2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems

Visual Data-Analytics of Large-Scale Parallel Discrete-Event Simulations

Caitlin Ross and
Christopher D. Carothers
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
Email: rossc3@rpi.edu,
chrisc@cs.rpi.edu

Misbah Mubarak, Philip Carns,
and Robert Ross
Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, IL 60439
Email: mmubarak@anl.gov,
(carns,rross)@mcs.anl.gov

Jianping Kelvin Li and
Kwan-Liu Ma
Computer Science Department
University of California, Davis
Davis, CA 95616
Email: kelli@ucdavis.edu,
ma@cs.ucdavis.edu

Abstract*—Parallel discrete-event simulation (PDES) is an important tool in the codesign of extreme-scale systems because PDES provides a cost-effective way to evaluate designs of high-performance computing systems. Optimistic synchronization algorithms for PDES, such as Time Warp, allow events to be processed without global synchronization among the processing elements. A rollback mechanism is provided when events are processed out of timestamp order. Although optimistic synchronization protocols enable the scalability of large-scale PDES, the performance of the simulations must be tuned to reduce the number of rollbacks and provide an improved simulation runtime. To enable efficient large-scale optimistic simulations, one has to gain insight into the factors that affect the rollback behavior and simulation performance. We developed a tool for ROSS model developers that gives them detailed metrics on the performance of their large-scale optimistic simulations at varying levels of simulation granularity. Model developers can use this information for parameter tuning of optimistic simulations in order to achieve better runtime and fewer rollbacks. In this work, we instrument the ROSS optimistic PDES framework to gather detailed statistics about the simulation engine. We have also developed an interactive visualization interface that uses the data collected by the ROSS instrumentation to understand the underlying behavior of the simulation engine. The interface connects real time to virtual time in the simulation and provides the ability to view simulation data at different granularities. We demonstrate the usefulness of our framework by performing a visual analysis of the dragonfly network topology model provided by the CODES simulation framework built on top of ROSS. The instrumentation needs to minimize overhead in order to accurately collect data about the simulation performance. To ensure that the instrumentation does not introduce unnecessary overhead, we perform a scaling study that compares instrumented ROSS simulations with their noninstrumented counterparts in order to determine the amount of perturbation when running at different simulation scales.

I. Introduction

The Time Warp algorithm provides an optimistic synchronization protocol for parallel discrete-event simulations (PDES) [1]. As opposed to conservative synchronization in PDES, in optimistic synchronization each logical process (LP) in the simulation independently processes events without frequent global synchronization among the processing elements (PEs). When the causality constraint is broken (i.e., an LP receives an event that should have happened in its past), the LP must be rolled back to the virtual time immediately before the timestamp of the straggler event, and the events must be executed again.

Because optimistic PDEs involve less global synchronization, they can be more scalable than conservative PDES, provided they are tuned correctly. One way to tune the performance of optimistic PDES is to control the frequency of the global virtual time (GVT) computation. GVT is a global control mechanism provided by Time Warp implementations. It is computed as the minimum of all unprocessed events across all PEs in the simulation. Performing GVT less often means that some PEs can get too far ahead of other PEs, causing the number of rollbacks to increase. On the other hand, computing GVT more often can help control overly optimistic behavior at the cost of increased synchronization overhead.

Although optimistic synchronization with reverse computation has been shown to improve the scalability of PDES [2], choosing the optimal configuration of parameters can be difficult. Many factors can affect the rollback behavior of an optimistic PDES, such as how often GVT is computed and the mapping of LPs to PEs. How the interaction of these factors affects the rollback behavior of optimistic PDES simulations needs to be better studied in order to enable large-scale and long-running simulations.

Rensselaer's Optimistic Simulation System (ROSS) is a Time Warp simulator that uses reverse computation (instead of state saving) as its rollback mechanism [3]. Reverse computation requires the model developer to write a reverse event handler for each event that "undoes" the computation in order to restore the previous state of the respective LP. PDES using reverse computation has been shown to achieve higher performance than state saving [4], [5]. Several performance studies have shown that ROSS can process up to billions of events per second [6], [7], [2].

To better understand rollback behavior and its effect on performance, we have instrumented ROSS to collect detailed information about the simulation engine itself. Previously, ROSS has reported information only at the end of the simulation, such as simulation runtime, the amount of time spent in various stages of the simulation (e.g., GVT computation), number of events, and remote messages. ROSS also provides a measure of efficiency for the simulation to gauge how much time was spent processing real events versus reverse events,

978-1-5090-5218-9/16 $31.00 © 2016 IEEE
DOI 10.1109/PMBS.2016.14

but the granularity is over the full simulation and does not give detailed information as to how the rollback behavior changes over the duration of the simulation. The instrumentation we have added to ROSS can be collected in several ways over the duration of the simulation: immediately after GVT, at real time intervals, and per event. The instrumentation also provides the model developer the ability to change the granularity of the metrics collected (e.g., at the LP level, instead of PE). Letting model developers choose the level of detail allows them to decide between data collection and performance based on their needs.

To use the instrumentation data, we developed an interactive visual analysis tool for visualizing the components of optimistic PDES. It provides several views that are linked together to aid in understanding how the behavior of the simulation entities are connected, and it allows the user to view the data at different levels of granularity. The tool also correlates real time with virtual time and provides a way for the user to zoom in on specific time points in the simulation. We demonstrate the usefulness of the visual analysis tool and instrumentation with a case study using the dragonfly network topology model provided by the CODES simulation framework [8], [9], [10]. CODES is built on top of ROSS and provides a number of models, such as high-performance computing (HPC) network topologies, storage systems, and workloads [11]. Using the visual analysis tool, we visualize the instrumentation data for two 5K node dragonfly models with different settings for the simulation parameters. The visual analysis tool enables us to see differences in the behavior of the simulation engine between these two configurations.

When adding instrumentation to an optimistic PDES engine, the overhead added by instrumentation needs to be minimized in order to avoid becoming a performance bottleneck for the simulation. In addition, the instrumentation can perturb the simulation enough to change rollback behavior and mask performance bottlenecks that would otherwise be experienced in a non-instrumented simulation. In this work, we analyze the amount of perturbation on the simulations by comparing instrumented ROSS with noninstrumented ROSS. We perform experiments with a 5K node and 83K node dragonfly on a Blue Gene/Q to see how the instrumentation affects the runtime and rollback behavior of the model at different scales.

Our contributions are as follows.

1) **ROSS Instrumentation:** We have added three modes of instrumentation in ROSS: sampling at real time intervals, GVT-based sampling, and event tracing. The three modes can be configured to provide data at varying levels of fidelity about the performance of the underlying ROSS simulation engine.
2) **Simulation Performance Visualizations:** Using data provided by the instrumentation, we have developed an interactive visualization tool that gives insight into the rollback behavior of the interconnect simulation model at two scales. The visualizations show what is happening on a global level in the simulation, as well as allowing for detailed local analysis at varying granularities.
3) **Perturbation Analysis:** We have performed a scaling study of our instrumentation work so we can determine the amount of perturbation when running at different simulation scales. We found a 2% average difference in the number of events rolled back for the 5K node dragonfly configurations running on the BG/Q and a -4% average difference in events rolled back for the 83K node configurations.

The rest of the paper is organized as follows. Section II describes related work. Section III provides further background on the ROSS framework. In Section IV, we discuss the instrumentation added to ROSS, along with implementation details. Section V describes the experimental setup for our simulation runs and presents perturbation analysis results. Section VI discusses insights gained from the visualizations of the instrumentation data. Conclusions and future work are discussed in Section VII.

II. Related Work

There are a variety of parallel simulators that have been developed to perform systems simulations and provide different features and parallel implementations. Manifold is a parallel simulation framework that provides both time-stepped and discrete-event simulations, with the ability to use different time scales in multi-scale models [12]. For PDES, Manifold provides several conservative algorithms as well as an optimistic time quantum algorithm, however the time quantum algorithm does not preserve causality across LPs unlike Time Warp. Similarly, the Structural Simulation Toolkit (SST) is another PDES framework developed for large-scale HPC systems simulations [13]. The parallel implementation of the SST is not Time Warp-based, but instead uses pairwise synchronization of MPI ranks for exchanging remote events.

Multiple approaches have been employed to study the performance of parallel programs. The Warwick Performance Prediction (WARPP) simulator uses PDES to generate application performance models of high-performance parallel scientific codes [14]. These performance models can then be used to simulate application codes for problem sizes larger than can be empirically tested on existing machines.

Another approach to performance modeling and analysis of parallel programs is to analyze execution traces, and various tools have been developed to perform such analyses. For example, the KOJAK tool [15], [16] uses parallel execution traces from MPI and OpenMP programs to perform automatic performance analysis and to detect problems. Scalasca, the successor to KOJAK, was developed to be more scalable for analyzing larger-scale systems [17]. ScalaMemTrace is a memory trace compression tool that can generate memory traces with small near-constant sizes and provides on-the-fly scalable trace compression [18].

Some approaches also provide visualization tools to aid in the performance analysis of parallel programs. The visualization environment VAMPIR provides postmortem analysis of MPI traces [19]. Users are able to view aggregate system activity, as well as detailed execution timelines and communication patterns. Jumpshot is another postmortem MPI trace analyzer; it displays MPI processes in a Gantt chart and shows which MPI function calls are being executed over the runtime of the MPI application [20]. The Tuning and Analysis Utilities (TAU) system provides performance visualizations using Gantt

charts, communication matrices, and call graphs. TAU also has tracing capabilities and provides tools to convert traces to other formats for use with other performance tools [21]. The Ravel visualization tool uses logical time instead of physical time to order the events in visualizations, while still applying metrics from physical time [22]. They found that visualizing the trace based on logical time helps to elucidate the communication structure of the application. For more details on these and other approaches, we refer the reader to [23], which provides a survey of approaches in performance visualization.

Although much work has been done in performance analysis of large-scale parallel programs, the tools do not fully suit our needs. Optimistic PDES use MPI for parallel execution; however, the approaches just discussed would not be able to detect problems related to rollback behavior. Previous work in performance modeling and analysis of PDES has been performed in different ways such as using numerical models and metasimulation. Overeinder and Sloot [24] developed an environment with a performance model and tools for performance measurement and parallelism analysis. Their tools determine the critical path and compare the performance metrics collected in order to determine the amount of parallelism that is realized. The approach to performance modeling of PDES taken by Ewald et al. [25] used simulation of PDES executions to predict performance. This simulation could be used to study the performance of an algorithm before implementation. Their predictions agreed with real performance in many cases they examined, despite abstracting away multiple factors such as threading overhead.

Another approach is to develop visual performance tools; however, these have typically been for smaller-scale simulations. For example, Carothers et al. [26] developed PVaniM-GTW as an extension of the PVaniM visualization system for the Georgia Tech Time Warp (GTW) simulator. PVaniM-GTW provides in situ performance modeling of simulations by sampling in real time metrics related to rollback events. The evaluations of PVaniM-GTW were performed with a personal communication services network model with 4,096 LPs on eight workstations. As long as the sampling interval was at least 1 second, the developers found that execution time increased less than 2%. Their monitored runs had 17% fewer rollbacks than did the nonmonitored run, but rollback event processing accounted for only 3% of the computation, so this was deemed acceptable. Our approach differs in the data collected by the instrumentation. We are also able to run our instrumented simulations at a larger scale, while minimizing perturbation at smaller sampling intervals.

Lee et al. [27] developed a trace-based performance analysis and visualization tool called Projections for applications running in CHARM++. Projections shows application performance data in several formats, such as processor utilization, time profiles of CHARM++ methods, and timeline views. It has been used to analyze the performance of the NAMD molecular dynamics application to improve scaling [28]. Although Projections can also be used with PDES applications that run in CHARM++, our work differs in that we focus on visual data analysis to help understand performance as it relates to the rollback mechanism of optimistic PDES.

III. ROSS Optimistic Discrete-Event Simulator

ROSS is an open source discrete-event simulator that provides sequential, conservative parallel, and optimistic parallel event scheduling. Because the optimistic mode of ROSS has been used for large-scale simulations [2], [9], [29], [30], we focus on optimistic execution to study rollback behavior of the simulation engine. During simulation initialization, ROSS preallocates event memory. Because of the rollback mechanism, an event must continue to be stored in memory until the system can be sure it will not be rolled back (i.e., its timestamp is less than GVT). Fossil collection occurs after GVT to reclaim buffers of events that have been committed.

In order to reduce fossil collection overheads, ROSS introduces the kernel process (KP). Each KP manages a shared processed event list for a collection of LPs mapped to a single PE [3]. This results in fossil collection and rollbacks happening on a KP basis instead of an LP basis. Although KPs help mitigate fossil collection overheads, having too many LPs mapped to a KP can result in performance degradation due to false rollbacks. The reason is that when a KP rolls back, it must roll back all its LPs to the same point in virtual time; hence, some LPs could be rolled back unnecessarily. This tradeoff is important to keep in mind when choosing the number of KPs at runtime. The default is set to 16 KPs per PE, which has been shown to be the optimal setting for running the PHOLD benchmark on ROSS [6].

In Time Warp systems, including ROSS, each event has a corresponding cancelation event. The positive and canceled events are both represented as a single event in memory, using a cancel bit to represent its type. Canceled events are sent when a positive event was sent erroneously. In ROSS, each PE is an MPI process, and message passing between PEs is done asynchronously. In each iteration of the main scheduling loop, the PE processes the network queues, inbound event queue, and canceled event queue; computes GVT if necessary; and then performs batch event processing. The `batch` parameter determines the number of events processed in each iteration of the main scheduling loop and affects how often the network is polled for events.

The GVT computation is a global synchronization that allows for reclaimation of event memory. ROSS uses a variation of Mattern's GVT algorithm, which colors events in order to account for all unprocessed events in the system [31]. Further details of the ROSS GVT implementation can be found in [6]. In ROSS, the frequency of the GVT computation is determined by the `GVT-interval` parameter. This determines the number of iterations through the main scheduling loop before the GVT computation is initiated. Thus, the GVT computation occurs every `batch` × `gvt-interval` events, although some situations can force it to occur earlier (e.g., if the simulation runs out of free event buffers).

IV. ROSS Instrumentation

This section describes the three types of instrumentation added to ROSS: (1) metric sampling immediately after each GVT computation, (2) metric sampling over a specified real time interval, and (3) event tracing. In order to provide flexibility to the users, each of these can be used in conjunction

TABLE I
CONFIGURATIONS OF DRAGONFLY NETWORK TOPOLOGY MODEL

Name	p Terminals per Router	a Routers per Group	h Global Channels	Router Radix	Total Routers	Total Terminals	Total LPs
5K	6	12	6	23	876	5,256	11,388
83K	12	24	12	47	6,936	83,232	173,400

with each other or independently to reduce perturbation of the simulation as well as decrease the amount of data output to file. For the GVT and real-time instrumentation, no changes in simulation model code are necessary. However, event tracing requires minor model changes, as explained in more detail in Section IV-C.

For the instrumentation, we need to minimize its overhead for two reasons: (1) to ensure that the instrumentation does not become a performance bottleneck and (2) to ensure that the instrumentation does not mask performance bottlenecks by perturbing the rollback behavior. Because some ROSS/CODES models consume a lot of memory, we are typically unable to store all the instrumentation data in memory for the full simulation. Therefore we store the instrumentation data in a buffer and write to file only when the buffer is close to being full. The size of the buffer can be specified by the user at runtime. After GVT (and any sampling for the GVT instrumentation, if turned on), ROSS checks to see how full the buffer is. By default, if the buffer has less than 15% free space, the buffer is output to file. If the buffer does fill before reaching the next check, a warning is printed. The total amount of bytes missed because of a full buffer is output at the end of the simulation. MPI I/O is used for the file I/O, with all PEs writing to a single file for a given instrumentation type. We perform the I/O immediately after GVT and before event processing resumes. Since the PEs have just synchronized, writing the buffer to file at this point should have the smallest effect on the simulation.

A. GVT Instrumentation

The sampling point for the GVT instrumentation occurs immediately after the GVT computation. This allows for the collection of data about the committed state of the simulation engine. In the GVT-based instrumentation, we collect the counts for the number of forward events processed, events rolled back, total rollbacks, primary rollbacks, secondary rollbacks, network sends, and network receives. Primary rollbacks are those caused when an LP receives an event that should have been processed in its past, while secondary rollbacks are those caused by receiving cancelation events. Either type of rollback can cause one or more events for the LPs on a KP to be rolled back. Events rolled back is the actual number of events rolled back, whereas total rollbacks (the number of primary and secondary rollbacks) is the number of times any KP has had to roll back one or more events.

By default, all this data is collected on a PE basis, but the user can choose at runtime to collect data at a KP and LP level (dependent on the metric being sampled). When this option is chosen, the number of total, primary, and secondary rollbacks is tracked on a per KP basis, because all LPs belonging to a given KP are rolled back to the same point in virtual time. The number of forward events processed, events rolled back, network sends, and network receives is tracked on an LP basis.

B. Real-Time Sampling

For real-time sampling, the various metrics are sampled at real time intervals specified by the user. To turn this sampling on, the user can specify at runtime the interval length in milliseconds. When the simulation is initialized, ROSS converts the interval length to the number of clock cycles. During batch processing of events, a check is performed to see whether enough clock cycles have passed. If so, the PE will perform the sampling.

This instrumentation type collects all the same data as the GVT instrumentation (described in Section IV-A), as well as two sets of additional metrics. The first of these additional metrics is the difference between GVT and a KP's local virtual time (LVT). Since rollbacks happen at a KP basis, we can track this at the KP granularity. A KP's LVT is based on the last event it has processed. KPs with relatively larger values for this metric are operating ahead of the other KPs and are more likely to receive a message that will make them roll back. A relatively lower value for the difference between GVT and KP LVT means the KP is a straggler and is more likely to send events that trigger rollbacks. A KP can have a negative value for this metric (in other words the KP's local virtual time clock is less than the GVT). In this case, the KP has processed no events since the last GVT computation.

The other metric tracked in real-time sampling is the current values of the cycle counters for the different types of processing that ROSS performs. The cycle counters track time spent on GVT computation, fossil collection, forward event processing, rollback processing, and the computation related to managing data structures for the processed and pending events lists. Similarly to the GVT instrumentation, this data (except the amount of time KPs are ahead of GVT) is collected on a PE basis by default. Cycle counters are always collected on a PE basis, but the user can choose to collect the other data on a KP and LP basis as described for the GVT instrumentation.

C. Event Tracing

When event tracing is turned on, ROSS collects some high-level data of the event as to when it is processed. This collected data includes the source and destination LPs, the virtual receive timestamp of the event, and the real time when the event was processed. Some ROSS and CODES models implement multiple types of event handlers, but ROSS knows only whether events are positive or cancel events and is unable to directly determine model-level event types. With event tracing turned on, users can collect information on event types, which requires implementing a small amount of additional code at the model level. In order to allow for the event types to

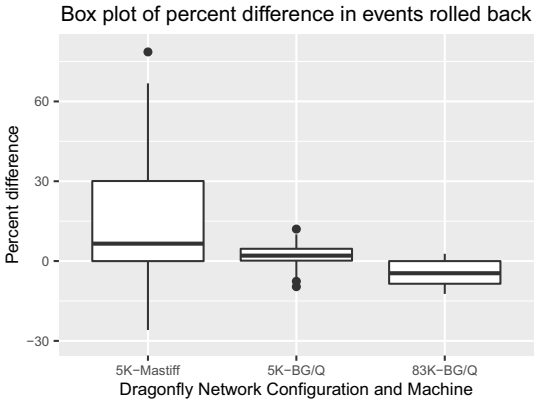


Fig. 2. Box plot of the percent difference in events rolled back between the noninstrumented runs and the fully instrumented runs for the 5K node configurations on Mastiff and the BG/Q, as well as the 83K node configurations run on the BG/Q. Outliers are the data points that are not within 1.5 times of the interquartile range. Two outliers for the 5K node Masiff results are not shown.

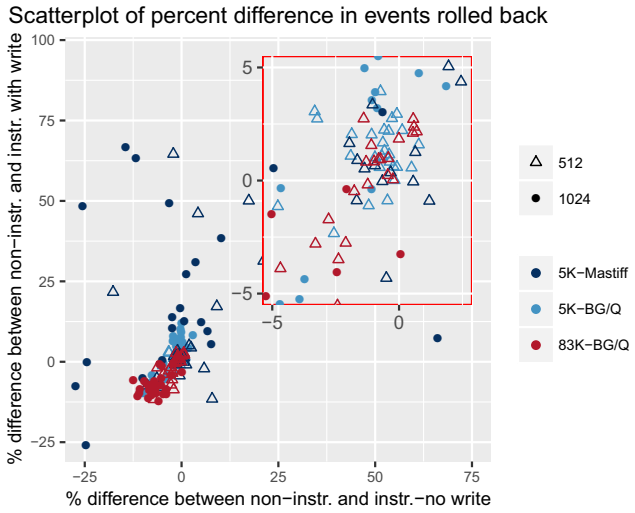


Fig. 3. Scatterplot of the percent difference in events rolled back between the noninstrumented runs and the instrumented without writing the buffer (x-axis) and the noninstrumented runs and the fully instrumented runs (y-axis). Outliers from the 5K node Mastiff results are not shown. The inset is zoomed in on the (0,0) region to see detail.

be collected, a model can implement a function that will store the event type in the buffer. A pointer to this function should be passed to ROSS during the initialization of LPs, along with the size of the data that will be pushed to the buffer. When ROSS saves information about the event that has just been processed, this function will be called, and ROSS will handle the I/O as described previously.

Event tracing has been implemented in a way that allows the model developer to turn on/off the collection on an LP basis. This allows the user to determine which LPs are of interest and thus reduce the amount of data collected. The model developer can choose to use this function to collect other data as well, such as metrics about the model.

V. Experiments

In this section, we describe the experiments performed for evaluating the ROSS instrumentation. First we describe the CODES dragonfly network topology model. Next we discuss the simulation configurations, along with the setup the simulations were performed on. Then we analyze the simulation perturbation in order to determine the effect of instrumentation on the number of rollbacks in the simulation.

A. Dragonfly Network Topology Model

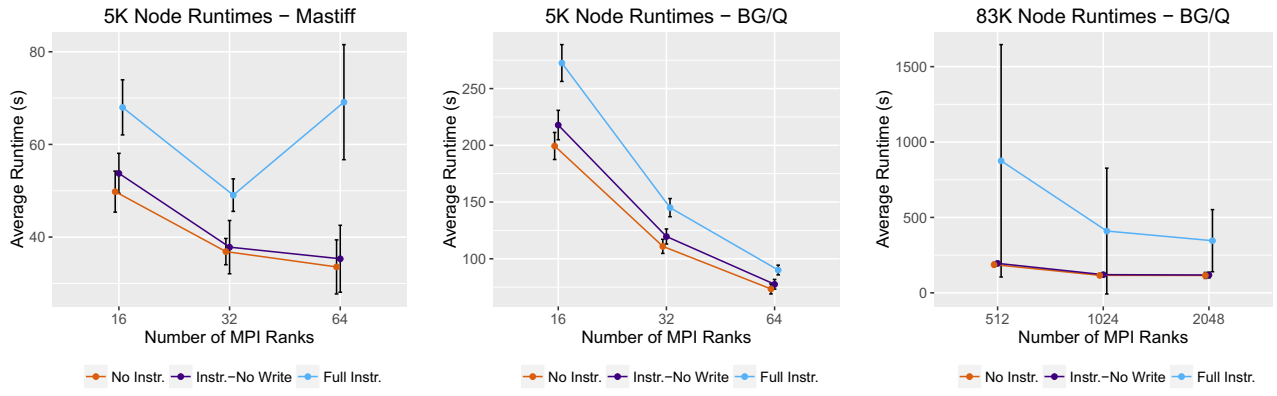
For our experiments, we use the dragonfly model provided by the CODES simulation framework [29]. The CODES dragonfly model is based on the dragonfly network topology proposed in [32], which has a hierarchical structure, with the system being broken into groups of routers. Each router is connected to p terminals, and each group has a routers connected by local channels. Each router also has h global channels for intergroup connections. The radix of each router is $k = p+h+a-1$. The recommended configuration of routers, terminals, and global channels is determined by $a = 2p = 2h$. The number of groups in a given dragonfly configuration is $g = a \times h + 1$, and $N = p \times a \times g$ gives the total number of terminals [32].

The workload for the CODES dragonfly model can be provided by a trace or generated synthetically. For this work, we use the synthetic workload generator with uniform random traffic. With this traffic pattern, each packet randomly chooses a destination node from a uniform distribution. For the routing protocol, we use an adaptive routing algorithm, which allows for dynamic selection of a minimal or nonminimal routing path based on network traffic.

Each of the routers in the network is represented by an LP. Each end point in the network is represented by two LPs: a server LP that generates the synthetic workload and a network node LP that handles the packet send and receive functionality. The server LP generates traffic based on two parameters: the number of messages to simulate and the interarrival time between two messages. The message size is 2 KB per message. For all experiments in this study, we set the number of messages to 20 with each message of 2 KB and the interarrival time between messages to 1000 ns, resulting in 40 KiB of data transferred per rank over an interval of 20 microseconds. For further details about the implementation of the dragonfly network in the CODES and ROSS frameworks, we refer the reader to [29].

B. Setup

For this work, we used two configurations for the dragonfly network size: one with approximately 5K nodes and a second with approximately 83K nodes. The dragonfly configurations are shown in Table I. We performed the simulations on two different systems. The 5K node simulations were performed on Mastiff, a symmetric multiprocessing system with 4 AMD Opteron 6272 processors. Each processor has 16 cores, and the system has a total of 512 GB of RAM. We use 16, 32, and 64 cores for the 5K node dragonfly simulation runs to observe the relationship between simulation performance and number



(a) 5K Node Results - Mastiff

(b) 5K Node Results - BG/Q

(c) 83K Node Results - BG/Q

Fig. 1. Runtimes of each configuration for the 5K node simulations on Mastiff (a) and on the BG/Q (b) and the 83K node simulations on the BG/Q (c). The runtimes are grouped by number of MPI ranks and averaged over the configurations for `batch`, `GVT-interval`, and number of KP parameters. The error bars show the standard deviation. Some jitter is added to the x-axis values for clarity.

of cores. We also ran the 5K node configurations as well as the 83K node configurations on AMOS, a 5-rack IBM Blue Gene/Q system at the Center for Computational Innovation at Rensselaer Polytechnic Institute. We performed the 5K node simulations with 2, 4, and 8 nodes and the 83K node simulations with 64, 128, and 256 nodes. All configurations used 8 tasks per node. Thus, we used 16, 32, and 64 total PEs/MPI ranks for the 5K node configurations and 512, 1024, and 2048 total PEs/MPI ranks for the 83K node simulations.

For both sets of experiments, we varied the number of KPs, `batch`, and `GVT-interval` parameters since these parameters can affect the number of rollbacks in the simulations (discussed in Section III). For the number of KPs, we configured the simulations with 16, 64, and 256 KPs per PE. The `batch` parameter was set to 2, 8, or 16, and we varied the `GVT-interval` parameter based on the selected batch and the number of processed events between GVT computations. We used configurations where `batch` × `GVT-interval` equaled 512 or 1,024 events, since this number has proved effective for many large-scale ROSS simulations. For 512 events, this results in `GVT-interval` values of 256, 64, and 32 for `batch` values of 2, 8, and 16, respectively. For 1,024 events, this results in `GVT-interval` values of 512, 128, and 64 for `batch` values of 2, 8, and 16, respectively.

C. Perturbation Analysis

For the perturbation analysis, we ran each simulation configuration three times: (1) without instrumentation, (2) with all modes of instrumentation turned on, but never writing the buffer to file (the buffer overwrites itself once it is full), and (3) with all modes of instrumentation on and writing to file. We ran the instrumentation in these ways in order to determine how much the instrumentation itself affects the simulation separate from writing out the data. For the real-time sampling, we set the sampling interval to 100 ms. For the event tracing, we turned it on for every LP. Our aim was to study the perturbation of the simulations in an extreme case.

Figure 1 shows the runtimes for the 5K node configurations on Mastiff (a) and the BG/Q (b) and the 83K node configurations (c). For all three sets of configurations, the runtimes of the instrumented runs that do not write the buffer to file are close to the noninstrumented runtimes when compared with the full instrumentation runtimes (i.e., instrumentation including writing the buffer to file). The 5K node simulations have slower runtimes on the BG/Q than on Mastiff because of the slower processor on the BG/Q.

For the instrumented runs with output, the 5K node average runtimes range from 1.3x to 2.1x slower than the noninstrumented for the Mastiff results and 1.1x to 1.4x slower for the BG/Q results. The fully instrumented runtimes for the 83K node configurations are 3–4.7x slower on the BG/Q for their noninstrumented counterparts. Both sets of dragonfly configurations do show strong scaling on the BG/Q, but this is not the case for the 5K node instrumented simulations on Mastiff. The 64-rank runs on Mastiff have more variation in the runtimes than the simulations using a smaller number of ranks. For all simulations performed on the BG/Q, the variation in the runtimes decreases as the number of MPI ranks increase. This is likely due to the fact that Mastiff has a maximum of 64 cores and is a shared system, whereas the BG/Q provides dedicated compute resources for jobs.

To compare the number of events rolled back between instrumented and non-instrumented runs, we calculated the percent difference in events rolled back between the two simulations. A positive difference means more events were rolled back in the instrumented run than in the corresponding noninstrumented run. Figure 2 shows a box plot of the percent difference in events rolled back between the non-instrumented runs and the instrumented runs with writing to file. The 5K node configurations result in more variation in percent difference than do the runs for both sets of configurations on the BG/Q. The outliers for the 5K node case are all configurations with 64 MPI ranks except in the case of one 5K node configuration on the BG/Q, which used 16 MPI ranks.

We also examined the difference in events rolled back

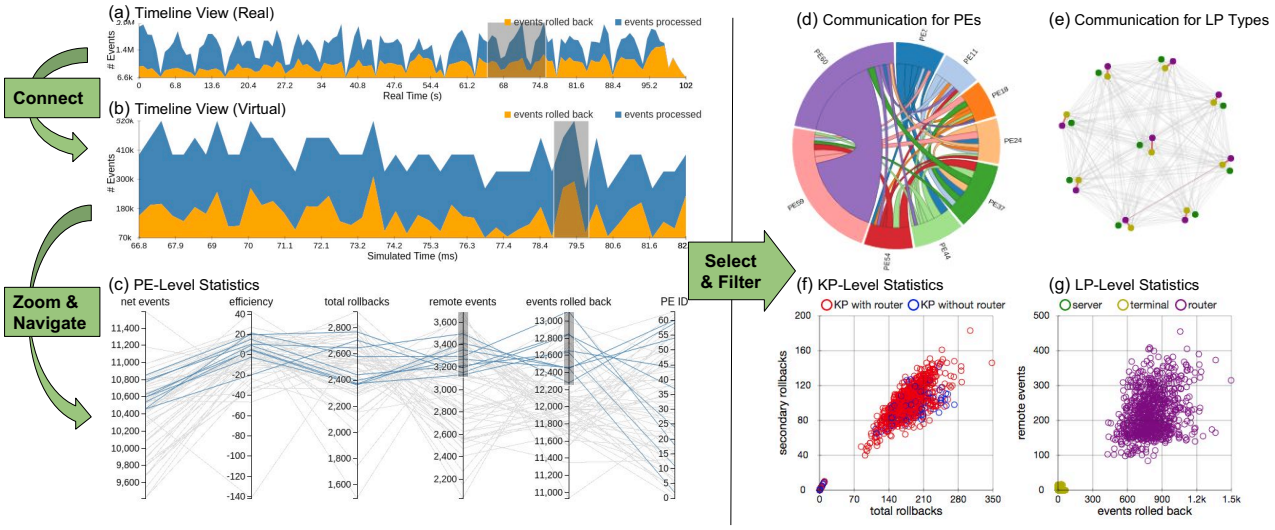


Fig. 4. Interactive visualizations showing the simulation results of a 5K node dragonfly network. Our user interface provides multiple views for interacting with the data collected from instrumented ROSS, allowing users to connect, zoom, and select data from different types of instrumentation and at different granularities.

between the noninstrumented runs and the instrumented runs with no file output. We would expect a positive correlation between these differences and the differences between the noninstrumented runs and the instrumented runs with file output. The results are shown in the scatterplot in Figure 3. On the BG/Q, both sets of configurations show an expected positive correlation in the differences (light blue and red data points). On Mastiff, however, there seems to be no relationship between the two types of instrumented runs for the 5K node simulations (dark blue data points). The inset figure shows the zoomed-in region around $(0, 0)$. We see that the configurations with 512 events processed between successive GVT computations (regardless of the exact `batch` and `GVT-interval` values) are more likely to provide the least difference in events rolled back.

Overall, for the 5K node simulations run on the BG/Q, we find the perturbation in events rolled back to be reasonable, with a 2% average difference in events rolled back from the noninstrumented runs. The magnitude of the average does increase for the 83K node configurations to approximately -4%; however, configuring the event tracing to be more selective should help to decrease that perturbance. The 5K node runs on Mastiff has more configurations with unacceptable levels of perturbance, but this result appears to be partially influenced by the number of MPI ranks used. We therefore suggest on a shared system to run the instrumentation with a smaller number of ranks, if possible. The runtime overhead does increase for all configurations on both machines. This increase is due largely to the file I/O, but this extra cost is incurred during GVT synchronization. The instrumentation configuration we used provided an extreme case. Being more selective in the instrumentation choices should improve the perturbation in both runtime and rollbacks.

TABLE II
CONFIGURATION OF EVALUATED 5K NODE DRAGONFLY SIMULATIONS

Simulation	PEs	KPs	Batch	GVT-Interval	Efficiency
Best Efficiency	16	256	2	256	96.56%
Worst Efficiency	64	16	8	128	27.87%

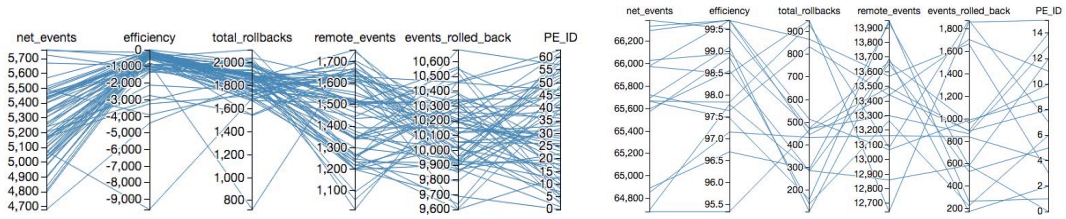
VI. EVALUATION

Using the data provided by the instrumentation of ROSS, we are able to develop interactive visualizations for analyzing the performance and rollback behavior of optimistic PDES. Visualizing various metrics collected by the three types of instrumentation at different granularities with the required time resolution enables the users to explore various aspects of the PDES. For our evaluation, we chose from the simulations performed for the perturbation analysis the two configurations that resulted in the best and worst efficiencies. For ROSS, the efficiency of a simulation is defined below [2].

$$efficiency = 1 - \frac{events\_rolled\_back}{total\_events} \quad (1)$$

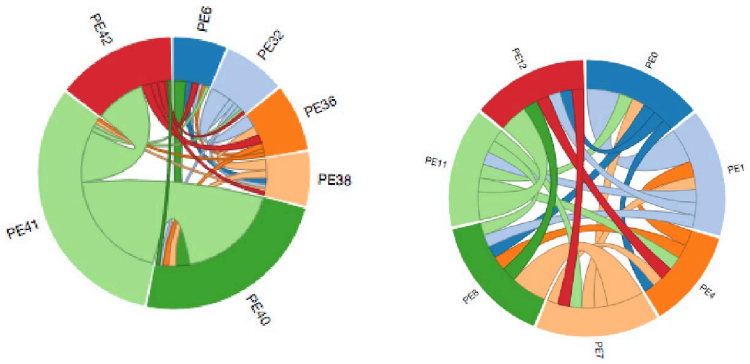
This efficiency metric measures the time a simulation spends performing productive work versus unproductive work (i.e., forward event processing versus rollback processing). Since we want to study rollback behavior with the instrumentation, we focused our evaluation on the configurations that had the worst and best efficiency for the 5K node simulations. Table II shows the configuration settings for both runs.

For the best-efficiency configuration, the noninstrumented and instrumented runtimes were 57.94 s and 77.85 s, respectively. For the worst-efficiency configuration, the noninstrumented and instrumented runtimes were 47.53 s and 75.96 s, respectively. Both configurations have positive efficiencies, so they are spending more time doing forward event processing



(a) Worst Efficiency Configuration (b) Best Efficiency Configuration

Fig. 5. Parallel coordinates for the worst (a) and best (b) efficiency configurations approximately halfway through the simulation runtime.



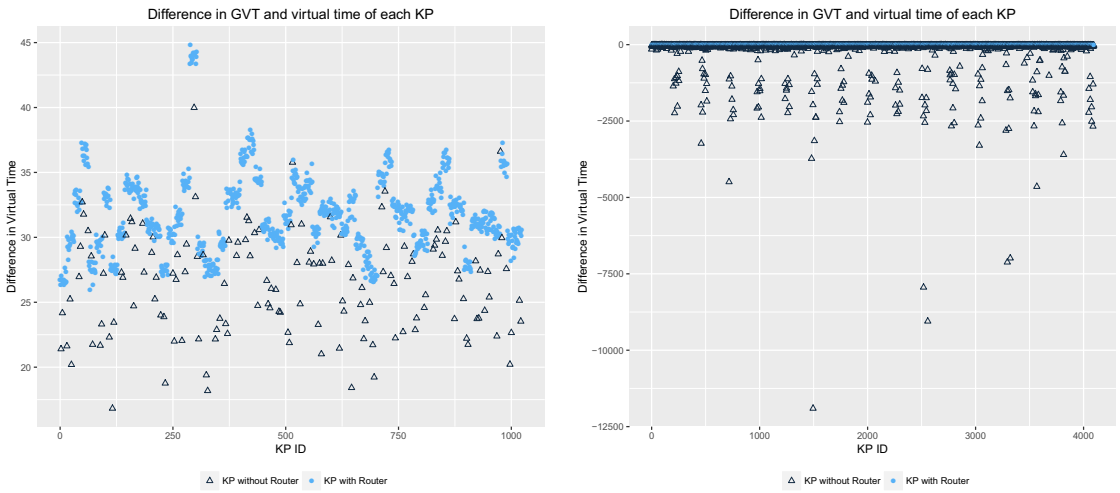
(a) Worst-Efficiency Configuration (b) Best-Efficiency Configuration

Fig. 6. The chord diagram showing the communication pattern between selected PEs for the worst (a) and best (b) efficiency configurations for the 5K node simulations. In each case, the 7 PEs with the lowest efficiencies were chosen for viewing.

than reverse event processing. The worst-efficiency configuration has a faster runtime than the best-efficiency configuration, but this is likely due to the worst-efficiency configuration having 4x more PEs. Since the efficiency metric does not consider other factors that can affect parallel performance (such as running simulation on a different number of cores), the increased efficiency in the best case does not directly result in faster runtime in this case.

Figure 4 shows the user interface of our analysis tool with multiple linked views that we have developed to support effective visual analysis of the data collected from instrumented ROSS. The visualization tool uses data from all three modes of instrumentation described in Section IV. The stacked graph in Figure 4(a) depicts the rollback behavior by showing the events processed versus the events rolled back over real time. Users can interact with this stacked graph to select an interval in the real time domain. At default, Figures 4(b)–(g) show various metrics at different granularities for the entire simulation. Once an interval is selected in the real time domain, all the visualizations are updated to show the corresponding information for the selected time interval. Time intervals in the simulated or virtual time domain can also be further selected by interacting with the stacked graph in Figure 4(b). The parallel coordinates in Figure 4(c) show various metrics collected at the PE basis, which can also be used for selecting PEs for further analysis. The chord diagram in Figure 4(d) depicts the amount of communication between the selected PEs, where the size of the ribbon and arc encodes the number of remote events received. The node-link diagram in Figure 4(e) shows the communication of different LP types, where a node represents all the LPs of the same type within a PE and the color of the edges encodes the amount of communication. The scatter plots in Figures 4(f) and (g) show the primary metrics collected at the KP and LP basis, respectively. Color in the scatter plots is used to encode different KP types (KP with router and KP without router) and LP types (server, terminal, and router). By supporting filtering, zooming, highlighting, and linking between these multiple coordinated views, our visual interface enables users to explore various performance aspects in optimistic PDES. The added instrumentations provide powerful visual data-analytics for analyzing simulation performance.

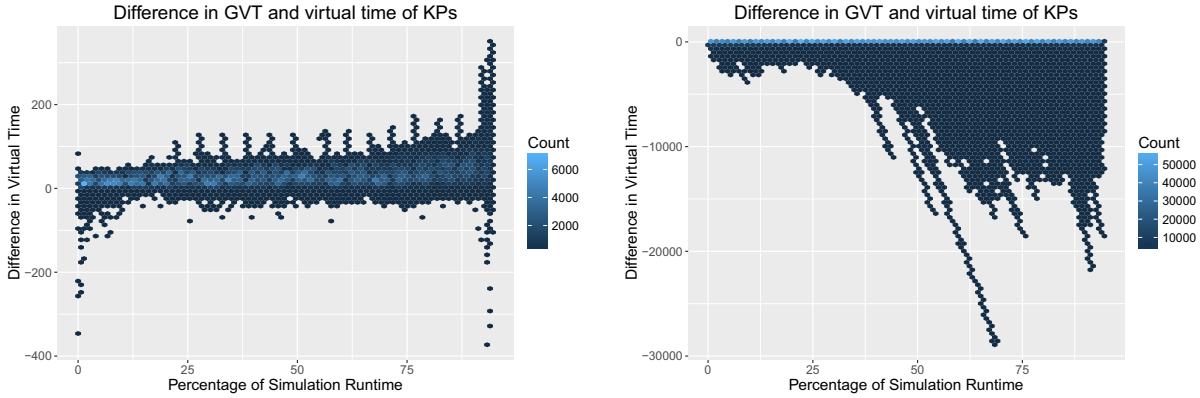
Figure 4 actually shows the simulation results of the 5K node dragonfly network for the case with the worst efficiency. To use our visual interface to analyze this result, we selected an interval in virtual time with high rollbacks (Figure 4(b)), and we also selected the PEs with the most remote events and rollback events using the parallel coordinates. As shown in Figure 4(d), the communication between PEs is unbalanced: the amount of communication between PE59 (pink) and PE60 (purple) is much higher than the communication between other PEs. From Figures 4(f) and (g), we see that the KPs with router LPs are experiencing the most rollbacks as well as sending the most events to other PEs. In Figure 4(e), we see



(a) Worst Efficiency Configuration

(b) Best Efficiency Configuration

Fig. 7. The difference in GVT and KP virtual time at approximately the 50% mark of the simulation for the worst (a) and best (b) efficiency 5K node dragonfly configurations. Light blue circles represent KPs with a router LP and dark blue triangles are KPs without a router LP.



(a) Worst Efficiency Configuration

(b) Best Efficiency Configuration

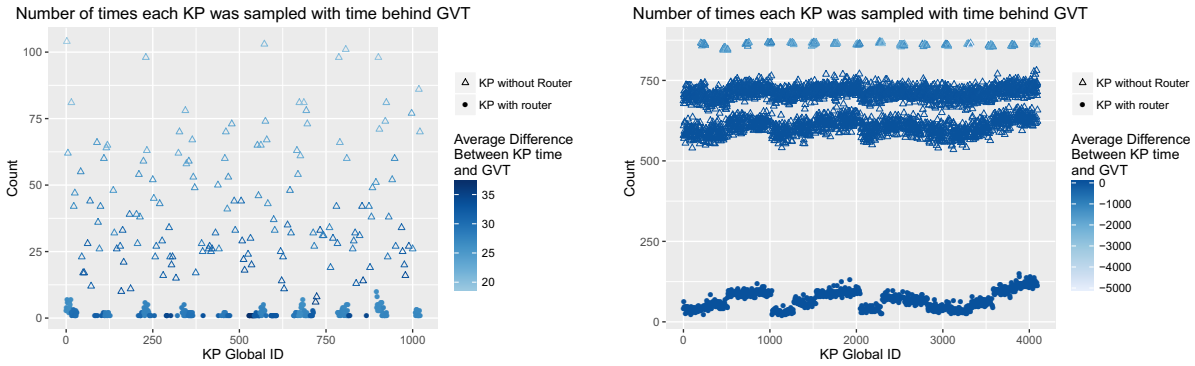
Fig. 8. Difference between virtual time and GVT for each KP taken at real-time sampling points for the worst (a) and best (b) efficiency 5K node dragonfly simulations. The results are binned based on the KP distance from GVT and sampling point in the simulation. Coloring represents the number of observances, with lighter coloring representing a higher count of observations. Both figures show results for the first 94% of the simulation runtime.

that the communication between the router LPs (purple nodes) and terminal LPs (yellow nodes) within the same PE is much higher than with other connections.

Figure 5 shows the parallel coordinates from our analysis tool for the worst (a) and best (b) efficiency scenarios at approximately 50% of the way through the simulation. In the worst-efficiency configuration, many PEs have negative efficiency. We selected the 7 PEs with the lowest efficiency in each case. Figure 6 shows the communication pattern of these PEs. The best-efficiency case (Fig. 6(b)) has more balanced communication than do the PEs in the worst-efficiency case (Fig. 6(a)). In the worst-efficiency case, PE41 has the lowest efficiency and is rolling back the most events at this point in the simulation. This PE is also the one sending the most remote events at this time point, so it is likely sending cancel events, causing secondary rollbacks for other PEs.

To determine how the fast the KPs are moving with respect to the GVT, we graphed the difference in GVT and the local virtual time of each KP. Figure 7 shows the virtual time of KPs relative to GVT at approximately the 50% point of the simulation. Light blue circles represent KPs with a router LP, and dark blue triangles show the KPs without routers. For the worst-efficiency configuration, there is a cluster of KPs belonging to the same PE that are the farthest ahead of all other KPs. At this instant in the simulation, all KPs are ahead of GVT, although the KPs without routers are often straggling behind the others. For the best-efficiency configuration, some KPs without routers are straggling behind, with a few KPs extremely behind GVT.

We also investigated how the KPs move in virtual time



(a) Worst-Efficiency Configuration

(b) Best-Efficiency Configuration

Fig. 9. Number of times each KP was sampled with a virtual time behind GVT for the worst (a) and best (b) efficiency 5K node dragonfly simulations. The difference in virtual time and GVT is averaged for each KP for the first 94% of the simulation runtime and is represented by color. The shape represents whether or not a KP has a router LP assigned to it.

with respect to real time in the simulation. The results, shown in Figure 8, are binned based on the sampling time and the distance from GVT. In the figure, we omit the final 6% of the execution time because the virtual time varies artificially as LPs complete execution and is not indicative of steady-state performance. In the worst-efficiency results, Figure 8(a), most KPs stay within approximately 50 time units of GVT (the lightest blue bins), along with some clusters of KPs that fall behind GVT. As the simulation continues, some KPs also tend to get increasingly farther ahead of GVT. Figure 8(b) plots the variation in virtual time for the best-efficiency configuration, which shows much different behavior. Many KPs are above zero, as shown by the light blue line at $y = 0$. Some groups of KPs, however, fall behind GVT; and the difference gets larger as the simulation continues (note the difference in scale for Figures 8(a) and (b)).

Figure 9 depicts how often each KP was sampled behind GVT, with the worst-efficiency run shown in 9(a) and the best-efficiency run shown in 9(b). Here, the KPs are split into two types: those with a router LP and those without. We separate the KPs into these types because the router LPs will tend to have a higher workload than will the terminal and server LP types throughout the duration of the simulation. The KP ID is in order of the PE ID, so for the worst-efficiency figure, the first 16 KPs belong to PE 0, the next 16 belong to PE 1, and so on. The ordering of KP IDs by the PE ID is the same for the best-efficiency configuration. In the worst-efficiency configuration, 613 of the KPs (59.9%) were never sampled behind GVT and are not shown in the figure. These were all KPs with routers. For the best-efficiency case, all KPs were sampled behind GVT at least once.

In both configurations, each PE has a cluster of KPs that have router LPs that have been sampled behind GVT relatively few times. For the worst-efficiency case, these KPs are also on average the farthest ahead of GVT. In the best-efficiency scenario, the KPs with routers are on average the closest to GVT. The behavior in KPs without routers is different between the two configurations. For the best-efficiency scenario, these KPs appear to form three clusters within each PE. Specifically, each PE has a small cluster of KPs with an average distance farthest from GVT, whereas two clusters of KPs are on average closer to the GVT.

VII. Conclusion

The visual analysis tool, along with the ROSS instrumentation, provides insight into the behavior of the simulation engine that can aid in tuning the parameters that affect simulation performance. This PDES-specific tool provides a way to explore simulation data at different granularities so we can understand the interplay of the different simulation entities during optimistic execution. In our case study, we were able to assess the communication patterns of the PEs at specific time points in the simulation and determine which of those PEs are the likely causes of rollbacks. The instrumentation also enabled us to examine how fast KPs move in virtual time, relative to GVT. From this information, we determined that KPs had different behavior patterns based on the LP types assigned to them. In the future, we will tie this instrumentation into the visual tool in order to have better control of linking other metrics to the KPs variation of its local virtual time. We also plan to refine our visualization tool in order to support simulation models with a larger number of LPs, KPs, and PEs, such as the 83K node dragonfly model we used in the perturbation analysis.

Our perturbation analysis determined that in many configurations run on the BG/Q, the difference in rollbacks between the instrumented and noninstrumented versions is reasonable. This perturbation should improve when choosing a larger time interval for the real-time sampling. Being more selective for the event tracing should play a large part in improving the simulation perturbation, since this is the largest part of the data collected. We plan to explore various avenues for data reduction in the three modes of instrumentation, such as collecting data only when a situation of interest occurs in the simulation. This approach would allow us to collect fine-grained data that will give us insight into the simulation engine behavior without needing to collect such data throughout the whole simulation.

ACKNOWLEDGMENTS

This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computer Research (ASCR), under contract DE-AC02-06CH11357 and DE-SC0014917.

REFERENCES

[1] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, 1985.

[2] P. D. Barnes Jr, C. D. Carothers, D. R. Jefferson, and J. M. LaPre, "Warp speed: Executing Time Warp on 1,966,080 cores," in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2013, pp. 327–336.

[3] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low-memory, modular Time Warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[4] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 9, no. 3, pp. 224–253, 1999.

[5] C. D. Carothers and K. S. Perumalla, "On deciding between conservative and optimistic approaches on massively parallel platforms," in *Proceedings of the 2010 Winter Simulation Conference*, 2010, pp. 678–687.

[6] D. W. Bauer Jr, C. D. Carothers, and A. Holder, "Scalable Time Warp on Blue Gene supercomputers," in *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, 2009, pp. 35–44.

[7] A. O. Holder and C. D. Carothers, "Analysis of Time Warp on a 32,768 processor IBM Blue Gene/L supercomputer," in *2008 Proceedings European Modeling and Simulation Symposium (EMSS)*, 2008.

[8] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Enabling parallel simulation of large-scale HPC network systems," *IEEE Trans. on Parallel and Distributed Systems*, 2016.

[9] ——, "A case study in using massively parallel simulation for extreme-scale torus network codesign," in *Proceedings of the 2nd ACM SIGSIM/PADS conference on Principles of advanced discrete simulation*, 2014, pp. 27–38.

[10] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "CODES: Enabling co-design of multi-layer exascale storage architectures," in *Proceedings of the Workshop on Emerging Supercomputing Technologies*, 2011.

[11] S. Snyder, P. Carns, R. Latham, M. Mubarak, R. Ross, C. Carothers, B. Behzad, H. V. T. Luu, S. Byna, and Prabhat, "Techniques for modeling large-scale HPC I/O workloads," in *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*, 2015.

[12] J. Wang, J. Beu, R. Bheda, T. Conte, Z. Dong, C. Kersey, M. Rasquinha, G. Riley, W. Song, H. Xiao, P. Xu, and S. Yalamanchili, "Manifold: A parallel simulation framework for multicore systems," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, 2014, pp. 106–115.

[13] A. F. Rodrigues, J. Cook, E. Cooper-Balis, K. S. Hemmert, C. Kersey, R. Riesen, P. Rosenfeld, R. Oldfield, and M. Weston, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.

[14] S. D. Hammond, G. R. Mudalige, J. Smith, S. A. Jarvis, J. Herdman, and A. Vadgama, "Warpp: a toolkit for simulating high-performance parallel scientific codes," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 19.

[15] F. Wolf and B. Mohr, "Automatic performance analysis of hybrid MPI/OpenMP applications," *Journal of Systems Architecture*, vol. 49, no. 10, pp. 421–439, 2003.

[16] ——, "Automatic performance analysis of MPI applications based on event traces," in *European Conference on Parallel Processing*. Springer, 2000, pp. 123–132.

[17] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," *Concurrency Computation Practice and Experience*, vol. 22, no. 6, pp. 702–719, 2010.

[18] S. Budanur, F. Mueller, and T. Gamblin, "Memory trace compression and replay for spmd systems using extended prsds," *The Computer Journal*, vol. 55, no. 2, pp. 206–217, 2012.

[19] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach, "VAMPIR: Visualization and analysis of MPI resources," *Supercomputer*, vol. 12, no. 1, pp. 69–80, 1996.

[20] O. Zaki, E. Lusk, W. Gropp, and D. Swider, "Toward scalable performance visualization with Jumpshot," *International Journal of High Performance Computing Applications*, vol. 13, no. 3, pp. 277–288, 1999.

[21] S. S. Shende and A. D. Malony, "The TAU parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[22] K. E. Isaacs, P.-T. Bremer, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, and B. Hamann, "Combing the communication hairball: Visualizing parallel execution traces using logical time," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2349–2358, 2014.

[23] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the art of performance visualization," *EuroVis 2014*, 2014.

[24] B. Overeinder and P. Sloot, "Parallel Discrete Event Simulation Performance Modeling and Evaluation," 1995.

[25] R. Ewald, J. Himmelspach, A. M. Uhrmacher, D. Chen, and G. K. Theodoropoulos, "A simulation approach to facilitate parallel and distributed discrete-event simulator development," in *Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'06)*, 2006, pp. 209–218.

[26] C. D. Carothers, B. Topol, R. M. Fujimoto, J. T. Stasko, and V. Sunderam, "Visualizing parallel simulations in network computing environments: A case study," in *Proceedings of the 29th conference on Winter simulation*, 1997, pp. 110–117.

[27] C. W. Lee, T. L. Wilmarth, and L. V. Kalé, "Performance visualization and analysis of parallel discrete event simulations with projections," Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, Tech. Rep. 05-19, 2005.

[28] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar, "Scaling applications to massively parallel machines using projections performance analysis tool," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 347–358, 2006.

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

[30] N. Wolfe, C. D. Carothers, M. Mubarak, R. Ross, and P. Carns, "Modeling a million-node Slim Fly network using parallel discrete-event simulation," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, 2016, pp. 189–199.

[31] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423–434, 1993.

[32] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, 2008, pp. 77–88.