Optimizing Checkpoint Intervals for Reduced Energy Use in Exascale Systems

Daniel Dauwe*, Rohan Jhaveri*, 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*—In today's high performance computing (HPC) systems, the probability of applications experiencing failures has increased significantly with the increase in the number of system nodes. It is expected that exascale-sized systems are likely to operate with mean time between failures (MTBF) of as little as a few minutes, causing frequent interrupts in application execution as well as substantially greater energy costs in a system that will already consume large amounts of energy. State-of-the-art HPC resilience techniques proposed for use in these future systems complicate the energy problem further as the overhead associated with utilizing these techniques also further increases energy use. While work has been done that attempts to analyze and improve the energy use of systems utilizing resilience techniques, our work offers a new approach through the optimization of checkpoint interval lengths that allows a system designer the freedom to choose between intervals that optimize for application performance efficiency or energy use in both a traditional checkpoint and multilevel checkpoint approach to resilience. We create a set of equations able to optimize for either performance efficiency or energy use, demonstrate that distinct intervals exist when optimizing for either one metric or the other, and examine the sensitivity of this phenomena to changes in several system parameters and application characteristics.

Keywords: exascale resilience; checkpoint restart; multilevel checkpointing; fault tolerance; HPC energy efficiency;

I. Introduction

With the number of CPU cores in large-scale computing systems increasing exponentially over time, the failure rates in these systems have increased exponentially as well. It is expected that the next generation of high performance computing (HPC) machines will experience failures up to several times an hour, making the need for effective fault resilience important for building tomorrows's HPC systems [1].

Another important consideration for the development of extreme-scale HPC systems is the increasingly high energy costs associated with using the system once it is operational. Today's largest HPC system consumes approximately 15.371 MW at full capacity (not including the costs for cooling) [2]. Assuming even very low electricity costs of $0.06 per kWh [3] today's systems cost over $8M per year to operate. Based on the conservative exascale system assumptions we outline in Section III, extrapolating these costs to exascale indicates that system operation alone will cost a minimum of $47M per year. Reducing an exascale system's energy requirements has been credited as being one of the most important and challenging roadblocks associated with developing such a system [4]. While some work has been done to explore the role that fault resilience plays in the energy use of large scale systems, little work has been performed that specifically attempts to reduce the impact that resilience techniques have on system energy use.

Performance efficiency is defined to be the ratio of an application's baseline execution time over the application's execution time with slowdowns from failures or resilience technique overheads (e.g., delays when taking a checkpoint). There is a critical need to allow a system designer the opportunity to trade-off application performance efficiency (corresponding to minimizing application execution time) with minimizing system energy use in extreme-scale systems, to optimally balance performance goals with energy overheads.

Because the costs of operating an exascale system are very high, even for scientific applications with an execution time as short as one day without overheads from failure and resilience related delays, optimizing checkpoint intervals for energy use at the cost of a slight increase in application execution time would likely allow for over a million dollars of energy savings by the end of the application's execution. Our results demonstrate that for a few percent reduction in performance, the energy required to execution an application can be reduced by as much as 10%.

In summary, this work makes the following novel contributions:

- we develop a set of execution time and energy use prediction equations that can be utilized to determine optimial checkpoint intervals for both traditional checkpointing and multilevel checkpointing-based HPC fault resilience techniques;
- we provide a methodology for simulating the execution of applications operating at exascale system sizes in the presence of uncertainty due to failures;
- we use our methodology to model an exascale computing environment and utilize this environment to simulate the execution and energy use of both a traditional checkpointing technique as well as a multilevel checkpointing technique proposed for future exascale-sized HPC systems;
- we simulate each resilience technique's performance and energy use when optimizing checkpoint intervals for either application performance efficiency or lower energy use and demonstrate that a trade-off exists between optimizing for either metric;
- we perform a sensitivity analysis of the parameters associated with this trade-off.

The remainder of this paper is organized as follows. Section II discusses the resilience techniques we examine and discusses other work that has considered exascale resilience and energy use. In Section III, we describe the modeling methodology we use for our system simulator. Our implementation of HPC resilience is presented in Section IV. Section V details the equations we derive and use for determining optimal checkpoint intervals. Section VI describes the simulated studies we perform to demonstrate the trade-off between performance efficiency and energy use and provides a sensitivity analysis on these results. We conclude with a summary of this work in Section VII.

II. Related Work

The prior work we discuss here focuses on system-level checkpointing-based HPC resilience that allows application programmers and users of the system to be oblivious to the strategies for HPC resilience that are being employed on their behalf. All checkpointing-based techniques rely on the notion of periodically saving the system's executing state and restarting from an earlier error-free state after the occurrence of a system failure [5] [6]. We focus on two of the most popular checkpointing-based techniques, a traditional checkpoint/restart based technique and the more recently proposed multilevel checkpointing technique.

Traditional checkpoint/restart is by far the most commonly used resilience technique employed by today's large-scale HPC systems. However, the length of time associated with checkpointing, restarting, and recomputing work lost to a system failure for a large-scale application can be quite large. Moreover, the frequency at which the system needs to take checkpoints for very large-scale applications when implementing traditional checkpointing techniques has been shown to significantly degrade performance with increasing system sizes [7]. Traditional checkpointing alone is not expected to be capable of providing satisfactory resilience to exascale-sized systems.

Because different types of failures can affect a computing system by different amounts, not all failures require restarting the system from a checkpoint to the parallel file system [8]. Multilevel checkpointing exploits this by providing the system with several levels of checkpointing. A system employing a multilevel checkpointing scheme may allow for levels that trade-off between checkpoints to RAM (that are faster but able to recover from fewer types of failures), to checkpoints saved to a partner node's RAM (that are less frequent and slower, but able to recover from more types of failures), to checkpoints saved to the system's parallel file system (that are the slowest but allow recovery from almost all failures). Each level offers a trade-off between the time required by the system to checkpoint or restart, and the level of failure severity from which the checkpoint can recover [9].

One challenge associated with using a multilevel checkpointing technique is in determining the optimal number of checkpointing levels to support in the system, and the optimal computation intervals between checkpoints at each level. Various solutions to this problem have been proposed [9] [10] [11] [12]. We also provide our own solution to the problem of determining optimal multilevel checkpoint intervals that we describe in Section V.

Assessing the energy use associated with fault resilience techniques has been considered in several works [13] [14] [15] [16] [17]. However, none of these works specifically optimize checkpoint intervals to attempt to minimize energy use as our work does.

The work in [18] does specifically attempt to minimize system energy use for the checkpoint/restart resilience technique. However, the authors achieve this through power capping and not through checkpoint interval optimization as our work does. The authors in [19] do specifically reduce energy use in a checkpoint restart-based technique by optimizing checkpoint intervals, but their work is primarily focused on checkpointing in mobile devices that store checkpoints over the internet and is of limited use for the exascale-size HPC systems that we consider in our work. Both of these works also do not consider energy reduction for multilevel checkpointing.

An approach to predict checkpoint/restart execution time and energy use was explored in [7] and [15]. We have greatly extended this prior work to allow for much more accurate predictions and further extended the initial ideas to allow for modeling multilevel checkpointing with an arbitrary number of checkpoint levels.

III. Exascale Modeling Methodology

A. Overview

We have designed an event-based simulator used for modeling HPC systems of arbitrary size [17] [20] [21]. The system experiences randomly generated failures that affect the simulated execution of applications in the system. Throughout the system's simulation an application's execution is affected by events associated with:

- *computation*: execution toward the application's completion,
- *failures*: the simulated failure of a system node,
- *checkpoints*: saving a backup of computation progress,
- *restarts*: restoring the application progress saved in the last system checkpoint after a failure occurs,
- *recovery*: recomputing progress lost to a failure,
- *failed checkpoints or restarts*: behavior of the system when failures occur during checkpoint or restart events.

Checkpoints, restarts, and recovery are all resilience-technique specific events that determine how an application behaves in a system with failures. This is discussed in detail in Section IV. The remaining events associated with computation and failures are all attributes of the system and behave the same regardless of the resilience technique being used by the system. In particular, while failure events have a large impact on the behavior of the resilience-technique related events, failure events themselves are a function of the size of the system and the reliability of the system's nodes, and are not affected by the resilience technique employed by the system.

B. Applications Model

We consider synthetic applications that exhibit weak scaling so that as the number of nodes used by the application increases with application size, the application's attributes of computation and memory used per node remain constant. For all simulated studies performed here, applications are defined to execute for a full day of execution time when executed without delays from failures or events related to resilience (such as time spent checkpointing). This delay-free execution time is the application's *baseline execution time* and is represented by the variable T_B.

Memory needed for the application is represented by the variable N_m. Most of the simulations performed in this work have a value of N_m = 32GB of memory required per node. Details about the size and memory use of applications in each simulated study are discussed further in Sections VI.

C. Simulated System Setup

The simulated exascale system is a homogeneous system inspired by the architecture used in China's Sunway TaihuLight 125 petaflop supercomputer [2], the world's highest performing system since June 2016 and still the world's top system as of June 2017 [22]. Each Sunway TaihuLight system node has a multicore architecture composed of four clusters of 64 computational processing elements (CPEs) with each cluster managed by a single management processing element (MPE) that also performs computational work allowing for a total of 65 cores of computation in each core cluster. The four core clusters in a system node provide a total of about 3.1 TFLOPs over 260 cores. As systems trend towards manycore architectures, with hundreds or thousands of CPU cores on a single socket, component failure rates are likely to increase [23] [24]. Our simulated exascale system assumes an approximate $4\times$ increase in the number of CPU cores per processor over the Sunway TaihuLight system by the time an exascale machine is developed, allowing for a total of 1028 cores per node providing approximately 12 TFLOPs of compute power for each system node. A system composed of 120,000 of these high performing nodes would perform at an exascale level.

The Sunway TaihuLight system has 8 GB of DDR3 RAM at each of its four core clusters, giving each node a total of 32 GB of RAM. We again assume that future systems are likely to have memory increases of about a factor of four in comparison to today's systems giving our simulated system a total memory capacity of 128GB per system node, enough to allow for the multilevel checkpointing resilience technique to operate with a working memory size of up to 40GB and still be able to accommodate the additional memory required for lower-level checkpoints (discussed in Sections III-D and IV). In addition to an increase in volume, we also assume that future memory is likely to utilize newer architectures, allowing for increased aggregate memory bandwidth, B_M. Today's best memories perform at a rate of up to 25 GB/s [25], we conservatively estimate that future memories will be able to perform at about $B_M = 40$ GB/s.

Reports on the Sunway system [2] [26] mention that a system node requires around 375 watts to operate during computation. We assume that the total power required for computation will increase because of the large increase in the number of cores per system node, however, it can also be expected that the energy efficiency of future processors will improve. Given that we are assuming a 4× increase in the number of cores per system node in our simulated system, we conservatively assume the power required for computation work, P_W, will double from the power requirements of the Sunway system.

Because the system halts all computation during checkpoints and restarts, the power required by the system during these events is much lower than the power required for computation. Based on power measurements we have made on several server-class Xeon processor based servers, we assume that the power required during a checkpoint or restart, P_δ, will be equal to the node's idle power (we assume this to be 150 watts) plus the power required for network communication (discussed in Section III-E).

D. System Failure Model

Failures are characterized by two attributes: the time that the failure occurs and the "severity class" of the failure. The uncertainty associated with each attribute is modeled using random variables. We assume independence between individual failure occurrences as well as between both the attributes of each failure.

A common assumption when modeling failures in an HPC system is that failures follow an exponential distribution and can be modeled by a Poisson process [27]. Every failure occurs according to the previous failure's arrival time ($T_{F_{i-1}}$, with $T_{F_0} = 0$) plus a random variate generated from an exponential distribution $\mathcal{T}_i \sim Exp(\lambda)$ with an expected arrival rate of $E[\mathcal{T}_i] = \frac{1}{\lambda}$. The parameter λ indicates the average failure rate of the application, and is defined as the number of nodes required by the application, N_A, divided by the mean time between failures (MTBF) of the system nodes, M_n, i.e.,

$$\lambda = \frac{N_A}{M_n} \ . \tag{1}$$

The severity class of failure corresponds to the type of failure that has occurred in the system. This attribute is used by multilevel checkpointing to determine which level of saved checkpoint is necessary to enable the system to recover from a specific type of failure and is also used when determining the optimal duration of intervals between checkpoints of different levels. These assumptions and the effect of a failure's class on multilevel checkpointing's behavior is discussed further in Section IV.

The mapping of failure types to failure severity classes is based on the analyses of types of failures present in modern day HPC systems presented in [28] and [29]. We define the probability of experiencing a class i severity failure according to the ratio of the number of failures that occur at each failure severity class, to the total number of failures, measured over an extended interval of time. The resulting discrete set of ratios for the set of classes is used to create a probability mass function from which random variates are sampled to define the severity attribute of each failure. We denote the probability of each of the L severity classes as $S_1, ..., S_L$. We assume that types of failures in a future exascale-sized system will occur in similar relative amounts to those experienced by today's system, but with the total number of failures occurring more frequently.

For this work we define three severity classes. The first class (S_1) assumes that failures can be recovered from using a checkpoint stored in a node's local RAM, the second class (S_2) requires the restarting application from a checkpoint stored in a partner node's RAM, and the third and highest class (S_3) requires the system to checkpoint and restart from a parallel file system. The data presented in [28] and [29] details the frequencies of thirty-three types of failures in the Blue Waters system. Using this information we determined the probability of failure severities for our three-class failure model to be approximately $S_1 = 0.138$, $S_2 = 0.784$, and $S_3 = 0.078$.

E. Communication Model

System communication plays a key role in the performance of checkpoints written to both a partner node and the parallel file system. We account for the effects of communication on application checkpoints taken in the system. We assume that future exascale systems are likely to have improved communication over today's systems, and base the communication model for the studies performed here on the "NDR InfiniBand" network described in [30]. For most simulations in Section VI, our communication network assumes a latency value of $L_N = 0.5\mu$s, a bandwidth value of $B_N = 600$GB/s, and a maximum number of simultaneous connections at each switch $N_S = 12$. The effects of these values on the corresponding time required to checkpoint the system is discussed further in Section IV.

The power used for system communication is calculated for each system node. Given a switch power of $P_S = 200$ watts and a network interface controller of a single node that consumes $NIC = 15$ watts at full utilization [31], the power spent by a single node for communication during a time of high network traffic (such as during a checkpoint or a restart), P_N, and defined as

$$P_N = \frac{P_S}{N_S} + NIC \ , \tag{2}$$

making P_N equal to about 28.33 watts. Power associated with communication during application computation is assumed to be accounted for in the power values taken from [2].

IV. Fault Resilience Techniques

A. Overview

Two HPC fault resilience techniques are considered in our work and were implemented in our system simulator: a traditional checkpoint restart based technique, *checkpoint/restart*, as well as the implementation of multilevel checkpointing proposed for next-generation HPC systems in [9]. The following subsections present details of how each resilience technique was modeled.

B. Checkpoint Restart

Our implementation of the checkpoint/restart resilience technique performs periodic, blocking checkpointing, with checkpoints saved to a parallel file system. The time that the checkpoint/restart technique requires to read and write its checkpoint data to a parallel file system, $T_{C_{PFS}}$, is dependent on number of nodes required by the application, N_A, memory use N_m (defined in Section III-B), and communication bandwidth B_N (defined in Section III-E) to give

$$T_{C_{PFS}} = \frac{N_m}{B_N} * N_A \ . \tag{3}$$

The optimal checkpoint period is dependent on the application's checkpoint time and failure rate. As defined in Section III-D, the value for an application's failure rate is dependent on the application's size. We describe our calculation of the optimal interval between application checkpoints (τ) when optimizing for either maximum performance efficiency or minimum energy use in Section V.

C. Multilevel Checkpointing

We implement the three-level multilevel checkpointing technique from [9] in our simulator. Each checkpointing level offers a trade-off between the time required to save or restore a checkpoint and the severity class of the failure from which it can recover.

The first checkpoint level writes to the node's local RAM. All applications save the checkpoint concurrently with the time required for taking a level one checkpoint being simply the amount of memory per node required by the application divided by the node's memory transfer rate

$$T_{C_{L1}} = \frac{N_M}{B_M} \ . \tag{4}$$

The second checkpoint level stores its checkpoints to RAM in a partner node. The time for a level two checkpoint is equal to the time required to send the data to the partner node (calculated using variables defined in Section III-E) plus the time required to write the data to memory

$$T_{C_{L2}} = 2(T_{C_{L1}} + L_N + \frac{N_M}{B_M}) \ . \tag{5}$$

The equation is multiplied by two to account for both the time required for half of the system to concurrently send checkpoint data to their respective partner nodes as well as the time required for those nodes to receive their partner's checkpoint data.

The third level checkpoint is written to a parallel file system, and the time required is the same as presented in Eqn. 3. We assume checkpoint and restart times are symmetric. Failure severity classes are defined according to Section III-D and dertimination of optimal checkpoint intervals for each level is describe in Section V.

V. Execution Time and Checkpoint Interval Estimation

A. Overview

Sections V-B and V-C discuss equations we derive that predict the expected execution time of applications executing in the presence of failures when employing either the traditional checkpoint/restart or the multilevel checkpointing techniques described in Section IV. Due to lack of space, the derivations for the equations used throughout Section 4 are described in more detail in [32]. Section V-D discusses how these execution time prediction equations are then extended for use in predicting the system's expected energy use when executing the application. These equations are general and able to be used on systems and applications of any type provided the relevant system and application parameters can be estimated.

All equations are organized to estimate the expected values of each type of resilience, failure, and execution-related events during application execution (the events described in Section III-A) and the sum of these values is the total expected execution time. The application expected execution time for all occurrences of each event type is generally calculated by multiplying an estimator for the expected number of occurrences of the event by the expected execution time of the event.

Checkpoint interval optimization is performed by sweeping the equation's decision variables through values in the solution space and determining which values maximize performance efficiency or minimize application energy use. For the checkpoint/restart technique, this means a single decision variable (the computation interval) and for multilevel checkpointing the number of decision variables will be equal to the number of checkpoint levels, L.

B. Execution Time Model with Checkpoint/Restart

The expected application execution time when using checkpoint/restart, T_{CR}, is equal to the sum of the application's time for all:

- computation of the application without overhead from resilience or failures (baseline execution time), T_B;
- successful checkpoints, T_δ;
- failed checkpoints, $T_{\delta'}$;
- successful restarts, T_R;
- failed restarts, $T_{R'}$;
- recomputation of work lost to a failure occurring during a computation interval, T_{W_τ};
- recomputation of work lost to a failure occurring during a checkpoint, T_{W_δ}.

This gives a total expected execution time of

$$T_{CR} = T_B + T_\delta + T_{\delta'} + T_R + T_{R'} + T_{W_\tau} + T_{W_\delta} \ . \tag{6}$$

For the checkpoint/restart technique, an estimate of the application's baseline execution time, T_B, is assumed to be known by the system designer. The time required for a single checkpoint, δ, is also known (from profiling) so the total time for successful checkpoints, T_δ, can be precisely calculated as the total number of successful checkpoints multiplied by δ, with the number of successful checkpoints equal to the application's baseline execution time divided by the computation interval, τ, giving

$$T_\delta = (\frac{T_B}{\tau} - 1)\delta \ . \tag{7}$$

All remaining terms in Eqn. 6 are dependent on the number of failures that occur during the application's execution and must therefore be estimated. The estimated total number of failures throughout the application's execution is the product of the failure rate, λ, and the application's total execution time giving an estimated total number of failures equal to (λT_{CR}). This estimator will be relied on for all terms in Eqn. 6 that are dependent on failures.

Each term's expected value is estimated as the expected number of occurrences of the event multiplied by the expected time of the event if a failure occurs. 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). The use of this approach in our equation-based model is flexible and allows for any integrable PDF to be used to represent the distribution of system failures.

As stated in Section III-D, we are assuming that failures follow an exponential distribution, making the probability of a failure occurring during any given interval of time t for a failure rate λ equal to

$$P(t, \lambda) = 1 - \exp^{-\lambda t} \ . \tag{8}$$

As opposed to the expected value of the general PDF, which is calculated over the entire domain, $[0, \infty)$, 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

$$E(t, \lambda) = \frac{\frac{1}{\lambda} - \exp^{-\lambda t}(\frac{1}{\lambda} + t)}{P(t, \lambda)} \ . \tag{9}$$

The estimator for the expected number of failures that occur during a checkpoint, α', can be shown to be a function of the number of successful checkpoints and the probability of a failure occurring during a checkpoint to give

$$\alpha' = \frac{P(\delta,\lambda)(\frac{T_B}{\tau}-1)}{1-P(\delta,\lambda)} \ . \tag{10}$$

Using Eqns. 9 and 10 we can calculate the expected time that the application wastes due to failed checkpoints as

$$T_{\delta'} = \alpha' E(\delta,\lambda) \ . \tag{11}$$

In addition to the time incurred for the failed checkpoint itself, each failed checkpoint also incurs overhead associated with the lost computation interval when the checkpoint fails. This value, T_{W_δ}, is calculated as

$$T_{W_\delta} = \alpha' \tau \ . \tag{12}$$

Note that α' can also be divided by the value of the total number of failures during execution (λT_{CR}) to give the percentage of total failures that occur during a checkpoint, denoted as α.

For the checkpoint/restart technique, the expected percentage of failures that occur during a restart, ζ, can be shown to be simply equal to the probability of a failure occurring during a restart

$$\zeta = P(R,\lambda) \ . \tag{13}$$

This makes the total expected time that the application spends for successful restarts equal to the total number of failures that do not occur during a restart multiplied by the time for a successful restart

$$T_R = (\lambda T_{CR})(1-\zeta)R \ . \tag{14}$$

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

$$T_{R'} = (\lambda T_{CR})(\zeta)E(R,\lambda) \ . \tag{15}$$

The calculation of T_{W_τ} in Eqn. 6 uses the values of α and ζ to estimate the total number of failures that occur during computation and estimates the total expected time that the application loses due to failures during computation as

$$T_{W_\tau} = (\lambda T_{CR})(1-\zeta-\alpha)E(\tau,\lambda) \ . \tag{16}$$

Once all values are defined, the expected execution time of the application can then be calculated by solving Eqn. 6 for T_{CR}.

C. Execution Time Model with Multilevel Checkpointing

The application execution time model when using multilevel checkpointing is similar to that of Eqn. 6, however it is defined recursively to account for each level of the multilevel checkpointing technique. For this technique, failure rates of different levels must be differentiated. For a failure severity, $i = 1,...,L$, the corresponding failure rate, λ_i, is defined as the product of the system failure rate, λ, and the probability of a failure at that severity, S_i, making the failure rate $\lambda_i = S_i\lambda$. Each of the other variables defined for the checkpoint/restart technique (i.e., T_δ, $T_{\delta'}$, ζ, R, etc.) are now L-dimensional vectors.

The multilevel checkpointing technique from [9] 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 defining the number of L_{i-1} checkpoints that must occur before each L_i checkpoint is taken are the set of $L-1$ integer decision variables $N_1, ..., N_{L-1}$ used for optimizing the equation. The last decision variable is the computation interval, a real number which we define as τ_0. This set of decision variables recursively 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.

The amount of total time spent between each level i checkpoint is referred to as the level $i+1$ computation interval. Each higher level computation interval, τ_{i+1}, is calculated as

$$\tau_{i+1} = T_{B_i} + T_{\delta_i} + T_{\delta'_i} + T_{R_i} + T_{R'_i} + T_{W_{\tau_i}} + T_{W_{\delta_i}} \tag{17}$$

with the application's total expected execution time when using multilevel checkpointing $T_{ML} = \tau_{L+1}$.

The time spent checkpointing at each level is now defined as

$$T_{\delta_i} = N_i\delta_i \ . \tag{18}$$

The estimator for the expected number of failures that occur during each level i checkpoint, α'_i, is now defined as

$$\alpha'_i = \frac{P(\delta_i, \sum_{j=1}^{i}\lambda_j)N_i}{1-P(\delta_i, \sum_{j=1}^{i}\lambda_j)} \ . \tag{19}$$

Making the expected time that is wasted due to failed checkpoints

$$T_{\delta'_i} = \alpha'_i E\Bigg(\delta_i, \sum_{j=1}^{i}\lambda_j\Bigg) \ . \tag{20}$$

The overhead associated with the lost computation interval caused by the failed checkpoint is calculated as

$$T_{W_{\delta_i}} = \alpha'_i \sum_{k=1}^{i} S_k\tau_k \ . \tag{21}$$

The expected percentage of failures that occur during a restart of level i, ζ_i, is now

$$\zeta_i = S_i P\Bigg(R_i, \sum_{j=1}^{i}\lambda_j\Bigg) \ . \tag{22}$$

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

$$T_{R_i} = (\lambda T_{ML})(S_i - \zeta_i)R_i \ . \tag{23}$$

The total time that the application spends for failed restarts is now

$$T_{R'_i} = (\lambda T_{ML})(\zeta_i)E\Bigg(R_i, \sum_{j=1}^{i}\lambda_j\Bigg) \ . \tag{24}$$

The calculation of T_{W_τ} becomes

$$T_{W_{\tau_i}} = (\lambda T_{ML})(S_i - \zeta_i - \alpha_i)E(\tau_i, S_i\lambda) \ . \tag{25}$$

D. Energy Use Model

The energy use models for both checkpoint/restart and multilevel checkpointing are calculated by multiplying the expected execution time of each event type with the power used to perform that event. For computation events, system power use is equal to the power used by a node during computation (P_W, defined in Section III-C) multiplied by the number of nodes used by the application (N_A from Section III-D). For checkpoint or restart related events, power use is defined by the number of nodes used by the application multiplied by the power used by a node while it is checkpointing or restarting

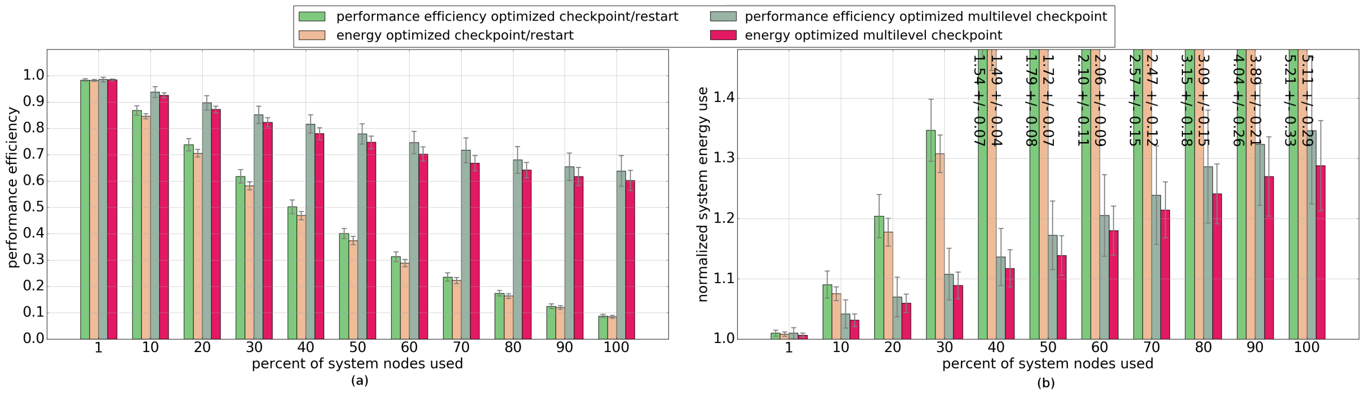

Fig. 1: Application **(a)** performance efficiency and **(b)** normalized energy use, when resilience technique checkpoint intervals are optimized for either performance efficiency or energy use and the percentage of system nodes used by the application is increased. Bars in the figure represent the average of 200 simulated trials. Standard deviations are shown for each bar. Annotations in the figure indicate values for the average and standard deviation of bars that have been truncated.

(P_δ, defined in Section III-C). Once the values for the total execution time have been calculated the expected execution time for each event type can be calculated. This makes the expected total energy for checkpoint/restart, E_{CR}, equal to

$$\begin{aligned} E_{CR} = P_W N_A(T_B + T_{W_\tau} + T_{W_\delta}) \\ + P_\delta N_A(T_\delta + T_{\delta'} + T_R + T_{R'}) \ , \end{aligned} \qquad (26)$$

and E_{ML}, the expected energy for multilevel checkpointing

$$\begin{aligned} E_{ML} = P_W N_A(T_B + T_{W_{\tau_L}} + T_{W_{\delta_L}}) \\ + P_\delta N_A(T_{\delta_L} + T_{\delta'_L} + T_{R_L} + T_{R'_L}) \ . \end{aligned} \qquad (27)$$

E. *Checkpoint Interval Optimization*

To optimize Eqn. 6 for maximum performance efficiency we simply sweep through the set of values over the interval $(0, T_B)$ for the decision variable τ and evaluate Eqn. 6 to find the minimum execution time for T_{CR}. Similarly, selecting decision variables that maximize performance efficiency for the multilevel checkpointing technique is accomplished by evaluating T_{ML} 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 for τ_0 by the interval $(0, T_B)$, and bounded for $N_1, ..., N_{L-1}$ such that the product of these values and τ_0 is greater than zero and less than the application's baseline execution time, i.e., $0 < \tau_0 \left(\prod_{i=1}^{L} N_i \right) < T_B$. For both checkpoint/restart and multilevel checkpointing, 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 it is here.

We use the same brute force sweeping technique when optimizing decision variable's for minimal energy use except that instead of evaluating for execution time we evaluate Eqns. 26 and 27 to find decision variable values that produce the minimum expected energy use. When optimizing for energy use, the solution space for decision variable values has the same bounds as when optimizing for performance efficiency.

VI. Simulation Experiments

A. *Overview*

We use the equations defined in Section V to calculate checkpoint intervals that either maximize performance efficiency or minimize energy use and use the resulting interval values with the simulation environment discussed in Section III to conduct several simulated experiments in Section VI-B that examine the trade-off between performance and energy. In Section VI-C, we perform a sensitivity analysis that explores how this trade-off is affected when various application characteristics and system parameters are scaled through a range of values.

B. *Optimization Trade Off*

In Figure 1, we demonstrate the performance of and energy use of the system as the simulated application is scaled in size from one percent of the exascale system (about 1.2 million CPU cores, similar in size to some of today's largest applications) through to an exascale-sized application requiring 123 million CPU cores. For these experiments, the baseline execution time for each application is defined as $T_B = 86400$ seconds, or one full day of execution. The energy use values are normalized to the calculated value of the application's baseline energy use, $E_B = P_W N_A T_B$, for each application size. Most of the prior work we consider assume a node MTBF of ten years for current HPC systems. We assume component failure rates will increase linearly with the increased size of system nodes, and consequently for our experiments we assume an MTBF of $M_n = 2.5$ years. We assume that the application uses $N_m = 32$GB of memory per node. Simulated trials vary because of uncertainty associated with randomly occurring failures. Bars in the figure represent the average of 200 simulated trials and the error bars indicate standard deviations.

Figure 1 shows that for both checkpoint/restart and multilevel checkpointing there exists a distinct trade-off between optimizing checkpoint intervals for performance efficiency and optimizing checkpoint intervals for energy use. Checkpoint/restart is 0.5-3% more efficient when optimizing for performance efficiency than when optimizing for energy use but consumes as much as 15% more energy. Multilevel checkpointing has as much as 4% higher performance efficiency when optimizing for performance efficiency but can consume as much as 7% less energy when optimizing for minimum energy use. This difference in optimality arises because of the difference in power requirements when the system is checkpointing or restarting as opposed to when it is performing computation. If a system designer desires to use less energy, then the results in Figure 1 indicate that it is more beneficial to slightly increase the execution time of the application by taking more frequent checkpoints that require less power but require less time be spent performing power costly computation when recovering from a failure.

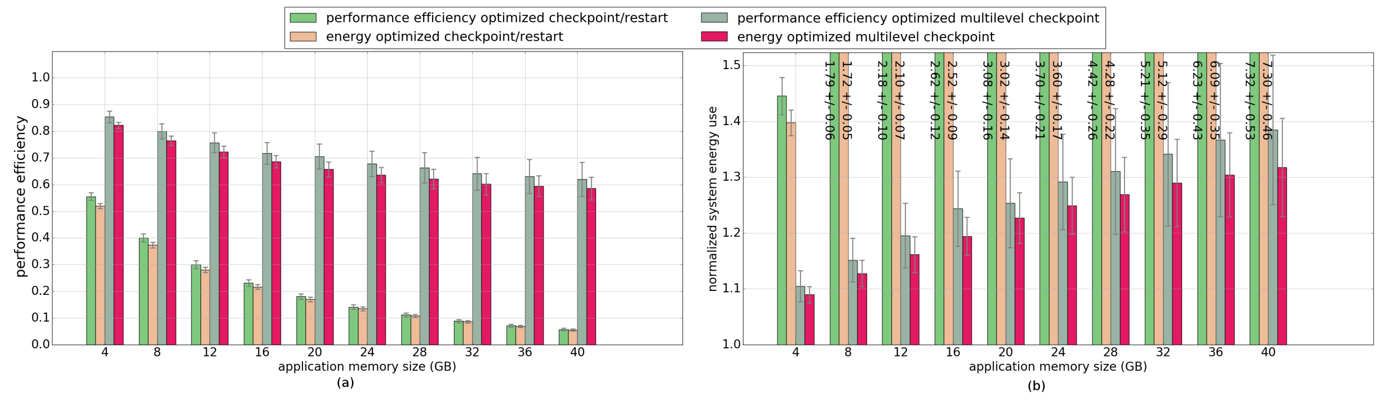

Fig. 2: Application **(a)** performance efficiency and **(b)** normalized energy use, when resilience technique checkpoint intervals are optimized for either performance efficiency or energy use and the amount of memory used by the application is increased. Bars in the figure represent the average of 200 simulated trials. Standard deviations are shown for each bar. Annotations in the figure indicate values for the average and standard deviation of bars that have been truncated.

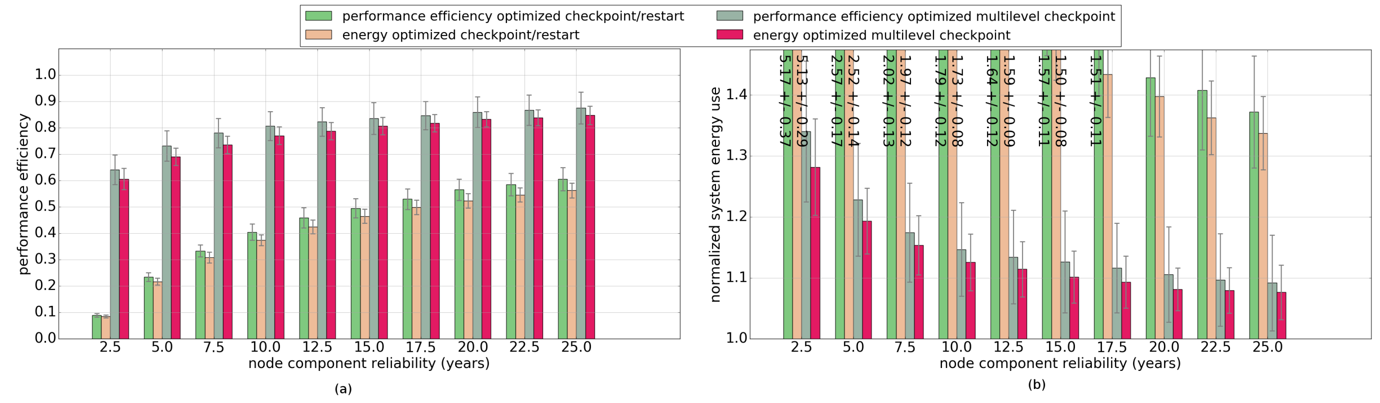

Fig. 3: Application **(a)** performance efficiency and **(b)** normalized energy use, when resilience technique checkpoint intervals are optimized for either performance efficiency or energy use and the reliability of system nodes is increased. Bars in the figure represent the average of 200 simulated trials. Standard deviations are shown for each bar. Annotations in the figure indicate values for the average and standard deviation of bars that have been truncated.

As the application's size increases it can be observed that when optimizing for energy use with the multilevel checkpointing technique the loss in performance efficiency increases at a slower rate than the decrease seen in energy use, allowing the same burden on efficiency to provide larger decrease in energy use for larger application sizes. This effect is less prevalent in the results for checkpoint/restart.

Another trend that is seen with both checkpoint/restart and multilevel checkpointing is that while the variance in results for both efficiency and energy use increases with application size, the variance of results when optimizing for minimum energy use increases more slowly than the variance in efficiency results. This indicates that in terms of both performance efficiency and system energy use execution results are more consistent if checkpoint intervals are optimized for minimal energy use rather than higher performance efficiency. If a system designer desires more predictability of application execution then it is better to optimize for minimum energy use.

C. *Sensitivity Analysis*

We analyzed the sensitivity of the results from Section VI-B to a variety of application characteristics and system parameters including: application memory use, system component reliability (expected time between component failures), communication network bandwidth, communication network latency, and system node memory transaction speed. All simulations analyzing sensitivity were performed by simulating an exascale-sized application with all system parameters and execution characteristics that were not being scaled for the sensitivity analysis (i.e., the parameters that are held constant) remaining as described in Section III. All experiments we performed supported the results discussed in Section VI-B indicating the presence of a trade-off when optimizing for efficiency or energy use as well as all analyzed parameters indicating that there is less variance present in application execution when optimizing checkpoint intervals to minimize system energy use.

A subset of the sensitivity results are shown in Figures 2 and 3. The figures show the efficiency and energy use when scaling values of application memory use and system component reliability, respectively. Bars in the figures represent the average of 200 simulated trials with standard deviations shown for each bar.

Increase in application memory use (Figure 2) has a particularly large effect on both the efficiency and energy use results as well as the trade-off in improvements that can be gained by optimizing for either efficiency or energy use. Though not all results are shown, the results for sensitivity to changes in application memory size had the largest impact on performance efficiency and system energy use for an exascale-sized application. Checkpoint/restart shows a 5-14% decrease in energy use for at most a 3% decrease in performance efficiency when optimizing for energy use over performance efficiency. Multilevel checkpointing achieves between a 2-6% decrease in energy use for a 2-3% decrease in performance efficiency when optimizing for energy use over performance efficiency.

The results in Figure 3 show that as component reliability (MTBF) increases, the performance of the system increases and the energy improvement gained when optimizing energy use over performance efficiency decreases. However, even in a highly reliable system where failures are less common, the trade-off as well as the increased predictability of performance when optimizing for energy use are still present and can be taken advantage of by a system designer.

VII. Conclusions

HPC resilience has become an increasingly important topic as we approach exascale system sizes and failures become more frequent. Similarly, as systems begin to be developed that require hundreds of thousands of system nodes, the energy requirements of these systems becomes extremely costly and reducing this burden continues to be an important consideration for system designers.

We described a methodology that can be used to simulate exascale HPC systems (with the ability to scale to arbitrary system sizes) and model the effects that extreme-scale systems have on performance efficiency and energy use in the presence of node failures.

We utilize our simulation models to evaluate two techniques for HPC resilience, the traditionally employed checkpoint/restart technique, as well as the multilevel checkpointing technique proposed for next generation large-scale systems and use a set of equation-based models we have developed to optimize the checkpointing intervals of these techniques to either maximize performance efficiency or minimize system energy use. Our analyses indicate that a performance trade-off exists between optimizing these techniques for either metric. Given the presence of this trade-off, we performed a sensitivity analysis on several parameters associated with application and system behavior and conclude that this trade-off exists in all circumstances we test. Our results also indicate that optimizing for minimal energy use provides the system with less variable ranges of execution times.

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] 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.

[2] J. Dongarra, "Report on the Sunway Taihulight System," Tech. Rep. UT-EECS-16-742, June 2016.

[3] P. Gas and E. Company, "Electric schedule e-19," Tech. Rep. ELEC -SCHEDS-E-19, Apr. 2015.

[4] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, Sep.

[5] D. P. Jasper, "A discussion of checkpoint restart," *Software Age*, vol. 3, pp. 9–14, Oct. 1969.

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

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

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

[9] 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*, Nov.

[10] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello, "Optimization of multi-level checkpoint model for large scale HPC applications," in *Int'l Par. and Dist. Proc. Symp.*, pp. 1181–1190, May 2014.

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

[12] 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.

[13] E. Meneses, O. Sarood, and L. V. Kalé, "Assessing energy efficiency of fault tolerance protocols for hpc systems," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, pp. 35–42, Oct.

[14] O. Sarood, E. Meneses, and L. V. Kalé, "A 'cool' way of improving the reliability of HPC machines," in *2013 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov.

[15] E. Meneses, O. Sarood, and L. V. Kal, "Energy profile of rollback-recovery strategies in high performance computing," *Parallel Computing*, vol. 40, pp. 536–547, Oct. 2014.

[16] B. Mills, T. Znati, R. Melhem, K. B. Ferreira, and R. E. Grant, "Energy consumption of resilience mechanisms in large scale systems," in *22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 528–535, Feb.

[17] 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*, Dec. 2016.

[18] R. R. Chandrasekar, A. Venkatesh, K. Hamidouche, and D. K. Panda, "Power-check: An energy-efficient checkpointing framework for HPC clusters," in *International Symposium on Cluster, Cloud and Grid Computing*, pp. 261–270, May 2015.

[19] S. H. Lim, S. W. Lee, B. H. Lee, and S. Lee, "Power-aware optimal checkpoint intervals for mobile consumer devices," *IEEE Trans. on Consumer Electronics*, vol. 57, pp. 1637–1645, Nov. 2011.

[20] 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)*, pp. 914–923, May 2017.

[21] D. Dauwe, E. Jonardi, R. D. Friese, S. Pasricha, A. A. Maciejewski, D. A. Bader, and H. J. Siegel, "HPC node performance and energy modeling with the co-location of applications," *The Journal of Supercomputing*, vol. 72, pp. 4771–4809, Dec. 2016.

[22] "Top500 June 2017," accessed Aug. 2017.

[23] A. Marowka, "Back to thin-core massively parallel processors," *Computer*, vol. 44, pp. 49–54, Dec. 2011.

[24] T. Simunic, K. Mihic, and G. De Micheli, "Optimization of reliability and power consumption in systems on a chip," in *Int'l Workshop on Integrated Circuit and System Design Power and Timing Modeling, Optimization, and Simulation*, pp. 237–246, Sep. 2005.

[25] J. S. S. T. Association, "DDR4 SDRAM standard," Tech. Rep. JESD79-4B, June 2017.

[26] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "The Sunway Taihulight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, 24 pp., June 2016.

[27] G. Yang, *Life Cycle Reliability Engineering*. Hoboken, NJ: John Wiley & Sons, second ed., 2007.

[28] C. D. Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of Blue Waters," in *Int'l Conf. on Dependable Systems and Networks*, pp. 610–621, June 2014.

[29] 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, 2015.

[30] N. R. Tallent, K. J. Barker, D. Chavarria-Miranda, A. Tumeo, M. Halappanavar, A. Marquez, D. J. Kerbyson, and A. Hoisie, "Modeling the impact of silicon photonics on graph analytics," in *Int'l Conf. on Networking, Architecture and Storage (NAS)*, 11 pp., Aug. 2016.

[31] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *Int'l Symp. on Computer Architecture*, pp. 338–347, June 2010.

[32] D. Dauwe, "Resource managaement for extreme scale high performance computing systems in the presence of failures," in *Ph.D Thesis, Colorado State University*, under preparation.