2014 43rd International Conference on Parallel Processing

Characterizing the Impact of Rollback Avoidance at Extreme-Scale: A Modeling Approach

Scott Levy
Dept. of Computer Science
University of New Mexico
Albuquerque, New Mexico
slevy@cs.unm.edu

Kurt B. Ferreira
Sandia National Laboratories
Albuquerque, New Mexico
kbferre@sandia.gov

Patrick G. Bridges
Dept. of Computer Science
University of New Mexico
Albuquerque, New Mexico
bridges@cs.unm.edu

***Abstract*—Resilience to failure is a key concern for next-generation high-performance computing systems. The dominant fault tolerance mechanism, coordinated checkpoint/restart, is projected to no longer be a viable option on these systems due to its predicted overheads. Rollback avoidance has the potential to prolong the viability of coordinated checkpoint/restart by allowing an application to make meaningful forward progress, perhaps with degraded performance, despite the occurrence or imminence of a failure. In this paper, we present two general analytic models for the performance of rollback avoidance techniques and validate these models against the performance of existing rollback avoidance techniques. We then use these models to evaluate the applicability of rollback avoidance for next-generation exascale systems. This includes analysis of exascale system design questions such as: (1) how effective must an application-specific rollback avoidance technique be to usefully augment checkpointing in an exascale system? (2) when is rollback avoidance on its own a viable alternative to coordinated checkpointing? and (3) how do rollback avoidance techniques and system characteristics interact to influence application performance?**

I. Introduction

Resilience is a key obstacle to next-generation extreme-scale systems. If current predictions hold, the increased scale of these systems could mean that they experience multiple failures per hour [1]. Effective and efficient mechanisms for recovering from or avoiding failures will be necessary for applications to make meaningful forward progress. Although coordinated checkpointing remains effective on current systems, the scale of next-generation systems will result in more and more time being taken away from the application for fault tolerance activities (e.g., writing checkpoints, recovering from failures). At the scales projected for the first exascale system, less than half of the system's time will be available for advancing the application's computation [2], [3].

Many approaches have been proposed to reduce the performance impact of failures on checkpoint/restart systems. These include failure prediction and preventive migration [4], [5], replication-based approaches [2], [6], [7], fault-tolerant algorithms [8]–[12], and software-based memory error correction [13]–[15]. The common principle underlying these approaches is that enabling an application to continue executing (perhaps with some degradation) despite the occurrence or imminence of a failure will improve its performance.

These techniques typically rely on reducing checkpointing costs by significantly reducing the frequency with which the application is forced to roll back to a previous checkpoint. Analysis of these techniques is difficult, however, for two reasons: (i) checkpoint/restart costs vary non-linearly with system mean time to interrupt (MTTI) [16]; and (ii) these techniques frequently can mitigate only a subset of system failures (e.g., memory corruption) [9]. As a result, the suitability of these techniques in exascale systems is not always clear.

To address this problem, this paper presents a general model for analyzing the performance impact of techniques for avoiding failure. Its contributions include:

- A general conceptual model that captures the key features of rollback avoidance techniques;
- Two analytic models of the impact of failure avoidance on application performance, one when failure avoidance is used in conjunction with coordinated checkpointing and one when failure avoidance is used as replacement for coordinated checkpointing;
- Case studies mapping the performance of both replication and fault prediction techniques to this model;
- An analysis of when rollback avoidance techniques are viable either on their own or in concert with coordinated checkpoint/restart in next-generation HPC systems.

The remainder of this paper is organized as follows: the next section provides background on these failure avoidance techniques. Section III introduces our analytical models, while Section IV provides validation of these models against a previously validated simulator. Sections V and VI describe two case studies using our models: process replication and failure prediction, respectively. Section VII uses these models to analyze the effectiveness of failure avoidance both with and without checkpointing and how the performance of these techniques drive the requirements of future systems. Section VIII summarizes related work, and Section IX concludes.

II. Background

The most widely-studied approaches for handling faults in large-scale systems can be grouped into three broad categories: *failure avoidance*, *failure effect avoidance* and *failure effect repair* [17]. Failure avoidance techniques use prediction to forecast when a failure is likely to occur and, based on this prediction, take action to minimize the impact of the failure on

0190-3918/14 $31.00 © 2014 IEEE
DOI 10.1109/ICPP.2014.49

the application. For example, sophisticated analysis of system logs may provide enough advance notice to allow the processes threatened by an imminent failure to be migrated to safer hardware resources [4]. Failure effect avoidance techniques allow the application to continue to execute despite the occurrence of failures. This category includes fault tolerance algorithms, replication, and software-based memory error correction. For example, two matrices that are being multiplied together can be augmented with redundant data that allows the contents of the matrices to be reconstructed if a failure occurs [18]. Finally, failure effect repair techniques restore normal execution after the application has been compromised by a failure. The most widely-studied method in this category is checkpoint/restart and its variants.

In this paper, we focus on the first two categories: failure avoidance and failure effect avoidance. The techniques in these two categories allow the system to handle failures such that the application need not roll back to a previous checkpoint. For ease of exposition, we refer to these two categories collectively as *rollback avoidance*.

III. An Analytical Model of Rollback Avoidance

Several models exist for specific rollback avoidance techniques (e.g. [2], [4]), but not for rollback avoidance in general. A general analytical model of rollback avoidance is important for understanding the strengths, limitations, and tradeoffs involved with these techniques. Because these techniques are frequently used in concert with other techniques, for example checkpoint/restart, understanding their general behavior tradeoffs can provide important guidance on the design of next-generation systems. It can also guide research on current and future rollback avoidance techniques, providing information about the best way to address resilience challenges.

A. Developing a Model of Rollback Avoidance

We begin by a developing a general conceptual model of rollback avoidance. We model rollback avoidance stochastically in terms of two characteristics: (i) the ability to avoid rollback; and (ii) the overhead costs. When a failure occurs, a rollback avoidance technique prevents the application from experiencing the failure with probability p_a, the *rollback avoidance probability*. We assume that failures are exponentially distributed. We further assume that the failures that result in rollback despite the best efforts of these techniques are also exponentially distributed. Effectively, this probability describes an increase in the system Mean Time To Interrupt (MTTI) because it represents a probabilistic decrease in the number of failures that occur in the system. The resulting expression of system MTTI (M') is shown in Equation 1.

$$M' = \frac{\Theta}{1 - p_a} \tag{1}$$

where

$$\begin{aligned} \Theta &= \text{ native system MTTI} \\ p_a &= \text{ rollback avoidance probability} \end{aligned}$$

We model overhead as an extension of the application's solve time. *Rollback avoidance overhead* (o_a) describes the overhead cost as a fractional increase in solve time due to a rollback avoidance technique. This fraction represents the expected increase in solve time due to the overhead imposed by rollback avoidance, including preparation for future failures, migration to avoid imminent failures, and application degradation (e.g., slower rate of convergence) due to partial correction. However, it does not include time spent recovering from a failure that could not be avoided. We assume that the application is otherwise unperturbed. The resulting expression of the application's solve time (T_s') is shown in Equation 2.

$$T_s' = T_s(1 + o_a) \tag{2}$$

where

$$\begin{aligned} T_s &= \text{ application's native solve time} \\ o_a &= \text{ overhead of rollback avoidance} \end{aligned}$$

To understand how these parameters map to two rollback avoidance techniques, we consider two examples: repairing memory errors and proactive process migration. When an ECC error is detected in memory, the memory controller raises a machine check exception (MCE) in the processor. Current HPC operating systems terminate the offending process or reboot the entire node in response to a MCE. However, some proposed techniques allow detected memory errors to be corrected by either using application knowledge [9], [19] or by leveraging redundant information in the application's memory [14]. In these cases, p_a is the probability that a memory error can be corrected such that the application can continue without experiencing a failure and rolling back. The overhead, o_a, of these techniques is the expected value of the sum of the additional time that is necessary to prepare for failure and the time required to recover from memory errors as they occur.

Another approach to rollback avoidance is to proactively migrate processes away from hardware that is predicted to fail. A common approach is to continuously examine system event logs looking for sequences of log entries that are believed to be indicative of impending failure [20]. Another approach is to use health monitoring to determine when a particular node is likely to fail based on environmental observations (e.g., temperature, input voltage, etc.) [21]. In both cases, p_a is equal to the *recall* of the prediction mechanism. Recall captures the fraction of failures in the system that are correctly predicted. The overhead, o_a, of these techniques is comprised of two principal components: (i) the cost of the prediction mechanism, including gathering and processing the information necessary to make predictions; and (ii) the cost of migrating processes following the prediction of a failure, including costs due to false positives.

B. Augmenting Coordinated Checkpointing

Given this general conceptual model, we build an analytical model of the impact of rollback avoidance techniques on application performance when used in conjunction with traditional coordinated checkpointing. We begin with Daly's model[1] of application performance [16]. Our conceptual model of rollback avoidance allows us to modify the system's mean time to interrupt (MTTI) and the application's solve time to

[1]We could have used our conceptual model to extend any accurate application performance model. We chose Daly's because it is accurate and widely accepted.

account for the impact these techniques have on application performance. The resulting expression for application runtime (T_w) is shown in Equation 3.[2] Because rollback avoidance effectively increases the system's MTTI, the optimal checkpoint interval also increases.

$$T_w(p_a, o_a) = M' e^{R/M'} \left(e^{(\tau'_{opt}+\delta)/M'} - 1 \right) \frac{T'_s}{\tau'_{opt}} \quad (3)$$

where

R = time to required to restart a failed node
M' = system MTTI with rollback avoidance (see Equation 1)
δ = checkpoint commit time
τ'_{opt} = optimal checkpoint interval computed using M'
T'_s = solve time of the application (see Equation 2)

C. Replacing Coordinated Checkpointing

It is also instructive to consider how effective rollback avoidance would need to be in order to be a viable replacement for coordinated checkpoint/restart. We again build upon our conceptual model of rollback avoidance and Daly's model of coordinated checkpoint/restart. The resulting model of rollback avoidance in the absence of checkpoint/restart is shown in Equation 4.[3]

$$T_w(p_a, o_a) = M' e^{R/M'} (e^{T'_s/M'} - 1) \quad (4)$$

where

R = time to required to restart a failed node
M' = system MTTI with rollback avoidance (see Equation 1)
T'_s = solve time of the application (see Equation 2)

IV. Validation

In this section, we validate the accuracy of the two models we introduced in the preceding section. We accomplish this by comparing the application runtime predicted by our model to the application runtime predicted by a freely available validated [2], [22], [23] simulator. Because the simulator did not account for rollback avoidance, we modified it to ignore failures on simulated nodes with probability p_a. Following a node failure, the interarrival time of the next failure is the sum of the interarrival times of the failures that are successfully avoided and the interarrival time of the next failure that cannot be avoided.

Figure 1 shows a comparison between our model of the impact of failure avoidance and coordinated checkpointing (Equation 3) and the modified simulator. To minimize the modifications to the simulator, the simulator's computation of the optimal checkpoint interval is unmodified. As a result, for the purposes of this comparison, we modified the model such that it computed the checkpoint interval based on the system MTTI without rollback avoidance. Each of the three subfigures show the results for a different pair of values for o_a and p_a. In each case, the application runtime predicted by the model closely matches (within 1%) the value predicted by the simulator.

Figure 2 shows a comparison between our model of the impact of rollback avoidance (Equation 4) and the modified simulator. Once again, each of the three subfigures show the results for a different pair of values for o_a and p_a. In each case, the application runtime predicted by the model closely matches (within 2%) the value predicted by the simulator.

V. Case Study: Process Replication

To demonstrate the power of our model we use it to examine an existing process replication library, *r*MPI [2]. *r*MPI is a user-level MPI library that facilitates process replication in HPC systems by ensuring that each process and its replica receive all application messages even if one of the processes fails.

A. Model Parameters

Modeling *r*MPI requires appropriate values for the overhead (o_a) and the avoidance probability (p_a). Naively, the overhead is equal to the overhead of replicating messages. Because each process and its replica run simultaneously, this would allow us to accurately model time-to-solution. However, this underestimates the cost of replication. To more completely account for the cost of replicating every process, o_a must be at least 1.0. In this case, our model will not yield raw execution time; it measures resource usage in terms of the number of node-hours required for the computation.

The runtime overhead of *r*MPI is generally less than 5%, depending on the application [2]. Because this overhead affects each process and its replica, we choose $o_a = 1.10$. The probability of avoiding the effects of a failure by using *r*MPI is given by the birthday problem [2]. On average, the number of faults that will occur before the application observes a node failure (i.e., a process and its replica are down simultaneously) is given by Equation 5.

$$F(n) \approx \sqrt{\frac{\pi n}{2}} + \frac{2}{3} \quad (5)$$

where n is the total number of nodes that comprise the system. The probability of correcting any single failure can be derived from this expression and is shown in Equation 6.

$$p_a(n) = \frac{3\sqrt{\pi n} - \sqrt{2}}{3\sqrt{\pi n} + 2\sqrt{2}} \quad (6)$$

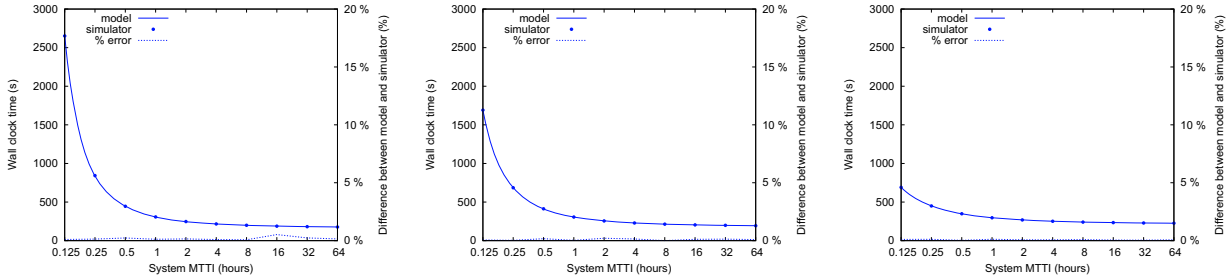
The remainder of the model parameters are duplicated from the evaluation of rMPI. A summary of the model parameters is shown in Table I.

[2] Although T_w is undefined for $p_a = 1.0$, we note that the model is correct in the limit:

$$\lim_{p_a \to 1.0} M' e^{R/M'} \left(e^{(\tau'_{opt}+\delta)/M'} - 1 \right) \frac{T'_s}{\tau'_{opt}} = T'_s$$

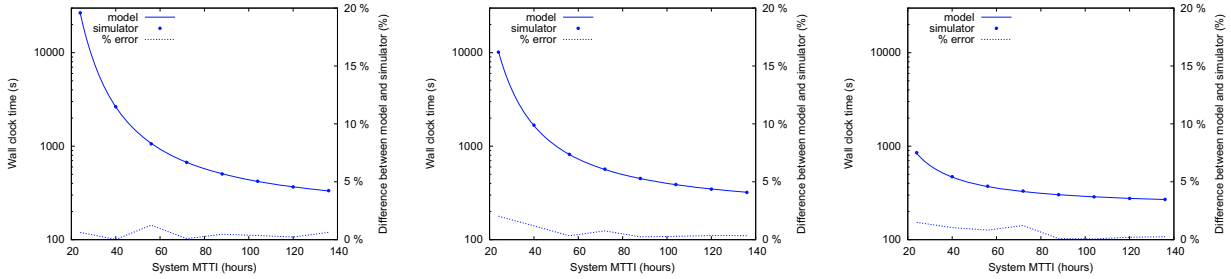
[3] As with the previous model, T_w is undefined when $p_a = 1.0$. However, the model is again correct in the limit:

$$\lim_{p_a \to 1.0} M' e^{R/M'} (e^{T'_s/M'} - 1) = T'_s$$



(a) $p_a = 0.00$, $o_a = 0.00$ (b) $p_a = 0.25$, $o_a = 0.10$ (c) $p_a = 0.75$, $o_a = 0.30$

Figure 1. *Augmenting C/R*. Validation of our general analytic model of the impact of augmenting coordinated checkpoint/restart with rollback avoidance on application performance against an existing simulator [2], [22]. We modified the simulator to account for rollback avoidance. The model and the simulator use identical values for the solution time ($T_s = 168$ hours), the checkpoint commit time ($\delta = 5$ minutes) and node MTBF ($\Theta_n = 5$ years). Both use the optimal checkpoint interval (τ_{opt}). The subfigures of this figure compare the results across several values of p_a and o_a. The values predicted by the model and the simulator error differ by less than 1% in all cases.



(a) $p_a = 0.00$, $o_a = 0.00$ (b) $p_a = 0.25$, $o_a = 0.10$ (c) $p_a = 0.75$, $o_a = 0.30$

Figure 2. *Replacing C/R*. Validation of our general analytic model of the impact of replacing coordinated checkpoint/restart with rollback avoidance on application performance against an existing simulator [2], [22]. We modified the simulator to account for rollback avoidance. The model and the simulator use identical values for the solution time ($T_s = 168$ hours) and node MTBF ($\Theta_n = 5$ years). The subfigures of this figure compare the results across several values of p_a and o_a. The values predicted by the model and the simulator error differ by less than 2% in all cases.

Parameter	Description	Value
T_s	SOLVE TIME	168 hours
Θ_n	NODE MEAN TIME TO INTERRUPT (MTTI)	5 years
δ	CHECKPOINT COMMIT TIME	15 minutes
R	RESTART TIME	15 minutes
o_a	ROLLBACK AVOIDANCE OVERHEAD	1.10
p_a	PROBABILITY OF ROLLBACK AVOIDANCE	*see* Equation 6

Table I. MODEL PARAMETERS FOR EXAMINING THE PERFORMANCE OF PROCESS REPLICATION.

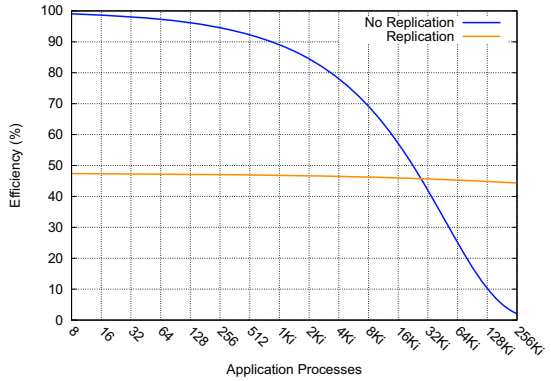


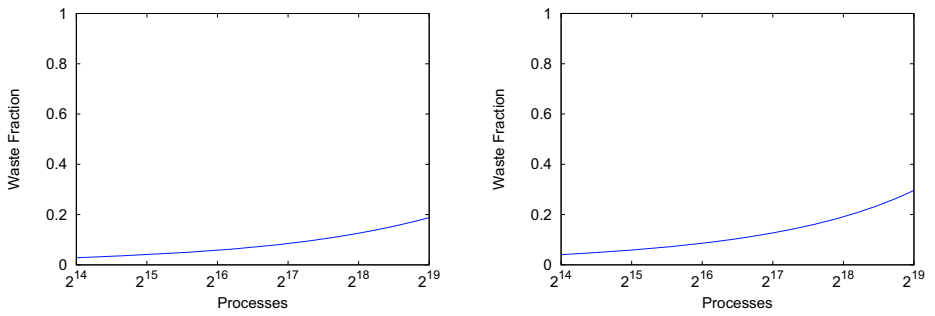
Figure 3. Comparison of application efficiency with and without process replication. The results obtained using our approach-independent model closely match existing data on process replication (*see* Figure 7 of Ferreira [2]). These data were collected using the parameters in Table I.

B. Model Performance

Given our model and this set of parameters, we can compare the efficiency of process replication against the efficiency of coordinated checkpoint/restart. Figure 3 shows the system efficiency as a function of system size both with and without process replication. The key observation is that this figure closely matches the data collected in the evaluation of rMPI (cf. Figure 7 in [2]). For small systems, the overhead of replication is prohibitive. However, as system size increases and failures become more likely, replication is more efficient than the baseline approach.

VI. CASE STUDY: FAULT PREDICTION

Techniques for predicting the occurrence of failures have been widely studied [4], [20], [25]–[27]. Accurately predicting failures before they occur may allow the system to take corrective action that could prevent the application from being compromised. The benefit of this family of approaches is typically characterized by *recall*: the fraction of the total number of failures that can be predicted. In terms of our model, p_a is equal to the method's recall. There are two principal costs of failure prediction: (i) *runtime overhead* (o_{rt}), the costs associated with processing system information (e.g., collecting and analyzing system log files); and (ii) *false positive overhead*



(a) Good Predictor $P = 0.82$, $R = 0.85$ (b) Bad Predictor $P = 0.4$ $R = 0.7$

Figure 4. Impact of failure prediction on the fraction of waste time (i.e., time spent not doing useful work). These result closely match existing modeling and simulation data for exponentially distributed failures (*cf.* [24] : compare the good predictor results to the red line in Figure 3(b); compare the bad predictor results to the red line in Figure 4(b) [24]). These data demonstrate that our model is capable reproducing key results in the field while being general enough to a account of additional important scenarios.

(o_{fp}), the costs associated with unnecessarily initiating proactive response (e.g., needlessly migrating a process). As shown in Equation 7, the false positive overhead can be expressed in terms of the recall and precision of a given prediction method.

$$o_{fp} = \frac{(1-P)R}{PM}c \tag{7}$$

where

$P =$ precision of the prediction method
$R =$ recall of the prediction method
$c =$ average time required for a proactive response
$M =$ system MTTI

As a result, we can express the overhead of failure prediction as shown in Equation 8.

$$o_a = \frac{(1-P)R}{PM}c + o_{rt} \tag{8}$$

where

$o_{rt} =$ runtime overhead of the prediction method expressed as a fraction of total execution time

Figure 4 compares the results of our model to existing data on the impact of fault prediction on application performance [24]. Specifically, we compare two specific cases of predictor performance. Figure 4(a) shows the fraction of waste time (i.e., time spent by the system doing something other than directly advancing the application's computation) for a "good" predictor ($P = 0.82$, $R = 0.85$) as a function of system size. Figure 4(b) shows the same result for a "bad" predictor ($P = 0.4$, $R = 0.7$). These results demonstrate that our model closely matches existing modeling and simulation data on the the impact of fault prediction (*cf.* [24]: compare the red lines in Figure 3(b) to the good predictor results (Figure 4(a)), and the red lines in Figure 4(b) to the bad predictor results (Figure 4(b))). We obtain similar results by using our model to reproduce the other figures presented by Aupy et al. that assume exponentially distributed failures. Although our model is technique-independent, we are able to closely match the results of this technique-specific model.

Our model also allows us to more closely examine the claim–first articulated by Aupy et al. [24]–that recall is more important than precision for fault prediction. In Figure 5, we examine the relative impacts of precision and recall on application performance. For the data in this figure, we fixed the system MTTI (M) at 45 minutes, a value that is well within the range currently projected for the first exascale machine. We also fixed the proactive response cost at two minutes. While this value is longer than the prediction window of current techniques, it is also shorter than most current proactive measures (e.g., checkpointing) would require (*cf.* [20]). The data in this figure show that above a modest threshold (e.g., 50%—many modern methods are above 90%), precision has relatively little influence on application execution time.

Figure 6(a) illustrates the impact of precision relative to runtime overhead. The data in this figure were collected with the recall value fixed at 40%—a value achieved by many current methods. Each of the lines in this figure corresponds to a different precision, ranging from 25% to 95%. We observe that these four lines almost entirely overlap one another. Although the existing literature is largely silent on the costs of predicting failures, this figure shows that runtime overhead has a much larger effect on application runtime than precision does. Moreover, in this configuration, once the runtime overhead exceeds 20% the benefits of failure prediction disappear and application execution time increases. Finally, the impact of runtime overhead is significant. Below 20% the slopes of these lines are nearly -1; any increase in the runtime overhead results in a commensurate increase in execution time. As a result, efforts to improve precision that result in increases in the runtime overhead will yield little benefit.

Figure 6(b) shows the impact of recall relative on runtime overhead. The data in this figure were collected with the precision value fixed at 95%. Similar to the previous figure, each of the lines in this plot correspond to a different recall value. If we consider horizontal slices of this figure, we observe that increasing the recall value even if it results in relatively large increases in the runtime overhead may be fruitful. For example, suppose that we have a method for which the runtime overhead is 0% and the recall is 50% (i.e., in Figure 6(b) our method sits at the intersection of the orange curve and the y-axis–a speedup of 1.31). For a recall value of 75%, we would see the same speedup if the runtime overhead

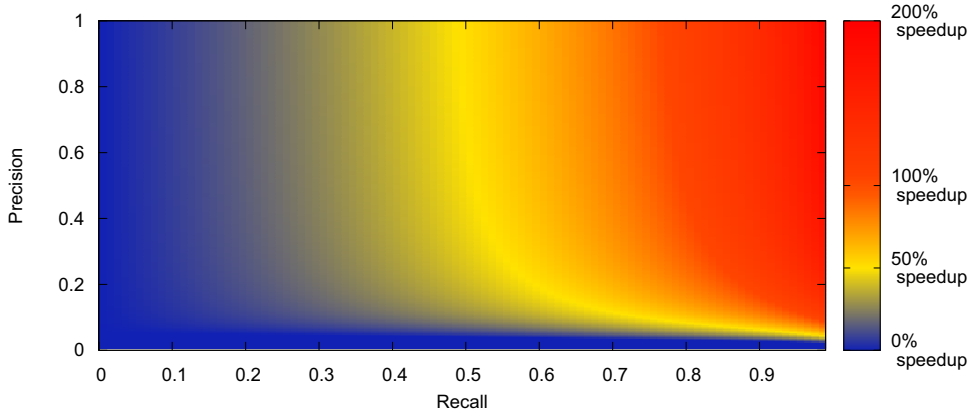
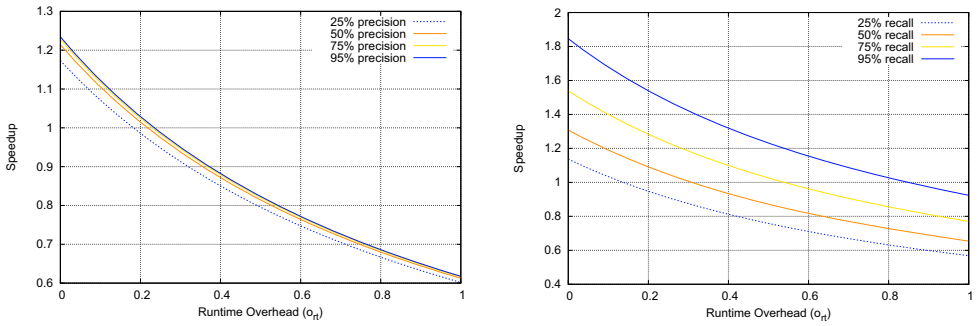


Figure 5. Heat map showing the impact of precision and recall on application speedup. This figure shows that the performance impact of fault prediction is much more strongly influenced by recall than precision. These results were generated using a solve time (T_s) of 168 hours, a system MTTI (M) of 45 minutes, a checkpoint commit time (δ) of 15 minutes, and a restart time (R) of 10 minutes. Speedup is calculated relative to the execution time with no rollback avoidance.



(a) Precision (b) Recall

Figure 6. Comparing the relative impact of precision and recall on application speedup as a function of the runtime overhead of fault prediction. These figures confirm that precision has much less impact on application performance than either recall or runtime overhead.

was 17.8%. In other words, any method that increases the recall value from 50% to 75% and imposes a runtime overhead of less than 17.8% will yield a benefit. The point is that this figure shows that innovative techniques that increase recall—even at the cost of runtime overhead—may increase the overall benefit of failure prediction.

VII. Analysis & Discussion

A. Designing Rollback Avoidance for Exascale

Due to the projected overheads of coordinated checkpoint/restart at extreme-scale, significant effort has been devoted to developing new rollback avoidance techniques. In particular, considerable attention has been paid to application-specific techniques.

In this section, we use our model to explore the projected design space of the first exascale system to determine where new application-specific techniques may offer the greatest benefits. We begin by examining how the relationship between the avoidance probability and the runtime overhead affects the design. For many application-specific methods—fault-tolerant algorithms, for example—the overhead represents how much longer the computation takes to converge than if no error had occurred. Figure 7 shows the application speedup for several avoidance probabilities as a function of overhead. By way of comparison, the dashed horizontal lines in this figure show the speedups that can be achieved using existing application-independent techniques. Even if they impose relatively large overheads, there is room for application-specific techniques to outperform existing techniques if they can avoid a high percentage of all sources of rollbacks. For the many application-specific methods that can only avoid rollbacks that are due to a subset of system failures, achieving such high avoidance probabilities will be challenging.

Figure 7 represents a single point in the exascale design space. To evaluate the potential benefits over the entire design space, we consider a strawman. Our strawman is able to avoid 80% of all rollbacks while imposing a 10% overhead.

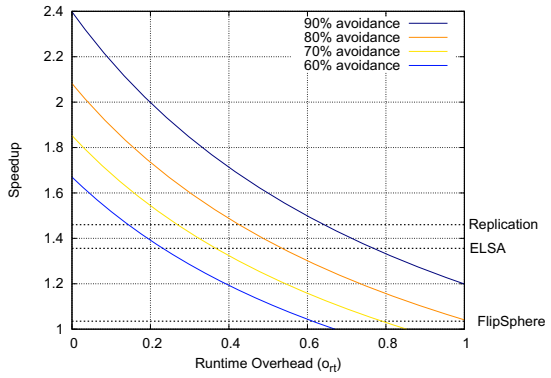


Figure 7. Application speedup (i.e., the relative reduction in application execution time) for several hypothetical rollback avoidance techniques as a function of the overhead imposed by the technique. For comparison, the dashed horizontal lines correspond to the speedup achieved by existing techniques. These results were generated using a solve time (T_s) of 168 hours, a system MTTI (M) of 45 minutes, a checkpoint commit time (δ) of 5 minutes, a restart time (R) of 10 minutes, and no runtime overhead ($o_a = 0.0$). Speedup is calculated relative to the execution time with no rollback avoidance.

This is an aggressive strawman given that many application-specific techniques only protect against rollbacks caused by memory corruption, which is a fraction of all rollbacks. Given this strawman, we compare its relative performance to three existing techniques: replication, fault prediction (ELSA) and FlipSphere. For replication, we assume a perfectly strong scaling application (i.e., running the same application with process replication requires twice as much time). ELSA is a fault prediction toolkit with a precision of 93% and a recall of 43% [20]. Although the runtime overhead of ELSA has not been publicly documented, we assume that it is no more than 5%. FlipSphere is a software-based memory error correction library. It protects 90% of application memory and imposes an overhead of 40%. Because FlipSphere only protects against rollbacks that are due to memory corruption, its protective benefit is less than the fraction of memory that it protects. We generously assume that 50% of all rollbacks are due to some form of memory corruption. As a result, we use $p_a = 0.45$ for FlipSphere in our analysis. Figure 8 shows the results of the comparison of our strawman to these three techniques. In these heat maps, the blue regions are where our strawman provides little or no improvement, the red regions are where our strawman offers significant benefits. These figures indicate that the benefits of this strawman depend on where in this design space the first exascale system appears. For systems with small MTTI and small checkpoint bandwidths, our strawman will be outperformed by replication. For systems with large MTTI and large checkpoint bandwidths, our strawman will be outperformed by fault prediction. However, somewhere between these two extremes in the design space there appears to be a significant opportunity for rollback avoidance techniques such as our strawman to provide significant benefits.

B. Replacing Coordinated Checkpoint/Restart

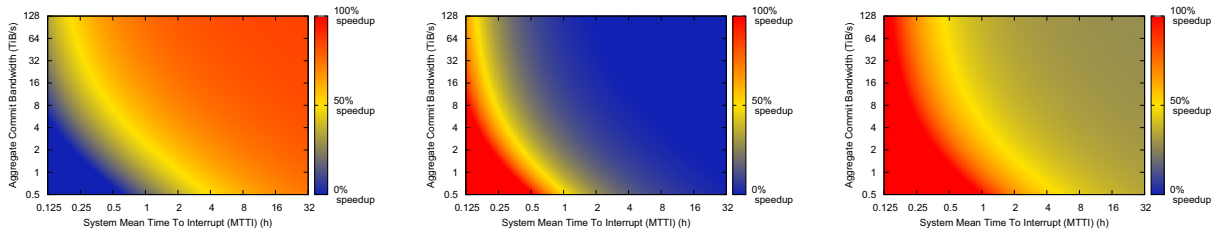
Throughout this paper, we have focused on using rollback avoidance to reduce the performance impact of coordinated checkpoint/restart. In principle, rollback avoidance could also be used as a replacement for coordinated checkpointing. However, for any rollback avoidance to be an effective replacement, its probability of avoiding rollback would have be very close to 1.0. To see why this is, suppose we have a a technique that avoids 90% of rollbacks on a system with an MTTI of 1 hour. In this case, the effective MTTI given by Equation 1 is 10 hours. Given exponentially distributed failures, the probability of completing a 168-hour job without encountering a failure (which is the criteria for success in this case) is approximately 5.0×10^{-8}. Figure 11 illustrates this phenomenon more concretely. We discuss this figure in more detail later, but for our current purposes we note the relative performance of coordinated checkpoint/restart (C/R only) and a rollback avoidance technique that avoids 99% of rollbacks and imposes no overhead (Replacing C/R). Even for this effective technique, there are still circumstances in which it cannot compete with coordinated checkpoint/restart. In particular, for values of system MTTI below one hour this hypothetical replacement technique is no longer competitive with traditional coordinated checkpointing. Therefore, our model shows that rollback avoidance alone is unlikely to perform well on exascale systems.

C. Assessing the Impact of Model Parameters

Our model for predicting application performance also allows for a careful exploration of the design-space for rollback avoidance techniques. Examining the characteristics that have the greatest impact on application can inform the design, development and refinement of current and future rollback avoidance techniques. In this section, we consider the impact of three of our model's parameters: (i) probability of rollback avoidance (p_a); (ii) rollback avoidance overhead (o_a); and (iii) system MTTI (M).

1) Probability of Rollback Avoidance: Combining rollback avoidance techniques and coordinated checkpoint/restart has the potential to improve application performance by reducing the frequency with which the application is forced to roll back to a previous checkpoint. We begin by considering the impact that the probability of rollback has on application performance. Figure 9 shows the decrease in application runtime (i.e., speedup) as a function of the system MTTI for several values of p_a. To isolate the impact of p_a, we consider approaches that impose no overhead (i.e., $o_a = 0.0$). To put the system MTTI into context, a system MTTI of 0.5 hours corresponds to a system comprised of 65,536 (64Ki) nodes, each with an MTTI of 3.75 years. Similarly, a system MTTI of 8 hours corresponds to a system comprised of nodes whose MTTI is 60 years. These two values roughly represent the range of node MTTIs that are currently projected for the first exascale system [1].

The most striking feature of this figure is how unreliable the system must be before even very good rollback avoidance techniques (e.g., able to avoid rollback 80% of the time) significantly improve application performance. For systems with an MTTI greater than 8 hours, we observe very little speedup in application runtime. This is an especially stark result given that these figures assume zero overhead. However, for unreliable systems (e.g., systems with an MTTI that is less than 2 hours) rollback avoidance yields significant benefits, in some case reducing the application runtime by more than 81%.



(a) Replication (b) ELSA (c) FlipSphere

Figure 8. Heatmaps showing the improvement of a strawman application-specific technique ($p_a = 0.80$, $o_a = 0.10$) against three existing application-independent techniques. Each of these heatmaps covers the range of MTTI (M) and an aggregate checkpoint commit bandwidth (β) that is currently projected for exascale. We assume a system comprised of 128Ki nodes, each of which has 8 GiB of memory. These figures collectively show that this aggressive strawman would improve application performance over a fraction of the exascale design space.

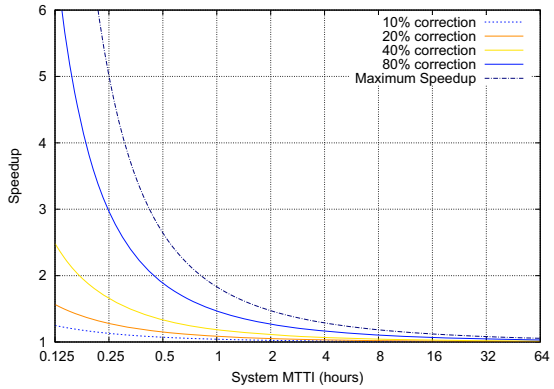


Figure 9. The effect of a range of rollback avoidance probabilities on application speedup as a function of system reliability. These results were generated using a solve time (T_s) of 168 hours, a checkpoint commit time (δ) of 5 minutes, a restart time (R) of 10 minutes, and no runtime overhead for rollback avoidance ($o_a = 0.0$). Speedup is calculated relative to the execution time with no error correction. "Maximum Speedup" represents the maximum increase in application performance that can be achieved by eliminating all wasted time.

The reason for this behavior is due to a phenomenon that is closely related to Amdahl's Law [28]. An application that runs on a system with an MTTI of eight hours operates with an efficiency of 85%. In other words, 85% of the system time required to run the application is used to perform useful work. A direct consequence of this fact is that the maximum possible speedup is approximately 18% (1.00/0.85). The dashed curve in Figure 9 shows the maximum speedup as a function of system MTTI. This curve represents the maximum application speedup that could be achieved if *all* wasted time (e.g., writing checkpoints, restarting nodes, redoing lost work) was eliminated. Until the system MTTI drops below approximately 8 hours, even heroic efforts to eliminate wasted system time will yield only modest gains in application speed.

2) Rollback Avoidance Overhead: We next consider the impact of the overhead of rollback avoidance techniques on application performance. Figure 10 shows the relationship between the probability of avoiding rollback and the overhead that avoidance imposes on the application. This figure considers a case where the efficiency of the application with checkpoint/restart is low: the system MTTI is low (45 minutes) and the checkpoint commit time is high (15 minutes). Although these values are well within the range projected for the first exascale system, they represent a system in which less than half of the application's runtime is available for useful computation. This figure also demonstrates the importance of minimizing the overhead of rollback avoidance for techniques that only avoid a modest fraction of failures. For example, a technique that imposes a 20% overhead will not improve application performance unless it is able to avoid more than 23% of all failures. In contrast, reducing the overhead to 10%, shows improvement when avoiding just 12% of failures. For techniques that avoid a large fraction of failures (e.g., more than 70%), overhead has a lesser impact; even when overhead consumes a significant fraction of the application's system time we observe application speedup.

3) System MTTI: As we have already observed, system MTTI has an (unsurprisingly) large impact on the effectiveness of rollback avoidance mechanisms. To isolate the effects of system MTTI and to reduce the number of dimensions in the configuration-space, we consider three representative approaches to fault tolerance: (i) coordinated checkpoint/restart only; (ii) coordinated checkpoint/restart augmented with a rollback avoidance technique for which the probability of failure avoidance is 25% and the overhead is 10%; and (iii) a rollback avoidance technique that is able to avoid 99% of all failures with 1% overhead. All other parameters are taken from Table I. We consider a range of MTTIs from 30 minutes to 64 hours. Figure 11 shows the performance of these three approaches as a function of system MTTI. The approach that yields the best application performance depends on the system MTTI which of three different regions a system is operating in. In the configuration shown in this figure, augmenting coordinated checkpoint/restart with rollback avoidance is the most effective approach for unreliable systems, those with an MTTI below approximately 81 minutes. For reliable systems, those with an MTTI above approximately 2.9 hours, the rollback avoidance technique by itself is most effective. Coordinated checkpoint/restart is the preferred approach in a small range of system MTTI values between these two extremes.

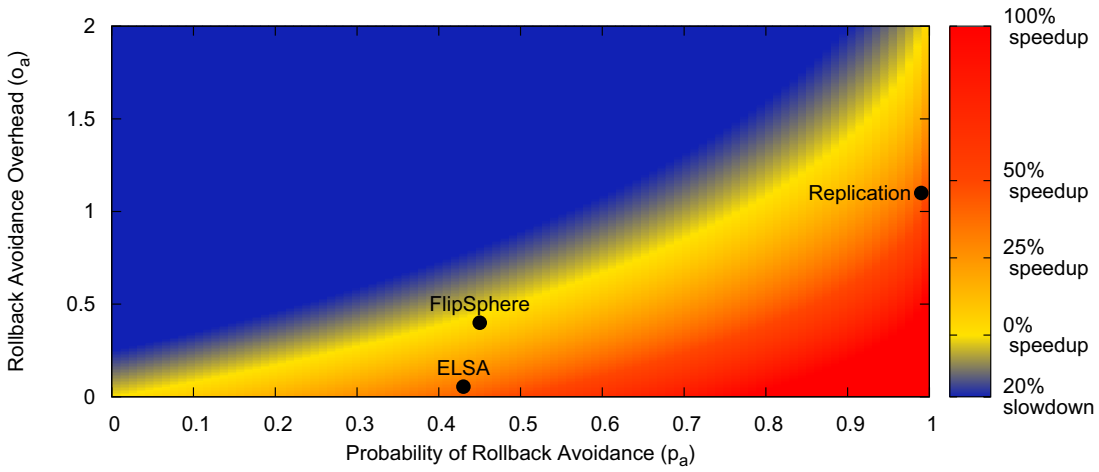


Figure 10. The impact of overhead and the probability of rollback avoidance on application performance. These results were generated using a solve time (T_s) of 168 hours, a system MTTI (M) of 45 minutes, a checkpoint commit time (δ) of 5 minutes and a restart time (R) of 10 minutes. Speedup is calculated relative to the execution time with no error correction. For comparison, the performance of several rollback avoidance techniques are shown: *r*MPI, process-level replication for HPC [2] ($p_a = 0.99$, $o_a = 1.1$); ELSA, log-based prediction library [20] ($p_a = 0.43$, $o_a = 0.05$); and FlipSphere, software-based error correction library for HPC [15]($p_a = 0.45$, $o_a = 0.4$).

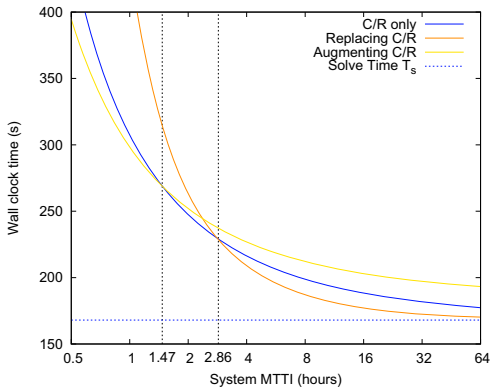


Figure 11. The impact techniques on application performance as a function of system MTTI. We compare three different approaches to fault tolerance: (i) Coordinated Checkpointing only; (ii) Augmenting Checkpointing with Rollback Avoidance ($p_a = 0.25$ / $o_a = 0.10$); and Rollback Avoidance only ($p_a = 0.99$ / $o_a = 0.00$). The results in this figure were generated using a solve time (T_s) of 168 hours, a checkpoint commit time (δ) of 5 minutes, and a restart time (R) of 10 minutes.

VIII. Related Work

A number of approaches for avoiding rollback have been proposed [2], [4], [9], [14], [19], [21]. In addition, several approach-specific models have been developed to evaluate the performance impact of these approaches on applications running on extreme-scale systems. Ferreira et al., for example, construct a probabilistic model for process-level replication in the context of high-performance computing [2]. Using this model, they show where in the exascale design-space replication outperforms traditional coordinated checkpoint restart to a parallel filesystem.

Wingstrom introduced the notion of modeling waste time (i.e., time not available for application computation) rather than total application execution time [29]. Instead of building on Daly's checkpointing model, the existing models of waste time optimize for steady-state operation rather than per-job performance. This approach has been widely adopted in the fault prediction literature. Cappello et al. used this model to evaluate the performance impact of preventive migration and preventive checkpointing [5]. Gainaru et al. incorporate precision and recall into the model to evaluate the effectiveness of using signal processing to predict failure [4]. Aupy et al. uses the model to examine the impact of fault prediction on preventive checkpointing [24]. They also contend that Daly's model of the optimal checkpoint interval contains an error and propose an alternative formulation.

Unlike these technique-specific models, the model described in this paper deals with the class of approaches in general and can reproduce the results of at least two approach-specific models. Moreover, by considering rollback avoidance generally our model allows us to provide guidance to the design and development of future rollback avoidance techniques. It also facilitates direct comparisons between proposed approaches.

IX. Conclusions

In this paper, we introduced and validated two analytical models for evaluating the impact of rollback avoidance on application performance in large-scale systems. These models allow us to consider failure avoidance both in conjunction with coordinated checkpointing and in isolation. Using

these models, we examined the impact of failure avoidance techniques based on system characteristics. In particular, we showed that for reliable systems, using rollback avoidance to augment checkpointing yields only modest performance gains. However, as systems grow in size and failures occur more frequently, rollback avoidance can yield significant improvements. We also showed that even very effective rollback avoidance techniques are unlikely to replace coordinated checkpointing unless future systems are much more reliable than currently projected. More broadly, these models allow for an exploration of system and application parameters for which rollback avoidance can potentially provide significant benefits.

ACKNOWLEDGMENTS

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U. S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] K. B. et al, "Exascale computing study: Technology challenges in achieving exascale systems," http://www.science.energy.gov/ascr/Research/CS/DARPAexascale-hardware(2008).pdf, Sep. 2008.

[2] K. Ferreira, R. Riesen, P. Bridges, D. Arnold, J. Stearley, J. H. L. III, R. Oldfield, K. Pedretti, and R. Brightwell, "Evaluating the viability of process replication reliability for exascale systems," in *SC*, S. Lathrop, J. Costa, and W. Kramer, Eds. ACM, Nov. 2011.

[3] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth, "Modeling the impact of checkpoints on next-generation systems," in *24th IEEE Conference on Mass Storage Systems and Technologies*, Sep. 2007, pp. 30–46.

[4] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into HPC systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC12)*, 2012, p. 77.

[5] F. Cappello, H. Casanova, and Y. Robert, "Checkpointing vs. migration for post-petascale supercomputers," in *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 2010, pp. 168–177.

[6] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. B. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for HPC," in *Proceedings of the* 32^{nd} *International Conference on Distributed Computing Systems (ICDCS) 2012*. Macau, SAR, China: IEEE Computer Society, Los Alamitos, CA, USA, Jun. 18-21, 2012, pp. 615–626.

[7] D. Fiala, F. Mueller, C. Engelmann, K. B. Ferreira, R. Brightwell, and R. Riesen, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the* 25^{th} *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2012*. Salt Lake City, UT, USA: ACM Press, New York, NY, USA, Nov. 10-16, 2012, pp. 78:1–78:12.

[8] Y. Kim, J. S. Plank, and J. J. Dongarra, "Fault tolerant matrix operations for networks of workstations using multiple checkpointing." in *High Performance Computing on the Information Superhighway. HPC Asia '97*. Seoul, South Korea: Los Alamitos, CA, USA : IEEE Comput. Soc. Press, 1997, April 1997, pp. 460–465.

[9] P. G. Bridges, M. Hoemmen, K. B. Ferreira, M. A. Heroux, P. Soltero, and R. Brightwell, "Cooperative application/OS DRAM fault recovery," in *Euro-Par 2011: Parallel Processing Workshops*. Springer, 2012, pp. 241–250.

[10] Z. Chen and J. Dongarra, "Algorithm-based checkpoint-free fault tolerance for parallel matrix computations on volatile resources," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, april 2006, p. 10 pp.

[11] Z. Chen, "Extending algorithm-based fault tolerance to tolerate fail-stop failures in high performance distributed environments," in *IPDPS*. IEEE, 2008, pp. 1–8.

[12] H. Kuang-Hua and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. 33, pp. 518–528, June 1984. [Online]. Available: http://dx.doi.org/10.1109/TC.1984.1676475

[13] P. P. Shirvani, N. R. Saxena, and E. J. McCluskey, "Software-implemented EDAC protection against SEUs," *Reliability, IEEE Transactions on*, vol. 49, no. 3, pp. 273–284, 2000.

[14] S. Levy, P. G. Bridges, K. B. Ferreira, A. P. Thompson, and C. Trott, "Evaluating the feasibility of using memory content similarity to improve system resilience," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2013, p. 7.

[15] D. Fiala, K. B. Ferreira, and F. Mueller, "FlipSphere: A software-based DRAM error detection and correction library for HPC," Sandia National Laboratories, Technical Report SAND2014-0438C, Feb. 2014.

[16] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, no. 3, pp. 303–312, 2006.

[17] F. Cappello, "Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities," *IJHPCA*, vol. 23, no. 3, pp. 212–226, 2009.

[18] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *Computers, IEEE Transactions on*, vol. 100, no. 6, pp. 518–528, 1984.

[19] J. N. Glosli, D. F. Richards, K. Caspersen, R. Rudd, J. A. Gunnels, and F. H. Streitz, "Extending stability beyond CPU millennium: a micron-scale atomistic simulation of Kelvin-Helmholtz instability," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, p. 58.

[20] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the Shrew: Modeling the Normal and Faulty Behaviour of Large-scale HPC Systems," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 1168–1179.

[21] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in HPC environments," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 43.

[22] R. Riesen, K. Ferreira, J. Stearley, R. Oldfield, J. H. Laros III, K. Pedretti, R. Brightwell *et al.*, "Redundant computing for exascale systems," Technical report SAND2010-8709, Sandia National Laboratories, Tech. Rep., 2010.

[23] Sandia National Laboratories, "`App_model` - application simulator (gpl)," http://www.cs.sandia.gov/web1400/1400_download.html, Feb. 10 2014.

[24] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing algorithms and fault prediction," *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2048–2064, 2014.

[25] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Failure prediction for hpc systems and applications: Current situation and open issues," *International Journal of High Performance Computing Applications*, vol. 27, no. 3, pp. 273–282, 2013. [Online]. Available: http://hpc.sagepub.com/content/27/3/273.abstract

[26] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, no. 3, pp. 10:1–10:42, Mar. 2010.

[27] E. W. Fulp, G. A. Fink, and J. N. Haack, "Predicting computer system failures using support vector machines," in *Proceedings of the First USENIX conference on Analysis of system logs*, ser. WASL'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 5–5.

[28] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[29] J. Wingstrom, "Overcoming the difficulties created by the volatile nature of desktop grids through understanding, prediction and redundancy," Ph.D. dissertation, Ph. D. thesis, University of Hawai i at Manoa, 2009.