

# Performance Implications of Periodic Checkpointing on Large-scale Cluster 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

## Abstract

*Large-scale systems like BlueGene/L are susceptible to a number of software and hardware failures that can affect system performance. Periodic application checkpointing is a common technique for mitigating the amount of work lost due to job failures, but its effectiveness under realistic circumstances has not been studied. In this paper, we analyze the system-level performance of periodic application checkpointing using parameters similar to those projected for BlueGene/L systems. Our results reflect simulations on a toroidal interconnect architecture, using a real job log from a machine similar to BlueGene/L, and with a real failure distribution from a large-scale cluster. Our simulation studies investigate the impact of parameters such as checkpoint overhead and checkpoint interval on a number of performance metrics, including bounded slowdown, system utilization, and total work lost. The results suggest that periodic checkpointing may not be an effective way to improve the average bounded slowdown or average system utilization metrics, though it reduces the amount of work lost due to failures. We show that overzealous checkpointing with high overhead can amplify the effects of failures. The study also suggests that new metrics and checkpointing techniques may be required to effectively handle job failures on large-scale machines like BlueGene/L.*

## 1. Introduction

Large scale parallel machines, such as IBM's BlueGene/L (BG/L henceforth) are expected to play a key role in taking on the demands of long-running scientific applica-

tions. Because application running times can be on the order of weeks to months, failures that force a job to restart would be catastrophic without an effective checkpointing scheme in place. Applications in these systems typically checkpoint periodically, but there is a conspicuous lack of evidence suggesting that such periodic checkpointing is effective under real-world job loads and failure distributions. This paper fills that gap in the literature by studying the behavior of periodic application checkpointing on a three-dimensional toroidal communication architecture, using real job logs and failure traces from large-scale systems.

Most scientific applications perform their own checkpoints at times during their execution when there is minimal state, often between iterations of an outer loop. Within an application, these checkpoints tend to be regular both in terms of frequency and overhead. An effective checkpointing scheme improves application performance in the presence of failures by reducing the amount of re-computation following a fault. It is also imperative that failures do not incur severe performance penalties. Unfortunately, the probabilistic likelihood of job failure increases with the number of components on which the job is running. Indeed, it is likely that failures are correlated [17, 12], hence the probability of failure grows superlinearly with the number of nodes. Therefore, massively parallel applications are expected to fail more frequently than single-processor jobs with the same running time. Even though the design of BG/L has a strong emphasis on reliability, we expect that fatal errors (those resulting in job failure) will have a significant impact on performance.

In the literature, periodic checkpointing is largely a consequence of assumptions made regarding the distribution of failures in large-scale systems. In particular, it is generally

assumed that failures are independent and identically distributed. Studies of real systems [17, 12], however, show that failures are correlated temporally and spatially, are not identically distributed, and can be predicted with surprising accuracy. Furthermore, the behavior of checkpointing schemes under these realistic failure distributions does not follow the behavior predicted by standard checkpointing models [14].

This paper explores the effect of varying checkpoint intervals and overheads on system-wide performance metrics such as average bounded slowdown, response time, and wait time. Our simulation-based studies use an actual supercomputer job log from LLNL [7], as well as real failure data from a large AIX cluster [17, 18]. While such a study, as it is, has never been done, we also perform all experiments on a three-dimensional communication torus, which imposes additional restrictions. The simulations using toroidal interconnects reflect the architecture of BG/L, and take into account the inherent job scheduling constraints [10, 13].

While much work on checkpointing has been devoted to determining the *optimal checkpoint interval*, this study has no such motivation. Indeed, BG/L does not support system-initiated checkpointing [1]. Checkpoints are strictly initiated by the applications at appropriate points in their execution. Within the paper, when we refer to a periodic checkpointing scheme, we assume a static checkpoint interval for all applications. We investigate the effect of this checkpoint interval on system performance in the presence of failures. Similarly, checkpoint overhead is assumed to be a system-wide static characteristic.

This paper demonstrates the behavior of periodic checkpointing with a real supercomputer job log in the presence of a real failure distribution on a toroidal communication architecture. The central result is that periodic checkpointing, while the most common method of mitigating job-level loss from failures, may not be appropriate for improving system-level metrics on BG/L. Indeed, the checkpointing can be more detrimental than the failures, themselves. This suggests the need for more intelligent checkpointing schemes.

## 2. Related Work

Checkpointing, and its role in high-performance computing, is a rich field of research. There are a number of theoretical and simulation studies covering various aspects of checkpointing in the literature. It includes studies on the effects of failure distributions, different checkpointing algorithms, and application programming interfaces like MPI. We, however, confine our literature review to three primary aspects of high-performance computing: (1) failures, the role of failures, and fault tolerant aspects of large-scale parallel systems, (2) checkpointing algorithms applicable for supercomputers like BG/L or similar large-scale systems

and (3) the role of job scheduling vis-a-vis failures and checkpointing strategies for these systems.

*Faults* Algorithmic improvements or increased hardware resources are often overshadowed by reliability issues. There have been several approaches to deal with this problem, including managing system failures [4, 5, 11]. Most theoretical work focuses on providing failover mechanisms, such as hardware or software redundancy. These efforts, in practice, not only add overhead and complexity to the programming environment, but also to the application running environments.

*Checkpointing* Application-initiated checkpointing is the dominant approach for most large-scale parallel systems. Recently, Agarwal et al [2] developed application initiated checkpointing schemes for BG/L. There are also a number of studies reporting the effect of failures on checkpointing schemes including system performance. Most of these works assume Poisson failure distributions. A thorough list can be found in [14], where a study on system performance in presence of real failure distributions concludes that Poisson failure distributions are unrealistic. Similarly, a recent study by Sahoo et. al. [18, 12], analyzing the failure data from a large scale cluster environment and its impact on job scheduling, reports that failures tend to be clustered around a few sets of nodes, rather than following a particular distribution. They also report how a job scheduling process can be tuned to take into account the failures on a large-scale cluster by following certain heuristics [23].

*Job Scheduling* There are a number of research efforts analyzing job scheduling and the impact of job scheduling on system performance for large-scale parallel systems [6, 8, 9, 10, 21]. Most of these studies address either temporal or spatial job scheduling, and consider checkpointing [15], data locality, type of workload, and operating environment for fault tolerant scheduling [3, 16], to name a few. For large-scale systems like BG/L [1] and Earth Simulator [21] there are very few research efforts considering job scheduling in the presence of system failures, and none, that we know of, that also considers checkpointing. Plank's work correlates job scheduling to the failures on a cluster to study the performance [14]. However, the workloads in that study are artificial, and only a flat architecture is considered. A theoretical framework for performance analysis of checkpointing schemes is presented in [20]. In addition to considering arbitrary failure distributions, they present the concept of an *equicost* checkpointing strategy, which varies the checkpoint interval according to a balance between the checkpointing cost and the likelihood of failure. Previously from some of our earlier studies, job scheduling was recognized as playing an important role in system performance [6, 21], but was not considered to be a potential tool for dealing with failures. However, in [13], we extended

some of our own work on event prediction [18, 17, 22] to improve job scheduling performance in the presence of uncertainty. Recently, a number of statistical and machine learning-based failure prediction techniques [17, 22] have been proposed for proactive system management.

A short review of related works shows that checkpointing and associated job scheduling policies play an important role for large-scale cluster system performance in the presence of failures. To the best of our knowledge, no paper has studied checkpointing behavior using both real job logs and real failure logs. We not only fill the gap in literature, but also extend our work to alternative communication topologies. We believe that both of these contributions are unique in the literature.

### 3. Checkpointing for BlueGene/L

Our checkpointing study focuses on investigating system performance, efficiency, and throughput in the presence of failures on a toroidal topology. We first present background information on BG/L. Few details on the job scheduler is presented as well, since its impact on system performance in the presence of checkpointing is one of our major contributions of this paper. A number of other components of the study, including the simulator, the metrics, and BG/L itself, are covered in detail elsewhere [13, 1]. We review some of these here for the sake of completeness.

#### 3.1. BlueGene/L Architecture

The BG/L supercomputer system [1] is a three-dimensional torus of compute nodes. These compute nodes are also interconnected in a tree topology, which is used for reduction operations and for I/O. Communication with the external environment is accomplished through Gigabit Ethernet links attached to the I/O nodes, placed strategically in the tree interconnects.

Most systems with toroidal interconnects, including BG/L, are limited by certain constraints when scheduling jobs [10, 13]. Jobs are required to be placed in distinct, contiguous, cuboidal 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; 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 integer orthotope of nodes. Hence, the job scheduler sees BG/L as a torus of these *super-nodes*, with each supernode having from 32 to 512 compute nodes, depending on the particular deployment. In this study, we model BG/L as being composed of  $4 \times 4 \times 8$  supernodes.

#### 3.2. Checkpointing

BG/L provides system support and libraries for applications to perform checkpoint/restarts. 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 latency* ( $L$ ) is defined such that job failure between times  $t$  and  $t + L$  will force the job to restart from the previous checkpoint, rather than the current one; failure after time  $t + L$  means the checkpoint was successful and the application can restart as though continuing execution from time  $t$ . There is also a *checkpoint recovery* parameter ( $R$ ) which is the time required for a job to restart from a checkpoint. It was shown [14] that  $L$  typically has an insignificant impact on checkpointing performance for realistic failure distributions. Furthermore, 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  $R$ . Furthermore, the downtime for nodes in a supercomputer tend to be substantially less than in a typical cluster. Therefore, this study will treat  $C \approx L$  and  $R = 0$ .

While it is possible for systems to initiate checkpoints of MPI applications at mostly arbitrary times [19], this capability can have a high implementation cost and is typically accompanied by degraded checkpointing performance. Consequently, BG/L exclusively supports application-initiated checkpointing. Thus, the checkpointing interval is a feature of the applications, not of the system. In practice, these checkpoints are only approximately periodic, often corresponding to loop iterations. We describe this behavior with a static interval for simplicity. To ensure that  $C$  is as small as possible, BG/L uses techniques such as incremental checkpointing and memory hashing [2]. The projected upper bound for checkpointing any application on BG/L is 720 seconds. We therefore use a 720 second overhead in our study. In order to ascertain the sensitivity of our results to the  $C$  parameter, we also tested 60 second and 3600 second overheads, as considered elsewhere [14].

#### 3.3. Job 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 (not considered in this study). For every job  $j$ , the scheduler knows the job size in nodes ( $s_j$ ) and the estimated execution time of the job ( $e_j$ ). After a job  $j$  has been scheduled to start (begin) at time  $b_j$ , the scheduler can compute the estimated completion time of the job ( $f_j = e_j + b_j$ ). Once a job completes execution, the estimated value for  $f_j$  is replaced by

its actual value. Migration is disabled in this study, and is not currently planned for BG/L.

The scheduler operates under the following constraints, which are based on earlier job scheduling work for BG/L [10, 13] and new constraints related to failures.

- Only one job may run on a given node at a time. There is no co-scheduling or multitasking.
- A job partition must be a cuboid.
- 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.

Another important difference from [10] is the backfilling mechanism. We use a weak backfilling strategy that actually reserves a partition for the job at the front of the queue. Specifically, the job reserves the best partition such that the maximal free partition (MFP) at the current time is not blocked (if possible), which also maximizes the MFP at the reservation time. Previously, that job had only gotten a reservation time and size, with the actual partition being selected later. This new strategy aims to give the earlier job priority to the most desirable partition at the reservation time. We feel this is more in keeping with the principles of a FCFS scheduler.

Qualitatively speaking, a toroidal architecture has an impact on performance behavior because spare nodes can not be incorporated in the computation unless they are part of the partition. If a job is using its entire partition, and a single node fails, that job will need to be rescheduled in a different partition; a node outside the partition cannot simply be used as a spare.

### 3.4. Metrics

The goal of the system, particularly the job scheduler and checkpointing mechanisms, is to minimize the job wait time and system idle time, and maximize the system utilization. In this checkpointing study, we consider metrics similar to our fault-aware job scheduling studies [13] and other job scheduling studies for BG/L [10]. The actual job execution time is calculated based on start time  $b_j$  and actual finish time  $f_j$  of each job. Similarly,  $b_j$ ,  $f_j$ , and job arrival time ( $a_j$ ) can be used to calculate wait time  $w_j = b_j - a_j$ , response time  $r_j = f_j - a_j$ , and bounded slowdown  $bs_j = \frac{\max(r_j, \Gamma)}{\min(e_j, \Gamma)}$ , where  $\Gamma = 10$  seconds. Therefore, we consider the following metrics when evaluating overall system performance: (1)  $\{Average[w_j]\}$ , (2)  $\{Average[r_j]\}$  and (3)  $\{Average[bs_j]\}$ .

In our calculations, we used the so-called “last start time” of each job, which is the latest time at which the job started running in the cluster. There may be many start times, because a failed job returns to the wait queue. It would be misleading to use the first start time, because a job may fail

many times, spend time checkpointing in the cluster, and wait in the queue, all after the initial start time. However, due to our choice of start time in this study,  $w_j$  tends to be similar to  $r_j$ . In our experience, utilization is a misleading performance metric. We therefore do not focus on it in this study, but mention it in the context of maximizing system utilization and minimizing lost work, as defined in section 4.

## 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. The simulations produce measurements of the metrics covered in section 3.

### 4.1. Simulation Environment

The event-driven Java 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: checkpoint overhead, checkpoint interval). The events include: (1) *arrival events*, (2) *start events*, and (3) *finish events*, similar to other similar job scheduling simulators [10, 13]. 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.

Compared to earlier work [13], the following changes were made to the simulation environment.

- Jobs may be checkpointed, and these checkpoints have an overhead. The interval and overhead cost are parameters of the simulation.
- The downtime of a failed node is set at a constant 120 seconds, which is estimated to be the restart time of a BG/L node. While down, no jobs may be run on the node.
- The job scheduler uses no event prediction or migration. It is similar to the scheduler used by Krevat [10] with only backfilling. Any changes are noted in section 3.3.

We exclude event prediction from this study in order to focus our attention on the behavior of checkpointing in existing systems. In particular, we examine checkpointing in the presence of real job and failure logs on multiple communication architectures. This study is a crucial first step toward our future work: applying event prediction to improve

checkpointing. As mentioned before, it also fills a conspicuous gap in the checkpointing literature.

The simulation produces values for the last start time ( $b_j$ ) and finish time ( $f_j$ ) of each job, which are used to calculate wait time ( $w_j$ ), response time ( $r_j$ ), and bounded slowdown ( $bs_j$ ). We calculated system capacity *utilized* and *work lost* based on the following formulations.

If  $T = (\max_{\forall j}(f_j) - \min_{\forall j}(a_j))$  denotes the time span of the simulation, then the capacity utilized ( $\omega_{\text{util}}$ ) is the ratio of work accomplished to computational power available.

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

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

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

There is an additional component ( $\omega_{\text{unused}}$ ) which measures capacity that is unused because of a lack of job(s) requesting node(s) or other reasons unrelated to failures. However, we do not consider this parameter for our study.

Given a torus of fixed size  $N$ , failure distribution, and a set of jobs (so that  $s_j$  and  $e_j$  are fixed  $\forall j$ ), maximizing  $\omega_{\text{util}}$  is equivalent to the goal of minimizing  $T$ , while minimizing  $\omega_{\text{lost}}$  requires minimizing the expected distance between a failure and the last checkpoint, particularly for large jobs. Often  $T$  itself is used as a performance metric. While this is justifiable for schedulers that start with all jobs waiting in the queue, our simulations model jobs as arriving over time. Thus, a long-running job arriving late in the simulation can completely dominate  $T$ , rendering moot many scheduling decisions made earlier in the simulation.

## 4.2. Workload and Failure Models

We considered a job log from the parallel workload archive [7] to induce the workload on the system. The parallel job log is from Lawrence Livermore National Laboratory’s 256 node Cray T3D, collected in 1996 (referred to as LLNL job logs)<sup>1</sup>. The log contained 10,000 jobs. The average size in nodes was 10.2, the average runtime was 1024 seconds, and the maximum runtime was 41 hours. The LLNL log was used because both T3D and BGL

<sup>1</sup> As mentioned earlier, experiments were also done using logs from NASA Ames’s 128-node iPSC/860 machine collected in 1993, and from the San Diego Supercomputer Center’s 128-node IBM RS/6000 SP (1998-2000). We restrict our results in this paper to the LLNL logs, which most closely match BGL.

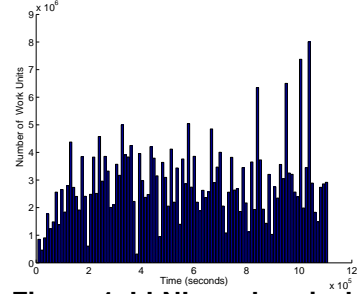


Figure 1. LLNL work arrival.

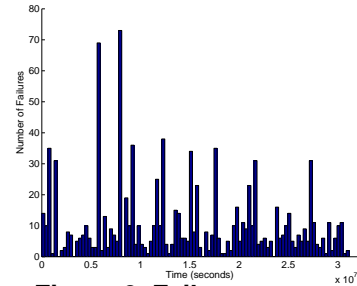


Figure 2. Failure events.

are large-scale supercomputers with a three-dimensional toroidal interconnect architecture that are primarily used by the Lawrence Livermore National Laboratory for long-running scientific applications. There is every reason to believe that both machines will see similar workloads.

Job sizes were restricted to powers of two. Strictly by number of jobs, the log was dominated by short jobs of size one; this is the case in every real log we observed. Performance of the system, however, was primarily dictated by the larger and longer jobs. The log had a large number of job arrivals near the beginning of the simulation, but these jobs tended to be small. A more instructive measure is the arrival rate of work to the cluster. Thus, the arrival at time  $a_j$  of a job  $j$  of size  $s_j$  nodes with an execution time of  $e_j$  seconds would be plotted as  $s_j e_j$  work units at time  $a_j$ . The arrival of work is shown in Figure 1. Because we are measuring work, not the number of jobs, the effects of the short jobs that arrive near the beginning of the simulation are not visible.

For failure logs, we used filtered, normalized traces collected for a year from a set of 350 AIX machines for a previous study on failure analysis [18]. A failure, in this paper, is any critical event (hardware or software) that would result in the failure of a job running on that node. We use failures from the first 128 such machines, resulting in 1,021 failures, an average of 2.8 failures per day. The MTBF on any node in the cluster was 8.5 hours, meaning a job running on all nodes would fail every 8.5 hours, on average. Therefore, the timing and distribution of failures used in this

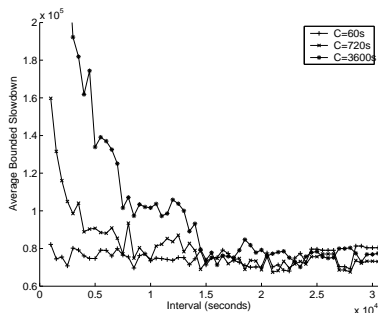


Figure 3. LLNL, bounded slowdown vs. checkpoint interval.

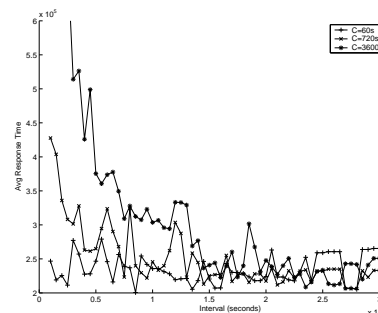


Figure 4. LLNL, response time.

study reflect the behavior of actual hardware and software in a large cluster. A single failure in the log corresponds to the failure of a supernode in BG/L, so the implicit assumption is that BG/L hardware will be 32-512 times more reliable than AIX cluster hardware. Failure logs from prototype BG/L machines suggest that this assumption is reasonable. Indeed, both the rate and distribution of the failures in the AIX cluster closely resembled the failure behavior of the BG/L prototype. The density of these critical events over time is shown in Figure 2.

## 5. Simulation Results

We present our results for the LLNL job log. Initially, we cover the general characteristics of performance for a toroidal cluster in the presence of failures, followed by an analysis of the impact of failures and of the topology constraints on these characteristics. The results presented here necessarily represent a small subset of the simulation data we gathered. In all, the simulations for all logs represent a cumulative 121,000 days of machine time, and the scheduling of 21,600,000 jobs. The graphs displayed here are exclusively for the LLNL log, and using a toroidal architecture, except where noted. Data are included either because they are representative, or because they most clearly demonstrate an interesting behavior.

Figure 3 shows bounded slowdown versus checkpoint interval in seconds. The three curves represent the three checkpoint overheads we considered. The intent was to shoot both high and low, while the 720 second overhead came from the projections for BG/L. Average response time and average wait time, plotted in Figures 4 and 5, respectively, tended to be similar to average bounded slowdown, but did not exhibit some features as prominently. As we noted earlier, response time and wait time also tended to be similar to each other. Therefore, our analysis of the results focuses on bounded slowdown. The most prominent characteristic of Figure 3 is the exponential worsening in the

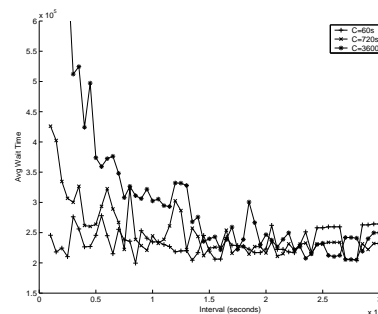


Figure 5. LLNL, wait time.

metric as the checkpoints become more frequent. A similar exponential relationship can be seen in the system utilization, plotted in Figure 6. This behavior is almost entirely a consequence of the cost of performing the checkpoints, but an additional cost is paid for failures during checkpoints (covered in detail later). In other words, overly-frequent checkpointing tends to make backfilling more difficult, because jobs require longer reservations.

In Figure 3, we anticipated a U-shaped curve, illustrating the tradeoff made by many checkpointing schemes. The lowest point on that curve would be approximately the *optimal checkpoint interval* for this metric. Moving to the left (more frequent checkpoints) should increase the bounded slowdown because the cost of performing the checkpoints outweighs the benefits. Moving to the right (less frequent checkpoints) should also increase the metric, this time because more work is being lost due to failures. While this behavior was observed in the NASA log (not shown) for  $C = 60$  seconds, we were surprised by its absence when using the LLNL log. One reason for this was that the NASA workload was much lighter than the LLNL workload, meaning that excessive checkpointing would be less likely to negatively impact the scheduling of later jobs. For the LLNL log, checkpointing frequently enough to counteract the effects of failures already meant checkpointing too frequently to permit effective scheduling of the remaining jobs.

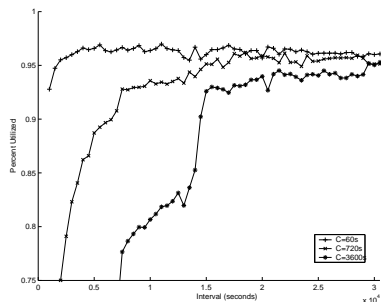


Figure 6. LLNL, average system utilization vs. checkpointing interval.

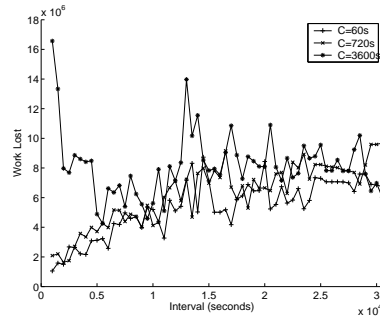


Figure 7. LLNL, work lost from failures vs. checkpointing interval.

Another major feature is the jaggedness of the curves. Such behavior is a consequence of several factors, including (1) interaction of the FCFS algorithm and the torus constraints and (2) the occurrence of failures<sup>2</sup>. Furthermore, enforcement of the FCFS policy makes the scheduler very sensitive to small changes. Small changes in effective job running times are amplified by the scheduler, particularly in conjunction with torus fragmentation and failure bursts. For example, a job at the front of the queue must be scheduled before any other jobs of that size or larger. If that job is unlucky and fails multiple times, or is oddly-shaped, or is slightly too long, there may be many jobs behind it that are delayed as a result. In some situations, these effects are even more pronounced than anything related to checkpointing.

The curves in Figures 3-5 flatten as the interval increases. If the plots were extended further to the right, each curve would converge on a value at or before the interval equal to the longest running time of any job in the log. Once the interval is longer than a particular job, that job no longer checkpoints. Thus, as the interval increases further, fewer jobs are affected by the change.

We consider the amount of work lost due to failures in Figure 7, which plots that metric vs. checkpoint interval. Again, we find that the  $C = 3600$  seconds overhead curves up on the left, suggesting that overly-frequent and expensive checkpointing can negatively impact performance. However, this is work lost due to failures, not wasted due to checkpointing. The upward tail of this curve comes from failures during checkpointing. Checkpoints increase the effective running time of the application, thereby increasing the chance of failure. In this plot, we can see that an overhead of  $C = 720$  seconds, as in BG/L, is acceptable for checkpointing that aims to minimize work lost from failures, for this log.

2 This statement was further investigated, and confirmed, by considering fat architectures and failure-free clusters, the results of which are not presented here.

## 6. Contributions and Future Work

Periodic checkpointing is one of the most common technique to counteract the effects of failures in case of large-scale computer systems. In order to deal with realistic failure distributions, periodic checkpointing may not be optimal. Presuming that the job log and failure trace are accurate predictions for the behavior of BG/L, which is reasonable, these simulations suggest that the optimal checkpointing strategy for BG/L, considering *only* bounded slowdown and utilization, is to never checkpoint at all. We certainly do not advocate that checkpointing strategy (or lack thereof) for BG/L, of course, but propose that one of several conclusions may be drawn. First, averaging bounded slowdown over all jobs may not be a relevant metric for checkpointing performance; the larger jobs are the ones for which checkpointing is most crucial, and there is no reason to believe that checkpointing will reduce bounded slowdown, because the checkpointing itself tends to increase effective job running times. Second, periodic checkpointing may not be the optimal strategy in the presence of realistic failure distributions; in real life, failures are generally *not* independent or identically distributed, do not behave like a Poisson arrival process, and are not strictly unpredictable. In other words, while BG/L must have checkpointing, realistic machine behavior demands new performance metrics and new checkpointing strategies.

If this failure distribution holds for BG/L, and if the estimated checkpoint overhead is accurate, overly-frequent checkpointing will likely be a more severe problem than failures with regard to overall system performance. Since BG/L checkpoints are always application-initiated, this is an important observation: user applications that checkpoint overzealously can degrade overall system performance without benefitting even themselves. Of course, this analysis does not consider user-level metrics such as perceived speed or fairness.

As with any simulation-based study using real logs, these

results may not apply to all loads or failure distributions. However, we feel that there is sufficient evidence that these intuitive results extend to most inputs. In particular, we believe that the results indicate that naïve periodic checkpointing on BG/L will primarily degrade overall performance. This suggests the use of more clever techniques, such as those using event prediction.

We adding new abilities to the fault-aware cluster simulator, with work toward including the following:

- Adapt checkpointing behaviors according to runtime conditions, such as the results of event prediction. Propose heuristics for improving checkpointing performance.
- Exploit regularity in the use and behavior of these systems to more intelligently utilize resources.
- Extend the fault-aware job scheduling to consider other system software and programming environment parameters, including operating system and memory management parameters.
- Make quality of service guarantees at job submission time, and schedule/checkpoint jobs so as to meet those promises.

## References

- [1] N. Adiga and et. al. An overview of the bluegene/l supercomputer. In *Supercomputing (SC2002) Technical Papers*, November 2002.
- [2] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira. Adaptive incremental checkpointing for massively parallel systems. In *ICS 2004*, pages 277–286, 2004.
- [3] S. Albers and G. Schmidt. Scheduling with unexpected machine breakdowns. *Discrete Applied Mathematics*, 110(2-3):85–99, 2001.
- [4] 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.
- [5] 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.
- [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] E. Krevat, J. G. Castanos, and J. E. Moreira. Job scheduling for the bluegene/l system. In *JSSPP*, pages 38–54, 2002.
- [11] 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.
- [12] Y. Liang, Y. Zhang, R. K. Sahoo, J. E. Moreira, and M. Gupta. Filtering failure logs for a bluegene/l prototype. In *Intl. Conf. on Dependable Systems and Networks (DSN-2005) (submitted)*, 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] 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.
- [16] 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.
- [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 and Data Mining*, pages 426–435, August 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] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. In *Proceedings, LACSI Symposium, Sante Fe, New Mexico, USA*, October 2003.
- [20] A. N. Tantawi and M. Ruschitzka. Performance analysis of checkpointing strategies. In *ACM Transactions on Computer Systems*, volume 110, pages 123–144, May 1984.
- [21] A. Uno, T. Aoyagi, and K. Tani. Job scheduling on the earth simulator. *NEC Jl. of Research and Development*, 44:47–52, January 2003.
- [22] R. Vilalta and S. Ma. Predicting rare events in temporal domains. In *Proceedings IEEE Conf. on Data Mining (ICDM.02)*, pages 474–481, 2002.
- [23] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo. Performance implications of failures in large-scale cluster scheduling. In *10th Workshop on JSSPP, SIGMETRICS*, 2004.