Toward a Performance/Resilience Tool for Hardware/Software Co-Design of High-Performance Computing Systems

Christian Engelmann and Thomas Naughton
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
engelmannc@ornl.gov and naughtont@ornl.gov

***Abstract*—xSim is a simulation-based performance investigation toolkit that permits running high-performance computing (HPC) applications in a controlled environment with millions of concurrent execution threads, while observing application performance in a simulated extreme-scale system for hardware/software co-design. The presented work details newly developed features for xSim that permit the injection of MPI process failures, the propagation/detection/notification of such failures within the simulation, and their handling using application-level checkpoint/restart. These new capabilities enable the observation of application behavior and performance under failure within a simulated future-generation HPC system using the most common fault handling technique.**

***Keywords*-Fault Injection; Message Passing Interface; Parallel Discrete Event Simulation; High-performance Computing;**

I. Introduction

With the recent deployment of the first 10-20 PFlop/s (1 PFlop/s = 10^{15} floating-point operations per second) supercomputers and the exascale computing roadmap targeting 100, 250, and eventually 1,000 PFlop/s (1 EFlop/s) over the next decade, the trend in high-performance computing (HPC) architectures goes clearly in only one direction. Due to the end of processor frequency scaling caused by chip-level power consumption and heat dissipation limitations, HPC systems will dramatically scale up in compute node and processors core counts to continue to increase performance. By 2020, an exascale system may have 100,000-1,000,000 compute nodes with 1,000-10,000 light- and/or heavy-weight processor cores per node. This poses several challenges related to power consumption, performance, resilience, productivity, programmability, data movement, and data management [1], [2], [3], [4], [5].

Within the same time frame, semiconductor process technology is continuing to shrink (to potentially 7 nm operating at near-threshold voltage), introducing unknown aging effects, increased variability, and increased soft error vulnerability. Resilience, i.e., providing efficiency and correctness in the presence of faults [6], is one of the most important exascale computer science challenges as systems are expected to scale up in component count and component reliability is expected to decrease.

For HPC applications, there are different types of resilience requirements: (1) long-running applications span weeks or even months and easily exceed a system's mean-time to interrupt, (2) bit-wise reproducible application runs needed for certifiable results require correct and deterministic execution, and (3) extreme-scale applications require the entire system to work efficiently at scale even in the presence of continuously occurring faults. The challenge of long running, bit-wise reproducible, and extreme-scale HPC application executions is also compounded by the tight coupling (time-sensitive data dependency) of the individual application processes running on the system.

Major challenges for HPC resilience on the road to exascale include the improvement of existing technologies, such as checkpoint/restart, for short-term impact (evolutionary path), the development of alternative approaches, such as programming model support for resilience, algorithm-based fault tolerance, and self-aware runtime systems, for long-term impact (revolutionary path), as well as, fully understanding the HPC resilience problem, such as creating a HPC resilience taxonomy and identifying primary fault classes and their distributions. A number of advanced resilience technologies have been developed and/or are currently in development, including checkpoint/restart-specific file and storage systems, incremental/differential checkpointing, message logging for uncoordinated checkpointing, fault tolerant message passing interface (FT-MPI), containment domains, algorithm-based fault tolerance (ABFT), rejuvenation, reliability-aware scheduling, proactive migration, and redundancy [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19]. However, there are currently no tools, methods, and metrics to compare them fairly, especially at extreme scale, and to identify the cost/benefit trade-off. HPC hardware/software co-design that considers all aspects of resilience, including the performance/power/resilience trade-off, is crucial to enable exascale scale computing.

A central facet of resilience is to understand root causes and propagation, as well as, avoidance, masking, and recovery efficiency to aid in and validate the improvement of the hardware/software environment. The presented work focuses on developing a resilience co-design toolkit with definitions,

metrics, and methods to evaluate the cost/benefit trade-off of resilience solutions, identify hardware/software resilience properties, and coordinate interfaces/responsibilities of individual hardware/software components.

As a first step, this paper presents recent accomplishments and ongoing work in providing support for investigating the most common resilience strategy today, checkpoint/restart, on future-generation HPC architectures using simulation. While previous accomplishments in modeling and simulation of checkpoint/restart primarily focused on the relationship between a generalized overhead of saving and restoring state and the failure frequency of a system at increasing scales, this effort targets a more detailed investigation that includes application properties, such as the amount of application state that needs to be saved/restored at different scales and times, and architectural features, like the capabilities offered by different checkpoint file/storage systems and by the I/O network infrastructure.

To this end, we have extended the Extreme-scale Simulator (xSim) [20], [21], [22], [23] with fault injection, propagation, detection, notification, and handling capabilities to allow for such more detailed investigations at scale. xSim is a simulation-based performance investigation toolkit that permits running native HPC applications in a controlled environment with millions of concurrent execution threads (each as an MPI task), while observing application performance in a simulated extreme-scale system for hardware/software co-design using simulation models. The work presented in this paper details newly developed features for xSim that permit the injection of MPI process failures, the propagation/detection/notification of such failures within the simulation, and their handling using application-level checkpoint/restart. These new capabilities enable the observation of application behavior and performance under failure within a simulated future-generation HPC system using the most common fault handling technique.

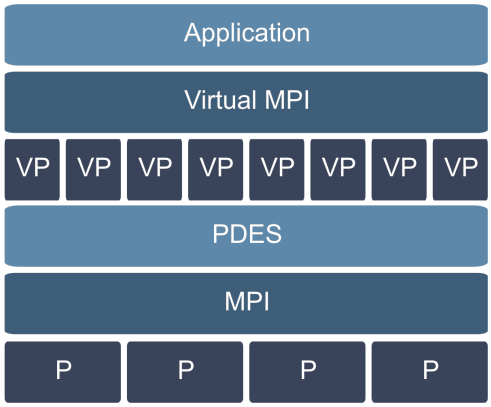
The paper is structured as follows. Section II briefly discusses related work, while Section III describes the taken approach. Section IV details the implementation and Section V shows experimental results. Section VI concludes this paper with a summary and an outlook on future work.

II. Related Work

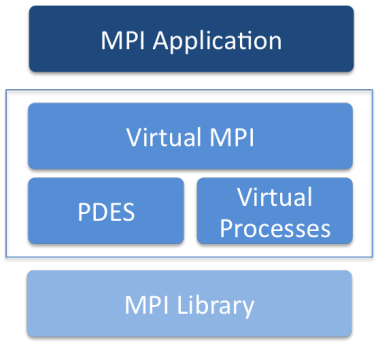
In the following, related work in hardware/software co-design simulation tools, resilience modeling and simulation, and fault injection tools is examined.

A. Hardware/Software Co-Design Simulation Tools

The Extreme-scale Simulator (xSim) [20], [21], [22], [23], developed by our team at Oak Ridge National Laboratory (ORNL), is a performance investigation toolkit (Figure 1) that permits running native HPC applications or proxy/mini applications in a controlled environment with millions of concurrent execution threads (virtual processes (VPs)), while



(a) Architecture (© 2010 IEEE [22])



(b) Design (© 2010 IEEE [22])

Figure 1. xSim's implementation architecture and design.

observing application performance in a simulated extreme-scale system for hardware/software co-design. Using a lightweight parallel discrete event simulation (PDES), xSim executes an application on a much smaller HPC system in a highly oversubscribed fashion with a virtual wall clock time, such that performance data can be extracted based on a processor and a network model with the appropriate simulation scalability/accuracy trade-off. xSim is designed like a traditional performance tool, as an interposition library that sits between the MPI application and the MPI layer, using the MPI performance tool interface. It is able to run up to 134,217,728 (2^{27}) communicating MPI tasks, each with its own process context, using just a 960-core Linux-based cluster. xSim is relatively easy to use and its comparably small code base makes it primary target for adding resilience modeling and simulation features. The scalability/accuracy trade-off offered by xSim provides a unique opportunity for extreme-scale studies. xSim currently does not support threaded execution models, such as OpenMP, and accelerators, such as GPGPUs.

The Structural Simulation Toolkit (SST) [24] (http://www.cs.sandia.gov/sst) offers simulation of novel compute-node architectures, including processor, memory, and network, of future-generation HPC systems. It is a modular PDES

framework atop MPI that scales to a few hundred nodes with different levels of accuracy, utilizing a variety of external modeling and simulation tools. Its value is in the ability to investigate the performance of future node architectures and to generate models/traces for larger-scale system simulations. SST does support running native applications, including proxy/mini applications [25]. SST/macro is a complementary simulation toolkit that processes output from the MPI tracing library DUMPI (http://sst.sandia.gov/about_dumpi.html) for performance evaluation. SST and SST/macro enable the synergy between small-scale cycle-accurate and large-scale communication-accurate simulations by generating DUMPI traces at smaller-scale with SST and extrapolating performance at extreme-scale using SST/macro. While SST is mature, it is quite complex to use due to its interaction with a variety of modeling and simulation tools. SST/macro is currently under development and not generally available.

Other trace-driven PDES solutions exist. DIMEMAS [26] processes MPIDTrace traces and generates output for the performance tools, PARAVER [27] (http://www.bsc.es/computer-sciences/performance-tools/paraver) and Vampir [28] (http://www.vampir.eu). Also, SimGrid [29] and OMNeT++ [30] are multi-purpose simulation toolkits that have been used to investigate the runtime capabilities of future system architectures.

Further performance modeling and simulation tools for HPC co-design include (a) communication-level tracing tools, (b) architecture-level (processor and memory) emulators/simulators, and chip-, node-, and system-level power consumption and heat dissipation models and simulators.

B. Resilience Modeling and Simulation

Modeling and simulating for resilience in HPC has mostly focused on 1) assuring component, sub-system, and/or system reliability and 2) optimizing checkpoint/restart.

HPC vendors, i.e., manufacturers of individual components, such as processors and memory modules, and integrators, internally perform modeling and simulation of component, sub-system, and/or system reliability to assure a certain upper bound on the failure in time (FIT, the number of failures that can be expected in 10^9 hours of operation) rate. This permits vendors to make design decisions about trading off component reliability for deployment/operation costs and for deploying additional mitigation.

As the standard practice for resilience in HPC is checkpoint/restart, modeling and simulating primarily aimed at optimizing this particular mitigation technique, such as by finding the optimal checkpoint interval [31]. Recent work in incremental checkpointing and in redundancy for HPC used modeling and simulation to compare these mitigation techniques with the standard checkpoint/restart to identify their overhead costs and benefits [18], [7].

C. Fault Injection Tools

Finject [32] is a component-based framework, developed by our team at ORNL, for inducing errors in key subsystems of the Linux 2.6 kernel and in applications. Its architecture consists of injectors, detectors, analyzers, and a controller, and is primarily designed for performing robustness testing experiments. The available injector components utilize the capabilities of the `ptrace(2)` system call to inject bit flips in the core image and registers of a victim process, and of the fault injection feature in Linux [33] to inject slab, page allocation, and disk I/O errors. For example, register bit flips were introduced into a user-space application (victim) using `ptrace(2)`. While the detector watches the victim process and reports on its exit, the analyzer counts the injections and detections. In the tests, an arbitrary maximum of 100 injected faults was set, with application failures occurring at varied points (see Table I). In another experiment, the Linux fault injection feature was used for inducing slab-based memory faults [32].

Table I
FAULT (BIT FLIP) INJECTION RESULTS [32]

Field	Value	Description
Victims	100	# of victim application instances
Injections	2197	# of injected failures for all runs
Minimum	1	# of injections to victim failure
Maximum	98	# of injections to victim failure
Mean	21.97	# of injections to victim failure
Median	17	# of injections to victim failure
Mode	4	# of injections to victim failure
Std.Dev.	21.42	# of injections to victim failure

Our recent work in resilience at the MPI layer produced the redMPI prototype, which permits the execution of applications with process-level redundancy. RedMPI [34] is capable of online detection and correction of soft errors (bit flips) without requiring any modifications to the application using double or triple redundancy. It can be also used as a fault injection tool by disabling the online correction and keeping replicas isolated. A failure free execution can be compared to the redundant execution with an injected fault using the online detection mechanism to track propagation. Depending on the application properties, a single bit flip can corrupt all MPI processes of an application within a short period of time, or may be corrected by the application's computational structure, such as by an iterative algorithm.

Another study on soft errors, such as memory bit flips and message corruption, in MPI-based applications using injection campaigns showed noticeable effects on applications and an in incorrect output 28-71 % of the time [35]. While not strictly a fault-injection mechanism, instrumentation tools, like Pin [36], [37], KernInst [38], DynInst [39], and DTrace [40], can be adapted to create injectors and detectors. These tools operate at the binary code level, re-writing sections of the executable either on disk or at runtime in

memory. In the case of dynamic instrumentation, the runtime change may be done in kernel-space [36], [40], [41], [38] or user-space [39], [40], [37].

There has also been work to leverage common hardware components, e.g., performance counters, debug registers, to improve the efficiency of the actual fault-injection mechanism. The Xception [42] tool leverages these capabilities to provide low-overhead mechanisms, which can be invoked when a particular event is triggered. These triggers are tied to available debugging features of the hardware, *e.g.*, breakpoint registers. This tool was also used in an early study of the effects of processor-based faults on parallel systems (Parsytec PowerXplorer [43]).

In addition to the described software implemented fault injection (SWIFI) methods, Sass et al. [44] used field-programmable gate array (FPGA) hardware to create a testbed for fault injection and performance degradation experiments on a 64-node cluster running MPI applications. These hardware-level effects are reproducible, but require additional hardware on each node in the experiment.

III. Approach

xSim's ability to simulate the architectural properties of extreme-scale HPC systems at reasonable accuracy and to run real applications or proxies makes it the prime candidate for a HPC resilience hardware/software co-design toolkit for investigating the performance/resilience/power trade-off of future HPC architecture choices with different resilience properties (vulnerabilities) and strategies (mitigation techniques). The presented work focuses on enhancing xSim with initial resilience simulation features.

A. Overall Approach

To extend the xSim performance investigation tool to a HPC resilience hardware/software co-design toolkit, the overall effort focuses on adding advanced features to (1) permit the injection of different faults, errors, and failures into the simulation, (2) model various propagation, isolation, and detection properties of the simulated system, (3) support a variety of avoidance, masking, and recovery strategies, (4) model the power consumption of the entire simulated system, and (5) study the performance, resilience, and power consumption impact with different parameter sets for (1), (2), (3), and (4) using standardized methods and metrics. This work is novel in several ways.

The improved xSim would be (a) the first fault injection toolkit for extreme-scale systems, (b) the only fault injection toolkit with parameterized resilience properties and strategies, and (c) the first holistic HPC co-design toolkit that considers architectural performance and resilience parameters to optimize parallel application performance within a given power consumption budget.

B. Targeted Approach

The first step toward these outlined goals is to enable the injection of MPI process failures into the running simulation. They are the most common experienced failures by an application and caused either by a fault within the MPI process, for example an illegal memory access, or by an error or failure of another component, such as a file I/O error reported by the parallel file system.

In any case, when such an MPI process failure is detected by another MPI process, the execution of the entire MPI application is aborted according to the current MPI fault model. Hence, the detection of an MPI process failure and the subsequent MPI abort should be part of the simulated HPC system that xSim exposes.

Lastly, the most common fault handling technique in HPC today is application-level checkpoint/restart. Application state is regularly written out to the parallel file system as a checkpoint. In case of a failure, the application is restarted and the last written out checkpoint is read back in to continue the execution with a previously saved state. The progress between the time the last checkpoint was written and the application failed is lost and needs to be recomputed. This fault handling technique needs to be supported by xSim, such that applications can save/restore checkpoints and restart after an abort. While application aborts may interrupt a simulation, an application restart should resume the simulation to account for the time an application spends on checkpointing and on failure/restart cycles.

The following Section outlines the implementation of an MPI process failure injection facility, a corresponding detection/notification capability, an MPI abort feature, and application-level checkpoint/restart capability within xSim.

IV. Implementation

To better understand the implementation of the MPI process failure and MPI abort features in xSim, we first briefly describe the simulator's execution of simulated MPI processes. More details can be found in our earlier publications [20], [21], [22], [23].

A. Simulated MPI Process Execution

xSim employs its own user-space process threading for optimal performance. Each simulated MPI rank has its own full thread context (CPU registers, stack, heap, and global variables). xSim retains full control over the thread schedule. It always executes one simulated MPI process per native MPI process at a time. xSim itself is a parallel MPI application and each native MPI process executes a simulated MPI process until the simulated MPI process yields to xSim by receiving an MPI message or performing any simulator-internal function. Context switches between simulated MPI processes are only performed upon receiving an MPI message, receiving a simulator-internal message, or termination. In the typical oversubscribed execution, where

there are more simulated than native MPI processes, the execution of the simulated MPI processes is sequentialized and interleaved at each native MPI process, using a schedule based on message receive time stamps. For certain simulator-internal functions, such as the processor, network and file system model, the clock maintained by xSim for each simulated MPI process separately needs to be updated/advanced. This occurs every time a timing function is called, like `gettimeofday()`, the file system is accessed, or MPI communication is performed.

B. Simulated MPI Process Failure Injection

Simulated MPI process failures are injected by scheduling them at the targeted simulated MPI process. Each simulated MPI process has its own time of failure that is initialized to 0, i.e., fail never, on startup. A simulator-internal function allows to trigger a process failure at a certain virtual MPI process execution time, including immediately, by setting this time of failure value. Any condition-based injection, such as to simulate a specific failure scenario, needs to be implemented either by a simulator reliability model (planned for the future) or by the application itself, calling this simulator-internal function or returning from `main()` or calling `exit()` without having called `MPI_Finalize()`. xSim additionally offers to pass a simulated MPI process failure schedule in the form of rank/time pairs on the command line or via an environment variable on startup. This is the typical method for injecting failures at this point.

A scheduled simulated MPI process failure is activated when the targeted simulated MPI process is executing, updates its simulated process clock, and the clock reaches or goes beyond the simulated MPI process' time of failure value. As this happens only when the simulator regains control, the simulated process clock may be well beyond the scheduled time of failure. To assure accurate and consistent simulated MPI process failure injection, the simulated process is failed with the simulated process time the simulator regains control when it has reached or passed the time of failure. In other words, the scheduled time is the earliest time of failure, while the actual time of failure depends on when the simulator regains control.

Once a simulated MPI process fails, its execution ends and all messages directed to this simulated MPI process are deleted. An informational message is printed out on the command line to let the user know of the time and location (rank) of the failure. A simulator-internal message is broadcast to notify all simulated MPI processes of the failure and the time of failure. Each simulated MPI process maintains its own list of failed simulated MPI processes and their corresponding time of failure.

C. Simulated MPI Process Failure Detection

As part of the simulated MPI layer, failure detection is simulated as well. As all simulated MPI processes get notified internally about a failed simulated MPI process, the simulated MPI process failure detection relies on the simulator-internal notification message to release (and fail) unmatched message receive requests from the failed simulated MPI process another simulated MPI process is waiting on. The simulated network communication time of the waiting simulated MPI process is adjusted for the time of failure, simulating a configurable network communication timeout according to the network model.

The existing simulator-internal synchronization mechanism [21], which assures a conservative PDES execution with deadlock detection, is used to release (and fail) unmatched `MPI_ANY_SOURCE` receive requests. Any similar receive requests waited on after receiving the simulator-internal notification message fail based on the per-process list of failed simulated MPI processes. Similarly, any message send requests waited on after receiving the simulator-internal notification message fail based on this list. The existing synchronization mechanism is also used to assure that waited-on message send requests can fail at the correct simulated MPI process time by allowing the receiving process that may fail before a send request completes to proceed beyond (or fail at or before) the request completion time.

The currently implemented simulated MPI process failure detection is purely based on simulated network communication timeouts when trying to communicate with a failed simulated MPI process. The simulated network communication timeout is configurable as part of xSim's network model. Each simulated network, such as the on-chip, on-node, and system-wide network, has its own network communication timeout simulated based on assumptions of the architectural features of the simulated HPC system. The implemented feature does permit using different simulated MPI process failure detectors, including a simulated HPC monitoring system that notifies the MPI layer about process failures. This capability is currently under development.

D. Simulated MPI Abort

Once the simulated MPI layer detects a process failure, `MPI_Abort()` is invoked if the error handler of the particular communicator is set to the default value of `MPI_ERRORS_ARE_FATAL`. Note that xSim does support other error handlers, such as `MPI_ERRORS_RETURN` and user-defined error handlers. If a simulated MPI process calls `MPI_Abort()`, xSim prints out an informational message on the command line to let the user know of the time and location (rank) of the abort. A simulator-internal message is broadcast to notify all simulated MPI processes of the abort and the time of the abort. Similar to a failure notification, this message is used to release (and fail) waits on any unmatched receive requests at the time of the abort. The synchronization mechanism is used for waits on unmatched `MPI_ANY_SOURCE` receive requests and for waits on send

requests in the same fashion as for dealing with simulated MPI process failures.

Similar to the simulated MPI process failure activation, a simulated MPI process abort is activated once the simulated MPI process is executing, updates its simulated process clock, and the clock reaches or goes beyond the simulated MPI process' time of abort. The actual simulated MPI process abort time depends on when the simulator regains control. The abort is executed for each simulated MPI process. Simulated MPI process timing statistics (minimum, maximum, and average) are printed out, similar to a non-aborted simulation execution, during the shutdown of the simulator. The simulator terminates after all simulated MPI processes aborted.

E. Application-level Checkpoint/Restart

To support continuous virtual timing after an abort and a following restart, xSim optionally writes out the simulated time of the application exit (maximum simulated MPI process time) to a file. This file can be read in upon restart to initialize the clock of all simulated MPI processes with this time. With this simple addition, xSim fully supports the simulation of application-level checkpoint/restart triggered by injected simulated MPI process failures.

V. Results

A. System Setup

The experiments were performed on a 960-core Linux cluster with 40 compute nodes, two 1.7 GHz AMD Opteron 6164 HE processors per node, 12 cores per processor, 64 GB RAM per node, and a bonded dual non-blocking 1 Gbps Ethernet interconnect (1.7 Gbps measured point-to-point TCP bandwidth). The system is running Ubuntu 12.04 LTS, Open MPI 1.6.4, and GCC 4.6. Despite its small size, in comparison to the simulated systems, this system has been proven to be particularly useful due to its large amount of total RAM (2.5 TB), which can accommodate a large number of simulated MPI processes (over 100 million in a previous experiment).

B. Targeted Application

To demonstrate xSim's new capabilities, we performed experiments using a simple MPI application that iteratively solves the heat equation of a regular 3D grid. It decomposes the 3D problem by splitting it into cubes distributed across the MPI ranks. Each rank performs the same total number of iterations, in which each data point is updated using the values of the surrounding data points. A halo exchange between neighboring cubes is performed at a certain iteration interval. This structures the application into distinct computation and communication phases. A checkpoint is written to disk at a certain iteration interval, containing the application's configuration and the current iteration's data. After writing out a checkpoint, a global barrier synchronizes all processes, such that the previous checkpoint can be deleted safely. In case of a failure, the application can be restarted using the same number of MPI ranks. It automatically loads the last checkpoint and automatically deletes any corrupted checkpoint (checkpoint file that exists, but misses some information). Before such a restart, incomplete checkpoints (missing checkpoint files due to a failure during checkpointing) are deleted using a shell script.

The application parameters are (1) the problem size, (2) the total iteration count, (3) the halo exchange interval, and (4) the checkpoint interval. For simplicity, the problem size and the total iteration count are fixed and not considered in the experiments. For demonstration purposes, the halo exchange interval is set so the checkpoint interval, i.e., a halo exchange takes place right before a checkpoint. To demonstrate the newly implemented capabilities in xSim, the checkpoint interval is varied.

C. Simulated System

The simulated future HPC system is configured with 32,768 (2^{15}) nodes organized in a $32\times32\times32$ 3-D wrapped torus with 1 μs link latency and 32 GB/s link bandwidth. The number of cores/node is not considered as an MPI+X, e.g., MPI+OpenMP, programming model is assumed. Therefore, each simulated MPI rank is placed on one simulated compute node. The simulated eager communication threshold is set to 256 kB, i.e., MPI payloads above 256 kB utilize the simulated rendezvous protocol. MPI collectives utilize linear algorithms. For demonstration purposes, the simulated compute node is operating at a speed 1000x slower than a single 1.7 GHz AMD Opteron 6164 HE core. This lessens the native computational load, as the native 960-core system is oversubscribed by a factor of 35, and permits simulations with more realistic failure frequencies.

The MPI process failure location is chosen randomly, i.e., a random MPI rank within the total number of simulated MPI ranks and a random time within $2 * MTTF_s$ (system mean-time to failure, system MTTF). This evenly distributed simulated system MTTF applies to each application run separately, i.e., from start to finish/failure and from restart to finish/failure. In this worst case scenario, the application MTTF can differ significantly from the system MTTF [45]. Since the individual checkpoint files are extremely small and xSim's file system model is a work in progress, the file system overhead for checkpoint/restart was not considered in the experiments.

The simulated system parameters are (a) the system size (MPI rank count), (b) the system configuration (network and processor model), and (c) the system MTTF for injected process failures. For simplicity, the system size and the system configuration are fixed and not considered in the experiments. To demonstrate the newly implemented capabilities in xSim, the system MTTF is varied.

D. First Impressions

First experiences with executing the application using 32,768 simulated MPI ranks and injecting a process failure resulted in interesting failure cases. According to the implemented fault model, an injected failure is detected when communicating with the failed MPI process. As the application has the cycle of computation, halo exchange, checkpoint, and barrier, the observed application failure modes were quite interesting.

As the computation phase is by orders of magnitudes significantly longer than the communication and checkpoint phases, the probability of failure during the computation phase is correspondingly larger. However, a failure during the computation phase is detected in the halo exchange due to failing communication. Also, a failure during the checkpoint phase is detected in the following barrier. As detected failures lead to an application abort, the application aborted during the halo exchange and/or checkpoint phase, *always* resulting in an incomplete or corrupted checkpoint, or during the barrier phase resulting in only partially deleted old checkpoints. Even this simple experiment demonstrates xSim's capability in aiding the investigation of system and application resilience properties.

E. Performance under Failure

To demonstrate xSim's capability to investigate performance under failure in application-level checkpoint/restart scenarios, the heat application is executed using 32,768 simulated MPI ranks. The total iteration count is fixed to 1,000 and the problem size is $512 * 512 * 512$, divided up among the 32,768 simulated MPI ranks in $32 * 32 * 32$ cubes. The checkpoint (and halo exchange) interval is varied between 125, 250, and 500 iterations, and the simulated system MTTF between 3,000 s and 6,000 s.

Table II illustrates the results. The first row below the table's heading provides the baseline performance with no failures and only one checkpoint that writes out the result after the last iteration. The following rows are grouped by system MTTF ($MTTF_s$), each group using checkpoint intervals (C) at 50%, 25%, and 12.5% of the total iteration count. For each row, the simulated execution time without failures (E_1), the simulated execution time with failures and restarts (E_2), the actual number of randomly injected failures (F), and the experienced application mean-time to failure ($MTTF_a$) is provided.

Each row represents the execution of 1,000 iterations, including any failure/restart cycle, with randomly injected MPI process failures. The experiments are repeatable as the simulator and the application are deterministic. The results demonstrate how the checkpoint interval (C) and the system MTTF ($MTTF_s$) influence the simulated execution time with failures and restarts (E_2). As experienced in the real world, a shorter checkpoint interval does not cost much for this particular application (see E_1 column) and provides the benefit of losing less computational progress with higher failure frequencies (see E_2 column). This experiment further demonstrates xSim's capability in aiding the investigation of application and system resilience properties.

Table II
VARYING THE CHECKPOINT INTERVAL AND SYSTEM MTTF

$MTTF_s$	C	E_1	E_2	F	$MTTF_a$
—	1000	5,248 s	—	0	—
6,000 s	500	5,258 s	7,957 s	1	3,978 s
6,000 s	250	6,377 s	7,074 s	1	3,537 s
6,000 s	125	6,601 s	6,750 s	1	3,375 s
3,000 s	500	5,258 s	10,584 s	2	3,528 s
3,000 s	250	6,377 s	8,618 s	2	2,872 s
3,000 s	125	6,601 s	7,948 s	2	2,649 s

VI. Conclusion

This paper presents our initial accomplishments in developing a HPC hardware/software co-design toolkit for investigating the performance/resilience of future HPC architecture choices with different resilience properties (vulnerabilities) and strategies (mitigation techniques). We extended xSim, a simulation-based performance investigation toolkit, with an MPI process failure injection facility, a corresponding detection/notification capability, an MPI abort feature, and application-level checkpoint/restart capability. Our results demonstrate the newly added capability in aiding the investigation of application and system resilience properties. Specifically, the results show how a system and an application interact during a failure.

The presented work is only a first step toward the ultimate goal of performance/resilience/power HPC system co-design. Our ongoing and future work focuses on, among other things, (1) injecting soft errors, (2) developing component-based system reliability models, (3) simulating MPI user-level failure mitigation (ULFM), (4) creating file system models, and (5) developing power consumption models. For example, we recently added the tracking of dynamic memory allocation of simulated MPI processes, which was the last piece needed to develop a soft error injector. We have also recently added initial ULFM support according to the pending MPI ULFM proposal. ULFM handles process faults at the application through MPI-level error notification, i.e., the `MPI_ERR_PROC_FAILED` error code, and MPI calls for remote process notification, i.e., `MPI_Comm_revoke()`, and communicator reconfiguration, i.e., `MPI_Comm_shrink()`.

Acknowledgements

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. De-AC05-00OR22725. This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725

REFERENCES

[1] F. Cappello, A. Geist, B. Gropp, L. V. Kale, W. Kramer, and M. Snir, "Toward exascale resilience," University of Illinois at Urbana-Champaign (UIUC) - Institut National de Recherche en Informatique et en Automatique (INRIA) Joint Laboratory on PetaScale Computing, Tech. Rep. TR-JLPC-09-01, Jun. 2009. [Online]. Available: http://institutes.lanl.gov/resilience/docs/Toward%20Exascale%20Resilience.pdf

[2] M. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. Plank, P. Ranganathan, and J. Simons, "System resilience at extreme scale," Defense Advanced Research Project Agency (DARPA), Tech. Rep., 2008. [Online]. Available: http://institutes.lanl.gov/resilience/docs/Toward%20Exascale%20Resilience.pdf

[3] A. Geist and R. F. Lucas, "Major computer science challenges at exascale," International Exascale Software Project, Tech. Rep., Feb. 2009, whitepaper. [Online]. Available: http://www.exascale.org/mediawiki/images/8/87/ExascaleSWChallenges-Geist_Lucas.pdf

[4] P. Kogge et al., "ExaScale computing study: Technology challenges in achieving exascale systems," Defense Advanced Research Project Agency (DARPA) Information Processing Techniques Office (IPTO), Tech. Rep., 2008, http://users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf.

[5] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," in *Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference 2007*, vol. 78. Boston, MA, USA: Institute of Physics Publishing, Bristol, UK, Jun. 24-28, 2007, pp. 2022–2032. [Online]. Available: http://www.iop.org/EJ/abstract/1742-6596/78/1/012022

[6] N. DeBardeleben, J. Laros, J. T. Daly, S. L. Scott, C. Engelmann, and B. Harrod, "High-end computing resilience: Analysis of issues facing the HEC community and path-forward for research and development," Whitepaper submitted to the U.S. National Science Foundation's High-end Computing Program, Dec. 2009. [Online]. Available: http://www.christian-engelmann.info/publications/debardeleben09high-end.pdf

[7] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for HPC," in *Proceedings of the* 32^{nd} *International Conference on Distributed Computing Systems (ICDCS) 2012*. Macau, China: IEEE Computer Society, Los Alamitos, CA, USA, Jun. 18-21, 2012. [Online]. Available: http://www.christian-engelmann.info/publications/elliott12combining.pdf

[8] C. Engelmann and S. Böhm, "Redundant execution of HPC applications with MR-MPI," in *Proceedings of the* 10^{th} *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2011*. Innsbruck, Austria: ACTA Press, Calgary, AB, Canada, Feb. 15-17, 2011, pp. 31–38. [Online]. Available: http://www.christian-engelmann.info/publications/engelmann11redundant.pdf

[9] C. Engelmann, G. Vallée, T. Naughton, and S. L. Scott, "Proactive fault tolerance using preemptive migration," in *Proceedings of the* 17^{th} *Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2009*. Weimar, Germany: IEEE Computer Society, Feb. 18-20, 2009, pp. 252–257. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/PDP.2009.31

[10] G. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pjesivac-grbovic, K. London, and J. Dongarra, "Extending the mpi specification for process fault tolerance on high performance computing systems," in *In Proceeding of International Supercomputer Conference (ICS)*, 2003.

[11] N. Gottumukkala, B. Leangsuksun, N. Taerat, R. Nassar, and S. L. Scott, "Reliability-aware resource allocation in hpc systems," in *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, ser. CLUSTER '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 312–321. [Online]. Available: http://dx.doi.org/10.1109/CLUSTR.2007.4629245

[12] P. H. Hargrove and J. C. Duell, "Berkeley Lab Checkpoint/Restart (BLCR) for Linux clusters," in *Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference 2006*, vol. 46. Denver, CO, USA: Institute of Physics Publishing, Bristol, UK, Jun. 25-29, 2006, pp. 494–499. [Online]. Available: http://www.iop.org/EJ/article/1742-6596/46/1/067/jpconf6_46_067.pdf

[13] M. Li, S. Vazhkudai, A. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman, "Functional partitioning to optimize end-to-end performance on many-core architectures," in *Proceedings of the* 23^{rd} *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2010*. New Orleans, LA, USA: ACM Press, New York, NY, USA, Nov. 13-19, 2010, pp. 1–12. [Online]. Available: http://www.christian-engelmann.info/publications/li10functional.pdf

[14] P. Lemarinier, A. Bouteiller, T. Herault, and G. Krawezik, "Improved message logging versus improved coordinated checkpointing for fault tolerant mpi," in *in IEEE International Conference on Cluster Computing (Cluster 2004). IEEE CS*. Press, 2004.

[15] N. Naksinehaboon, N. Taerat, C. Leangsuksun, C. F. Chandler, and S. L. Scott, "Benefits of software rejuvenation on hpc systems," in *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications*, ser. ISPA '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 499–506. [Online]. Available: http://dx.doi.org/10.1109/ISPA.2010.82

[16] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "A job pause service under LAM/MPI+BLCR for transparent fault tolerance," in *Proceedings of the* 21^{st} *IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2007*. Long Beach, CA, USA: ACM Press, New York, NY, USA, Mar. 26-30, 2007. [Online]. Available: http://www.csm.ornl.gov/~engelman/publications/wang07job.pdf

[17] ——, "Proactive process-level live migration in HPC environments," in *Proceedings of the IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2008*. Austin, TX, USA: ACM Press, New York, NY, USA, Nov. 15-21, 2008. [Online]. Available: http://www.csm.ornl.gov/~engelman/publications/wang08proactive.pdf

[18] ——, "Hybrid checkpointing for MPI jobs in HPC environments," in *Proceedings of the* 16^{th} *IEEE International Conference on Parallel and Distributed Systems (ICPADS) 2010*. Shanghai, China: IEEE Computer Society, Los Alamitos, CA, USA, Dec. 8-10, 2010, pp. 524–533. [Online]. Available: http://www.christian-engelmann.info/publications/wang10hybrid2.pdf

[19] ——, "Proactive process-level live migration and back migration in HPC environments," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 72, no. 2, pp. 254–267, Feb. 2012. [Online]. Available: http://www.christian-engelmann.info/publications/wang12proactive.pdf

[20] C. Engelmann, "Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale," *Future Generation Computer Systems (FGCS)*, 2013, to appear.

[21] S. Böhm and C. Engelmann, "xSim: The extreme-scale simulator," in *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS) 2011*. Istanbul, Turkey: IEEE Computer Society, Los Alamitos, CA, USA, Jul. 4-8, 2011, pp. 280–286. [Online]. Available: http://www.christian-engelmann.info/publications/boehm11xsim.pdf

[22] C. Engelmann and F. Lauer, ""Facilitating co-design for extreme-scale systems through lightweight simulation"," in *Proceedings of the* 12^{th} *IEEE International Conference on Cluster Computing (Cluster) 2010:* 1^{st} *Workshop on Application/Architecture Co-design for Extreme-scale Computing (AACEC)*. Hersonissos, Crete, Greece: IEEE Computer Society, Sep. 20-24, 2010, pp. 1–8. [Online]. Available: http://www.csm.ornl.gov/~engelman/publications/engelmann10facilitating.pdf

[23] I. S. Jones and C. Engelmann, "Simulation of large-scale HPC architectures," in *Proceedings of the* 40^{th} *International Conference on Parallel Processing (ICPP) 2011:* 2^{nd} *International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*. Taipei, Taiwan: IEEE Computer Society, Los Alamitos, CA, USA, Sep. 13-19, 2011, pp. 447–456. [Online]. Available: http://www.christian-engelmann.info/publications/jones11simulation.pdf

[24] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1964218.1964225

[25] Sandia National Laboratories, Albuquerque, MN, USA, ""Mantevo Project"," 2012, https://software.sandia.gov/mantevo/.

[26] S. Girona, J. Labarta, and R. M. Badia, ""Validation of dimemas communication model for MPI collective operations"," in *Lecture Notes in Computer Science: Proceedings of the* 7^{th} *European PVM/MPI Users' Group Meeting (EuroPVM/MPI) 2000*, vol. 1908. Balatonfüred, Hungary: Springer Verlag, Berlin, Germany, Sep. 10-13 2000, pp. 39–46. [Online]. Available: http://dx.doi.org/10.1007/3-540-45255-9_9

[27] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "PARAVER: A Tool to Visualize and Analyze Parallel Code," in *Proceedings of WoTUG-18: Transputer and occam Developments*, mar 1995, pp. 17–31.

[28] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The vampir performance analysis tool-set," in *Tools for High Performance Computing*, M. Resch, R. Keller, V. Himmler, B. Krammer, and A. Schulz, Eds. Springer Berlin Heidelberg, 2008, pp. 139–155.

[29] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, "Single Node On-Line Simulation of MPI Applications with SMPI," in *International Parallel & Distributed Processing Symposium*. Anchorange (AK), États-Unis: IEEE, May 2011, rR-7426 RR-7426. [Online]. Available: http://hal.inria.fr/inria-00527150

[30] C. Minkenberg and G. R. Herrera, "Trace-driven co-simulation of high-performance computing systems using omnet++," in *OMNeT++ 2009: Proceedings of the 2nd International Workshop on OMNeT++ (hosted by SIMUTools 2009)*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[31] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computing Systems (FGCS)*, vol. 22, no. 3, pp. 303–312, 2006. [Online]. Available: http://dx.doi.org/10.1016/j.future.2004.11.016

[32] T. Naughton, W. Bland, G. Vallée, C. Engelmann, and S. L. Scott, "Fault injection framework for system resilience evaluation – Fake faults for finding future failures," in *Proceedings of the* 18^{th} *International Symposium on High Performance Distributed Computing (HPDC) 2009:* 2^{nd} *Workshop on Resiliency in High Performance Computing (Resilience) 2009*. Munich, Germany: ACM Press, New York, NY, USA, Jun. 9, 2009, pp. 23–28. [Online]. Available: http://doi.acm.org/10.1145/1552526.1552530

[33] "Linux fault injection capabilities infrastructure," documentation available at: http://lxr.linux.no/linux/Documentation/fault-injection/. [Online]. Available: http://lxr.linux.no/linux/Documentation/fault-injection/

[34] D. Fiala, F. Mueller, C. Engelmann, K. Ferreira, R. Brightwell, and R. Riesen, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the* 25^{th} *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2012*. Salt Lake City, UT, USA: ACM Press, New York, NY, USA, Nov. 10-16, 2012, pp. 78:1–78:12. [Online]. Available: http://www.christian-engelmann.info/publications/fiala12detection2.pdf

[35] C. da Lu and D. A. Reed, "Assessing fault sensitivity in mpi applications," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 37.

[36] P. P. Bungale and C.-K. Luk, "PinOS: A programmable framework for whole-system dynamic instrumentation," in *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07)*. New York, NY, USA: ACM, 2007, pp. 137–147.

[37] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*. New York, NY, USA: ACM, 2005, pp. 190–200.

[38] A. Tamches and B. P. Miller, "Fine-Grained Dynamic Instrumentation of Commodity Operating System Kernels," in *Proceedings of 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, Feb. 1999.

[39] B. Buck and J. K. Hollingsworth, "An api for runtime code patching," *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 4, pp. 317–329, 2000.

[40] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, "Dynamic instrumentation of production systems," in *Proceedings of the USENIX Annual Technical Conference (USENIX'04)*. USENIX, June 27 – July 2, 2004, pp. 15–28. [Online]. Available: http://www.usenix.org/events/usenix04/tech/general/full_papers/cantrill/cantrill.pdf

[41] R. G. Minnich, "A Dynamic Kernel Modifier for Linux," in *Proceedings of the 2002 LACSI Symposium (LACSI'02)*, Oct. 2002.

[42] João Carreira and Henrique Madeira and João Gabriel Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, Feb. 1998. [Online]. Available: http://www.xception.org/files/IEEETSE98.pdf

[43] J. G. Silva, J. Carreira, H. Madeira, D. Costa, and F. Moreira, "Experimental assessment of parallel systems," in *Proceedings of the 26th Annual International Symposium on Fault-Tolerant Computing (FTCS'96)*, Jun. 25-27, 1996, pp. 415–424.

[44] R. Sass, R. R. Sharma, and N. DeBardeleben, "Towards a hardware fault-injection testbed to support reproducible resiliency experiments," in *Proceedings of the* 18^{th} *International Symposium on High Performance Distributed Computing (HPDC) 2009:* 2^{nd} *Workshop on Resiliency in High Performance Computing (Resilience) 2009*. New York, NY, USA: ACM, 2009, pp. 15–22.

[45] J. T. Daly, L. A. Pritchett-Sheats, and S. E. Michalak, "Application MTTFE vs. platform MTTF: A fresh perspective on system reliability and application throughput for computations at scale," in *Proceedings of the* 8^{th} *IEEE International Symposium on Cluster Computing and the Grid (CCGrid) 2008: Workshop on Resiliency in High Performance Computing (Resilience) 2008*. Lyon, France: IEEE Computer Society, May 19-22, 2008. [Online]. Available: http://xcr.cenit.latech.edu/resilience2008/program/resilience08-10.pdf