2016 IEEE International Conference on Computer and Information Technology

A Performance and Energy Comparison of Fault Tolerance Techniques for Exascale Computing Systems

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
Email: ddauwe@rams.colostate.edu, sudeep@colostate.edu, aam@colostate.edu, hj@colostate.edu

***Abstract*—As the computing power of large scale computing systems increases exponentially with time, their failure rates are increasing exponentially as well. While current high performance computing (HPC) systems experience failures of some type every few days, projections indicate that the next generation exascale machines will experience failures up to several times an hour. The resilience techniques implemented in today's HPC and cloud computing systems do not efficiently scale up to the exascale level. Several more scalable resilience techniques have recently been proposed for next generation HPC systems. However, thus far it is not known how these techniques directly compare to one another in terms of performance and energy use. This work explores four resilience techniques, and considers each technique's ability to handle varying levels of system reliability and system sizes. We demonstrate how each technique compares in terms of application performance and energy use and provide recommendations on their suitability at an exascale level.**

Keywords: resilience; checkpoint restart; multilevel checkpointing; message logging; fault tolerance; energy efficiency;

I. Introduction

System reliability has recently become a looming and unsolved problem in the field of high performance computing (HPC). Today's HPC systems are nearing the performance capability of one-hundred petaflops [1]. Current HPC systems experience failures on the order of every few days, but models indicate that an exascale-sized system will likely experience system failures several times an hour [2].

Contemporary HPC systems have thus far been able to successfully mitigate the effects of system failures through the use of checkpoint-based rollback recovery and redundant execution of code using additional hardware. However, as the mean time between failures (MTBF) of future HPC systems decreases, these traditional approaches to system resilience either do not scale to meet the system's increased demand for performance or require too much energy to be feasibly implemented [3].

Recent works have proposed several alternative resilience techniques that are potentially better able to handle the increasing numbers of failures associated with these larger scale systems [2] [4] [5]. However, the relative performance and energy use of each of these techniques when compared with one another is unclear.

This work provides a simulated comparison of the performance and energy use of three of these new resilience techniques for HPC systems. The simulated system is based on analytical models for the NAS Block Tridiagonal benchmark application [6] executing at extreme scale, as well as measurements taken from the Block Tridiagonal benchmark executed on real-world server-class processors. Each resilience technique's performance and energy use is simulated with varying system sizes and hardware component reliability levels. Analysis is performed detailing the strengths and weaknesses of each resilience technique. We conclude with a comparison between each technique's performance relative to the others, as well as each technique's performance compared to a traditional checkpoint restart resilience approach.

With this work we make the following novel contributions:

- we construct a methodology for simulating the execution, power consumption, and energy use of extreme-scale computing systems operating with the uncertainty of hardware failures;
- we provide a comparison of the relative performance and energy consumption of four strategies for extreme-scale high performance and cloud computing resilience.

II. Related Work

A. Overview

We limit the resilience techniques considered to those that are transparent to application programmers and system users. While several other techniques for mitigating the effects of system failures exist, we refer the reader to the summaries provided of such works in [7], [8], and [9]. Specifically this work focuses on examining techniques utilizing rollback recovery and redundancy.

B. Rollback Recovery

Rollback recovery based techniques rely on periodically saving the system's executing state and rolling back to an earlier state after the occurrence of a failure. Such a technique is referred to as checkpointing [10] [11]. All rollback recovery techniques rely on this notion of checkpointing in some form. Because of their nature of loading an earlier system state

978-1-5090-4314-9/16 $25.00 © 2016 IEEE
DOI 10.1109/CIT.2016.44

after a failure, all rollback recovery techniques necessarily lose some productivity as the restarted applications must recompute work lost between the time of the failure and the time of the last checkpoint. Our work examines three types of rollback recovery techniques: checkpoint restart, multilevel checkpointing, and message logging.

1) Checkpointing and Restarting: Checkpointing is by far the most commonly used resilience technique employed by today's HPC systems. The most general implementation of the checkpointing technique operates by stopping the system's execution at regular intervals to save the state of all executing applications to a permanent storage device, typically a parallel file system. Such a checkpointing technique is called a blocking, coordinated checkpoint [7]. Several variations and improvements on this technique have been made since its initial inception. Attempts have been made to create non-blocking or semi-blocking checkpointing, which allows the system to continue to execute while checkpoints are saved to permanent storage [12] [13]. Attempts have also been made to allow for uncoordinated checkpoints of the system, preventing the need for all processes in the system to restart when a failure occurs [14]. However, the length of time associated with checkpointing, restarting, and recomputing work lost to a system failure, and the frequency that the system needs to take checkpoints for very large scale applications with any of these checkpointing techniques has been shown to provide diminishing returns with increasing system sizes. Traditional checkpointing alone is thus not expected to be capable of providing resilience to systems at exascale sizes.

2) Multilevel Checkpointing: Multilevel checkpointing involves multiple levels of checkpointing, each level offering 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 [4]. In general, checkpoints of different levels correspond to saving data to different levels of the memory subsystem or allow for saving checkpoints across the memory of one or more partner nodes. Checkpoint levels also may employ various encoding techniques (such as RAID or Reed-Solomon coding) to improve the resilience offered by a particular checkpoint level [4] [15]. Attempts also have been made to reduce checkpointing's dependence on the parallel file system [16] [17] [18]. One challenge associated with using multilevel checkpointing is in determining the optimal number of checkpointing levels and the optimal computation intervals between checkpoints at each level. Various solutions to this problem have been proposed [4] [19] [20].

3) Message Logging: Message logging attempts to provide resilience to a system by recording messages sent among processes to create snapshots of the system's execution distributed across system memory [21]. When a failure occurs, the failed node is able to use messages stored in the memory of other system nodes to save on the amount of rework that is performed by the system when restarting from a checkpoint on a failure [22].

C. Redundancy

Redundancy based techniques improve a system's reliability by executing redundant copies of the same piece of code [23]. It is possible to implement redundancy in either hardware or software [8], but in either case the improved reliability comes at a cost of using additional resources.

Because of its necessity for using more resources, redundancy alone is typically not considered to be viable resilience solution for exascale systems. However, recent been made to allow the system to utilize redundancy in less resource-intensive ways. Dynamic redundancy allows for the executing application to choose a subset of processes for redundant execution [24]. Partial redundancy combines redundancy with checkpointing, and allows for applications to redundantly execute a portion of processes in the system, providing improved resilience for part of the system, while using only a portion of the necessary system resources [5].

III. HPC System Simulator

A. Overview

Because no exascale system is currently available, we perform experiments via simulation. We have created an event-based simulator capable of simulating application execution on arbitrarily large systems [25] [26]. Simulated applications execute under the influence of randomly generated system failures, but are able to employ various resilience techniques capable of helping to mitigate the impact that those failures have on the application's execution. Throughout an application's simulated execution, values of execution time and energy use are recorded for the simulated events associated with computation (execution toward the application's completion), checkpoints (saving a backup of the application's current computation progress), restarts (restoring the application progress saved in the last system checkpoint after a failure occurs), recovery (recomputing progress lost to a failure after the system has restarted), and failed checkpoints or restarts (when failures occur during checkpoint or restart events).

With the exception of computation events, the actions taken for each simulated event are determined according to the resilience technique employed by the simulated system, as outlined in Section III-C.

B. Modeling System Failures

The probability of failures occurring in HPC systems are commonly modeled according to exponential distributions [27]. Our simulator follows this assumption and generates failures according to a Poisson process, with each new failure arriving according to the previous failure's arrival time ($T_{A_{i-1}}$, with $T_{A_0} = 0$) plus a random variate generated from an exponential distribution $\mathcal{T}_i \sim Exp(\lambda_{sys})$ with an expected arrival rate of $E[\mathcal{T}_i] = \frac{1}{\lambda_{sys}}$. The parameter λ_{sys} indicates the average failure rate of the entire system, and is defined as the number of nodes in the simulated system, N_{sys}, divided by the mean time between failures of the system nodes, $MTBF_{Node}$,

$$\lambda_{sys} = \frac{N_{sys}}{MTBF_{Node}} , \quad (1)$$

with N_{sys} and $MTBF_{Node}$ explicitly defined in Sections III-C, V-B, and V-C for each experiment.

In addition to the time at which failures occur, some resilience technique's behavior depends on having information about which system node has failed (in the case of redundancy) and the severity of the failure (in the case of multilevel checkpointing). Failure generation in our simulator accommodates both of these additional failure attributes. When determining which node has failed the simulator assumes a uniform random distribution over all active nodes in the system and selects one node at random as the failed node. The level of failure severity is determined by the type of system failure and the implementation of the multilevel checkpointing technique. We have assumed the multilevel checkpointing implementation of [4]. During simulation, the probability of having a failure at a severity level j is determined by the ratio of the number of failures that occur at failure severity level j, λ_{L_j}, to the total number of failures, $\lambda_{L_{tot}}$, for failures measured from a system over an extended interval of time. The resulting discrete set of probabilities for each of the j levels is used to create a probability mass function. During simulation, random variates are sampled from this probability mass function to produce integers that define the severity level of each failure.

C. Resilience Technique Simulation

Four resilience techniques have been implemented in our simulator. A traditional checkpoint restart based technique, *Checkpoint Restart*, as well as three techniques proposed for next-generation HPC systems. The multilevel checkpointing approached described in [4], *Multilevel Checkpoint*, an implementation of message logging outlined in [2], *Parallel Recovery*, and a technique combining traditional checkpointing with partial redundancy of the system hardware in [5], *Redundancy*. We now describe the implementation of these techniques.

1) Checkpoint Restart: The Checkpoint Restart technique implemented in our simulator is the traditional periodic, blocking, coordinated checkpointing technique, with its checkpoints saved to a parallel file system. This basic strategy for checkpointing has been the baseline that is compared against by many contemporary works in HPC resilience. All three of the next-generation resilience techniques explored in this work not only use this traditional checkpointing technique as a baseline for comparing their own work, but all of them, to some extent, base their own improved techniques on this approach.

2) Multilevel Checkpointing: The approach in [4] implements a three-level checkpointing model. Associated times for checkpointing, $T_{C_{Lj}}$, times for restarting, $T_{R_{Lj}}$, and failure severity ratios, $\frac{\lambda_{L_j}}{\lambda_{L_{tot}}}$, are defined according to [4]. The optimal checkpoint intervals at each level also are determined according to the Markov model in [4].

3) Parallel Recovery: The technique in [2] adapted in our simulator is an improvement to the message logging resilience technique. Parallel Recovery allows for faster recovery from a system failure by allowing the failed node's work to be temporarily parallelized across several nodes after being restarted, thereby reducing the time needed by the system to recompute the work lost to a failure. As with all message logging techniques, parallel recovery benefits the system by allowing functional nodes being used by the executing parallel job that included the failed node to remain idle while the failed node recomputes work lost due to a failure. This decreases both the system power needed during recovery as well as the chance that a failure will interrupt the recovering system in comparison to other checkpoint based techniques. However, unlike other message logging techniques, parallel recovery improves checkpointing and restart time by utilizing in-memory checkpointing as outlined in [28].

4) Partial Redundancy: The Partial Redundacy technique in [5] combines traditional checkpointing with varying degrees of hardware redundancy. "Partial" redundancy is achieved by allowing only a fraction of the total system nodes required by the executing application to have redundant hardware during its execution. For example, a degree of redundancy of $r = 2.5$ dictates that each *virtual process* of an executing application requiring a single node will have at lease two physical nodes performing the same computation, and half of the virtual processes will have three physical nodes performing the same computation. At the same time, checkpoints are taken by the system at regular intervals. When failures occur on system nodes, the system only requires a restart if failures occur on all (possibly redundant) physical nodes associated with one of the application's virtual processes before the next checkpoint.

IV. Exascale Modeling Methodology

A. Overview

The foundation of each of our simulated system experiments is either based on measurements taken from a real-world system or extrapolated from analytical equations. This section details the methodology we use to construct various system setups for simulating applications executing at extreme scales.

B. NAS Block Tridiagonal Benchmark at Extreme Scales

Without access to an exascale system it is not possible to directly measure application performance, energy use, or behavior of the application operating when employing a given resilience technique. However, the work in [29] provides an asymptotic analysis of the execution of the NAS Parallel Benchmark's Block Tridiagonal (BT) application [6] at extreme scales. An instance of the BT application executes with: P MPI ranks (with each rank mapped to a single CPU core), $q = \sqrt{P}$ data blocks per MPI rank, a problem size of N^3 total data points, and $(\frac{N}{q})^3$ data points per data block. Analysis in [29] indicates that, for each executed time step of the application, each MPI rank will have $2994\frac{N^3}{q^2}$ total floating point operations of computation work, occupy $368\frac{N^3}{q^2}$ bytes of data in memory, send a total of $6q$ messages between MPI ranks, and transmit a total of $1320\frac{N^2}{q}$ bytes of messages.

Because the BT application exhibits weak scaling, by definition as the number of MPI ranks P increases to P' the computational work per MPI rank remains constant. With the computational work of each MPI rank constant, it is shown in [29] that the amount of memory used by each

MPI rank also remains constant, and that the data volume of communications increases by a factor of $\sqrt[6]{P'/P}$ to give a total scaled communication volume of

$$V = 1320\frac{N^2}{q}\sqrt[6]{P'/P}\,, \tag{2}$$

sent in a total of

$$K = 6\sqrt{\frac{P'}{P}} \tag{3}$$

messages. For a network latency of L converted to units of seconds and a bandwidth of BW in units of gigabytes per second, using Equations 2 and 3 we can derive the time spent on communication in each time step as

$$T_{COMM} = KL + \frac{V}{BW}\,. \tag{4}$$

For a node with N_{cores} number of cores, executing at R_{FLOPS} floating point operations per second, the time spent on computation in each time step, for each MPI rank is

$$T_{COMP} = \frac{2994N_{cores}}{R_{FLOPS}}\frac{N^3}{q^2}\,. \tag{5}$$

C. Real-World System Measurements

Several parameters required for performing experiments with our simulated system require real-world measurements of execution of a node. Node measurements were taken from the execution of an HP Z820 workstation [30]. The target Z820 machine used has two sockets, with each socket supporting a 12-core Intel Xeon E5-2697v2 processor [31], for a total of 24 cores in the compute node.

1) System Power measurement: Power measurements of the Z820 system were taken using a "Watts Up? PRO" power meter [32]. The "Watts Up? PRO" monitors the instantaneous power use of the Z820 at the "wall outlet" level, recording samples at one second intervals. Power use of the system was measured for the BT application during application execution, P_{NODE}, application checkpointing, P_{NODE_C}, and application restarting, P_{NODE_R}. The system idle power, P_{idle}, was measured by recording system power use during execution of the Linux sleep command.

2) Checkpoint and Restart Measurements: Checkpointing and restarting of the BT application was performed on our target system using the Distributed Multithreaded Checkpointing (DMTCP) system [33]. The BT application was executed using 25 MPI ranks (making most efficient use of the Z820 system node), and checkpoint and restart times were measured natively by DMTCP. Checkpointing to RAM was measured by writing to and reading from a system RAM disk. Checkpoint time to and restart time from RAM were measured from an average of ten samples as $T_{C_{RAM}} = 0.34$ seconds and $T_{R_{RAM}} = 0.24$ seconds, respectively.

3) Node Performance: Measurement of the Z820 system node performance was accomplished using processor performance counters. Performance counters are built into the Intel Xeon E5-2697v2 hardware [31] and allow the user to monitor hardware events during an application's execution, with minimal impact on the application's performance [34]. The system node performance parameter (R_{FLOPS}) was calculated by monitoring the floating point operations executed by the BT application and averaging that total by BT's execution time.

D. Communication Power Model

The power used for system communication is calculated for each system node. Given a switch power of $P_S = 100$ watts, the number of nodes linked by a single switch (node concentration) of $NC = 12$ nodes, and given that a network interface controller of a single node consumes $NIC = 10$ watts at full utilization [35], the power spent by a single node for communication during computation is

$$P_{COMM_{Comp}} = \frac{P_S}{NC} + NIC\frac{T_{COMM}}{T_{COMP} + T_{COMM}}\,. \tag{6}$$

The power spent by a single node for communication during a time of high network traffic, such as during a checkpoint or restart (CR), is

$$P_{COMM_{CR}} = \frac{P_S}{NC} + NIC\,. \tag{7}$$

E. Resilience Technique Specific Parameters

In addition to the parameters measured for a system node's execution, each of the four resilience techniques implemented in the simulator requires its own set of parameters to govern the technique's execution.

1) Checkpoint Restart Parameters: Checkpoint Restart reads and writes its checkpoint data to a parallel file system. For our experiments, we assume an equal checkpoint and restart time using a parallel file system of $T_{C_{PFS}} = T_{R_{PFS}} = 20$ minutes, as observed in prior work [3]. The optimal checkpoint period, τ, is derived from the system's checkpoint time and failure rate according to [36] as

$$\tau = \sqrt{\frac{2T_{C_{PFS}}}{\lambda_{sys}}} - T_{C_{PFS}}\,. \tag{8}$$

2) Multilevel Checkpointing Parameters: Our experiments use a simulated Multilevel Checkpointing system of three levels. The first level writes checkpoints to the node's local RAM, the second level writes checkpoints to RAM in a partner node, and the third level checkpoint is written to a shared parallel file system. Additionally, whenever a higher level checkpoint is taken, all lower level checkpoints are taken simultaneously, with lower level checkpoint time masked by the time of the higher checkpoints, as outlined in [4].

The time used for taking a level one checkpoint, $T_{C_{L1}}$, is $T_{C_{L1}} = T_{C_{RAM}}$ and time for a level one restart, T_{RL1}, is $T_{R_{L1}} = T_{R_{RAM}}$. Checkpointing to and restarting from partner nodes are each expected to take about two minutes [2]. We use this same assumption for our simulation experiments, setting level two checkpoint and restart times to $T_{C_{L2}} = T_{R_{L2}} = 2$ minutes. Level three checkpoints to the parallel file system are defined to be twenty minutes. The optimal computation interval and number of lower level checkpoints taken before

taking a higher level checkpoint is determined using the Markov model in [4].

As outlined in Section III-B, multilevel checkpointing requires knowing the probability of failures at each severity level. We derive these values from data presented in [4] as the level one, two, and three failure ratios $\frac{\lambda_{L1}}{\lambda_{L_{tot}}} = 0.308$, $\frac{\lambda_{L2}}{\lambda_{L_{tot}}} = 0.545$, and $\frac{\lambda_{L3}}{\lambda_{L_{tot}}} = 0.147$, respectively.

3) Parallel Recovery Parameters: For our simulation study, we assume that the number of system nodes available for parallel recovery after a failure is $\phi = 8$. We assume that this level of parallelism will reduce the time for recovery by a factor of $\sigma = 7.26$, as measured in [2]. The temporary slowdown of the application due to load imbalance after recovery and until the next checkpoint is assumed to be $\Lambda = \frac{\phi+1}{\phi}$. As with multilevel checkpointing, the checkpoint and restart time for parallel recovery from RAM is two minutes.

4) Partial Redundancy Parameters: All parameters associated with the Partial Redundancy resilience technique remain the same as the Checkpoint Restart technique, except for the inclusion of the parameter r, indicating the system's degree of redundancy. We vary the parameter r from $1.25\times$ to $2\times$ redundancy based on the experiment being performed, as described in Section V.

F. Simulated System Setup

Studies are performed using three different simulated systems: small scale, sunway, and exascale. Values for the parameters of each simulated system are presented in Table I.

1) Small Scale System: The *small scale* system is a homogeneous system composed of 10,923 nodes. The small scale system's nodes are based on the same HP Z820 machines that were used in the real-world system measurements in Section IV-C. Measurements for network latency and bandwidth were taken from [37]. Parameter values for the small scale system are listed in the second column of Table I. The power and performance values listed are averaged over ten samples, and are rounded to the nearest integer.

2) Sunway System: The *sunway* system is our recreation of a system capable of performing similar to China's Sunway TaihuLight supercomputer, the world's highest performing system as of June 2016 [1]. The third column of Table I shows the system parameters we use to simulate the sunway system. While the sunway system has a theoretical peak performance of 125 petaflops, we run our experiments with the simulated system node performance parameter (R_{FLOPS}) calculated from the sunway's maximum measured performance of 93 petaflops. We have assumed a linear increase in power consumption of checkpointing, restarting, and idling from the small scale system to the sunway system. We also assume that the time that it takes for a sunway system node to checkpoint and restart are the same as the values used for our small scale system. All other values for the sunway system's simulation parameters are taken from [38].

3) Exascale System: Similar to the sunway system the *exascale* system is a homogeneous system. Each system node is similar in architecture to the sunway system, but we assume that at the time of an exascale machine's construction the number of computation processing entities per node will have doubled from 256 to 512, with each node still having an additional four cores used for both computation and node management. The number of system nodes is then scaled to perform at an exascale level. Parameter values for the exascale system are displayed in the fourth column of Table I. With the exascale system, we assume that node power consumption will continue to decrease, allowing system node power to remain the same as for the sunway system, despite the increase in the number of cores per node. We also assume internode communication will continue to improve, allowing an exascale system to have a network latency $L = 0.8\mu$s and a bandwidth $BW = 192$ Gb/s. Using the scaled value for node performance, an instance of the BT application executing at a problem size requiring 134,217,728 MPI ranks would require 260,112 system nodes, and operate at 1.2 EFLOPS.

TABLE I
SIMULATED SYSTEM PARAMETERS

parameter	small scale	sunway	exascale
P	262,144	10,647,169	134,217,728
N_{sys}	10,923	40,960	260,112
N_{cores}	24	260	516
R_{FLOPS}	27 GFLOPS	2271 GFLOPS	4507 GFLOPS
P_{NODE}	312 watts	375 watts	375 watts
P_{NODE_C}	278 watts	334 watts	334 watts
P_{NODE_R}	278 watts	334 watts	334 watts
P_{idle}	127 watts	153 watts	153 watts
L	1.2μs	1μs	0.8μs
BW	40 Gb/s	96 Gb/s	192 Gb/s
TS	100	100	100
performance	295 TFLOPS	93 PFLOPS	1.2 EFLOPS

P: number of processor cores (MPI ranks) used by BT;
N_{sys}: number of system nodes;
N_{cores}: number of cores per node;
R_{FLOPS}: performance of a node;
P_{NODE}: power consumed by a node during computation;
P_{NODE_C}: power consumed by a node when checkpointing;
P_{NODE_R}: power consumed by a node when restarting;
P_{idle}: power consumed by a node when idle;
L: network communication latency;
BW: network communication bandwidth;
TS: timesteps executed by BT (determines application execution time).
performance: calculated total system performance

V. SIMULATION STUDIES

A. Overview

We utilized our simulation environment to conduct two sets of studies. Each study simulated the BT application according to the methodology presented in Section IV. We refer to each independent simulation of a study as a *trial*. The outcomes of each trial vary based on the randomness associated with modeling failures outlined in Section III-B For each study we executed 200 independent trails, recording in each trial the amount of simulated time it took the BT application to run to completion, as well as the energy used by the system

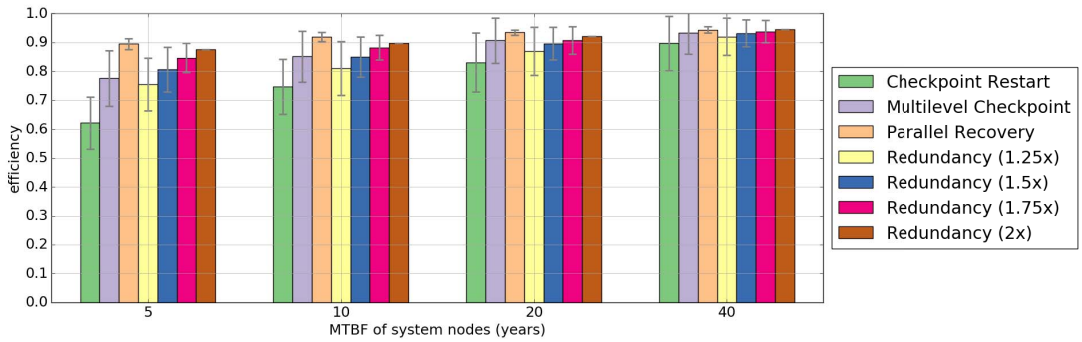

Fig. 1. Resilience technique efficiency at various levels of system node reliability. Efficiency is defined to be the ratio of an application's time without slowdowns (from failures or checkpointing) over the application's execution time with slowdowns (from failures or checkpointing). Each bar in the figure represents the average of 200 trials. Standard deviations are shown for each bar.

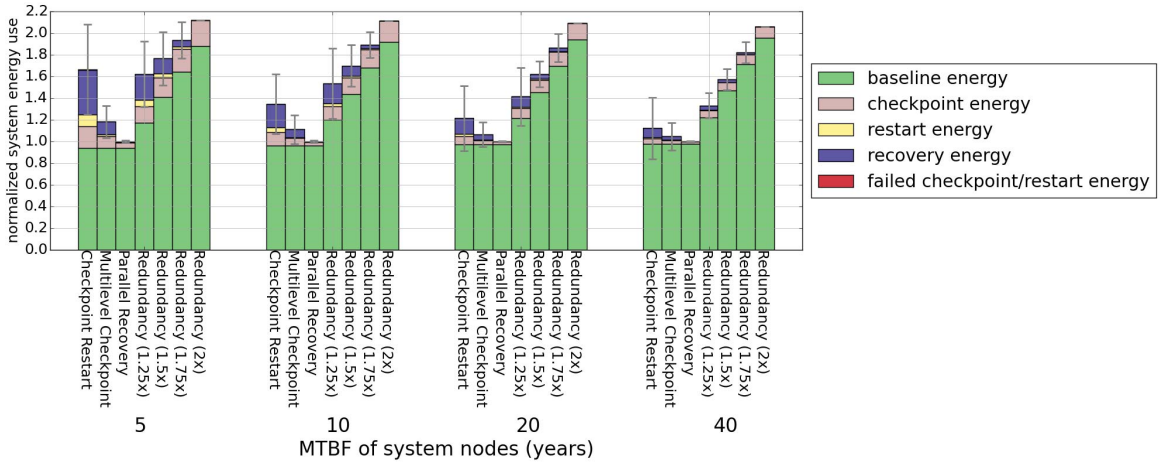

Fig. 2. Resilience technique energy use for four different levels of system node reliability. The height of each bar in the figure depicts the normalized total energy consumed by each technique for each study. The colors of each bar represent how system energy was consumed. Total energy consumption is normalized to the Parallel Recovery technique at each respective level of system node reliability. Each bar in the figure represents the average of 200 trials. Standard deviations of each technique's normalized total energy consumption are also shown for each bar.

during computation, checkpointing, restarting, recovering after restarts, and failed checkpoints and recoveries.

B. System Node Reliability

The first set of studies focuses on investigating how each of the four resilience techniques behaves under varying levels of system node reliability. We ran each simulation on the small scale system, varying the system node reliability by modifying the $MTBF_{Node}$ parameter to values of 5, 10, 20, and 40 years of mean time between failures. We tested each level of system reliability with each of the four resilience techniques. For the Partial Redundancy technique, we explored four different degrees of system redundancy, $1.25\times$, $1.5\times$, $1.75\times$, and $2\times$. The results of the experiments are shown in Figures 1 and 2. In each figure, groupings of bars along the x-axis indicate the different values of $MTBF_{Node}$.

Figure 1 shows the *efficiency* of each resilience technique. Efficiency is defined to be the ratio of an application's time without slowdowns (from failures or checkpointing) over the application's execution time with slowdowns (from failures or checkpointing). The color of each bar indicates the resilience technique in the experiment, and the height of each bar indicates the average efficiency of the trials.

Figure 2 shows the normalized system energy consumption of the same experiments shown in Figure 1. Each bar in Figure 2 has been normalized to the average value of the Parallel Recovery experiment data in each $MTBF_{Node}$ grouping. The colors of each bar indicate the breakdown of energy consumed in each experiment, with the colored portions of each bar representing the portion of total energy used for each type of simulated event defined in Section III. The height of each bar indicates the average of the total normalized energy consumed for the trials in the experiment.

Comparing the results in Figure 1 and Figure 2 it can be observed that the Parallel Recovery technique provides the most efficiency for every node reliability level except $MTBF_{Node} = 40$, and uses the least amount of energy for all

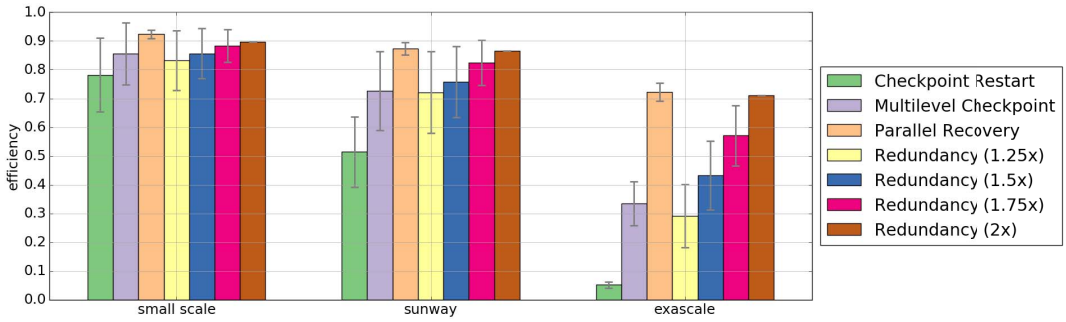

Fig. 3. Resilience technique efficiency at various system sizes. Efficiency is defined to be the ratio of an application's time without slowdowns (from failures or checkpointing) over the application's execution time with slowdowns (from failures or checkpointing). Each bar in the figure represents the average of 200 trials. Standard deviations are shown for each bar.

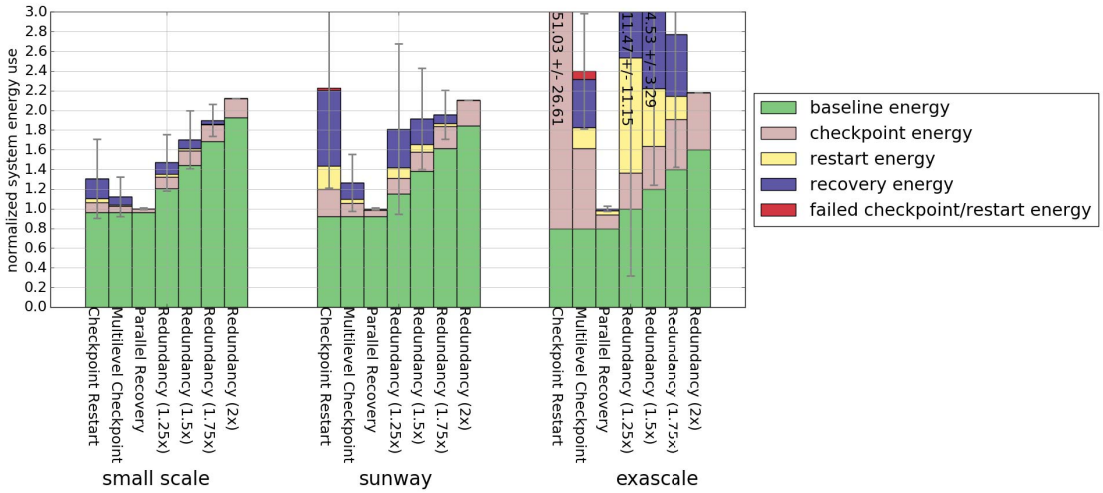

Fig. 4. Resilience technique energy use at various system sizes. The height of each bar in the figure depicts the normalized total energy consumed by each technique. The colors of each bar represent how system energy was consumed. Total energy consumption for each system size is normalized to the Parallel Recovery technique. Each bar in the figure represents the average of 200 trials. Standard deviations of each technique's normalized total energy are shown for each bar. Annotations in the subset of the exascale results represent each truncated bar's average value and standard deviation.

node reliability levels. However, while the Partial Redundancy technique is more efficient in one case, the system efficiency provided by the Partial Redundancy technique clearly comes at a high energy cost, due to the increase in total system nodes necessary to utilize the technique.

As is to be expected, all resilience techniques result in an increase in efficiency with more reliable system nodes (increasing values of $MTBF_{Node}$). From the standard deviations of each experiment, it can be observed that Parallel Recovery and Partial Redundancy with a redundancy degree of $2\times$ are much more consistent (have smaller standard deviation values) than the other resilience techniques.

C. System Size Scalability

The second set of simulations investigates how each of the four resilience techniques behave as the system size increases. We simulated each of the resilience techniques from the studies in Section V-B, but varied the simulated system among the three systems described in Section IV-F. A node mean time between failures of $MTBF_{Node} = 10$ years was used for all simulations. The results from our system size scalability simulations are in Figure 3 and Figure 4, with each figure's organization similar to Figure 1 and Figure 2, except that bar groupings indicate system size instead of node reliability. Each of the results in Figure 4 has been normalized to the result for the Parallel Recovery technique. The Checkpoint Restart experiments and the Partial Redundancy experiments with a degree of redundancy of $1.25\times$ and $1.5\times$ require an excessive amount of energy and the bars depicting their results have been truncated, with the averages and standard deviations of their respective sets of trials indicated on each respective bar.

Examining the results in Figures 3 and 4 it is clear that only the Parallel Recovery technique and the Partial Redundancy technique with a degree of redundancy of $2\times$ are capable of scaling to exascale system sizes with any reasonable execution efficiency. However, Partial Redundancy is outperformed by Parallel Recovery for every system size, and use of the Partial

Redundancy technique also comes at a high energy cost relative to Parallel Recovery. Examining the results shown for the sunway system, it can be observed that even for today's largest HPC system sizes (i.e., sunway) the energy cost of using traditional checkpointing is larger than for Partial Redundancy, despite the high energy cost of using additional (redundant execution) nodes with Partial Redundancy. Checkpoint Restart also provides far less benefit than the other techniques from an efficiency standpoint, indicating that there exist better alternatives to the traditional Checkpoint Restart technique even for HPC systems being used today.

VI. Conclusions

We performed simulations comparing the performance and energy use of four HPC and cloud computing resilience techniques. The techniques considered are the traditional Checkpoint Restart technique, as well as Multilevel Checkpointing, Parallel Recovery, and Partial Redundancy, three techniques proposed for next generation large-scale HPC systems. Our results indicate that Parallel Recovery and Partial Redundancy with a degree of redundancy of $2\times$ are the best at maintaining the performance of executing applications, both with decreasing system node reliability, and increasing system sizes, from today's largest computers through to exascale-sized machines. When comparing system energy efficiency, the Parallel Recovery technique was the most energy efficient in all system sizes and node reliabilities.

Acknowledgment

This work is supported by the National Science Foundation (NSF) under grants CCF-1252500 and CCF-1302693. This research also used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386.

References

[1] "Top500 June 2016," accessed Aug. 2016.

[2] 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, pp. 2061–2074, July 2015.

[3] F. Cappello, "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," *Int'l Journal of HPC Applications*, vol. 23, no. 3, pp. 212–226, 2009.

[4] 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., 2010.

[5] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for HPC," in *Int'l Conf. on Dist. Computing Systems*, pp. 615–626, 2012.

[6] "NAS parallel benchmarks," accessed Aug. 2016.

[7] F. Cappello, A. Geist, B. Gropp, L. Kalé, B. Kramer, and M. Snir, "Toward exascale resilience," *Int'l Journal of HPC Applications*, 2009.

[8] 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, no. 3, pp. 1302–1326, 2013.

[9] S. Limam and G. Belalem, "A migration approach for fault tolerance in cloud computing," *Int'l Journal of Grid and HPC*, vol. 6, no. 2, pp. 24–37, 2014.

[10] D. P. Jasper, "A discussion of checkpoint restart," *Software Age*, vol. 3, no. 10, pp. 9–14, 1969.

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

[12] C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, and F. Cappello, "Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI," in *Conf. on Supercomputing*, 2006.

[13] X. Ni, E. Meneses, and L. V. Kalé, "Hiding checkpoint overhead in HPC applications with a semi-blocking algorithm," in *Int'l Conf. on Cluster Computing*, pp. 364–372, 2012.

[14] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello, "Uncoordinated checkpointing without domino effect for send-deterministic MPI applications," in *Int'l Par. Dist. Proc. Symp.*, pp. 989–1000, May 2011.

[15] 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. 32:1–32:32, 2011.

[16] K. Mohror, A. Moody, G. Bronevetsky, and B. de Supinski, "Detailed modeling and evaluation of a scalable multilevel checkpointing cystem," *IEEE Trans. Par. and Dist. Systems*, vol. 25, pp. 2255–2263, Sep. 2014.

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

[18] L. A. B. 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, 2010.

[19] 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.

[20] 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*, to appear 2016.

[21] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 63–75, Feb. 1985.

[22] D. B. Johnson and W. Zwaenepoel, "Recovery in distributed systems using asynchronous message logging and checkpointing," in *Symp. on Principles of Dist. Computing*, pp. 171–181, 1988.

[23] J. F. Wakerly, "Microcomputer reliability improvement using triple-modular redundancy," *Proc. of the IEEE*, vol. 64, no. 6, 6 pp., 1976.

[24] S. Hukerikar, P. C. Diniz, and R. F. Lucas, "A case for adaptive redundancy for HPC resilience," in *Euro-Par 2014: Par. Proc. Workshops*, pp. 690–697, 2014.

[25] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, "Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system," *Sustainable Computing: Informatics and Systems*, vol. 5, pp. 14–30, 2015.

[26] 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*, to appear 2016.

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

[28] G. Zheng, L. Shi, and L. V. Kalé, "FTC-Charm++: An in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI," in *Int'l Conf. on Cluster Computing*, pp. 93–103, Sept 2004.

[29] R. F. Van der Wijngaart, S. Sridharan, and V. W. Lee, "Extending the BT NAS parallel benchmark to exascale computing," in *Int'l Conf. on HPC, Networking, Storage and Analysis*, pp. 94:1–94:9, 2012.

[30] "HP Z820 workstation," Tech. Rep. 4AA4-0130ENUC, April 2015.

[31] "Intel Xeon E5-2697v2 processor," accessed Aug. 2016.

[32] "Watts Up? PRO plug load meters," accessed Aug. 2016.

[33] J. Ansel, K. Arya, and G. Cooperman, "DMTCP: Transparent checkpointing for cluster computations and the desktop," in *Int'l Par. and Dist. Proc. Symp.*, 12 pp., May 2009.

[34] "Intel 64 and IA-32 architectures software developer's manual," Tech. Rep. 325462, Dec. 2015.

[35] 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, 2010.

[36] J. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, 2006.

[37] "Infiniband performance," accessed Aug. 2016.

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