Evaluating Cooperative Checkpointing for Supercomputing Systems

Adam Oliner[1,2] and Ramendra Sahoo[3]

[1]Stanford University
Department of Computer Science
Palo Alto, CA 94305-9025 USA
oliner@cs.stanford.edu

[3]IBM
T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532 USA
rsahoo@us.ibm.com

Abstract

Cooperative checkpointing, in which the system dynamically skips checkpoints requested by applications at runtime, can exploit system-level information to improve performance and reliability in the face of failures. We evaluate the applicability of cooperative checkpointing to large-scale systems through simulation studies considering real workloads, failure logs, and different network topologies. We consider two cooperative checkpointing algorithms: work-based *cooperative checkpointing uses a heuristic based on the amount of unsaved work and* risk-based *cooperative checkpointing leverages failure event prediction. Our results demonstrate that, compared to periodic checkpointing, risk-based checkpointing with event prediction accuracy as low as 10% is able to significantly improve system utilization and reduce average bounded slowdown by a factor of* 9*, without losing any additional work to failures. Similarly, work-based checkpointing conferred tremendous performance benefits in the face of large checkpoint overheads.*

1 Introduction

Providing reliability and performance in the presence of failures is a central problem in supercomputing. Large-scale systems, like IBM's Blue Gene/L (BG/L), are simultaneously capable of more intensive parallel computation and susceptible to a greater number of failures [9]. Because the application space is dominated by long-running scientific applications, with runtimes of the order of weeks or months, checkpointing schemes are crucial for providing reliability. Many supercomputing systems, including BG/L, do not support system-initiated checkpointing. Thus, while most supercomputing systems do not have the ability to initiate a checkpoint, almost all can be easily modified to *ignore* an application's call to perform a checkpoint.

Cooperative checkpointing allows the system to dynamically skip checkpoints requested by applications [10]. Checkpoint requests are placed liberally throughout the code, wherever a checkpoint may be efficient. At runtime, each request is either granted or denied. This scheme is based on the philosophy that applications know best *when* to checkpoint, while systems know best *when not* to checkpoint. This paper considers cooperative checkpointing algorithms that use system-wide information to skip requested checkpoints that appear either inefficient or unlikely to be used for rollback. Specifically, we present *work-based* cooperative checkpointing, which considers workload, and *risk-based* cooperative checkpointing, which uses failure event prediction.

We evaluate the applicability of cooperative checkpointing to large-scale supercomputing systems with predominantly long-running applications. Hence, runtime techniques used in this paper assume that the system knows information such as the checkpoint overhead cost and the checkpoint interval. A trace-based simulation study is carried out for different supercomputer topologies (general cluster and 3-D torus) while using failure and job logs used in earlier studies [12, 13].

This paper answers the following questions, among others. What is the impact of cooperative checkpointing on common metrics like bounded slowdown and utilization? How does it fare relative to the standard reliability technique, periodic checkpointing? What role does the network topology play in machine reliability? How accurate must event prediction be to yield a benefit, and what benefit might be expected?

[2]Work was performed at IBM T.J. Watson Research Center as part of the VI-A Internship Program, an MIT cooperative education program with Industry, together with a VI-A Fellowship funded by IBM.

2 Related Work

Checkpointing, including checkpointing algorithms for supercomputing systems, is a rich field of research. Recently, there have been a number of studies on checkpointing based on certain failure characteristics [15], including Poisson distributions. Plank and Elwasif [14] carried out a study on system performance in the presence of real failure distributions and concluded that it is unlikely that failures in a computer system would follow a Poisson distribution. The workloads considered by Plank and Elwasif were artificial. Tantawi and Ruschitzka [18] developed a theoretical framework for performance analysis of checkpointing schemes. In addition to considering arbitrary failure distributions, they present the concept of an *equicost* checkpointing stategy, which varies the checkpoint inverval according to a balance between the checkpointing cost and the likelihood of failure.

It has been well recognized that job scheduling plays an important role in system performance [3]. There are a number of research efforts analyzing job scheduling and its impact on system performance [2, 5, 6, 7]. For large-scale supercomputers like Blue Gene/L [1] and Earth Simulator, there are very few research efforts considering failure distributions, critical event predictions, and job scheduling aspects all together for different communication topologies.

Absence of any research in the area of fault-aware checkpointing involving real job logs, communication topologies, scheduling policies, and real failures is the main motivation for this work. Furthermore, this paper is intended to experimentally validate the general claims of previous work on cooperative checkpointing theory [10]. Other experimental work evaluated the theoretical results more explicitly [11], but did not simulate a system with the complexity and robustness as in this paper.

3 Problem Description

This section describes the problem and explains the cooperative checkpointing behavior. Details of the cooperative checkpointing theory are covered elsewhere [10]. A few aspects of our experiments, including the simulator, the metrics, and the architectures, share similarities with [13]. Some are also covered here for the sake of completeness.

3.1 Terms and Definitions

Most high-performance systems provide OS support and libraries for applications to perform checkpoint/restarts. We use the following model of behavior in this paper. 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) was shown in [14] to typically have an insignificant impact on checkpointing performance. There is also a *checkpoint recovery* parameter (R) which is the time required for a job to restart from a checkpoint; this, like the downtime, is unavoidable. Therefore, we let $C \approx L$ and $R = 0$.

Most scientific applications tend to be run many times, and the machines are largely homogeneous. As a result, there is a great deal of predictability. Estimation techniques for parameters like C and I are beyond the scope of this paper, but we assume the system can determine them. Despite such efforts to reduce checkpoint overhead, these costs continue to increase with the size of the machine, and are heavily dependent on system conditions like network traffic and disk activity. For BG/L, the target upper bound for checkpointing any application is $C = 12$ minutes (used in Section 4).

3.2 Cooperative Checkpointing Strategies

Let s_{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 s_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 s_{i+1} be the time at which the next checkpoint will be started. We define I to be the checkpoint interval such that $I = f_{i-1} - s_i$ $\forall i \geq 1$, unless checkpoint $i-1$ is skipped, in which case the interval is $dI = f_{i-d} - s_i$, 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 s_i. Note that $C_i = f_i - s_i$, 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 $n \cdot k$.

3.2.1 Work-Based Cooperative Checkpointing

For work-based checkpointing, at time s_i when job j requests a checkpoint, we pose the following question: *should j perform the checkpoint or skip it?* Because checkpoints can be initiated only by the application, this choice is the only opportunity for the system to exert its influence over the checkpointing strategy. The question the system must answer is expressed by Equation 2, which compares the cost of performing the checkpoint with the risk associated with skipping it. If the inequality holds, the checkpoint should be performed. In this way, the system cooperates with the application to select a mutually agreeable checkpointing strategy.

The inequality is calculated by considering the worst-case failure scenario, in which a failure would occur just before the successful completion of checkpoint $i+1$. We consider the amount of time by which the successful completion of checkpoint $i+1$ would be delayed from the scenario in which no checkpoint is performed, and no failure occurs. That is, we measure the delay beyond $I + C_{i+1}$. For example, if we skip checkpoint i, then after paying $I + C_{i+1}$ and failing, we would roll back to f_{i-d} (where $i-d$ is the last checkpoint that was performed), pay dI to return to where we were, and then pay another $I + C_{i+1}$ to finish the checkpoint. On the other hand, performing the checkpoint would mean that we only roll back to f_i. Performing the checkpoint already delayed progress by C_i, but to finish checkpoint $i+1$, it is only necessary to pay another $I + C_{i+1}$. So we perform the checkpoint if

$$\begin{aligned} n_j \cdot ((d+1)I + C_{i+1}) &\geq n_j \cdot (I + C_{i+1} + C_i) & (1) \\ dI &\geq C_i & (2) \end{aligned}$$

We call the checkpointing heuristic represented by Equation 2 *work-based*, because it compares the amount of unsaved work to the opportunity cost, in units of work, of performing the checkpoint. This claim is intuitive; it does not make sense to perform a checkpoint that takes longer than the work you are trying to save. In such a situation it would be cheaper to recompute than to checkpoint. Note that dI in Equation 2 represents the time since the last completed checkpoint, *not* merely the static checkpoint interval. Consequently, it is possible to have checkpoints which can be skipped with some regularity.

3.2.2 Risk-Based Cooperative Checkpointing

Work-based cooperative checkpointing considers the worst-case loss due to failures. A more realistic measure is expectation: how much work do we expect to lose before checkpoint $i+1$ is completed? If that measure is greater than the cost of checkpointing, then we perform the checkpoint. This strategy is called *risk-based* cooperative checkpointing. Let p_f be the probability that the partition on which job j is running will fail before f_{i+1}. Using the same measure as above, 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 $C_{i+1} \approx C_i$, this reduces to the heuristic for risk-based checkpointing, which is expressed in Equation 4.

$$\begin{aligned} p_f((d+1)I + C_i) &\geq p_f(I + 2C_i) + (1-p_f)C_i & (3) \\ p_f dI &\geq C_i & (4) \end{aligned}$$

Thus, work-based is the special case of risk-based where $p_f = 1$. That is, even if failure is a certainty, we should not spend more time checkpointing job progress than would be required to simply recompute. Additionally, work-based bounds how badly risk-based cooperative checkpointing will do in the presence of false positives in the event prediction; a 100% false positive rate will do at most as many checkpoints as work-based cooperative checkpointing.

3.3 Event Prediction

In order to use risk-based cooperative checkpointing to make runtime decisions about whether to skip or grant a given checkpoint, the system must have a mechanism for estimating p_f, the probability that a failure occurs before f_{i+1}. Such event prediction systems [16] have been demonstrated to be capable of accuracies near 70%. Even without statistical techniques, event prediction can be achieved with great accuracy. For example, scheduled maintenance events can be communicated to the system, allowing it to estimate $p_f = 1$ with absolute confidence for the duration of that time.

This raises the question of how accurate event prediction needs to be in order to be useful. Because our model of the problem assumes that all checkpoints will be performed unless the strategy explicitly skips it, we have a simple performance baseline. Work-based and risk-based strategies only lose in one situation: a checkpoint is skipped and a failure occurs. Therefore, the performance of these strategies will only be worse if there is a high rate of false negatives.

3.4 Topology and Job Scheduling

The simulations (Section 4) cover the two most common communication topologies: flat (all-to-all) and toroidal. Examples of flat architectures include large Linux/AIX clusters like ASCI Purple. Cray T3D and Blue Gene/L [1] are examples of toroids. Most systems with toroidal interconnects, including BG/L, are limited by certain constraints when scheduling jobs [8, 2, 3]. Jobs are required to be placed in distinct, contiguous, cuboidal partitions. In order to satisfy these requirements, a job partition must be composed as a three-dimensional integer orthotope of compute node blocks. Hence, the job scheduler sees the cluster as a torus of these blocks or *supernodes* (nodes).

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) and the estimated execution time of the job (e_j), and the 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.

The scheduler operates under the constraints for toroidal architectures based on earlier job scheduling work for BG/L [8] with new constraints related to failures. Specifically, there is no co-scheduling or multitasking, a job partition must be a cuboid, and 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. There is no cuboid requirement for flat interconnect architectures.

3.5 Metrics

We consider metrics similar to those in Krevat's scheduler [8]. The actual job execution time is calculated based on start time s_j and actual finish time f_j of each job; when measuring utilization, we use execution time excluding checkpoints. Similarly, s_j, f_j, and job arrival time (a_j) can be used to calculate wait time $w_j = s_j - a_j$, 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 order to be consistent, we treat checkpointing overhead as being wasted work. That is, e_j is the execution time of the job *without* checkpoints. Therefore, values for bounded slowdown, for example, may seem unusually high. In fact, we believe that this is a more accurate representation of the performance of the cluster; if the checkpoints could be skipped, the baseline optimal may be improved.

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 [17].

4.1 Simulation Environment

The event-driven simulator models a 128 (super)node cluster with either flat or three dimensional ($4 \times 4 \times 8$) torus topology. The simulator is provided with a job log, a failure log, and other parameters (e.g.: C and I). The events include: (1) *arrival events*, (2) *start events*, and (3) *finish events*. 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 downtime of a failed node is set at a constant 120 seconds, which is estimated to be a modest restart time for nodes in any large-scale computer system.

The simulation produces values for the last start time (s_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 (ω_{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 jx be the job that fails as a result of x, which may be *null*. If c_{jx} is the time at which the last successful checkpoint for jx started, then the amount of work lost as a result of failure x is $(t_x - c_{jx})n_{jx}$ (this equals 0 for $jx = null$). Hence, the total work lost (ω_{lost}) is $\omega_{\text{lost}} = \sum_{\forall x}(t_x - c_{jx})n_{jx}$. There is an additional component (ω_{unused}), but we do not consider it in this paper.

4.2 Workload and Failure Models

We considered job logs from the Parallel Workload Archive [4] to induce load on the system. These include a log from NASA Ames's 128-node iPSC/860 machine collected in 1993 (NASA log, henceforth), San Diego Supercomputer Center's 128-node IBM RS/6000 SP (1998-2000) job log (SDSC log), and Lawrence Livermore National Laboratory's 256 node Cray T3D (1996) job log (LLNL log). Each log contained 10000 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 350 AIX machines for a previous

Job Log	Avg Size	Avg RT (s)	Max RT (hr)
NASA	6.3	381	12
SDSC	9.7	7722	132
LLNL	10.2	1024	41

Table 1. Job log characteristics. RT stands for the runtime, or execution time.

study on failure analysis and event prediction [17, 16]. By failure we mean any temprary or permanent failure leading to a failure of a job. 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. Therefore, the timing and distribution of failures used in this study reflect the behavior of actual hardware and software in a large cluster.

5 Simulation Results

The simulations, in all, represent more than 600,000 days of cluster time, and involve the scheduling of more than 30 million jobs. The results presented here are necessarily a subset of these simulations. Included results are representative.

5.1 System Performance

Figure 1 plots checkpointing interval against average bounded slowdown for the SDSC log, on a flat cluster, with a checkpoint overhead of 12 minutes (720 seconds). The same runs for the NASA log and runs for the SDSC log on a toroidal interconnect architecture are not shown, but exhibited the same properties as Figure 1; the NASA bounded slowdowns were decreased by a constant factor and the toroidal values were increased by a constant factor. The five curves represent periodic, work-based, and risk-based cooperative checkpointing for three accuracy levels respectively. Periodic checkpointing means every checkpoint is performed at intervals defined by the x-axis value. Similarly, work-based cooperative checkpointing is performed according to the definition in Equation 2.

Risk "a" indicates risk-based checkpointing with a false negative rate of $1-a$. Thus, $a = 0$ implies that the predictor will always return $p_f = 0$ (no checkpointing is performed). Despite the fact that no checkpoints are performed, the metric for Risk 0 varies with the checkpoint interval because the job scheduler must estimate the completion time of the job. The scheduler estimates the total running time (R) as if all checkpoints

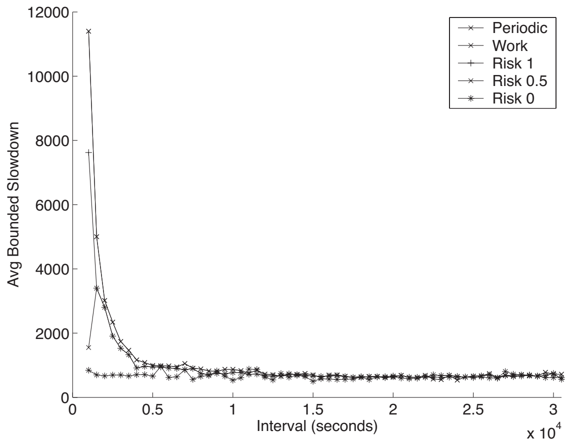


Figure 1. Bounded slowdown vs. checkpoint interval in seconds for the SDSC job log, with a checkpoint overhead of 720 seconds.

are to be performed based on $R = r + C \cdot \lfloor (r/I) \rfloor$. Therefore, a smaller I makes performance enhancing techniques, like backfilling, less likely.

For a checkpoint overhead of 720 seconds ($I < C$) work-based cooperative checkpointing results in the same curve as periodic checkpointing. As the checkpointing interval is decreased, bounded slowdown for the periodic checkpointing scheme increases exponentially. In general, risk-based cooperative checkpointing, at any accuracy, results in a lower bounded slowdown compared to either work-based or periodic checkpointing. This is because the bounded slowdown is dominated by the checkpointing overhead. Risk-based will never perform more checkpoints than work-based, and work-based will never perform more checkpoints than periodic checkpointing.

As a representative case of checkpointing results for higher overheads (say $C = 3600$ seconds), Figure 2 plots bounded slowdown for the SDSC log on a flat cluster. Between the intervals of $I = 3500$ seconds and $I = 4000$ seconds, work-based cooperative checkpointing diverges suddenly and dramatically from periodic checkpointing. The checkpoint overhead is 3600 seconds, so $I > 3600$ seconds means that every checkpoint will be performed. Below 3600 seconds, the work-based heuristic takes effect. At $I = 3500$ seconds, for example, every other checkpoint is performed, starting with the second one. This immediately results in a 7-fold decrease in average bounded slowdown. Again from Figure 2, there is nearly a 50% gap in performance between the *Work* and *Risk 1* maximum values. A similar gap can be seen in all Figures for $C = 720$ seconds. Work-based will perform every checkpoint such that $dI > C$, whether or not the event predictor indicates

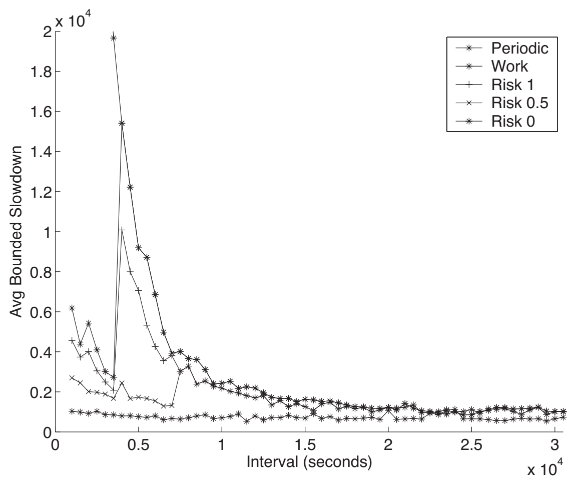


Figure 2. Bounded slowdown vs. checkpoint interval in seconds for the SDSC job log, using a checkpoint overhead of 3600 seconds.

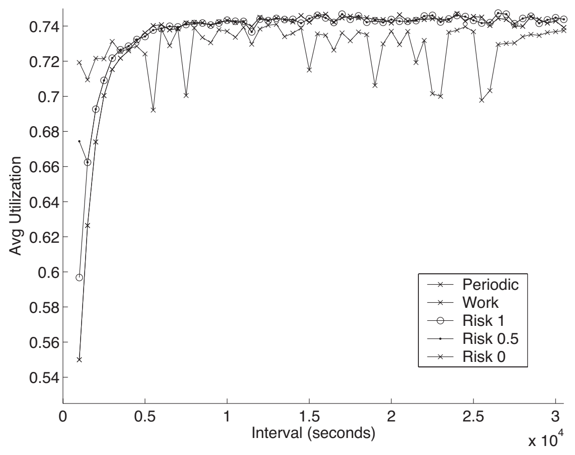


Figure 3. System utilization on the torus for the SDSC log. This is representative of our results with other inputs. (C = 720 seconds)

that a failure is likely. On the other hand, risk-based, with no false positives, will only perform a checkpoint when a failure is predicted to occur before the end of the subsequent checkpoint. Consequently, work-based performs more checkpoints than risk-based. If, hypothetically, the false positive rate was set at 1 (always predicts a failure) and the predictor, therefore, always returned $p_f = 1$, then the Risk 1 and Work curves would be identical. We therefore call the gap between these curves the *false positive gap*.

Because of our choice of job start time, response time and wait time tend to be similar to each other. In general, bounded slowdown, response time, and wait time curves for the same input parameters are similar in nature. Bounded slowdown curves strongly exhibit important characteristics, however, so we chose to focus on that metric.

5.2 System Utilization

Results for the SDSC log on a torus ($C = 720$ seconds) are presented in Figure 3. We see that naïve checkpointing can reduce effective utilization from $\sim 74\%$ to $\sim 55\%$; when $C = 3600$ seconds, utilization dropped from $\sim 67\%$ to $\sim 20\%$. Simple work-based checkpointing increases utilization by more than 25%. In general, for these parameters, not checkpointing decreases effective utilization. For smaller intervals, however, Risk 0 is best for this metric.

We conclude that checkpointing does not generally act to improve system-level metrics like utilization and bounded slowdown, for the workloads and failure distributions we observed. Checkpointing increases the effective running time of jobs, and makes efficient scheduling more difficult. Work-based and risk-based cooperative checkpointing mitigate this loss of efficiency by skipping checkpoints that the heuristics perceive as being superfluous. In Section 5.4, we examine the tradeoff made for those improvements in performance.

5.3 Work Lost

Checkpointing is intended to be a selfish act: a job checkpoints in order to minimize the amount of recomputation it will need to perform after a failure, oblivious to outside constraints. Minimizing the work lost parameter inherently satisfies the goal of checkpointing, and is the basis for the requirements of work-based and risk-based cooperative checkpointing.

Figure 4 shows the total amount of work lost due to failures for the SDSC log on a flat cluster. The most outstanding feature is the curve for Risk 0 (no checkpointing), which is distinctly separate from the other curves. Compared to no checkpointing, the amount of work lost from failures is reduced by more than 79% when the accuracy of the predictor is raised to 10%, and by 92% at 40% accuracy. In other words, predicting and checkpointing ahead of only 10% of all failures makes a huge impact in the amount of lost work. Predicting around half of the failures has the same effect on lost work as checkpointing periodically, whether or not a failure is expected.

For curves other than Risk 0, where checkpointing is being performed, a higher interval tends to increase the amount of lost work. This is reasonable, because more freqent checkpointing is a common strategy to minimize lost work. The fluctuations in Risk 0 are a

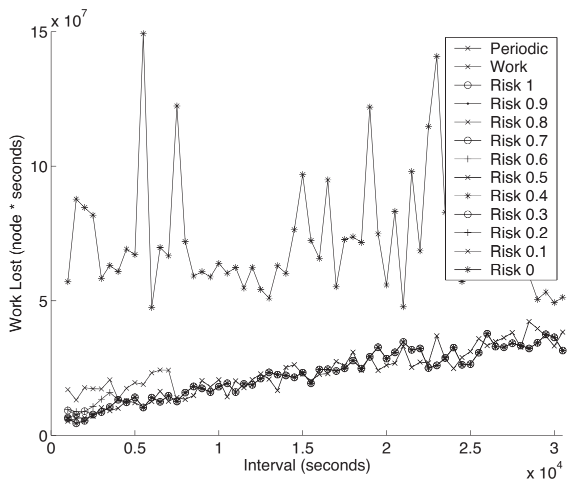


Figure 4. Lost work, SDSC, torus, all accuracies. At 10% prediction accuracy, lost work is reduced nearly as much as with perfect prediction.

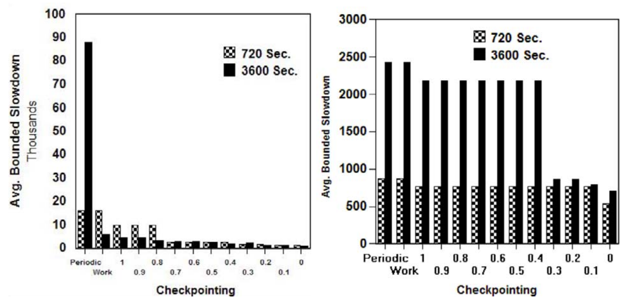
consequence of the way in which the jobs happen to be scheduled, and illustrates the variance in the amount of work that may be lost without checkpointing.

5.4 Strategy Comparison

This section presents a different view of the results from the previous section, and summarizes the trade-offs offered by our new checkpointing heuristics. Figures 5 and 6 show results for the SDSC log on a flat cluster. The x-axis indicates the type of checkpointing that was used. All plots are for $C = 720$, 3600 seconds and $I = 1000, 10000$ seconds.

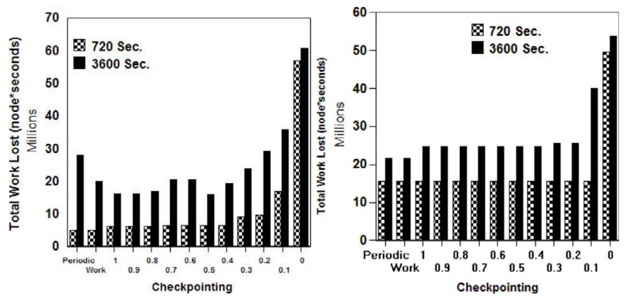
Consider first the results for $I = 10000$ seconds. The bounded slowdowns in Figure 5 show a gradual decrease in this metric as fewer checkpoints are performed, for both overheads. While Risk 0 is best, note that Risk 0.1 gives nearly the same values. Utilization (not shown) showed a similar pattern, with utilization tending to increase as fewer checkpoints are performed.

For $I = 1000$ seconds, periodic checkpointing performs significantly worse than either of our heuristics in both bounded slowdown and utilization. In that case, with $C = 3600$ seconds, work-based cooperative checkpointing gave an immediate 25% utilization boost, with an additional 20% being possible if all checkpoints are skipped. Work-based cooperative checkpointing reduced bounded slowdown, in this extreme case, by more than a factor of 90. By themselves, these measurements of bounded slowdown and utilization give the impression that checkpointing should be abandoned entirely.



(a) $I = 1000$ sec (b) $I = 10000$ sec

Figure 5. Bounded slowdown, SDSC log, C = 720 and 3600 seconds. Suggests checkpointing as infrequently as possible.



(a) $I = 1000$ sec (b) $I = 10000$ sec

Figure 6. Total work lost, SDSC log, C = 720 and 3600 seconds. Unlike Figure 5, suggests checkpointing as frequently as possible.

Figure 6, however, indicates just the opposite. For $I = 10000$ seconds, the amount of lost work increases as the amount of checkpointing decreases. For a system with $C = 720$ seconds, a prediction accuracy of 10% reduces the amount of lost work as much as periodic checkpointing, while also bringing bounded slowdown and utilization to near optimal values. Recall that event prediction with accuracy as high as 70% has already been achieved [16]. We conclude that an application should spend as little time checkpointing as possible, but no less, and that those important checkpoints can be effectively identified with event prediction. The results for $I = 1000$ seconds were slightly more complicated, but the conclusion is similar: by intelligently *skipping* checkpoints according to the work-based heuristic, the amount of lost work can be *decreased.* Using event prediction with a mere 10% accuracy, the amount of lost work can drastically reduced, while simultaneously increasing bounded slowdown and utilization to near-optimal levels.

6 Conclusions and Contributions

Cooperative checkpointing, in which the system can dynamically skip checkpoints requested by applications at runtime, is an effective technique for providing reliability on large-scale systems without sacrificing performance. Application intitiated checkpointing is central to providing reliability, but naïve checkpointing can be just as deleterious to performance as the failures [12]. In this paper, we evaluated cooperative checkpointing strategies [10] for two new checkpointing heuristics: risk-based (event prediction) and work-based (workload characteristics).

This paper makes the following contributions:

- Presents the results of trace-based simulations using real workloads and failure logs, suggesting that the performance impact of communication topology is secondary to the scheduling policies and failure charateristics.

- Shows that risk-based and work-based cooperative checkpointing can dramatically improve utilization and other system-level metrics, while simultaneously reducing the amount of work lost due to failures.

- Demonstrates that forecasting or predicting failures with an accuracy as low as 10% can provide tremendous performance benefits. Even with a false-negative rate of 90%, risk-based cooperative checkpointing cut the amount of work lost to failures many-fold, while providing exceptional bounded slowdown and utilization values.

References

[1] N. Adiga and et. al. An overview of the bluegene/l supercomputer. In *Supercomputing (SC2002) Technical Papers*, November 2002.

[2] D. Feitelson. A survey of scheduling in multiprogrammed parallel systems. *IBM Research Technical Report*, RC 19790, 1994.

[3] 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.

[4] D. G. Feitelson. Parallel workloads archive. *http://cs.huji.ac.il/labs/parallel/workload/index.html*, 2001.

[5] H. Franke, J. Jann, J. E. Moreira, and P. Pattnaik. An evaluation of parallel job scheduling for asci bluepacific. In *Proc. of SC'99. Portland OR, IBM Research Report RC 21559 , IBM TJ Watson Research Center*, November 1999.

[6] B. Gorda and R. Wolski. Time sharing massively parallel machines. In *Proc. of ICPP'95. Portland OR*, pages 214–217, August 1995.

[7] B. Kalyanasundaram and K. R. Pruhs. Fault-tolerant scheduling. In *26th Annual ACM Symposium on Theory of Computing*, pages 115–124, 1994.

[8] E. Krevat, J. G. Castanos, and J. E. Moreira. Job scheduling for the bluegene/l system. In *JSSPP*, pages 38–54, 2002.

[9] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. K. Sahoo. Blue gene/l failure analysis and prediction models. In *Submitted to Intl. Conf. on Dependable Systems and Networks (DSN)*, June 2006.

[10] A. J. Oliner, L. Rudolph, and R. K. Sahoo. Cooperative checkpointing theory. In *IEEE IPDPS, Intl. Parallel and Distributed Processing Symposium*, Apr. 2006.

[11] A. J. Oliner, L. Rudolph, and R. K. Sahoo. Robustness of cooperative checkpointing. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN), submitted*, June 2006.

[12] A. J. Oliner, R. K. Sahoo, J. E. Moreira, and M. Gupta. Performance implications of periodic checkpointing on bluegene/l systems. In *Submitted to IEEE IPDPS, Intl. Parallel and Distributed Processing Symposium*, 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] 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.

[17] R. K. Sahoo, A. Sivasubramanian, 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.

[18] A. N. Tantawi and M. Ruschitzka. Performance analysis of checkpointing strategies. In *ACM Transactions on Computer Systems*, volume 110, pages 123–144, May 1984.