

Fault-aware Job Scheduling for BlueGene/L Systems

A. J. Oliner

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139-4307 USA
e-mail: oliner@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,moreira,mgupta}@us.ibm.com

A. Sivasubramaniam

Department of Computer Science and Engineering
Pennsylvania State University
316 Pond Laboratory, University Park, PA 16802-6106 USA
e-mail: anand@cse.psu.edu

Abstract

Large-scale systems like BlueGene/L are susceptible to a number of software and hardware failures that can affect system performance. In this paper evaluate the effectiveness of a previously developed job scheduling algorithm for BlueGene/L in the presence of faults. We have developed two new job-scheduling algorithms considering failures while scheduling the jobs. We have also evaluated the impact of these algorithms on average bounded slowdown, average response time and system utilization, considering different levels of proactive failure prediction and prevention techniques reported in the literature. Our simulation studies show that the use of these new algorithms with even trivial fault prediction confidence or accuracy levels (as low as 10%) can significantly improve the performance of the BlueGene/L system.

1. Introduction

The demand for high computational power continues to drive the design and development of large-scale parallel systems. In addition to the existing applications taking on larger problems/datasets, there are entirely new application domains (e.g., drug discovery) which need considerably more computational resources than what is available today. Large scale parallel machines, such as IBM's BlueGene/L (BG/L henceforth), which has 65536 compute nodes, are expected to play a key role in taking on the demands of such applications. Since the probability of occurrence of a

failure grows linearly with the number of nodes (and perhaps even faster if failures are correlated), faults are likely to be quite frequent on these machines. Even though BG/L has been designed with a strong emphasis on reliability, we expect it to have about one fatal error each day. We expect transient faults to be much more frequent than permanent faults. It is imperative to ensure that we do not incur severe performance degradation when faults occur. Towards this goal, this paper presents and evaluates two techniques for scheduling jobs on a large scale parallel machine (IBM's BG/L in particular) that exploits prediction of faults to improve system utilization.

Scheduling of parallel jobs on large scale parallel machines has a significant impact on job response times and the overall system utilization. It is computationally intensive to implement a perfect schedule even from the theoretical perspective. On the practical side, there are considerations about how to share the machine across multiple jobs – should nodes allocated to a job be reserved until its completion (referred to as space sharing) or should they time share across jobs. Space sharing reduces context switch overheads, while time sharing can provide better response times by avoiding fragmentation. In this study, our focus is on space sharing systems, which is the design point chosen for BG/L for simplicity of system software. Furthermore, previous studies [6, 11] have shown that space sharing enhanced with techniques such as backfilling, can do as well, if not better, than time sharing. The implementation of space sharing is itself quite complex in BG/L, which has a toroidal architecture that puts restrictions on how nodes can be allocated. Nodes need to be allocated as contiguous

(tridimensional) rectangular partitions, which makes communication between those nodes more efficient (helps avoid contention with traffic from other jobs) and secure. Such a restriction adds to the complexity of scheduling nodes to jobs. Previous work [6, 11] has proposed heuristics for allocating jobs to rectangular partitions.

The occurrence of failures adds a new dimension of complexity to the scheduling problem. A naive strategy could be to ignore the possibility of occurrence of failures and deal with them when the faults occur (restart the jobs affected). However, such an algorithm can lead to waste of resources due to jobs affected by faults. As our results will show, even for a fault occurrence rate of about one failure in four days, there is nearly a 70% increase in job slowdown when one does not make any provision for the occurrence of faults in the job scheduler. This serves as a motivating reason for developing fault-aware scheduling mechanisms for these large-scale machines.

There are several ways to address this loss in performance under the presence of faults. One way is to periodically checkpoint the jobs, or at least checkpoint it close to the time when one of its nodes is likely to fail (if that can be anticipated), to reduce the amount of work that needs to be re-done upon occurrence of a fault. This requires support in the OS for checkpointing and recovery, which are provisioned in BG/L. An alternative (or even complementary, since it can be used in conjunction with checkpointing) strategy is to perform fault-aware scheduling, where prediction of faults can be used to allocate jobs to partitions that are likely to be healthy. This paper investigates the benefits of this fault-aware scheduling strategy by presenting and evaluating two techniques, extensions to Krevat's original BG/L scheduler [11], which exploit the prediction information about node failure for space sharing. The basic idea is to allocate a partition picked by Krevat's scheduler only when it is not likely to fail over the duration of that job. This paper makes the following contributions:

- We show that scheduling mechanisms that do not consider the occurrence of faults can incur significant performance penalties on a large scale machine such as BG/L.
- We present two simple techniques for extending spatial (space sharing) scheduling to make them fault-aware, and demonstrate their performance benefits with several real workload logs from supercomputing environments.
- We demonstrate that fault-aware scheduling can be effective even with modest prediction accuracy.

The rest of this paper is organized as follows. Section 2 describes related work on job scheduling for large-scale cluster systems. Section 3 presents a brief description of the

job-scheduling problem, including BG/L-specific requirements. Section 4 describes various event prediction mechanisms followed for the present study and the motivation to develop new job-scheduling algorithms. Section 5 presents our fault-aware job-scheduling algorithms. Section 6 describes the simulation environment, job logs, the methodology to link job logs with the failure logs, and various parameters considered for the study. Section 7 reports experimental results. Finally, we conclude the paper with a summary of the results and our future work plans in Section 8.

2. Related Work

With an increased popularity of large-scale clusters in high performance computing community, there has been much interest in attaining better system utilization for these environments, including management of system failures [3, 4, 12]. To deal with faults, most research efforts focus on providing failover mechanisms either within the operating/programming environment or through application checkpointing. All these efforts add additional overhead and complexity not only to programming environment, but also to the application running environments including additional hardware costs.

There are a number of research efforts analyzing job scheduling and impact of job scheduling on system performance for large-scale parallel systems [5, 6, 8, 9, 10, 11, 22]. Most of these studies address either temporal or spatial job scheduling [19, 20, 21] considering checkpointing [13], data locality [15], type of workload, and operating environment for fault tolerant scheduling [2, 14] to name a few. For large-scale systems like BG/L [1] and Earth Simulator [22] there are very few research efforts considering job scheduling in presence of system failures. Recently, a number of statistical and machine learning based failure prediction techniques [17, 18, 23] have been proposed for proactive system management. Since job scheduling and job scheduling algorithms play an important role in evaluating the system performance [6, 22], within this paper we extend some of our prior prediction efforts [16, 17, 23] to address the job scheduling mechanism for BG/L. We evaluate some of our new job-scheduling algorithms developed to handle system failures including a strategy for failure prediction.

3. Fault-aware Job scheduling for BlueGene/L

Our fault-aware job scheduling study targets improving the system performance, efficiency and throughput in the presence of failures. We first present background information on BG/L.

3.1. BlueGene/L Architecture

The BG/L computer system [1] is a $32 \times 32 \times 64$ three-dimensional torus of compute nodes (cells). The compute nodes are also interconnected in a tree topology, which is used for reduction operations and for I/O. Communications with the external environment is accomplished through 1024 Gigabit Ethernet links attached to the I/O nodes, placed at specific points in the tree interconnect.

Most systems with toroidal interconnects, including BG/L, are limited by certain constraints when scheduling jobs [11, 5, 6]. Jobs are required to be placed in distinct, contiguous, rectangular partitions. Each job on BG/L is scheduled on an electrically isolated partition. This ensures that communication traffic for one job cannot interfere with the traffic for another job, and enables the system software to be kept simple, as protection across jobs is ensured by hardware isolation. In order to satisfy these requirements, a job partition on BG/L must be composed as a three-dimensional rectangle of $8 \times 8 \times 8$ node blocks. Hence, a job scheduler sees BG/L as a $4 \times 4 \times 8$ torus of these *supernodes*, with each supernode having 512 compute nodes.

3.2. Input to Scheduler

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 (s_j) and the estimated execution time of the job (t_j^e). After a job j has been scheduled to start at time t_j^s , the scheduler can compute the estimated completion time of the job ($t_j^f = t_j^e + t_j^s$). Once a job completes execution, the estimated value for t_j^f is replaced by its actual value. The fault prediction algorithm accepts a node and a time window, and outputs an estimated probability that the node will fail within that time window [17, 23]. The scheduler is able to migrate jobs around the torus. In order to migrate a job in BG/L, the job must first be checkpointed, then moved to and restarted on a new partition. The scheduler is also able to force jobs to checkpoint. However, the present study does not consider checkpointing.

3.3. Scheduling Constraints

The scheduler operates under the following constraints, which are based on earlier job scheduling work for BG/L [11] and new constraints related to failures.

- Only one job may run on a given node at a time. (No co-scheduling, or multitasking at each node.)
- Job partitions must be contiguous and rectangular in three dimension.

- Nodes are imperfect and may fail at any time; if a job is running on a node when it fails, all unsaved work on that job is lost.

3.4. Goal

The goal of the job scheduler is to minimize the job wait time and system idle time, and maximize the system utilization. Our scheduler attempts to minimize metrics similar to metrics considered in Krevat’s scheduler [11]. The actual job execution time are calculated based on start time t_j^s and (actual) finish time t_j^f of each job. Similarly, t_j^s and t_j^f , along with job arrival time (t_j^a) can be used to calculate wait time $t_j^w = t_j^s - t_j^a$, response time $t_j^r = t_j^f - t_j^a$, and bounded slowdown $t_j^b = \frac{\max(t_j^r, \Gamma)}{\min(t_j^e, \Gamma)}$, where $\Gamma = 10$ seconds. Our objectives of optimizing the job-scheduling algorithm falls into two categories: timing metrics and utilization metrics. Our timing goals are to minimize: (1) $\{Average[t_j^w]\}$, (2) $\{Average[t_j^r]\}$ and (3) $\{Average[t_j^b]\}$. Our utilization goals are to maximize system utilization and minimize lost capacity, as defined in Section 6.

4. Prediction Mechanism

A prediction mechanism analyzes and predicts the probability of occurrence of an event or survival of a node within a specified time window in the future [17]. There may be false-positives or false-negatives associated with the prediction. In this study, rather than using an actual prediction algorithm (*predictor*), we use “*job log*” traces and associated “*failure log*” traces to provide information on failures. We simulate different levels of accuracy of a prediction algorithm using a parameter “ a ” that represents the *confidence* (for balancing scheduler – Section 5.2.1), or *accuracy* (for tie-break scheduler – Section 5.2.2). More details of job log traces and failure log traces used for simulation are explained in Section 6.

4.1. Balancing Predictor

The balancing predictor is used in the context of balancing scheduler algorithm (Section 5.2.1). The prediction of $p_n^f(s)$, the probability of failure of a node n during the execution of a job for s seconds (based on job logs and associated failure logs) is obtained as:

- a , if the failure log contains a failure event for node n in the next s seconds, or
- zero, if the failure log does not contain a failure event for node n in the next s seconds.

The probability that the partition P will fail within s seconds is computed as: $P_f(s) = \max_{n \in P}(p_n^f(s))$. More details on the probability parameter (P_f) estimation are covered in Section 5.2.1.

4.2. Tie-Breaking Predictor

The tie-breaking predictor used in the context of tie-breaking algorithm (Section 5.2.2) is much simpler than the balancing predictor. It takes a boolean value, and is used to break the tie between two partitions of the same size. When asked about the possibility of failure of node(s) or partition(s) within the next s seconds, the tie-breaking predictor responds with the following:

- If the failure log contains a failure event for node n in the next s seconds, the predictor responds ‘yes’ with probability a , and ‘no’ with probability $1 - a$. That is, the probability of a false negative p_{f-} is $1 - a$.
- If the failure log does not contain a failure event for node n in the next s seconds, the predictor returns ‘no’, that is, there are no false positives.

A *tie-breaking* algorithm only needs to predict *if* a partition is going to fail, and does not care, for the purposes of scheduling, about the probability with which a failure would happen. Given a partition and a time window, the new predictor must assert whether or not it expects any node in that partition to fail within that window. When every candidate partition is predicted to fail, the scheduler makes an arbitrary choice. Instead of choosing a new confidence parameter for tie-breaking predictor, we choose an *accuracy* parameter based on the false-negatives associated with the probability of prediction used for balancing predictor. By *accuracy* (a), we define $1 - p_{f-}$, where p_{f-} is the probability of a false negative associated with the prediction. We chose to leave false positives out of our analysis, since previous studies of actual cluster failure data [17, 23] have shown that even a very simple predictor can keep the probability of a false positive p_{f+} well below p_{f-} . In fact, p_{f+} has been shown to be typically less than half of p_{f-} values.

5. Algorithms

Most of previous job scheduling algorithms are based on system performance optimization. The optimizations are targeted either to minimize average bounded slow-down, average response time and/or to maximize system utilization. This section covers the new algorithms proposed including a brief mention of the previous job scheduling algorithm developed for BG/L. We propose two new scheduling algorithms: the *balancing* algorithm and the *tie-breaking* algorithm. Each of these algorithms use fault prediction to improve system performance for job scheduling. Additionally, we developed a new algorithm for obtaining candidate partitions when scheduling a job, which has a better asymptotic running time than previous attempts [11].

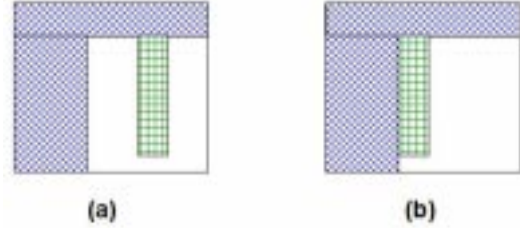


Figure 1. Placing a job according to the MFP heuristics. Placing the job as in a results in a smaller MFP than placement b , so the scheduler would prefer the partition in b .

5.1. Previous Algorithm

There are a number of studies [5, 6, 11] to evaluate job scheduling algorithms for toroidally connected system like BG/L. Krevat et al. [11] have developed a job scheduling specifically targeting job scheduling for BG/L. The algorithm takes into account the rectangular constraint requirement to avoid job fragmentation for better system performance. Krevat’s algorithm [11] is based on *first come first serve (FCFS)* with or without considering backfilling and/or job migration [11, 19, 20, 21]. This algorithm for BG/L puts all the arriving jobs in a queue of *wait jobs*, prioritized according to the order of job arrival. For every job arrival and termination the scheduler is invoked for scheduling of new jobs. The scheduler uses a maximal free partition (MFP) heuristic, illustrated in Figure 1, to schedule the jobs. An MFP is defined as the maximum contiguous rectangular partition available for a job. In the next step the scheduler calculates the MFP using a Projection of Partitions (POP) algorithm. The POP algorithm uses a dynamic programming approach with a time complexity of $O(M^5)$ [11].

5.2. Job Scheduling Algorithm with Faults

We develop two new job scheduling algorithms to deal with the faults in a proactive manner.

5.2.1 Balancing Algorithm

The *balancing* algorithm is based on the FCFS scheduler for BG/L [11], including backfilling and migration, using a new set of heuristics for job placement. The heuristics for selecting a partition for scheduling a job on the torus are modified to include the integrity of the partitions according to a fault-prediction algorithm. Given a torus (mostly with existing running jobs), and a job of particular size (s_j) and expected running time (t_j^e), the algorithm selects a contiguous rectangular partition P of s_j nodes for job placement. The new heuristic considers two factors: (1) the difference

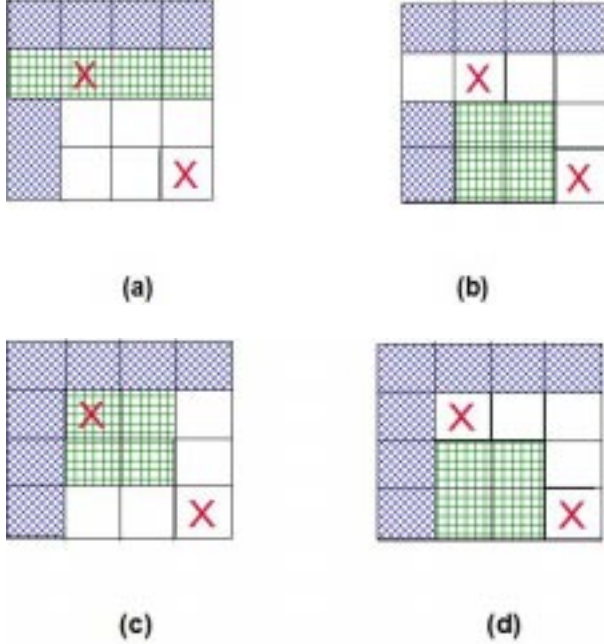


Figure 2. Two possible choices for job placement. Each X marked box represents the chances that these nodes will fail in future. Placement a has a larger MFP, but the partition is predicted to fail. The job placement depends on the confidence parameter of the prediction. With two placements that result in an equal optimal MFP (c and d) the placement d is better than c.

between the size of the maximal free partition (MFP) before and after placing the job in the candidate partition (L_{MFP}), and (2) the expected loss due to node failure as a result of placing the job in the candidate partition (L_{PF}).

The first factor, L_{MFP} , defines the amount of free space we would “lose” as a result of placing the job in a particular partition versus not placing it at all. This is a loss from the point of view of any job immediately following the one being placed, which may require a partition as large as an MFP. The L_{MFP} factor is calculated by computing the MFP difference before and after placing the job within a torus and candidate partition, respectively. We calculate MFP using a new partition algorithm (Appendix 9) in place of Krevat’s POP algorithm. Our Partition finder algorithm provides a $O(M^3)$ solution for finding maximum free partition.

The loss due to possible failure (L_{PF}) is calculated by finding the probability that a partition P will fail before the job completes execution. In other words, if each node n in a partition has a probability of failure p_n^f between now and the estimated completion time of the job, then the probability that the partition will fail is $P_f = 1 - \prod_{n \in P} (1 - p_n^f)$ and the expected loss is $L_{\text{PF}} = P_f \times s_j$. Hence, the balancing

algorithm tries to compute the total expected loss (E_{loss}) defined as: $E_{\text{loss}} = L_{\text{MFP}} + L_{\text{PF}}$. This is the worst-case expected loss associated with any job size s_j , as it is assumed that the job fails just before completion, and there are no checkpointing performed in case there are failures.

Results from our initial runs using the balancing algorithm demonstrate that trading MFP size for a more stable partition is not always beneficial for overall system performance or utilization. When the confidence parameter is high, the algorithm starts selecting a stable partition, even if it is required for the waiting jobs to wait much longer. In other words, the algorithm pushes FCFS to an extreme, seeking to minimize the finishing time of the next job at the expense of all those jobs after it. A low confidence parameter, such as 0.1, would usually cause the scheduler to pick the partition with the largest MFP, since the E_{loss} would be $\approx 1/10$ the job size. Only when two partitions, with equal or similar MFP, have different P_f values, the algorithm would select the more stable one. Such an occurrence can be well explained through Figures 2 (a) and (b). In Figure 2(a) there is a L_{MFP} of 3 and $L_{\text{PF}}=4P_f$, whereas for Figure 2(b), we have $L_{\text{MFP}}=5$ and $L_{\text{PF}}=0$. Hence, when choosing between (a) and (b), the P_f would be the deciding factor. From simple calculations, it is obvious that the choice between (a) and (b) are decided based on whether P_f is greater than 0.5. This problem becomes more prominent for larger MFPs and needs another algorithm to break the tie between two free partitions of same size. This led us to develop the *tie-breaking* algorithm.

5.2.2 Tie-Breaking Algorithm

The tie-breaking algorithm has been formulated to break a tie when the scheduler arrives at a situation with two or more partitions available for a job. As shown in Figure 2(c) and (d), if there are two situations with same MFP, then the scheduler is allowed to break the tie by calculating the accuracy parameter for both the partitions. If X represents the chance of failing nodes, certainly choice (d) is a better solution than choice (c). It uses the tie-breaking predictor in place of confidence parameter as described in Section 4.

6 Experiments

We perform quantitative comparisons among the schedulers 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. The simulations produce information regarding the efficiency of the schedulers, according to various metrics covered in Section 3.

6.1 Simulation Environment

The event-driven simulator models a 128 (super)node torus in a three-dimensional $4 \times 4 \times 8$ configuration. The

simulator is provided with a job log, a failure log, and other parameters (for example : prediction confidence level, load scale coefficient). The events include three parameters :(1) *arrival events*, (2) *start events*, and (3) *finish events*, similar to other similar job scheduling simulators [11]. Additionally, the simulator supports (4) *failure events*, which occur when a node fails, and has a provision to add (5) *checkpoint events*. For the present set of experiments the checkpointing parameter has not been considered. Compared to earlier work [11], the following changes are considered while carrying out simulation runs.

- Jobs are always scheduled for immediate execution (i.e., there is no delay between being scheduled and starting execution).
- The failures we consider are transient. If a job is running on a node that fails, unsaved work on that job is lost and the node is immediately available for scheduling the same or other job(s).
- If a failure happens on one of the nodes assigned to a job while executing, then the entire job is assumed to have failed and the job has to be restarted from the beginning.

The simulation produces values for the last start time (t_j^s) and finish time (t_j^f) of each job, which are used to calculate wait time (t_j^w), response time (t_j^r), and bounded slowdown (t_j^b). Other parameters like system capacity *utilized*, *unused* and *lost* are calculated based on the following formulations. If $T = (\max_{\forall j} (t_j^f) - \min_{\forall j} (t_j^a))$ denotes the time span of the simulation, then the capacity utilized (ω_{util}) is the ratio of work accomplished to computational power available.

$$\omega_{\text{util}} = \sum_{\forall j} \frac{s_j t_j^e}{TN}.$$

If $f(t)$ denotes the number of free nodes in the torus at time t and $q(t)$ represents the total number of nodes requested by jobs in the waiting queue at time t , then amount of unused capacity, resulting from a lack of jobs requesting nodes can be calculated using the following equation.

$$\omega_{\text{unused}} = \int_{\min(t_j^a)}^{\max(t_j^f)} \frac{\max(0, (f(t) - q(t)))}{TN} dt.$$

Hence, the total lost capacity in the system, due to work lost from failures, an inability to schedule jobs, and the delay before a scheduled job can execute, is

$$\omega_{\text{lost}} = 1 - \omega_{\text{util}} - \omega_{\text{unused}}.$$

Clearly, given torus of fixed size N and a set of jobs (so that s_j and t_j^e are fixed $\forall j$), maximizing ω_{util} is equivalent to the goal of minimizing T .

6.2 Workload and Failure Models

We considered job logs from parallel workload archive [7] to induce the workload on the system. The parallel job logs include logs from NASA Ames’s 128-node iPSC/860 machine collected in 1993 (referred as NASA log henceforth), San Diego Supercomputer Center’s 128-node IBM RS/6000 SP (1998-2000) job logs (referred as SDSC logs), and Lawrence Livermore National Laboratory’s 256 node Cray T3D (1996) job logs (referred as LLNL job logs). In order to consider the effect of different loads on the system, in addition to injecting the job parameters directly from the trace, we also use a scaling factor “ c ” multiplied to each job’s execution time. The higher the “ c ”, the higher is the induced load on the system. Though we consider c values ranging from 0.5 to 1.5 in increments of 0.1, only results comparing for $c = 1$ and 1.2 are presented in this paper. This is due to the significant changes in system performance, when we increased the standard load by 20%.

For failure logs we used traces (after filtering and normalizing) collected for a year from a set of 350 nodes for a previous study on event prediction [17]. We varied prediction confidence and prediction accuracy respectively for the two algorithms from 0.0 to 1.0 in increments of 0.1.

Assuming similar workload and failure distribution for parallel workload archives, we scaled up/down the number of hardware failures for the study to have the same average number of failures per node and per day. Moreover, the failure timing considered for the workloads are based on actual collected hardware failure trace timings. Based on these calculations, we considered 4000 failures for each of NASA and SDSC job log based simulation studies, and 1000 failures for LLNL job log based studies. In addition, using the SDSC job logs as the basis, we studied the impact of different rates of failures by artificially varying the number of failures from 0 to 3500 or 4000 in intervals of 500.

7. Simulation Results

We present our results for the NASA, SDSC and LLNL job logs. Initially we cover the effect of failure rate followed by comparisons of the system performance with job scheduling (average bounded slowdown and average response time) for two different load scales (represented by $c = 1.0$, $c = 1.2$). We also analyze the effectiveness of the balancing and the tie-breaking algorithms in terms of system utilization and conversion of unused work to used work when the load parameter increases for these three different job logs.

7.1. Failure Distribution

The results of our experiments with failure densities demonstrate that failures can have a significant impact on

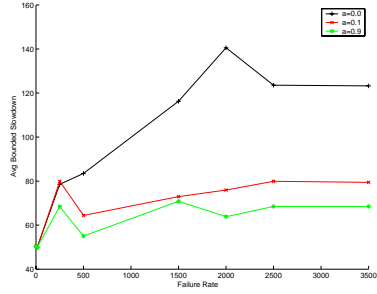


Figure 3. Average bounded slowdown vs. failure rate for SDSC job logs with and without prediction. $a = 0.0$ (no prediction), $a = 0.1$ and $a = 0.9$ (with prediction success 10% and 90% respectively)

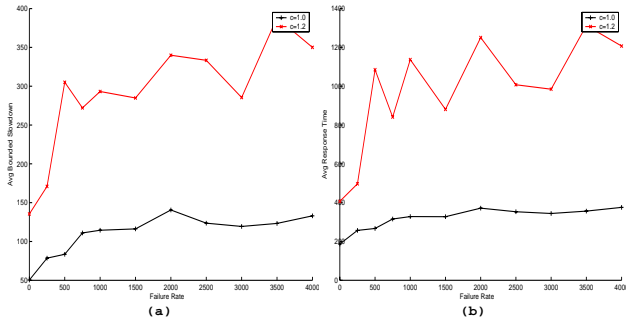


Figure 4. Average bounded slowdown vs. failure rate for SDSC job logs using balancing algorithm for different loads.

the performance of the system (Figures 3 and 4). As failures are introduced, system performance drops quickly, eventually reaching a point beyond which further increase in failure rate would have little additional effect. This failure traces contain many instances of multiple failure events, simultaneously reported from different nodes. As more and more of these failures appear for nodes from a single partition, an early possible partition failure declaration would put the partition out of the loop for a possible job submission till other stable partitions are available. Hence introduction of new failures to the same partition would not make the bounded slowdown or response time much worse. This is also due to our assumption that the nodes would reappear as normal nodes instantly once the job failure instance has been recorded through the job logs. Hence the job scheduler would not get affected by any of the new failures within an already failed partition. However, in reality the node failures would be associated with certain amount of down time. Hence appearance of multiple node failures would introduce the occurrence of the failure of any job which is assigned to the same partition. This would result a sharp rise in average bounded slowdown parameter.

The results in Figures 3 and 4 further show that, if there are on an average 1 failure per four days (corresponding to

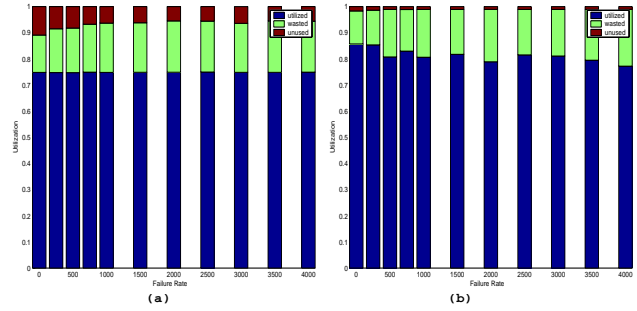


Figure 5. Utilization vs. failure rate for SDSC job logs using balancing algorithm (a) $c = 1.0$, (b) $c = 1.2$.

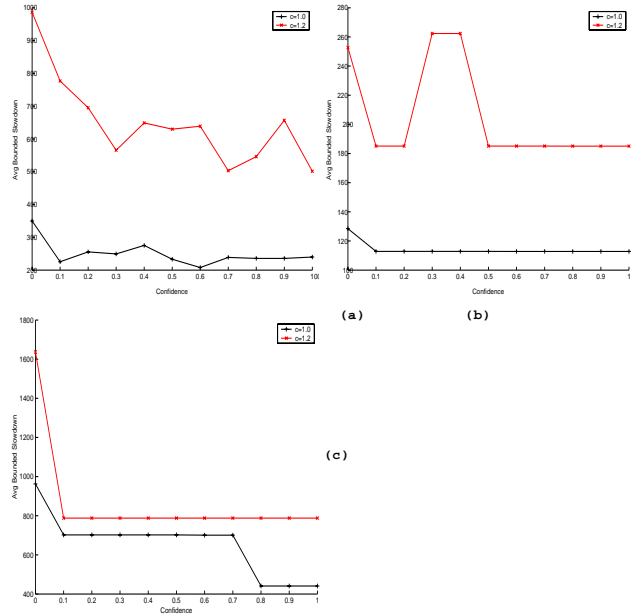


Figure 6. Average bounded slowdown vs. confidence for (a) SDSC, (b) NASA, (c) LLNL job logs using balancing algorithm.

1000 failure rate) even with a 10% prediction confidence, in the bounded slowdown parameter, our balancing algorithm leads to improvements up to 50%. However with the increase in prediction confidence level (from 10% to 90%) the rate of returns are comparatively low (another maximum improvement of 20%). A comparison of the utilization in Figure 5 also shows a 20% increase in load do not have any equivalent impact to bounded slowdown or average response time, apart from converting marginal amount of *unused* work to *used* work.

7.2. Balancing Algorithm

The following results reflect experiments performed using the balancing algorithm for SDSC, NASA and LLNL job logs. A comparison of the performance through confidence versus average bounded slowdown Figure 6, reveals, that maximum improvement for standard load would

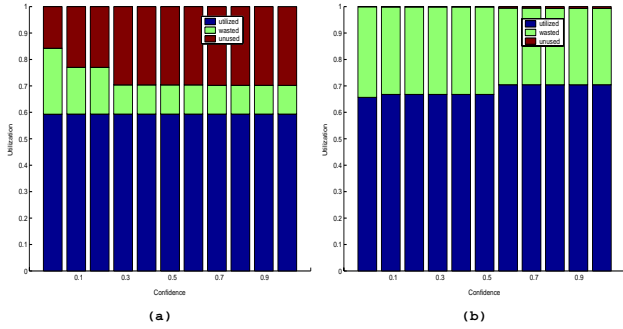


Figure 7. Utilization vs. confidence for NASA job logs using balancing algorithm (a) $c = 1.0$, (b) $c = 1.2$.

happen within the first 10% of prediction confidence. For higher loads ($c = 1.2$) the improvements are far more significant (20% increase in load would result in 70 – 80% improvement) for similar confidence levels. There are two trends of interest in the results using the balancing scheduler, both of which motivate the development a tie-breaking scheduler. First, the performance of the job scheduler does not necessarily show a continuous improvement as the predictor confidence increases. This can be linked to the characteristics of the balancing algorithm. It is quite clear that, when there is a choice for the algorithm to choose between an optimal partition and a stable partition, it would opt for the stable one. For example, when the confidence level is 0.1 the scheduler would pay attention to the predictor at the cost of a large MFP. It is also clear from the results that when the prediction confidence level is very high, the algorithm would make the jobs wait longer in order to select stable partitions. This is due to the scheduling metrics (Section 3) giving more importance to minimizing the finishing time for the next job, at the expense of waiting time for subsequent jobs in the queue. However, the algorithm picks up larger MFPs with increasing confidence levels, hence its performance would not show a linear improvements for similar confidence levels. Within bounded slowdown metrics versus coefficient Figure (Figure 6) note that there is little correlation between the value of the confidence and the overall performance of the scheduler. It shows, irrespective of intermediate fluctuations, that even a small confidence prediction level can significantly improve the job scheduler performance. Most of the intermittent fluctuations are due to the presence of two different parameters (L_{MFP} and L_{PF}) within the optimized metrics, whereas the figures show the variation of system performance parameters when different levels of prediction confidence parameters are considered.

The balancing algorithm system utilization data presented in Figures 7 and 8 can be divided into two groups: low load and high load. Figures 7(b) and 8(b) illustrate high load systems corresponding to comparable low load

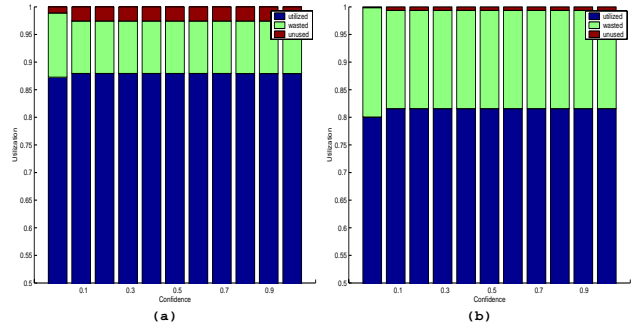


Figure 8. Utilization vs. confidence for LLNL job logs using balancing algorithm (a) $c = 1.0$, (b) $c = 1.2$.

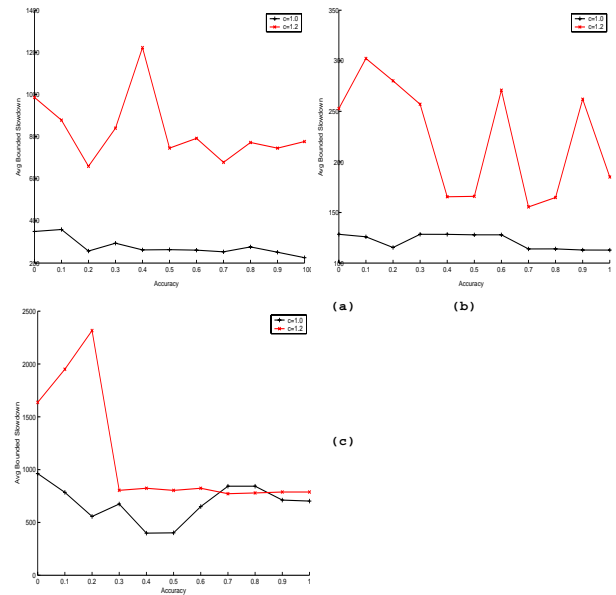


Figure 9. Average bounded slowdown vs. accuracy for (a)SDSC, (b)NASA, (c)LLNL job logs using tie-breaking algorithm.

systems through Figures 7(a) and 8(a). For systems with high load, as the confidence of the balancing scheduler increases, more and more wasted work is converted to useful work. As we introduce high load ($c = 1.2$), the benefit of prediction to scheduling gets reduced. This is because of the availability of fewer number of free partitions to choose from with increased load.

7.3. Tie-Breaking Algorithm

As discussed earlier, the tie-breaking algorithm tries to break the tie between two or more partitions of same size through a comparison of level of accuracy. The results in Figure 9 for tie-breaking algorithm are based on the same set of experiments corresponding to balancing algorithm discussed earlier. Figure 10 shows the effect of the tie-breaking algorithm on system utilization.

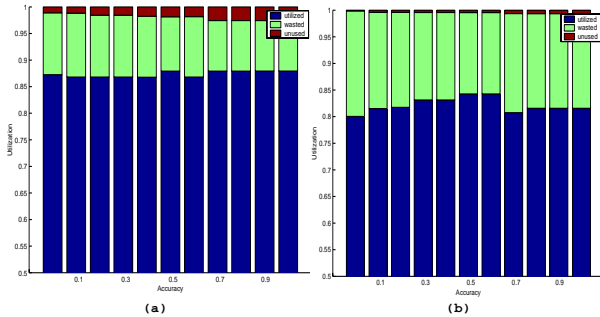


Figure 10. Utilization vs. accuracy for LLNL job logs using tie-breaking algorithm. (a) $c = 1.0$, (b) $c = 1.2$

The use of false-negatives for partition accuracy parameter calculations result the tie-breaking algorithms to represent overall worst-possible scenario in terms of performance improvement. A comparison of the results corresponding to SDSC, NASA and LLNL job logs for average bounded slowdown show moderate gain for standard load ($c = 1.0$). For example, SDSC job logs show 60 to 70% improvement of average bounded slowdown compared to 20% and 50% for NASA and LLNL job logs respectively. However, when the load parameter is increased by 20% (from $c = 1.0$ to $c = 1.2$) the NASA and LLNL job logs show initial degradation (within accuracy levels of 10 to 20%) of the bounded slowdown performance.

A comparison of the system utilization for the tie-breaking algorithm with the balancing algorithm results depict a shifting of the system utilization for higher useful work when the load is increased (from $c = 1.0$ to $c = 1.2$) similar to the balancing algorithm results. However, due to the aggressiveness of the tie-breaking algorithm, the improvements in useful work through the use of tie-breaking algorithm are not as significant as compared to the balancing algorithm improvements reported in Figures 7, and 8.

8. Conclusions and Future Work

Large scale systems like BlueGene/L are susceptible to a number of software and hardware failures, thus significantly affecting the system performance. We have proposed two new job-scheduling algorithms taking advantage of prediction of system failures leading to job failures. We have also evaluated the usefulness and the impact of these algorithms on average bounded slowdown, average response time and system utilization, considering different levels of proactive failure prediction and prevention techniques reported in literature [17, 18]. Our simulation studies show that the use of these new algorithms, with even modest fault prediction confidence or accuracy levels (as low as 10%) can significantly improve the system performance in terms of job scheduling for BlueGene/L cluster.

We are in the process of extending the fault-aware simulator to include the following:

- Consider and adapt checkpointing intervals and overheads to be based on the prediction confidence and accuracy levels.
- Consider the fault-aware job scheduling to cover other system software and programming environment parameters, including operating system and memory management parameters while making scheduling decisions.

References

- [1] N. Adiga and et. al. An overview of the bluegene/l super-computer. In *Supercomputing (SC2002) Technical Papers*, November 2002.
- [2] S. Albers and G. Schmidt. Scheduling with unexpected machine breakdowns. *Discrete Applied Mathematics*, 110(2-3):85–99, 2001.
- [3] M. F. Buckley and D. P. Siewiorek. Vax/vms event monitoring and analysis. In *FTCS-25, Computing Digest of Papers*, pages 414–423, June 1995.
- [4] M. F. Buckley and D. P. Siewiorek. Comparative analysis of event tupling schemes. In *FTCS-26, Computing Digest of Papers*, pages 294–303, June 1996.
- [5] D. Feitelson. A survey of scheduling in multiprogrammed parallel systems. *IBM Research Technical Report*, RC 19790, 1994.
- [6] D. Feitelson and M. A. Jette. Improved utilization and responsiveness with gang scheduling. In *In IPPS 97 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291, April 1997.
- [7] D. G. Feitelson. Parallel workloads archive. <http://cs.huji.ac.il/labs/parallel/workload/index.html>, 2001.
- [8] H. Franke, J. Jann, J. E. Moreira, and P. Pattnaik. An evaluation of parallel job scheduling for ascii blue-pacific. In *Proc. of SC'99. Portland OR, IBM Research Report RC 21559*, IBM TJ Watson Research Center, November 1999.
- [9] B. Gorda and R. Wolski. Time sharing massively parallel machines. In *Proc. of ICPP'95. Portland OR*, pages 214–217, August 1995.
- [10] B. Kalyanasundaram and K. R. Pruhs. Fault-tolerant scheduling. In *26th Annual ACM Symposium on Theory of Computing*, pages 115–124, 1994.
- [11] E. Krevat, J. G. Castanos, and J. E. Moreira. Job scheduling for the bluegene/l system. In *JSSPP*, pages 38–54, 2002.
- [12] I. Lee, R. K. Iyer, and D. Tang. Error/failure analysis using event logs from fault tolerant systems. In *Proceedings 21st Intl. Symposium on Fault-Tolerant Computing*, pages 10–17, June 1991.
- [13] J. S. Plank and M. G. Thomason. Processor allocation and checkpoint interval selection in cluster computing systems. *Journal of Parallel and Distributed Computing*, 61(11):1570–1590, November 2001.

- [14] X. Qin, H. Jiang, and D. R. Swanson. An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *Proceedings of the 30th. International Conference on Parallel Processing*, pages 360–368, August 2002.
- [15] K. Ranganathan and I. T. Foster. Decoupling computation and data scheduling in distributive data-intensive applications. In *Proc. HPDC*, pages 352–358, 2002.
- [16] 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'2002*, June 2002.
- [17] 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 *Proceedings of the ACM SIGKDD, Intl. Conf. on Knowledge Discovery Data Mining*, pages 426–435, August 2003.
- [18] 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*, August 2003.
- [19] M. S. Squillante. *Matrix-Analytic Methods in Stochastic Parallel-Server Scheduling Models*. Advances in Matrix-Analytic Methods for Stochastic Models, Notable Publications, 1998.
- [20] M. S. Squillante, F. Wang, and M. Papaefthymiou. *Stochastic Analysis of Gang Scheduling in Parallel and Distributed Systems*. Technical Report, IBM Research Division, 1996.
- [21] M. S. Squillante, Y. Zhang, A. Sivasubramanian, N. Gautam, J. Moreira, and H. Franke. Modeling and analysis of dynamic coscheduling in parallel and distributed environments. *Performance Evaluation Review*, 30(1):43–54, June 2002.
- [22] A. Uno, T. Aoyagi, and K. Tani. Job scheduling on the earth simulator. *NEC Jl. of Research and Development*, 44:47–52, January 2003.
- [23] R. Vilalta and S. Ma. Predictive rare events in temporal domains. In *Proceedings IEEE Conf. on Data Mining (ICDM.02)*, pages 474–481, 2002.

9. Appendix: Partition-Finder Algorithm

While placing a job in the torus, it is necessary to determine candidate partitions in which the job might be scheduled. With a job of size s , all free, contiguous, rectangular partitions containing s nodes must be determined before the scheduling algorithm can compare them to determine the optimal partition. Previous algorithms [11] have performed an exhaustive search, finding all free partitions of any size, and then selecting the subset of partitions of size s . On an empty $M \times M \times M$ torus, this naive algorithm requires $O(M^9)$ time. Based on a Projection of Partition algorithm (POP) algorithm, Krevat et al [11] improved the algorithm to $O(M^5)$.

For our simulations, we used an algorithm for finding free partitions of size s that runs with better asymptotic

running time. In particular, the running time on an empty torus is $O(M^3 \times s^3 \times (f(s))^3)$, where $f(x)$ is the size of the set $D = \{y | x \bmod y = 0, y \leq x\}$. The algorithm requires $O(M \times g(M))$ time to initialize, where the initialization can be performed as the algorithm is running on an as-needed basis, and $g(x)$ is the time needed to determine the $f(x)$ divisors $d \in D$ of x . The algorithm is given as input an $M \times M \times M$ torus, where some nodes may have been allocated for jobs, and a size $s \leq M^3$. It returns the set of all free partitions of size s . Let SHAPES = $\{\langle x, y, z \rangle \mid x, y, z \in Z \wedge xyz = s\}$, be the set of possible partition shapes. Therefore, the set of all possible partitions of size s is PARTS = $\{\langle B, S \rangle\}$ where B is a base location (i, j, k) and $\{S \in \text{SHAPES}\}$. The algorithm returns the set FREEPARTS = $\{P \in \text{PARTS} \mid \text{the partition is free}\}$. Since we check each element in PARTS to see if it belongs to FREEPARTS, this allows us to find SHAPES in $O(f(s)^3)$ time. Moreover, finding PARTS, if they belong to FREEPARTS, can take $O(M^3 \times f(s)^3)$ time.

We can further improve the running time of the algorithm by traversing PARTS in a particular order. Specifically, if we consider the base location of the torus dimensions (x, y, z) in increasing orders of x, y and z , then there is no need to search in any of the dimensions further, once we hit the value for that dimension of required length. With this algorithm, we get a significant performance improvement over the naive algorithm and POP-based partition finder algorithm considered in earlier job scheduling studies.