Using Simulation to Evaluate the Performance of Resilience Strategies at Scale

Scott Levy, Bryan Topp and Dorian Arnold
Department of Computer Science
University of New Mexico
{slevy,betopp,darnold}@cs.unm.edu

Kurt B. Ferreira and Patrick Widener
Scalable System Software
Sandia National Laboratories
{kbferre,pwidene}@sandia.gov

Torsten Hoefler
Computer Science Department
ETH Zürich
htor@inf.ethz.ch

***Abstract*—Fault-tolerance has been identified as a major challenge for future extreme-scale systems. Current predictions suggest that, as systems grow in size, failures will occur more frequently. Because increases in failure frequency reduce the performance and scalability of these systems, significant effort has been devoted to developing and refining resilience mechanisms to mitigate the impact of failures. However, effective evaluation of these mechanisms has been challenging. Current systems are smaller and have significantly different architectural features (e.g., interconnect, persistent storage) than we expect to see in next-generation systems. To overcome these challenges, we propose the use of simulation. Simulation has been shown to be an effective tool for investigating performance characteristics of applications on future systems. In this work, we: identify the set of system characteristics that are necessary for accurate performance prediction of resilience mechanisms for HPC systems and applications; demonstrate how these system characteristics can be incorporated into an existing large-scale simulator; and evaluate the predictive performance of our modified simulator. We also describe how we were able to optimize the simulator for large temporal and spatial scales–allowing the simulator to run 4x faster and use over 100x less memory.**

I. Introduction

Fault-tolerance has been identified as a major challenge for exascale-class systems. As systems grow in scale and complexity, failures become increasingly likely; impacting performance and scalability. Current predictions suggest that for next-generation systems the mean time between failures could fall to hours, or even minutes [1]. As failure rates increase, more time is spent preparing for and recovering from failures and less time is spent doing (useful) application work. Given these dire predictions and the dynamics of fault-tolerance techniques, significant effort has been and is being devoted to investigations aimed at improving system resilience and related mechanisms.

Effective evaluation of fault-tolerance strategies on extreme-scale systems has been difficult for several reasons. Most significantly, researchers often need to study machines that either: are larger than those that are currently available; or have hypothetical architectures or configurations. As a result, existing systems are not sufficient to evaluate the performance impact of fault tolerance techniques on next-generation extreme-scale systems. Tests performed on these systems cannot accurately account for the impact of scale and may not be able capture the impact of architectural features (e.g. interconnect technologies) whose performance varies dramatically from current systems. Second, the largest and most advanced current machines generally are not accessible to most researchers. Third, analytic techniques for predicting performance in future systems are lacking. While good models for coordinated checkpointing exist [2], [3], we lack analytical tools for predicting the performance impact of many other fault tolerance mechanisms (for example, message-logging [4], communication-induced checkpointing [5] and hierarchical checkpointing [6]).

The broader objective of this project is to study general fault-tolerance techniques and their impacts on application performance. However, for the work presented in this paper, we focused on *checkpoint/restart*. Checkpoint/restart (or *rollback recovery*) is the technique most commonly used on today's systems. During normal operation, checkpoint/restart protocols [7] periodically record the state and address space of all application processes to stable storage devices. When a process fails, a new incarnation of the failed process is *recovered* from the most recent checkpoint – therefore limiting lost work. For distributed applications, coordinated checkpointing pauses all processes to record a globally consistent snapshot of the application's state. Uncoordinated checkpointing protocols avoid synchronization overheads and I/O contention by allowing each process to checkpoint independently. Uncoordinated checkpointing protocols also avoid rolling back non-failed processes. While there have been a number of studies which show that the overheads of checkpoint/restart could be prohibitively expensive for future extreme-scale systems [8]–[10], there has been a great effort in the research community to optimize these rollback/recovery protocols to ensure they remain viable [4], [9], [11]–[17].

Researchers have shown that simulation is an effective tool for investigating the performance characteristics of applications on current and hypothetical future systems [9], [18]–[20]. In this paper, we focus on efficient simulation of the impact of coordinated and uncoordinated checkpoint/restart protocols on application performance. Our approach is motivated by two observations: (1) simulation can be very computationally expensive, and simulation efficiency is maximized by considering only the features of the computing environment that are relevant to the performance impact of checkpoint/restart; and (2) the coarse-grained operation of checkpoint/restart (on the order of minutes to hours) allows us to forego the over-

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

heads and complexities of cycle-accurate simulation. Based on these observations, we hypothesize that like operating system noise [21], [22], resilience mechanisms (e.g., writing checkpoints, restarting after a failure or redoing lost work) can be modeled as CPU detours. A CPU detour is a number of CPU cycles that are used for something other than the application.

In this work, we provide a principled approach to simulating checkpoint/restart based fault-tolerance for large-scale HPC systems in a failure-prone environment. Based on this approach, we also present an efficient and accurate framework for simulating the performance impact of coordinated and uncoordinated checkpoint/restart protocols for existing and hypothetical extreme-scale systems and applications. Specific contributions of the work include:

- A survey of system, application, and resilience characteristics required for accurate and efficient simulation of extreme-scale workloads in a failure-prone environment;
- A prototype checkpoint/restart simulation framework, based on functional and performance-oriented extensions to `LogGOPSim` [20] which decrease memory consumption by over 100x and runtime by 4x;
- A validation of our hypothesis that resilience overheads can be modeled as CPU detours; and
- An evaluation of the predictive performance of our simulation approach showing an error of less than 3% against an analytic model for checkpointing in a failure-free environment.

The organization of this paper is as follows: in the next section, we discuss the relevant system, application, failure and resilience characteristics that must be considered by our framework. Additionally, this section offers more background on checkpoint/restart protocols and shows how they factor into our considerations. Section III provides an overview of `LogGOPSim`, the simulator that serves as the basis of our prototype. In Section IV, we describe the functional and performance-oriented extensions we made to `LogGOPSim` to improve its ability to simulate coordinated and uncoordinated checkpoint protocols at our desired time and space scales. In Section V, we evaluate the impact of our performance optimizations and validate the accuracy of the extended simulator for checkpoint/restart. We then discuss related works in Section VI, and finally, we summarize the impact our current contributions and plans for future work in Section VII.

II. Considerations for Resilience at Scale

Toward the goal of efficient, large-scale simulations that allow us to evaluate resilience techniques, we must identify the relevant hardware and software characteristics that impact simulation performance. We now describe our principled approach to identifying these characteristics: in turn, we consider system features, application behavior, fault-tolerance mechanisms and then the impact of failures.

A. Hardware Characteristics

Our objective is to develop a simulation framework that will enable us to evaluate resilience techniques on current and future systems. The simulator must be able to accurately and efficiently model the impact of faults and fault tolerance on application performance given the: (a) temporal scale, (b) spatial scale and (c) architectural features of next-generation extreme-scale systems.

1) Temporal Scale: Faults and fault tolerance mechanisms typically operate at large time-scales (for example, minutes, hours or even weeks). As we stated in the introduction, projected mean-time-to-interrupt (MTTI) on the first exascale machines are on the order of hours. Additionally, many of the target applications are long running, and the behaviors of the applications as well as the systems are expected to be dynamic. As a result, simulating resilience requires a simulator that can model relatively long periods of application execution.

2) Spatial Scale: The largest current HPC systems are comprised of tens of thousands of nodes. If current predictions hold, the first exascale system may be nearly an order of magnitude larger. Our simulator must be capable of modeling the behavior of systems that are much larger than any that are currently available.

3) Architectural Features: The first exascale system is not projected to appear until sometime after 2020 [23]. In the intervening span of years, we expect to see advances in interconnect and persistent storage technologies. Our simulator should also allow us to evaluate the impact of these advances on resilience mechanisms.

B. Application Characteristics

Our simulator must be capable of accounting for the performance aspects of the applications behavior. Prior research and experience has shown that it may be sufficient to do this at the course granularity of the target application's computation, specifically: its *communication graph*, a description of how processes communicate with each other; its *computation time*, the time between communication events; and its *dependencies*, a partial ordering of all communication and computation events. In the next section, we show how these characteristics interplay with resilience mechanisms.

C. Impact of Checkpoint/Restart Mechanisms

In checkpoint/restart protocols [24], the application or system saves snapshots of application state, *checkpoints*, to persistent storage. In coordinated checkpointing, all processes checkpoint at the same time (in order to mark a consistent global state), and in the event of a process failure, all processes must revert to their most recent checkpoint. While coordinated checkpoint/restart is the predominant approach, it suffers several limitations including increased overhead with system size and global process perturbations during checkpoint and recovery phases. Uncoordinated checkpoint/restart protocols, in which processes can checkpoint and recover independently, address these limitations – though they introduce new ones. In addition to these coarse protocols, many optimizations have been proposed including: diskless [25]–[27], hierarchical [6], [28] communication-induced [29] and incremental checkpointing [14], [30]. Despite the proliferation of resilience mechanisms, we lack effective methods for evaluating the true costs of each of these approaches on exascale systems [31].

Given the large temporal and spatial scales of the simulated systems that we wish to consider, effective simulation demands that we eliminate unnecessary detail. Existing work on modeling and simulation of coordinated checkpointing provides us with a guidepost on the required components and level of details [2], [3], [32].

In a failure-free environment, we can accurately model the impact of coordinated checkpointing by considering: the *checkpoint time*, amount of time that checkpointing activities prevent the application from executing, the *checkpoint interval*, time between consecutive checkpoints, and *work time*, the amount of time that the application would execute in the absence of checkpointing activities. Checkpoint time may need further refinement potentially including a process *coordination* phase, the *checkpoint calculation* phase during which time the checkpoint data is computed, the *checkpoint commit* time to write the checkpoint to stable storage and the *resumption* phase to continue normal application execution.

For approaches like uncoordinated checkpointing that lack explicit coordination, we also need to consider the application characteristics like communication patterns described previously. Consider a simple uncoordinated checkpointing strategy where each process generates checkpoints strictly according to local policies. Communication dependencies may cause checkpointing activities in one process may perturb the behavior and performance of other processes. For example, if the recipient of a message is currently busy generating a checkpoint then reception of the message may be delayed until the checkpoint is complete. Further, all actions that are dependent on the reception of the message will also be delayed. Additionally, many asynchronous resilience techniques use message logging [24] to mitigate recovery costs. Accounting for this activity also requires that we incorporate information about communication patterns into our simulation.

D. Impact of Failures

Meaningful evaluation of resilience mechanisms necessarily includes the introduction of failures. Initially, we consider only fail-stop failures. To accurately simulate the impact of the occurrence of failures on application performance, at a minimum, we need to consider: (a) failure characterization; (b) restart time; and (c) recovery description.

1) Failure Characterization: To evaluate the impact of faults in the context of a resilience mechanism, we require a description of how failures occur in the simulated system. Although this could be expressed in many ways, the most common and succinct description of failure occurrences is in the form a probability distribution.

2) Restart Time: When a failure occurs, some time elapses before any computation can be undertaken on the failed node. To account for this fact, we need to know the time between the occurrence of a failure and the moment when the failed node can resume computation. This includes time to restart failed nodes and processes and to read checkpoints from persistent storage, but does not include any time for recovery. For example, in the case of coordinated checkpointing, the end of the restart interval coincides with the beginning of rework (i.e., redoing work lost due to the failure).

3) Recovery Model: When the failed node has restarted and is able to resume computation, there is typically some amount of work that needs to be redone before the system can again make meaningful forward progress. For example, in coordinated checkpointing, all of the computation between the last valid checkpoint and the occurrence of the failure needs to be redone. Typically, each resilience mechanisms presents a different method for recovering from a failure. Therefore, to accurately account for the cost of recovering from a failure, we need a model for each resilience mechanism that allows us to determine the amount of time that will elapse before the application resumes forward progress.

III. LogGOPSim

In this section, we describe `LogGOPSim` [20], [33], the simulator we extend to meet the requirements prescribed by the considerations in Section II. We choose `LogGOPSim` because it is shown to be accurate, freely available and fast enough to support large-scale simulations while already capturing many of the application and hardware characteristics we require (as we discuss). As described in Section IV, functionally, we simply needed to extend it to account for checkpoint/restart and failure recovery.

A. Simulating Application Characteristics

`LogGOPSim` is an application simulator based on the popular LogP model [34]. LogP and its variants have a long history of accurately predicting the performance of large-scale parallel applications and algorithms. The simulation framework consists of three major components: a trace collector (`liballprof`), a schedule generator (`SchedGen`), and an optimized discrete-event simulator (`LogGOPSim`).

The trace collector records the actual MPI communication of the target application. The schedule generator uses the MPI traces to generate a schedule that captures the required characteristics of control- and dataflow of the application while preserving the happens-before relationship of events within the application. The discrete-event simulator reads the generated schedule, performs a full LogGOPS simulation and reports the completion time of each process.

This validated simulation framework was developed to simulate applications at scale, and has the ability to extrapolate from traces collected on smaller scale systems. This allows for the simulation of platforms larger than those currently in existence while keeping the same communication characteristics (equivalent to weak-scaling of the application). Although the extrapolated trace may not precisely represent the communication pattern on the larger system, the impact of this inaccuracy has been shown to be small [20] if extrapolation factors are bounded. This framework has been used to evaluate the performance of collective communications [35] and the impact of OS noise [22] on large-scale applications. A detailed study of the simulation framework and its functionality is presented in [20].

B. Simulating Hardware Characteristics

Because `LogGOPSim` was initially developed to model application performance in large-scale systems [22], it allows us to model systems with the characteristics described

Required to Model	Parameter Name	Parameter Description
All Checkpointing	COORDINATION TIME	time for processes to coordinate the taking of a checkpoint
	CHECKPOINT COMPUTATION	time to compute a checkpoint
	CHECKPOINT COMMIT TIME	time to write a checkpoint to stable storage
	CHECKPOINT INTERVAL	time between consecutive checkpoints
	WORK TIME	time-to-solution without failures or resilience mechanisms
Uncoordinated Checkpointing	COMMUNICATION GRAPH	details of inter-process communication
	COMPUTATION EVENTS	failure-free computation pattern of the application
	DEPENDENCIES	partial ordering of communication and computation events
Failure Occurrences	FAILURE CHARACTERIZATION	rate and distribution of failures
	RESTART TIME	time to read a checkpoint from stable storage after a failure
	RECOVERY MODEL	a model of the time required before forward progress can resume

TABLE I. SUMMARY OF THE PARAMETERS NEEDED FOR ACCURATE SIMULATION OF HPC APPLICATIONS IN A FAILURE-PRONE SYSTEM.

in the preceding section. First, it provides the simulation scale necessary for evaluating checkpointing techniques. For a single collective operation, `LogGOPSim` can simulate up to 10,000,000 processes. For more general workloads, it is capable of simulating more than 64,000 processes.

Second, with some minor modifications, `LogGOPSim` is also capable of simulating the necessary temporal scale. The initial implementation of `LogGOPSim` was intended for comparatively short simulations. As a result, the temporal scope of the simulations that can be executed by the unmodified simulator is significantly limited by the size of the simulating system's memory. To achieve the temporal scale that we needed with reasonable quantities of system memory, we made some simple modifications to `LogGOPSim`. These modifications are discussed more fully in a subsequent section.

Third, `LogGOPSim` also allows us to model the impact of emerging interconnect technologies. Working within the LogGOPS model, we can simulate the impact of many changes in network behavior on resilience techniques by modifying the model's parameters. In addition, as we discuss more fully below, our model of resilience mechanisms allows us to evaluate how improvements to persistent storage systems (e.g., the widespread availability of local SSDs) will affect the performance of resilience mechanisms.

IV. EXTENDING LOGGOPSIM FOR LARGE SCALE RESILIENCE RESEARCH

A. Simulating Failures and Resilience

The key insight that allows us to use `LogGOPSim` is that resilience mechanisms (e.g., writing checkpoints, restarting after a failure, redoing lost work) can be modeled as CPU *detours*. A CPU detour is a number of CPU cycles that are used for something other than the application, similar to OS noise [21], [22]. One key difference between OS noise and these resilience detours is that resilience "noise" events may need to be replayed synchronously with the application communication/computation pattern rather than asynchronously as is typical of OS noise.

We model resilience in `LogGOPSim` using a new library, `libsolipsis`, that generates CPU detours based on a specified resilience mechanism and the application's communication pattern. Similar to `liballprof`, the library links to the application using the MPI profiling interface, intercepting all MPI calls. The output of this library is a per-process detour file that can be provided as input to `LogGOPSim`. The detour file contains the timestamp and the duration of each of the resilience mechanism detours. The duration of detours, T_{detour}, that represent checkpoints are computed using the following expression.

$$T_{detour} = T_{coord} + T_{ckpt} + T_{commit}$$

where

$$\begin{aligned} T_{coord} &= \text{time to coordinate the taking of a checkpoint} \\ T_{ckpt} &= \text{time to compute a checkpoint} \\ T_{commit} &= \text{time to commit the checkpoint to stable storage} \end{aligned}$$

We also generate detours to represent the impact of node failure and optimistic message logging. In the case of failure, the duration of the detour includes both the restart and rework time on the failed node; `libsolipsis` computes the rework time by calculating the amount of simulated time that has elapsed since the previous checkpoint. For optimistic message logging, `libsolipsis` calculates the time required to write the message to the log given a bandwidth to stable storage.

For the purposes of this work, we focus on the libraries' ability to emulate performance of two popular resilience mechanisms: coordinated checkpointing and asynchronous checkpointing with message logging [7].

We focus on these on these two methods because coordinated checkpoint/restart is currently the most popular approach and asynchronous checkpointing has been proposed as a low-overhead checkpoint option for future extreme-scale systems.

For asynchronous checkpointing with message logging, our library generates detour files that contain the timestamp and the duration of the local checkpoints. Because no coordination is required, $T_{coord} = 0$. Also, for simplicity, we currently assume that $T_{ckpt} = 0$. For pessimistic message logging [7], we modify the CPU overhead parameter (o in the LogGOPS model) for send operations (o_s) to account for the log write to stable storage. The `LogGOPSim` simulator uses a single detour file to simulate the local checkpoints in the system; to model the asynchronous nature of these checkpoints, each node starts at a different location in the file.

For coordinated checkpoint/restart, the library generates a detour file that contains the timestamp and the duration of each checkpoint taken by the application. For this work, we have assumed bulk-synchronous parallel (BSP) applications. Because applications of this type are largely self-synchronizing, we set $T_{coord} = 0$. And again, for simplicity, we are currently assuming that $T_{ckpt} = 0$. When the simulation is run, we use the "`--noise-cosched`" option of the `LogGOPSim` simulator. This option ensures that the detour file is co-scheduled on all processors, thereby simulating coordinated checkpoint/restart. We also force each process to start at the beginning of the detour file to ensure proper timing of checkpoints.

To simulate failure, the library generates failure times for each node from a random distribution based on a per-node mean time between failure (MTBF). When a failure is generated, the library adds a detour event that includes the the time required to restart from the last checkpoint and the time required for rework (i.e., the time since the last checkpoint). The `LogGOPSim` simulator will ensure that all communication in the trace file that depends on the failed node will be delayed until the node has "recovered".

B. *Optimizing `LogGOPSim` for Scale*

To simulate periods of execution long enough to be meaningful for fault tolerance (i.e., application wallclock times long enough that application failures are expected) while keeping traces manageable, we extended `LogGOPSim` to support automatic execution trace extrapolation. Because `LogGOPSim` was originally designed to simulate single collective operations and short application traces it assumed a comparatively small input dataset. In our use cases, the extent to which the existing `LogGOPSim` could scale up the length of simulated execution and the number of simulated nodes was severely limited by the amount of available memory.

`LogGOPSim`, as originally published, requires a pre-processing step which performs the extrapolation to generate communication data for all simulated nodes. The simulator binary then attempts to map this file into virtual memory and use it directly as input data about simulated events. The size of this file is proportional to both the length of the simulated execution and the number of simulated nodes. As a result, simulating long running applications or large-scale systems requires very large data sets. Additionally, when collecting data at varying scales, a user would be required to re-run the entire toolchain from the trace data to the simulator with different parameters.

We re-wrote the input handling portion of `LogGOPSim` to include two critical changes. First, the modified simulator performs extrapolation in main memory as needed, rather than as a pre-processing step on disk. The traces generated from profiling an MPI application are used directly as input, and are proportional in size to the original profiled node count, rather than the extrapolated node count. Second, the simulator works on a small sliding window of input data, rather than mapping it in all at once. The code loads and extrapolates data from the traces at fine granularity, loading only a small portion of the trace file at a time. Because of these changes, the simulator's memory usage, shown in Fig. 1, remains constant independent of input trace size. In other words, the same amount of memory would be required to simulate a minute, hour, day, week or month of application execution time!

In theory, an operating system should be able to perform this type of efficient memory allocation when using a system call such as `mmap`. However, on the Linux 2.6.32 systems that we used, windowing the input data at the application level allowed for much greater scales before the system started to thrash.

V. Evaluating our `LogGOPSim` Extensions

A. *Correctness of the Extensions*

We have verified with a careful comparison of the sequences of simulation events generated by each of the two simulators that our modified `LogGOPSim` produces exactly the same sequence and timing of simulated events as the original validated `LogGOPSim` tools. Moreover, the two simulators produce identical simulated runtimes.

B. *Evaluating Performance Enhancements*

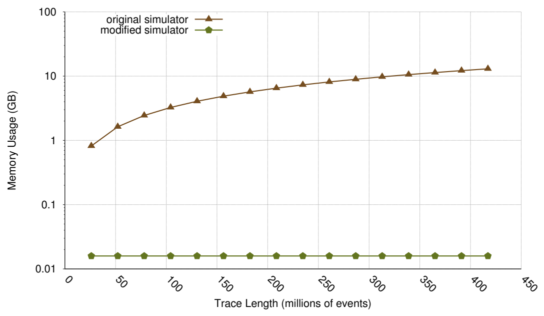
In this section, we evaluate the performance impact of our modifications to `LogGOPSim`. We consider two important metrics to evaluate our changes, maximum memory requirement and simulation performance in events/second, and then finally examine overall wallclock time for simulating the same problem

1) Memory Usage: With our changes, the amount of disk space needed is no longer proportional to the node extrapolation factor, and the amount of RAM needed is no longer proportional to the length of the trace data. This enables simulation of long executions of many nodes: for the traces used here, memory usage decreased dramatically as shown in Fig. 2. Memory usage dropped by a factor of 20 for HPCCG, 60 for LAMMPS, and 900 for CTH, with the magnitude of the drop related to an applications communication pattern and the greatest distance between the initiation of a non-blocking operation and waiting for its completion. This increase of available memory allowed us to simulate over 12 minutes of HPCCG at 256K nodes and over 7 minutes of LAMMPS at 256K nodes in a short amount of time, as shown in Figures 2(a) and 2(b), respectively.

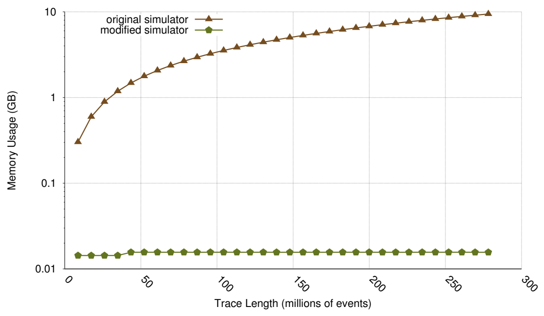
2) Simulation Performance: Fig. 3 and Fig. 4 show the increase in performance for our simulation framework. We show this increase both in terms of event per second of the simulator (Fig. 3) and the wall clock time to perform the simulation (Fig. 4). We see from these figures, a factor of 2.5 to 4X increase in performance from our modifications. We believe that the substation performance benefits stem from the smaller cache footprint of our implementation. We note that simulation performance decreases slightly as the number of simulated nodes increases. We are working on characterizing this decrease. However, we conclude that the achieved performance is sufficient for our purposes.

C. *Validating Checkpoint Simulation*

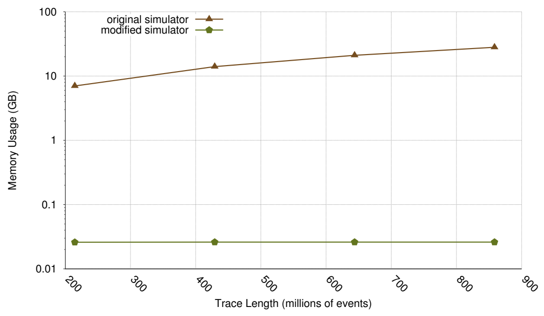
In this section, we present the data we collected to validate our simulator. We use both analytic models and small-scale testing to ensure that our simulator accurately models the impact of resilience mechanisms in failure-free and failure-prone environments.



(a) HPCCG

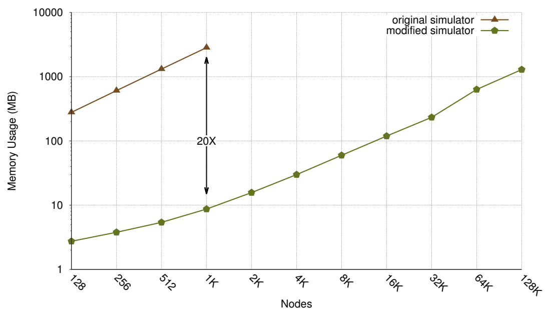


(b) LAMMPS

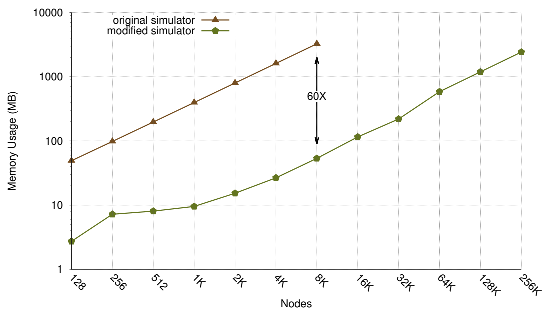


(c) CTH

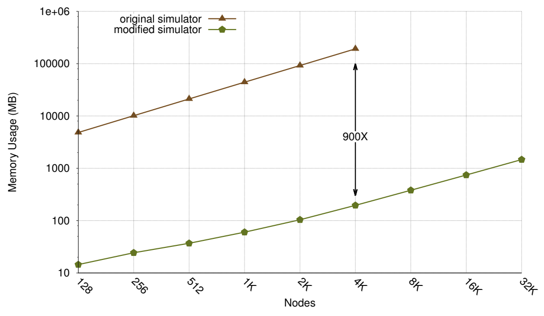
Fig. 1. Comparison of the memory consumption required to simulate a system running one of three applications using the original `LogGOPSim` simulator and our modified version as a function of input trace length. Our windowing protocol decouples memory usage from trace length. As a result, with a fixed memory budget, our modifications allows us to simulate much longer periods of applcation execution than was possible with the original simulator.



(a) HPCCG



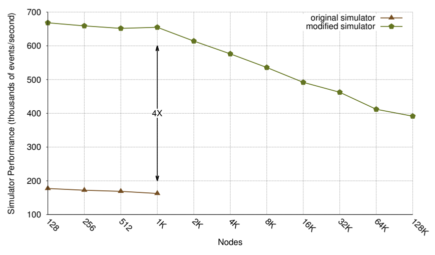
(b) LAMMPS



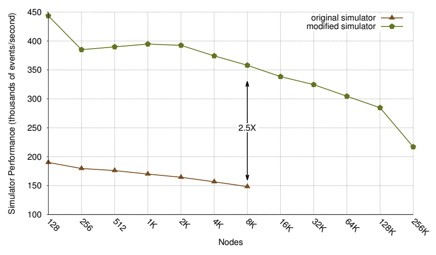
(c) CTH

Fig. 2. Comparison of the memory consumption required to simulate a system running one of three applications using the original `LogGOPSim` simulator and our modified version. With a fixed memory budget, our modifications allows us to simulate systems that are significantly larger than could be simulated with the original simulator. The memory consumption decrease varies by communication pattern and varies from 20X for HPCCG to 900X for CTH.

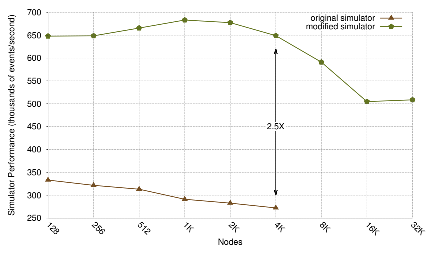
1) Failure-Free Analytic Model of Coordinated Checkpointing: We begin with a simple analytic model for coordinated checkpointing. Equation 1 models application performance in terms of its wall clock time-to-solution, T_w, in a failure-free



(a) HPCCG

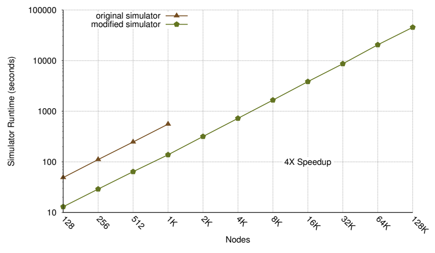


(b) LAMMPS

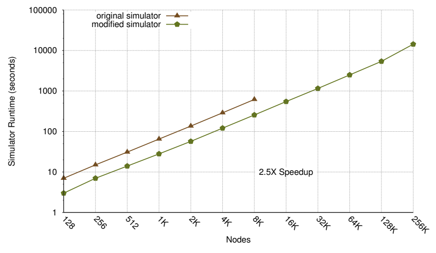


(c) CTH

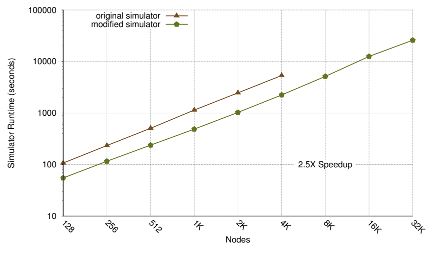
Fig. 3. Comparison of simulator performance, measured in events/second, when simulating a system running one of three applications using the original LogGOPSim simulator and using our improved version. Due to the memory requirements of the original simulator, we were unable to obtain results for simulations of large-scale systems using the original simulator. The simulation performance increase varies by application communication pattern and varies from 2.5X for LAMMPS and CTH to 4X for HPCCG.



(a) HPCCG



(b) LAMMPS



(c) CTH

Fig. 4. Comparison of simulator runtime, measured in seconds (lower is better), for the original LogGOPSim simulating systems running three different applications using simulator and using our improved version. Due to the memory requirements of the original simulator, we were unable to obtain results for simulations using the original simulator for large-scale systems. Similar to Fig. 3, the simulation wallclock speedup varies by application; from 2.5X for LAMMPS and CTH to 4X for HPCCG.

environment.

$$T_w = T_s + \frac{T_s}{\tau} \times \delta \tag{1}$$

where T_w is the wall clock time, T_s is the solve time of the application without any resilience mechanism, τ is the checkpoint interval [2], and δ is the checkpoint commit time (time to write one checkpoint). For coordinated checkpointing to shared stable storage, we can express the checkpoint commit time as:

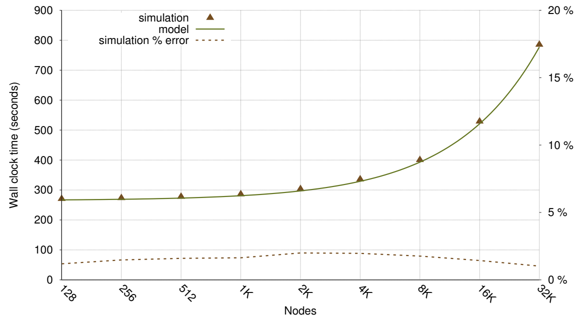
$$\delta = \frac{N * ||c_{avg}||}{\beta} \tag{2}$$

where N is the number of nodes, $||c_{avg}||$ is the average checkpoint size per node, and β is the aggregate write bandwidth to stable storage.

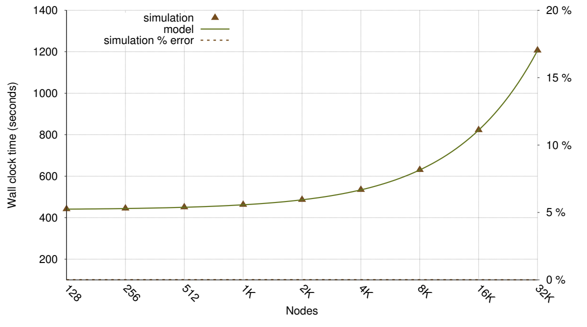
In Figures 5(a) and 5(b), we compare the output of this model to the output of our simulator. The times-to-solution for CTH predicted by the simulator are very accurate, about 3% greater than the model's predictions. More importantly, the simulator closely matches scaling trends predicted by the model. Moreover, the simulated times-to-solution for LAMMPS are within 1% of the analytic model. On the whole, these data suggest that the simulator is accurately modeling how the impact of resilience mechanisms scales with system size.

2) Small-scale testing: To further validate our simulator, we compared it against the results of small-scale tests on real hardware. The simulator provides us with fine-grained control over the checkpoint interval and duration. To mimic this degree of control on real hardware, we constructed an MPI profiling library, `libchkpt`. This library, based on the the `libhashckpt` incremental checkpointing library [14], also has the ability to take both full coordinated and uncoordinated checkpointing techniques, in additional to its incremental coordinated techniques. The full coordinated checkpointing functionality ensures all checkpoints are taken simultaneously on each node, while the uncoordinated approach takes checkpoints independently. While taking checkpoints, the CPU is taken from the application until the checkpoint commit time has completed.

For our purposes here, we focus on validating the failure-free case. Fig. 6 and Fig. 7 show the results of this validation. These figures compare the total wall clock time simulated by `LogGOPSim` and measured with `libchkpt` running on our test platform. For reference, each figure also includes the total wall clock time in the absence of any failures. Note the performance of CTH in Fig. 6 exhibits a distinct sawtooth pattern. This pattern is an artifact of how CTH scales the computation as nodes counts increase. The simulator accurately predicts this complex sawtooth pattern. We also see in this figure the simulators error in prediction. We also note that the predictive performance of the simulator is less accurate for CTH in comparison to LAMMPS, with the error in time to solution bounded by 20%. This is due to the simulator not accounting for OS noise on the node and limitations of the network model used. In our testing, OS interference is not being generated to simplify analysis, though the simulator allows for such accounting. This OS interfere has been shown to greatly influence impact CTH performance [21]. Also, though the `LogGOPSim` simulator is capable of sophisticated

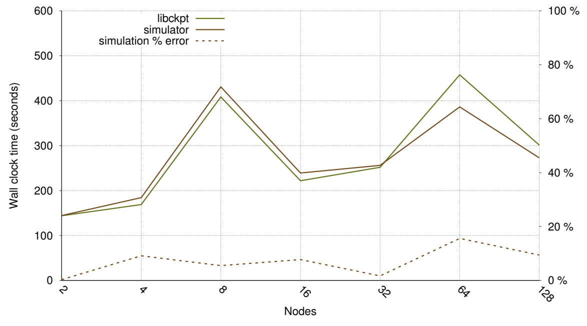


(a) CTH

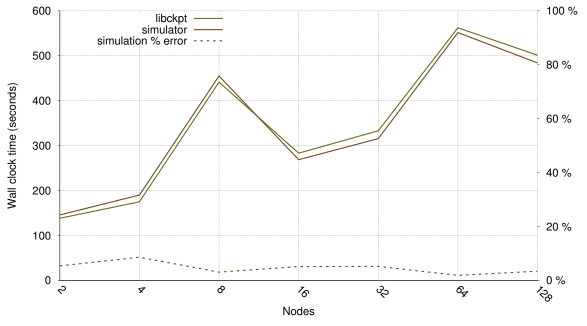


(b) LAMMPS

Fig. 5. Validation of the simulator against the simple analytic model described in Equation 1 for coordinated checkpointing to stable storage in a failure-free environment for CTH and LAMMPS. The model and the simulator use identical values for the T_s (for each application), τ, and δ. The simulation error is less than 3% for CTH and less than 1% for LAMMPS across the tested node count range.



(a) Coordinated Checkpointing



(b) Uncoordinated Checkpointing

Fig. 6. Performance of `LogGOPSim` simulation against a coordinated and uncoordinated checkpointing library for CTH. The simulator and `libchkpt` use identical values for T_w (failure free performance), τ (checkpoint interval), and δ (checkpoint commit time). The simulation error in this figure is less than 20%, with this differences attributed to platform features not being simulated. For example, interference from the OS is not being generated in this case to simplify analysis. This OS interfere has been shown to greatly influence impact CTH performance [21].

network models, in this work we use a simple network model which does not account for network contention. As CTH does a fair amount of bulk data transfer, network contention can be an issue.

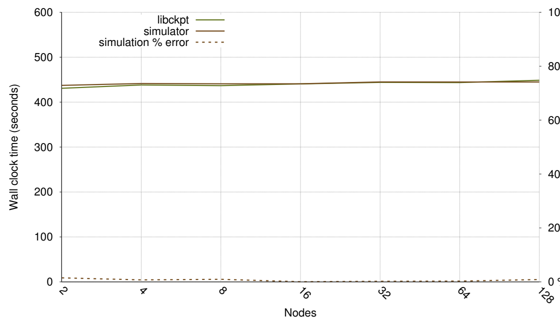
Overall, these figures show that `LogGOPSim` closely tracks the results measured with `libchkpt`. For all the configurations that we examined, the absolute wall clock time simulated by `LogGOPSim` is within 20% of the measured values. More importantly, `LogGOPSim` closely mimics the trends we observe with `libchkpt` even as performance deviates from performance on actual hardware.

VI. Related Work

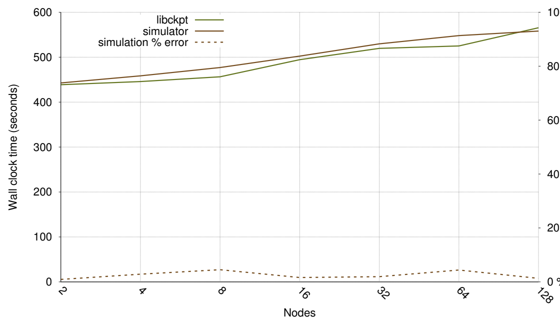
Although fault tolerance for HPC has been a very active area of research, few tools exist that allow us to project behavior beyond small-scale systems. As we discussed above, simulating fault tolerance techniques requires an appropriate level of detail about the communication of the target application. Without an accurate representation of application communication, we cannot accurately simulate some fault tolerance techniques (e.g., asynchronous checkpointing). Too much detail unnecessarily reduces simulator performance. The application simulators for fault tolerance that do exist tend to fall to either extreme; either they are not communication-accurate or they simulate communication in much greater detail than believed necessary.

In [32], Riesen et al. present a simulator that can model the impact of node failure on application performance in the context of traditional coordinated checkpoint/restart. This simulator can also account for process replication. Tikotekar et al. take a similar approach in [36]. They present a simulator that models coordinated checkpointing and can also simulate fault prediction and process migration. While these tools have been shown to be effective for their stated purposes, they are not communication-accurate. As a result, they are unable to account for fault tolerance techniques whose performance may be influenced by communication patterns.

At the other extreme is xSim [37]. xSim builds on the MPI profiling interface and interposes itself between the application and the MPI library. As a result, the simulator is able to run unmodified HPC applications. Scaling is achieved by oversubscribing the nodes of the system used for validation. While this provides us with a tremendous amount of detail about the performance of the application, it imposes a significant cost. Due to limits on the degree of oversubscription, large-scale systems are required to simulate systems that approach extreme-scale. Moreover, as the size of the simulated system grows and the degree of oversubscription therefore increases, the time required to simulate the system grows dramatically. Lastly, this oversubscription could place significant limits on



(a) Coordinated Checkpointing



(b) Uncoordinated Checkpointing

Fig. 7. Performance of `LogGOPSim` simulation against an coordinated and uncoordinated checkpointing library for LAMMPS. The simulator and `libchkpt` use identical values for T_w (failure free performance), τ (checkpoint interval), and δ (checkpoint commit time). The simulation error in this figure is shown to be less than 5% in the range tested.

the size of the problem that can be solved as the memory for each simulated node must exist in the memory of one physical node. In contrast, our approach allows us to simulate fault tolerance mechanisms for systems comprised of tens or hundreds of thousands of nodes on very modest hardware (e.g., a single node). In some cases, this simulation completes in less time than it would take to run the application itself, but with the less detail of the computation.

Boteanu et al. present a fault tolerance extension to an existing simulator in [38]. However, they target a datacenter environment where each job is a discrete unit that is assigned to a single processing element. As a result, their simulator does not map well to HPC workloads.

Finally, SST/macro [39], [40] is a coarse-grained, lightweight simulator designed to simulate the performance of existing and future large-scale systems. By collecting traces of application execution, SST/macro is able to simulate the application's computation and computation patterns at scales and on hardware that does not yet exist. However, SST/macro does not currently account for the impact of CPU detours (OS jitter). As a result, because the foundation of our approach is based on the observation that resilience can be modeled as CPU detours, we concluded that SST/macro was not a suitable starting point for our investigation.

VII. Conclusion & Future Work

We presented in this work, a new and promising approach to simulation at scale of fault-tolerance mechanisms based on the checkpoint/restart model. We identified a set of platform, application, and resilience characteristics required for accurate and efficient simulation; described a prototype framework based on extensions to a validated and freely-available application simulator implementing the LogP model; shown how resilience processing overheads can be effectively modeled as CPU detours; and demonstrated empirically that our approach accurately predicts, with an error of less than 3%, the impact of resilience mechanisms. Our modifications to the LogGOPSim simulator greatly decreased its memory consumption by a factor of 100 or greater and runtime by a factor of 4. This performance increase allows us to evaluate potential resilience solutions at meaningful application and temporal scales, while also enabling the modeling of future interconnection and storage technologies.

The design space for evaluating resilience methods in large-scale HPC applications is young and still evolving. While our simulation framework has expanded that space in new and useful ways, several areas for future work remain. Among these, we intend to extend the framework to provide failure injection for large-scale simulation. Additional failure types should also be modeled, e.g. corruption of application memory or network traffic. We understand that this may mean re-evaluating the granularity of our simulation to ensure proper and effective simulation. We are also investigating mechanisms to integrate both coarse- and fine-grained simulation for failures. This will allow us to use coarse-grained simulation in areas where failures do not occur, and fine-grained simulation when failures or other interesting events do occur. We also plan to address support for additional resilience mechanisms such as hierarchical checkpointing, replication-based approaches, process migration and cloning, as well as integration with on-going standards efforts like the current fault tolerance proposal put forth in the MPI forum [41]. Finally, we plan to further investigate performance limitations of our current simulation framework, including analyzing the benefit of parallelizing the simulator.

References

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

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

[3] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent, "A flexible checkpoint/restart model in distributed systems," in *Parallel Processing and Applied Mathematics*. Springer, 2010, pp. 206–215.

[4] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello, "Uncoordinated checkpointing without domino effect for send-deterministic MPI applications," in *International Parallel Distributed Processing Symposium (IPDPS)*, may 2011, pp. 989–1000.

[5] L. Alvisi, E. Elnozahy, S. Rao, S. Husain, and A. de Mel, "An analysis of communication induced checkpointing," in *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, 1999, pp. 242–249.

[6] S. Monnet, C. Morin, and R. Badrinath, "A hierarchical checkpointing protocol for parallel applications in cluster federations," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 211.

[7] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, “A survey of rollback-recovery protocols in message-passing systems,” *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375–408, 2002.

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

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

[10] B. Schroeder and G. A. Gibson, “A large-scale study of failures in high-performance computing systems,” in *International Conference on Dependable Systems and Networks (DSN)*, Jun. 2006.

[11] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic, “Optimizing checkpoints using nvm as virtual memory,” in *Proceedings of the nternational Parallel and Distributed Processing Symposium*, ser. IPDPS ’13. New York, NY,USA: ACM, 2013.

[12] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie, “Leveraging 3d pcram technologies to reduce checkpoint overhead for future exascale systems,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC ’09. New York, NY, USA: ACM, 2009, pp. 57:1–57:12. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654117

[13] G. Bronevetsky, D. Marques, K. Pingali, S. McKee, and R. Rugina, “Compiler-enhanced incremental checkpointing for openmp applications,” in *IEEE International Symposium on Parallel&Distributed Processing*, 2009, pp. 1–12.

[14] K. B. Ferreira, R. Riesen, R. Brightwell, P. G. Bridges, and D. Arnold, “Libhashckpt: Hash-based incremental checkpointing using GPUs,” in *Proceedings of the 18th EuroMPI Conference*, Santorini, Greece, September 2011 [to appear].

[15] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, “Design, modeling, and evaluation of a scalable multi-level checkpointing system,” in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’10)*, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.18

[16] D. Ibtesham, D. Arnold, P. G. Bridges, K. B. Ferreira, and R. Brightwell, “On the viability of compression for reducing the overheads of checkpoint/restart-based fault tolerance,” *2012 41st International Conference on Parallel Processing*, vol. 0, pp. 148–157, 2012.

[17] A. Guermouche, T. Ropars, M. Snir, and F. Cappello, “HydEE: Failure containment without event logging for large scale send-deterministic mpi applications,” in *IPDPS*. IEEE Computer Society, 2012, pp. 1216–1227.

[18] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns, “Modeling a million-node dragonfly network using massively parallel discrete-event simulation,” in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*. IEEE, 2012, pp. 366–376.

[19] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé, “Simulation-based performance prediction for large parallel machines,” *International Journal of Parallel Programming*, vol. 33, no. 2-3, pp. 183–207, 2005.

[20] T. Hoefler, T. Schneider, and A. Lumsdaine, “LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model,” in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, Jun. 2010, pp. 597–604.

[21] K. B. Ferreira, P. Bridges, and R. Brightwell, “Characterizing application sensitivity to os interference using kernel-level noise injection,” in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 19.

[22] T. Hoefler, T. Schneider, and A. Lumsdaine, “Characterizing the Influence of System Noise on Large-Scale Applications by Simulation,” in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC’10)*, Nov. 2010.

[23] H. D. Simon, “Barriers to exascale computing,” in *High Performance Computing for Computational Science-VECPAR 2012*. Springer, 2013, pp. 1–3.

[24] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, “A survey of rollback-recovery protocols in message-passing systems,” *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[25] J. S. Plank, K. Li, and M. A. Puening, “Diskless checkpointing.” *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 10, pp. 972–986, October 1998.

[26] J. S. Plank, Y. B. Kim, and J. J. Dongarra, “Algorithm-based diskless checkpointing for fault tolerant matrix operations.” in *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*. Pasadena, CA, USA: Los Alamitos, CA, USA : IEEE Comput. Soc. Press, 1995, June 1995, pp. 351–360.

[27] L. M. Silva and J. G. Silva, “An experimental study about diskless checkpointing.” in *24th EUROMICRO Conference*. Vasteras, Sweden: IEEE Computer Society Press, August 1998, pp. 395 – 402.

[28] S. Monnet, C. Morin, and R. Badrinath, “Hybrid checkpointing for parallel applications in cluster federations,” in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*. IEEE, 2004, pp. 773–782.

[29] L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. De Mel, “An analysis of communication induced checkpointing,” in *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*. IEEE, 1999, pp. 242–249.

[30] R. Gioiosa, J. C. Sancho, S. Jiang, F. Petrini, and K. Davis, “Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers,” in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 9.

[31] P. Widener, K. Ferreira, S. Levy, P. G. Bridges, D. Arnold, and R. Brightwell, “Asking the right questions: benchmarking fault-tolerant extreme-scale systems,” in *Proc. 6th Workshop on Resiliency in High Performance Computing*, Aachen, Germany, August 2013, in conjunction with Euro-Par 2013.

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

[33] T. Hoefler, “LogGOPSim - A LogGOPS (LogP, LogGP, LogGPS) Simulator and Simulation Framework,” http://www.unixer.de/research/LogGOPSim/, Apr. 10 2013.

[34] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, “Logp: towards a realistic model of parallel computation,” *SIGPLAN Not.*, vol. 28, no. 7, pp. 1–12, Jul. 1993. [Online]. Available: http://doi.acm.org/10.1145/173284.155333

[35] T. Hoefler, C. Siebert, and A. Lumsdaine, “Group Operation Assembly Language - A Flexible Way to Express Collective Communication,” in *ICPP-2009 - The 38th International Conference on Parallel Processing*. IEEE, Sep. 2009.

[36] A. Tikotekar, G. Vallée, T. Naughton, S. L. Scott, and C. Leangsuksun, “Evaluation of fault-tolerant policies using simulation,” in *Cluster Computing, 2007 IEEE International Conference on*. IEEE, 2007, pp. 303–311.

[37] S. Bohm and C. Engelmann, “xSim: The extreme-scale simulator,” in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*. IEEE, 2011, pp. 280–286.

[38] A. Boteanu, C. Dobre, F. Pop, and V. Cristea, “Simulator for fault tolerance in large scale distributed systems,” in *Intelligent Computer Communication and Processing (ICCP), 2010 IEEE International Conference on*. IEEE, 2010, pp. 443–450.

[39] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, “A simulator for large-scale parallel computer architectures,” *International Journal of Distributed Systems and Technologies (IJDST)*, vol. 1, no. 2, pp. 57–73, 2010.

[40] “Sst: The structural simulation toolkit,” http://sst.sandia.gov/about_sstmacro.html, 2011.

[41] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra, “An evaluation of user-level failure mitigation support in mpi,” in *Recent Advances in the Message Passing Interface*, ser. Lecture Notes in Computer Science, J. L. Träff, S. Benkner, and J. J. Dongarra, Eds. Springer Berlin Heidelberg, 2012, vol. 7490, pp. 193–203. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33518-1_24