Asynchronous Approximate Simulation Algorithm for Stream Processing Platforms (WIP)

Veronica Gil-Costa
Universidad Nacional de San Luis
San Luis, Argentina
gvcosta@unsl.edu.ar

Emilio Tapia
Universidad de Santiago
Santiago, Chile
emilio.tapia@usach.cl

Mauricio Marin
CeBiB, DIINF, Universidad de Santiago
Santiago, Chile
mauricio.marin@usach.cl

ABSTRACT
Simulating streaming software applications running on top of a distributed stream processing platform can be useful to evaluate their performance before putting them into production. However, simulating them in parallel is a complex task as the execution of events across simulation processors is not trivial because causality related events can form different sequences distributed on many processors. In this work, we present a window-based simulation algorithm for approximate parallel discrete-event simulation of streaming software applications. The algorithm is suitable for distributed stream computing platforms containing facilities for supporting fully asynchronous processing elements and downstream event flows. Namely, our simulations can be executed on the same streaming platform where the streaming application is intended to be run. The parallel simulation algorithm is based on two time control barriers. The first one is a time window barrier used to process events with time-stamps within the time window. The second one is an oracle time barrier used to define the values of the time window barrier in a manner that is based on the intrinsic features of the system being simulated rather than the current value of the time window barrier of the actual parallel simulation. Both barriers are computed asynchronously. For experimentation, on top of the distributed stream computing platform we simulate a Web search engine. The experiments show that our proposal is capable of efficiently scaling up and at the same time achieving good approximate results with respect to the results obtained from sequential simulations of the same Web search engine model.

Author Keywords
Stream processing platform; Parallel simulation.

ACM Classification Keywords
I.6.4 SIMULATION AND MODELING : Parallel.

INTRODUCTION
Stream processing platforms are useful for performing complex operations on multi-source, high-volume, unpredictable dataflows. They have been designed to deal with the online processing of high volume data streams to achieve real-time data analysis. Currently, there is a number of platforms for stream computing which are available from cloud computing service providers such as SPC (Stream Processing Core) [2], Storm [1], Spark [19] and D-Stream [19].

In particular, the S4 (Simple Scalable Streaming System) [15] is a general-purpose distributed platform designed to analyze massive data processing. The S4 world-view is that streams are passed through a graph (DAG) formed by processing elements (PEs) which are connected to each other in a downstream manner. Each PE performs a given primitive operation on the received stream and generates output streams. Data is routed through the PEs by means of keys, which are specified by users.

Stream computing platforms process millions of events that produce traces with information regarding user's activity. These traces can be used to detect anomalies, to predict the behavior and trends of customers, among other activities that can improve the productivity of a company or institution. In addition, those traces can feed a simulator used to deal with the complexities of understanding and optimizing applications running on the stream processing platform.

In a sequential simulation all events are executed in chronological order and a single clock is updated after the execution of each event. In distributed simulations -as the ones needed for stream processing platforms- this becomes a main difficulty as events are processed in different processors, and each processor has its own local clock. Thus, a major difficulty of Parallel Discrete Event Simulation (PDES) is to efficiently process all events in parallel in global time-stamp order. Synchronizing event execution across processors is not trivial as causality related events can form different sequences distributed on many processors.

In this work we propose an approximate simulation algorithm for an asynchronous distributed stream computing platform attached to a Web search engine infrastructure. In Web search engines, stream computing platforms are usually available for performing operations related to user mining. To control the event time advance in each processor, the proposed algorithm uses two barriers. A time window barrier B used to block events with time-stamps $e.t_r > B$ and an oracle time barrier R used to adaptively compute the values of B across the simulation. The experimental evaluation is performed on the S4 platform which makes the deployment of PEs on the cluster of processors transparent and enables their efficient parallel and distributed execution. The results show that our proposal is capable of reducing in about 80% the running time of sequen-

SCSC 2016, July 24-27, Montreal, Quebec, Canada

tial simulations but it introduces imprecision on the statistics of the simulated model. However the lost in precision is 5,8% at most. To the best of our knowledge, no research has been conducted so far on optimistic and approximate simulation algorithms for distributed stream processing platforms.

The remaining of this paper is as follows. Section introduces preliminary concepts and describes related work. Section presents our asynchronous parallel simulation algorithm. Section presents our experiment results and section presents concluding remarks.

BACKGROUND AND RELATED WORK

Efficient strategies for parallel discrete-event simulation (PDES) have been widely studied in the late 90's and early 2000's literature [8]. In PDES, parallelism is introduced by partitioning the system into a set of concurrent simulation objects called logical processes (LPs). Events take place within LPs, their effect is the change of LP states, and LPs may schedule the occurrence of events in other LPs by sending event messages to them. Once the LPs are placed onto the processors, a major problem is synchronizing the occurrence of events that take place in parallel so that the final outcome is equivalent to that of a sequential simulation of the same events. There are two simulation approaches widely used to solve this problem. Optimistic simulation represented by the Time Warp (TW) protocol and its many variations [8], and a number of conservative protocols [4]. TW is capable of processing events in parallel in correct chronological order by optimistically processing the occurrence of events available in processors and correcting errors that are timely detected. When TW detects that the simulation of events have been missed in the chronological simulation time, a reverse computation called roll-back is executed in the involved processors to re-simulate previous events and to include the missed events in the right chronological order. The conservative protocols ensure safe simulation events by imposing rules on the time of the next events that prevent the late arrival of events in the processors. To this end, the simulation in any given LP is blocked until it can be guaranteed that no event with a smaller time-stamp will later be received.

PDES has deserved recent attention in the context of clusters of multi-core processors [10]. The work in [10] proposes a multi-grained parallelism technique based on the discrete Event System Specification (DEVS) methodology. The work in [14] proposes the DEVS/SOA framework which provides the feature of run-time composability of coupled systems. The work in [5] presented a global schedule mechanism approach for improving the effectiveness based on a distributed event queue. In [7] the authors presented a distributed approach based on the migration of simulated entities. The work presented in [18] evaluated symmetric optimistic simulation kernels for a multi-core cluster which allows dynamic reassignment of cores to kernel instances.

The work in [6] proposed a hierarchical Colored Petri Net-based scheme to represent a Web search engine. The hierarchical approach is useful to build large complex models and parallelism can be captured by distributing modules on LPs to enable parallel simulation on clusters of multi-core processors. The authors in [17] presented a simulator for stream processing systems. However, the proposal is designed for optimistic and exact simulations, and it uses synchronization mechanism to guarantee the correct chronological execution of events. Synchronization mechanisms includes additional costs on the parallel simulation algorithm [11, 6]. The strategy presented in [12] is an early uni-thread processor version of the windowing-scheme presented below in this paper.

The parallel approximate simulator [13], is based on multi-threading and a bulk-synchronous message passing strategy to automatically conduct simulation time advance. The parallelization of execution of simulations is simplified as no roll-backs are considered to correct erroneous computations. The simulation is organized as a sequence of steps. During a step, processors may perform computations on local data and/or send messages to other processors. At the end of a step there is always a synchronization barrier. Messages sent during the current step are available for processing at their destinations at the next step. In each processor there is one master thread that synchronizes with all other $P - 1$ master threads to execute the steps and exchange messages. Then, in each processor and step the remaining $T - 1$ threads synchronize with the master thread to start the next step, though they may immediately exchange messages during the current step as they share the same processor main memory.

Additionally, a key-based approach is used in [13]. Each event has a key which determines the operation to be performed and by means of a hash function it obtains the processor identifier and the thread responsible to simulate that operation. Processing events in parallel during periods of time where no messages from other processors (remote threads or LPs) are received can lead to the problem of missing the arrival of event messages at the right simulation time. The approach presented in [13] simply ignores such situations but proposes a strategy to significantly reduce the arrival of those "straggler" messages.

Stream Processing Platforms: S4

Applications executed on a stream processing platform receive, process and emit events. Typically those events are generated on-line in an unpredictable way. The union of events forms a continuous stream of information that may have dynamic variations in intensity of traffic. In this context, the process used to store and organize/index events in a conveniently way to then process them in batch can be very costly given the huge volume of data and the amount of computational resources required for processing them. But even if this is feasible, it is often desirable or even imperative to process the events as soon as they are detected to deliver results in real time. Stream processing corresponds to a distributed computing paradigm that supports the process of gathering and analyzing large volumes of heterogeneous data stream to assist decision making in real time [3].

In particular, S4 is a stream processing platform which allows applications to process data flows continuously without restrictions [16]. This platform uses Adapter applications to convert external stream into stream of S4 events. These events are routed to Processing Elements (PE) which are the basic units of the platform and messages are exchanged between them. Events are described as a pair (key, attribute). PEs are allocated into processing nodes (PNs) servers. The PNs are responsible for: a) receiving incoming events, b) routing the events to the corresponding PEs and c) dispatching events through the communication layer. The events are distributed using a hash function over the key of the events. Furthermore, the communication layer uses Zookeeper [9] which provides management and automatic replacement clusters if a node fails.

ASYNCHRONOUS APPROXIMATE PARALLEL SIMULATION

In this section we describe an efficient window-based approximate simulation algorithm suitable for distributed stream computing platforms containing facilities for supporting fully asynchronous PEs and downstream event flows. The algorithm removes the roll-back mechanism imposed by classical optimistic approaches like TW, while applying a windowing scheme to restrain optimistic simulation time advance, so that the rate of potential roll-backs is kept very low. This leads to approximate simulations, which is capable of producing overall system statistics which are precise enough. The rewards are simulations of large and complex models that run very fast on clusters of processors which enable their application to on-line capacity planning studies [6].

Each PE of the stream processing platform executes a single LP of the simulation model, and each physical processor can hold one or more PEs. Streams take the form of a collection of "events" that are "emitted" by upstream PEs to create more refined streams containing data for downstream PEs. Stream events are tuples (e, v, d) where e is the type of event, v is a value associated to the e, and d is data associated to e. Upon reception of an upstream event, the PE executes user code which receives the incoming event as input so that it can perform computations on it and emit new events.

The proposed algorithm is based on the use of two barriers: 1) a window barrier named B and 2) a step counter barrier named R which leads to the periodic calculation of a new value for B. The B barrier is used to process events with time-stamps within the time window B. The R barrier is used to estimate the number of steps (oracle steps) executed by the simulation when running in a synchronous way, where each step ends with a synchronization barrier. Thus, the R barrier helps bringing the asynchronous simulation close to the synchronous simulation which tends to reduce the number of stragglers events (events executed in a non-chronological order).

Figure 1 shows the main steps executed by the proposed algorithm. Each event e stores the simulation time at which e is created (t_s) and the occurrence time of the event (t_r). After receiving a new event e, the algorithm checks whether the creation time of the event $e.t_s$ is greater than the current oracle time barrier R. If so, R is updated with the time of occurrence of the event ($e.t_s$) and the number of oracle steps (C) is increased (lines 1-3). The event e is inserted into the EventList,

```
Variables in each PE
$C$   = Oracle step counter in the PE (init $C = 0$).
$R$   = Oracle time barrier in the PE (init $R = 0$).
$C_E$ = step counter in the PE (init $C_E = 0$).
$B$   = Barrier in the PE (init $B = 0$).
$W$   = Window in the PE (init $W = 0.0001$).
$D$   = Distance between C values in the PE (init $D = 10.0$).

PE (streaming event $e$) //A new event e arrives to the PE
1.  if ($e.t_s > R$) then
2.      $R = e.t_r$
3.      $C = C + 1$
4.  EventList.Insert( $e.t_r, e$ )
5.  if ($C \ mod \ D == 0$ ) then
6.      $\Delta$ = simulation time elapsed since last visit here
7.      $W = \Delta / C$
8.  while($EventList.not\_empty()$ and $(C_E - C \leq D)$)do
9.      $e = EventList.Extract()$
10.     if ( $e.t_r > B$ )then
11.         $B = B + W$
12.         $C_E = C_E + 1$
13.     Simulate event $e$
14.     Emit(new events generated by event $e$)
```

Figure 1. Parallel simulation using S4 streaming processing elements.

which is sorted by the occurrence time of the events $e.t_r$ (line 4).

The value of W is automatically calculated as the average simulation time increment in the PE (lines 5-7). W is computed as $W = \Delta/C$. C is the oracle step counter in the PE and Δ is the elapsed time since the last update of W. This computations is performed after N steps.

We control the optimistic execution of PEs imposing that no PE advances beyond D steps with respect to the oracle steps. Thus, if the difference between the real number of steps of the PE (C_E) and the oracle number of steps (C) is greater than a user defined value (D), the event e is not simulated and the control flow of the algorithm goes back to wait for another incoming event (line 8). Otherwise, the first event of the EventList is recovered.

If the time of occurrence of the event ($e.tr$) is greater than the window barrier time (B), it means that the simulation has reached the global barrier synchronization (all PEs have reached the same point of execution of the simulated algorithm), and the window barrier is updated. The initial value of $B = 0$. With this instruction the simulation time is advanced W units of time. Also, the number of real steps is increased by one (lines 10-12). Finally, the event is simulated (line 13) and the algorithm emits the new events generated by e (line 14).

EXPERIMENTAL RESULTS

Case of Study: Web search engine

Typically, Web search engines (WSE) are composed by three services devised to quickly process user queries in an on-line manner: Front-Service (FS), Caching-Service (CS) and

Index-Server (IS). These services are deployed on a large set of processors forming a cluster of computers. They are implemented as arrays of $P \times R$ processors, where P indicates the level of data partitioning and R the level of replication of data. Hence, this architecture makes a high usage of partitioning and redundancy to enhance the query response time and throughput.

In a WSE a query submitted by a user goes through different stages. Initially, it is received by a node of the FS, which redirects the query to a node of the CS. The partition of the CS is selected by applying a hash function on the query terms. The replica is selected in a round-robin manner. The selected CS node checks whether the same request has already been performed and verifies if the result (document IDs) are stored in the cache memory of the server. The CS node can answer to the FS node with a cache hit. In this case, the CS node sends the query results to the FS node which builds the Web page with the query results and sends it to the user. Otherwise, if the CS node sends a cache-miss to the FS node and the FS re-routes the query to all partitions in the IS. Replicas of the IS are selected in a round-robin manner. The IS nodes compute the top-k document results by executing a ranking algorithm. Then, each IS node send the local top-k document results to the FS. Finally, the FS merge the local top-k documents received from the P IS nodes, builds the Web page with the global top-k query results, sends it to the user and sends a message to update the CS.

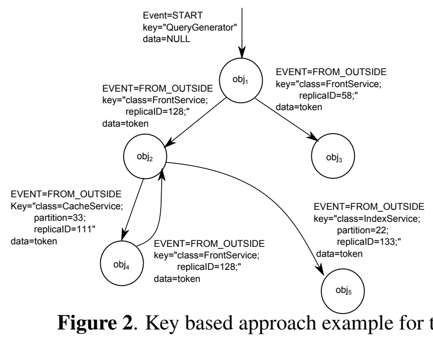


Figure 2. Key based approach example for the S4 platform.

A WSE is deployed on the S4 platform using a key-based approach. Figure 2 illustrates this process. The typical content of a key is a string like "class= Class-Name; instance=ID", say"class= FrontService; replica=128", or "class= IndexService; partitionID=22; replicaID=133". In this example, the only requirement is to specify the class identification field to instantiate the right PE. Figure 2 shows an example where the initial key is generated with the START event which creates a Query Generator object. The other objects are created as soon as the query generator sends events to them. In the figure, obj_1 contains the query generator component, objects obj_2 and obj_3 are Front-Service components, and obj_4 is a Caching-Service node whereas obj_5 is an Index-Service node.

Experimental setup

The proposed asynchronous parallel simulator was evaluated on a cluster of 16 64-bits CPUs Intel Q9550 Quad Core 2.83 GHZ and 4GB DDR3 RAM 1333 Mhz. The evaluation was performed by modeling a Web search engine (WSE). The simulator was tuned to emulate the costs of the relevant operations executed by a WSE. Thus, we run benchmark programs to determine those costs (e.g. merge and ranking), and the hardware related costs (e.g. network communication) [6]. We simulate a total of $Q = 500000$ queries.

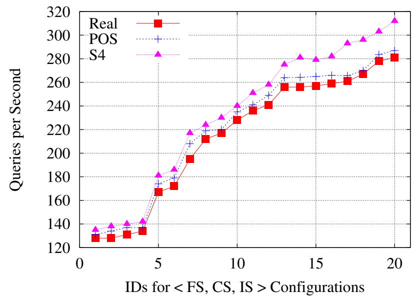


Figure 3. Simulation versus real implementations.

We first validate our proposed asynchronous simulator and then we present experiments performed to evaluate the accuracy and the performance of the parallel algorithm. Figure 3 shows results from sequential simulations of the WSE system described. The simulators were constructed following the above guidelines. The curve labeled POS and S4 stand for process-oriented and the S4 asynchronous simulations. The curve labeled Real shows results from an actual implementation of the same system. The figure shows query throughput results for different configurations given by specific values for the number of replicas of the FS, and the number of partitions and replicas of the CS and IS. In total, these configurations range from 115 to 240 processors. Overall, the results clearly indicate that simulations are able to predict performance trend of the service configurations.

Accuracy Evaluation

In this section we present the query response time reported by the POS, the synchronous approximate algorithm presented in [13] and the proposed approximate asynchronous parallel simulator. We analyze the average query response time metric because it is a sensitive metric. It includes the delays produced by the network latency and the waiting queues inside each services.

Figures 4.(a) and 4.(b) show the average query response time for a small service configuration with 829 nodes and for a large service configuration with 5448 nodes. The x-axis shows query traffic which range from low (at left) to high (at right). For the small system, the lowest query traffic is 400 queries per second and the highest query traffic is 2200 queries per second. For the large service configuration, the lowest query traffic starts with 2000 queries per second and it is increased to 3600 queries per second. Results show that the small system is rapidly saturated affecting query response times. Whereas for the large system, there is still room for increasing the query traffic rate before saturating the system.

Regarding the accuracy evaluation, the synchronous algorithm reports query response times close to the ones presented by the POS algorithm. For a small service configuration, it presents a maximum error of 2,44% and 1,94% in average. For a large service configuration the maximum error is 2,6% and the average error is 2,1%. The proposed asynchronous parallel simulator reports an error of 5,3% at most when executing a small service configuration. The error rises up to 5,8% when executing a large service configuration.

The Pearson Correlation reported by the proposed algorithm is $0,97$ for a small service configuration and $0,91$ for a large service configuration. The values reported by the synchronous algorithm are $0,99$ and $0,88$ respectively. In other words, there is a positive correlation among the results from the sequential (POS) and both approximate parallel simulations.

Figure 4.(c) shows the percentage of straggler events reported by the synchronous and the proposed asynchronous simulation approaches. The experiment was performed for small (829 nodes), medium (2435 nodes) and large (5448 nodes) service configuration systems. The results show that the synchronous algorithm achieves around 30% of stragglers events, meanwhile the proposed algorithm reports 60% of stragglers events in the worst case.

These results indicate that the synchronous algorithm is capable of drastically reducing the number of events executed in a non-chronological order. It reports 30% less stragglers events than the proposal. However, the proposal asynchronous algorithm is a competitive algorithm in terms of effectiveness, as the maximum errors are kept below 6% and the Person Correlation is very high.

Performance Evaluation

In this section we evaluate the performance of the proposed asynchronous parallel algorithm in terms of scalability. In Figure Figure 5.(a) we show the simulation time reported by the asynchronous parallel simulator to simulate the execution of $Q = 500000$ queries. We show results with small, medium and large service configurations. As expected, as queries arrive faster to the simulator, the simulation time tends to decrease. In other words, when traffic is low, the LPs are put to sleep until the next event arrives and the time elapsed between two consecutive query arrivals increments the total simulation time (the clock). But when the query traffic is high, the LPs are always busy simulating the processes required to solve a query. Thus, the time elapsed between two consecutive queries arrivals does not affect the simulation time (clock).

This figure also shows the effect of saturating the different service configurations while increasing query traffic. In particular, for high query traffic the simulation time of the small service configuration is 22% higher than the one reported by the large service configuration and 13% higher than the one reported when executing a medium service configuration.

In Figure 5.(b) we show the total time in nanoseconds reported by the S4 stream processing platform to execute the asynchronous parallel simulation with different number of processors. Execution time was measured with the $System.nanoTime()$ instruction of java. We show the execution time for completing a total of Q=$\{$1K,3K,5K,7K,9K,12K$\}$ queries. We simulate a system with 2100 nodes and a query arrival rate of 1000 queries per second.

Results show that the execution time grows logarithmically as we increase the total number of queries (amount of work). Additionally, the total execution time is reduces with more processors.

Finally, Figure 5.(c) shows the running time reported by the proposed algorithm running on the S4 platform with $P = 16$, the synchronous approximate algorithm presented in [13] running with $P = 16$ and the POS simulator ($P = 1$). The S4 approach drastically reduces by 80% in average the running time reported by the POS simulator, and by 60% the time reported by the synchronous approximate approach. The synchronous approach tends to report higher running times as the WSE model size increases, because it executes more steps to finish the simulation and in each step a synchronous barrier is executed among all involved processors. In addition, this approach compute the value of the window increment W gathering statistic information from all processors, which introduces a communication overhead.

CONCLUSIONS

In this work we proposed and evaluated an asynchronous parallel simulator designed for the distributed stream processing platform S4. The algorithm is designed to control the time advance of events executed in different physical processors. The control mechanism is based on two barriers which includes a window barrier and a step barrier. The first barrier is used to avoid the execution of events with time-stamps too far ahead in the simulation time. The second one is used to periodically calculate a new value for the simulation time barrier. This calculation is based on the event causality dependencies across processors. We evaluate the proposed algorithm using a Web search engine as a case of study. Results show that the proposal is capable of estimating the query response time when the web search engine is running under different conditions with a maximum error of 5,8%. Additionally, the algorithm is capable of scaling with more processors ($P = 16$) as query traffic (workload) and the service configurations are increased. In the near future, we plan to evaluate more cases of studies including social networks and others stream processing platforms like Storm and Spark.

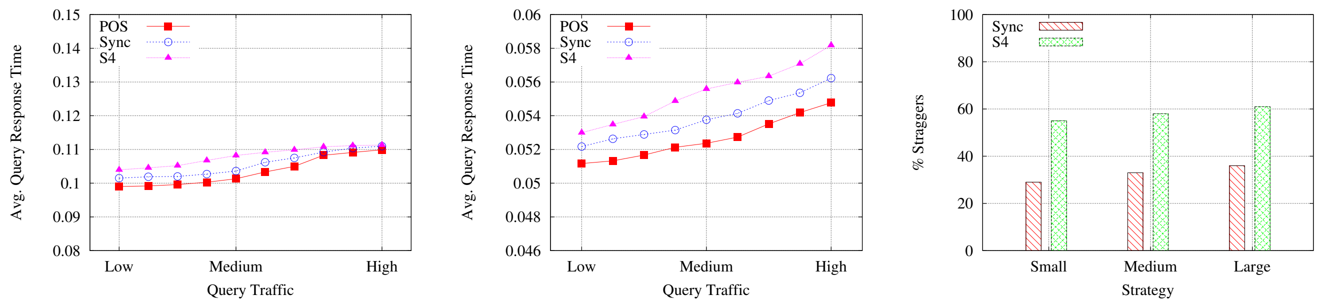
ACKNOWLEDGMENTS

This work has been partially funded by CONICYT Basal funds FB0001.

REFERENCES

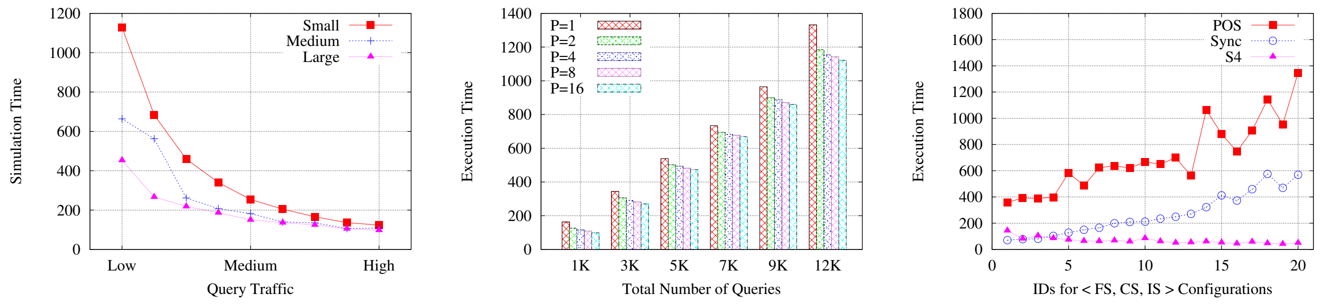
1. Storm. [Online]. Available: https://github.com/nathanmarz/storm/wiki.

2. Amini, L., Andrade, H., Bhagwan, R., Eskesen, F., King, R., Selo, P., Park, Y., and Venkatramani, C. Spc: A distributed, scalable platform for data mining. In *DM-SSP* (2006), 27–37.



(a) (b) (c)

Figure 4. Query response time for (a) small and (b) large service configurations. (c) Percentage of stragglers events.



(a) (b) (c)

Figure 5. (a) Execution time with (a) different query traffic, (b) different number of queries and (c) POS vs. S4.

3. Andrade, H., Gedik, B., and Turaga, D. *Fundamentals of Stream Processing Applications Design*. System and Analytics, Cambridge University Press, 2014.
4. Boukerche, A., and Das, S. K. Dynamic load balancing strategies for conservative parallel simulations. *SIGSIM Simul. Dig. 27*, 1 (1997), 20–28.
5. Chen, L.-l., Lu, Y.-s., Yao, Y.-p., Peng, S.-l., and Wu, L.-d. A well-balanced time warp system on multi-core environments. In *PADS* (2011), 1–9.
6. Costa, V. G., Marín, M., Inostrosa-Psijas, A., Lobos, J., and Bonacic, C. Modelling search engines performance using coloured petri nets. *Fundam. Inform. 131*, 1 (2014), 139–166.
7. DAngelo, G., and Bracuto, M. Distributed simulation of large-scale and detailed models. *IJSPM 5*, 2 (2009), 120–131.
8. Fujimoto, R. Parallel discrete event simulation. *Comm. ACM 33*, 10 (1990), 30–53.
9. Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIXATC* (2010), 11–11.
10. Liu, Q., and Wainer, G. A. Multicore acceleration of discrete event system specification systems. *Simulation 88*, 7 (2012), 801–831.
11. Marín, M. An evaluation of conservative protocols for bulk-synchronous parallel discrete-event simulation. In *European Simulation Multiconference* (2000), 83–90.
12. Marín, M. Acontrolling optimistic execution in bulk-synchronous parallel discrete-event simulation. In *SCS European Simulation Symposium* (2001).
13. Marín, M., Gil-Costa, V., Bonacic, C., and Solar, R. Approximate parallel simulation of web search engines. In *SIGSIM-PADS* (2013), 189–200.
14. Mittal, S., Risco-Martn, J. L., and Zeigler, B. P. Devs/soa: A cross-platform framework for net-centric modeling and simulation in devs unified process. *SIMULATION 85*, 7 (2009), 419–450.
15. Neumeyer, L., Robbins, B., Nair, A., and Kesari, A. S4: Distributed stream computing platform. In *ICDMW* (2010), 170–177.
16. Neumeyer, L., Robbins, B., Nair, A., and Kesari, A. S4: Distributed stream computing platform. In *ICDMW* (2010), 170–177.
17. Park, A. J., Li, C.-H., Nair, R., Ohba, N., Shvadron, U., Zaks, A., and Schenfeld, E. Towards flexible exascale stream processing system simulation. *Simulation 88*, 7 (2012), 832–851.
18. Vitali, R., Pellegrini, A., and Quaglia, F. Towards symmetric multi-threaded optimistic simulation kernels. In *PADS* (2012), 211–220.
19. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. Discretized streams: Fault-tolerant streaming computation at scale. In *SOSP* (2013), 423–438.