Improving the Performance of the Extreme-scale Simulator

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*—Investigating the performance of parallel applications at scale on future high-performance computing (HPC) architectures and the performance impact of different architecture choices is an important component of HPC hardware/software co-design. The Extreme-scale Simulator (xSim) is a simulation-based toolkit for investigating the performance of parallel applications at scale. xSim scales to millions of simulated Message Passing Interface (MPI) processes. The overhead introduced by a simulation tool is an important performance and productivity aspect. This paper documents two improvements to xSim: (1) a new deadlock resolution protocol to reduce the parallel discrete event simulation management overhead and (2) a new simulated MPI message matching algorithm to reduce the oversubscription management overhead. The results clearly show a significant performance improvement, such as by reducing the simulation overhead for running the NAS Parallel Benchmark suite inside the simulator from 1,020% to 238% for the conjugate gradient (CG) benchmark and from 102% to 0% for the embarrassingly parallel (EP) and benchmark, as well as, from 37,511% to 13,808% for CG and from 3,332% to 204% for EP with accurate process failure simulation.**

***Index Terms*—Performance Prediction; Message Passing Interface; Parallel Discrete Event Simulation; High-performance Computing;**

I. Introduction

The path to exascale high-performance computing (HPC) poses several challenges related to power, performance, resilience, productivity, programmability, data movement, and data management. Investigating the performance of parallel applications at scale on future HPC architectures and the performance impact of different architecture choices is an important component of HPC hardware/software co-design. Without having access to future architectures at scale, simulation tools provide an alternative for estimating parallel application performance on potential HPC architecture choices. As highly accurate simulations are extremely slow and less scalable, different solution paths exist to trade-off simulation accuracy in order to gain simulation performance and scalability.

The Extreme-scale Simulator (xSim) [1], [2], [3], [4], [5], [6], [7] is such a simulation-based toolkit for investigating the performance of parallel applications at scale. The overhead introduced by a simulation tool is an important performance and productivity aspect. xSim scales to millions of simulated Message Passing Interface (MPI) processes. As xSim employs a lightweight parallel discrete event simulation (PDES) mechanism, it has to deal with inherent issues, such as the deadlocks of a conservative PDES. xSim also supports highly-oversubscribed operation, i.e., running millions of simulated MPI processes on just a thousand physical MPI processes. The overhead for dealing with simulated MPI process management and context switches can be significant.

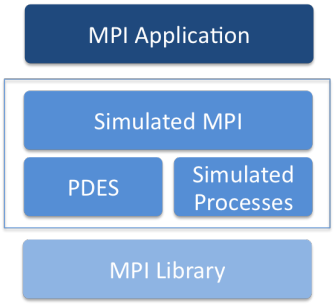
This paper documents two improvements to xSim: (1) a new deadlock resolution protocol to reduce the PDES overhead and (2) a new simulated MPI message matching algorithm to reduce the oversubscription overhead. This paper is structured as follows. Section II describes the existing work in xSim, Section III briefly illustrates related work, Section IV documents the improvements in the deadlock resolution protocol, Section V documents the improvements in the simulated MPI message matching algorithm, Section VI presents experimental results, and Section VII concludes this paper.

II. Background

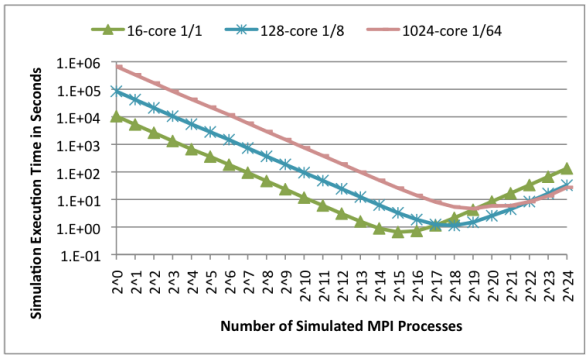
The Extreme-scale Simulator (xSim) [1], [2], [3], [4], [5], [6], [7] is a performance investigation toolkit that permits running a high-performance computing (HPC) application in a controlled environment with millions of concurrent execution threads, while observing its performance in a simulated extreme-scale system. Using a lightweight PDES, xSim executes a MPI application on a much smaller system in a highly oversubscribed fashion (Fig. 1(a)) with a virtual wall clock time, such that performance data can be extracted based on a processor and a network model. xSim is designed like a traditional performance tool, as an interposition library that sits between the MPI application and the MPI library (Fig. 1(b)). It currently does not support threaded execution models, such as OpenMP [8], task-based execution models, such as High Performance ParalleX (HPX) [9], [10], and accelerators, such as general-purpose graphics processing unit (GPGPUs).

The capabilities and usefulness of xSim have been recently demonstrated [3] by (a) scaling it to 134,217,728 (2^{27}) simulated MPI ranks (each with its own process context) on a 960-core Linux cluster (a world record in extreme-scale simulation), (b) evaluating different MPI collective communication algorithms on a simulated future-generation system with 2,097,152 (2^{21}) MPI ranks using the same cluster, and (c) investigating a Monte Carlo solver using different architectural

(a) Architecture (© 2010 IEEE [7])



(b) Design (© 2010 IEEE [7])



(c) Scaling a Monte Carlo application (© 2014 Elsevier [3])

Fig. 1. xSim: The Extreme-scale Simulator

parameters (Fig. 1(c)) with 16,777,216 (2^{24}) simulated MPI ranks using the same cluster.

xSim has a fault injection feature [2] that allows to simulate MPI process failures, which can be scheduled at a specific point in time or triggered on conditions. Fault notifications are propagated (according to the simulated architecture) to each simulated MPI process, which in turn reacts to the fault. xSim also offers full support for error handling within the simulated MPI, including default MPI error handlers, user-defined MPI error handlers, and `MPI_Abort()`. According to the MPI standard, MPI process failures trigger a simulated abort if the MPI error handler is set to `MPI_ERRORS_ARE_FATAL`. Any simulated abort results in a termination of the simulation with performance results and information about the source and time of the abort. xSim offers continuous virtual timing after such an abort and a following restart to fully support the simulation of application-level checkpoint/restart triggered by injected simulated MPI process failures.

xSim further implements the fault-tolerant MPI extensions [1], as proposed by the MPI Fault Tolerance Working Group. They offer user-level failure mitigation (ULFM) [11] capability at the simulated MPI layer to support algorithm-based fault tolerance (ABFT). MPI process failures can be handled by the application using the ULFM extensions if the MPI error handler is set to `MPI_ERRORS_RETURN` or is user-defined. The solution permits investigating performance under failure and failure handling of ABFT solutions. xSim is the very first performance tool that supports ULFM and ABFT.

xSim also supports simulations using MPI application communication traces. Instead of replaying a trace within the simulation, the recently developed framework generates a benchmark from it and runs this benchmark within the simulation. This provides a number of benefits, such as eliminating the data intensive trace replay and enabling simulations at different scales. The framework uses ScalaTrace II [12], [13] to generate trace files, ScalaBenchGen II [14] to generate the benchmark, and xSim to run the benchmark within a simulation. Experimental results showed that the benchmarks generated from the NAS Parallel Benchmark suite [15] perform with a mean error of 5.5% and a maximum of 13.3%.

III. Related Work

The following projects have already been discussed in earlier xSim publications [1], [2], [3], [4], [5], [6], [7].

The Java Cellular Architecture Simulator (JCAS) [16] is xSim's predecessor and was developed in 2001 to investigate scalable and fault-tolerant algorithms for large-scale systems with about 100,000 processor cores. The prototype was able to run up to 500,000 simulated processes on a Linux cluster computer with 5 processor cores (using 1 exclusively for visualization) solving basic mathematical problems. While it was able to run algorithms at scale, it lacked important features, such as time-accurate simulation, high performance, support for running the simulator atop MPI, and a fully functional simulated MPI.

The BigSim [17] project studied programming issues in large-scale HPC systems. The BigSim Emulator was developed for application testing and debugging at scale and ran atop Charm++/AMPI [18]. It supports up to 100,000 simulated MPI processes distributed over 2,000 processor cores. Similar to JCAS, it does not offer time-accurate simulation. It offers more functionality, but scales less. The BigSim Simulator was developed for identification of performance bottlenecks and uses a trace-driven PDES that models architectural parameters of a HPC system. For time-accurate simulation, it supports a variable-resolution processor model and a detailed network model. While it uses a PDES to maintain accuracy, it only supports trace replay and does not run applications.

$\mu\pi$ [19] is a PDES-based system that was under development for predicting the performance of parallel programs.

While it was intended to be quite similar to xSim, it targeted different grafting methods for interfacing applications with the simulation, such as at the source code, library (actually implemented), and virtual machine level. It supported conservative and optimistic execution based on the μsik PDES engine. A prototype was tested on 216,000 cores of the Jaguar Cray XT5 supercomputer, providing over 27 million simulated MPI ranks, each with its own thread context, and all ranks synchronized by simulated time. xSim is far more scalable as $\mu\pi$ requires an extreme-scale system to simulate an extreme-scale system. $\mu\pi$'s PDES engine μsik is, however, superior.

The Structural Simulation Toolkit (SST) [20] offers simulation of novel architectures, including processor, memory, and network. It is a modular PDES framework atop MPI that scales to a few hundred simulated multi-core nodes. Its value is in the ability to investigate the performance of future node architectures and to generate models for larger-scale simulations. SST/macro is a complementary simulation toolkit that processes output from the MPI tracing library DUMPI for performance evaluation. Similar to the BigSim Emulator/Simulator combination, SST and SST/macro enable the synergy between small-scale cycle-accurate and large-scale communication-accurate simulations. While SST is mature, it is quite complex to use. SST/macro is still under development.

IV. Deadlock Resolution

The first improvement to xSim that is discussed in this paper concerns the deadlock resolution protocol that is at the core of the PDES. In xSim, each simulated MPI process maintains its own simulated process clock. Computation performed by a simulated MPI process advances the simulated process clock based on a processor model. The scaling processor model simply measures the time the computation took on the physical processor core and scales it to the estimated performance of the simulated processor core. xSim executes simulated MPI processes piece-wise, i.e., the simulated process clock starts with entering the application's `main()` function, stops upon entering a simulated MPI call, resumes upon exiting a simulated MPI call, and halts with exiting `main()`. Context switches between simulated MPI processes that reside on the same physical MPI process may occur due to oversubscription while the simulated process clock is stopped or halted. As the simulation implements cooperative user-space multi-threading, context switches are performed as needed to startup and run multiple simulated MPI processes and to receive simulated MPI messages in their corresponding context.

Communication performed by a simulated MPI process advances the simulated process clock based on a network model. The network model calculates the latency and bandwidth between the source and destination and enforces the simulated time for waiting on the completion of a communication. This assures proper accounting for the wait and busy times for sending and receiving MPI messages. The simulated process clock is advanced by the wait and busy time amount. As long as communication operations are deterministic, the network model maintains causality in simulated time. For example, the simulated wait time for receiving messages ensures that a message can not be received at a point in simulated time before it has been transmitted at that point in simulated time.

However, the MPI standard does offer a number of operations that can cause causality errors in the simulation, such as when the physical (actual) message ordering differs from the simulated MPI message order. This is due to the fact that the simulation is in simulated time and may not match the actual wall-clock time. It is also due to the asynchronous operation of the simulator, where each simulated MPI process is executed piece-wise, in parallel, and potentially in an oversubscribed fashion. While MPI messages between two simulated MPI processes will be in the same physical receive order with every execution of the simulation, the physical receive order of MPI messages from different simulated MPI processes that are addressed to the same simulated MPI process may differ.

For example, `MPI_ANY_SOURCE` receives generally need to match the MPI message that is received first in time. To achieve this in the simulation, the simulator needs to evaluate all potential MPI messages that could match this condition. This assures correct matching of the MPI message that is received first in simulated time, independent of their physical receive order. To maintain causality and reproducibility for `MPI_ANY_SOURCE` receives, the simulator needs to wait until all potential MPI messages have been received to make this determination. Therefore, xSim has to wait until either: (a) all other simulated MPI processes have passed the simulated timeline for sending an MPI message to the simulated MPI process that is waiting on a `MPI_ANY_SOURCE` receive, or (b) all other simulated MPI processes that have not passed the simulated timeline are deadlocked on receiving a simulated MPI message themselves. In most cases, this results in a deadlock of the simulator. Other simulated MPI calls that have this behavior are: `MPI_Iprobe()`, `MPI_Testall()`, `MPI_Testany()`, `MPI_Testsome()`, `MPI_Test()`, `MPI_Waitany()`, and `MPI_Waitsome()`.

In addition to these simulated MPI calls, the fault-tolerant MPI extensions implemented in xSim also may cause causality errors. The impact of a simulated MPI process failure may not be accurately simulated due to the asynchronous operation of the simulator. For accurate simulation, a sending simulated MPI process needs to know that the receiving simulated MPI process has not failed yet at the simulated time of the completion of the send operation. Similar to `MPI_ANY_SOURCE` receives, the sending simulated MPI process needs to wait until the receiving simulated MPI process has passed this simulated time line or is deadlocked on receiving a simulated MPI message. Similar to `MPI_ANY_SOURCE` receives, this results in a deadlock of the simulator in most cases. As this feature is only needed when performing simulated MPI process fault injection experiments, it is optional and can be controlled via a command line argument.

In all these cases, the simulator enters a deadlock by design, due to the conservative PDES implementation, in order to maintain causality and reproducibility. The deadlock

resolution protocol in xSim is a mechanism to detect these deadlocks and to release them while maintaining causality and reproducibility.

A. Previous Protocol

xSim features a deadlock resolution protocol that is derived from other PDES implementations in the literature. The basic concept relies on, so called, local and global virtual time management and on walking the clock of a deadlocked simulated MPI process. Since xSim supports oversubscription, the local virtual time is, at any point during the simulation, the lowest simulated MPI process clock of those simulated MPI processes located at the same physical MPI process. The local virtual time is always up to date as it is managed locally at each physical MPI process. In contrast, the global virtual time is, at any point during the simulation, the lowest simulated MPI process clock of all simulated MPI processes. The global virtual time typically lags behind as it is managed in a distributed fashion, asynchronously at each physical MPI process using local virtual time update messages. The previous implementation of the deadlock resolution protocol performs the following actions to maintain local and global virtual time:

- When a simulated MPI process clock is advanced, the local virtual time is set to the lowest simulated MPI process clock on the physical MPI process.
- When the local virtual time is advanced, it is (linearly) broadcast to all other physical MPI processes using an explicit "local virtual time update" message.
- Simulated MPI messages transmitted between physical MPI processes piggyback the local virtual time.
- A "local virtual time update" message is only transmitted if it was not already transmitted before.
- When a "local virtual time update" message is received, (i) its containing local virtual time is stored with the associated rank of the sending physical MPI process, and (ii) the global virtual time is set to the lowest stored local virtual time.

In order to detect and resolve deadlocks, the previous implementation of the deadlock resolution protocol performs the following actions:

- When the incoming simulated MPI message queue is empty, or contains messages that can not be matched, the physical MPI process is deadlocked, i.e., waiting on other physical MPI processes.
- When a physical MPI process is deadlocked: (i) if it is the lowest rank, and (ii) its value for local virtual time and global virtual time are equal, then the rank is responsible for resolving the deadlock.
- To resolve a deadlock, the physical MPI process advances its local virtual time to the next simulated MPI process clock or local virtual time, whichever is lower.
- As the local virtual time is advanced, update messages are transmitted to the other physical MPI processes and the global virtual time is advanced.
- As the global virtual time identifies the simulated timeline all simulated MPI processes have passed, the deadlock is resolved by advancing the global virtual time.

This protocol essentially follows the concept of letting the simulation run in a conservative fashion. The timeline used to maintain causality and repeatability is established via the global virtual time. If the simulation is in a deadlock, the global virtual time is safely advanced from the lowest simulated MPI process clock to the next. The weaknesses of this protocol are as follows:

- "Local virtual time update" messages are communicated after most (if not all) simulated MPI process clock updates as the simulator's fair scheduling makes sure that the simulated MPI process with the lowest simulated MPI process clock is next in line for receiving its simulated MPI message and thus advancing its clock.
- The piggyback mechanism is circumvented by the simulated MPI process clock update and any resulting "local virtual time update" messages that occur upon entering a simulated MPI call.
- The walking of the global virtual time by the lowest physical MPI process often results in another physical MPI process becoming responsible for resolving the deadlock, resulting in subsequent global virtual time walks and message storms during deadlock resolution. This is especially the case when the simulated MPI process with the highest simulated MPI process clock is deadlocking all the other simulated MPI processes.

These weaknesses are all performance related and have been identified while running experiments. The main motivation for changing to a different (better performing) protocol was the high performance impact observed when accurately simulating MPI process failures. In this case, each simulated send operation becomes a potential global synchronization point. The results in Section VI demonstrate this issue. For example, the simulation overhead increased from 1,020% (cg.C.128 in Fig. 4(b)) to 37,511% (cg.C.128 in Fig. 5(b)) by just switching on the capability for accurate MPI process failure simulation on the command line.

B. New Protocol

To eliminate the previously stated weaknesses, the deadlock resolution protocol has been improved to reduce the amount of communication introduced from "local virtual time update" messages and to speed up the deadlock resolution. The new protocol still follows the concept of letting the simulation run in a conservative fashion and using the global virtual time to maintain causality and repeatability. However, local virtual time updates are performed less frequently and global virtual time walks do a better job of considering other deadlocked physical MPI processes. The new implementation of the deadlock resolution protocol performs the following actions to maintain local and global virtual time:

- When a simulated MPI process clock is advanced, the local virtual time is set to the lowest simulated MPI process clock on the physical MPI process.
- When the incoming simulated MPI message queue is empty, or contains messages that can not be matched and there is a potential for a deadlock, the local virtual time is (linearly) broadcast to all other physical MPI processes using an update message.
- Simulated MPI messages transmitted between physical MPI processes piggyback the local virtual time.
- A "local virtual time update" message is only transmitted if it was not already transmitted.
- Also, an update message is only transmitted if the local virtual time is equal to, or exceeds, the known local virtual time of the destination physical MPI process, essentially implementing a vector clock.
- When a "local virtual time update" message is received: (i) its containing local virtual time value is stored with the associated rank of the sending physical MPI process, and (ii) the global virtual time is set to the lowest stored local virtual time of the non-deadlocked physical MPI processes or, if none exist, to the *highest* stored local virtual time of the deadlocked physical MPI processes.

The three main improvements are that (1) "local virtual time update" messages are only communicated in the case of a potential deadlock, (2) "local virtual time update" messages are only communicated when they change the vector of the clock, i.e., have the potential to change the lowest physical MPI process with the global virtual time, and (3) the global virtual time is entirely based on the local virtual time of the lowest non-deadlocked, or highest deadlocked, physical MPI process. In order to resolve deadlocks, the new implementation of the protocol performs the following actions:

- Each physical MPI process maintains a local counter that indicates the deadlock potential. It is increased when a locally residing simulated MPI process is performing an action that may lead to a deadlock, such as when posting an `MPI_ANY_SOURCE` receive. It is decreased when it has completed such an action.
- As already mentioned in the global/local virtual time management, when the incoming simulated MPI message queue is empty, or contains messages that can not be matched and there is a potential for a deadlock, the local virtual time is (linearly) broadcast to all other physical MPI processes using an update message.
- When a physical MPI process is deadlocked: (i) if it is the lowest rank, and (ii) its value for local virtual time and global virtual time are equal, then the rank is responsible for resolving the deadlock.
- To resolve a deadlock, the physical MPI process advances its local virtual time to the next simulated MPI process clock, or local virtual time of non-deadlocked physical MPI processes, whichever is lower.
- As the local virtual time is advanced, update messages are not immediately transmitted to the other physical MPI processes, instead, either a locally residing simulated MPI process can proceed (resolving the deadlock) or another deadlock situation is encountered.
- A "local virtual time update" message is eventually transmitted, advancing the global virtual time and resolving the deadlock.

The three main improvements are that: (1) "local virtual time update messages" are only communicated in case of a potential deadlock, (2) the local virtual time is advanced to a non-deadlocked simulated timeline, and (3) the resolution of a deadlock does not immediately cause "local virtual time update" messages to be communicated. The overall design of the new deadlock resolution protocol is significantly more conservative, as it does not constantly update the global virtual time and relies on an iterative approach to resolve deadlocks. It is also more efficient in resolving deadlocks as it skips already known deadlocks.

Another new feature was added to permit changing the deadlock resolution protocol from the described dynamic (on-demand) to a static (always on) behavior. In the described dynamic behavior, "local virtual time update messages" are only communicated when there is an actual deadlock potential. In contrast, "local virtual time update messages" are communicated independent from the deadlock potential in the static behavior, i.e., whenever the simulated MPI message queue is empty or contains messages that can not be matched. This permits a faster resolution of deadlocks as local virtual time information is communicated more frequently and is more up-to-date. However, it also incurs an overhead for sending additional "local virtual time update messages".

V. Message Matching

The second improvement to xSim discussed in this paper concerns the matching of simulated MPI messages. As mentioned before, xSim simulates MPI processes piece-wise, in parallel, and potentially in an oversubscribed fashion. Before entering the application's `main()` function, the entire process context (stack, heap, and meta data) is replicated, when running with oversubscrition, to create each simulated MPI process context. xSim runs a simulation `pthread` thread on each physical MPI process with a user-space stack, which is split between multiple simulated MPI processes using a custom user-space `fork()` call. A context switch changes the stack frame for this simulation thread within this bigger user-space stack and copies out/in the static heap (`.text`, `.bss`, `.data`, and similar segments). This capability is supported under Linux and OS X.

Context switches may occur whenever a simulated MPI process relinquishes control back to xSim, i.e, before entering the application's `main()` function, during simulated MPI calls, and after exiting `main()`. Similar to other PDES solutions, xSim maintains an incoming message queue that contains simulated MPI messages. It is filled by a separate communication `pthread` thread that is responsible for receiving MPI messages from other physical MPI processes.

The simulation thread matches a simulated MPI message in the incoming message queue with the corresponding receiving simulated MPI process. It returns the message to the simulated MPI process in the context of the receiving call. The matching itself involves a context switch to the receiving simulated MPI process to identify the matching parameters, such as the MPI message source and tag of a posted simulated MPI receive. As context switches can be expensive, mainly due to copying out and in of the static heap and due to cache pollution, efficient matching that limits or eliminates unnecessary context switches reduces simulation overhead.

A. Previous Algorithm

The original matching algorithm is straight forward and was derived from other PDES implementations in the literature. Fig. 2 shows the simple linear search of the incoming message list, where each message is checked if it matches and every check potentially involves a prior context switch. While it is simple, it has the major disadvantage that the message list is searched linearly, likely incurring context switches to the same simulated MPI processes at different times during the search. It also incurs unnecessary context switches for messages that can not be matched. The advantage, however, is that messages are matched in message queue order, which is based on the simulated timestamps of messages. As the message with the lowest simulated timestamp is matched, fair scheduling between simulated MPI processes is guaranteed.

```
1  /* Search for matching message. */
2  for (index = 0; index < queue.count; index++) {
3    message = queue.array[index];
4    switch_context(message.dest);
5    /* Check matching criteria. */
6    if match(message) {
7      return message;
8    }
9  }
10 /* No message match, or queue is empty. */
11 deadlock_resolution();
```

Fig. 2. Previous message matching algorithm

B. New Algorithm

The new algorithm, shown in Fig. 3, reduces the amount of context switches to a minimum by batching up the message matching for each destination (simulated MPI process). It also performs the context switch only after a message has been matched to completely eliminate unnecessary context switches. While the number of context switches is reduced to a minimum, messages are matched by chronological order for each destination separately. Fair sharing between simulated MPI processes is not guaranteed as newer messages to one destination may be matched before older messages to another. This could potentially result in imbalances, where one part of the simulation is going further ahead of another. However, as MPI programs are generally communication synchronized, this should not be a significant problem.

VI. Results

Both improvements have been implemented and tested. The following describes the experiments and the results.

```
1  /* Search for matching message (no context switch). */
2  for (index = 0; index < queue.count; index++) {
3    message = queue.array[index];
4    /* If current process is destination for message. */
5    if (current_process == message.dest) {
6      /* Check matching criteria. */
7      if match(message) {
8        return message;
9      } else {
10       message.checked = 1;
11     }
12   } else {
13     message.checked = 0;
14   }
15 }
16 /* Search for matching message (with context switch). */
17 for (index = 0; index < queue.count; index++) {
18   message = queue.array[index];
19   /* Skip previously checked messages. */
20   if (message.checked == 1) {
21     continue;
22   }
23   for (index2 = index      ;
24        index2 < queue.count;
25        index2++           ) {
26     message = queue.array[index2];
27     /* If current process is destination for message. */
28     if (current_process == message.dest) {
29       /* Check matching criteria. */
30       if match(message) {
31         switch_context(message.dest);
32         return message;
33       } else {
34         message.checked = 1;
35       }
36     }
37   }
38 }
39 /* No message match, or queue is empty. */
40 deadlock_resolution();
```

Fig. 3. New message matching algorithm

A. System Setup

The experiments were performed on a 128-core Linux cluster computer with 16 compute nodes, two 2.4 GHz AMD Opteron 2378 processors per node, 4 cores per processor, 8 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. This system was chosen specifically to perform two different types of experiments.

The first set of experiments investigate the simulation overhead for executing the NAS Parallel Benchmark (NPB) suite [15]. They demonstrate the improved performance of xSim when running real HPC applications that perform computation and bursty communication. The second set of experiments take a look at the simulation overhead for executing a benchmark using oversubscription that was generated from the Sweep3D benchmark using ScalaTrace II [12], [13] and ScalaBenchGen II [14]. They show the improved performance of xSim when running trace replays that perform bursty communication only.

B. NAS Parallel Benchmark Suite

The NPB suite [15] is a set of programs developed by the NASA Advanced Supercomputing (NAS) Division to help evaluate the performance of parallel supercomputers. They are derived from computational fluid dynamics, unstructured adaptive mesh, parallel I/O, multi-zone, and computational

grid applications. Problem sizes of these applications are predefined and indicated as different classes, e.g., A, B, C, and D. The experiments in this paper focus on the following benchmarks and classes:

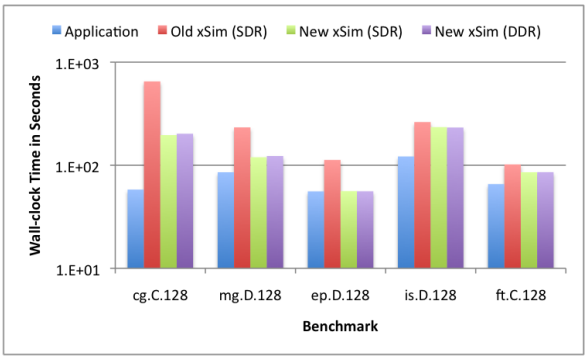
- CG, a conjugate gradient solver with irregular memory access and communication patterns (class C)
- MG, a multi-grid solver on a sequence of meshes with long- and short-distance communication patterns and memory intensive operations (class D)
- EP, an embarrassingly parallel application (class D)
- IS, an integer sort with random memory access patterns (class D), and
- FT, a discrete 3D fast Fourier transform with all-to-all communication patterns (class C)

The selected benchmarks from the NPB suite were executed without the simulator to get a baseline wall-clock execution time, with the old version of xSim, and with the new version of xSim. While the old xSim only supports static deadlock resolution (SDR), the new xSim supports both, SDR and dynamic deadlock resolution (DDR). The benchmarks were executed on 128 physical (native) MPI processes without xSim and on 128 simulated MPI processes using xSim without oversubscription.

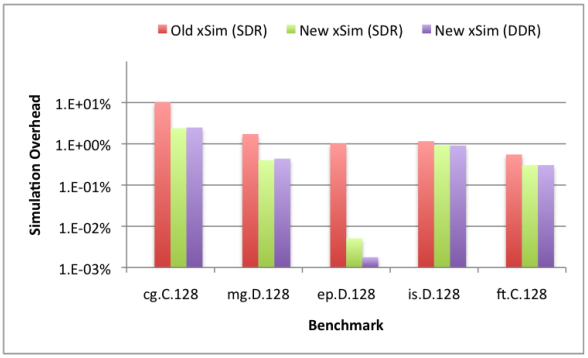
The results in Fig. 4 show that the wall-clock execution time of the simulation was significantly improved. For example, the execution time overhead introduced by xSim was reduced from 1,020% to 238% for the CG benchmark and from 102% to 0% for the EP benchmark. The shown improvements are due the new deadlock resolution protocol only, as the new message matching algorithm only impacts simulations with oversubscription.

Fig. 5 shows the results for the NPB suite using the command line option for accurate process failure simulation. When this option is enabled each simulated MPI message send operation verifies at the outset that the receiving (destination) simulated MPI process has not been declared dead via an injected simulated MPI process failure. A simulated MPI message receive operation simply receives a simulated MPI process failure notification for a failed peer it is waiting on. In the case of simulated MPI message send operations, the timeline for the send and receive must be synchronized to determine if the simulated send operation should fail. This send/receive timeline synchronization is based on the simulated MPI process clock when sender and receiver reside at same physical MPI process, or on the local virtual time of the remote physical MPI process the receiver resides. This results in a significant number of deadlocks and subsequent deadlock resolutions.

The results in Fig. 5 demonstrate the significant reduction in simulation overhead with accurate process failure simulation. For example, the execution time overhead introduced by xSim was reduced from 37,511% to 13,808% for CG and from 3,332% to 204% for EP, both with SDR. Using DDR, additional execution time overhead reductions were measured, such as to 12,871% for CG and to 105% for EP.

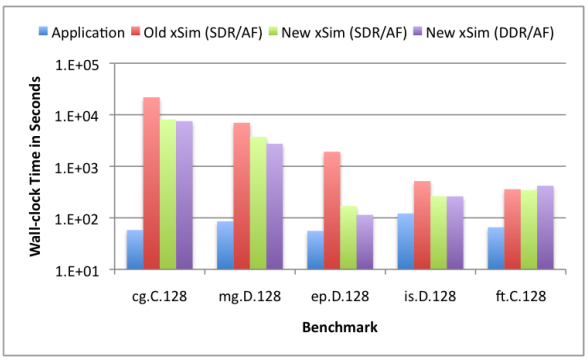


(a) Wall-clock execution time in seconds

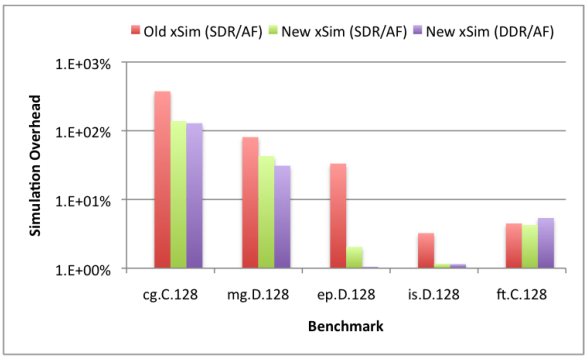


(b) Execution overhead relative to application

Fig. 4. NPB results

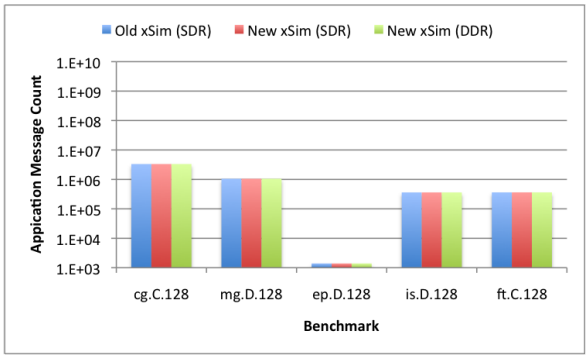


(a) Wall-clock execution time in seconds

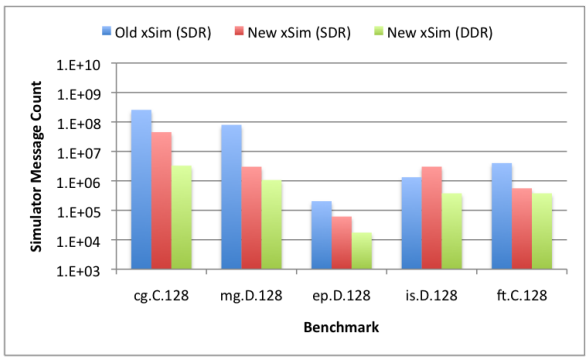


(b) Execution overhead relative to application

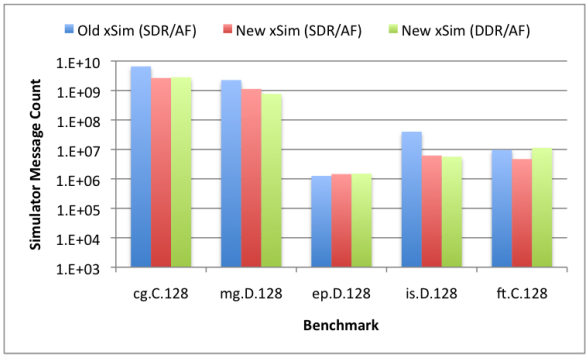
Fig. 5. NPB results with accurate process failure simulation



(a) Application only



(b) Simulator only



(c) Simulator only with accurate process failure simulation

Fig. 6. MPI message counts

The shown improvements are, again, only due to the new deadlock resolution protocol. For FT, however, only a slight improvement could be observed with SDR from 448% to 425% and a slight degradation with DDR to 539%. This is likely due to the all-to-all communication patterns of this particular benchmark.

Fig. 6 shows the differences in the total MPI message count between the application running in xSim and the simulator itself. For simulations without accurate process failure simulation (Fig. 6(b)), the CG, MG, EP, and FT benchmarks show improvements for the simulator's total MPI message count of an order of magnitude or more. For CG, for example, the simulator's total MPI message count was reduced from 256,668,463 to 3,280,382 with DDR. For IS, improvements are only observed when using DDR, reducing the simulator's total MPI message count from 1,326,008 to 377,318. For simulations with accurate process failure simulation (Fig. 6(c)), CG, MG and IS show improvements of up to an order of magnitude. For IS, for example, the simulator's total MPI message count was reduced from 39,937,855 to 5,642,266 with DDR. However, the simulator's message count for EP has not improved. For FT, improvements are only observed when using SDR, reducing the simulator's total MPI message count from 9,673,438 to 4,695,771.

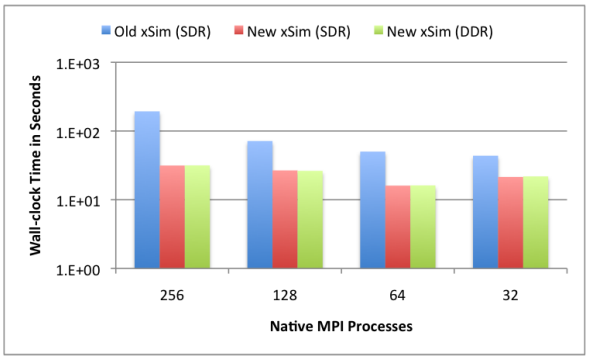
C. *Sweep3D - ScalaBenchGen*

The Sweep3D trace replay benchmark was obtained from earlier experiments, running the Sweep3D application on 256 physical (native) MPI processes on a different Linux cluster with InfiniBand network interconnect. One of the main motivation of the work presented in this paper was that larger scale Sweep3D trace replays were impossible with the old version of xSim due to the high simulation overhead. The following results show the improvements made by running the Sweep3D trace replay benchmark in "application mode", i.e., atop xSim as a normal MPI application, and in "model mode", i.e., atop xSim as a MPI application model. In the model mode, xSim does not transmit simulated MPI message payloads. Instead, only the simulated MPI message envelope is transmitted, saving valuable resources and speeding up the trace replay simulation. As the Sweep3D trace replay benchmark requires 256 simulated MPI processes and the system has only 128 cores, the experiments were performed with native to simulated MPI process ratios of 1:1, 1:2, 1:4, and 1:8, using 256, 128, 64 and 32 physical MPI processes. In the case of 256 physical MPI processes, the 128-core Linux cluster itself was oversubscribed.

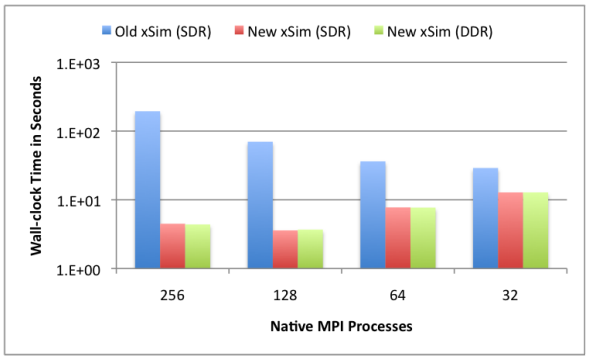
Fig. 7 shows the significant improvements in application mode and the stellar reduction in simulation execution time in model mode, when comparing the old xSim version against the new one. Using 128 physical MPI processes, the new xSim executes the Sweep3D trace replay benchmark in 37% of the old xSim's wall-clock time in application mode, and in 5% in model mode. The shown improvements are due both, the new deadlock resolution protocol and, when employing oversubscription, the new message matching algorithm.

Similar results can be observed with accurate process failure simulation in Fig. 8, where in the same case the Sweep3D trace replay benchmark was executed in 36% of the old xSim's wall-clock time in application mode, and in 79% in model mode. Note that the old xSim was unable to finish the Sweep3D trace replay benchmark on 256 physical MPI processes in application mode within a 4-day period due to the significant amount of deadlock resolutions.

Furthermore, the old xSim had a programing error that resulted in some failures not to be simulated accurately. This was due to a missing synchronization point, which was only discovered after implementing the new deadlock resolution protocol. As this synchronization point is missing in the old xSim, its performance on 256 physical MPI processes in model mode is actually faster than it should be. For executions on

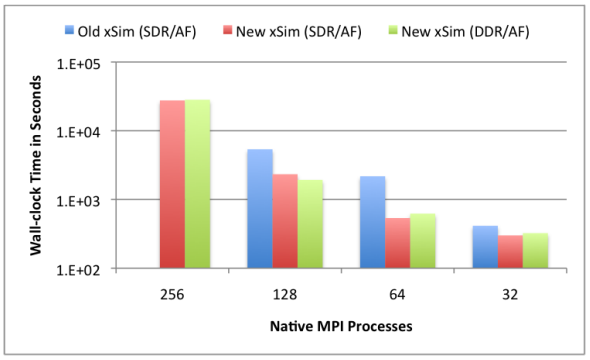


(a) Trace replay in application mode

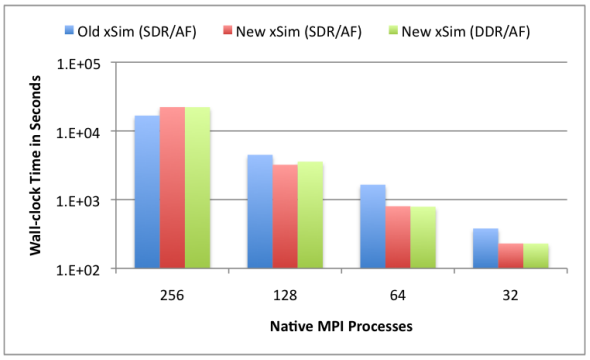


(b) Trace replay in model mode

Fig. 7. Sweep3D results

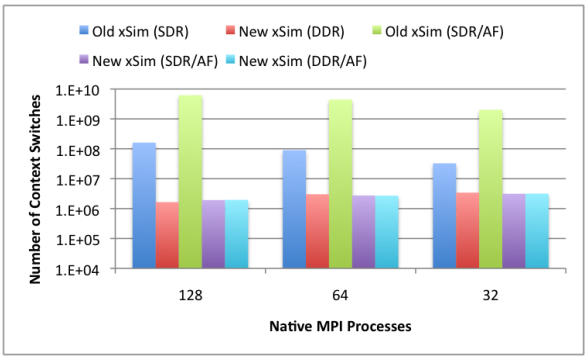


(a) Trace replay in application mode

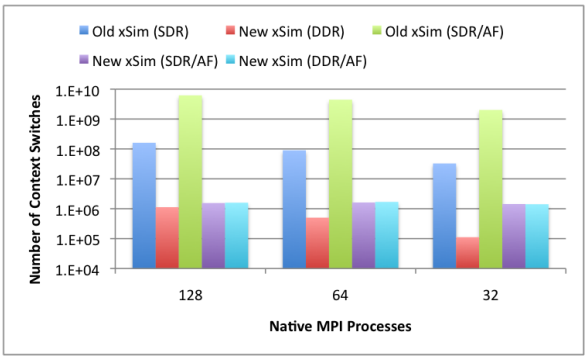


(b) Trace replay in model mode

Fig. 8. Sweep3D results with accurate process failure simulation



(a) Trace replay in application mode



(b) Trace replay in model mode

Fig. 9. Sweep3D simulation context switches

less than 256 physical MPI processes, only slight differences between SDR and DDR can be observed.

Fig. 9 shows the reduction in context switches when employing oversubscription, i.e., using 128, 64 and 32 physical MPI processes. In many cases, the number of context switches could be reduced by an order of magnitude or more. For example, the number of context switches was reduced from 160 million to 1.6 million on 128 physical MPI processes with an oversubscription ratio of 1:2 and SDR in application mode, and from 160 million to 1.1 million in model mode. for the new xSim, there is no significant difference between SDR and DDR with accurate process failure simulation.

VII. Conclusion

This paper documented two improvements to xSim: (1) a new deadlock resolution protocol that reduces the PDES management overhead and (2) a new simulated MPI message matching algorithm that reduces the oversubscription management overhead. The results clearly showed a significant performance improvement, such as by reducing the simulation overhead from 1,020% to 238% for CG and from 102% to 0% for EP, as well as, from 37,511% to 13,808% for CG and from 3,332% to 204% for EP with accurate process failure simulation. This performance improvement enhances the overall productivity of system architects and application scientists when using xSim for HPC hardware/software co-design to estimate application performance on future architectures.

Future work in xSim will focus on a number of aspects, such as on improving the fidelity of the employed processor and network models, adding a file system model, supporting soft error (silent data corruption) injection, and power modeling. Specifically, this paper did not address the accuracy of the simulation, which has been discussed in prior publications, as the presented work was only concerned about the performance of the simulator. Ongoing work targets the improvement of the network model in particular, to improve simulation accuracy by adding better modeling of network contention.

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 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

REFERENCES

[1] T. Naughton, C. Engelmann, G. Vallée, and S. Böhm, "Supporting the development of resilient message passing applications using simulation," in *Proceedings of the* 22^{nd} *Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2014*. Turin, Italy: IEEE Computer Society, Los Alamitos, CA, USA, Feb. 12-14, 2014, pp. 271–278.

[2] C. Engelmann and T. Naughton, "Toward a performance/resilience tool for hardware/software co-design of high-performance computing systems," in *Proceedings of the* 42^{nd} *International Conference on Parallel Processing (ICPP) 2013:* 4^{th} *International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*. Lyon, France: IEEE Computer Society, Los Alamitos, CA, USA, Oct. 2, 2013, pp. 962–971.

[3] 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)*, vol. 30, no. 0, pp. 59–65, Jan. 2014.

[4] ——, "Investigating operating system noise in extreme-scale high-performance computing systems using simulation," in *Proceedings of the* 11^{th} *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2013*. Innsbruck, Austria: ACTA Press, Calgary, AB, Canada, Feb. 11-13, 2013.

[5] 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.

[6] 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.

[7] 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.

[8] OpenMP Architecture Review Board, "OpenMP Application Program Interface version 3.0," 2013.

[9] C. Dekate, M. Anderson, M. Brodowicz, H. Kaiser, B. Adelstein-Lelbach, and T. Sterling, "Improving the scalability of parallel n-body applications with an event-driven constraint-based execution model," *International Journal of High Performance Computing Applications (IJHPCA)*, vol. 26, no. 3, pp. 319–332, Aug. 2012.

[10] STELLAR Group, Louisiana State University, "HPX version 0.9.7," 2013.

[11] 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 *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface*, ser. EuroMPI'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 193–203.

[12] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. R. de Supinski, "Scalatrace: Scalable compression and replay of communication traces for high-performance computing," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 69, no. 8, pp. 696–710, Aug. 2009.

[13] X. Wu and F. Mueller, "Elastic and scalable tracing and accurate replay of non-deterministic events," in *Proceedings of the 27th international ACM conference on International conference on Supercomputing (ICS) 2013*, ser. ICS '13. New York, NY, USA: ACM, 2013, pp. 59–68.

[14] X. Wu, V. Deshpande, and F. Mueller, "Scalabenchgen: Auto-generation of communication benchmarks traces," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS)*, ser. IPDPS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1250–1260.

[15] National Aeronautics and Space Administration, "NAS Parallel Benchmarks," 2014.

[16] C. Engelmann and A. Geist, "Super-scalable algorithms for computing on 100,000 processors," in *Lecture Notes in Computer Science: Proceedings of the* 5^{th} *International Conference on Computational Science (ICCS) 2005, Part I*, vol. 3514. Atlanta, GA, USA: Springer Verlag, Berlin, Germany, May 22-25, 2005, pp. 313–320.

[17] G. Zheng, G. Kakulapati, and L. V. Kale, "BigSim: A parallel simulator for performance prediction of extremely large parallel machines," in *Proceedings of the* 18^{th} *IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2004*. Santa Fe, New Mexico: IEEE Computer Society, Apr. 26-30, 2004.

[18] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng, "Programming petascale applications with Charm++ and AMPI," in *Petascale Computing: Algorithms and Applications*. CRC Press, Dec. 2007, pp. 421–441.

[19] K. S. Perumalla, "$\mu\pi$: A scalable and transparent system for simulating mpi programs," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTools '10. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, pp. 62:1–62:6.

[20] 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.