2018 IEEE International Parallel and Distributed Processing Symposium Workshops

An Analysis of Multilevel Checkpoint Performance Models

Daniel Dauwe*, Sudeep Pasricha*†, Anthony A. Maciejewski* and Howard Jay Siegel*†

*Department of Electrical and Computer Engineering
†Department of Computer Science
Colorado State University, Fort Collins, CO, 80523, USA
Email: ddauwe@rams.colostate.edu, sudeep@colostate.edu, aam@colostate.edu, hj@colostate.edu

***Abstract*—Periodic application checkpointing to a parallel file system has for years been the standard strategy for providing high performance computing (HPC) systems with resilience to system failures. The traditional checkpoint/restart protocol has until recently proved sufficient for mitigating the impact of these failures on application performance. However, as system sizes approach exascale levels, the frequency of failures and the time required to checkpoint/restart an exascale-size application increases and the efficiency of traditional checkpointing decreases substantially, making it no longer a viable option for providing resilience to future HPC systems. The most frequently proposed solution for providing future systems with resilience has been to design multilevel checkpointing protocols. However, the relationship between system failure rates, checkpoint/restart overhead, and duration of time between successive checkpoints is complex and finding the optimal duration of time between successive checkpoints is an open and challenging problem. This work presents a novel execution time prediction model we have developed that takes into consideration execution events that have not been considered by previous multilevel checkpointing models. We show how this model can be used to select checkpoint intervals and demonstrate why consideration of these execution events is important. We validate our work through simulation and provide a comparison to several optimization strategies proposed in other work to demonstrate the advantage gained by considering these execution events.**

Keywords: exascale resilience; checkpoint restart; multilevel checkpointing; checkpoint optimization; fault tolerance.

I. Introduction

As applications have demanded more computing power over time, high performance computing (HPC) systems have required exponentially increasing numbers of system CPU cores to provide this performance [1]. With HPC systems approaching exascale computing complexities, it is expected that they will require several million CPU cores. Simultaneously, the drive to improve performance and energy-efficiency by fabricating at smaller transistor technologies has decreased component reliability [2].

Because of these trends, as HPC systems approach extreme scales, system failure rates have increased rapidly. A recent study of the Blue Waters system in [3] indicates that a $2.2\times$ increase in application size (from $10,000$ system nodes to $22,000$ system nodes) resulted in a $20\times$ increase in the probability of application failure. Given that [4] suggests that an exascale application is likely to require at least $100,000$ system nodes, an exascale system can be expected to experience significantly higher failure rates. Moreover, the study in [3] concludes that for the 13.1 petaflop Blue Waters system, on average an application failure caused by a system-related issue occurs every 15 minutes. An exascale machine is therefore likely to experience failures much more frequently and has been estimated to have a system mean time between failures (MTBF) of as little as three minutes in extreme cases [5].

In such an environment, it is imperative that protocols are in place that allow HPC systems to respond to failures when they occur and mitigate their impact on application performance. However, analysis has shown that current HPC resilience protocols such as traditional checkpoint/restart and redundancy-based resilience are not suitable for scaling to exascale system sizes [2] [6] [7] [8] [9]. One solution for future systems that has been researched over the last decade in anticipation of these extreme-size HPC systems is multilevel checkpointing. Multilevel checkpointing protocols exploit the fact that not all failures require costly restarts of the application from a parallel file system (PFS), and that less severe failures can be restarted in significantly less time from higher levels of memory (e.g., local or remote DRAM).

As with traditional checkpoint/restart protocols, when performing a checkpoint or restart operation the system must temporarily halt application execution. While this is necessary for successful computation in failure-prone systems, every checkpoint incurs overhead that slows the progress of application execution. Just as application progress is impeded by failures if the duration of computation between checkpoints is too large, checkpoints taken too frequently also incur overhead that prevents application progress. This interaction between a given execution environment and the duration of the interval of computation between checkpoints becomes significantly complex with a multilevel checkpointing system. There is an optimal set of successive intervals between levels of a multilevel checkpointing protocol. Determining these optimal intervals is an open problem associated with multilevel checkpointing and is one of the major challenges with the successful implementation of the protocol.

In this work we make the following contributions:

- we provide a detailed comparison among several state-of-the-art techniques for determining multilevel checkpointing intervals;
- we demonstrate the necessity of accounting for failures during checkpoint and restart events when modeling extreme-scale systems;
- we derive an application execution time prediction model in the presence of multilevel checkpointing that can be used for determining the performance of checkpoint intervals;

978-1-5386-5555-9/18/$31.00 ©2018 IEEE
DOI 10.1109/IPDPSW.2018.00125

- we show some of the limits under which multilevel checkpointing ceases to be a viable option for providing resilience to extreme-scale systems;
- we demonstrate the superior accuracy of execution time predictions made with our model, as well as situations in which our model outperforms other state-of-the-art techniques for determining multilevel checkpoint intervals.

The remainder of this paper is organized as follows. We discuss the historical development and recent progress of multilevel checkpointing, and discuss the challenge of constructing accurate models for both application execution time prediction and checkpoint interval optimization in Section II. We present our own approach to modeling a multilevel checkpointing protocol with an arbitrary number of levels in Section III. We provide a set of simulation studies comparing several state-of-the-art multilevel checkpointing models in Section IV. We end the paper with some concluding remarks in Section V.

II. Related work

A. Overview

Traditional checkpoint/restart techniques have been used for decades to mitigate the effects of system failures in HPC systems. The first attempt to optimize checkpoint intervals was Young's first-order approximation in [10]. This execution time model was substantially improved by Daly's higher order execution time approximation in [11]. Daly's work remains the most common approach for optimizing traditional checkpointing. However, traditional checkpointing has been shown to not provide adequate resilience for extreme-sized systems.

The first notion of utilizing different levels of checkpoints for recovering from different types of failures was presented in [12] and this was extended to a more practical (two-level) multilevel checkpointing protocol in the Markov model presented in [13]. Whereas traditional checkpoint/restart performs a checkpoint or restarts from one level (typically the PFS), multilevel checkpointing relies on checkpoints and restarts from multiple levels of memory (e.g., local DRAM, remote DRAM, PFS). The benefits of a multilevel checkpointing model is that time-consuming higher severity failures that typically occur less frequently can restart from a checkpoint to a slower and more reliable (lower) level of memory such as the PFS; whereas the more frequently occurring lower severity failures can restart much more quickly from higher (faster) levels of memory such as local/remote DRAM. Checkpoints that are a "higher" level help restore the system from "higher" severity failures but typically store checkpoint data in correspondingly lower levels of memory. It is usually the case that for a multilevel checkpointing protocol with L checkpoint and restart levels (with durations denoted δ_i and R_i, respectively) and L levels of failure severity (with rates denoted λ_i) we have $\lambda_1 > \lambda_2 > ... > \lambda_L$ while $\delta_1 < \delta_2 ... < \delta_L$ and $R_1 < R_2 ... < R_L$. This relationship benefits multilevel checkpointing by a direct reduction in the number of level L checkpoints/restarts taken and from the fact that lower-level checkpoints to higher levels of the memory hierarchy are able to utilize resources across the system as a whole more effectively, allowing for better scalability and a probable reduction in network overhead. High-level checkpoints/restarts to or from a single PFS tend to be much more dependent on the number of nodes used by the application, i.e., the number of nodes needing to store or retrieve data.

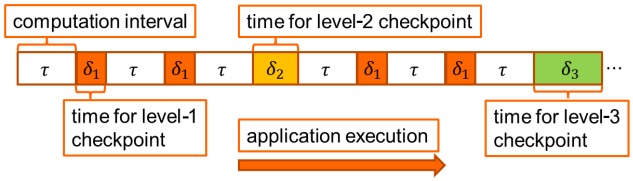

Fig. 1: A checkpoint interval pattern for a three-level checkpointing protocol with its computation interval denoted τ, its checkpoint lengths of each level i denoted δ_i, and a pattern that performs two level-1 checkpoints before a level-2 checkpoint and a single level-2 checkpoint before each level-3 checkpoint.

B. Multilevel Checkpointing Techniques Considered

Our work, and that of the other multilevel checkpointing models presented here, consider HPC systems that employ two types of multilevel checkpointing protocols. We discuss the assumptions and common implementation of each protocol below.

1) *Scalable Checkpoint/Restart* (SCR) [5] is an extension of the ideas from [13]. The authors (Moody et al. [5]) develop their own Markov model that is capable of modeling application execution under a multilevel checkpointing protocol with an arbitrary number of checkpoint levels. SCR is designed as a pattern-based multilevel checkpointing model. This assumes that the duration of computation between successive checkpoints is a fixed amount of time, and that the duration of time between higher checkpoint levels is determined by the discrete number of lower level checkpoints (e.g., each level-2 checkpoint will occur after some number of level-1 checkpoints and each level-1 checkpoint will occur after a fixed interval of computation). These checkpoint interval "patterns" define the frequency of checkpoints at each level. Figure 1 provides an example of a checkpoint interval pattern for a three-level checkpointing protocol with its computation interval denoted τ, its checkpoint lengths of each level i denoted δ_i, and a pattern that performs two level-1 checkpoints before a level-2 checkpoint and a single level-2 checkpoint before each level-3 checkpoint. Though it is not shown in the figure, when a higher-level checkpoint is performed the SCR protocol first performs all lower-level checkpoints (e.g., the length of a level-2 checkpoint, δ_2, would include the time required to first perform a level-1 checkpoint).

SCR is somewhat limiting in its use of patterns because it both restricts checkpoints to be taken at discrete intervals and mandates that patterns be identical throughout an application's execution. It is not known (and hard to prove) if under these assumptions it is possible to produce checkpoint intervals that are truly optimal for multilevel checkpointing protocols with an arbitrary number of levels. Nevertheless, these assumptions are important from a practical standpoint when considering the model's implementation in production HPC systems. SCR has been highly influential in the development of multilevel checkpointing and most multilevel checkpointing models follow these assumptions.

This model was developed as part of the SCR library and implemented as a three-level checkpointing protocol on

a BlueGene/L system. The protocol stores its lowest-level checkpoints in the node's local RAM, second level checkpoints are stored across partner nodes using XOR encoding, and last level checkpoints are stored in the PFS. The authors present the effectiveness of their model by analyzing its effect on application *efficiency*, which they define as the ratio between the minimum run time required to complete a portion of work (with no overhead from checkpointing or failures) and the expected run time to complete that same portion accounting for checkpoint and recovery overheads as predicted by the model. We use this same performance metric for the analyses we perform in this paper.

2) *The Fault Tolerance Interface* (FTI) [14] extends the three-level SCR checkpointing protocol defined in [5] with work from [15] and [16]. The FTI protocol incorporates Reed-Solomon encoding to provide an additional checkpointing level that is more reliable (and more computationally costly) than SCR's level-two XOR-encoded checkpoint between partner nodes, but less reliable than a checkpoint to the PFS, and is therefore categorized as the third checkpoint level out of four. FTI uses the scalable SCR Markov model from [5] for estimating application efficiency and determining optimal checkpoint intervals.

C. Multilevel Checkpoint Interval Optimization

Progress has been made in recent years toward optimizing checkpoint intervals in multilevel checkpointing systems. One key requirement for all techniques when determining optimal checkpoint intervals is having an accurate model of the application's execution behavior under the influence of overhead from failures and resilience.

The work that we consider by Moody et al. in [5] utilizes their Markov model to perform a brute-force search of all possible checkpoint intervals to determine the best efficiency when optimizing SCR. Because this Markov model is frequently used in other work (e.g., the FTI protocol), this same model is used for optimizing those implementations of multilevel checkpointing. While an optimal checkpoint pattern for an L-level checkpoint protocol is not guaranteed to have identical sub-patterns, optimizing a multilevel checkpoint protocol with identical sub-patterns is of practical importance to HPC system design, as mentioned earlier.

There have been two recently proposed optimization techniques in [17] and [18] that use novel approaches for determining optimal checkpoint intervals and we consider them in this work. An optimization of pattern-based multilevel checkpointing is considered by Benoit et al. in [18]. Work by Di et al. in [17] includes both pattern-based and *interval-based* optimization techniques. An interval-based multilevel checkpointing protocol allows the interval of time between checkpoints at each level to be independent of the inter-checkpoint time at other levels (unlike pattern-based protocols where higher-level checkpoints have intervals that are multiples of lower-level checkpoint intervals). Their work indicates that the interval-based optimization can perform better than pattern-based optimization. However, as noted in [18], challenges exist with practical implementations of interval-based optimization techniques that might limit their use in real systems. In particular, determining how the system should behave if checkpoints of different levels are scheduled to occur simultaneously. Furthermore, to the best of our knowledge no multilevel checkpointing protocols exist that have been designed to accommodate anything other than pattern-based multilevel checkpointing. We therefore only consider the offline pattern-based optimization technique from [17].

In our work we do not assume that checkpoint and restart events are free from failures, as is common in several other state-of-the-art models (including [17] and [18]). Indeed, as we will show in Section IV, the assumption that these events are free from failures negatively impacts the prediction accuracy of their models. Our work also considers the effect that application execution time has on interval optimization, which is not considered by the work in [5] and [18].

III. Multilevel Checkpointing Model

A. Overview

In this section, we discuss our proposed multilevel checkpointing model. We first present the set of equations used to estimate the execution time of an application employing multilevel checkpointing and we end the section with a brief description of the model's use for determining optimal checkpoint intervals.

B. Execution Time Prediction Model

Our model is a set of continuous equations that estimate the execution of an application that employs a pattern-based multilevel checkpointing protocol following the behavior of SCR described in [5]. We model the equation's prediction of application execution time hierarchically, which allows for the expected execution time of each lower level checkpoint interval (including application computation as well as all overhead associated with checkpointing and failures) to be utilized in the computation of higher level checkpoint intervals.

We define the baseline execution time of the application, T_B, as the time to execute the application without overhead from resilience or failures. In addition to T_B, the expected execution time of the application when using multilevel checkpointing, T_{ML}, is equal to the sum of the application's time spent executing each type of event associated with checkpointing and failures (each variable is an L-dimensional vector):

- successful level i checkpoints, T_{δ_i};
- level i checkpoints that have failed (failed checkpoints), $T_{\delta'_i}$;
- successful level i restarts, T_{R_i};
- level i restarts that have failed (failed restarts), $T_{R'_i}$;
- re-computation of work lost to a failure occurring during a level i computation interval, $T_{W_{\tau_i}}$;
- re-computation of work lost to a failure occurring during a level i checkpoint, $T_{W_{\delta_i}}$.

Each term's expected value is estimated as the expected number of occurrences of the event multiplied by the expected time of the event. For a chosen probability density function (PDF) used to model the probability of a failure occurring, we calculate the expected execution time for any event in which a failure has occurred as the expected value of the PDF with its domain truncated to the duration of that event and normalized to the probability of a failure occurring during the event's duration (a truncated distribution).

We assume that failures follow an exponential distribution as assumed in most prior work in the area [5] [11] [17] [18]. This makes the probability of a failure occurring during any given interval of time t for some failure rate X equal to

$$P(t, X) = 1 - e^{-Xt} \ . \tag{1}$$

In contrast to the expected value of the general PDF, which is calculated over the entire domain ($[0, \infty)$ in this case), the truncated domain is calculated over $[0, t]$ and makes the expected value of the truncated PDF for the event when using an exponential distribution equal to

$$E(t, X) = \frac{\frac{1}{X} - e^{-Xt}(\frac{1}{X} + t)}{P(t, X)} \ . \tag{2}$$

We define the failure rates associated with each checkpoint level i as λ_i. The system failure rate, λ, is equal to the sum of each λ_i and this value is also equal to the inverse of the system's MTBF. We define a failure's *severity class* to indicate the level of checkpoint required to restart the system after the failure occurs. Each failure severity class variable, S_i, indicates the probability of experiencing a failure of severity i, and is equal to the ratio of λ_i to λ. This also means that for a failure severity, $i = 1, ..., L$, the corresponding failure rate, λ_i, is the product of the system failure rate, λ, and the probability of a failure at that severity, S_i, making the failure rate $\lambda_i = S_i\lambda$.

The multilevel checkpoint protocol from [5] that we are modeling defines each higher-level checkpoint to occur after some number of occurrences of the previous level of checkpoint (e.g., an L_2 checkpoint to a partner node's RAM occurs after some number of instances of L_1 checkpoints to the node's local RAM). These values define the number of L_{i-1} checkpoints that must occur before each L_i checkpoint is taken, and are the set of $L - 1$ integer decision variables $N_1, ..., N_{L-1}$ used for optimizing the equation. Thus, the variable N_i is the number of level i checkpoints before a level $i+1$ checkpoint. The last decision variable is the computation interval, a real number that we define as τ_0. This set of decision variables defines the amount of computational progress made by the application once each level i checkpoint has been completed. The variable N_L, while not a decision variable, represents the number of level L checkpoints that will occur during the execution of the entire application and is defined based on the amount of computational progress that is made for each level L checkpoint interval, i.e.,

$$N_L = \frac{T_B}{\tau_0 \prod_{i=1}^{L-1}(N_i + 1)} \ . \tag{3}$$

An advantage to estimating total execution time hierarchically is that execution time predictions for lower level computation intervals do not need to account for the occurrence of higher severity failures when predicting the duration of each application execution event. Level i events only need to account for failures of levels less than or equal to i making the failure rate in most of the terms equal to $\sum_{j=1}^{i} \lambda_j$ and we denote this value as λ_c.

The amount of total time spent between each level $i + 1$ checkpoint (including overhead from resilience and failures) is referred to as the level i execution interval. Each higher level execution interval, τ_{i+1}, is calculated as

$$\tau_{i+1} = \tau_i(N_i+1)+T_{\delta_i}+T_{\delta'_i}+T_{R_i}+T_{R'_i}+T_{W_{\tau_i}}+T_{W_{\delta_i}} \tag{4}$$

with the application's total expected execution time when using multilevel checkpointing $T_{ML} = \tau_{L+1}$. The remainder of this section discusses how the terms in Eqn. 4 are obtained.

For each level $i+1$, the term $\tau_i(N_i+1)$ represents the total time of lower level intervals, τ_i, occurring in τ_{i+1}. We define the total number of failures during τ_i as γ_i and estimate this value using a negative binomial distribution to obtain

$$\gamma_i = \frac{P(\tau_i, \lambda_i)}{1 - P(\tau_i, \lambda_i)} \ . \tag{5}$$

Note that because lower level intervals have already accounted for lower severity failures during computation, the failure rate used to calculate both γ_i and the expected value no longer needs to be summed and becomes just λ_i. This makes $T_{W_{\tau_i}}$ equal to the expected number of failures multiplied by the expected time of those failures multiplied by the number of τ_i intervals occurring during τ_{i+1}, i.e.,

$$T_{W_{\tau_i}} = \gamma_i E(\tau_i, \lambda_i)(N_i + 1) \ . \tag{6}$$

The total time for successful checkpoints at each level is defined as

$$T_{\delta_i} = N_i\delta_i \ . \tag{7}$$

The estimator for the expected number of failures that occur during each level i checkpoint, α_i, can be modeled using a negative binomial distribution and is calculated using

$$\alpha_i = \frac{P(\delta_i, \lambda_c)N_i}{1 - P(\delta_i, \lambda_c)} \ , \tag{8}$$

so that the expected time that is wasted due to failed checkpoints is given by

$$T_{\delta'_i} = \alpha_i E(\delta_i, \lambda_c) \ . \tag{9}$$

The additional overhead associated with the execution progress that is lost due to the failed checkpoint is equal to the number of failed checkpoints at this level (α_i) multiplied by the sum of the entire failed interval plus the expected value of the overhead associated with that failed interval level multiplied by the percent of checkpoints of that level (S_k) for each level up to and including i. This is expressed as

$$T_{W_{\delta_i}} = \alpha_i \sum_{k=1}^{i} (\tau_k + \gamma_k E(\tau_k, \lambda_k))S_k \ . \tag{10}$$

The estimator for the expected number of successful restarts is calculated from the total number of level i severity failures that occur in τ_{i+1} during computation and checkpointing. We call this value β_i and it is summed using α_i and γ_i as

$$\beta_i = S_i\alpha_i + \gamma_i(S_i\alpha_i + N_i + 1) \ . \tag{11}$$

The expected number of failures that occur during restarts of level i, ζ_i, is once again modeled with a negative binomial distribution as

$$\zeta_i = \frac{P(R_i, \lambda_c)\beta_i}{1 - P(R_i, \lambda_c)} \ . \tag{12}$$

The total expected time that the application spends for successful restarts is equal to

$$T_{R_i} = \beta_i R_i \ , \tag{13}$$

and the total time that the application spends for failed restarts is equal to

$$T_{R'_i} = \zeta_i E(R_i, \lambda_c) \ . \tag{14}$$

C. Checkpoint Interval Optimization

Optimizing Eqn. 4 by selecting decision variables that minimizing execution time is accomplished by evaluating the equation's execution time at every point in a bounded region of the solution space and determining which decision variable values provide the shortest execution time. This sweep of decision variable values is bounded by the interval $(0, T_B)$ for τ_0, and also bounded such that the product of $N_1, ..., N_{L-1}$ with N_L and τ_0 is greater than zero and less than the application's baseline execution time, i.e., $0 < \tau_0 \left(\prod_{i=1}^{L} (N_i + 1) \right) \leq T_B$. We can guarantee a global optimum is found when bounding the solution space in this way because decision variable values outside of this region produce infinitely large execution times when the system's MTBF is less than the application's baseline execution time as is the case here. Efficiency is then calculated by dividing the application's baseline execution time by the calculated expected execution time.

IV. Simulation Studies

A. Overview

We performed a set of simulation studies to both validate the behavior of our equation-based multilevel checkpointing execution time prediction model and to provide a comparison of its performance with the performance of prior work in the field when executing applications at extreme scales. In addition, we also demonstrate the effects that both higher failure rates and longer checkpoint/restart times have on application efficiency and model prediction accuracy. We identify the importance of modeling failures during checkpoint and restart events in systems with high failure rates. We also show the advantage that consideration of application execution time has for the interval optimization of short-running applications.

B. HPC System Simulator

Because exascale systems do not exist, we turn to simulation to analyze the performance of each multilevel checkpointing model. The simulations performed in this section use an event-based simulator that models all events that occur throughout an application's execution in a system operating with the uncertainty of system failures. We provide a more detailed explanation of our simulator's operation in our prior work [8].

C. Performance on Prior Work Test Systems

This section provides a comparison between our technique for multilevel checkpoint interval optimization that we described in Section III and some of the techniques by others discussed in Section II-C. Specifically, in addition to our own technique we consider the work by Di et al. from [17], the work by Moody et al. from [5], the work by Benoit et al. from [18], and the classic optimization of single-level checkpointing described by Daly in [11]. We simulate the performance of each of these techniques on systems with varying characteristics that have been defined in prior work. Each unique combination is referred to as a *test system* and the details of each is outlined in Table I.

The test systems are organized in order of monotonically increasing difficulty of providing fault resilience to the system. The challenge of providing fault resilience primarily increases through either a decrease in system MTBF (due to higher failure rates) or increasing checkpoint/restart times (particularly to the PFS). However, in addition to differences in failure rates and checkpoint costs, the systems also differ in the number of checkpointing levels supported by each system as well as the distribution of failures for each failure severity class. All values in Table I are functionally identical to the information provided by each paper indicated in column two, however the values have been converted in format to allow for consistency so that all time values are now in minutes and failures for each severity are expressed as probability distributions. Checkpoint times are assumed to be equal to restart times for each system, as assumed in prior work [5] [17] [18]. In the case of Daly's traditional single-level checkpoint/restart model and the two-level checkpointing model from Di et al., when considering systems that have more available checkpointing levels than the model is able to accommodate, only the highest levels are considered (i.e., traditional checkpoint/restart only ever uses the highest level-L checkpoints to a PFS and Di et al. uses only level-L and level-$L-1$ checkpoints).

Each of our following experiments simulates the execution of a single large application that employs multilevel checkpointing (with the exception of Daly's equation which uses traditional checkpoint/restart) to mitigate the effects of system-wide failures on application execution. In all cases, the checkpoint intervals of the simulated test case are optimized using the multilevel checkpointing (or checkpoint restart) modeling technique indicated in each figure.

Figure 2 shows the performance (efficiency) of the five checkpointing techniques considered (our technique is shown in green) for each of the test systems described in Table I. Bars in the figure represent the average of 200 simulation trials with random failures for each specific setup, with the standard deviations of those trials shown around each bar. The diamonds that are color-coded to each bar in the figure indicate the prediction of the system's efficiency by each technique. Predictions are made based on the system's execution characteristics and the checkpointing intervals determined by each technique with accurate predictions being those located closer to the tops of each bar.

The first trend highlighted in Figure 2 is the improved efficiency that a multilevel checkpointing approach can have over traditional checkpoint/restart (Daly). The figure shows how even though Daly's equations for traditional checkpoint/restart are highly accurate at predicting application efficiency (and consequently good at selecting appropriate checkpoint intervals) the traditional checkpoint/restart protocol does not perform as well as the multilevel checkpointing protocol when optimized by either Dauwe et al., Di et al., or Moody et al. Daly's checkpoint/restart's efficiency is 50% less than that of multilevel checkpointing in the worst case. In addition

TABLE I: Test Systems Examined in Prior Work

test system	*paper* (system name)	*num. C/R levels*	*MTBF (minutes)*	*failure distribution (probability per level)*	*checkpoint/restart time (minutes per level)*	*baseline execution time (minutes)*
M	[5](BlueGene/L Coastal)	3	6944.45	$(0.083, 0.75, 0.167)$	$(0.008, 0.075, 17.53)$	1440.0
B	[19] (BlueGene/Q Mira)	4	333.33	$(0.556, 0.278, 0.139, 0.027)$	$(0.167, 0.5, 0.833, 2.5)$	1440.0
D1	[17] (ANL Fusion case 1)	2	51.42	$(0.857, 0.143)$	$(0.333, 0.833)$	1440.0
D2	[17] (ANL Fusion case 2)	2	24.0	$(0.833, 0.167)$	$(0.333, 0.833)$	1440.0
D3	[17] (ANL Fusion case 4)	2	12.0	$(0.833, 0.167)$	$(0.167, 0.667)$	1440.0
D4	[17] (ANL Fusion case 5)	2	6.0	$(0.833, 0.167)$	$(0.167, 0.667)$	1440.0
D5	[17] (ANL Fusion case 3)	2	12.0	$(0.833, 0.167)$	$(0.333, 1.67)$	1440.0
D6	[17] (ANL Fusion case 6)	2	6.0	$(0.833, 0.167)$	$(0.167, 1.67)$	720.0
D7	[17] (ANL Fusion case 7)	2	4.0	$(0.833, 0.167)$	$(0.667, 3.33)$	360.0
D8	[17] (ANL Fusion case 8)	2	3.13	$(0.870, 0.130)$	$(0.833, 5.0)$	360.0
D9	[17] (ANL Fusion case 9)	2	3.13	$(0.870, 0.130)$	$(0.833, 5.0)$	180.0

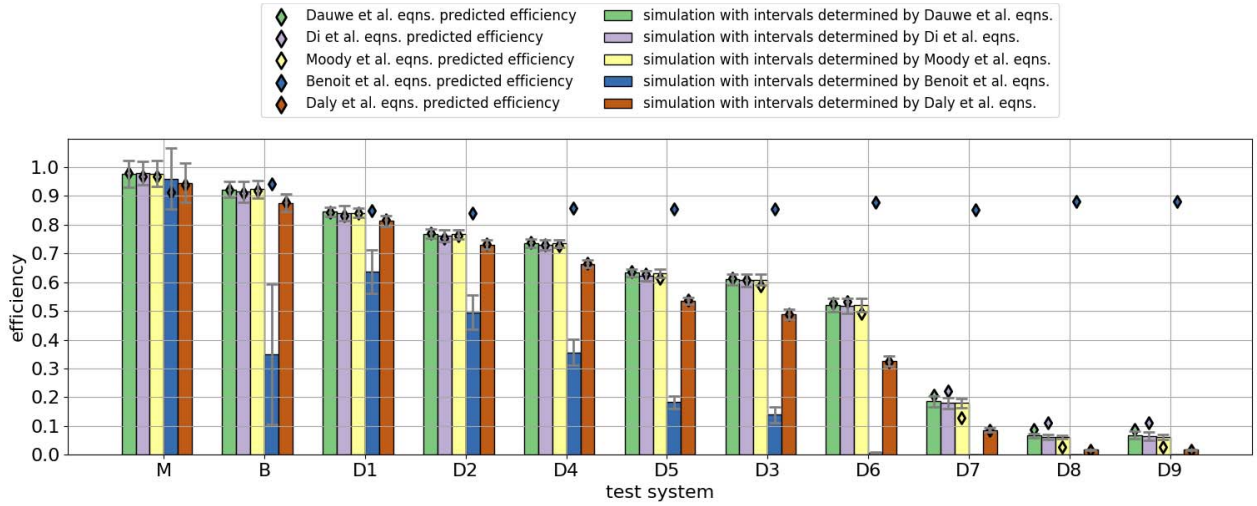

Fig. 2: Performance of the *multilevel checkpoint* and *traditional checkpoint/restart* checkpoint interval optimization techniques executing on the test systems from Table I. Bars in the figure indicate the average of 200 simulation trials with randomly occurring failures. Diamonds in the figure indicate each technique's prediction of the simulated performance. Standard deviations are shown for each bar.

to reaffirming the conclusions from prior work, the data for traditional checkpoint/restart in Figure 2 also helps to highlight the importance of having an accurate multilevel checkpointing model that is capable of making an appropriate selection of checkpoint intervals. In particular, while the multilevel checkpoint technique by Benoit et al. performs well on test system M, because some of the approximations made by the model have large effects on prediction accuracy, its efficiency on more challenging test systems is worse than for a well-optimized implementation of traditional checkpointing.

The efficiency predictions of the equations modeled by Benoit et al. in [18] are optimistic because they do not consider the effect of failures during checkpoints or restarts and only consider failures during computation. Consequently, the corresponding computation intervals determined by these equations are excessively long. For all test systems, the length of the computation interval chosen by these techniques is at least $2.5\times$ greater than that of the other multilevel checkpointing techniques. This disparity also increases as the challenge of providing resilience to the system increases and can be seen in Benoit et al.'s model's faster decrease in efficiency in comparison to the other techniques displayed in Figure 2. At the same time, the execution time prediction model results (blue diamonds for Benoit et al.) indicates that the chosen intervals have optimistically low execution time predictions resulting in optimistically high efficiency predictions. The sharp drop in efficiency of Benoit et al.'s equations on test system B is due to the decreasing accuracy of their equations as the number of checkpoint levels increase. While the decrease in performance of the other multilevel checkpointing techniques is monotonic and follows the increase in difficulty of providing resilience to each system, the Benoit equations drop sharply from system M (with three checkpoint levels) to system B (with four checkpoint levels) and subsequently increase in efficiency in system D1 (with two checkpoint levels).

Simulated performance of multilevel checkpointing when optimized by either Dauwe et al. (our work), Di et al., or Moody et al. is similar across all test systems, however the prediction accuracy for each of these techniques can be seen to decrease slightly as test system difficulty increases. The causes of this will be discussed in detail in the upcoming sections.

D. Failures During Checkpoints and Restarts

Figure 3 shows the breakdown of how application time is spent when executing an application in a failure-prone environment and employing a resilience protocol to mitigate

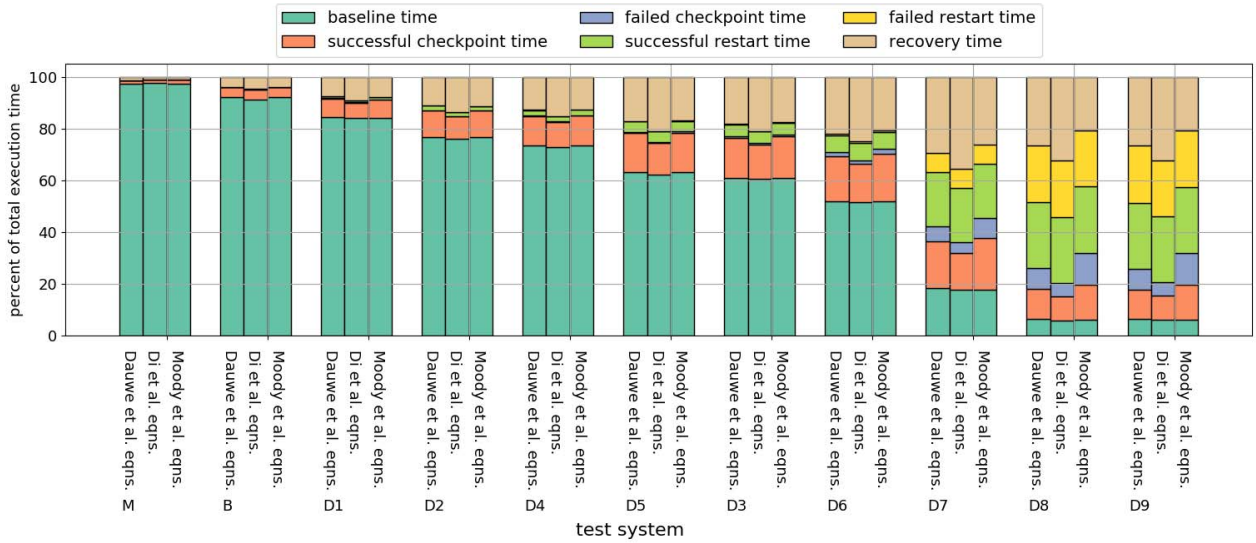

Fig. 3: Percentage of application execution spent on baseline execution of the application as well as all resilience and failure event related overhead during the application's execution. Each test scenario shown represents the average of 200 trials with randomly occurring failures.

the effects of failures. Each test scenario shown represents the average of 200 trials with randomly occurring failures. The figure shows the percentage of time spent on baseline execution time as well as time lost to overhead from each of the resilience and failure-related events discussed in Section III. Data is shown for the same test systems presented in Table I, but we limit our analysis to the three best-performing techniques from Figure 2.

It is evident from the data in Figure 3 that as the difficulty in providing resilience to systems increases, applications lose increasing amounts of time to failed checkpoints and restarts with at least 30% of application time spent in these areas in the most extreme cases. Furthermore, this increase is non-linear (in fact the increase follows the α_i and ζ_i variables of Eqns. 8 and 12) and affects the more extreme *D7, D8,* and *D9* systems to a greater degree than the other systems shown in the figure. Results for test systems *D8* and *D9* look almost identical because these test systems are identical in all respects except their baseline execution time.

The rapid increase in the number of failed checkpoint and restart events is caused by the system MTBF approaching (or even becoming less than) the length of time required to checkpoint or restart to the PFS. While optimal intervals between checkpoints can be adjusted to compensate for decreasing MTBF, checkpoint and restart times cannot, and this can force the system to retry checkpoint and restart events several times before they can complete successfully. Because extreme-scale systems experience increased amounts of failures during checkpoint and restart events, consideration of these events is necessary for accurate execution time modeling.

E. Performance at Extreme-Scale System Difficulty

For the studies discussed in this section we analyze the four-level checkpointing system from [19] (the system defined in Table I from Section IV-C as system B) under a variety of exascale-like execution scenarios. Specifically, we scale both system MTBF and the length of time required by the system for checkpointing to or restarting from the PFS. It has been noted in [5] that exascale systems are likely to experience failures with an MTBF between $3-26$ minutes and therefore we explore five system MTBF values in this range.

Data from [4] suggests that the improvements to network speed will increase at a similar rate as the data required to checkpoint larger applications and consequently checkpoint times to a PFS will likely remain constant as system sizes increase. From [4], we assume that checkpoint times to a PFS for an exascale-sized application will likely be between $20-40$ minutes. We examine four values for the time costs associated with checkpointing and restarting to the PFS. These values range from 10 minutes, likely to be a conservative estimate, to 40 minutes. We only consider scaling of the level L checkpoint/restart time because (as noted in Section III) checkpoint/restart levels less than L spread checkpoint data across system resources. Lower level checkpoints are therefore less affected by application size. Checkpoint and restart times for lower level checkpoints remain the same as those values listed for test system B in Table I.

The results of this study are shown in Figure 4. The difficulty of providing resilience to each test scenario increases both across each x-axis (as system MTBF decreases) and across sections (a)-(d) of the figure (as the time penalty for checkpoints/restarts to a PFS increases). Bars in the figure indicate the average of 200 simulation trials with randomly occurring failures. Diamonds in the figure indicate each technique's prediction of the simulated performance. Standard deviations are shown for each bar. It is evident that multilevel checkpointing's ability to provide resilience to an exascale HPC system will be more impacted by system MTBF than increased checkpoint/restart times. Decreasing system MTBF from 26 minutes to 3 minutes can decrease efficiency from over 60% to less than 1% in some cases but increasing checkpoint/restart times from 10 minutes to 40 minutes produces a maximum decrease of about 40% efficiency. It is also clear that some of these possible exascale applications push the limit of the resilience that multilevel checkpointing can provide. The

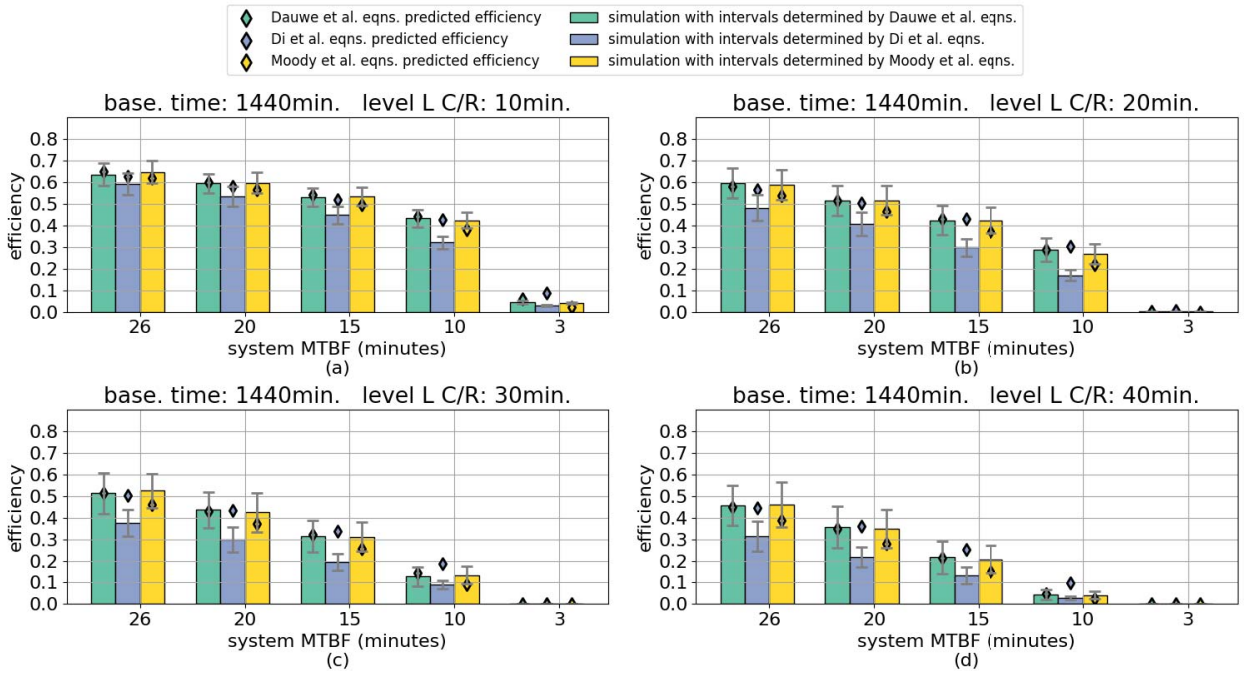

Fig. 4: The execution of a 1440 minute application under a variety of execution scenarios with level-L checkpoint and restart times of **(a)** 10 minutes, **(b)** 20 minutes, **(c)** 30 minutes and **(d)** 40 minutes. Bars in the figure indicate the average of 200 simulation trials with randomly occurring failures. Diamonds in the figure indicate each technique's prediction of the simulated performance. Standard deviations are shown for each bar.

most extreme case of a 3 minute MTBF produces less than 1% efficiency for checkpoint/restart lengths greater than 10 minutes. Even a system with a 15 minute MTBF produces less than 50% efficiency for checkpoint/restart lengths greater than 10 minutes.

The results in Figure 4 also clearly show the negative effect of Di et al.'s constraint of only considering two checkpoint levels. For all execution scenarios that produce more than 1% efficiency, the performance of checkpoint interval optimizations from Di et al. is noticeably worse than that of Dauwe et al. and Moody et al., which utilize all four checkpoint levels. It should be noted that the poorer performance of Di et al.'s equations is primarily due to its constraint of two checkpoint levels. We deduce this by noting that the simulated performance of all three techniques are close to equal for the two-level checkpointing test systems from Figure 2 indicating that the decreased performance of Di et al.'s equations in Figure 4 is caused by its restriction to only using two checkpoint levels.

Figure 4 also indicates that the prediction accuracy of all optimization techniques decrease with both decreasing system MTBF and increasing checkpoint/restart times. Prediction accuracy of each technique is further discussed in Section IV-G.

F. Consideration of Application Execution Time

One advantage that both our equations and those of Di et al. have over those of Moody et al. is our consideration of application execution time. Because even for the most pessimistic MTBF values the highest severity system failures are still infrequent, under execution scenarios with high level-L checkpoint/restart times it is more efficient (on average) for shorter applications not to take time-consuming level-L checkpoints and instead risk a total application restart. As our equations calculate the expected execution time, this effect is identified by our equations and those of Di et al. Consequently, when selecting checkpoint intervals for those scenarios our equations (Dauwe et al.) and those of Di et al. correctly select intervals that are optimized not to include level-L checkpoints, while the equations from Moody et al. select interval values that are appropriate only for longer running applications.

We demonstrate this in Figure 5, which shows the same set of execution scenarios discussed in Figures 4a and 4b but with a shorter application that executes for only 30 minutes. Bars in the figure now indicate the average of 400 simulation trials with randomly occurring failures. Diamonds in the figure indicate each technique's prediction of the simulated performance, with standard deviations shown for each bar. Because the application's execution time is less than the mean time between level-L severity failures, our equations and those of Di et al. do not take level-L checkpoints in any of the experiments in Figure 5. Here we have shown the execution of a 30 minute application as an extreme example but we have found these same results occur to a lesser extent for an application that is 120 minutes in length. We expect that this result is present for all extreme-size applications that have a baseline execution time that is shorter than the mean time between the highest severity failures. Benefits to short applications from not including level L checkpoints increase with both the increase in checkpoint/restart lengths and decreasing MTBF values, and provide as much as a 20% efficiency improvement in some cases. While the advantage gained in other cases may be lower, we have determined with 95% confidence that all improvements in the figure are statistically significant.

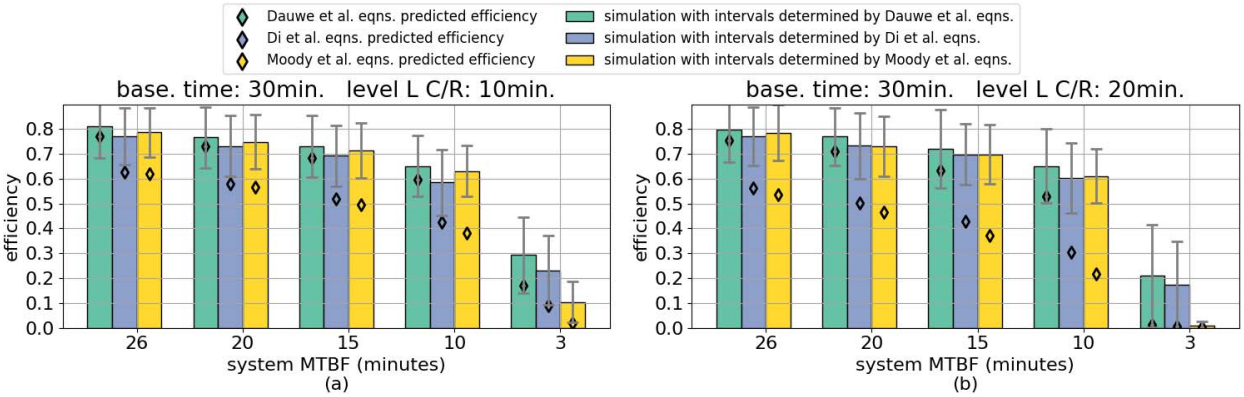

Fig. 5: The execution of a 30 minute application under a variety of execution scenarios with level-L checkpoint and restart times of **(a)** 10 minutes and **(b)** 20 minutes. Bars in the figure indicate the average of 400 trials with randomly occurring failures. Diamonds in the figure indicate each technique's prediction of the simulated performance. Standard deviations are shown for each bar.

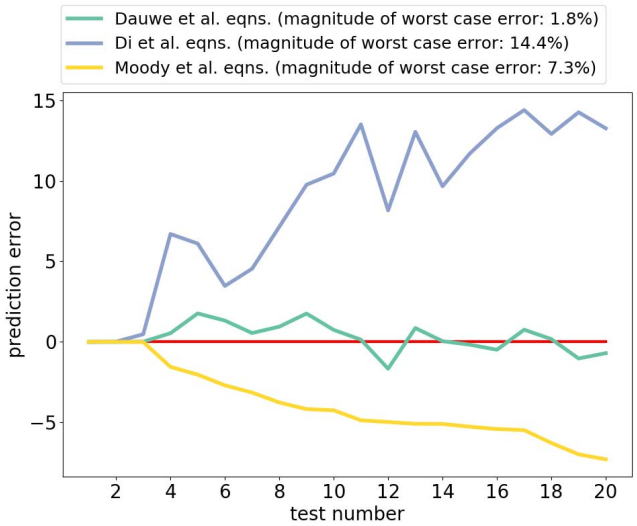

Fig. 6: Prediction error for the 20 application execution scenarios shown in Figure 4. The prediction error shown is simply the difference between each multilevel checkpoint model's prediction of efficiency and the corresponding efficiency determined through simulation. The tests are ordered by increasing magnitude of error of the Moody et al. model. The red line indicates a value of zero (the target error).

Because this effect is only beneficial on average, one difference between the results in Figure 5 and those of the longer application in Figure 4 can be seen when comparing standard deviations between techniques. While standard deviations are nearly identical between techniques for every execution scenario tested in Figure 4, the standard deviations shown in Figure 5 for the execution scenarios that have had their level-L checkpoints excluded by our equations now have a slightly greater variation in execution time than the results for the equations from Moody et al. that still perform a level-L checkpoint.

G. Model Prediction Accuracy

Figure 6 shows the prediction accuracy of the 20 system scenarios shown in Figure 4 in terms of each model's prediction of application efficiency minus the efficiency value determined through simulation. The system scenarios in the figure have been sorted according to increasing magnitude of error of the results for Moody et al. and show each optimization technique's deviation from the ideal error of zero. Here, we have only shown results for the long duration applications discussed in Section IV-E because they provide the clearest depiction of model prediction performance. However, for shorter applications, our equations still has the best prediction accuracy in most cases.

The results in Figure 6 demonstrate the benefit that our multilevel checkpointing model provides in terms of prediction accuracy over both Di et al.'s and Moody et al.'s models. As the test numbers on the x-axis increase, the difficulty in prediction also increases and Moody et al.'s model tends to underestimate application efficiency (by as much as 7.3%) while Di et al.'s model tends to overestimate application efficiency (by as much as 14.4%).

The difference in prediction accuracy between our work and that of Moody et al. [5] and Di et al. [17] are the assumptions made by each set of equations about the system's behavior during failed restarts. Our equations assume that if the system is restarting from a class i severity failure and experiences a second failure of severity less than or equal to i then the system can still be restarted from a subsequent level i checkpoint. The simulations make this assumption for all techniques.

Di overestimates efficiency because it neglects considering the effects of failures during restarts entirely. Specifically, it does not account for the increasing impact of repeated failed restarts discussed in Section IV-D that occur more frequently with both decreased MTBF and increased checkpoint/restart times. Di et al. are aware of this limitation in their model, and made a note of its effect on prediction accuracy in [17].

The model by Moody et al. underestimates efficiency because of its pessimistic assumption about failures during restarts. Specifically, if the system is restarting from a level-i severity failure and experiences a second failure of level-i severity then the system needs to subsequently restart from a level $i+1$ stored checkpoint. This causes an unrealistic escalation of failure levels for extreme-sized systems that experience significant lower severity failures. For example, if one of the 100,000 nodes in the test system B considered here required a restart from local RAM it is unreasonable to assume that a second event of that type occurring on any

other node in the system would necessitate any response other than attempting to load the same checkpoint from RAM a second time. This assumption causing escalating failures in conjunction with the presence of rapidly increasing numbers of failures (as discussed in Section IV-D) causes Moody et al.'s model to have increased prediction error. Although the effect on system behavior implied by this model assumption would have been present in several of the more extreme test cases that Moody et al. explore with their Markov model in [5], they do not perform any simulations of their model demonstrating the necessity of this assumption in an actual HPC system nor do they discuss this effect in their results.

V. Conclusions

With this work we have developed a hierarchical continuous equation-based model for a multilevel checkpointing resilience technique operating with an arbitrary number of checkpoint levels. Through simulation of exascale HPC systems, we have shown the model's ability to accurately predict application execution time in failure-prone environments as well as for determining optimal checkpoint intervals. We have implemented several multilevel checkpointing optimization techniques from the literature and shown the benefit that our technique can provide over these techniques in both model prediction accuracy and determining checkpoint intervals for short duration applications. In all circumstances, our model either outperforms the models from others' work or it is capable of performing within 1% of their model.

Using our simulation data we have shown that extreme-scale systems experience increasing numbers of failures during checkpoint and restart events. We determined that the increase is caused by decreasing MTBF values approaching the values of increasing checkpoint/restart length times. The increased probability of failure during checkpoint/restart events causes an extremely rapid (non-linear) increase in the amount of time lost to these events at extreme scales, and makes the consideration of these events a necessity for accurate execution time modeling - something that is frequently ignored.

We have also more generally shown some of the limitations of multilevel checkpointing. Simulations of single level checkpoint/restart (using Daly's equation) show that the usefulness of a single-level resilience protocol is limited to petascale-sized systems with MTBF on the order of hours, and indicate that larger systems require more advanced resilience protocols such as multilevel checkpointing. Similarly, our data also suggests the limits of multilevel checkpointing. When utilizing multilevel checkpointing, a system with even a 15 minute MTBF will drop below 50% efficiency for checkpoint/restart lengths greater than 10 minutes. In such cases, regardless of the checkpoint interval optimization technique used, the system will spend less than half its time on useful computation. Given that operating exascale systems are likely to cost tens of millions of dollars a year, this level of resilience is likely to be unacceptable. As system sizes increase further, other strategies may need to be employed to complement (or possibly replace) the multilevel checkpointing protocol.

Acknowledgments

This work was supported by the NSF under grants CCF-1252500 and CCF-1302693. The authors thank Hewlett Packard (HP) of Fort Collins for providing us some of the machines used for testing.

References

[1] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, pp. 1302–1326, Sep. 2013.

[2] F. Cappello, A. Geist, B. Gropp, L. Kalé, B. Kramer, and M. Snir, "Toward exascale resilience," *Int'l Journal of HPC Applications*, vol. 23, no. 4, pp. 374–388, 2009.

[3] C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer, "Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 HPC application runs," in *IEEE Int'l Conf. on Dependable Systems and Networks*, pp. 25–36, June 2015.

[4] F. Cappello, "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," *Int'l Journal of HPC Applications*, vol. 23, pp. 212–226, Aug. 2009.

[5] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Int'l Conf. for HPC, Networking, Storage and Analysis*, 11 pp., Nov. 2010.

[6] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomputing Frontiers and Innovations*, vol. 1, pp. 5–28, June 2014.

[7] E. Meneses, X. Ni, G. Zheng, C. Mendes, and L. Kalé, "Using migratable objects to enhance fault tolerance schemes in supercomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 26, July 2015.

[8] D. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "A performance and energy comparison of fault tolerance techniques for exascale computing systems," in *The 6th IEEE Int'l Symp. on Cloud and Service Computing*, pp. 436–443, Dec. 2016.

[9] D. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "An analysis of resilience techniques for exascale computing platforms," in *19th Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*, in the proceedings of 2017 Int'l Parallel and Distributed Processing Symp. Workshops (IPDPSW 2017), pp. 914-923, May 2017.

[10] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Comm. of the ACM*, vol. 17, pp. 530–531, Sep. 1974.

[11] J. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, pp. 303–312, Feb. 2006.

[12] E. Gelenbe and D. Derochette, "Performance of rollback recovery systems under intermittent failures," *Comm. of the ACM*, vol. 21, pp. 493–499, June 1978.

[13] N. H. Vaidya, "A case for two-level distributed recovery schemes," *SIGMETRICS*, vol. 23, pp. 64–73, May 1995.

[14] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, "FTI: High performance fault tolerance interface for hybrid systems," in *Int'l Conf. for HPC, Networking, Storage and Analysis*, pp. 32:1–32:12, Nov. 2011.

[15] L. A. Bautista-Gomez, N. Maruyama, F. Cappello, and S. Matsuoka, "Distributed diskless checkpoint for large scale systems," in *Int'l Conf. on Cluster, Cloud and Grid Computing*, pp. 63–72, May 2010.

[16] L. B. Gomez, A. Nukada, N. Maruyama, F. Cappello, and S. Matsuoka, "Low-overhead diskless checkpoint for hybrid computing systems," in *2010 Int'l Conf. on High Performance Computing*, Dec. pp. 10, 2010.

[17] S. Di, Y. Robert, F. Vivien, and F. Cappello, "Toward an optimal online checkpoint solution under a two-level HPC checkpoint model," *IEEE Trans. Parallel and Distributed Systems*, vol. 28, pp. 244–259, Jan. 2017.

[18] A. Benoit, A. Cavelan, V. L. Fvre, Y. Robert, and H. Sun, "Towards optimal multi-level checkpointing," *IEEE Trans. Computers*, vol. 66, pp. 1212–1226, July 2017.

[19] P. Balaprakash, L. A. Bautista-Gomez, M.-S. Bouguerra, S. M. Wild, F. Cappello, and P. D. Hovland, "Analysis of the tradeoffs between energy and run time for multilevel checkpointing," in *5th Int'l. Workshop on High Performance Computing Systems*, pp. 249–263, Nov. 2015.