Received: 4 July 2016 | Revised: 15 November 2016 | Accepted: 25 February 2017
DOI: 10.1002/cpe.4149

WILEY

SPECIAL ISSUE PAPER

Semi-asynchronous approximate parallel DEVS simulation of web search engines

Alonso Inostrosa-Psijas[1] | **Veronica Gil-Costa**[2] | **Mauricio Marin**[1,3] | **Gabriel Wainer**[4]

[1]CITIAPS, Universidad de Santiago de Chile, Santiago, Chile
[2]CCT-CONICET and National University of San Luis, Argentina
[3]CeBiB, DIINF, Universidad de Santiago, Chile
[4]Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada

Correspondence
Alonso Inostrosa-Psijas, University of Santiago, Chile.
Email: alonso.inostrosa@usach.cl

Summary

Discrete Event System Specification (DEVS) is a formalism for the modeling and analysis of discrete event systems. Parallel DEVS (PDEVS) is an extension of DEVS for supporting Parallel and Discrete Event Simulation, which is a powerful tool for evaluating the performance of large scale systems. In this work, we propose an optimistic approximate and semi-asynchronous parallel strategy. The level of optimism is efficiently managed throughout the simulation execution, and it is automatically adjusted based on the simulation evolution. Load balance and model partitioning is automatically made by means of an algorithm that takes advantage of the communication pattern of the simulated model. Our proposal is designed for Web search engines, which are complex and highly optimized systems devised to operate on large clusters of processors and dealing with dynamic and unpredictable user query bursts. The results show that our proposal is able to reduce both execution times and memory usage of standard optimistic simulations of Web search engine models, at the expense of small errors.

KEYWORDS
approximate parallel simulation, DEVS, PCD++, web search engines

1 | INTRODUCTION

Sequential discrete-event and process-oriented simulations have proved to be useful for performance evaluation studies of small scale models. However, for studies of large scale models, parallel discrete-event simulation (PDES) has the potential to provide better results, because it allows overcoming critical resource constraints such as processing time and memory footprint of the simulation code. A PDES program is normally built as a collection of logical processes (LPs), each of which is in charge of simulating a portion of the complete model under study. Logical processes communicate with each other by exchanging time-stamped event messages. Processing events in parallel in global time-stamp order is complex, and synchronization of the time-stamps of the different events executing across processors is not trivial since causality related events might form different sequences.

There have been 2 major approaches for solving the causality problems caused by these parallel and distributed simulations, the conservative[1] and the optimistic protocols.[2] The main purpose of both methods is to guarantee the event causality constraint. The conservative protocols ensure a safe simulation of events with no causality errors by imposing time-related rules that prevent the late arrival of events to 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 be received later in the LP. On the other hand, the optimistic protocols (in particular Time Warp—TW—and other similar approaches) process all the events available in processors and if there are causality errors they are corrected. When TW detects that the simulation of events has been missed in the chronological simulation time, a reverse computation called rollback is performed in the involved processors to resimulate previous events, but this time including the missed ones in the right chronological order.

In Marin et al,[3] the performance of TW was increased by removing the rollback mechanism while applying a windowing scheme to restrain optimistic simulation time-advance to keep the rate of potential rollbacks very low. This leads to approximate simulations. The benefits of this are very fast simulations of large and complex models, executed on clusters of processors, which enable their application to online capacity planning studies.

In this work, we propose an optimistic approximate and semi-asynchronous parallel simulation strategy for parallel Discrete-Event System Specification (PDEVS) models. Discrete-Event System Specification is a simulation worldview offering a number of advantages for modeling and simulating systems,[4,5] such as hierarchical and modular development of the model, a formal specification mechanism and a way to specify models that enables them to be embedded into ones of higher level. Several implementations of DEVS can be found

since it is independent from the simulation mechanism, one of them is CD++.[6]

In particular, our proposed strategy is devised to model and simulate Web search engines (WSE), which are complex and highly optimized systems operating over large clusters of processors. Web search engines manage dynamic and unpredictable user query bursts. Their performance evaluation is of paramount importance both for data center deployment and operation, and research tasks. A DEVS model of a WSE has been previously developed and evaluated with CD++.[7] This model was modified to be efficiently simulated in parallel by using PCD++[8] and different load balance algorithms were evaluated in Inostrosa-Psijas et al.[9]

In this article, we show that efficient parallel simulations of DEVS models can be attained using an approximate and semi-asynchronous approach. In particular, besides the Spectral Clustering[10] load balance algorithm presented and evaluated in Inostrosa-Psijas et al,[9] we introduce a strategy for modeling and simulating WSE based on (1) an approximate approach to reduce the running time and memory consumption of parallel simulations. No rollback mechanisms are used to correct time causality errors.[3] In particular, the simulation of WSE can easily trigger rollbacks because of causality related events (created by the same query) are propagated throughout several LPs. (2) Window barriers used to reduce the events simulated in nonchronological order (straggler events). A dynamic algorithm evaluates how frequently the time increment of the window barriers has to be computed to obtain a good trade-off between the running time of the simulations and the effectiveness of the estimated metrics. (3) A semi-asynchronous algorithm, which relaxes the time window barrier. This algorithm aims to reduce the number of straggler events and the number of global synchronization barriers executed by synchronous approximate simulation approaches.[3] To the best of our knowledge, this work is the first attempt to include an approximate simulation approach for WSE models into the PCD++[8] framework. Results show that our proposal is able to reduce both execution times and memory usage of standard optimistic simulations. The approximate simulation results introduce an error of 2%.

The remainder of this article is organized as follows. In Section 2, we present a brief introduction to parallel simulations and review related works. In Section 3, we introduce the PDEVS formalism. Section 4 describes the components of Web search engines. Section 5 presents our proposed parallel and approximate simulation algorithm. Section 6 details our experimental setup and evaluation. Conclusions follow in Section 7.

2 | RELATED WORK

Discrete-event simulation (DES) is a tool used to dynamically understand the behavior of complex discrete dynamic systems whose models may not be solved analytically. DES is a computing program that emulates the behavior of a physical system or a real system. In DES, a physical system is represented by a collection of entities that interact to each other to accomplish a global objective. States of the physical system are represented by a set of state variables that are used to describe the system state at any time. Changes of states in the physical system are represented by the execution of discrete time-stamped events. However, one of the main disadvantages of DES is the computational time required for simulating large-scale and highly complex systems.

Parallel discrete-event simulation is used to overcome the time consuming problem of DES. A PDES consists of executing single DES program on a parallel computer. The simulation program is normally decomposed into a set of concurrent processes called LP that can be executed independently on different processors. Each LP is composed of a separate set of state variables for the simulation program. Events take place within LPs, change the states of the LPs, and LPs may schedule the occurrence of new events in other LPs by sending event messages to them. Once the LPs are placed onto the processors, one is faced with the nontrivial problem of 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. Thus, the challenge in PDES is to execute LPs concurrently and lead to correct simulation results. To this end, synchronization protocols must be used to ensure that the parallel simulation of events is executed in correct time-stamp order (not violating the temporal causality constraint). Synchronization protocols can be classified as conservative or optimistic.[11]

Conservative protocols[1] strictly avoid the possibility of any causality error by using some strategy to determine if it is safe to process an event. Typically, the process is blocked until all execution conditions are satisfied (there is no event for which causal dependencies are still pending). It requires developers to enhance model definition with additional constructs such as lower bounds on the time the effects of events are propagated across the model.

In optimistic algorithms, like TW,[2] the occurrence of events is optimistically processed as soon as they are available at the LPs and then the LPs locally re-execute events whenever the simulation of earlier events is missed in one or more LPs. In other words, the simulation time may advance at differing time intervals in each LP. Whenever an LP receives a “straggler" message carrying an event *e* with time-stamp in the “past," the LP is “rolled back" into the state just before the occurrence of *e* and the simulation of the LP is resumed from this point onwards. That is, all the events with time-stamps later than *e*'s time-stamp are resimulated. The LPs save their states periodically to support rollbacks. In addition, any change propagated to other LPs as a result of processing erroneous events is corrected with a similar rollback method. To this end, special (anti-)messages are sent to the affected LPs to notify them of the previous sending of erroneous messages. During simulation, a global virtual time (GVT) is calculated periodically. Its value is such that no LP can be rolled-back earlier than it, and thereby the processed events and anti-messages with time-stamps less than GVT are discarded to release memory. global virtue time steadily advances forward regardless the amount of rollbacks. When the rate of rollbacks is kept small, performance becomes very efficient.

Parallel discrete-event simulation has been widely used in the past years to evaluate the performance of large scale systems.[12,13] In fact, there are various proposals to make parallel simulation more efficient.[14–16] One of them is the Lightweight Time Warp or LTW[17] strategy, it only saves external events (those sent among LPs), internal events are not saved into the state variables since they can be recreated from external events after a rollback occurrence. The efficiency of

parallel simulation has deserved recent attention in the context of clusters of multicore processors.[18,19]

Alternatively, the idea of ignoring rollbacks was first proposed by Rao et al,[16] where the introduced error was evaluated for small queuing network models parallelized using up to 3 LPs. They observed errors below 5%. Given the small number of LPs used in the experiments, the problem of controlling the correct execution of events was not relevant, as in this case, simulations are close to a sequential one in causality dependencies across LPs. Relaxation of event causality in LPs has been studied for systems in which events may occur in time intervals.[20] This is called temporal uncertainty, and represents systems where it is equally valid for these events to occur at any point in the respective intervals.

In a previous study,[14] the authors proposed a relaxation strategy based on the observation that 2 events must be executed according to their time-stamps if their out of order execution would produce a different result. This leads to explicitly identify which events concurring at the same LP object may be executed in any order, involving extra work for the user.

In the Adaptive Time Warp[21] the LPs with excessive rollback rate are suspended some time. The length of this interval is determined from statistics collected during execution, and its specific value intends to minimize the cost of either remain blocked or process the rollbacks. The authors in a previous study[22] indirectly control the optimistic execution by limiting the number of memory buffers assigned to keep uncommitted events. The LPs block themselves when they have no buffers available and wait for the next fossil collection. The fossil collection process frees the memory that stores events with time-stamps lower than the GVT. The Moving Time Window protocol[23] was proposed for synchronous simulations. The processing of events is driven by a time interval $[t, t + \Delta)$. Each processor is allowed to process the occurrence of events e with time-stamps $t \leqslant e.t < t + \Delta$. Other similar schemes are in previous studies.[24–27]

Recently, in Marin et al,[3] a new PDES approach called approximate parallel simulation was introduced. The implementation of the parallel simulator is based on multithreading and a bulk-synchronous message passing strategy to automatically conduct simulation time-advance. The simulation design is based on the Bulk Synchronous Parallel (BSP) computing model.[28] Under the BSP model, computation is organized as a sequence of supersteps. During a superstep, processors may perform computations on local data and/or send messages to other processors. At the end of a superstep, there is always a synchronization barrier. It permits that messages sent during the current superstep are available for processing at their destinations at the next superstep. The underlying communication library ensures that all messages will be available at their destinations before starting the next superstep. In each processor, there is one master thread that synchronizes with all other $P - 1$ master threads to execute the BSP superstep and to exchange messages. Then, in each processor and superstep the remaining $T - 1$ threads synchronize with the master thread to start the next superstep, though they may immediately exchange messages during the current superstep as they share the same processor main memory. We are interested in applying these parallel simulation methods with a formal modelling technique: the PDEVS[29] to simulate large-scale WSE.

3 | PDEVS OVERVIEW

The DEVS[4] is a modeling and simulation formalism of discrete-event dynamic systems. Discrete-Event System Specification provides the means for describing discrete-event systems by using 2 different kind of elements to model a real system: atomic and coupled models. Atomic models are the most elemental and basic entity to represent systems. Atomic models can react to internal and external events, which allows to define a way of specifying systems whose states change upon the reception of an input event or the expiration of a time delay.

Parallel Discrete-Event System Specification is a DEVS extension for supporting the simultaneous processing of events handled by an additional function. In PDEVS, an atomic model is formally defined according to the following structure[29]:

$$PDEVS = < X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \delta_{\text{con}}, \lambda, ta >,$$

where

X is the set of *external* events;
Y is the set of *output* events;
S is the set of *sequential* states;
$\delta_{\text{ext}} : Q \times X^b \rightarrow S$ is the *external* state transition function, where $Q := \{(s, e) \mid s \in S, \quad 0 \leqslant e \leqslant ta(s)\}$ and e is the elapsed time since the last state transition.
$\delta_{\text{int}} : S \rightarrow S$ is the *internal* state transition function;
$\delta_{\text{con}} : S \rightarrow X^b$ is the *confluent* transition function;
$\lambda : S \rightarrow Y^b$ is the *output* function;
$ta : S \rightarrow R_0^+ \quad \cup \quad \infty$ is the *time advance* function.

The semantics for this is as follows. At any given time, an atomic PDEVS model is in state $s \in S$ and, if no external events are received, it will remain in that state for a period of time as defined by $ta(s)$. The $ta(s)$ function can take any real value in the interval $[0, \infty]$. A state for which $ta(s) = 0$ is called a **transient state**. On the contrary, if $ta(s) = \infty$, then the system will stay in that state forever unless an external event is received. In such case, s is called a **passive state**. Transitions that occur due to the expiration of $ta(s)$ are called **internal transitions**. When an internal transition occurs, the system outputs the value $\lambda(s)$, and changes to a state defined by $\delta_{\text{int}}(s)$. A state transition can also take place when an external event occurs. In this case, the new state is defined by δ_{ext} according to the input value, the current state, and the elapsed time. Parallel Discrete-Event System Specification also includes the so-called confluent transition function, that allows the modeler to define the behavior of the system when collisions occur, that is, when a component receives external events at the same time that an internal transition is set to happen.[29] The confluent function takes a bag of inputs (X^b), which is a set of possible combinations of inputs occurring at the same time.

Coupled models are composed of 2 or more atomic or coupled models,[4] and can be regarded as another PDEVS model because of the closure property. Parallel Discrete-Event System Specification coupled models can be integrated to form a model hierarchy, allowing model reuse. A coupled model is defined as a structure of the form

$$DN =< X_{\text{self}}, Y_{\text{self}}, D, \quad \{M_i\}, \{I_i\}, \{Z_{ij}\} >,$$

where D is a set of components, and

for each $i \in D$,
M_i is a component with constraint that
$M_i =< X_i, Y_i, S_i, \delta_{iext}, \delta_{iint}, \delta_{icon}, \lambda_i, ta_i >$ is a PDEVS model;
for each $i \in D \cup \{self\}$,
I_i is the set of the influences of i;
for each $j \in I_i$,
$Z_{i,j}$ is a function, $i - to - j$ output-input translation.
I_i is a subset of $D \cup \{self\}$, i is not in I_i,
$Z_{self,j} : X_{self} \to X_j$
$Z_{i,self} : Y_i \to Y_{self} Z_{i,j} : Y_i \to X_j$

Coupled models may have their own input and output events, as defined by the X_{self} and Y_{self} sets. Upon the arrival of an external event, a coupled model has to redirect the input to one or more of its components. In addition, when a component produces an output, it has to be mapped as another component's input or as an output of the coupled model itself. Input and output mappings are defined by the Z function. Coupled models represent the structure of a system, whereas atomic models represent its behavior.

In traditional DEVS models, there can be ambiguity in coupled models when there is more than 1 component scheduled for an internal transition at the same time (collision). The first model to make its internal transition will produce an output that may be translated to an external event being received by another component model that is already scheduled for an internal transition at that time. But then, should this second model process the external transition first with $e = ta(s)$? or is it the internal transition that should be executed first and then the external transition with $e = 0$? The DEVS formalism solves this problem by using a function called select. With this function, only 1 model of the group of imminent models will be allowed to have $e = 0$. The other imminent models will be divided into 2 groups: those that do receive the external output from this model, and those that do not. The first group will execute their external transitions functions with $e = ta(s)$ and the second group will be among the group of imminent models for the next simulation cycle, which may require again the use of the select function to decide which model will execute first.[4,5] The select function is replaced in PDEVS[29] by the confluent transition function (δ_{con}) to solve the collision of events. The confluent function allows all imminent components to produce their outputs, which are sent through the corresponding mapping of input/output ports (using the coupling information) to the corresponding components.

Discrete-Event System Specification and its extension PDEVS, are independent from any simulation mechanism. They can be implemented with several simulation tools, tackling different needs, and providing advantages on specific scenarios. A noncomprehensive list includes

- DEVSJAVA[30] which allows hierarchical model definition and visualization.
- JDEVS[31] is a visual DEVS framework.
- SimStudio[32] which is a web-based framework.
- CD++[33] which supports simulation of DEVS and Cell-DEVS[34] models.
- Parallel CD++ (PCD++),[8] is the parallel/distributed version of CD++ supporting conservative and optimistic protocols.[35,36]

4 | WEB SEARCH ENGINES

Large-scale WSE like Bing, Google, or Yahoo! support heavy user demands and they must fulfill them within a reasonable time frame. To accomplish this, queries are processed as soon as they arrive to keep throughput at query arrival speed and to maintain query response time below a given upper bound. Web search engines must deal efficiently and effectively with sudden and drastic increases in query traffic from users, mostly related to natural disasters or social incidents that attract worldwide interest. Because of this, WSEs normally operate at a steady average hardware usage of around 30% to 40% to efficiently respond to increases in query traffic.[37]

Data centers for large WSEs contain thousands of processors arranged in high-communicating groups called services. In general, each service is devoted to a single specialized operation related to the processing of user queries arriving to the search engine. Services are software components executing operations such as (1) calculation of the top-k documents that best match a user query; (2) routing queries to the appropriate services and blending of results coming from them; (3) construction of the result Web page for queries; (4) advertising related to query terms; and (5) query suggestions, among many other operations. The realization of these services may involve the use of different algorithms and data structures devised to support efficient query processing. To deal with the complexity of WSEs, discrete-event simulation can be used as a suitable tool for assisting the development cycles associated with the algorithm engineering required to achieve an efficient and scalable query processing.[38,39]

Figure 1 shows an example of an organization of services for a WSE,[40,41] where typical services are the Front Service (FS), the Cache Service (CS), and the Index Service (IS). The FS comprises several replicated nodes. It handles and manages submitted user queries by routing them to the appropriate service (CS or IS), performs the blending of partial results returned from the IS, and it also manages the delivery of final top-k results to the user. The CS implements a partitioned distributed cache that stores previously computed results for the most popular queries (they are query results composed of document IDs). The IS holds an index built on top of the document collection (e.g. HTML documents from a big sample of the Web) and it is responsible of calculating the top-k document results (document IDs) that best match a query.

These services are deployed on clusters of computers and its processing nodes are allocated in racks connected via network switches. The CS and IS are partitioned and replicated in arrays of computer service nodes. Partitioning allows to host larger indexes and more caching capacity. Replications helps reducing query response times and increasing throughput of queries. It also helps supporting fault tolerance. Thus, the CS and IS are implemented as arrays of $P \times R$ processing nodes, where P is the level of partitioning and R is the replication level of each partition of a service.

4.1 | Query Processing

Figure 1 depicts the operations performed by a WSE to process a given query. Front service nodes independently receive user queries (step 1) and send back the top-k document results to the user. Once a query arrives to a FS node, a CS node is chosen to determine whether a

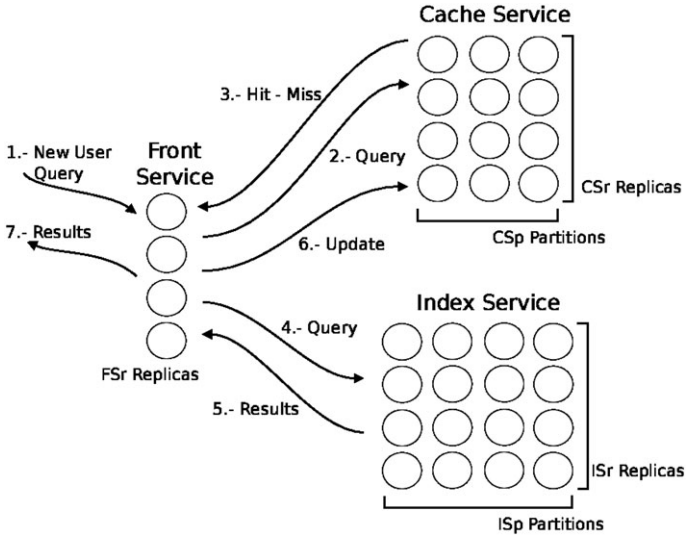


FIGURE 1 Query processing

query has been previously processed and its computed results are still cached (step 2). A simple Least Recently Used approach can be used. Additionally, the memory cache partition can be performed by means of a distributed memory object caching system named Memcached,[42] where a given query is always assigned to the same CS partition. Memcached makes use of a hash function with uniform distribution over the query terms to select the partition that should contain the precomputed document results for the query. Therefore, at any given time, different queries can be solved by different replicas of the same partition. Replicas are selected in a round-robin way.

If the query document results are cached, the CS node sends the top-*k* document IDs to the FS node (step 3), which sends the query results to users (step 7). If the query results are not cached, the CS node sends a hit-miss message to the FS. Then, the FS node sends an index search request to the IS cluster (step 4). Each query is sent to all of the *P* IS partitions and, in parallel, a random replica in each partition determines the local top-*k* document IDs.

The IS uses an index that allows the fast mapping among query terms and documents. Due to the amount of documents and the index size are huge, they are evenly distributed onto a large set of *P* processor. The index stored in each IS node is the so-called inverted index.[43] The inverted index is a data structure used by all well-known WSEs. It is composed of a vocabulary table (which contains the *V* distinct relevant terms found in the document collection) and a set of posting lists. The posting list for term $c \in V$ stores the identifiers of the documents that contain the term *c*, along with additional data used for ranking purposes. To solve a query, posting lists must be fetched for the query terms, then intersection among them is computed, and finally, the ranking of the resulting intersection set is performed by using algorithms like BM25 or WAND.[44] Hence, an inverted index allows for the very fast computation of the top-*k* relevant documents for a query, because of the precomputed data. The aim is to speed up query processing by first using the index to quickly find a reduced subset of documents that must be compared against the query, to then determine which of them have the potential of becoming part of the global top-*k* results.

Finally, the local top-*k* results computed by the IS are collected by the FS node (step 5) to determine the global top-*k* document IDs (by means of a merge operation) to update the CS (step 6) and to deliver the final results to the user (step 7).

4.2 | Modeling WSE with PCD++

In this section, we present a model of a WSE implemented with PCD++ devised to be executed in a distributed cluster of processors. Our PDEVS-based model of the WSE, as shown in Figure 2, was built using atomic models and by including the input/output ports inside the atomic models. Shaded areas of Figure 2 are intended only for the reader to easily identify the different services in the figure. The Query Generator atomic model generates user queries and delivers them to the FS by selecting its replicas in a round-robin fashion. Queries are sent to the FS through the corresponding output ports $(out_1, \ldots, out_n)$. The query inter-arrival time is simulated by using an exponential distribution.

Once an FS replica receives a new query (on its input port), it is immediately sent to a single replica of the CS through one of its $outCS_{ij}$ output ports. The FS replica chooses the *ith* partition by computing a hash function on the query terms, and the *jth* replica—of the selected CS partition—is chosen in a round-robin way. Then, the CS replica responds the query to the FS with a "hit/miss" message. In case there is a hit, the query now contains the top-*k* document results for the query, and the FS responds with those results to the user. However, if the answer has a miss, the query is sent to the *P* partitions of the IS through the $outIS_{ij}$ output ports of the FS. Index Service replicas are also selected in a round-robin way. Once the FS node has received the partial results from all of the selected IS replicas, it performs a merge operation to produce the top-*k* final document results.

5 | PCD++ SEMI-ASYNCHRONOUS APPROXIMATE SIMULATIONS

In this section, we present our simulation strategy for the PDEVS framework PCD++,[8] devised for simulating large scale Web search

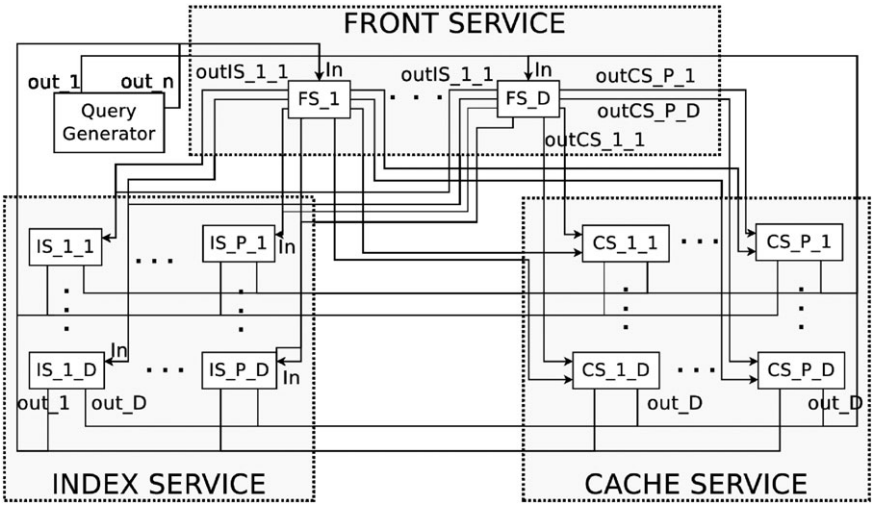


FIGURE 2 Parallel Discrete-Event System Specification model of a Web search engine

engines. It is based on (1) an approximate algorithm, which uses a time window barrier to prevent the occurrence of straggler events; (2) a dynamic algorithm used to adjust the time window increment according to the stable/unstable state of the simulation (we assume a simulation is stable when similar number of events are executed during different time intervals); (3) a semi-asynchronous simulation scheme; and (4) a load balance algorithm.

The proposed strategy has an optimistic approach. When an LP receives a straggler event, instead of using expensive recovery mechanisms such as rollbacks, it simply reset the local clock without executing additional operations. In other words, the local clock takes the time value of the straggler event. However, by not applying any mechanism to correct the temporal causality errors of events, imprecision can be introduced into the computed metrics.

5.1 | Approximate algorithm

In a previous work,[3] the approximate synchronous strategy does not implement mechanisms to satisfy causality constraints, instead it allows to process stragglers events. Thus, whenever an LP detects a straggler event, the local clock of the LP is set to the value of the time-stamp of the straggler event. As a consequence, the estimated metrics in the simulation are not identical to those of a sequential simulation. They are approximate. To control the simulation time-advance and the error introduced in the estimated metrics, the approximate synchronous strategy uses a time window barrier *B*. logical processes can only process events with time-stamp lower than the value of the time window *B*.

In Marin et al,[3] the time window *B* is updated with a time increment *W*. *W* is calculated every *n* steps or supersteps according to the computer model BSP.[28] The variable *n* is a user-defined variable. Each superstep ends with a barrier synchronization. In this case, the barrier synchronization of a superstep s_i corresponds to the window barrier B_i. Thus, each superstep is delimited by *B* and at the end of each superstep the time window barrier is updated to $B = B + W$.

To compute *W*, each processor stores tuples (t, s, d) for each message sent to another processor, where *t* is the time-stamp of the event, *s* is the source processor and *d* is the destiny processor. At the end of *n* supersteps, all processors send their tuples to a master processor, for example P_0. Afterwards, P_0 calculates the minimum number of supersteps *S* required to process tuples (t, s, d). Then, if Δ is the simulation time elapsed since the first and the last tuples were processed, the time increment is computed as $W = \Delta/S$. Then, the new *W* value is distributed among the processors running the parallel simulation.

5.2 | Dynamic time window algorithm

When modelling a WSE, the number of events simulated in each LP can vary abruptly in different time intervals. This is mainly because the simulation is driven by traces containing real user queries. In other words, the simulations reflect the dynamic and unpredictable behavior of users in a WSE. In this work, we propose to take advantage of this unpredictable behavior to improve the efficiency and the accuracy of the approximate simulation. Eg, when the simulation is in a stable state (a similar number of events is executed during each time window *B*), we can reduce the number of updates of *W* and therefore reduce the communication and computation costs of the simulation. On the other hand, if the simulation is in an unstable state (the number of events executed during each time window *B* changes drastically) we need to update *W* more frequently to reduce the error of the estimated metrics.

Therefore, to improve the performance of parallel simulations and control the number of stragglers events more efficiently, in this work we propose to adaptively adjust the number of supersteps (*n*) elapsed between 2 consecutive updates of *W*. To this end, we compute the moving average[45] of previously computed *W* values to detect burst of user queries represented in the traces. A sequence of time increments $W_1, W_2, \ldots, W_N$ of size *N*, where W_i is the time increment computed in the superstep *i*, is used to compute the moving average (MA_w) of size *w*, where $N > w$. The standard deviation σ on the moving average is used to compute upper bounds and lower bounds, which define an interval $[MA_w - \sigma, MA_w + \sigma]$ used to detect unstable states of simulations.

If the new value of *W* is outside of the interval, the simulation is considered to be in an unstable state and we update *W* more frequently (decrease the value of *n*). Otherwise, the simulation is considered to be in a stable state and we update *W* less frequently (increase the value of *n*). To avoid $n = 0$ and $n = \infty$ we set 2 thresholds. The threshold λ_{LOW}

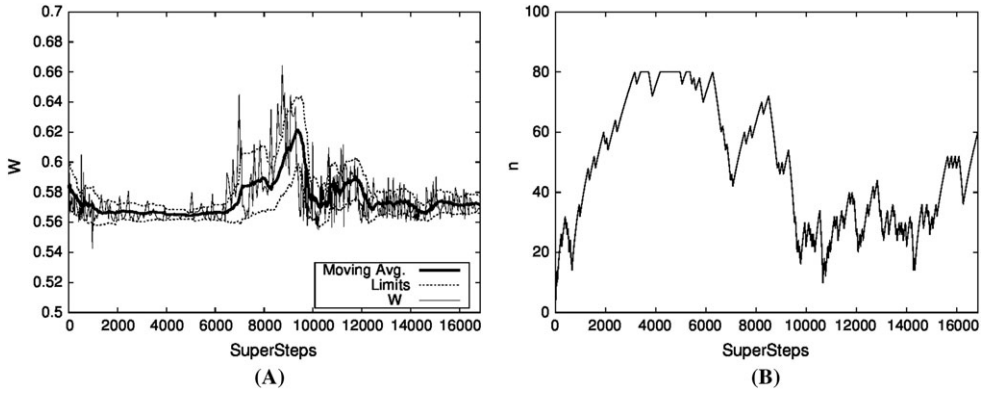


FIGURE 3 Dynamic algorithm; A, update of *W* and B, frequency (supersteps - *n*) of updating *W*

is used to prevent that the time required to calculate *W* is greater than 30% of the runtime of the simulation. The threshold λ_{UP} is adjusted so that the time required to calculate *W* is 5% of the runtime of the parallel simulation.

Figure 3 shows the values of *W* and *n* obtained when simulating a WSE with a total of 528 service nodes. The service configuration used in this experiment is 32 FS, 32 CS partitions and 8 CS replicas, 16 IS partitions with 15 IS replicas. We simulate different query traffic rates. At the beginning of the simulation the query traffic is stable. After the superstep 6000, the query traffic increases emulating a peak of user queries. Finally, after the superstep 10 000 the query traffic is kept stable again.

In Figure 3A the *x*−axis shows the simulation time-advance and the *y*−axis shows how *W* is automatically adjusted every *n* supersteps. This figure shows the values of the moving average, the computed value of *W*, and the limits of the interval. At the beginning of the simulation there is a high variability in the values of *W*. Then, between supersteps 2500 and 5000 the simulation tends to calculate similar values of *W*, which indicates that the simulation is in a stable state. Later, the simulation re-enters in an unstable state.

In Figure 3B, the *y*-axis shows how the value of *n* is automatically adjusted as the simulation time-advance and the new value of *W* is computed. Between supersteps 2500 and 5000, the value of *n* tends to grow until it reaches the maximum value bounded by λ_{SUP}, where it remains constant for a brief interval. When the value of *W* becomes unstable, the value of *n* tends to decrease to update *W* more frequently. Thus, these figures show that the proposed dynamic algorithm can react to different query traffic represented by the simulated traces.

5.3 | Synchronization schemes

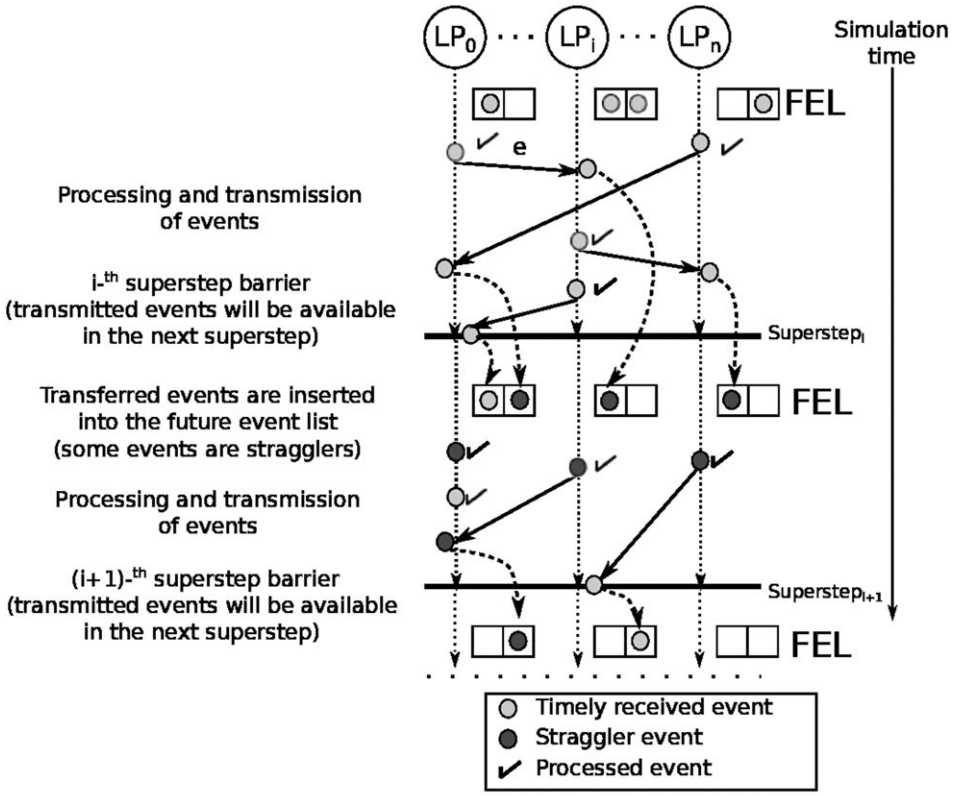
In PDES, synchronization mechanisms are used to ensure the correct execution of events[2] and/or to reduce the number of straggler events.[3] In particular, Figure 4A describes the approximate synchronous simulation strategy presented in Marin et al,[3] which is based in the BSP model.[28] Assuming that each LP runs on a different processor, in a superstep *i* if the emitting LP and the receiving LP are the same, the event is executed in the current superstep *i*, if the time-stamp of the event is lower than the time window ($e.tr < B$). However, if the emitting LP and the receiving LP are allocated in different processors the event will be processed after the barrier synchronization. That is, in the next superstep $i+1$. For example, the LP_0 sends an event *e* to LP_i in the superstep *i*. The time-stamp of the event *e* is lower than the value of the time barrier B_i. But, due to the restriction imposed by the synchronous BSP model, the event *e* is received in the next superstep when the value of the time window has been increased to $B_{i+1} = B_i + W$. Therefore, the event *e* becomes a straggler event.

The synchronous strategy[3] tends to generate stragglers events that can be avoided in a semi-asynchronous scheme as proposed in this work. Our algorithm includes *I* iterations before reaching the time barrier *B*. During those *I* iterations, LPs can send and receive events from/to others LPs. Thus, inside each superstep ended with a time window barrier, we execute *I* iterations or loops, which allow to receive events from others LPs in the same superstep they are sent.

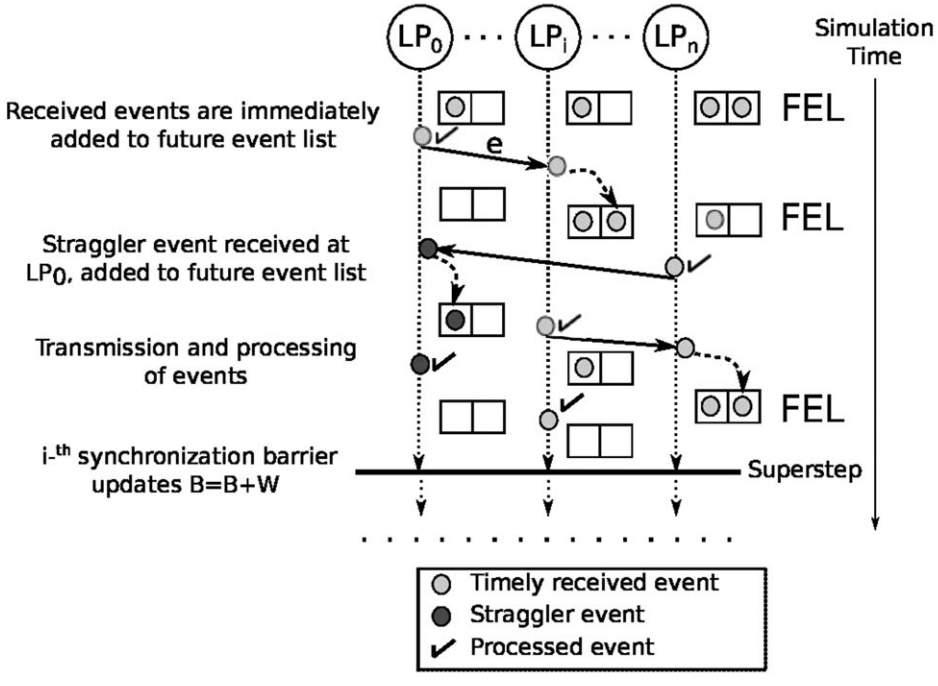
Figure 4B shows an example of the proposed strategy. Each LP executes local events, sends, and receives events. The LP_0 sends an event to LP_i, and the event is received and processed in the same superstep *i* before the time window barrier B_i. Thus, the event is not considered a straggler event.

The number of iterations *I* is automatically adjusted in each processor according to $I = \frac{\text{number of events committed at superstep } s}{\text{increment in oracle superstep during the current interval}}$.[46,47] The oracle simulation is effected by executing in each LP an asynchronous superstep counter code, which is driven by event messages carrying superstep counts from other LPs. To this end, we maintain one superstep counter $C[i]$ for each LP_i and each event message *e* carries an integer $e.s$ indicating the minimum superstep at which this event may take place in the parallel simulation being synchronized by the oracle simulation (not the actual optimistic simulation). For each message *e* that is received at a given LP_i we execute the code: if $e.s > C[i]$ then $C[i] := e.s$, and each new event e^* scheduled by the LP_i in another LP_j carries the superstep count $e^*.s = C[i] + 1$ if *j* is located in other processor or $e^*.s = C[i]$ if *j* is a co-resident LP.

Therefore, our proposed approximate algorithm developed in PCD++ relaxes the time window barriers by including *I* iterations inside the supersteps. This modification turned PCD++ into a semi-asynchronous parallel simulator, in which events sent among LPs are pushed into the destination's event list as soon as they arrive. Thus, events are immediately available for processing reducing the probability of straggler events.



(A) Approximate and synchronous parallel simulation.



(B) Semi-Asynchronous parallel simulation.

FIGURE 4 Synchronization schemes

5.4 | Load balance algorithm

In PCD++, the allocation of atomic models to the different LPs is defined by the user in an offline fashion by explicitly specifying the processor location for each model component. This requires knowledge about the communication and computation patterns of the simulated WSE. Moreover, entity migration among LPs is not allowed and PCD++ lacks of dynamic load balance capabilities.

In the WSE modeled with PDEVS, each service node is associated with an atomic model. The atomic models communicate to each other to simulate the query processing process. To reduce the

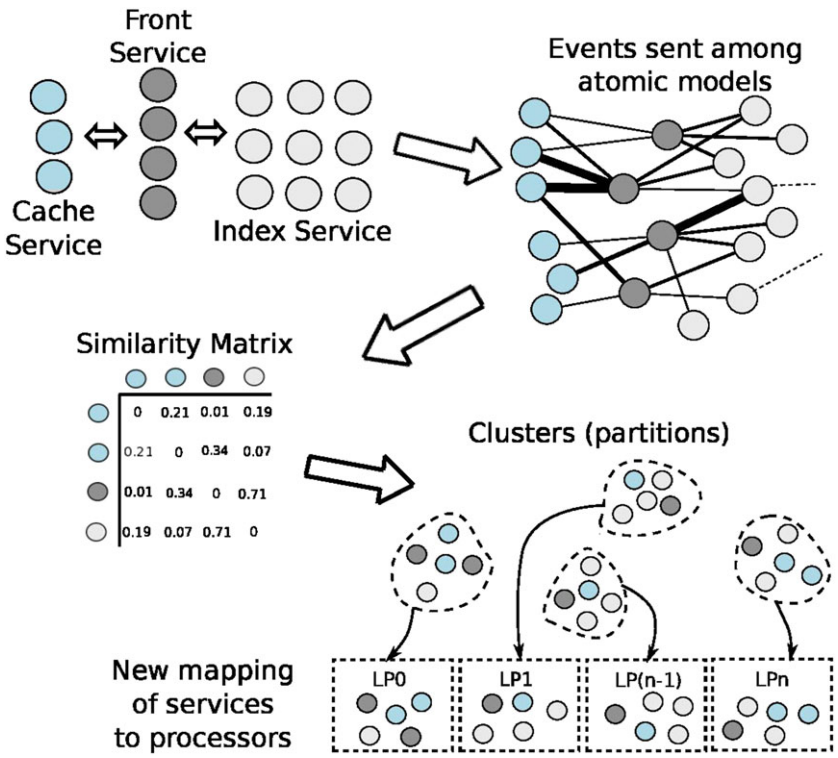


FIGURE 5 Load balance with the Spectral Clustering algorithm

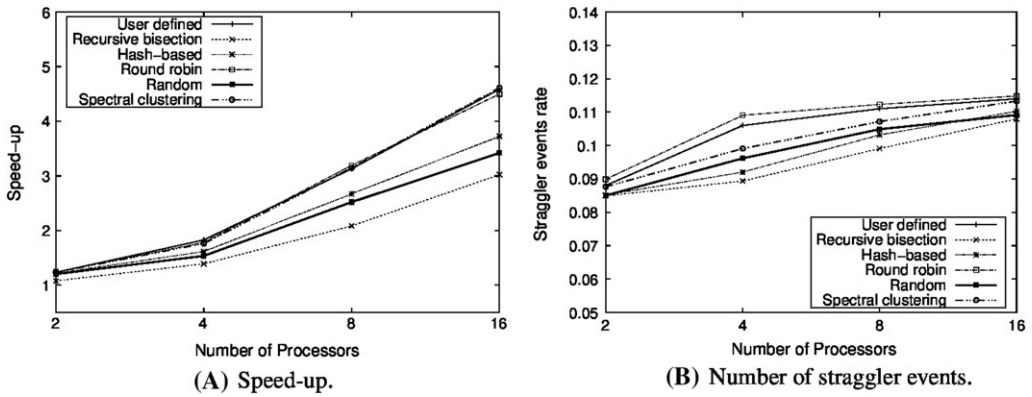


FIGURE 6 Evaluation of partitioning strategies when simulating a WSE with 1040 service nodes

communication costs between atomic models and to balance the workload among processors, we use the Spectral Clustering[10] algorithm.

The Spectral Clustering algorithm obtains *k* partitions of the WSE model, where *k* is the number of LPs. For this purpose, a similarity matrix based on the multiplicative inverse of all data transmitted between the different atomic models is constructed. Clusters are built by using the list of cluster algorithm and centers are selected with the heuristic that maximizes the sum of distances to the previous selected centers.[48] Thus, we obtain groups with equal numbers of atomic models.

Figure 5 describes the Spectral Clustering algorithm. Two atomic models are considered close if they have a high degree of communication. This is represented by a symmetric similarity matrix. Then, the Laplacian matrix is obtained and the list of cluster algorithm[48] is applied to obtain the set of *k* clusters whose centers tend to be distant from each other. Each cluster corresponds to a partition of the model that is assigned to a processor. Thus, each partition contains the atomic models that communicate the most.

In a previous work,[9] we proposed and evaluated different strategies for partitioning the WSE simulation model. In this work, we evaluate the proposed strategies with a larger WSE configuration. Figure 6 shows the speed-up and the straggler events rate reported by the following strategies (1) the user defined (oracle), which requires knowledge about the application and the communication patterns to properly allocate entities evenly at the different LPs; (2) recursive bisection implemented with the METIS software*; (3) hash-based which uses the Fowler-Noll-Vo non-cryptographic hash function; (4) Round robin; (5) random; and (6) the Spectral Clustering described above. In this experiment we simulated a WSE with 1040 service nodes.

* http://glaros.dtc.umn.edu/gkhome/views/metis

Figure 6A shows that the results obtained by the user defined (Oracle), the round-robin, and the Spectral Clustering strategies almost overlap to each other. These strategies present the highest speed-up as the number of processors increases. Regarding the straggler events rate, in Figure 6B we show that the round-robin and the user defined strategies present more straggler events than the reported by the Spectral Clustering algorithm. Therefore, as the Spectral Clustering algorithm uses information about the communication pattern among atomic models, it is capable of obtaining competitive performance and a low number of straggler events (0.114% for $P = 16$).

6 | EXPERIMENTS

In this section, we evaluate the performance and the accuracy of the proposed approximate and semi-asynchronous simulation strategy, which uses the Spectral Clustering partitioning algorithm. Experiments were executed using a PDEVS model of a WSE (see Figure 2) with 2 different services configurations, which is a user defined simulation parameter indicating partitioning and replication levels of each service of the WSE model (ie, <3, 4, 5, 6, 7> specifies a WSE with 3 FS, 4 CS partitions and 5 replicas per partition, and 6 IS partitions with 7 replicas each). In the following, we present experiments with configuration $A = <32, 32, 4, 16, 30>$ and configuration $B = <16, 16, 32, 16, 32>$ simulating 640 and 1040 physical processors, respectively.

To correctly simulate the behavior of a WSE and its relevant costs, the simulator uses actual query logs. In particular, we use a log of 36 389 567 queries submitted to the America Online Search service. It was pre-processed according to the rules described in Gan and Suel.[41] The costs of the most significant operations of a WSE were measured on production hardware[49] and then used by the PDEVS model to properly simulate the query processing process.

Experiments were executed on the Deepthought cluster at the ARS Laboratory at Carleton University using up to 16 HP PROLIANT DL Servers with Dual 3.2 GHz Intel Xeon processors and 2 GB of RAM memory. Our proposal was implemented with PCD++ and OpenMPI as the communication library.

6.1 | Simulation strategies

To evaluate the performance and the accuracy of our proposal, we perform experiments with the following strategies

- LTW: Lightweight Time Warp.[17]
- Fixed Window (PCD++ FW): Approximate semi-asynchronous strategy using a time window increment W of fixed size. This is an oracle algorithm, in which the size of W is precomputed. To this end, we execute our proposed strategy using the dynamic window algorithm and we obtain the average of all the W calculated during the simulation. Therefore, this strategy removes the communication and computation overhead introduced by the calculation of W.
- Proposal (PCD++ DW): Approximate semi-asynchronous strategy proposed in this work using the dynamic algorithm to compute the values of W.
- Asynchronous[50] (PCD++ Async): Each processor computes its local time window barrier B locally. Thus, there is no master processor collecting statistics to compute the time increment W. The fully asynchronous algorithm is implemented on top of the PCD++ framework. This strategy aims to improve the running time of the approximate parallel simulations, as a counterpart the accuracy of the algorithms tends to decrease.
- Synchronous (CPN-DW): The approximate simulation strategy as presented in Costa et al.[49] This strategy uses a synchronous mechanism to control the simulation time-advance and Coloured Petri Net to model the service nodes of a WSE. The value of W is computed every n fixed supersteps.

6.2 | Performance evaluation

In this section, we evaluate the performance of our proposed approximate and semi-asynchronous simulation strategy. To this end, in the following figures we show the running times reported by different simulation approaches. For a better representation, in Figure 7 we compare our proposal with the LTW algorithm and in Figure 8 we show results for different synchronization schemes.

Figure 7 shows the running time reported by different simulation strategies when increasing the total number of queries (from 100 to 100 000 queries). Experiments where performed with $P = 16$

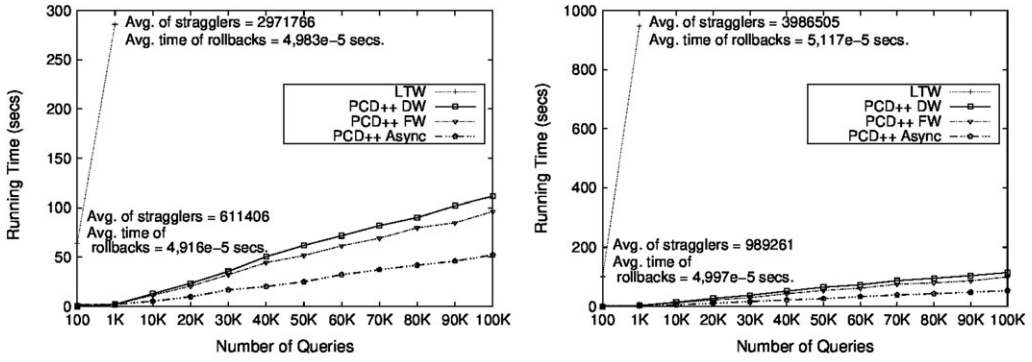


FIGURE 7 Running times obtained for the distributed approximate strategies implemented on top of PCD++ and the LTW strategy with 16 processors. LTW indicates Lightweight Time Warp; DW, dynamic window; FW, fixed window

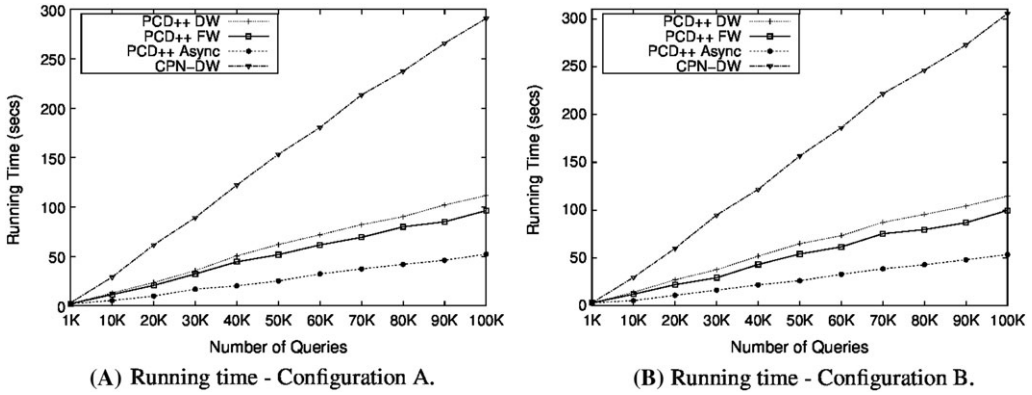


FIGURE 8 Running time reported by the approximate strategies implemented on top of PCD++ and the CPN-DW strategy. The simulations were executed with 16 processors. DW, indicates dynamic window; FW, fixed window

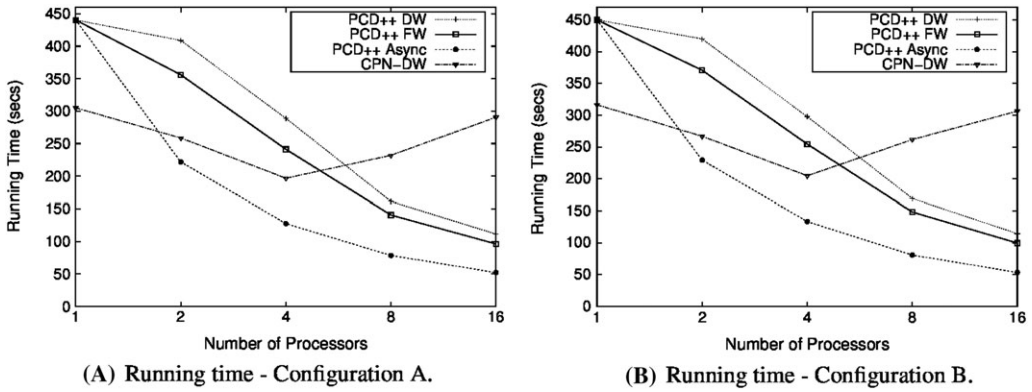


FIGURE 9 Running time reported by all approximate strategies with different number of processors. DW, indicates dynamic window; FW, fixed window

processors for configurations A and B. The LTW algorithm was only executed using 100 and 1000 queries. When simulating more queries, the executions took too long compared to the approximate strategies as observed in Figure 7. The behavior of the LTW is due to the inherent communication pattern of the WSE simulation model, in which each simulated service node of the FS can potentially communicate to every node of the other services and vice-versa. The outcome is a large amount of events sent among LPs, which tends to increase the probability of stragglers and its correspondent rollbacks. For 100 queries and configuration A, the LTW reports 2 971 766 straggler events and the execution of each rollback requires in average 4.983e-5 seconds. The average number of straggler events reported with configuration B is 3 986 505, and the average time required to execute each rollback is 5.117e-5 seconds. Therefore, the LTW algorithm is not considered in the following experiments.

All PCD++ approximate strategies perform well in running times when compared with the LTW. The PCD++ FW strategy outperforms our PCD++ DW strategy by 16% and 17% for configurations A and B, respectively. It is worth to remember that our dynamic strategy stores and transmits tuples to compute the value of W, which incurs into more communication than the PCD++ FW strategy. Thus, the overhead introduced by the dynamic algorithm used to compute W periodically is 17% at most.

Additionally, Figure 7 shows that the PCD++ Async approach reports running times 43% and 46% lower than the synchronous approaches for configuration A and B, respectively. This is mainly because the Async strategy computes W locally in each LP, so there is no communication overhead neither global barriers.

Figure 8 shows the running time obtained by the different strategies implemented with PCD++ and the synchronous CPN-DW strategy, as the number of queries increases. This experiment was performed with $P = 16$ processors. Both Figure 8A,B corresponding to experiments performed with configuration A and B, respectively, show that the synchronous CPN approach presents running times almost 3 times higher than the reported by the proposed semi-asynchronous strategy, say the PCD++ DW strategy. This is achieved mainly by applying the mechanism that dynamically adjusts n, the number of supersteps that must elapse before calculating a new value of W, and by applying a semi-asynchronous mechanism, which allows to process events in the same superstep they are send. As expected, the oracle PCD++ FW strategy presents running times lower than our proposal. However, again the algorithm reporting the lower running time is the PCD++ Async.

Figure 9 shows running times for the simulations of a total of 100 000 queries with different number of processors. We show that the PCD++ Async strategy presents the lowest running time. On the other hand, the PCD++ DW presents running times larger than its synchronous CPN counterpart when simulations are executed with few processors. However, when we use more processors ($P = 8$ and $P = 16$), our proposed simulation strategy tends to present better performance. This is because in the CPN-DW strategy, events sent in a superstep s are

TABLE 1 Memory consumption (MB) reported by the LTW and the approximate PCD++ strategies running on 16 processors

	Configuration A				Configuration B			
Queries	LTW	DW	FW	Async	LTW	DW	FW	Async
100	7.4	5.8	6.8	5.6	13	11	13	12
1000	25.9	18	16	17	34	24	22	21
10000	–	141	129	74	–	147	133	83
20000	–	279	243	126	–	284	249	134
30000	–	329	290	178	–	344	302	189
40000	–	513	463	220	–	524	475	231

Abbreviations: LTW indicates Lightweight Time Warp; DW, dynamic window; FW, fixed window

received in the next superstep $s + 1$. Then, the CPN-DW strategy tends to increase the total number of supersteps required to finish the simulation. Our experiments reported that the CPN-DW executes in average 33% more supersteps (barrier synchronizations) than the PCD++ DW strategy.

Table 1 shows the memory consumption reported by the LTW, the PCD++ Async and the approximate strategies with fixed (FW) and dynamic (DW) window approaches. The values reported by the FW and DW under both the proposed semi-asynchronous and the CPN synchronous approaches were very similar. In this table, we show that the different simulation strategies present similar memory consumption for small amounts of queries. The results reported by the LTW strategy to process 100 and 1000 queries, show that it does not drastically consume more memory than the approximate simulations. This is because of its state saving strategy, which only saves state variables and external events, reducing the total amount of events to be saved during the simulation. Nevertheless, executions with more queries are needed for a deeper conclusion in regard to the LTW strategy. However, as shown in previous figures, due to the running time reported by the LTW is clearly larger than the running time reported by the approximate approaches, we do not address this issue in this work.

Results obtained with a larger amount of queries (10 000–40 000) show that the DW approach consumes more memory than the FW approach. This is basically because the FW does not keep in memory the tuples required to compute the time increment *W*. Results show that the FW approach reduces by 9% in average the memory used by the DW strategy. In addition, the DW and the FW approaches report more memory consumption than the PCD++ Async strategy due to the time barrier *B*. An event not processed in a superstep is going to be stored in memory until its time-stamp is lower than *B*. In particular, with configuration A and for 40 000 queries, the PCD++ Async strategy presents memory consumption 2.33 times lower than the values reported by the DW approach and 2.1 times lower than the values reported by the FW approach. The results obtained with configuration B are very similar. The PCD++ Async strategy is 2.27 times lower than the DW approach and 2.05 times lower than the FW approach.

Finally, Figure 10 shows the speed-up reported by the different strategies with $P = 2, 4, 8$ and 16 processors for configuration A (top) and configuration B (bottom). In general, the gain in scalability is not very high, especially when comparing the PCD++ FW and the PCD++ DW strategies. This is because of the time window barrier used to control the simulation time-advance, the messages used to calculate the

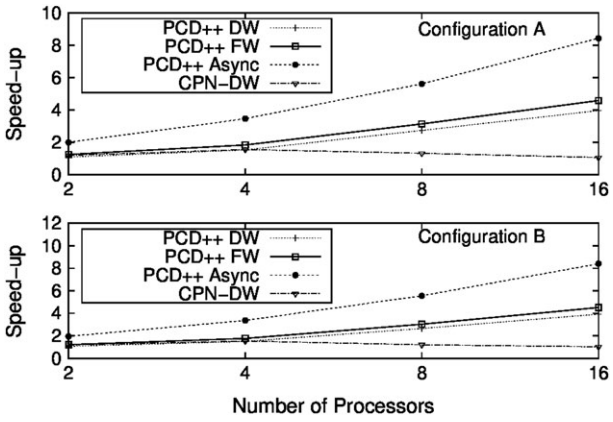


FIGURE 10 Speed-up achieved by the different parallel simulation strategies. DW, indicates dynamic window; FW, fixed window

value of *W* and then propagation of the updated value to all processors, which increases the communication cost. This graphic shows that the performance of the CPN-DW strategy tends to decrease as more processors are used. As explained before, this behavior is mainly because the communication constraint imposed by the BSP model. The Async approach, has the best performance in speed-up, because it does not use a global time barrier, which main propose is to reduce the rate of straggler events. However, the lack of this barrier clearly affect scalability as shown in the next section.

6.3 | Accuracy evaluation

In this section, we evaluate the effectiveness of our proposed approximate and semi-asynchronous simulation strategy. To this end, we study the straggler events rate, the Pearson correlation, and the root mean square error of the deviation, which is a measure of the differences between values obtained by the sequential simulation and the values reported by the parallel simulations. It is defined as $\varepsilon_m = \sqrt{(\sum (x_i - \bar{x})^2/(n \cdot (n-1))}$, and we calculate the relative error (e_r) as $\varepsilon_m/\bar{x}$.

In Figure 11, we show the straggler events rate reported by different simulation strategies. The PCD++ Async strategy reports the highest rate from 64.1% to 67.2% for both configurations A (Figure 11A) and B (Figure 11B). As mentioned above, this is because the PCD++ Async strategy is designed to improve the performance of simulations, rather than focusing on maintaining a low rate of straggler events. The mechanism used to control the time-advance in each LP is calculated using

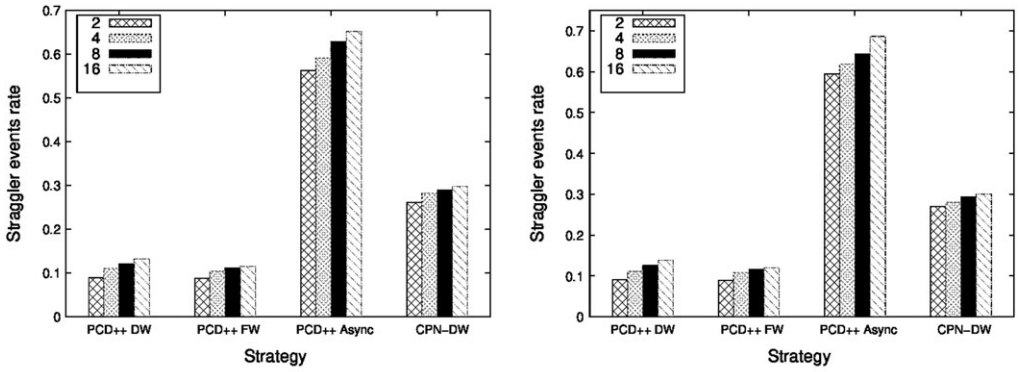


FIGURE 11 Straggler events rate reported by all approximate simulation strategies implemented with PCD++ and CPN. DW, indicates dynamic window; FW, fixed window

TABLE 2 Pearson correlation and relative error achieved to estimate the throughput of a WSE

	Configuration A		Configuration B	
	Pearson	Relative Error	Pearson	Relative Error
PCD++DW	**0.8748**	**0.0101**	**0.8564**	**0.0124**
PCD++FW	**0.9373**	**0.0082**	**0.9101**	**0.0088**
PCD++ Async	0.8105	0.1954	0.8023	0.2372
CPN-DW	0.8397	0.0202	0.8168	0.0214

Abbreviations: WSE indicates Web search engine; DW, dynamic window; FW, fixed window

local information. Therefore, there is no global view of the simulation time-advance.

The CPN-DW presents lower straggler rate than the PCD++ Async strategy. The number of stragglers events reported by CPN-DW reaches 29% and 29.3% with configurations A and B, respectively. This is mainly due to the parallel BSP model,[28] in which the events sent in a time window B_i are available to be processed in the next time window B_{i+1}. Thus, the local clock of the receiving LP may have advanced to a time higher than the time-stamp of the received events.

Regarding the proposed PCD++ DW strategy, the straggler events rate is much lower. This strategy reports between 12.1% and 13.1% straggler events rate for configurations A and B, respectively. This is mainly due to the proposed strategy uses I iterations in each superstep. Within each iteration it is possible to receive and send messages from and to other processors, which allows to process events in the same superstep they are sent. As expected, the oracle strategy (PCD++ FW) reports a number of straggler events rate slightly lower, which are 9.8% and 11.4% for configurations A and B, respectively.

In the following, we evaluate the accuracy of our proposal for estimating the throughput and the query response time metrics. We show results obtained when running the simulations with $P = 16$. Table 2 shows the Pearson correlation coefficients and the relative errors for the throughput metric. For both service configurations (A and B), the Pearson coefficients achieved by all strategies are close to 1 (greater than 80%), indicating a positive linear correlation between the estimated results obtained by the parallel approximate simulations and the values reported by the sequential simulation.

The highest Pearson correlations are achieved by the PCD++FW with values of .9373 and .9101 for configurations A and B, respectively, followed by our proposal, which is a more realistic strategy, as no previous simulation executions are required to adjust the value of W. On the other hand, the PCD++ Async strategy reports the lowest values of .8105 and .8023 for configuration A and B. The CPN-DW achieves competitive values.

In concordance with previous results, the PCD++ Async strategy reports the highest relative errors. More precisely, the PCD++ Async strategy obtains an error of 19.54% for configuration A and 23.72% for configuration B. The remaining simulation strategies present lower error values and similar to each other.

Table 3 shows the Pearson correlation and the relative error for the query response time. Again, the Pearson correlation tends to be close to 1, meaning a high correlation between the estimated values and the exact query response time values achieved by a sequential simulation. In particular, the oracle PCD++ FW strategy reports a Pearson correlation of .9223 and .9065 for the configurations A and B, respectively. The lowest correlations are reported by the PCD++ Async strategy with values of .8098 for configuration A and .7974 for configuration B. Our proposal, the PCD++ DW strategy, reports a Pearson correlation of .8664 and .8452. Finally, the synchronous CPN-DW strategy reports values of .8312 and .8093 for configurations A and B, respectively.

The relative error for the query response time metric follows the same tendency as showed for the throughput metric. The PCD++ Async strategy has the highest relative errors of 20.79% and 24.81% for configurations A and B, respectively. Other approximate strategies present similar relative errors which are below 2.7%.

TABLE 3 Pearson correlation and relative error achieved to estimate the query response time of a WSE

	Configuration A		Configuration B	
	Pearson	Relative Error	Pearson	Relative Error
PCD++ DW	**0.8664**	**0.0153**	**0.8452**	**0.0176**
PCD++ FW	**0.9223**	**0.0119**	**0.9065**	**0.0126**
PCD++ Async	0.8098	0.2079	0.7974	0.2481
CPN-DW	0.8312	0.0211	0.8093	0.0263

Abbreviations: WSE indicates Web search engine; DW, dynamic window; FW, fixed window

7 | CONCLUSIONS

In this work we presented an approximate and semi-asynchronous simulation strategy for the PCD++ platform. Our proposal is devised to model large WSE systems, which are composed by different services that interact to each other to solve a query. Web search engine presents modeling and simulation challenges such as drastic changes in user behavior and collaboration of different services to get the top-*k* document results. To the best of our knowledge, this is the first work introducing an approximate simulation strategy into the PCD++ framework for WSE modeled in PDEVS.

Our proposal is based on four elements (1) an approximate algorithm which aims to improve the running time of the simulations and uses a window barrier to control the simulation time advance, (2) a dynamic algorithm used to calculate the elapsed time of two consecutive window increment updates, (3) a semi-asynchronous scheme which aims to reduce the number of straggler events, and (4) a load balance algorithm used to automatically allocate the LPs into processors. Results show that our proposed strategy outperforms the optimistic LTW approach. We also are capable of reducing the number of straggler events in comparison to a synchronous and an asynchronous approximate simulation strategies.

ACKNOWLEDGMENTS

The autors would like to thank to Basal funds FB0001, Conicyt, Chile; Veronica Gil-Costa also thanks to PICT-2014-1146 and Basal project USA1555 USACH-MECESUP; and Gabriel Wainer thanks to NSERC, Canada and CFI, the Canadian Foundation for Innovation.

REFERENCES

1. Boukerche A, Das SK. Dynamic load balancing strategies for conservative parallel simulations. *SIGSIM Simul Dig*. 1997;27(1):20-28.
2. Fujimoto RM. Parallel discrete event simulation. *Commun ACM*. 1990;33(10):30-53.
3. Marin M, Gil-Costa V, Bonacic C, Solar R. Approximate parallel simulation of web search engines. *Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS'13*. ACM, New York, NY, USA; 2013:189-200.
4. Zeigler BP, Kim TG, Praehofer H. *Theory of Modeling and Simulation*. Orlando, FL, USA: Academic Press Inc.; 2000.
5. Wainer G. *Discrete-Event Modeling and Simulation: A Practitioner Approach*. Boca Raton, FL, USA: CRC Press. Taylor and Francis; 2009.
6. Wainer G. Experiences with devs modeling and simulation. *J Model Simul (Acta Press)*. 2001;21(2):138-147.
7. Inostrosa-Psijas A, Wainer GA, Costa VG, Marín M. Devs modeling of large scale web search engines. *Proceedings of the 2014 Winter Simulation Conference*, Savannah, GA, USA; December 7-10, 2014:3060-3071.
8. Liu Q, Wainer G. Parallel environment for devs and cell-devs models. *Simulation* 2007;6(83):449-471.
9. Inostrosa-Psijas A, Costa VG, Solar R, Marín M. Load balance strategies for DEVS approximated parallel and distributed discrete-event simulations. *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015*, Turku, Finland; March 4-6, 2015:337-340.
10. Ng AY, Jordan MI, Weiss Y. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*. Vancouver, British Columbia, Canada; December 3-8, 2001:849-856.
11. Fujimoto RM. *Parallel and Distribution Simulation Systems*. 1st ed. New York, NY, USA: John Wiley & Sons, Inc.; 1999.
12. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exper*. 2011;41(1):23-50.
13. Kunkel J. Using Simulation to Validate Performance of MPI(-IO) Implementations. *Supercomputing*; 2013:181-195.
14. Quaglia F, Baldoni R. Exploiting intra-object dependencies in parallel simulation. *Inf Process Lett*. 1999;70(3):119-125.
15. Quaglia F, Beraldi R. Space uncertain simulation events: Some concepts and an application to optimistic synchronization. *Proceedings of the Eighteenth Workshop on Parallel and Distributed Simulation*, Kufstein, Austria; 2004:181-188.
16. Rao DM, Thondugulam NV, Radhakrishnan R, Wilsey PA. Unsynchronized parallel discrete event simulation. *Proceedings of the 30th Conference on Winter Simulation*,Washington, DC, USA; 1998:1563-1570.
17. Liu Q, Wainer GA. Lightweight time warp a novel protocol for parallel optimistic simulation of large-scale devs and cell-devs models. *IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, 2008:131-138.
18. Liu J, Rong R. Hierarchical composite synchronization. *PADS, 2012*, Zhangjiajie, China; July 15-19, 2012:15-19.
19. Jafer S, Wainer GA. A performance evaluation of the conservative devs protocol in parallel simulation of devs-based models. *SpringSim (TMS-DEVS)*, Boston, MA, USA; 2011:103-110.
20. Fujimoto RM. Exploiting temporal uncertainty in parallel and distributed simulations. *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*, Atlanta, GA, USA; 1999:46-53.
21. Ball D, Hoyt S. The adaptive timewarp concurrency control algorithm. *SCS Multiconference Distrib Simul*. 1990;22(1):174-177.
22. Panesar K, Fujimoto R. Adaptive flow control in time warp. *11th Workshop on Parallel and Distributed Simulation (PADS'97)*, Austria; 1997:108-115.
23. Sokol L, Briscoe D, Wieland A. MTW: A strategy for scheduling discrete simulation events for concurrent execution. In *SCS Multiconference on Distributed Simulation 19 3*, San Diego, California, USA; 1988:34-42.
24. Lubachevsky B, Weiss A. An analysis of rollback-based simulation. *ACM Trans Model Comput Simul*. 1991;1(2):154-193.
25. Turner S, Xu M. Performance evaluation of the Bounded Time Warp algorithm. *6th Workshop on Parallel and Distributed Simulation (PADS'92)*, Newport Beach, California; 1992:117-126.
26. Steinman J. SPEEDES: a multiple-synchronization environment for parallel discrete event simulation. *Int J Comput Simul*. 1992;2(3):251-286.
27. Steinman J. Discrete-event simulation and the event-horizon. *8th Workshop on Parallel and Distributed Simulation (PADS'94)*, Edinburgh, Scotland ; 1994:39-49.
28. Valiant LG. A bridging model for parallel computation. *Commun ACM*. 1990;33(8):103-111.
29. Chow ACH, Zeigler BP. Parallel devs: A parallel, hierarchical, modular, modeling formalism. *Winter Simulation Conference*, Orlando, FL; 1994:716-722.
30. ACIMS. Devs java. http://acims.asu.edu/software. Accessed March 28, 2017. Revised: 2014-05-05.
31. Filippi JB, Bisgambiglia P. Jdevs: an implementation of a devs based formal framework for environmental modelling. *Environ Model Software*. 2004;19(3):261-274.
32. Touraille L, Traoré MK, Hill DRC. A model-driven software environment for modeling, simulation and analysis of complex systems. *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, Boston, MA, USA; April 3-7, 2011: 229-237.
33. Wainer GA. Cd++: a toolkit to develop devs models. *Softw Pract Exper*. 2002;32(13):1261-1306.

34. Wainer GA. Improved cellular models with parallel cell-devs. *Trans Soc Comput Simul Int*. 2000;17(2):73-88.
35. Liu Q. Distributed Optimistic Simulation of Devs and Cell-Devs Models with Pcd++. *Master's thesis*, Carleton University, 2006.
36. Jafer S. New Algorithms for the Parallel Cd++ Simulation Environment. *Master's thesis*, Carleton University, 2007.
37. Barroso LA, Dean J, Hölzle U. Web search for a planet: the google cluster architecture. *Micro*. 2003;23:22-28.
38. Freire VFA, Cacheda F, Carneiro V. Analysis of performance evaluation techniques for large scale information retrieval. *LSDS-IR*, Rome, Italy; 2013:215-226.
39. Gil-Costa V, Lobos J, Inostrosa-Psijas A, Marín M. Capacity planning for vertical search engines An approach based on coloured petri nets. *Petri Nets*, Hamburg, Germany; 2012:288-307.
40. Badue CS, Almeida JM, et al. Capacity planning for vertical search engines. *CoRR*, *abs/1006.5059*, 2010.
41. Gan Q, Suel T. Improved techniques for result caching in web search engines. *WWW*, Madrid, Spain; 2009:431-440.
42. Fitzpatrick B. Distributed caching with memcached. *J Linux*. 2004;2004:72-76.
43. Zobel J, Moffat A. Inverted files for text search engines. *J CSUR*. 2006;38(2):6.1-6.56.
44. Broder AZ, Carmel D, Herscovici M, Soffer A, Zien JY. Efficient query evaluation using a two-level retrieval process. *CIKM*, Louisiana, USA; 2003:426-434.
45. Vlachos M. Identifying similarities, periodicities and bursts for online search queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data and Symposium on Principles Database and Systems*, Paris, France; June 13-18, 2004:131-142.
46. Marin M. An empirical assessment of optimistic PDES on BSP. *10th SCS European Simulation Symposium (ESS 1998)*, Nottingham, United Kingdom; 1998.
47. Marin M. Discrete-Event Simulation On The Bulk-Syncrhonous Parallel Model. *PhD thesis*, Oxford University, 1998.
48. Navarro G. A compact space decomposition for effective metric indexing. *Pattern Recognit Lett*. 2005;26:295-306.
49. Costa VG, Marín M, Inostrosa-Psijas A, Lobos J, Bonacic C. Modelling search engines performance using coloured petri nets. *Fundam Inform*. 2014;131(1):139-166.
50. Gil-Costa V, Tapia E, Marin M. Asynchronous approximate simulation algorithm for stream processing platforms (wip). *Summer Comput Simul Conf*, Montreal, Canada; 2016:521-526.

How to cite this article: Inostrosa-Psijas A, Gil-Costa V, Marin M, Wainer G . Semi-asynchronous approximate parallel DEVS simulation of web search engines. *Concurrency Computat Pract Exper*. 2018;30:e4149. https://doi.org/10.1002/cpe.4149