Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

PARAMETERIZED BENCHMARKING OF PARALLEL DISCRETE EVENT SIMULATION SYSTEMS: COMMUNICATION, COMPUTATION, AND MEMORY

EunJung Park
Stephan Eidenbenz
Nandakishore Santhi
Guillaume Chapuis

Information Sciences (CCS-3)
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

Bradley Settlemyer

System Integration (HPC-5)
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

ABSTRACT

We introduce La-pdes, a parameterized benchmark application for measuring parallel and serial discrete event simulation (PDES) performance. Applying a holistic view of PDES system performance, La-pdes tests the performance factors of (i) the (P)DES engine in terms of event queue efficiency, synchronization mechanism, and load-balancing schemes; (ii) available hardware in terms of handling computationally intensive loads, memory size, cache hierarchy, and clock speed; and (iii) interaction with communication middleware (often MPI) through message buffering. La-pdes consists of seven scenarios for individual performance factors and an agglomerative stress evaluation scenario. The scenarios are implemented through concrete values of input parameters to La-pdes, which include number of entities and events, endtime, inter-send time distributions, computational and event load distributions, memory use distributions, cache-friendliness, and event queue sizes. We demonstrate through instrumentation that La-pdes assumptions regarding distributions are realistic and we present results of the eight scenarios on the PDES engine Simian.

1 INTRODUCTION

Benchmarks are a fundamental tool for measuring and describing the performance of a specific hardware and software combination. Benchmarking and benchmarks are widely used across multiple quantitative disciplines of computer science. The SPEC benchmark suite is used by computer architects to evaluate architectural improvements (e.g. cache sizes and eviction strategies) and used by compiler developers to measure the speedups induced by various optimizations. They are useful because they offer computer architects and compiler writers a simple toolset for evaluating the effects of an optimization or design choice. Within the high performance computing (HPC) community, the LINPACK benchmark is used as the single most relevant metric for ranking the performance of the world's fastest computers (Dongarra et al. 1979, TOP500). Unfortunately, because the LINPACK benchmark is a single application that measures the performance of linear algebra operations on a dense matrix, it is not necessarily representative or predictive in describing the performance for many classes of applications, such as adaptive mesh refinement and sparse matrix operations. Further, as the architecture of HPC systems have become heterogeneous (leveraging general-purpose graphics processing units (GPGPUs) and dense compute co-processors) new benchmarks are emerging that evaluate performance over a much larger range of workloads and hardware designs (Murphy et al. , Dongarra and Heroux 2013).

Within the PDES community, the Phold benchmark (Fujimoto 1990) and the recent benchmark based on Phold (Bahulkar et al. 2010) are important tools for describing the level of parallelism and scalability achieved by a parallel simulation toolkit. However, the Phold benchmark provides only a small degree of control over

the number and types of events occurring within a logical process versus events occurring between logical processes. With the proliferation of heterogeneous computer systems in the HPC community, it is necessary to develop new PDES benchmarks that are capable of representing the complexity and hierarchical nature of emerging compute platforms and their associated programming models. That is, the PDES community needs new benchmarks that evaluate the performance of the models that then simulate these complex, heterogeneous computer systems.

In this paper we describe a new benchmark suite for parallel simulators, *La-pdes*. La-pdes offers fine-grained control over the events processed within an entity, and those requiring service by an additional entity. Further, La-pdes enables the evaluation of a simulator under extremely diverse, yet realistic computation, memory access, and communication workloads. In detail, La-pdes evaluates the performance factors of (i) the (P)DES engine in terms of efficiency of the event queue management, the synchronization mechanism, and load-balancing schemes; (ii) the hardware in terms of handling computationally intensive loads, memory size, cache hierarchy, and clock speed; and (iii) the interaction with the communication middleware (often MPI) in terms of message buffering. La-pdes defines seven different scenarios to evaluate these performance factors individually and one additional scenario for an agglomerative evaluation that touches all aforementioned performance factors. These different scenarios are implemented through concrete values for the twelve input parameters in La-pdes, which can also be set to mimic specific real PDES applications, including agent-based simulations and network simulations. La-pdes parameters consist of the number of entities and events, end time, inter-send distributions, computational and event load distributions, memory use distributions, cache-friendliness, and event queue size. To validate this benchmarking approach, we instrument existing simulation applications and show the La-pdes distributions are realistic. In order to demonstrate the utility of the La-pdes benchmark, we implement the benchmark for a simulation toolkit.

2 RELATED WORK

La-pdes provides different scenarios to evaluate the efficiency of event queue management, handling computationally intensive loads, memory utilization, the synchronization mechanism, the load-balancing schemes, and quality of message buffering. Also one additional agglomerative scenario evaluates all of these factors together. Although benchmarks exist to evaluate (P)DES, to the best of our knowledge, La-pdes is the first benchmark that evaluates various performance factors of (P)DES both individually and in an agglomerative way.

The Phold benchmark is the most popular benchmark for evaluating the performance scalability of PDES, that is ensuring that a PDES engine improves performance as the number of processors and memory are increased across a distributed computer (Fujimoto 1990). The Phold benchmark mainly evaluates a PDES ability to handle computationally intensive loads and event queue scalability; La-pdes provides a more thorough list of scenarios that includes more performance factors than the Phold benchmark. Moreover, La-pdes differs from Phold in that La-pdes uses random variables with user-specified probability distributions to generate communication inter-send times, memory usage, and computational characteristics.

In Liljenstam *et al.*, the authors present a partitioning technique for personal communication systems (PCS), and evaluate the effectiveness of the partitioning technique in terms of balancing the load across multiple SWiMNet simulation instances (Liljenstam and Ayani 1997). Similarly, Thulasidasan *et al.* proposes a partitioning technique to achieve balanced computational load in distributed simulation environments and evaluates the load balancing using real applications including FastTrans (Thulasidasan et al. 2010). Researchers have also focused on the quality of event queue management algorithms as an important area for evaluating simulation engine performance (Yan and Eidenbenz 2006, Rönngren et al. 1991). The La-pdes benchmark recognizes that this is a vital area of PDES research, and thus provides a capability for evaluating these algorithmic advances across multiple application domains easily, without a dependence on detailed knowledge of the internals and implementation of a particular simulation application.

3 LA-PDES BENCHMARK APPLICATION

The La-pdes benchmark application is controlled through a set of twelve parameters that allows the benchmark to mimic the behavior of real discrete event simulation applications. The parameters enable fine-grained control over the numbers of events, the number of entities, the distribution of events to sending and receiving entities, the memory footprint per entity, the computational load incurred by the event handlers, and finally a "dial" to set the cache-friendliness of the selected computational loads. Although we only discuss an La-pdes implementation for the Python Simian PDES engine (Santhi 2015), the La-pdes application can be implemented for any discrete simulation engine.

In constructing our simulation benchmark, we postulate that the assignment of entities to LPs, and any potential entity migration between LPs, is a task that the simulation engine needs to solve for the application, rather than as an input from the application or benchmark. This notion is partially based on our simulation domain, where emerging compilers and programming models dynamically map parallel tasks onto heterogeneous compute resources to increase utilization and efficiency (Thulasidasan et al. 2010, Dagum and Menon 1998, Wienke et al. 2012, Grimshaw et al. 1999). Popular frameworks, such as Metis (Karypis and Kumar 1995), or, more simply, hints provided by the application programmer suffice to influence the entity mapping algorithms of the simulation engine. Interestingly, due to this base assumption, La-pdes is not well-suited to mimic the behavior of Phold, a popular PDES benchmark, which explicitly distinguishes between on-LP and off-LP events.

La-pdes has a simple high-level architecture. Each entity, E_i, has a *SendHandler*, a *ReceiveHandler*, and a local list data structure, L^i, consisting of floating-point value elements. The *SendHandler* is responsible for sending events to destination entities, and upon completing a *send*, the handler re-schedules itself for the next event. The *ReceiveHandler* is responsible for receiving an event and then calculating a specific number of floating point multiplications and additions using the list elements. The time spent in these calculations results in adjustments in the scheduling of additional events. Table 3 shows all of the parameters that make up the La-pdes input set.

We have implemented La-pdes for the Python version of the Simian PDES engine and report results in the subsequent sections. La-pdes as an application is about 250 lines of Python code, which can be ported to other simulation engines in a few days if the work is performed by an expert on the simulation engine. For instance, we have implemented La-pdes versions for Simian Lua (the Lua version of Simian PDES engine), which took an hour of work for the port. Similarly, it took two days of work to implement La-pdes for the C/C++ based MiniSSF (MiniSSF) by a MiniSSF expert. The Python/Lua version of La-pdes are found at the Simian website (Simian). As future work, we plan to implement and test La-pdes for ROSS (ROSS) and NS-3 (NS-3). We are also working towards a more comprehensive instrumentation and statistical analysis of existing DES applications.

3.1 COMMUNICATION CONTROLS

La-pdes provides controls for the *patterns of communication volume* for each entity. Consider an entity E_i, such that $i \in \{0,\ldots,n_{ent}-1\}$. Both a target for the number of messages to send, s_i, and a target for the number of messages to receive, r_i, are provided on a per entity basis.

Temporal communication patterns are controlled by setting an inter-send time interval of τ_i for entity E_i. More formally, let $S = n_{ent} \times s_{ent}$ be the total number of messages in the simulation (where n_{ent} is the number of entities and s_{ent} is the average number of sent events per entity). We define the probability that an entity sends a message as: $p^i_{send} := p_{send}(1-p_{send})^i$ where p^i_{send} is the probability that a message (out of all S messages) is sent by entity E_i. This probability is described by a *geometric distribution*, that is, the probability distribution of the number of messages to choose before selecting a message sent by E_i with probability p_{send}. The messages to be sent for entity E_i are thus: $s_i := p^i_{send} \times S$. Entity E_i computes s_i at initialization. It then computes its inter-send time interval τ_i: $\tau_i := \frac{T}{s_i}$, where T is the end time of simulation, which we set $T := S$.

Table 1: Description of parameters for La-pdes.

Parameter	Default	Value Range	Description
Communication Parameters			
[redacted]	-	$1,\ldots,\infty$	Number of entities
s_{ent}	-	$1,\ldots,\infty$	Average number of *send* events per entity. The total number of expected events is $n_{ent} \times s_{ent}$. Individual entities determine how many events they need to send based on p_{send} and their index. Then they adjust their local $inter-send\ time$ using an *exponential distribution*.
$endTime$	1000	$1.0,\ldots,\infty$	Duration of simulation. Note that *minDelay* is always 1.0 where *minDelay* represents a minimum delay value for synchronization between MPI ranks.
$p_{receive}$	0	$[0,1]$	Parameter for *geometric distribution* of destination entities indexed by entity index. Entity E_i receives a fraction of $p_{receive} \times (1-p_{receive})^{i-1}$ of all the requested messages. Lower-indexed entities receive larger shares. $p_{receive} = 0$: uniform distribution. $p_{receive} = 1$: only entity E_0 receives messages.
p_{send}	0	$[0,1]$	Parameter for *geometric distribution* of sender entities indexed by entity index. Entity E_i sends a fraction of $p_{send} \times (1-p_{send})^{i-1}$ of all the requested messages. Lower-indexed entities receive larger shares. $p_{send} = 0$: uniform distribution; $p_{send} = 1$: only entity E_0 sends messages.
invert	F	{F, T}	Flag to indicate whether *receive* and *send* distribution should be inverted. If it is set to True: the highest-indexed entity sends the most messages.
Memory Parameters			
[redacted]	1	$1,\ldots,\infty$	Average memory footprint per entity, modeled as the average linear list size (8 byte units). Each entity has a local list as a data structure that uses up memory.
p_{list}	0	$[0,1]$	Parameter for *geometric distribution* of linear list sizes. Set to 0 for uniform distribution; Set to 1.0 to make entity 0 the only entity with a list.
q_{avg}	1	$1,\ldots,\infty$	Average number of events in the event queue per entity at a point in time. For individual entities, this is made proportional to the number of total events that the entity needs to send. Default value is 1. Higher values will stress-test the event queue mechanism of the DES engine
Computation Parameters			
[redacted]	1	$1,\ldots,\infty$	Average number of operations per handler per entity. Computational cycle use is implemented as a weighted subset sum calculation of the first k elements of the list with randomly drawn weights. Each entity linearly scales down the number of operations based on its local list size as determined by p_{list}.
ops_{σ}	0	$[0,1]$	Variance of number of operations per handler per entity, as a fraction of ops_{ent}, drawn from *gaussian distribution*.
$cache\_friendliness$	0.5	$[0,1]$	Determines the way to access elements in the list L_i. Setting to p to accesses the first p fraction of list elements; Setting to 0.0 to accesses only the first list element; Setting to 1.0 to accesses all the elements. We set to 0.5 if no other value is known.

Upon being called for the kth time, the *SendHandler* of entity E_i first picks a destination entity E_j by drawing a random index j, where $j \in \{0,\ldots,n_{ent}-1\}$ according to a *geometric distribution* with parameter $p_{receive}$. It then sends the message to the *ReceiveHandler* of entity E_j with a time delay of *MinDelay*, where we set *MinDelay* to 1.0. Finally, the SendHandler determines the inter-send time τ_i^k between sending the kth and the $k+1$th event by drawing a value randomly from an *exponential distribution* with mean value τ_i and schedules its own next execution at time τ_i^k in the future.

The concept of having *send* and *receive* events distributed according to a *geometric distribution* is intended to mimic the observed behavior that a minority of the entities become bottlenecks in DES computations (Pooch and Wall 1992), as these entities handle large shares of the traffic. This behavior usually becomes more pronounced when the simulation model has a number of different types of entities. In standard mode, the lower-indexed entities both *send* and *receive* the highest volumes of communication traffic. However, by using the boolean *invert* parameter, La-pdes allows users to reverse the ordering for the number of *send* events, such that the high-index entities send the most traffic. To avoid this behavior, we set $p_{receive}$ to 0 to ensures all the entities receive an equal number of events, namely s_{ent}. On the other hand, setting $p_{receive}$ to 1 ensures entity E_0 is the only entity to receive all S, which is the total number of messages in the simulation. The same settings apply for controlling p_{send}.

Setting the *endTime* parameter only has an effect on performance while running in parallel mode, as it does not change the total number of events. However, a shorter *endTime* will compress the scheduling time frame, thus allows La-pdes and users to stress-test the underlying message passing system.

3.2 MEMORY CONTROLS

Memory allocation plays a crucial role in parallel discrete event simulation. Large memory requirements are often the main reason to move from a serial DES to parallel DES engine, trumping even CPU cycle concerns. In particular, this is often the case in communication network simulation, where large memory-hungry routing tables need to be maintained. La-pdes allows its user to control the quantity of memory allocated by each entity. We again define an average value of memory allocated per entity m_{ent}, thus resulting in a total memory allocation of $M := m_{ent}n_{ent}$ for the entire simulation. In our Python-based La-pdes implementation, we count memory requirements in units of eight bytes because a float element stored in a Python list is eight bytes. Memory allocation is in the form of a Python list composed of with random float elements. The list $L^i = \{l_0^i,\ldots,l_{m_i}^i\}$ is of length m_i and consists of fixed, but random valued floating point elements. At initialization, each entity E_i computes the number of elements m_i in its list as follows: $m_i := M \times [p_{list}(1-p_{list})^i]$ where p_{list} is the input parameter of a *geometric distribution*. Similar to the other *geometric distributions*: if $p_{list} = 0$, all entities have a list of equal length, specifically m_{ent}. If $p_{list} = 1$, only entity E_0 has a list of length M.

In addition to the memory allocated by entities to execute their event handlers, memory is also used by the simulation engine itself in maintaining the event queue. La-pdes controls the average *event queue length* by defining an input parameter q_{avg}, which is the average number of *SendHandler* events in the queue at any given time that has been generated by an entity. Thus, if run in serial mode, the number of *SendHandler* events in the queue should be $q_{avg}n_{ent}$. Each entity E_i schedules a number q_i of *SendHandler* events into the future, where q_i is a version of q_{avg} proportional to the total number of *send* events s_i that E_i generates: $q_i = \frac{q_{avg}}{s_{ent}}s_i$. Setting $q_{avg} = 1$ results in the classic timer behavior, where the entity just schedules its own *SendHandler*. If setting $q_{avg} = k$, the *SendHandler* generates the kth *SendHandler* event into the future. Making k very large will cause all events to be scheduled at the beginning of the simulation.

3.3 COMPUTATIONAL CONTROLS

Control over the time spent in computation is the third element of La-pdes. Let ops_i be the number of floating point operations that entity E_i needs to execute per received message. To compute ops_i, we need to return to looking at a global system view, rather than a per-entity view. The input parameter ops_{ent}

is the average number of float operations per entity per received message. Let $OPS := ops_{ent} n_{ent}$ be the total number of operations across all entities for their first received messages. We make the number of operations for an entity E_i proportional to its list length m_i as follows: $ops_i := \frac{OPS}{M} \times m_i$ In order to allow for computational load differences among the entities (even on a per received message basis), we actually construct the number of operations as a random variable that we draw from a *normal distribution* with a mean of ops_i as defined above and a variance of $ops_\sigma ops_i$, where ops_σ is an input parameter. For convenience, we define $ops^l = \frac{OPS}{M}$ as the number of operations per list element, making it easy to mimic algorithms that traverse every list element. If $cache\_friendliness$ is set to 1.0, the float operations are performed as a double nested-loop over all m_i list elements each with an inner loop over ops^l iterations, where in each inner loop iteration, a term $r \cdot l^i_j$ is added to an aggregate prefix sum. Variable r is a random float weight and l^i_j is the float value of the list element. This computation is shown in Figure 1.

```
m_i = {active elements in the list}; ops_l = {number of ops per active element};
r = random float weight; sum = 0
for i in range(m_i):
    for j in range(ops_l):
        sum += list[i][j] * r
```

Figure 1: A Pseudo Code for ReceiveHandler.

The choice of a randomly-weighted prefix sum computation is due to the fact that such a doubly-nested loop cannot be optimized away by advanced compilation techniques. On the other hand, it does lend itself nicely to hardware acceleration techniques (such as GPGPU processing elements or vectorized processing units). Since we require an additional floating point random number for each step, our actual FLOP count is at least twice as large and we should perhaps declare the unit of the input parameter ops_{ent} as two floating point operations if the user cares about absolute correctness.

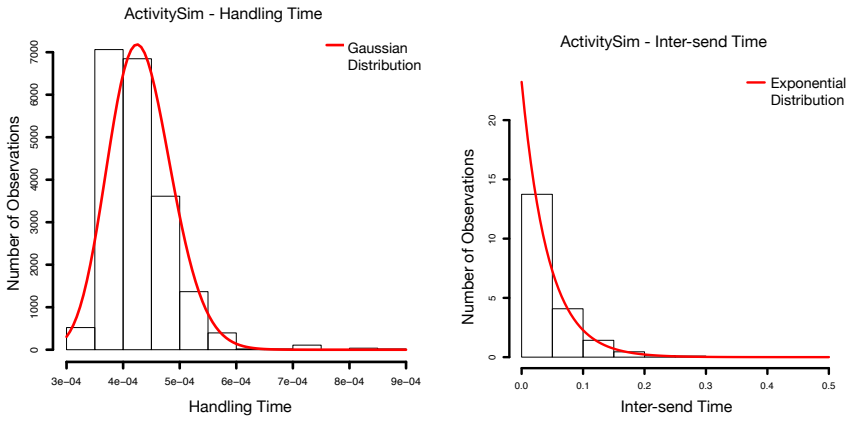
4 STATISTICAL ANALYSIS OF PDES APPLICATIONS

As a first step in validating the La-pdes, we instrumented a set of applications for Simian, and measured their empirical distributions to demonstrate that the provided distribution parameters for La-pdes are realistic and typically found in real PDES applications. The metrics we chose to collect included the following data:

- **nsent/nrecvd:** The number of events sent/received per entity,
- **memuse:** The memory usage per entity,
- **event inter-send time:** The time between each *send* event,
- **handling time:** The time spent in the communication and computation handlers,
- **queue depth:** The number of elements enqueued at every certain time stamps.

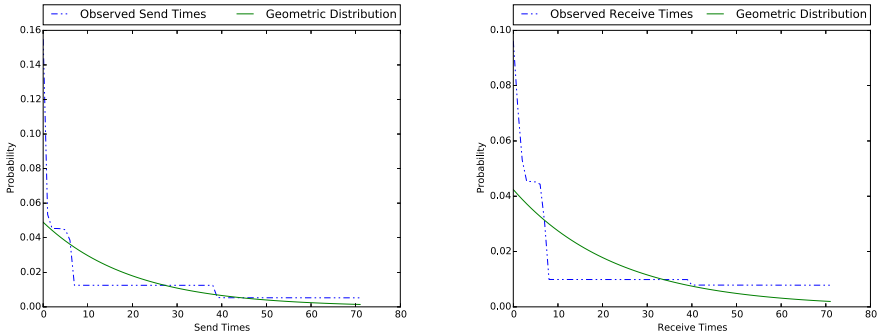
We performed our instrumentation using ActivitySim and Filesim. ActivitySim simulates a large-scale graph algorithm based on independent agents performing both independent and interactive activities (Galli et al. 2009). ActivitySim is representative of computationally intensive algorithms, and thus is representative of many computationally intensive simulation workloads. FileSim, which simulates the complex I/O behavior at scale for parallel file systems (Erazo et al. 2012), exists at the opposite end of the spectrum, and its execution time is dominated by extensive communication traffic between entities.

Figure 2-(a) shows the empirically collected measurements of the time spent in handlers (which are almost exclusively responsible for determining the time spent in computation) overlaid with a gaussian distribution using the empirically observed mean, and the observed standard deviation for ActivitySim. Similarly, Figure 2-(b) shows the event inter-send times overlaid with a exponential distribution. For FileSim, we plotted the observed nsent and nrecvd and the geometric distribution with parameter $p_{est} = \frac{1}{1+\mathbb{E}(\mathbf{X})}$, where X are the observed *nsent* or *nrecvd* as shown in Figure 3. We observed that nsent and nrecvd data in



(a) Handling Time (b) Simulated Inter-send Time

Figure 2: This figure plots the histograms and fitting curves for the observed handling times and event inter-send times for the ActivitySim application. The fitted curve overlaid on each histogram is based on a gaussian distribution for handling time and a exponential distribution for inter-send time, respectively.



(a) nsent (b) nrecvd

Figure 3: This figure plots the observed nsent/nrecvd and fitting curves for the FilsSim application. The fitted curve overlaid on each line graph is based on a geometric distribution with parameter P_{est}, where P_{est} is 0.048 for nsent and 0.042 for nrecvd, respectively.

the instrumented applications best fit a geometric distribution. Based on the empirical distributions found in our selected applications, we configured the distribution parameters for La-pdes benchmark.

5 THE BENCHMARK PROCESS WITH LA-PDES

In order to effectively use the La-pdes benchmark to stress a simulation engine, we have identified eight important parameter configurations that each stress a specific aspect of a PDES system as summarized in Table 5. For our purposes, a PDES engine must provide efficient implementations for the message passing infrastructure (e.g. MPI middleware running on a high performance interconnection network), the computational hardware (including the memory hierarchy, main memory size, and processor frequency), and the simulation engine. The first column of Table 5 describes the PDES aspect emphasized for evaluation in that respective configuration. The second column describes whether the configuration requires an MPI environment to run. That is, the second column describes whether the parallel aspect of the PDES system is stressed. The final column shows the La-pdes input set values used during the benchmark invocation. Any value not shown in the table uses the default values described earlier in Table 3.

Table 2: Eight validation configurations for La-pdes.

Evaluation Category	Test Object	MPI	Configuration
Efficiency of Event Queue Management	DES	No	$q_{avg} = 1, 0.2 \times s_{ent}, 0.8 \times s_{ent}, 0.8 \times s_{ent}, s_{ent}$
Computationally Intensive Loads	H/W	No	$ops_{ent} = 10^2, \ldots, 10^5$
H/W Memory Hierarchy Utilization	H/W	No	$m_{ent} = 10^6$, $ops_{ent} = 10^6$, cache-friendliness= 0, .5, 1.0
Synchronization Mechanisms	PDES,MPI	Yes	MPI rank = 1,4,8,16
Load Balancing	PDES	Yes	$p_{receive} = 0.5$, $ops_{ent} = 10^4$
Load Balancing with Unevenly Dist. Ops.	PDES,H/W	Yes	$ops_{ent} = 10^4$, $p_{list} = 0.5$
Quality of Message Buffering	PDES,MPI	Yes	$endTime = 3, 5, 10, 10^2, ..., 10^6$
Agglomerative Evaluation	PDES, H/W, MPI	Both	$p_{receive}$=.5, p_{send}=.5, *inverse*=T, $endTime$=100, m_{ent}=10^3, p_{list}=.5, $q_{avg} = .4 \times s_{ent}$, ops_{ent} = 1000,ops_{σ} = .3

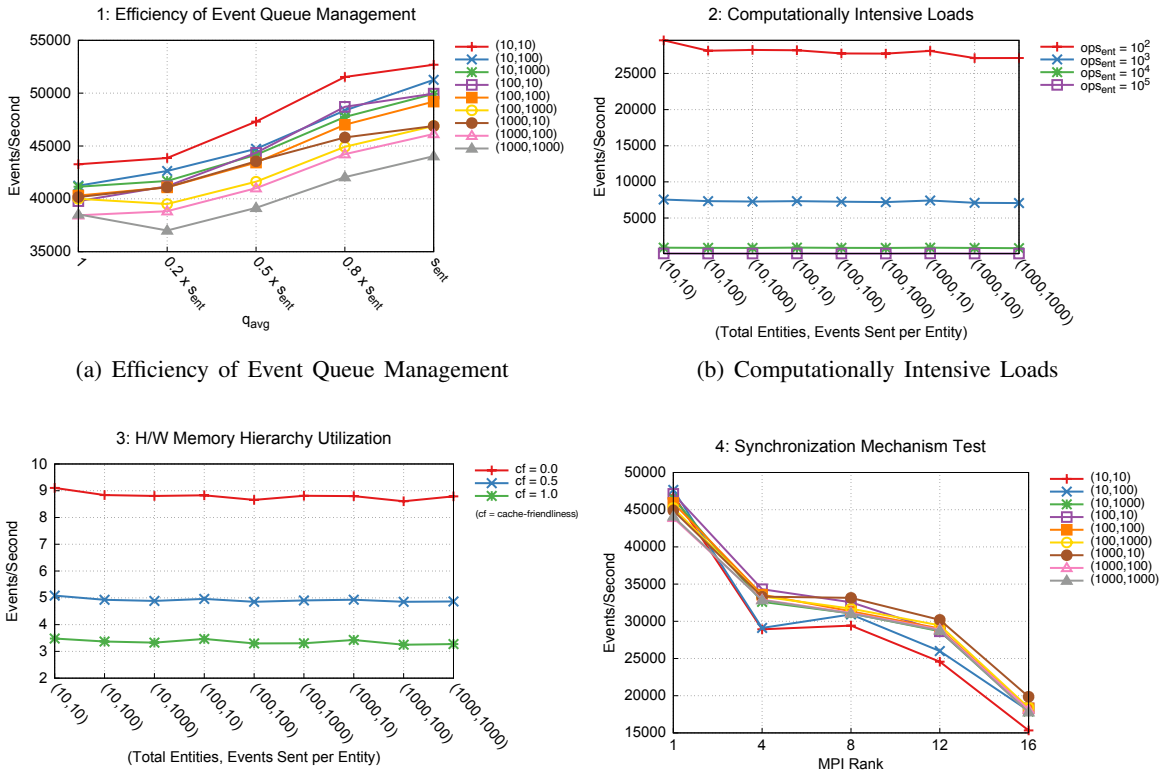
For each of the eight configurations La-pdes reports the number of events per second as the metric for evaluating performance. Fundamentally, the number of simulation events per second of wallclock time is the single best description of the performance for a PDES engine. Events per second is fundamentally analogous to the general computation speed, which is defined as the wall clock time divided by the simulation time. For these measurements, we only seek to observe the steady state behavior, thus long-running executions are typically not necessary to assess the speed of simulation.

Finally, we have identified four of the eight validation configurations as requiring MPI (i.e. executing the PDES in parallel rather than in serial). In order to examine the parallel simulator scalability we evaluate the simulator using 1, 4, 8 and 16 MPI processes. Our validation hardware platform, an Intel Xeon E5-based desktop with 12 cores operating at 2.7 GHz running Apple OSX, had 64GB of main memory and an L3 cache size of 30MB. Because the system has only 12 cores, we are not able to provide a dedicated core in the 16 process MPI configuration. Thus, across all of the multi-process configuration, we are able to assess the PDES scaling in the case where dedicated compute cores exist, and in the case where the total number of compute cores is over-subscribed by the number of simulation processes.

Figure 4 and Figure 5 show the resulting events per second for each of the benchmarking configuration executions La-pdes on the Simian PDES engine. In Figure 4-(a),(b),(c) and Figure 5-(d), the x-axis is the pair (n_{ent}, s_{ent}), which describes the number of entities in the benchmark configuration and the number of events sent per entity. That is, the axis label $(10, 10)$ indicates that 10 total entities sent 10 events each. The x-axis values range from $(10, 10)$ to $(1000, 1000)$. The total number of sent events then scales from 100 to 1000000; however, how these events are distributed among the entities is a key criteria for exploring the simulator performance. In Figure 4-(d) and Figure 5-(a),(b), x-axis represents the number of MPI processes and each pair of the number of entities and the number of events sent per entity are shown as a legend. Figure 5-(c) has a similar format except that it has different values for endTime in x-axis.

Efficiency of Event Queue Management: The benchmarking results shown in Figure 4-(a) demonstrates the efficiencies associated with how the simulation engine manages the event queue. This evaluation does not require MPI, and all parameters are set to their default values except q_{avg}. We configured q_{avg} to each of 1, $0.2s_{ent}$, $0.5s_{ent}$, $0.8s_{ent}$, and s_{ent}. We observe that irrespective of queue sizes, as the number of events sent per entity increases, the total events per second decreases. This is expected because each entity is servicing a larger number of events, leading to longer insertion times. However, we also note that as the queue size increases from 1 to s_{ent} entries, the events per second metric improves. Thus Simian is able to efficiently leverage a larger queue size to achieve a higher overall event rate.

Computationally Intensive Loads: Figure 4-(b), shows the benchmarking configuration we use to determine how well the PDES performs with computationally intensive loads. MPI does not need to be enabled for this configuration, as this type of computation is embarrassingly parallel. In La-pdes, we vary ops_{ent} from 10^2 to 10^5. We limit the upper bound to 10^5 because the time required to complete benchmark simulation run was excessive. We observe that Simian's performance in terms of events per



(a) Efficiency of Event Queue Management

(b) Computationally Intensive Loads

(c) H/W Memory Hierarchy Utilization

(d) Synchronization Mechanisms

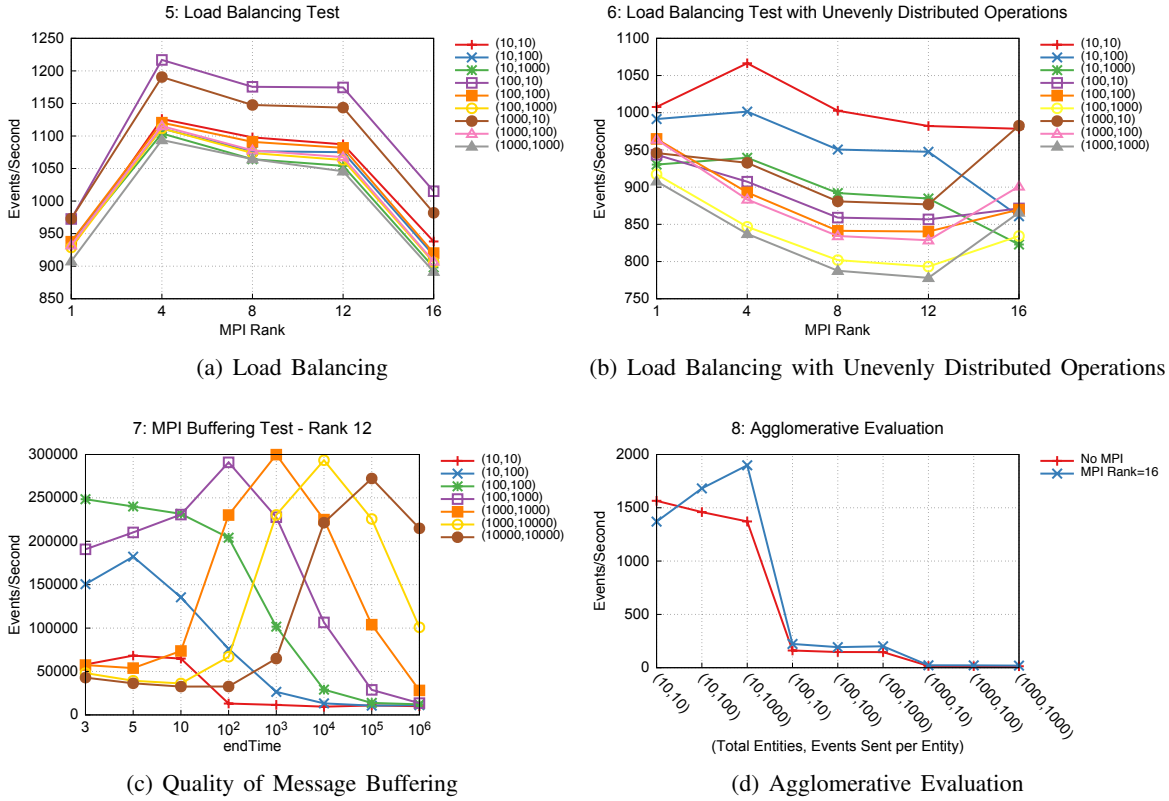
Figure 4: This figure plots the events/second for the first four configurations shown in Table 5.

second decreases as we increase the computational load (however, the performance is largely independent of the number of entities and events). This is natural, as we are increasing the quantity of computational work performed for each event per entity. However, we can observe that Simian's computational performance scaling is very similar send-event scaling demonstrated in the first figure.

H/W Memory Hierarchy Utilization Our third benchmarking configuration measures how efficiently the simulation engine takes advantage of a given memory hierarchy. For this scenario, we evaluated Simian with $m_{ent} = 10^6$, $ops_{ent} = 10^6$, and three different $cache\_friendliness$ parameters: 0.0, 0.5, and 1.0. Recall that a value of 0.0 causes only the first list element to be repeatedly access, while a value of 1.0 results in iteration over the entire list of floating-point elements. Figure 4-(c) then demonstrates that Simian is able to use the available cache independently of the number of entities or events. Further, Simian's performance is highly correlated to the $cache\_friendliness$ parameter, thus Simian does an excellent job simulating both cache-friendly and cache-unfriendly workloads.

Synchronization Mechanisms In order to explore the quality of the synchronization mechanisms used within the Simian PDES, we explore the simulator's performance as we vary the number of processes, and the number of entities and events. In Figure 4-(d) we can see that Simian is leveraging the hardware-level parallelism well, as when we over-subscribed the number of processes per core (a degenerative workload), performance falls dramatically as we add entities and events.

Load Balancing Because Simian provides powerful mechanisms for partitioning the entities among the logical processes, it is critical to examine the quality of the load balancing achieved by the PDES. To measure the load balancing effectiveness, we enable MPI during these runs and configure La-pdes such that $p_{receive} = 0.5$ and $ops_{ent} = 10^4$. This results in each entity receiving an equal proportion of the work, thus Simian must partition the workload so that entities are well balanced across each of the MPI processes. The results shown in Figure 5-(a) demonstrate that as the number of entities and events are increased,



(a) Load Balancing

(b) Load Balancing with Unevenly Distributed Operations

(c) Quality of Message Buffering

(d) Agglomerative Evaluation

Figure 5: This figure plots the events/second for the last four configurations shown in Table 5.

Simian is able evenly divide the work, and provides stable performance. Even in the degenerative case, where there are more processes than cores, Simian is able to provide stable performance.

Load Balancing with Unevenly Distributed Operations While partitioning an evenly distributed load is critical, it is also not a very difficult benchmark on which to excel. We also wish to examine the load balancing aspect of PDES engines when the entity workload is unevenly distributed, a workload that is hard to simulate with existing simulation benchmarks. We achieve this in La-pdes by configuring $ops_{ent} = 10^4$ and $p_{list} = 0.5$. Figure 5-(b) shows that again, Simian is able to achieve a stable performance even while the work per entity is unevenly distributed. We attribute this to Simian's ability to perform entity migration between processes, a critical performance optimization for parallel simulation engines simulating unbalanced workloads.

Quality of Message Buffering La-pdes also allows us to benchmark the message buffering capabilities of the PDES engine. We scaled the benchmark input parameters, so that the endTime ranged from 3, 5, 10, $10^2, \ldots, 10^6$ with 12 MPI tasks. In Figure 5-(c) we see that the results fall into two categories. The smaller simulations decrease in performance as the endTime is increased because a subset of the entities finish their processing before the endTime is reached. On the other hand, larger simulations initially exhibit low performance under the extreme communication workload, however, performance increases as the endTime is increased and the entities are able to balance the load effectively. Even so, given an extreme endTime, subsets of the entities complete processing prior to the endTime and reduce the overall events/second metric. This means that, in general, large simulations will take time proportional to the growth in the population of events and entities, rather than exhibiting a negative scaling factor as the number of processes increases.

Agglomerative Evaluation We created this benchmark configuration as a general-purpose/wildcard test. The agglomerative evaluation seeks to the all aspects of the PDES engine simultaneously, boiling all the previous results into a single evaluation metric. Thus, for this test we examined Simian with both

MPI enabled and disabled. When we enabled MPI, we used 16 MPI ranks (the most challenging parallel configuration since the physical compute cores are over-subscribed), and used the final configuration shown in Table 5. Figure 5-(d) depicts how Simian performed with this configuration. Unlike many of the earlier tests, we see that as the number of entities and events are increased, Simian's performance is reduced drastically. However, if we focus only on the 3 smallest size evaluations, we note that the parallel Simian invocation is trending higher, whereas, in the sequential invocation the events per second is trending lower. Exploring the large knee apparent in these measurements is our next task in improving the performance of Simian's parallel implementation.

6 CONCLUSION

This paper introduces La-pdes, a benchmark for PDES systems. The La-pdes benchmark emphasizes the use of probability distributions in its input parameter set so that message inter-arrivals, memory use, and time spent in computation are governed by random variables, rather than simple fixed values or ratios. We demonstrated that probability distributions are an effective way to mimic real simulation applications by noting the similarity of the distributions generated La-pdes and those created by the real simulation codes ActivitySim and FileSim. We then demonstrated the effectiveness of La-pdes at evaluating simulation engines, by describing how eight different input parameter sets can be used to stress a simulation engines management of event queues, computational efficiency, use of the memory hierarchy, the efficiency of its message passing infrastructure, and load balancing. Finally, we used La-pdes with these input sets to evaluate Simian, a Python parallel simulation engine that implements several advanced parallel optimizations.

This publication has been assigned the Los Alamos National Laboratory identifier LA-UR-15-22468.

REFERENCES

Bahulkar, K., N. Hofmann, D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev. 2010. "Performance evaluation of PDES on multi-core clusters". In *IPDPS*, 131–140. IEEE.

Dagum, L., and R. Menon. 1998. "OpenMP: an industry standard API for shared-memory programming". *Computational Science and Engineering, IEEE* 5 (1): 46–55.

Dongarra, J., and M. A. Heroux. 2013. "Toward a new metric for ranking high performance computing systems". Technical Report SAND2013-4744.

Dongarra, J. J., J. R. Bunch, C. B. Moler, and G. W. Stewart. 1979. *LINPACK Users' Guide*, Volume 8. SIAM.

Erazo, M. A., T. Li, J. Liu, and S. Eidenbenz. 2012. "Toward comprehensive and accurate simulation performance prediction of parallel file systems". In *DSN*, 1–12. IEEE.

Fujimoto, R. M. 1990. "Performance of Time Warp under synthetic workloads". In *Distributed Simulation Conference*, 23–28.

Galli, E., L. Cuéllar, S. Eidenbenz, M. Ewers, S. Mniszewski, and C. Teuscher. 2009. "ActivitySim: Large-scale Agent-based Activity Generation for Infrastructure Simulation". SpringSim '09.

Grimshaw, A., A. Ferrari, F. Knabe, and M. Humphrey. 1999. "Legion: An operating system for wide-area computing". *IEEE Computer* 32 (5): 29–37.

Karypis, G., and V. Kumar. 1995. "Metis ver. 4.0: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices". *Department of Computer Science, University of Minnesota*.

Liljenstam, M., and R. Ayani. 1997. "Partitioning PCS for parallel simulation". In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1997. MASCOTS'97., Proceedings Fifth International Symposium on*, 38–43. IEEE.

MiniSSF. "MiniSSF:Minimalistic Scalable Simulation Framework". Acc. Jun. 15, 2015. https://www.primessf.net/minissf.

Murphy, R. C., K. B. Wheeler, B. W. Barrett, and J. A. Ang. "Introducing the graph 500". *CUG*.
NS-3. "NS-3". Acc. Jun. 15, 2015. https://www.nsnam.org.
Pooch, U. W., and J. A. Wall. 1992. *Discrete event simulation: a practical approach*, Volume 4. CRC Press.
Rönngren, R., J. Riboe, and R. Ayani. 1991. "Lazy queue: an efficient implementation of the pending-event set". In *ACM SIGSIM Simulation Digest*, Volume 21, 194–204. IEEE Computer Society Press.
ROSS. "ROSS:Rensselaer's Optimistic Simulation System". Acc. Jun. 15, 2015. https://github.com/carothersc/ROSS/wiki.
Santhi, N. 2015. "Simian: Just in Time Parallel Discrete Event Simulation Engine". Technical Report LANL-2015-48/LANL-2015-50. Funded by LDRD 20150098.
Simian. "Simian Parallel Discrete Event Simulator". Acc. June. 15, 2015. https://pujyam.github.io/simian.
SPEC. "SPEC:Standard Performance Evaluation Corporation". Acc. Mar. 31, 2015. http://www.spec.org.
Thulasidasan, S., S. P. Kasiviswanathan, S. Eidenbenz, and P. Romero. 2010. "Explicit spatial scattering for load balancing in conservatively synchronized parallel discrete event simulations". In *Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, 150–158.
TOP500. "TOP500 Supercomputer Sites". Accessed Mar. 31, 2015. http://www.top500.org.
Wienke, S., P. Springer, C. Terboven, and D. an Mey. 2012. "OpenACC – first experiences with real-world applications". 859–870.
Yan, G., and S. Eidenbenz. 2006. "Sluggish calendar queues for network simulation". In *MASCOTS*, 127–136.

AUTHOR BIOGRAPHIES

EUNJUNG PARK is a Postdoctoral Research Associate of Information Sciences (CCS-3) at Los Alamos National Laboratory. She holds a Ph.D. degree in Computer Science from the University of Delaware. Her research interests include parallel discrete event simulation, performance prediction, compiler optimizations, and machine learning. Her email address is ejpark@lanl.gov.

STEPHAN EIDENBENZ is a Computer Scientist at Los Alamos National Laboratory. He obtained a PhD from the Swiss Federal Institute of Technology, Zurich (ETHZ) in Computer Science. His research interests include cyber security, computational codesign, communication networks, scalable modelling and simulation, and theoretical computer science. His email address is eidenben@lanl.gov.

NANDAKISHORE SANTHI is a Computer Research Scientist with the Information Sciences Group (CCS-3) at Los Alamos National Laboratory. He holds a PhD in Electrical and Computer Engineering from the University of California San Diego. His areas of research interests include parallel discrete event simulation, performance modeling of HPC systems, applied mathematics, communication systems and computer architectures. His email address is nsanthi@lanl.gov.

GUILLAUME CHAPUIS is a Post-doctoral Fellow with the Information Sciences Group (CCS-3) at Los Alamos National Laboratory. He holds a PhD degree in Computer Science from ENS Cachan (France) and a computer engineering degree from INSA Rennes (France). His research interests include Parallel Discrete Event Simulation, graph theory, High Performance Computing, General-Purpose Graphics Processing Units and bioinformatics. His email address is gchapuis@lanl.gov.

BRADLEY SETTLEMYER is a Research Scientist with Los Alamos National Laboratory's System Integration Group (HPC-5). He holds a PhD in Computer Engineering from Clemson University. His research interests include high-performance storage systems and wide area networking. His email address is bws@lanl.gov.