scheduling policies that are in place are not made entirely public, partially because users may not be entirely satisfied with their respective priorities, but also because specific knowledge of existing policies may allow users to attempt to "game" the system in order to obtain better turnaround time for their own jobs [22, 24]. Moreover, HPC resources are typically overcommitted. Because it is difficult to predict resource demand in an environment where demand is driven by research, and also to ensure that the utilization of expensive resources is maximized, total user allocations typically exceed feasible occupancy. For these reasons, predicting the time individual jobs wait in queue has proved to be a difficult problem [2, 6, 7, 27]. To give users some measure of predictability and control, many centers configure different queues with partially described scheduling priorities to allow users to make some form priority-motivated decision. For example, a "short" queue may accept jobs that have maximum run times no greater than 15 minutes which are given preference during backfilling. Users can use this information to gain faster turn-around for small amounts of work, but the exact degree of preference is typically not revealed, and local administrators change the specific policy parameters without announcement (sometimes frequently).

However, some users still attempt to influence their scheduling priority through the submission of multiple jobs they do not really wish to execute. One strategy, for example, is to try and submit a large spurious job that will cause the system to begin backfilling and then to try and exploit the temporary priority elevation that smaller jobs will receive as a result. At the same time users are attempting to develop their own scheduling strategies, administrators who are tracking machine and user activity may be altering policy to negate these strategies. In many cases, however, hysteresis is applied in the form of an active job limit in which a specific user or account can have no more than a small number of jobs (regardless of size) eligible for consideration across all queues.

The effects of this constantly changing interaction between users and the batch-scheduling policies in place, in conjunction with other factors such as hardware failures, preventive maintenance, etc. that may induce changes as well, often result in queuing delays that fluctuate through several orders of magnitude (c.f. Figures 2 and 5 for typical examples).

2.2 Grid Computing and Co-Allocation

Over the last decade, there has been a great deal of interest in the concept of grid computing [1, 14] (originally termed "meta-computing" [25]), which is essentially the idea of using multiple distributed, heterogeneous resources with minimal global centralized control structures to perform coordinated tasks, such as executing scientific applications. One of the fundamental research hurdles which needs to be overcome to realize a functional grid computing environment is that of resource co-allocation, where multiple disparate sets of resources must be made available simultaneously to some global scheduler. There have been many research efforts indicating that grid "meta-scheduler" systems [3, 9, 10, 15] can increases global system utilization while providing large resource pools, but for the most part these efforts require the use of a centralized, global batch scheduling entity to which *all* jobs (both global and local) are submitted. While this body of work shows a great deal of promise and utility using simulation and closed research environments, the modification and coordination burden placed on local site resource operators has proven to be so severe as to not be applicable in practice. The primary prohibitive modification these systems impose on local site schedulers is that of allowing regular users to make advance reservations in order to support resource co-allocation. Several studies have shown that allowing regular users the right to make advance reservations can have a negative impact on both system utilization and overall turnaround time for regular jobs. In [26], it was shown that the introduction of a general use advanced reservation system into a normal HPC workload can have a substantial impact on the

experience of regular batch users. In this work, the authors convert 10-20% of jobs in historical job traces into "reservation" requests for a upcoming time in the future. Even though the assumption is that the jobs requiring advanced reservations can tolerate some slippage (reservation start time can be delayed if scheduler cannot guarantee resources at the requested time), the average wait time increase for regular batch jobs increased by 9-37% depending on how many reservation jobs were made (10-20% of overall jobs, respectively). In [17], the authors evaluate the impact of advance reservations on a regular batch controlled workload in terms of percentage of reservation requests rejected, slowdown factor of regular jobs (termed "variable" jobs in the paper) and system utilization. Running various simulation experiments, using two reservation algorithms and a real job trace, the authors are able to determine that the introduction of advance reservations increases the queuing delay experienced by regular jobs, but it is difficult to gauge the magnitude of the impact based on the metric used. In essence, the authors show that as the number of accepted advance reservations increases, the negative impact on queuing delay of regular jobs increases as well. Finally, researchers in [28] perform a simulation based experiment, using the popular Maui scheduler [20], that attempts to determine the effect of advance reservations on regular HPC workloads; the authors suggest methods for minimizing this effect when compared to an alternate technique for co-allocation in which sites explicitly reserve a specific time every day for explicit meta-scheduler use. Although the authors make a compelling case for the use of advance reservations based scheduling to support cross-site co-allocation of resources, their experiments and conclusions indicate that the introduction of advance reservations have a negative impact on both system utilization as well as queue delay experienced by non-reservation jobs.

As a result of these studies, HPC site operators have been reluctant to adopt the use of general advance reservation systems to support off-site metaschedulers, and are unwilling or unable to relinquish local control of resource scheduling to a global scheduling system. In a particularly relevant work proposing a resource management system for metacomputing [5], the authors acknowledge the fact that local control must be maintained in order for HPC centers to subscribe to metacomputing methodologies, but argue that advance reservations must be supported to fully support co-allocation. Even so, the authors briefly propose a method for making a "best effort" batch submission to support co-allocation without advance reservation, but do not detail the specific mechanisms used to implement their solution, and make it clear that their solution is a temporary situation which will be replaced when sites adopt general advance reservation functionality. However, in the decade since this paper was published, general adoption of advance reservation capabilities has remained elusive and shows little sign of becoming enabled on production systems.

2.3 QBETS

We wish to be unambiguous about the novelty of our batch-queue job delay prediction system (QBETS) and virtual advance reservation system (VARQ). QBETS is prior art. It became operational in December of 2006, and is currently deployed on fifteen production HPC centers around the world, where it is used primarily as an advisory tool (through a number of different web-based interfaces) for users who wish to plan their job submissions. VARQ is a system we have developed that uses QBETS to implement an advanced reservation abstraction for its users, and it is novel. To provide adequate context for VARQ, we briefly summarize the function of QBETS without detailing its effectiveness (as we have in previous work). Instead, in addition to the analysis of its capabilities that we have reported on previously, in its current operational mode we continually monitor the correctness and accuracy of its predictions. To date, both exceed the initial levels we had hoped it would achieve.

QBETS predicts bounds, analogous to confidence intervals, on delay by estimating percentiles of the empirically observed delay distribution by analyzing a continually updated log of previous job delays. We have found, perhaps unsurprisingly in retrospect, that a bounds prediction on delay is more useful in this setting than a moment-based point-valued prediction. Users are often interested in "worst case" delay, since getting the results of a program execution earlier that expected generally does not imply a penalty or cost as might receiving them late. To estimate bounds on delay time, a QBETS user provides a description of the job as a 4-tuple (machine, queue, requested nodes, requested execution time) and a success probability corresponding to the percentile of interest.

For example, as user who wishes to be "95% certain" of having her job start before the time bound that will be returned by QBETS prompts the system to estimate and report the 95^{th} percentile of the delay distribution. It does so by using a non-parametric estimate of percentiles based on a binomial (success-failure) treatment of historical job delay data. QBETS also implements a specialized change-point detector so that it only considers history relevant (within the current region of stationarity) to the current prediction it is making. Finally, QBETS uses on-line model-based clustering to categorize jobs automatically in terms of the scheduling delay they have experienced. This clustering reverse-engineers the current priority mechanism in place by detecting which classes of jobs (based on their node counts and execution times) are experiencing similar delays. By maintaining a separate predictor for each class, the clustering has the practical effect of indicating (indirectly) where the backfilling cutoff is at any given moment.

QBETS computes a time bound on the delay a specific user job will experience. For this work, we have modified QBETS to invert this functionality so that it returns an integer percentile estimate between 1 and 100 for a specific time bound. By treating these percentiles as coming from a single empirical distribution, QBETS can return the probability that a job corresponding to a specific 4-tuple will begin executing before a specified period of time has elapsed (termed the deadline). This functionality, which is the foundation of our virtual resource reservation system, is expressed here as a function to be used for the remainder of this paper:

$$QBETS(m, q, nodes, wallTime, startDeadline) = prob$$

For example, if we were to submit a 4 node, one hour (3600 seconds) job to the UC/ANL TeraGrid supercomputer in the "dque" queue, QBETS estimates the probability that it will wait no longer than ten minutes (600 seconds) if submitted immediately:

$$QBETS(ucteragrid, dque, 4, 3600, 600) = prob$$

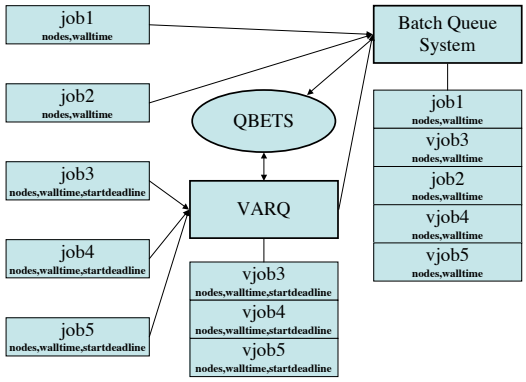

Figure 1. Overview of VARQ system interaction with local batch scheduler

At the time of this writing, $prob = .89$ for such a job. The accuracy of this function on all the systems we monitor continues to impress us under continual verification and despite dynamic system behavior.

In summary, by constantly monitoring the delays experienced by jobs submitted to a batch scheduler, adjusting the historical data considered, and categorizing jobs into service classes, QBETS makes surprisingly accurate on-line predictions of queue delay bounds for individual jobs.

3. VARQ Design and Implementation

VARQ implements a reservation by determining when (according to predictions made by QBETS) a job should be submitted to a batch queue so as to ensure it will be running at a particular point in future time. It does not actually reserve the resource, but rather achieves the same goal – the predictably scheduled execution of a user program – that a reservation enables. Because VARQ does not require modification to the local scheduling policies or scheduler submission protocols, it can function as an overlay. VARQ jobs do not appear different from non-VARQ submissions at the batch scheduler.

Figure 1 provides an overview of the functional relationships between VARQ, QBETS , and the local batch scheduler. User batch jobs may be submitted to VARQ for execution at a specific point in time in the future (specified by a deadline). VARQ then uses QBETS predictions to determine when the job should be submitted to the batch scheduler queue to ensure it is executing at the deadline. At the same time, non-VARQ jobs are being submitted to the batch scheduler queue. The delays these jobs experience affect QBETS predictions which in turn, affects VARQ decision making. We describe the nature of this interaction more completely in the following subsections.

3.1 Virtual Advanced Reservations

Using QBETS , we can estimate the probability, at time T, of a specific job beginning execution by a certain time in the future $T + startDeadline$, but we cannot say when between T and $T + startDeadline$ the job will actually start. Using the UC TeraGrid example above, we know that, at time T, QBETS reported a 0.89 probability of the specified job starting within ten minutes. However QBETS provides no information about the likelihood of the job starting at any specific time between $T + 1$ seconds and $T + 599$ seconds. Because of this uncertainly, this probabilistic prediction alone is not sufficient for certain applications which need to reserve a precise time slot in the future when the resources will be available.

One naive way to get around this deficiency is to attempt to submit a job that requests a runtime long enough to encapsulate both the time from T to $T + startDeadline$ and the requested time of the job itself (wallTime). Using such a tactic, we would submit a job which, instead of requesting $wallTime$ seconds of compute time, instead requests $wallTime + startDeadline$ seconds. This technique will guarantee that if the job begins execution between T and $T + startDeadline$, then it will be allowed to execute from $T + startDeadline$ to $T + startDeadline + wallTime$. If the user desires that the job start at $T + startDeadline$ and not before, the job simply needs to "sleep" or spin until time $T + startDeadline$. Recall from Section 2 that once the batch scheduler allocates nodes to a job, the job will not be prematurely terminated nor pre-empted. Thus any job is free to simply wait to begin doing useful work, however the accounting system will charge the user's allocation for occupancy starting at the moment the job acquires its nodes. Because this occupancy is by a user job, and the user's account is charged for it, the center does not need to, and indeed cannot, consider it lost utilization.

Potential Drawbacks

The disadvantages to this approach are twofold. First, when the desired $startDeadline$ is large (and it likely is), then the job could waste a substantial amount of allocation by holding the resources until the user's deadline arrives. Second, again when $startDeadline$ is large, the probability of such a large job making it through the queue is much lower than the job requesting the time actually needed for execution. For example, if $startDeadline = 43200$ and $wallTime = 3600$, then we would be requesting a 46800 second job when we only need 3600 seconds of compute time, 43200 seconds from now. The probability of a 46800 second job making it through the queue by $startDeadline$ is much lower than that of a 3600 second job starting by $startDeadline$, primarily because of the inability of the scheduler to use this job for backfilling.

Bounds Prediction Stability

Our solution to this problem is to find the amount of time to wait before submitting a job so that when we do submit, the job isn't so large as to make the probability of success prohibitively low because of the additional runtime necessary to cover the possibility that it begins running immediately. This approach is based on the observation that if the percentile estimates do not change, or change slowly, QBETS predictions made in the future will look very much like current predictions. Thus it is possible to predict the bounds on delay if the user were to wait a short time (thereby reducing the extra time needed to cover the delay until the deadline) before submitting a job.

In the process of verifying QBETS (a process that continues), we observed that while the queue delays may fluctuate to a great

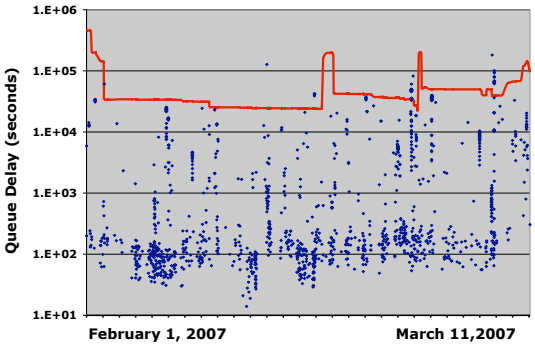

Figure 2. Queue delay measurements and QBETS 95^{th} percentile predictions on Datastar for the month of February, 2007.

degree, the time series of percentile predictions corresponding to those delays are relatively stable, often over many days. Figure 2 presents an example that compares queue delays observed for jobs submitted to the "normal" queue (the default work queue) on the San Diego Supercomputer Center's Datastar machine and the corresponding QBETS estimates for the upper 95^{th} percentile reported during the month of February of 2007. In the figure, the x-axis represents the submission time of a job, and the y-axis (using a log scale) describes the delay (so that the figure shows the *time series* of delays). Each point feature represents the delay observed for a single job, and the line feature traces the QBETS estimates. Notice that even though the job delays vary from between 10 and $110,000$ seconds, the percentile estimate is quite stable by comparison.

The clustering feature of QBETS enhanced the stability shown in Figure 2 by selecting a relatively homogeneous subset of the wait times. While the full trace shows several orders of magnitude variation, this trace for a single cluster automatically identified by QBETS shows only a range of only about 4 orders of magnitude. The reason for this effect is that the appearance of highly variable delay may be because the scheduler is interleaving jobs of different classes, each of which experiences a different "class" of delay. For example, if large jobs are experiencing roughly $100,000$ seconds of delay and small jobs are all being serviced in 10s of seconds, an interleaving of the two appears to have more variance than either taken separately.

Because the bounds predictions are so stable, it is possible to use the inverted predictor function $QBETS()$ to estimate the probabilities that jobs submitted at successive points in the future (each having a successively shorter requested execution time) will start running at some point before a specific deadline and will be able to continue executing until completion. However, the effect is not monotonic. Notice that in Figure 2 very few jobs waited less than 20 seconds between the time they were submitted and the time they began execution; a greater number waited between 20 and 100 seconds; *etc.* This effect occurs because in the short run, the scheduler attempts to implement a fair policy between jobs of equivalent resource requirements. Thus a VARQ job submitted near the deadline will contend with non-VARQ jobs for immediate initiation thereby, decreasing that job's probability of starting before the deadline. Therefore, as the submission time approaches the deadline, the probability of starting before the deadline tends to

```
INPUT(mach, queue, nodes, wallTime, startDeadline, reqProb)
OUTPUT(waitT, advWallTime)
T = current UNIX timestamp
currT = T
currProb = I = 0

WHILE (currT < startDeadline)
   advWallTime = wallTime + (startDeadline - currT)
   currProb = QBETS(nodes, advWallTime)
   probVec[I] = (currT, currProb)
   I = I + 1
   currT = currT + 30
ENDWHILE

I = LENGTH(probVec)
WHILE (I >= 0)
   (currT, currProb) = probVec[I]
   IF (currProb >= reqProb) THEN
      waitT = currT
      advWallTime = startDeadline - currT
      RETURN(waitT, advWallTime)
   ENDIF
   I = I - 1
ENDWHILE
```

Figure 3. Pseudocode describing VARQ determines how long to wait before submitting a VARQ job.

increase, possibly due to the backfilling, as less additional runtime is necessary to cover the time before the approaching deadline; but it tends to *decrease* due to contention by other submissions and the scheduler's need to enforce fairness among equivalent jobs. One might expect, then, to find a "sweet spot" at which the probability is maximized.

Probability Trajectories

To find the submission time in the future that will most likely meet the deadline or a submission time that corresponds to a user's reservation request, VARQ computes a *probability trajectory* for the user's job by considering the possibility of submitting a given job at successive 30 second intervals from the time the job is given to VARQ until the specified deadline. For each point in time, it decrements the additional runtime $startDeadline$ required by 30 seconds, and estimates the the probability of starting before the deadline using $QBETS()$. Specifically, it implements the algorithm described in pseudocode in Figure 3, where the algorithm accepts as input the 4-tuple job description, the required time when the resources must be available ($startDeadline$) and the minimum acceptable probability that the reservation request is successful ($reqProb$). Upon completion, the algorithm returns the number of seconds VARQ should wait before submitting the job ($waitT$), and the modified job walltime ($advWallTime$) required to ensure that the reservation can be met with probability $reqProb$.

Figure 4 depicts an example VARQ probability trajectory (denoted $probVec$ in the pseudocode) computed in this way. The data comes from a VARQ reservation made at 2:49 PM on March 6^{th}, 2007 on the NCSA TeraGrid machine for the "dque." For this reservation, the user requested 4 nodes for 1 hour of execution time starting at 2:49 AM on March 7^{th} (12 hours into the future). Time of day (given as a Unix timestamp) beginning at 2:49 PM on the left-hand side of the figure is shown along the x-axis. The y-axis shows the return values of $QBETS()$ which is the probability estimate for the job starting before the deadline at 2:49 AM (right-hand side of the graph) as function of when, in the future, it is submitted.

From time T at 2:49 AM until approximately $T + 21600$ sec-

onds, the probability of the requisite sized job starting before $T + 43200$ steadily drops from slightly below 0.4 to 0.25. This part of the graph illustrates the probability decay that occurs as the eventual submission time and the deadline draw closer together. However, at approximately $T + 21600$, we see a drastic increase in probabilistic prediction. This increase shows the effects of the clustering algorithm used by QBETS . At that point in time, the combination of node request and total requested execution time for the job put it in a different scheduler service class (presumably due to the possibility of backfilling). After this point, again the probabilities steadily approach 0 as the deadline approaches.

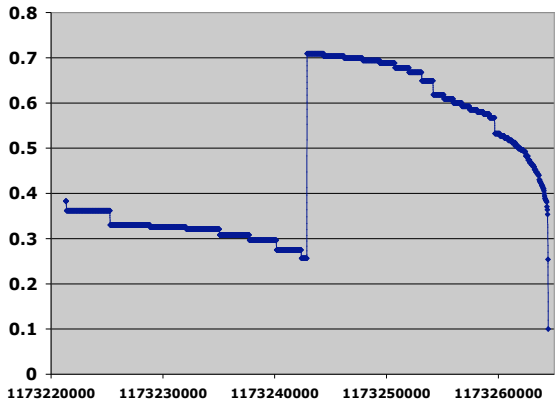

Figure 4. VARQ probability trajectory for a *4*-node, *1*-hour job in the "dque" on NCSA TeraGrid machine

The probability trajectory can be used to identify the point in time when VARQ should submit the job to the machine's queue that corresponds to the most probable success in attaining the reservation (at $T+21600$ in the figure the QBETS reported a maximum probability of approximately 0.7). VARQ supports this mode of operation, but in choosing the maximum, the user cannot explicitly tradeoff success probability for potentially lost allocation. If the user in this case requested VARQ to submit at the most probable point in time, and the job began running immediately, the user's allocation could be charged a maximum of an additional $4 * (43200 - 21600) = 86400$ node-seconds of allocation in addition to the $3600 * 4 = 14400$ node-seconds required for the job's execution.

Minimizing Lost Allocation

To allow somewhat greater efficiency and flexibility, VARQ also accepts a target success probability from the user and finds the *latest* submission time in the probability trajectory that can meet it as a way of minimizing the additional allocation overhead. For example, if the user specified success probability of 0.5 and the trajectory in Figure 4 were used, then VARQ walks backwards in $probVec$ until it encounters the first timestamp where the probability is equal to or exceeds 0.5. In this case, such a timestamp exists at $T + 40470$ which indicates that we should wait 40470 seconds and then submit a 4 node job requesting 6330 second job $(3600 + (43200 - 40470))$ to guarantee that the job will be running between $T + 43200$ and $T + 43200 + 3600$. The potential allocation overhead (i.e. the maximum possible additional allocation cost) is $4 * (43200 - 40470) = 9480$ node-seconds for the same job requiring 14400 node-seconds of execution time.

Thus the VARQ probability trajectory allows the user to trade estimated success probability for potential allocation overhead. In this example, reducing the desired success probability from 0.7 to 0.5 implies a reduction in potential extra allocation cost from 86400 node-seconds to 9480 node-seconds. VARQ, at present, reports only the maximum possible allocation overhead since it does not currently attempt to estimate at what time before the deadline the job is likely to begin executing. We believe a "best guess" in addition to the worst case allocation loss is possible, however, and we are pursuing it as part of current efforts.

Notice also that the probability trajectory associated with a particular VARQ reservation may indicate that there is no submission time corresponding to the user's specified success probability. Returning to the example, if the user had specified a desired success probability greater than 0.7, the system would have responded by indicating that no reservation is possible. This condition is analogous to the circumstance in which a "hard" reservation is denied because a conflicting reservation has already been made. In addition, to handle the possibility that a change-point occurs during a reservation period, VARQ continually generates new probability trajectories while waiting to submit a job.

In sum, VARQ exploits the slowly changing nature of QBETS bounds estimates to determine when in the future a job should be submitted so that it will be running at a specific deadline. Because QBETS estimates are upper bounds, the job may start earlier than the desired time and simply wait, incurring an extra allocation charge while it does. VARQ allows the user to control this cost explicitly by specifying a target success probability that it will try to honor with the minimum potential cost.

4. Experiments and Results

To explore the efficacy of VARQ, we report results from a series of empirical experiments conducted using the machines listed in Table 1. Each of these machines is currently in production use by a shared, and potentially competing, user community. To the best of our knowledge, our user login and account specification received "typical" treatment on each machine (with one possible caveat discussed below), and we did not inform the relevant system administrators of the experiments. We also conducted a simulation experiment to understand the effects of multiple users submitting VARQ jobs as a preliminary investigation of its potential generality. Without cooperation from site administrators, however, we felt it ill-advised to conduct "stress" tests involving multiple and frequent VARQ requests in live settings in which unsuspecting users could be exposed to unforeseen system response.

4.1 Efficacy Experiments and Apparatus

Table 1 describes the characteristics of the machines we chose to use for our experiments. We chose these machines for a number of reasons: Each machine is supported by QBETS , has a number of active users (although some machines are observed to be busier than others), and provides us the low-level ability to instrument the submission and tracking of job status necessary to perform an actual experiment and gather meaningful results.

In each experiment a submitting process formulates a job, and then selects a specific time in the future when the job needs to be

Machine	Processors	Batch Software	Description
datastar	2176	Load Leveler	SDSC IBM PowerPC Production Compute Cluster
ucteragrid	316	Torque/Maui	UC/ANL IBM/Intel Compute/Viz Linux TeraGrid Cluster
dante	35	Torque/Maui	RENCI Intel Xeon Research Linux Cluster
cnsidell	256	Torque/Maui	UCSB NanoScience Research Linux Cluster
ncsateragrid	1744	Torque/Maui	NCSA IBM/Intel Compute Linux TeraGrid Cluster
iuteragrid	32	PBS	IU AVIDD Compute Linux Cluster
ornlteragrid	56	Torque	ORNL IA64 Compute Linux Cluster

Table 1. HPC machines used in VARQ empirical experiment. Chosen systems represent a realistic set of distributed HPC resources on which users would have simulataneous access.

running, and a probability of success. Ideally, we would have liked to perform this experiment for all job sizes, with a large number of future deadlines, and for a multitude of success probabilities. However, since the experiments run in real time, and the delays on these machines can be substantial, exploring every reasonable combination of these factors is infeasible.

We have set up one machine at our host institution to act as a single point where all experiments are launched. On this machine, we run *submitting processes* designed to act as users that make reservation requests to VARQ for the HPC targets listed in Table 1.

Each process targets a specific machine and queue. When it is initiated, the job is passed a specific success probability and deadline (expressed as a duration until a reservation should be made) as parameters. It begins by composing a job for submission through VARQ using a randomly selected node count and run time from the following sets: either 1, 4, 8, 16, or 32 nodes and either 600, 1800, 3600, 7200, or 14400 seconds of run time. Once the submit process has crafted a job, it queries VARQ regarding the possibility of attaining a virtual reservation using the newly minted job and the deadline and success probability originally specified when the submit process was initiated. If, after computing the necessary probability trajectory for the reservation, VARQ cannot find a submission time in the future that will satisfy the reservation specification at the desired probability levels, it reports "unable to make reservation" to the submission process which sleeps for 15 seconds, composes a new random job, subtracts 15 seconds from the deadline (so that it targets the same point in time in the future), and retries. The process continues to retry every 15 seconds until VARQ accepts the reservation or until the deadline is decremented to zero. If the latter conditions occurs, the submission process resets the deadline it is attempting to its originally specified value (thereby picking a new target time in the future for a reservation) and continues to retry. Once the submit process successfully makes a reservation with VARQ, it then waits until shortly after the deadline has expired and starts again, attempting a new reservation one deadline duration into the future.

The intention of the protocol is to model a user who wishes to obtain a reservation that starts at a specific point in time, and who is willing to re-query the system in the event VARQ is unable to grant the request. It has the effect, however, of making the time between attempted reservations more or less equal. We do not believe this induced periodicity affects the results negatively, particularly since we observed a fair amount of "drift" in the experiment cycle for each process over the entire experimental period.

For instrumentation purposes, the experimental apparatus determines the success or failure of a VARQ job in meeting its deadline, and the actual allocation overhead incurred, by searching through the batch scheduler logs on the target machine *post facto*. We considered adding an instrumentation facility to VARQ itself to allow users to query the success history of their own reservations. Such an extension would increase the intrusiveness of VARQ substantially, however, since in its current form the only component that requires access to the local batch scheduler logging information is QBETS .

On the launching machine, we run an experiment process for each of three required probabilities (0.5, 0.75, and 0.95) and the same deadline duration (to speed the time to results). We stagger the start times of these processes so that they do not all target exactly the same moment as a deadline. Also, we for the sake of alacrity, we have chosen a deadline duration of 21600 seconds (six hours), both to improve the number of completed experiments, but also because we felt that six hours the shortest reasonable lead time a user would normally expect to be able to make static advanced reservations. As shown in Figure 2, the percentile time series for these machines is typically stable for several days. If a short reservation is possible, longer ones should be more likely within the confines of this stability.

4.2 Efficacy Results

To determine the efficacy of the VARQ system, we compare the percentage of successful VARQ attempts to the specified success probability. For example, a submit process attempting to make VARQ requests with 0.5 success probability, should have at least 50% of the submissions accepted by VARQ start before their specified deadlines.

In Table 2 we compare the target success probabilities with those we observed across all machines. The table is organized as follows. Each row corresponds to a specific machine (we used the default queue in each case). For each of three different success probabilities (0.5, 0.75, and 0.95) we show three columns of numbers: the average expected success probability used by VARQ, the actual fraction of jobs accepted by VARQ that met their deadlines, and the number of accepted jobs. Recall that VARQ uses the latest time in its probability trajectory that *exceeds* specified success probability as a way of attempting to potential allocation overhead. In some cases, this probability may be quite a bit larger than that specified, especially when the machine is lightly loaded. For example, using a specified 0.5 success probability on *ornlteragrid*, VARQ submitted a job when it "saw" predicted success

machine	.5			.75			.95		
	predicted	actual	count	predicted	actual	count	predicted	actual	count
datastar	**0.52**	**0.42**	**36**	**0.75**	**0.71**	**17**	0.97	0.00	1
ucteragrid	0.76	0.98	45	0.86	1.00	45	0.96	0.96	48
dante	0.90	0.80	61	0.93	0.78	59	**0.96**	**0.85**	**61**
cnsidell	0.54	0.71	62	0.76	0.88	66	0.00	0.00	0
ncsateragrid	0.53	0.74	23	0.76	0.77	13	0.00	0.00	0
iuteragrid	0.80	0.88	24	0.81	1.00	22	0.97	0.94	18
ornlteragrid	0.88	1.00	39	0.87	1.00	37	0.97	1.00	58
all	0.71	0.79	290	0.83	0.89	259	0.96	0.93	186

Table 2. Average predicted success probability and actual success fraction for VARQ reservations.

probability, on the average, of 0.88 in the probability trajectories it computed. 100% of the 39 jobs it submitted in this category met their deadlines (as shown in columns 2, 3, and 4 of Table 2 in the row for *ornlteragrid*).

These results indicate that VARQ, in the mode we have tested it, is quite successful. Of the 21 test cases (7 machines at 3 target probabilities each) only *dante* at the 0.95 target level and *datastar* at the 0.5 and 0.75 were probabilistic failures (shown in bold face in the table). There were several instances, however, where VARQ refused to accept any reservations, or only accepted one. These are not failures in the sense that the user (the submission process in our case) did not experience a different quality of service than the one VARQ agreed to deliver.

Returning to the observed failures, in *dante*'s case, of 61 jobs accepted by VARQ, with an average predicted success probability of 0.96, only 0.86 (52 jobs) successfully met their deadlines. We provide a more probing analysis of this case also in the next section. For *datastar*, however, the problem was that our job submissions were being assigned (accidentally) to an account used for educational purposes and not research. Apparently jobs submitted to this account receive degraded scheduling priority in comparison to the "average" research user tracked by QBETS . We discovered this anomaly only in *post mortem* analysis of the experiments. At the time of this writing, we have re-initiated the *datastar* experiment under the correct account, and the results for the small number of attempts show success. For the sake of uniformity, however, we felt it unwise to replace the *datastar* numbers with the new data since it was not part of the original experimental run and also because the number of attempts so far is too small to yield a meaningful inference.

For all other tests, however, in which the count of jobs attempted is not 0 or 1, the observed fraction of successes exceeds the average predicted success probability. These results combine to show that VARQ is conservative with respect to success probability. To generate Table 2 required 457 hours of wallclock time. We initiated the experiments at 11:30 AM on February 15^{th}, 2007 and terminated them at 12:48 PM on March 6^{th}, 2007. In many cases, despite retrying every 15 seconds, VARQ could not identify a single instance in a probability trajectory that it predicted would result in a successful reservation over the entire experimental period.

4.3 Allocation Overhead Results

In Table 3 we show the effects of VARQ on the allocation charges incurred during the experiment described above. Organized in a way similar to Table 2, each row corresponds to a specific machine, and each of the three major columns represents results for different specified success probabilities (0.5, 0.75, and 0.95 respectively). In Table 3, each major column shows the total allocation required to execute the jobs in that category (denoted *required*), the actual allocation used by VARQ in that category (denoted *used*) and the ratio of allocation used to allocation required (denoted *ratio*). The units of allocation in this table are node-hours. For example, columns 2, 3, and 4 of the row marked *ornlteragrid* show that the VARQ reservations submitted with a 0.5 success probability required 464 total node-hours of occupancy to execute the work in all jobs and 473 node-hours for that occupancy and the additional cost when jobs started early under VARQ. The ratio of 1.02 indicates the cost factor associated with the use of VARQ. That is, the submission process in this experiment "spent" 1.02 times as much allocation to obtain VARQ reservations as it would have spent had it simply submitted the jobs (without reservations) to the *ornlteragrid* machine. [2]

From the table, the allocation overhead penalty VARQ introduces varies from machine to machine. On the *cnsidell* machine, for example, VARQ reservations at the 0.5 probability level cost the allocation almost 2.5 times the non-reserved outlay compared to a cost factor of 1.08 on *ucteragrid*. This variability is consistent with our experience in developing QBETS previously in that each of the machines in this study displays a unique queue delay response profile. Also confirmed is the notion that greater certainty (in terms of higher success probability) implies a greater allocation cost since the cost factor increases monotonically from left to right in each row.

4.4 Generality Experiments and Apparatus

To be able to test the effectiveness of VARQ when a sizeable fraction of user-offered jobs are under its control, we constructed a faster-than-real time, trace-based simulator which uses the same VARQ infrastructure we used for the empirical experiment and the Maui [20] batch-scheduler running in simulation mode. The Maui

[2]Note that in these experiments we did not in fact incur these allocation charges, since once the job is allocated a set of processors, we garner no additional information by having it use them for the 3600-second execution period. Instead, each job simply computed for a few seconds to insure a log record would be generated and then exited. Since each job's execution time and node count are known, the allocation charges that would have resulted can be computed without wasting the actual allocation.

	.5			.75			.95		
machine	required	used	ratio	required	used	ratio	required	used	ratio
datastar	14	22	1.58	8	9	1.16	0	0	0.00
ucteragrid	1038	1120	1.08	877	956	1.09	563	1221	2.17
dante	884	944	1.07	604	735	1.22	831	2040	2.45
cnsidell	257	636	2.48	60	212	3.53	0	0	0.00
ncsateragrid	58	127	2.17	28	84	3.00	0	0	0.00
iuteragrid	82	82	1.00	91	91	1.00	110	258	2.34
ornlteragrid	464	473	1.02	628	640	1.02	83	94	1.13
all	2797	3405	1.22	2295	2725	1.19	1587	3612	2.28

Table 3. Non-VARQ and VARQ allocation costs and their ratio. Cost units are node-hours.

scheduler is the actual scheduler deployed at many of the sites we tested empirically (See Table 1). Maui includes a simulation capability that allows input job workloads to exercise a given scheduler policy so that potential performance effects can be identified prior to deployment. Because we did not have access to the specific scheduler policy files at each site, we chose Maui's default policy which is first-come-first-served with backfilling [20, 18] and a processor node count set to 272, which is the total number of nodes in the Datastar machine at SDSC, where each node has 8 processors.

We hasten to clarify that we do not claim the performance results generated by this simulator are representative of any actual machine or system hence we did not use it to investigate the effectiveness of VARQ. However, the question of how well VARQ performs when the fraction of offered workload controlled by VARQ increases is one we believe must be considered. To do so, we use the simulator to compare VARQ performance over repeated experiments where we vary only the fraction of jobs that use VARQ. If VARQ is to be a generally useful methodology, it must be able to support an appreciable fraction of the workload experienced by a given machine without breaking down or adversely impacting competitive users outside the prioritization specified in the local scheduler policy.

The simulator takes a workload trace (we chose the *datastar* "normal" queue since it seems particularly active), a fraction of jobs that should use VARQ and a success probability. Next, the simulator chooses regular time periods (six hours apart) within the trace indicating times of advance reservation start deadlines. This mode of operation represents the worst case where the given percentage of jobs request reservations starting at same time every six hours. Jobs are considered in submission order, and a job is selected to be considered a VARQ job randomly, but in proportion to the specified fraction (e.g. if the fraction is 0.10 each job has a 10% chance of being converted into a VARQ job). If the job is selected, it is presented to VARQ for execution at the next six hour deadline with the specified success probability.

4.5 Generality Results

The results of our simulation experiment are shown in Table 4. Note that although there are times when we failed to achieve the minimum expected success percentage using VARQ, in most cases VARQ was able to acquire a success percentage very close to the expected success percentage of reservations. Predictably, as the fraction of VARQ jobs increases and the success probability increases, VARQ's success rate decreases. However, in many cases the results are surprisingly close, given the extreme nature of the simulation. For example, if 10% of the jobs are VARQ jobs and they target the same deadline with a desired success probability of 0.95, the observed success fraction is 0.91. Only when 95% of the jobs are VARQ jobs and the desired success probability is equal to 0.75 do the simulations show VARQ's quality of service guarantees breaking down. Note that when 95% of the jobs are VARQ jobs and the desired success probability is either 0.5 or 0.95 VARQ almost succeeds. In the former case, the conservativeness of QBETS predictions furnishes VARQ with enough "slack" in the estimate of the 50^{th} percentile so that the predicted probability (0.68) is only slightly larger than the observed success fraction (0.64). In the latter case, QBETS is able to find few instances where it predicts the probability to be 0.95 or greater. A closer analysis of the simulation trace reveals that the 48 VARQ attempts in this case only occurred during period of light workload in the job trace we used. Thus we observe in this example there is a regime between the extremes of 0.5 and 0.95 success probability where VARQ clearly fails to operate. We believe this effect is general, but the precise failure regime will be site specific.

At a high level, these results indicate that VARQ is both likely to offer a larger user community valuable functionality without degrading resource performance or utilization. In the simulations, each VARQ job had its run time *increased* to cover the possibility of an early start. Either the simulated machine was under utilized by the original (non-VARQ) workload, in which case VARQ increases the utilization perceived by the system administration, or the machine was originally over committed, in which case VARQ does not cause utilization to be lost. Moreover *all* of the simulations executed in approximately the same simulated time interval as did the VARQ-free simulation (not shown). Thus the amount of work accomplished with active VARQ jobs is approximately the same as when no VARQ jobs are present. We present these results in order to provide evidence that a more aggressive field test of VARQ is warranted as its general use (even in the worst case) appears relatively benign.

4.6 Discussion

The results presented in the previous subsection show that VARQ implements a new advanced reservation abstraction for HPC users. Moreover, the abstraction is virtual. Existing batch systems, governed by complex and hidden local scheduling policies, do not need to change in any way and, in particular, do not need to agree to support any form of user-initiated reservation mechanism for VARQ to function. Finally, the virtualization is statistical. VARQ

Percent VARQ jobs	.5			.75			.95		
	predicted	actual	count	predicted	actual	count	predicted	actual	count
10	0.54	0.66	359	0.77	0.72	60	0.97	0.91	11
50	0.55	0.60	1869	0.77	0.68	1108	0.96	0.84	194
95	0.68	0.64	1960	0.43	0.81	939	0.98	0.97	48

Table 4. Average predicted success probability and actual observed success fraction for VARQ jobs in simulation.

uses predictions generated by QBETS to "manufacture" a reservation without local infrastructure support. As a fortuitous side-effect, each VARQ reservation can be characterized by success probability that is conservative. Users know the minimum success probability associated with each of their reservations. Together, these features enable VARQ to achieve a functionality first hypothesized as being useful almost a decade previously [5] and for which we believe there will be substantial demand.

New Capabilities

VARQ also offers several capabilities that would otherwise be difficult to implement as mechanism.

- It trivially implements a zero-overhead "best effort" reservation. In this mode, a user specifies a deadline and a success probability, but is willing to tolerate having the submitted job start before the deadline.
- It allows users to control what they pay in allocation as a function of how precisely they wish to have their deadlines met.
- It allows VARQ reservations to be combined from independent sites with predictable joint probabilities of success.

The first feature is simply a function of when a user job actually begins doing work after it has successfully be allocated a set of processors. If a user does not want to pay the allocation overhead necessary to wait until her deadline, her job need only begin executing immediately instead of 'spinning" until the deadline. In the current prototype, this spinning or waiting is implemented in the application itself. It is trivial to wrap the application in a script to avoid the need for user modification of the program. Either way, however, the user can control the cost at the time the job begins node occupancy.

For example, if the user would like her job to start between 5:00 PM and 6:00 PM, she can make a reservation for 6:00 PM, and have the job delay (wasting allocation) only until 5:00 PM. This level of control over the reservation cost and/or the character of the reservation is not possible with existing mechanisms.

Finally, because each VARQ reservation is associated with a success probability, it is trivially possible to combine reservations to meet a specific reliability target as long as the user can take advantage of which ever resource ultimately is delivered first. For example, if it is possible to obtain two different VARQ reservations for the same point in time, each with a success probability of 0.9, and the machines behave independently with respect to queue delay (which they almost certainly do at present), the probability of getting one or the other or both is 1.0 minus the probability that they will both fail. That is, the joint probability of success in this example is at least 0.99. While users may not take advantage of this simple approach manually since it involves canceling one of the submissions to avoid even greater allocation cost, in grid settings, where metaschedulers can manage this complexity automatically, the possibility is intriguing.

Unfortunately, the possibility of using VARQ for co-allocation induces the reverse effect. That is, the joint probability associated with co-allocated reservations is less than the probability of either one. However, it may be possible to use the "either-or" approach to mitigate this probability decay in a grid setting of sufficient scale. Because VARQ reservations are characterized by explicit success probabilities, using them coherently in combination under programmatic control is possible.

Limitations of VARQ

VARQ fails in terms of its ability to honor the probabilistic "quality of service" guarantees associated with a reservation when QBETS fails to identify stable percentiles in the queue delay distributions. As described in Section 4, this circumstance occurred during the empirical experiments on the *dante* machine. To illustrate the nature of this failure further, we show a time series trace of the queue delays experienced by jobs submitted to the *dante* machine during the experimental period in Figure 5. Note the region of the time

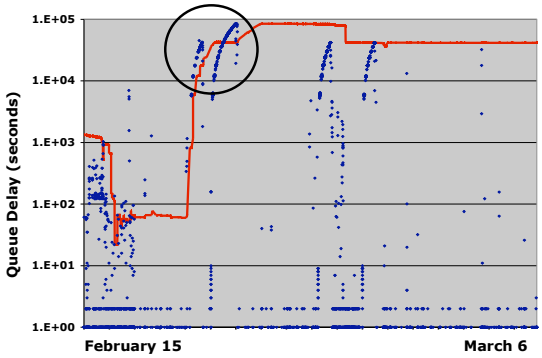

Figure 5. Queue delay measurements and QBETS 95^{th} percentile predictions on Dante for the experimental period.

series circled in the trace in which the delays exhibit sharply and monotonically increasing trends. During these periods, the delays are behaving deterministically (the delay experienced by each job is successively larger) and as a result, the predictive power of QBETS is lost. The QBETS predictor can only "guess" at the 95^{th} percentile based on the data it has seen previously. If each delay is greater than the next, the predictions will necessarily under-predict their values and the correctness of the bounds is eventually lost.

This sharply increasing trend behavior is usually indicative of

a machine failure. At many sites, the batch scheduler runs on a satellite machine that is unaffected by a crash of the actual HPC resource. Thus, users unaware that the resource is down continue to submit jobs only to see them "back up" in the queue. When the machine returns to service, the queues are rapidly cleared in, more or less, first-come-first-served order, but the delays recorded by QBETS include the down time.

We are currently working on developing a reliable down-time detector for QBETS that will alleviate this problem in the case when the machine is down. However, the trace in Figure 5 illustrates the more general limitations of VARQ. In this experiment, we do not actually know what the disposition of the *dante* machine was during the period we were trying to use it. In the unlikely case where this queue behavior is, in fact, the queue behavior desired by the site administrators (and not the result of a machine outage), then QBETS , and VARQ depending on it, will necessarily fail. More generally, VARQ depends on the machine being used in a competitive manner by enough users so that upper bounds on its queue delays can be inferred statistically from historical measurements. If the machine transitions into an operating regime where queue delays increase deterministically, VARQ through QBETS will not be able to obtain a correct upper bound prediction and hence will not be able to meet its quality of service guarantees.

5. Conclusion

One major hurdle the HPC community has yet solve generally is that of providing users and grid/metacomputing systems the ability to obtain advance reservations. In this work, we introduce VARQ, a statistical approach which provides users with a mechanism to obtain virtual advance reservations on existing systems, without affecting local site software or administration and scheduling policies. We show through empirical experiment that VARQ successfully obtains resources corresponding to a variety of user requests on real HPC systems in operation, and provide evidence that the introduction of VARQ as a more general tool is likely to be effective and will not cause a substantial impact on resource performance.

In the future, we plan to press for VARQ tests in live HPC settings, based on the strength of the results presented herein. We also plan to continue to use VARQ as part of applications that require deadline-based scheduling in large-scale distributed environments. Finally, adapting QBETS to meet the needs of VARQ has exposed several possible improvements (including the possibility of a machine down-time detector) which we plan to pursue.

Finally, by way of conclusion, we wish to express our sincere gratitude to the various organizations who have allowed us to access their scheduling systems in such an intimate way. Without their cooperation and openness, this research certainly would have been impossible.

6. REFERENCES

[1] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley and Sons, 2003.

[2] J. Brevik, D. Nurmi, and R. Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of PPoPP 2006*, March 2006.

[3] A. Bucur and D. Epema. The performance of processor co-allocation in multicluster systems. In *3rd IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2003.*

[4] S. Clearwater and S. Kleban. Heavy-tailed distributions in supercomputer jobs. Technical Report SAND2002-2378C, Sandia National Labs, 2002.

[5] C. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *International Parallel Processing Symp. – Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[6] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.

[7] A. Downey. Using queue time predictions for processor allocation. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997.

[8] K. Droegemeier, V. Chandrasekar, R. Clark, D. Gannon, S. Graves, E. Joseph, M. Ramamurthy, R. Wilhelmson, K. Brewster, B. Domenico, et al. Linked environments for atmospheric discovery (LEAD): A cyberinfrastructure for mesoscale meteorology research and education. *20th Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*.

[9] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002*, pages 39–47.

[10] C. Ernemann, V. Hamscher, and R. Yahyapour. Economic scheduling in grid computing, 2002.

[11] D. G. Feitelson. A survey of scheduling in multiprogrammed parallel systems.

[12] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling — a status report, 2004.

[13] M. P. I. Forum. Mpi: A message-passing interface standard. Technical Report CS-94-230, University of Tennessee, Knoxville, 1994.

[14] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[15] J. Gehring and T. Preiss. Scheduling a metacomputer with uncooperative sub-schedulers. *In Proc. JSSPP, pages 179*.

[16] M. Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, June 1999.

[17] F. Heine, M. Hovestadt, O. Kao, and A. Streit. On the impact of reservations from the grid on planning-based resource management. In *International Workshop on Grid Computing Security and Resource Management (GSRM 2005) at ICCS 2005, Atlanta, USA, Springer, LNCS 3516*, pages 155–162.

[18] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the maui scheduler. In *7th Workshop on Job Scheduling Strategies for Parallel Processing, 2001*.

[19] D. Lifka. *The ANL/IBM SP scheduling system*, volume 949. Springer-Verlag, 1995.

[20] Maui scheduler home page – `http://www.clusterresources.com/products/maui/`.

[21] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. In *IEEE Trans. Parallel and Distributed Syst. 12(6), Jun 2001*, pages 529–543.

[22] C. Ng, P. Buonadonna, B. N. Chun, A. C. Snoeren, , and A. Vahdat. Addressing strategic behavior in a deployed microeconomic resource allocator. In *In Proceedings of the 3rd Workshop on Economics of Peer-to-Peer Systems*, 2005.

[23] The southern california earthquake center home page – `http://www.scec.org/`.

[24] J. Shneidman, C. Ng, D. C. Parkes, A. AuYoung, A. C. Snoeren, and A. Vahdat. Why markets could (but don't currently) solve resource allocation problems in systems. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Operating Systems*, 2005.

[25] L. Smarr and C. E. Catlett. Metacomputing, 1992.

[26] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 127–132.
[27] W. Smith, V. E. Taylor, and I. T. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 202–219, London, UK, 1999. Springer-Verlag.
[28] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In *6th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 137–153, 2000.
[29] Reference removed for the purpose of blind submission.