

Probabilistic QoS Guarantees for Supercomputing Systems

A. J. Oliner, L. Rudolph

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139-4307 USA
e-mail: oliner@mit.edu, rudolph@csail.mit.edu

R. K. Sahoo, J. E. Moreira, M. Gupta

IBM T.J. Watson Research Center
1101 Kitchawan Road, Yorktown Heights, NY 10598-0218 USA
e-mail: {rsahoo,jmoreira,mgupta}@us.ibm.com

Abstract

Supercomputing systems must be able to reliably and efficiently complete their assigned workloads, even in the presence of failures. This paper proposes a system that allows the system and users to negotiate a mutually desirable risk strategy; in order to accomplish this, the system makes probabilistic guarantees on quality of service (QoS), of the form, “Job j can be completed by deadline d with probability p .” In order to make such guarantees, the system uses event prediction (forecasting) in conjunction with fault-aware job scheduling and cooperative checkpointing strategies. Using job logs and failure traces from actual high performance computing systems, we employ trace-based simulations to assess the effects of the prediction accuracy (a) and user risk strategy (U) on a variety of performance metrics. Compared to a system that does not use event prediction, a high forecasting accuracy resulted in QoS and utilization improvements of as much as 6%, along with an 89% reduction in the amount of lost work. Therefore, our results show that a system that makes probabilistic QoS guarantees using a market-based scheduling approach can increase both system performance and reliability.

1 Introduction

For many system management tasks, unilateral decisions on the part of either the users or the system are necessarily uninformed. This paper addresses one such task, the problem of scheduling jobs with quality of service (QoS) guarantees, such as deadlines, in the presence of failures. Users cannot decide their own deadlines, because the system will be unable to satisfy their requests. The system cannot decide deadlines for the users, because users will be unsatis-

fied with the service they receive. The solution is for the system and users to cooperate in order to devise a mutually desirable risk strategy. One possible semantics for this cooperation is for the scheduler to make guarantees of the form, “Job j can be completed by deadline d with probability p .” In such a scheduler, there exists a market-based interaction where users have an incentive to relax their deadlines. As a consequence, the system has greater scheduling flexibility and the users have more personalized service. The system we present relies on an event prediction mechanism to estimate the probability that the deadline will be met.

If computational resources are to be used in a realistic on-demand environment, we believe that supercomputing systems should make QoS guarantees to users, such as deadlines in a job scheduler. Deadlines are important for the end-user, who is not concerned about overall utilization. Users see latency, not throughput. In practice, deadlines are difficult to meet, in part due to the presence of hardware and software failures. Besides the obvious performance hit [16, 13], failures often lead to broken promises. Therefore, this paper pushes the following observation: *A system that makes unqualified performance guarantees is lying.* Any QoS promise must be accompanied either by the conditions under which the promise can be kept, or with a probabilistic confidence in the guarantee. No real system can promise a level of performance with 100% confidence, because any component or set of components may fail without warning.

Our system uses *event prediction*, or forecasting techniques, to make probabilistic QoS guarantees. The forecasting mechanism predicts any critical event that leads to the failure of a job (henceforth, failure). In order to keep these promises and proactively mitigate the cost of failures, the system uses event prediction to more cleverly schedule jobs [13] and to intelligently checkpoint applications [11]. Sahoo et al. [16] developed algorithms that demonstrated

the feasibility of event prediction in large-scale clusters. These algorithms often saw accuracies of 70%, in large part because failures in these clusters tend to be preceded by patterns of misbehavior. Then, Oliner [13, 12, 11] demonstrated that even low-accuracy (10% accuracy) prediction could be used to improve job scheduling and checkpointing, resulting in considerable performance benefits.

Within any supercomputer where users are required to supply non-factual or non-verifiable information regarding their job, it is possible that the user-supplied information is either inaccurate or approximate. Running times may be underestimated, priorities may be exaggerated, and deadlines may be fudged in an effort to manipulate job priority. To address these problems, the system uses a market-driven scheduling model that provides incentives for relaxing deadlines and increasing metaphorical budgets. Honesty and accuracy on the part of the users and the system confers benefits to both parties.

We investigate the behavior of this system through realistic trace-based simulations. Of particular interest is the effect of the accuracy of the event prediction. The user is able to influence the behavior of the system by communicating the preference of a particular risk strategy, so it is necessary to consider their behavior in the analysis, as well. Because these two parameters most clearly differentiate this system from current approaches, we focus our simulations on understanding the influence of prediction accuracy and user risk strategies on QoS and performance.

Our results reveal that probabilistic QoS guarantees do not need to come at the expense of efficient system utilization. In fact, in our experiments, improving QoS had the side effect of improving average utilization and decreasing the amount of work lost to failures. Both QoS and utilization saw increases of as much as 6%. The amount of work lost to failures is the most sensitive metric to changes in prediction accuracy or user behavior, decreasing in one case by nearly 90%. The results demonstrate that cooperating with users to devise a mutual risk strategy is an effective approach to achieving reliability in large-scale systems.

2 Related Work

There is a great deal of existing literature reporting QoS for customers involving computer resources [7, 8, 1]. An economy-driven job scheduling system called Libra [19] explored the idea that job scheduling should center around improving the value of utility to the user, rather than on maximizing system utilization. Libra allocated resources according to user QoS requirements, providing incentives for more relaxed requirements. Libra was able to provide improved system utility and user satisfaction. However, Libra did not explicitly handle failures, nor did it qualify its QoS guarantees. A different approach to QoS guarantees

[6] is based on the concept of a *schedulable region*, which is defined as being the space of offered loads for which QoS can be guaranteed. Similarly, Islam et al [7] report a QoS-based scheme for parallel job scheduling.

Many previous research efforts have looked at analyzing event logs and other system health signals [2, 3], and some have used event patterns to make predictions [16, 20, 22] in temporal as well as spatial domains. Recently, these predictions have been used effectively to improve system performance such as system time, job scheduling, and checkpointing [4, 13, 11], to name a few. The idea of using event prediction for proactive system management has also been explored [17, 15], as have mathematical approaches [21]. Sahoo et al [16] demonstrated the importance of system health monitoring and event prediction, as well as its feasibility for predicting critical rare events in large-scale clusters.

The main motivation of our work is to study the performance of a supercomputing system that makes probabilistic QoS guarantees in the presence of failures, as well as to analyze the effects of different user risk strategies on various system metrics.

3 System Description

QoS guarantees for reliability, availability, and usability for supercomputing systems involve a number of social and technical challenges. These include questions regarding appropriate interaction with users, and policy decisions regarding job scheduling and checkpointing. Probabilistic QoS guarantees require several components: system health monitoring, failure analysis and event prediction, job scheduling, and checkpointing. In addition, this section explains the negotiation process and metrics.

3.1 System Health Monitoring and Modeling

In order to make intelligent scheduling decisions and to estimate the probability with which a performance promise will be kept, the system monitors and models its health. This mechanism has access to both physical and logical data about the state of the machine, including information such as node temperatures, power consumption, error messages, problem flags, and maintenance schedules. The more information the health monitoring is able to access, the more useful its models will be. Such information helps not only the job scheduler, but also the checkpointing mechanism. The system health information for all nodes is collected at a centralized location and used to provide forecasts in terms of the probability of failure of a component within a certain future time frame. That is, the monitored health information is used as a tool for event prediction.

3.2 Event Prediction

The event prediction mechanism relies on the health modeling tool described above. This mechanism predicts critical events in hardware or software that might lead to the failure of a job. The event prediction uses a set of algorithms similar to those used before [13]. One such prediction algorithm [16] used two simultaneous models: linear time series models for the roughly continuous variables (e.g. node temperature and load) and Bayesian correlation models to recognize patterns in preceding system events (e.g. network errors and system warning messages). That method was able to predict up to 70% of the failures well in advance, and had a negligible rate of false positives. The prediction algorithm in this paper is given a set (partition) of nodes and a time window, and returns the estimated probability of failure. We study the effect of the accuracy (a) of this predictor on the performance of the system.

3.3 Job Scheduling

The job scheduling algorithm attempts to meet the deadlines negotiated between the users and the system, particularly for large jobs and those promised a high probability of success. This paper uses a fault-aware scheduling algorithm [13], specifically, a FCFS scheduler with backfilling, that uses event prediction to break ties among otherwise equivalent partitions. The scheduler tries to minimize p_f , the probability that a job's partition will fail at some point during the reservation time. The scheduler knows this probability from the predictor, and can inform the user accordingly. Because p_f is deadline-dependent, the scheduler could even suggest a deadline for the user, citing the increased probability of success as a motivating factor. For example, when a job arrives at an empty queue, the scheduler looks for the earliest possible time at which it could be run. At that time, it selects the partition with the lowest probability of failure. If this probability and deadline are acceptable to the user, the job is scheduled; otherwise, the user may agree to a later deadline in exchange for a greater probability of success.

The scheduler is given the following input: node topology, the current status of each node, a queue of waiting jobs, checkpointing information, and fault predictions. For every job j , the scheduler knows the job size in nodes (n_j), the estimated execution time of the job (e_j), and the estimated execution time including all checkpoints (E_j). After a job j has been scheduled to start at time s_j , the scheduler can compute the estimated completion time of the job ($f_j = E_j + s_j$). Once a job completes execution, the estimated value for f_j is replaced by its actual value. Migration is disabled in this study, because many supercomputers, such as IBM's BlueGene/L (BG/L), do not support it. Our simulations assume that the estimated execution times are

accurate. Although this is not always true in practice, there are techniques for dealing with this problem, including application profiling.

If a job fails, that job is returned to the wait queue to be restarted from the last completed checkpoint. Because there is no migration, other running jobs are not moved as a result of another job's failure. Furthermore, jobs that have already been scheduled for later execution retain their scheduled partition; there is no dynamic optimization of the schedule following a failure. In practice, predictions are less accurate as they stretch further into the future, and dynamic optimization may be desirable; the simulator, however, suffers from no such problem. The scheduler operates under the following constraints, based on earlier job scheduling work for BG/L [9, 13].

- Only one job may run on a given node at a time. There is no co-scheduling or multitasking.
- Nodes may fail at any time; if a job is running on a node when it fails, all unsaved work on that job is lost.

The scheduler also assumes that jobs have some minimum runtime. This assumption simplifies the analysis of system performance and prevents border cases that tend to occur when job runtimes are vanishingly small.

3.4 Checkpointing

The system uses a cooperative checkpointing scheme for supercomputing systems [11]. This technique uses event prediction to analyze the risk associated with skipping a checkpoint, and allows the system to perform only those checkpoints that it perceives as being most important. A summary follows.

When an application initiates a checkpoint at time t , progress on that job is paused for the *checkpoint overhead* (C) after which the application may continue. The *checkpoint interval* (I) is the time between the end of the previous checkpoint and the next checkpoint request. Let b_{i-1} be the time to which progress would be rolled back in the event of a failure. This may be either the start of the most recently-completed checkpoint or the time at which the application was first started. Let b_i be the time at which application j requests checkpoint i for $i \geq 1$, and let f_i be the time at which checkpoint i is completed. Let b_{i+1} be the time at which the next checkpoint will be started. Because checkpoints often occur at regular intervals, this value is relatively predictable. We define I to be the checkpoint interval such that $I = f_{i-1} - b_i \forall i \geq 1$, unless checkpoint $i-1$ is skipped, in which case the interval is $dI = f_{i-d} - b_i$, where $i-d$ is the last checkpoint that was performed.

Let C_i be the checkpoint overhead for checkpoint i of job j under the system conditions at time b_i . Note that

$C_i = f_i - b_i$, or 0 if the checkpoint is skipped. For a typical system, it is possible to predict C , as well as I , with relative accuracy by drawing on system-level performance guarantees and prior application behavior. Job j runs on n_j nodes. We define a unit of work to be a node-second, so occupying n nodes for k seconds consumes work $n \cdot k$.

If the expected amount of lost work before checkpoint $i + 1$ is completed is greater than the cost of checkpointing, then perform the checkpoint. This strategy is called *risk-based checkpointing* [11]. Let p_f be the probability that the partition on which job j is running will fail before f_{i+1} . The expected cost of skipping the checkpoint is $p_f((d + 1)I + C_{i+1})$, with no cost if a failure does not occur. The cost of performing the checkpoint is $p_f(I + C_{i+1} + C_i) + (1 - p_f)C_i$. Using $C_{i+1} \approx C_i = C$, this reduces to the heuristic for risk-based checkpointing, which is expressed in Equation 1. The system performs the checkpoint if the inequality holds.

$$p_f d I \geq C \quad (1)$$

The failure may occur before the completion of checkpoint i . Assuming the failure occurs independently of our decision, then the cost of either choice is the same. The system also uses the ability to skip checkpoints as a strategy for meeting deadlines. Even if $p_f d I \geq C$, the checkpoint will be skipped if doing so might allow a job to meet a deadline that it would otherwise miss.

3.5 Negotiation and Metrics

A major contribution of this paper is the introduction of a unique dialog between the system and the user. When submitting a job, the system and user negotiate a deadline. The system gives users an incentive to relax their deadlines: higher probabilities of success. To give an example of how this system may be used, consider the following situation. A user submits a job with a relatively tight deadline. This job j only has space and time to run on the cluster once, and in a single partition P . In other words, the scheduler has no choice in job placement. The system will consult the prediction mechanism and ask, ‘‘How likely is it that partition P is going to fail during the execution of this job with running time E_j ?’’ The predictor responds with probability p_1 of failure. The system tells the user that it can complete the job by the given deadline with probability p_1 , but that relaxing the deadline to a later time, such as t_d , increases the probability of success to p_2 . (Where $p_2 > p_1$.) In this way, the user is given an accurate sense of how likely it is that his request will be filled and gives him incentive to relax his requirements.

We consider the following standard metrics [9, 13, 11]: utilization and lost work. In our calculations, we used the so-called ‘‘last start time’’ of each job, which is the latest time at which the job was started in the cluster. There may

be many start times, because a failed job returns to the wait queue. In order to be consistent, and because checkpoints should be optional, we treat checkpointing overhead as being unnecessary work. That is, we use the execution time of the job *without* checkpoints (e_j). Calculations excluding checkpoint overheads are more accurate representations of the performance of the cluster; if the checkpoints could be skipped, the baseline optimal may be improved.

Each job has an associated arrival time (v_j), last start time (s_j), and the completed ones have a finish time (f_j). We calculated system *capacity utilized* and *work lost* based on the following formulations. Define a unit of *work* to be a single node occupied for one second. That is, occupying n nodes for k seconds consumes work $n \cdot k$ node-seconds. If $T = (\max_{\forall j} (f_j) - \min_{\forall j} (v_j))$ denotes the time span of the simulation, and N is the size of the cluster, then the capacity utilized (ω_{util}) is the ratio of work accomplished to computational power available. That is, imagine that all jobs had arrived simultaneously, and were completed without failure, without checkpoints, and that the system was being used at 100% capacity the entire time. Utilization is the ratio of actual behavior to that behavior. Recall that a job j runs on n_j nodes for e_j seconds, excluding checkpoints.

$$\omega_{\text{util}} = \sum_{\forall j} \frac{e_j n_j}{TN}$$

Let t_x be the time of failure x , and j_x be the job that fails as a result of x (j_x may be *null*). If c_{j_x} is the time at which the last successful checkpoint for j_x started, and n_{j_x} is the size of j_x in nodes, then the amount of work lost as a result of failure x is $(t_x - c_{j_x})n_{j_x}$ (this equals 0 for $j_x = \text{null}$). Hence, the total work lost (ω_{lost}) is calculated by the following equation:

$$\omega_{\text{lost}} = \sum_{\forall x} (t_x - c_{j_x})n_{j_x}$$

Given a cluster of fixed size N , failure distribution, and a set of jobs (so that n_j and e_j are fixed $\forall j$), maximizing ω_{util} is equivalent to the goal of minimizing T , while minimizing ω_{lost} requires minimizing the expected distance between a failure and the last checkpoint, particularly for large jobs.

In addition to these standard ones, we propose a new metric: *QoS*. We claim that this is the most important metric, and it is measured as follows. As before, let e_j be the execution time, and n_j be the size in nodes. Let q_j be an indicator variable such that $q_j = 1$ if j finishes by its deadline, and 0 otherwise. Let p_j be the probability of success for j that the system promised the user at submission time. Thus, *QoS* is defined as

$$QoS = \frac{\sum_j e_j n_j q_j p_j}{\sum_j e_j n_j} \quad (2)$$

In words, QoS measures how well the system kept the promises it made to the users. Larger jobs and those for which high probabilities were promised are most important to the system. Jobs that miss their deadline are useless to this metric. The scheduler behaves in such a way as to maximize QoS. This implies promising only as much as it can deliver, and delivering as much as it can. Because the metric rewards meeting deadlines and punishes failed jobs, QoS is as much a measure of reliability as of performance.

4 Experiments

We perform quantitative comparisons among various checkpointing and system parameters using a simulation-based approach. An event-driven simulator is used to process actual supercomputer job logs and failure data from a large-scale cluster [18]. The simulations produce measurements of the metrics covered in Section 3.5.

4.1 Simulation Environment

The event-driven simulator models a 128-node general cluster configuration, where the nodes are homogeneous but may fail independently. The simulator is provided with a job log, a failure log, and other parameters (for example, the checkpoint overhead and checkpoint interval). The events include (1) *arrival events*, (2) *start events*, and (3) *finish events*, similar to other job scheduling simulators [9]. Additionally, the simulator supports (4) *failure events*, which occur when a node fails, (5) *recovery events*, which correspond to a failed node becoming available again, (6) *checkpoint start events*, indicating the start of a job checkpoint, and (7) *checkpoint finish events*, which correspond to the completion of a checkpoint. The simulator produces enough information about the behavior of the system to calculate average utilization, work lost to failures and, most importantly, the QoS according to Equation 2.

4.2 User Behavior

In order to test the behavior of the system under a large variety of parameters, the simulator models user behavior. The job logs we use do not include information about deadlines, and certainly not regarding *actual* deadlines. Previous work on parallel QoS guarantees [7] have run the jobs under a particular set of parameters, and used these parameters to generate artificial deadlines. Because our job scheduling model relies on a dialogue between the user and scheduler, in which a deadline and promised probability of success are negotiated, we defined user behavior in terms of these values (p and d) and the job being submitted. Specifically, user behavior is defined by a parameter U , which relates to the amount of risk the user is willing to accept. For a given job

Job Log	Avg n_j (nodes)	Avg e_j (s)	Max e_j (hr)
NASA	6.3	381	12
SDSC	9.7	7722	132

Table 1. Job log characteristics.

j , with promised probability of success p_j , a simulated user will accept the *earliest* deadline such that

$$p_j \geq U. \tag{3}$$

Thus, U ranges from 0 to 1. As an example, $U = 0.5$ implies that all users will extend their deadline as little as possible in order to expect at least a 50% chance of success. All jobs in the log are used. Thus, a deadline may be pushed arbitrarily far into the future, but no further than necessary to satisfy Equation 3. Recall that the simulator will never return $p_f > a$. Consequently, $a < U$ implies $p_f < U$, and so simulation results are insensitive to changes in U when $a < U$.

From the point of view of the system, the user risk strategy amounts to the difference between idle nodes and lost work. If the user submits a restrictive deadline without regard for the probability of success, nodes in the cluster are less likely to be idle, but jobs are more likely to experience failures. If, instead, the probability of success is of paramount importance, then nodes will frequently be idle, but less work will be lost to failures.

4.3 Workload and Failure Models

The simulations considered job logs from the parallel workload archive [5] to induce the workload on the system. The parallel job logs include a log from NASA Ames’s 128-node iPSC/860 machine collected in 1993 (referred to as NASA log henceforth), and San Diego Supercomputer Center’s 128-node IBM RS/6000 SP (1998-2000) job log (referred to as SDSC log). Each log contained 10,000 jobs. Some characteristics are shown in Table 1, where runtimes do not include checkpoints.

For failure logs, we used filtered traces collected for a year from a set of 400 AIX machines for a previous study on failure analysis and event prediction [18, 16], and the filtration techniques are similar to those used to filter BG/L failures [10]. (At the time, no supercomputer failure logs were publicly available.) This filtration essentially involves isolating system events that are of the highest severity (i.e. FATAL or FAILURE), and further filtering to remove *clusters* of events that share a root cause. By ‘failure,’ we mean any event that would lead to the immediate failure of a job. Each failure event in the log corresponds to a failed node, which would result in job failure if one was running on the node at the time. We use failures from the first 128 such machines, resulting in 1,021 failures, an average of 2.8 failures per

N (nodes)	C (s)	I (s)	a	U	downtime (s)
128	720	3600	[0,1]	[0,1]	120

Table 2. Simulation parameters. Workloads and failure behavior were generated from actual machine traces.

day. The MTBF in the cluster as a whole was 8.5 hours, so the average MTBF for any individual node was around 6.5 weeks. For comparison, a 4-rack BG/L prototype [10] saw one failure every 6.18 hours. Therefore, the timing and distribution of failures used in this study reflect the behavior of actual hardware and software in a large cluster.

Failure predictions in our simulations are deterministic across runs. Each failure in the log has an associated static *detectability*, p_x , between zero and one, assigned randomly. When the predictor is asked for the probability of failure of a particular node (or partition) in a given time window, it retrieves all the corresponding failures from the log and considers them in order of time. Once a failure is encountered such that $p_x \leq a$, p_x is returned as the probability of failure. Otherwise, the predictor returns 0. Therefore, the false positive rate is 0 and the false negative rate is $1 - a$. An additional consequence of this method is that the probability of failure returned for any partition will never exceed a . The rationale for that decision is that a low-accuracy predictor should not make predictions with high confidence. Failures are the only reason for a deadline to be missed; in reality there are numerous factors that may contribute to QoS.

4.4 Simulation Parameters

For this paper, we restricted our study to parameters of the kind we expect to see in a large supercomputer, like IBM’s BlueGene/L. Thus, we set the node downtime to 120 seconds, meaning every failed node is back up after 2 minutes; this matches the restart time of a BG/L node. We also set $C = 720$ seconds and $I = 3600$ seconds. It was shown [14] that the *checkpoint latency* (L) typically has an insignificant impact on checkpointing performance for realistic failure distributions. In a log with 10,000 jobs, the number of checkpoints performed outnumbered the failed jobs by orders of magnitude; the cost of C is paid far more frequently than the cost of *recovery time* (R). Furthermore, downtime in supercomputing clusters is typically extremely expensive, and resources are usually on-hand to minimize this. Therefore, the simulations used $C \approx L$ and $R = 0$.

The simulations used the SDSC and NASA supercomputer job logs described above, and used AIX cluster failure characteristics. Experiments varied the accuracy a of the event prediction mechanism, as well as U , from 0 to 1 in increments of 0.1. The experiments used a flat (all-to-all)

communication architecture. Results of these experiments are presented in the following section.

5 Simulation Results

We now present our results for the SDSC and NASA job logs on a flat cluster, arranged in terms of the various metrics: QoS, utilization, and lost work. We examine the dependence of these metrics on prediction accuracy and on user behavior. The results presented here necessarily represent a subset of the total simulations. We chose to include a particular graph either because it was representative of our results, or because it accentuated an important feature. Exceptional results are noted as such.

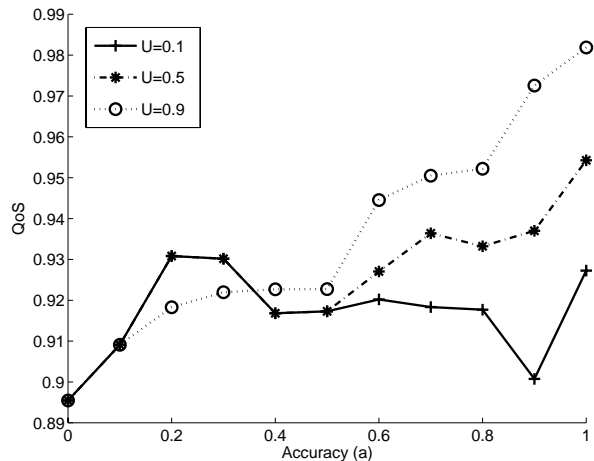


Figure 1. QoS vs. prediction accuracy, SDSC log, flat cluster, U = 0.1, 0.5, 0.9.

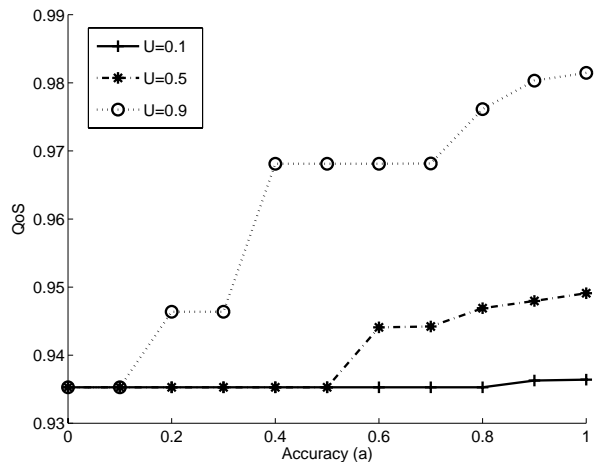


Figure 2. QoS vs. prediction accuracy, NASA log, flat cluster, U = 0.1, 0.5, 0.9.

5.1 Accuracy Sensitivity

In this section, we investigate the effects of prediction accuracy, a . As expected, the QoS tends to improve as more failures are accurately predicted, but we also found that the accuracy necessary depends largely on the user behavior. Figures 1 and 2 show QoS results with $U = 0.1$, 0.5, and 0.9. That is, each user accepts the earliest deadline that gives a probability of success greater than U . First, notice that QoS values tend to be in the 0.9 to 1 range for these failure logs. Recall that QoS is a measure not only of performance, but also of reliability. Second, the NASA job log does not show a benefit to QoS until around $a \geq U$, while SDSC shows an improvement even at $a = 0.1$. The NASA log contains job sizes that were powers of two and is a relatively lighter load, while the SDSC log contains jobs with generally longer running times and larger sizes. With odd-sized jobs, the SDSC log is more likely to experience temporal fragmentation. While generally considered bad for performance, fragmentation can benefit reliability; with event prediction, fragmentation means more opportunities to avoid failures.

The jaggedness of these curves is a consequence of several factors, including the burstiness of the failure distribution. More important, however, is the fact that we were limited by the availability of real failure traces. To our knowledge, there are no publicly available supercomputer RAS and failure traces. The only large-scale cluster failure log to which we had access was one we harvested. Because typical statistical failure models are poor indicators of actual system behavior [14], considering artificially-generated traces alone would have been misleading. Although the pursuit of a stochastic failure model is outside the scope of this paper, we believe it is worth further study.

Referring again to Figures 1 and 2, consider the case when $U = 0.1$, where the user does not particularly care about the probability of success, and typically chooses the earliest deadline. Under such user behavior, jobs tend to be scheduled on less stable partitions, and only very high-confidence predictions will cause the user to extend the deadline. For SDSC, QoS shows similar behavior at low accuracies as for $U = 0.5$. For NASA, no benefit is seen in QoS until $a = 0.8$, at which point the improvement is negligible. In all, these results demonstrate that much of the value of prediction can be negated if users ignore the probability of success, and operate purely based on the deadline.

By contrast, QoS has a very clear correlation to a when $U = 0.9$; QoS is nondecreasing as accuracy increases. Furthermore, QoS increases to nearly 1 (perfect) at perfect prediction. Because our predictor is idealized, with no false positives, the simulator was able to achieve QoS of 1 when $a = 1$ and $U = 1$. These results show that users who extend their deadlines in light of failure predictions are more likely

to have their job complete successfully.

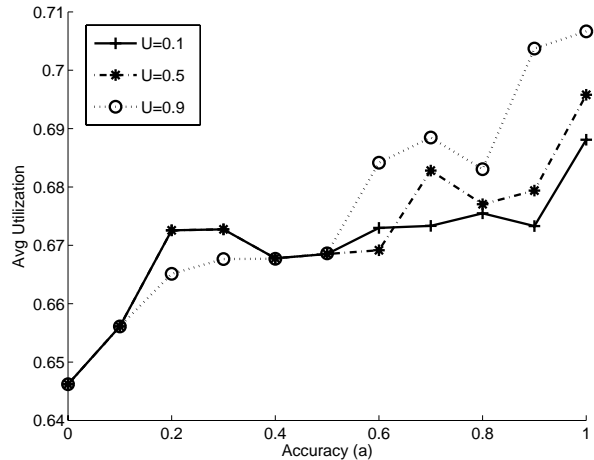


Figure 3. Average utilization vs. prediction accuracy, SDSC log, flat cluster, $U = 0.1, 0.5, 0.9$.

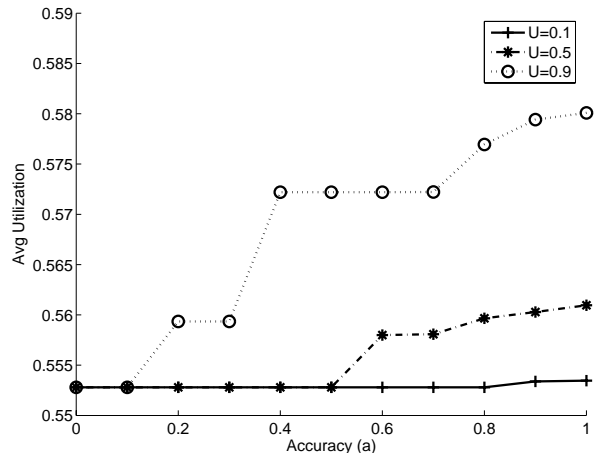


Figure 4. Average utilization vs. prediction accuracy, NASA log, flat cluster, $U = 0.1, 0.5, 0.9$.

One question raised with regard to a system that makes QoS guarantees is whether these promises come at the cost of utilization. We found that improvements in QoS coincided with increases in utilization and decreases in the amount of work lost to failures. Figures 3 and 4 show how utilization increases with accuracy. The equations for utilization and QoS share similar structure, so the relationship is not surprising. Lost work is shown in Figures 5 and 6. The SDSC log typically resulted in 10 times the amount of lost work as the NASA log. Figure 6 reveals that low prediction accuracy succeeds in reducing lost work, despite having no impact on QoS or utilization. For higher values of U , less work is lost to failures; when users pay attention to the predicted probabilities of success, the overall efficiency of the system is increased.

To summarize, QoS is closely related to average sys-

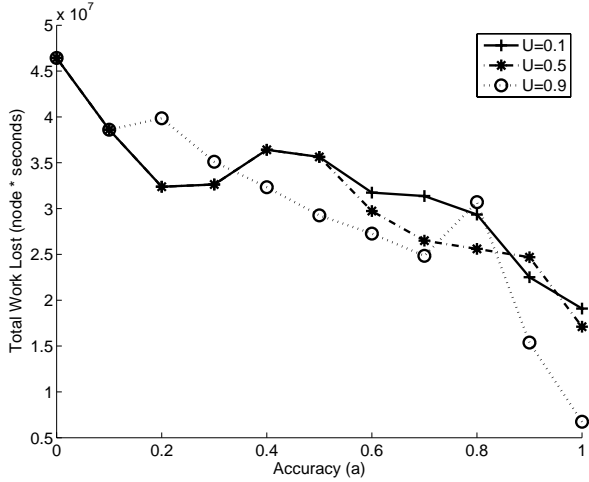


Figure 5. Lost work vs. prediction accuracy, SDSC log, flat cluster, $U = 0.1, 0.5, 0.9$.

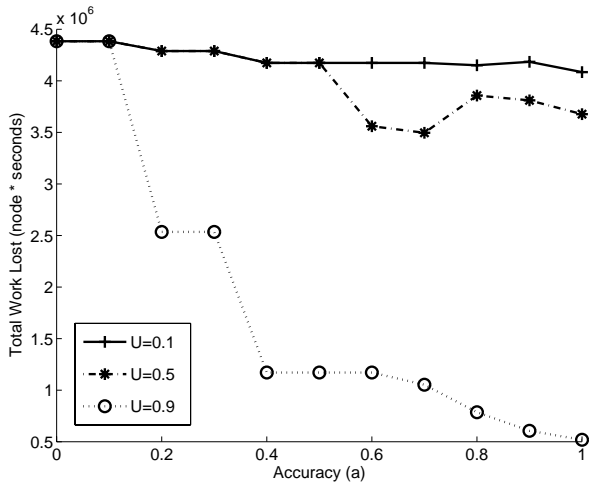


Figure 6. Lost work vs. prediction accuracy, NASA log, flat cluster, $U = 0.1, 0.5, 0.9$.

tem utilization, and an improvement in one typically corresponds to an improvement in the other. In addition, the amount of work lost to failures tends to be lower when QoS is higher. In order to maximize QoS, and therefore improve utilization and decrease lost work, it is important for both the system and the users to be honest. The more accurate the predictions, the better the QoS. The later the deadlines to which the users are willing to agree, the better the QoS. When users assigned high importance to the probability of success when setting a deadline, we observed QoS increases as high as 4%, corresponding with utilization improvements of 3%. Under the same conditions, the amount of work lost to failures dropped by a factor of 9 (from 4.5 to 0.5 in Figures 5 and 6). Similar trends were observed when varying the accuracy of the predictor. Perfect prediction enabled QoS and utilization improvements as high as 6%.

5.2 User Behavior Sensitivity

In this section, we examine the behavior of the metrics with respect to user behavior. When $a < U$, the metrics do not vary with U . This is because no partition will have a probability of failure greater than U , and thus the user behavior parameter will never come into play. Figure 7 illustrates this phenomenon. In order to show the behavior of the metrics at all values of U , the remainder of the results we present are for $a = 1$. Consider QoS versus U in Figure 8. As discussed above, QoS tends to increase with U . In other words, the higher the probability of success required by the users, the better the system is able to meet promised deadlines.

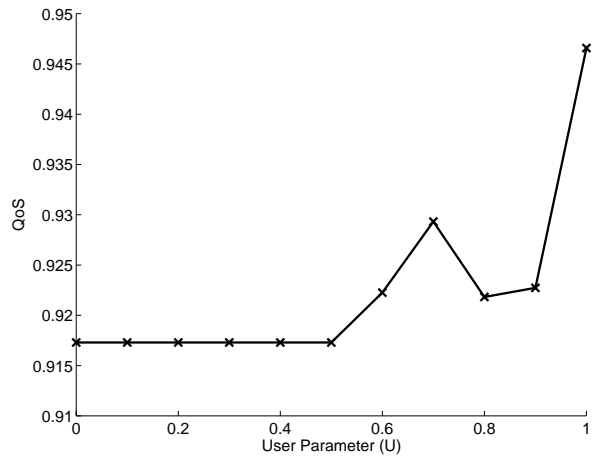


Figure 7. QoS vs. user behavior, SDSC log, flat cluster, $a = 0.5$.

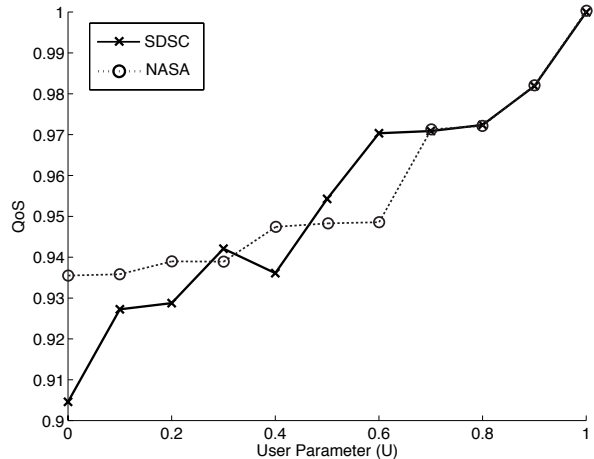


Figure 8. QoS vs. user behavior, flat cluster, $a = 1$.

As expected, average utilization and total work lost improve with QoS; higher values of U correspond to increased utilization and less lost work. When users extend their

deadlines, the scheduler has greater freedom with respect to where and when jobs can be run. This leads to increased performance and reliability. Figures 9 and 10 show utilization, and Figures 11 and 12 show lost work.

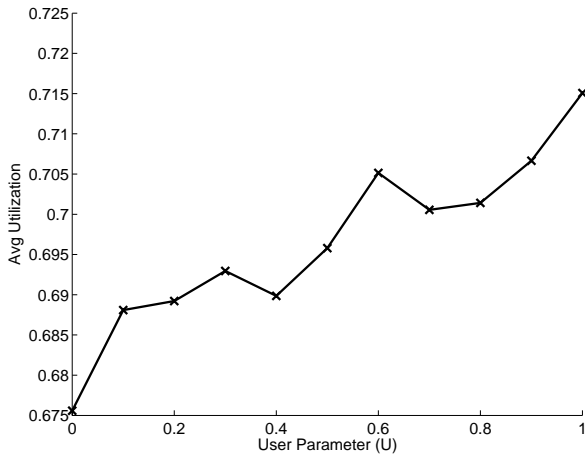


Figure 9. Average utilization vs. user behavior, SDSC log, flat cluster, $a = 1$.

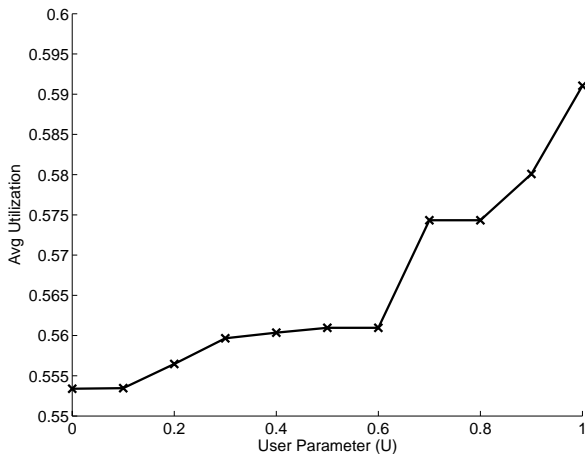


Figure 10. Average utilization vs. user behavior, NASA log, flat cluster, $a = 1$.

6 Contributions

This paper has presented the design, implementation, simulation, and analysis of a supercomputing system that makes probabilistic QoS guarantees. This section reviews those contributions.

- Describes the design of a supercomputing system that makes probabilistic QoS guarantees. The system makes promises of the form, “Job j can be completed by deadline d with probability p .” By giving the user access to this information, and thereby creating a

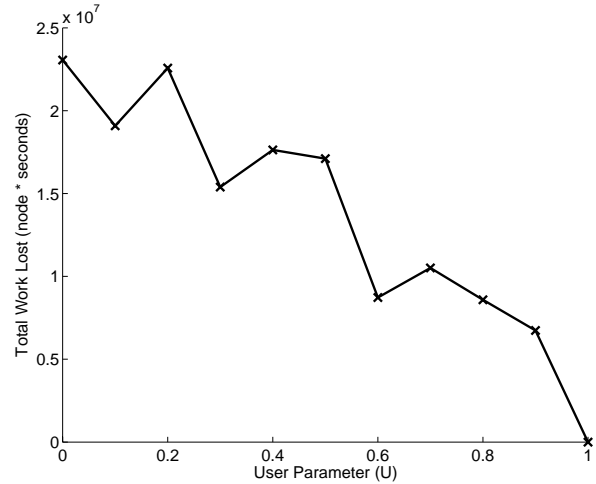


Figure 11. Lost work vs. user behavior, SDSC log, flat cluster, $a = 1$.

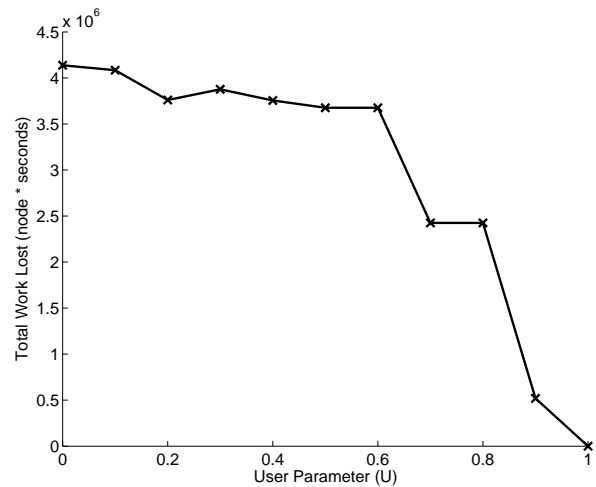


Figure 12. Lost work vs. user behavior, NASA log, flat cluster, $a = 1$.

market-driven scheduler, the control system is able to improve usability, reliability, and performance.

- Proposes how event prediction can be used in practice to make and keep promises in the presence of uncertainty. We believe the use of event prediction has widespread implications for improving the performance, usability, and reliability of computer systems.
- Details how a market-driven job scheduler and cooperative checkpointing mechanism can use event prediction to enforce QoS. We implemented such a system in a simulated supercomputing cluster environment, and tested its behavior on actual workloads and failure distributions. We examined the effects of event prediction accuracy and user behavior on several important performance metrics, including our definition of QoS.

- Defines a new probabilistic Quality of Service metric. As presented, QoS reflects not only the performance of the system, but also its reliability. It rewards a system for promising (only) as much as it can deliver, and delivering all it can.
- Demonstrates that a coordinated risk strategy between users and the system results in better QoS, utilization, and less lost work. As event prediction becomes more accurate, QoS, utilization, and work lost metrics all improve. Compared to a system without forecasting, high-accuracy prediction resulted in 6% increases in QoS and utilization, and reduced lost work by a factor of 9. With access to the probabilities of success, users have an incentive to relax their deadlines. The greater the extent to which they are willing to do this, the better those same metrics fare. Users who gave higher priority to the probability of success than to the deadline experienced QoS improvements of 4%, average utilization increases of 3%, and 9 times less work lost to failures.

This paper specifically targets supercomputing systems, but the algorithms and metrics may be applicable to any system in which job submission is used. Clearly, high-accuracy prediction, combined with users who have an incentive to relax their deadlines, results in a supercomputer with higher performance and efficiency at all levels. Fault-aware job scheduling and cooperative checkpointing are effective techniques for enforcing the QoS guarantees made by the system. With the current availability of relatively accurate event prediction algorithms, the techniques explained in this paper represent a feasible and practical technique for creating an efficient, reliable, and usable supercomputing system.

References

- [1] C. Aurrecochea, A. T. Campbell, and L. Hauw. A survey of qos architectures. In *Multimedia Systems*, volume 6, pages 138–151, 1998.
- [2] M. F. Buckley and D. P. Siewiorek. Vax/vms event monitoring and analysis. In *FTCS-25, Computing Digest of Papers*, pages 414–423, Pasadena, CA, June 1995.
- [3] M. F. Buckley and D. P. Siewiorek. A comparative analysis of event tupling schemes. In *FTCS-26, Intl. Symp. on Fault-Tol. Computing*, pages 294–303, June 1996.
- [4] P. Dinda. A prediction-based real-time scheduling advisor. In *IPDPS*, 2002.
- [5] D. G. Feitelson. Parallel workloads archive. <http://cs.huji.ac.il/labs/parallel/workload/index.html>, 2001.
- [6] J. M. Hyman, A. A. Lazar, and G. Pacifici. Real-time scheduling with quality of service constraints. In *IEEE Journal on Selected Areas in Communications*, number 9 in 1, pages 1052–1063, Sept. 1991.
- [7] M. Islam, P. Balaji, P. Sadayappan, and D. Panda. Qops: A qos based scheme for parallel job scheduling. In *JSSPP*, pages 252–268, 2003.
- [8] K. T. Kornegay, G. Qu, and M. Potkonjak. Quality of service and system design. In *Proceedings of the IEEE Computer Soc. Workshop on VLSI'99*, page 112, 1999.
- [9] E. Krevat, J. G. Castanos, and J. E. Moreira. Job scheduling for the bluegene/l system. In *JSSPP*, pages 38–54, 2002.
- [10] Y. L. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. E. Moreira, and M. Gupta. Filtering failure logs for a bluegene/l prototype. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, June 2005.
- [11] A. J. Oliner. Cooperative checkpointing for high performance computing systems. Master's thesis, Massachusetts Institute of Technology, 2005.
- [12] A. J. Oliner, R. K. Sahoo, J. E. Moreira, and M. Gupta. Performance implications of periodic checkpointing on large-scale cluster systems. In *IPDPS 2005 Workshop on System Management Tools for Large-Scale Parallel Systems*, 2005.
- [13] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for bluegene/l systems. In *IEEE IPDPS, Intl. Parallel and Distributed Processing Symposium*, Apr. 2004.
- [14] J. S. Plank and W. R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In *The 28th Intl. Symposium on Fault-tolerant Computing*, June 1998.
- [15] R. K. Sahoo, M. Bae, R. Vilalta, J. Moreira, S. Ma, and M. Gupta. Providing persistent and consistent resources through event log analysis and predictions for large-scale computing systems. In *SHAMAN, Workshop, ICS'02*, New York, June 2002.
- [16] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *ACM SIGKDD, Intl. Conf. on Knowledge Discovery and Data Mining*, pages 426–435, Washington, DC, Aug. 2003.
- [17] R. K. Sahoo, I. Rish, A. J. Oliner, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Autonomic computing features for large-scale server management and control. In *AIAC Workshop, IJCAI 2003*, Aug. 2003.
- [18] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 772–781, June 2004.
- [19] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: An economy driven job scheduling system for clusters. In *Proceedings of the 6th. Intl. Conf. on High Performance Computing (HPC Asia 2002)*, Bangalore, India, 2002.
- [20] M. M. Tsao. *Trend Analysis and Fault Prediction*. PhD dissertation, Carnegie-Mellon University, May 1983.
- [21] R. Vilalta and S. Ma. Predicting rare events in temporal domains. In *Proceedings of IEEE Conf. on Data Mining (ICDM'02)*, Japan, 2002.
- [22] G. M. Weiss and H. Hirsh. Learning to predict rare events in event sequences. In *Proceedings 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD'98)*, pages 359–363, 1998.