Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

DEVS MODELING OF LARGE SCALE WEB SEARCH ENGINES

Alonso Inostrosa-Psijas
Universidad de Santiago
Santiago, CHILE

Gabriel Wainer
Carleton University
Ottawa, CANADA

Veronica Gil-Costa
Universidad Nacional de San Luis
San Luis, ARGENTINA

Mauricio Marin
Yahoo! Labs Santiago
Santiago, CHILE

ABSTRACT

Modeling large scale Web Search Engines (WSEs) is a complex task. It involves many issues such as representing user's behavior, query traffic, several strategies and heuristics to improve query response time, etc. Typically, WSEs are composed of several services deployed in data centers, which must interact to get the best document results to user queries. Additionally, hardware specification like multithreading and network communications have to be taken into account. In this paper, we propose to model a service-based WSE using the Discrete Event System Specification (DEVS) formalism, which is one of the most powerful methodologies for discrete event systems. We validate our proposed model against an actual MPI implementation of the WSE and a process oriented simulation. We evaluate the accuracy of the proposed model by evaluating metrics such as query throughput and we show that there is no relevant differences, just small fluctuations of less than 4%.

1 INTRODUCTION

Large Web Search Engines (WSEs) must respond user queries within a fraction of a second. To accomplish this task, a WSE must process queries as soon as they arrive, deal with dynamic query arrival rates and maintain query response time below a predefined upper bound. Typically, processors workloads are kept below 40% at steady state operation to effectively support drastic increase in incoming query traffic. Since users normally react to natural disasters and social or political incidents that attract worldwide interest, sudden increases in query traffic are expected to happen at any time. To deal with these demands, WSEs must combine several strategies like caches, indexes, and different heuristics for query processing.

In these complex systems, where too many difficult issues must be taken into consideration, proper simulation models and simulation tools can be very useful for performance evaluation studies and testing of new query processing strategies. As usual in large scale systems, it is not always feasible to emulate real production environments due to the high cost of accessing large clusters of computers testing new algorithms without actual on-line query traffic. To be close to real settings, the simulation programs must (a) consider user behavior by feeding simulators with queries previously submitted by actual users from the WSE, (b) hide the complexities of simulating data center hardware, and (c) perform the same general steps than the actual implementation of the strategy where the model should allow the inclusion of user code associated with the processing of incoming queries so that new variants can be easily studied.

In this paper, we show that DEVS can be a useful tool for performing discrete-event simulation of large scale Web Search Engines. To do so, we built a DEVS model and compared with an actual implementation of a small experimental WSE and a process-oriented discrete-event simulator. DEVS provides a number of advantages for modelling and simulating similar systems (Zeigler et al. 2000, Wainer 2009):

1. DEVS is a hierarchical and modular technique that allows the description of the multiple levels needed for our model. DEVS allows models to be extended easily. Each model can be associated with an Experimental Framework used as a testing module. This approach improves testing.
2. DEVS provides a formal approach. Formal specification mechanisms are useful to improve the security and development costs of a simulation. DEVS supplies facilities to translate the formal specifications into executable models.
3. DEVS provides a way to specify models that can be coupled into higher level ones, which are later simulated by independent abstract entities (with a single processor or in parallel).
4. DEVS uses a continuous time base, which allows accurate timing representation. Precision of the conceptual models can be improved, and CPU time requirements reduced. Higher timing precision can be obtained without using small discrete time segments (that would increase the number of simulation cycles).
5. DEVS is a general discrete event formalism (i.e., every other method can be expressed as DEVS), and many techniques used for embedded systems have been mapped into DEVS (v.g., Verilog, VHDL, Fuzzy Logic, Petri Nets, etc.).
6. Logical and timing correctness rely on DEVS system theoretical roots and sound mathematical theory. The use of formal modeling techniques enables automated model verification.
7. DEVS is well adapted to make the composition of models in different methodologies put together. (a) The same model can be executed with different DEVS-based simulators, allowing for portability and interoperability at a high level of abstraction; (b) A well-defined separation of concerns enables models and simulators to be verified independently and reused in later combinations with minimal re-verification.

In our case, we want to model complex WSEs, which, typically are composed as a collection of different services where each service is deployed in a set of cluster processors dedicated exclusively to perform the service computations. Each processor is kept replicated to increase query throughput. In this paper, we refer to services such as Front Service (FS), Caching Service (CS) and Index Service (IS). Modeling each component of these distributed services is a complex task. Fortunately, ours is a coarse grain application where running costs of each service are dominated by a few primitive operations, so that once identified the relevant ones, it is fairly simple to model their actual costs on the target processors by means of benchmark programs. By exploiting this feature we have constructed DEVS models of WSEs where messages between atomic and/or coupled DEVS models represent queries exchanged among services.

The remaining of this paper is organized as follows. Section 2 presents related works and defines the DEVS formalism for modeling. Section 3 describes the WSE services and in Section 4 we define the DEVS model for a WSE. Section 5 presents experiments and we conclude in Section 6.

2 BACKGROUND

As WSEs are expensive to maintain, it is relevant for the engineers in charge of data-centers and the designers of new query processing strategies to have tools to predict the performance of these systems. In the literature of this area there have been few studies but, most of them are only suitable for small systems (Chowdhury and Pass 2003, Badue et al. 2010).

There have been various performance models presented in the literature for web search engines. A model based on process oriented simulation has been used in (Marin et al. 2010) to evaluate different cache mechanisms. The application of MVA algorithms (Menasce et al. 2004) for capacity planning of search engines is proposed in (Badue et al. 2006) and then extended in (Badue et al. 2010). However, this proposal was evaluated with only eight processors. The work in (Chowdhury and Pass 2003) presented a framework based on queuing network theory analyzing search systems in terms of operational requirements. They assumed perfect balance among the service times of processors and no validation was made against an actual implementation. The discrete-event simulation model developed in (Cacheda et al. 2007) represents a switched network. The simulation model was validated by comparing the estimated response times with those obtained using a real system.

The work in (Jiang et al. 2008) presented algorithms for performance analysis of general services in on-line distributed systems. The work in (Lu and Apon 2008) presented the design of simulation models to evaluate configurations of processors in an academic environment. The work presented in (Lin et al. 2005) proposed a mathematical model to minimize the resource cost for a server-cluster. In our application domain, the problem of using mathematical models is that they are not capable of capturing the dynamics of user behavior nor temporarily biased queries on specific query topics. The work in (Lin et al. 2005) was extended in (Zhang et al. 2009) to include fault tolerance.

Additionally, models based on Petri Nets have been widely used for performance analysis of various kinds of systems. Regarding WSEs, the work in (Gil-Costa et al. 2012) used the CPN tools (cpntools.org) to define a Petri Net aimed to estimate the performance achieved by small sized WSEs. However, this scheme requires large running time to simulate the query processing process. This work is extended in (Gil-Costa et al. 2014) by proposing a faster simulator capable of modeling larger WSEs.

2.1 DEVS

DEVS is a formalism for modeling and simulation of Discrete-Event Dynamic systems. It defines a way of specifying systems whose states change upon the reception of an input event or the expiration of a time delay. It also allows for hierarchical decomposition of the model by defining a way to couple existing DEVS models. The original DEVS model is a structure (Zeigler et al. 2000):

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

where

X	is the set of *external* events;
Y	is the set of *output* events;
S	is the set of *sequential* states;
$\delta_{ext} : Q \times X \to S$	is the *external* state transition function;

where $Q := \{(s,e) \mid s \in S,\ 0 \le e \le ta(s)\}$ and e is the elapsed time since the last state transition.

$\delta_{int} : S \to S$	is the *internal* state transition function;
$\lambda : S \to Y$	is the *output* function;
$ta : S \to R_0^+ \cup \infty$	is the *time advance* function;

The semantics for this definition are as follows. At any given time, a DEVS model is in a state $s \in S$ and in the absence of external events, 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 between 0 and ∞. Transitions that occur due to the expiration of $ta(s)$ are called **internal transitions**. When an internal transition takes place, the system outputs the value $\lambda(s)$, and changes to state $\delta_{int}(s)$. A state transition can also happen when an external event occurs. In this case, the new state is given by δ_{ext} based on the input value, the current state and the elapsed time. Figure 1, illustrates this definition by specifying a model of a computer processor using DEVS.
A *coupled model* is a structure:

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

where D is a set of components, and

for each $i \in D$,

M_i is a component with constraint that

$M_i = <X_i, Y_i, S_i, \delta_{i\ ext}, \delta_{i\ int}, \lambda_i, ta_i>$ is a DEVS 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

select is a tie-breaker function
I_i is a subset of $D \cup \{self\}$, i is not in I_i,
$Z_{self,j} : X_{self} \rightarrow X_j$
$Z_{i,self} : Y_i \rightarrow Y_{self}$
$Z_{i,j} : Y_i \rightarrow X_j$
select : Subset of $D \rightarrow D$
such that for any non-empty subset E,
$select(E) \in E$

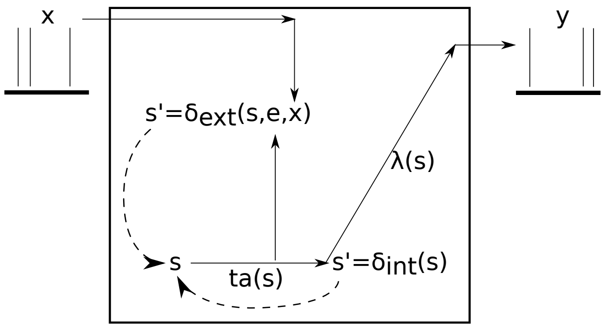


Figure 1: DEVS semantics specification of a computer processor using DEVS.

A coupled model groups several DEVS models together into a compound model that can be regarded, due to the closure property, as another DEVS model. This allows for hierarchical model construction. A DEVS model that is not constructed as a coupled model is known as an atomic model. A coupled model can have its own input and output events, as defined by the X_{self} and Y_{self} sets. Upon receiving an external event, the 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. All these input-output mappings are defined by the Z function.

When models are coupled together, ambiguity arises when there are more than one component scheduled for an internal transition at the same time. 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 way the DEVS formalism solves this problem is by the use of the select function. Only one model of the group of imminent models will be allowed to have $e = 0$. The other imminent models will be divided into two 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.

DEVS is independent from any simulation mechanism, which allowed several simulation tools to be developed, tackling different needs and providing advantages on specific scenarios. A non-comprehensive list includes: DEVSJAVA (ACIMS) which allows hierarchical model definition and visualization; the visual tool named JDEVS (Filippi and Bisgambiglia 2004), SimStudio (Touraille et al. 2011) which is a web-based framework and CD++ (Wainer 2002) which supports both standalone and parallel/distributed simulation of DEVS and Cell-DEVS models.

3 MODEL REQUIREMENTS

Modern WSEs are composed of services deployed on clusters of computers. Usually, each service is devoted to a single specialized operation related to the processing of user queries arriving to the search engine. Typical services are the Front Service (FS) or brokers, the Caching Service (CS) and the Index Service (IS). Each service is deployed on several nodes, where each node is hosted by a dedicated cluster processor. In other words, these services are deployed on clusters of computers and their processing nodes are allocated in racks connected via network switches. These services are usually implemented as arrays of $p \times r$ nodes, as shown in figure 2, which allows low response times and high query throughput. p indicates the level of data partitioning and r the level of data replication. The value of p has a direct impact on the response time of individual queries. For instance, for a cache service, p is directly related to the overall amount of main memory space used to hold the cache entries of the service. Replication refers to maintaining copies of the data associated with each partition. The value of r has a direct impact on the query throughput, namely the number of queries processed per time unit.

The FS handles and manages queries submitted by users by routing them to the CS or to the IS. It is also responsible for merging partial results for queries and delivery the final top-k results to users. The Cache Service keeps track of the most frequent queries and their results. The Index Service exploits an inverted index (Gan and Suel 2009) to calculate the top-k document results when the query is not found in the CS. The inverted index (also known as inverted file) is a data structure used by all well-known WSEs. It is composed of a vocabulary table that contains the distinct relevant terms found in the document collection, and a set of posting lists, one for each term. The posting lists for a given term c stores the document identifiers that contain the term c, along with additional data used for ranking purposes. In order to solve a query, one must fetch the posting lists for the query terms, compute the intersection among them, and then compute the ranking of the resulting intersection. For a given query, every IS partition performs the same operations with different parts of the inverted index.

The processing of queries in a WSE is depicted in figure 2. Upon the arrival of a new query (step 1), the FS initializes the query by adding some meta-data which defines, for example, whether to compute snippets. Then, it is delivered to a CS node (step 2) in order to check whether the query has been previously processed to avoid unnecessary computation. Inside the CS cluster, only one partition and replica is selected to process the query. If the query is cached, the CS node sends the query answer to FS (step 3), which in turn sends the results to the user. If the query is not found in the cache, the CS sends a hit-miss message to the FS. At this point (step 4), the FS sends the query to all IS_p partitions of the IS. In other words, the FS selects a replica in each partition which determines the local top-k query results. Once each IS partition has sent back their local results (step 5), the FS merges the partial results to deliver the top-k documents to the user (step 7). The cache is updated with the query and its document results (step 6).

The WSE is deployed in a data-center composed by N racks holding M processors each, which communicate with each other by means of a set of switches composing a Fat-Tree network topology (Al-Fares et al. 2008). This network allows a high level of parallelism and avoid congestion by providing different routes for sent messages depending of its destination by means of a two-level routing table.

We call $<FS_r, CS_p, CS_r, IS_p, IS_r>$ a service configuration, where FS_r indicates the amount of processing nodes that compose the FS, CS_p corresponds to the partitioning level and CS_r is the level of replication of each partition, used for the CS. The same convention is applied to the IS. Thus, the total number of processing nodes (TP) in a cluster holding these three services is $TP = FS_r + (CS_p \times CS_r) + (IS_p \times IS_r)$.

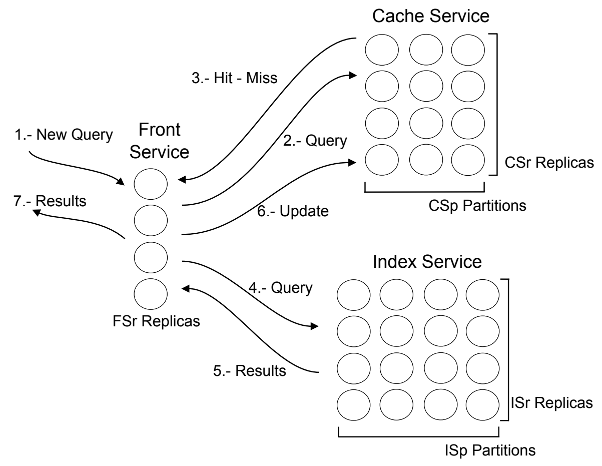


Figure 2: Service deployment and query processing process.

4 MODELING LARGE SCALE WEB SEARCH ENGINES WITH DEVS

The structure of the DEVS model that we defined for a service-based WSE and we used to conduct various experiments, is shown in figure 3. The WSE services are represented by coupled models (shaded grey areas). In addition, these coupled models are composed by coupled and atomic models. The coupled models are composed by a QueryRouter, which is responsible to deliver a query object to the corresponding service partition/replica. Each query object keeps information about the query identifier (qID), the query terms, and some statistics like the query arrival time and the query service time.

Query objects are generated by the *Query Generator* atomic model and sent to the Front Service. The Query Generator reads the query terms and posting list sizes of each term from a query log. Query inter-arrival time is simulated using an exponential distribution.

Inside the FS, the query objects are received through the *"In"* port and delivered to the QueryRouter atomic model, which routes them (in *round-robin* fashion) to one of the FS atomic models. The first time a query arrives to a FS atomic model we simulate the cost of initializing the query. Then it is sent to the *"toCS"* port of the Front Service, since it is connected to the Cache Service.

The CS and IS have a high level and a low level QueryRouter atomic model. The former, called QueryRouter_Part is used to select the service partition. The last one, called QueryRouter_i with $i = 1 \dots p$, delivers queries to the r processing elements representing replicas. Inside the CS, the high level QueryRouter selects a single partition whereas inside the IS the QueryRouter_Part sends the query object to all IS_p partitions. Then, when a query arrives to the CS, it is received by the QueryRouter_Part atomic model which routes the query to one of the "QueryRouter_i" atomic model. The $i-th$ partition is selected by computing a hash function on the query terms. Afterwards, the corresponding QueryRouter_i model selects a CS atomic model in *round-robin* fashion. The selected CS marks the query with a "Hit" or "Miss" and delivers it to the *"out"* port of the CS coupled model, which in turn sends the query back to the FS.

At the FS side, if the query has a "Hit" mark, then it is count as finished and statistic information regarding query processing time and other relevant metrics are sent to the *S*tatistic atomic model. On the other hand, if it has a "Miss" mark, it is sent to the Index Service by putting it in the *"toIS"* port.

The queries arriving to the Index Service are first delivered to the "QueryRouter_Part" atomic model which delivers a copy of the query to each of the IS_p "QueryRouter_i" models at the current coupled model. Next, each of these "QueryRouter_i" models select one IS atomic model (in *round-robin* fashion) to send the query. The IS atomic model simulates the computation of the top-k local document results and send them along the query object back to the Front Service via the *"out"* port of the Index Service coupled model. When the FS model receives IS_p messages containing the local top-k document results for a given query, a merge operation is simulated. The query is finished and statistic information is sent to the Statistic

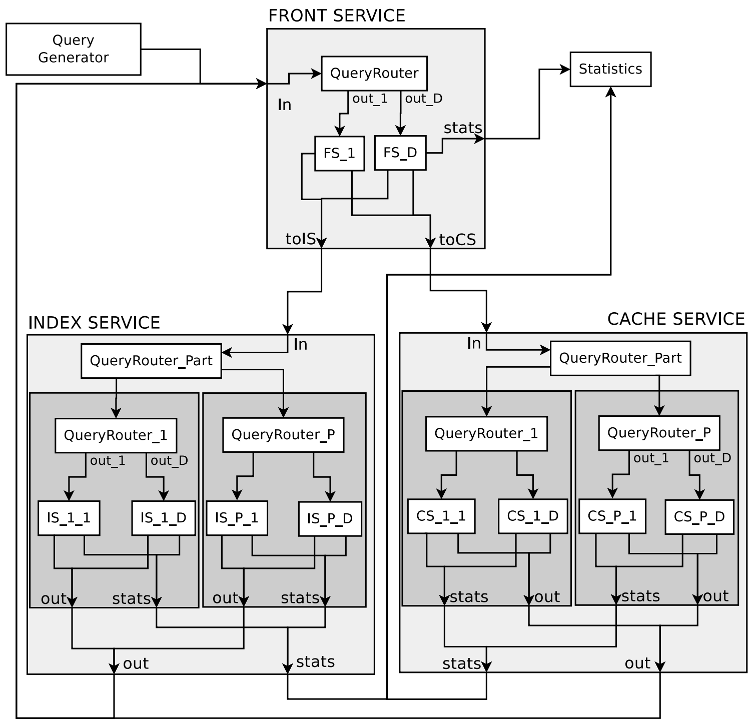


Figure 3: A WSE model and its internal and external connections.

atomic module. Finally, whenever a CS or IS atomic model processes a query, in addition of sending it to the corresponding *'out"* port it is also sent to the Statistics atomic model. This model is in charge of computing statistics such as throughput, query response time, number of cache hits, service utilization, etc.

We now present the FS expressed with the DEVS formalism. The DEVS formalism for others WSE components can be easily obtained from these mathematical forms. We do not include them in this paper for lack of space.

$$FrontService = \; < X_{FS}, Y_{FS}, D_{FS}, \{M_{d_{FS}}\}, \{I_{FS}\}, \{Z_{FS}\}, select_{FS} >$$

$$\begin{aligned}
X_{FS} &= (in, Q) \text{ Where Q is the set of Query Objects} \\
Y_{FS} &= \{(toIS, Q); (toCS, Q); (stats, Q)\} \\
D_{FS} &= \{\text{QueryRouter, FS}_i\} \; \forall \; i \in [1, D] \\
M_{d_{FS}} &= \{\text{QueryRouter, FS}_i\} \\
I_{FS} &= \{((\text{self, in}), (\text{QueryRouter, in}))\} \\
Z_{FS} &= \{((\text{QueryRouter, out}_i), (\text{FS}_i\text{, in})); ((\text{FS}_i\text{, toCS}), (\text{self,toCS})); ((\text{FS}_i\text{, toIS}), (\text{self, toIS})); \\
&\quad ((\text{FS}_i\text{, stats}), (\text{self, stats}))\} \\
select_{FS} &= \{\text{QueryRouter, FS}_i\}
\end{aligned}$$

5 EXPERIMENTS

In order to simulate the behavior of a WSE and its relevant costs properly, the simulator uses actual query logs, document posting lists and ranking times. (V.g., it is a trace driven simulator that uses actual data to study performance). We used a log of 36,389,567 queries submitted to the AOL Search service between March 1 and May 31, 2006. We pre-processed the query log following the rules applied in (Gan and Suel 2009) (removal of stopwords, term normalization, deletion of duplicated terms and assumption that two queries are identical if they contain the same words, no matter the order). The resulting log had 16,900,873 queries, where 6,614,176 are unique queries and the vocabulary consists of 1,069,700 distinct query terms. We also used a sample (1.5TB) of the UK-Web obtained in 2005 by the Yahoo! search engine, over which an inverted index with 26,000,000 terms and 56,153,990 documents was constructed. In this section we show how this data was used to validate our proposed DEVS model for a WSE. We executed the queries against this index in order to get the empirical cost distributions for our models. The document ranking method used was WAND (Broder et al. 2003) and the cache policy used was LRU.

In this section we compare our DEVS model against an actual MPI WSE implementation and a complex process oriented simulator (POS) (Marin et al. 2010) of the same system which uses co-routines. The process-oriented is developed in C++ using the libcppsim library (Marzolla 2004) presented in (Gil-Costa et al. 2013, Marin et al. 2010). Each process is implemented as a *co-routine* that can be paused and resumed at will during the execution of the simulation. To this purpose, it provides *passivate* and *activate* functions. This library also provides the *hold(t)* operation for pausing the co-routine for t units of simulated time, where t is the cost associated to each primitive operation. The dominant costs of primitive operations come from tasks related to ranking of documents, intersection of inverted lists, merge operation, etc. It simulates the main operations performed by a WSE using a very detailed model which includes the execution of a DAG associated with each query and the BSP (Valiant 1990) cost model to evaluate performance metrics. POS uses a processes and resources approach. Processes represent threads in charge of tasks processing. Resources are artifacts such as inverted lists, cache memory and hardware components like CPU and switches. Competition for resources among the co-routine is modeled with concurrent queuing objects.

In (Gil-Costa et al. 2014), we run benchmark programs to measure the costs on production hardware (ranking, merge, communication, etc.). We introduced these costs into the DEVS model to properly simulate the query processing process in a WSE. We evaluated various metrics related to precision of results, execution time to obtain results, and memory utilization. The experiments were executed on a HP PROLIANT DL Server with Dual 3.2GHz Intel Xeon processors and 2GB of RAM memory.

5.1 Accuracy Evaluation

Table 1: Service configuration and operation costs achieved by benchmark programs.

Processors	FS_r	CS_p	CS_r	IS_p	IS_r	Service Time of the IS	Cache hit rate
115	12	3	1	10	10	0.10036	0.332948
189	22	7	1	16	10	0.0629947	0.372335
200	24	6	1	17	10	0.0592222	0.39062
240	40	20	1	30	6	0.0338921	0.435852

In this section we evaluate the accuracy of the DEVS model by computing the Pearson correlation and the relative error between the DEVS and the MPI implementation. Table 1 shows several service configurations used in our experiments. In total, these configurations range from 115 to 240 processors. We also show the cost required by the IS and the number of cache hits reported by the CS due to these values depends on the number of partition/replicas used to deploy the services. We set $k = 10$, as the number of top-k document results and the query arrival time $\lambda = 1000$ queries per second (q/s). In particular, the cost associated with the ranking operation goes down as we increase the number of IS partitions from 0.10036 to 0.0338921, because the inverted index is evenly distributed among those partitions. Then, a larger number of partitions imply a smaller index in each IS node and therefore it reduces the computation costs associated with the ranking process. Similarly, the number of cache hits increases as we increment the number of CS partition as there is more memory space available to keep the most frequent queries along its top-k document results. The costs of the operations performed by the FS and CS are independent of the number of partitions/replicas. In this case, their cost depends on the number of top-k document results. After running the benchmark programs to measure the merge cost in the FS, the insert/update/erase operation costs in the CS, and the intersection and ranking costs in the IS, we passed these costs to the DEVS simulator.

Figure 4 shows query throughput results obtained by a real MPI implementation of the WSE, the POS and DEVS simulations 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 as described in Table 1. Overall, the results indicate that simulations are able to predict performance trend of the service configurations. Both POS and DEVS show that they can correctly estimate the behavior of the real system. The accuracy of the throughput metric achieved by the DEVS model reported a Pearson correlation of 0.98 which means a positive correlation between the values reported by the DEVS model and the real MPI implementation. We also obtained a small relative error of 0.005, which indicates how good the throughput computed by the DEVS model is relative to the real throughput. The relative error is computed as (e_r) as $\varepsilon_m/\bar{x}$, where $\varepsilon_m = \sqrt{(\sum(x_i - \bar{x})^2/(n \cdot (n-1))}$ and n is the number of service configuration (size of the sample).

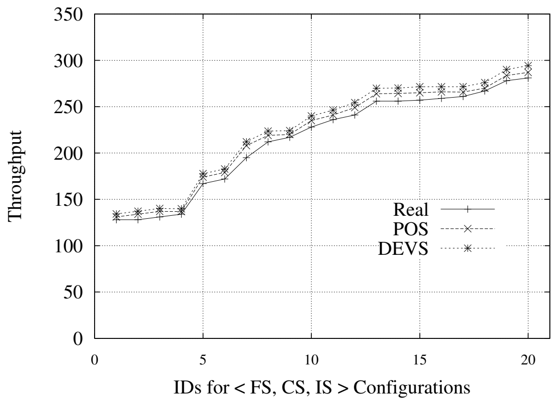


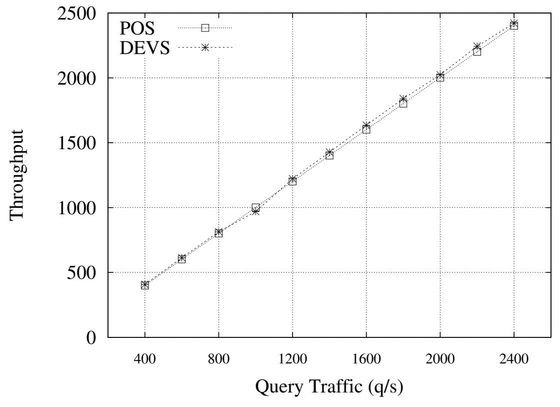
Figure 4: Simulation vs. real MPI implementation.

In Figure 5, we show the throughput and query response time of the DEVS and POS models when varying the query traffic. We fixed the service configuration with a total of 829 processors as $<15,7,2,10,80>$. Then, as we increased the query traffic, all services saturated. In other words, service utilization is higher and queries have to wait to be serviced. With a low query traffic (400 q/s) service utilization is 11%-8%-8% for the FS-CS-IS respectively. As we increase query traffic to 2400 q/s, the system reports a service utilization of 55%-45%-44%. Figure 5.(a) shows that the DEVS results almost overlaps the values reported by the POS simulator. In this case, the throughput grows linearly with the query traffic, as the number of processors simulated in this experiments are not saturated (55% at most) and therefore queries are processed as soon as they arrive to the system. The Pearson correlation for this experiment is 0.99 which indicates that the DEVS model can adequately estimate the throughput when the simulation scenario is changed.

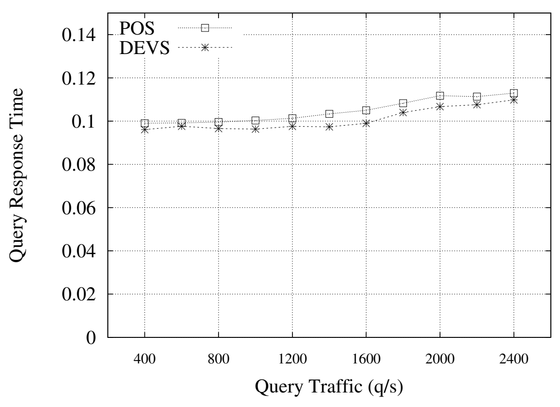
Finally, figure 5.(b) shows the average query response time estimated by both POS and DEVS models. Query response time presents a smooth increment with a high query traffic (2000-2400 q/s). As in previous experiments, this metric is also well-estimated by the DEVS model obtaining a Pearson correlation of 0.95.

5.2 Performance Evaluation

In this section we evaluate the performance achieved by the DEVS model. All results achieved a variance lower than 1%. We simulate the execution of 200.000 queries for the same service configurations depicted in Table 1. Figure 6.(a) shows at top the average memory consumption in Mb reported by the DEVS and POS approaches. At bottom, we show the maximum memory consumption. As expected, a larger configuration size demands more memory. Regarding the simulation approaches, the POS requires less memory, especially when simulating small service configurations. When simulating a larger number of resources (FS,CS,IS) both approaches tend to report almost the same memory consumption.



(a)



(b)

Figure 5: (a) Throughput achieved with different query traffics. (b) Query response time.

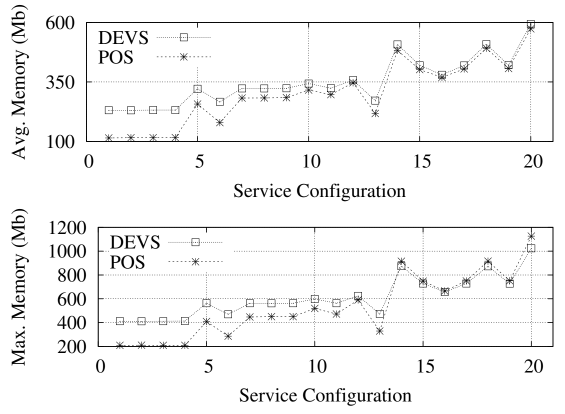
Figure 6.(b) shows the execution time in seconds reported by the POS and DEVS simulation approaches. Again, both curves tend to rise due to larger configuration sizes require larger running times. In average, the POS simulator reports execution times 32% lower than the ones reported by the DEVS. This is not only due to the additional memory consumption reporting DEVS, but when simulating with DEVS, the more coupled elements are used, the more objects are created. This, in turn, produces events and messages that are sent between atomic models (entities) that are in different coupled models. Messages are forwarded by each level of the hierarchical model. Output messages produced inside an entity are transformed to external message by the coordinator of the Logical Processors (LPs) before forwarding them to their destination. In other words, DEVS is a complex simulation framework with many control and coordinators entities. Additional computation cost is produced by control events generated to maintain the correct execution of events.

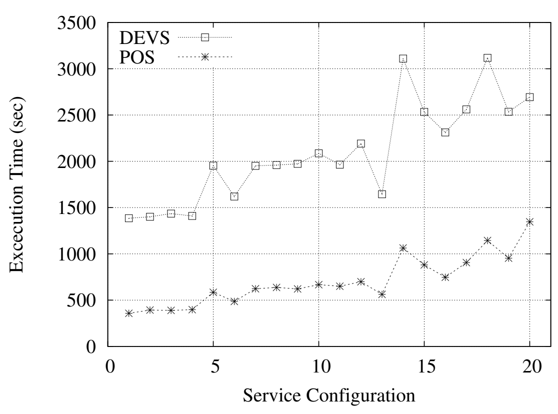
6 CONCLUSIONS

We have described a DEVS-based simulation model for understanding the performance of web search engines. We compared its usefulness against an actual implementation of a WSE and a complex processed-oriented simulation (POS) model. POS is difficult to develop and depends on the particular simulation library, whereas DEVS is easier to learn and use. The evaluation in accuracy of results shows no significant statistical differences with the real implementation and the process oriented simulator. Both, DEVS and POS, showed high levels of accuracy. Coupled models are a powerful feature of DEVS regarding code re-utilization. However, its higher level of abstraction becomes a drawback in large models since it demands high memory consumption and it requires messages passing between control and coordinators entities which has a direct impact on degrading execution time. Results show that POS requires lower execution times in comparison to DEVS. Nevertheless - despite the performance differences achieved - our work shows that DEVS is a powerful and useful multi-purpose formalism for modelling and simulation of WSEs. Its ease of use presents it as a more than viable alternative for modelling and simulating large scale WSE.

ACKNOWLEDGMENTS

This research was partially supported by NSERC, the ELAP program and the supercomputing infrastructure of the NLHPC (ECM-02) at Centro de Modelación y Computación Científica at Universidad de La Frontera CMCC-UFRO. Partially funded by Fondef project CA12i10314 and Mincyt-Conicyt CH1204.





(a) (b)

Figure 6: (a) Maximum and average memory consumption. (b) Execution time.

REFERENCES

ACIMS. “DEVSJAVA”. http://acims.asu.edu/software. Accessed: 2014-05-05.

Al-Fares, M., A. Loukissas, and A. Vahdat. 2008. “A scalable, commodity data center network architecture”. *SIGCOMM* 38:63–74.

Badue, C. S., J. M. Almeida, V. Almeida, R. A. Baeza-Yates, B. A. Ribeiro-Neto, A. Ziviani, and N. Ziviani. 2010. “Capacity Planning for Vertical Search Engines”. *CoRR* abs/1006.5059.

Badue, C. S., R. A. Baeza-Yates, B. A. Ribeiro-Neto, A. Ziviani, and N. Ziviani. 2006. “Modeling performance-driven workload characterization of web search systems”. In *CIKM*, 842–843.

Broder, A. Z., D. Carmel, M. Herscovici, A. Soffer, and J. Y. Zien. 2003. “Efficient query evaluation using a two-level retrieval process”. In *CIKM*, 426–434.

Cacheda, F., V. Carneiro, V. Plachouras, and I. Ounis. 2007. “Performance analysis of distributed information retrieval architectures using an improved network simulation model”. *Inf. Process. Manage.* 43 (1): 204–224.

Chowdhury, A., and G. Pass. 2003. “Operational requirements for scalable search systems”. In *CIKM*, 435–442.

Filippi, J.-B., and P. Bisgambiglia. 2004. “JDEVS: an implementation of a DEVS based formal framework for environmental modelling”. *Environmental Modelling and Software* 19 (3): 261–274.

Gan, Q., and T. Suel. 2009. “Improved techniques for result caching in web search engines”. In *WWW*, 431–440.

Gil-Costa, V., A. Inostrosa-Psijas, M. Marín, and E. Feuerstein. 2013. “Service Deployment Algorithms for Vertical Search Engines”. In *PDP*, 140–147.

Gil-Costa, V., J. Lobos, A. Inostrosa-Psijas, and M. Marín. 2012. “Capacity Planning for Vertical Search Engines: An Approach Based on Coloured Petri Nets”. In *Petri Nets*, 288–307.

Gil-Costa, V., M. Marín, A. Inostrosa-Psijas, J. Lobos, and C. Bonacic. 2014. “Modelling Search Engines Performance Using Coloured Petri Nets”. *Fundam. Inform.* 131 (1): 139–166.

Jiang, G., H. Chen, and K. Yoshihira. 2008. “Profiling services for resource optimization and capacity planning in distributed systems.”. *J. of Cluster Computing*:313–329.

Lin, W., Z. Liu, C. H. Xia, and L. Zhang. 2005. “Optimal capacity allocation for Web systems with end-to-end delay guarantees”. *Perform. Eval.*:400–416.

Lu, B., and A. Apon. 2008. “Capacity Planning of a Commodity Cluster in an Academic Environment: A Case Study”. In *LCI*.

Marin, M., V. Gil-Costa, and C. Gomez-Pantoja. 2010. “New Caching Techniques for Web Search Engines”. In *HPDC*, 215–226.

Marzolla, M. 2004. “LibCppSim: A SIMULA-like, portable process-oriented simulation library in C++”. In *ESM*.

Menasce, D. A., V. A. Almeida, and L. W. Dowdy. 2004. *Performance by Design: Computer Capacity Planning*. Prentice Hall.

Touraille, L., M. K. Traoré, and D. R. C. Hill. 2011. “A model-driven software environment for modeling, simulation and analysis of complex systems”. In *TMS*, 229–237.

Valiant, L. G. 1990. “A Bridging Model for Parallel Computation”. *Commun. ACM* 33 (8): 103–111.

Wainer, G. 2009. *Discrete-Event Modeling and Simulation: a practitioner approach*. CRC Press. Taylor and Francis.

Wainer, G. A. 2002. “CD++: a toolkit to develop DEVS models”. *Softw., Pract. Exper.* 32 (13): 1261–1306.

Zeigler, B. P., T. G. Kim, and H. Praehofer. 2000. *Theory of Modeling and Simulation*. 2nd ed. Academic Press, Inc.

Zhang, C., R. N. Chang, C.-s. Perng, E. So, C. Tang, and T. Tao. 2009. “An Optimal Capacity Planning Algorithm for Provisioning Cluster-Based Failure-Resilient Composite Services”. In *SCC*, 112–119.

AUTHOR BIOGRAPHIES

ALONSO INOSTROSA-PSIJAS received his Computer Science degree from the University of Santiago in 2007. Currently, he is a PhD Student at the University of Santiago. He is an intern at Yahoo! Labs Santiago. As a recipient of the ELAP scholarship awarded by the Canadian government in 2013, he worked as an intern at ARS Simulation Laboratory at Carleton University. His email address is alonso.inostrosa@usach.cl.

GABRIEL WAINER, SMSCS, SMIEEE, received the M.Sc. (1993) at the University of Buenos Aires, Argentina, and the Ph.D. (1998, with highest honors) at the Université d’Aix-Marseille III, France. In July 2000 he joined the Department of Systems and Computer Engineering at Carleton University (Ottawa, ON, Canada), where he is now Full Professor. He is the author of three books and over 280 research articles in areas of DEVS parallel and distributed simulation. He has the First Bernard P. Zeigler DEVS Modeling and Simulation Award, the SCS Outstanding Professional Award (2011) Carleton University’s Mentorship Award (2013), the SCS Distinguished Professional Award (2013) and Carleton University’s Research Achievement Award (2005,2014). His email address is gwainer@sce.carleton.ca.

VERONICA GIL-COSTA received her MSc (2006) and PhD (2009) in Computer Science, both from Universidad Nacional de San Luis (UNSL), Argentina. She is currently a professor at the University of San Luis, Assistant Researcher at the National Research Council (CONICET) of Argentina and a researcher for Yahoo! Labs Santiago (YLS). Her email address is gvcosta@unsl.edu.ar.

MAURICIO MARIN is a professor of computer engineering at University of Santiago, Chile. His PhD is from the University of Oxford, UK (1999) with a thesis dissertation on parallel computing. Currently he is the head of Yahoo! Labs Santiago in Chile which is a center for applied research on scalable infrastructure for the Web and information retrieval. His email address is mmarin@yahoo-inc.com.