Performance Analysis of Web-based Distributed Simulation in DCD++: A Case Study across the Atlantic Ocean

Gabriel Wainer, Qi Liu
Department of Systems and Computer Engineering
Carleton University Centre on Visualization and Simulation (V-Sim)
Carleton University, Ottawa, ON, Canada K1S 5B6
http://www.sce.carleton.ca/faculty/wainer

Julien Chazal, Loïc Quinet, Mamadou K. Traoré

LIMOS CNRS UMR 6158 UFR ST,
Université Blaise Pascal, Clermont-Ferrand 2 Campus des Cézeaux, 63177 Aubière Cedex, France
http://www.isima.fr/~traore

Keywords: DEVS, Cell-DEVS, CD++, Web Services

Abstract

This paper presents a case study of Web-based distributed simulation across the Atlantic Ocean between Canada and France. The distributed simulation engine, known as DCD++, extends the CD++ environment to expose the simulation functionalities as machine-consumable services based on the DEVS and Cell-DEVS formalisms and commonly-used Web Service technologies. DCD++ provides a platform that represents a step further towards transparent sharing of computing power, data, models, and experiments in heterogeneous environment on a global scale. Also, the simulation service can be easily integrated with other services such as visualization, network management, and geographic information services in a larger system. Experiments have been carried out to investigate simulation performance over commodity Internet connections, and major bottlenecks in the system have been identified. Based on the experimental results, we put forward several areas that warrant further research.

1. INTRODUCTION

With the computing power and advanced software tools available today, Modeling and Simulation (M&S) becomes a powerful tool for analyzing and designing a broad array of complex systems where a mathematical analysis is intractable. The Discrete Event System Specification (DEVS) [1] is a general modeling framework that has gained growing popularity in recent years, in part due to its clear separation between the model and simulation concepts, natural support for hierarchical and modular construction of models, and the ability to verify models and simulators independently and reuse them in later combinations with minimal re-verification. Since its first formalization, DEVS has been extended into various directions. The P-DEVS formalism [2] eliminates the serialization constraints in the original DEVS definition, allowing increased parallelism to be exploited in the simulation. The Cell-DEVS formalism [3] combines Cellular Automata (CA) [4] with DEVS theory to describe n-dimensional cell spaces as discrete-event models, where each cell is represented as a DEVS basic model that can be explicitly delayed using built-in timing constructions.

CD++ [5] is an open-source M&S environment that implements P-DEVS and Cell-DEVS formalisms and has been used to successfully solve a variety of sophisticated problems. Over the years, CD++ has been ported to different platforms, including several parallel versions, referred to as PCD++, that employ both conservative and optimistic synchronization protocols to achieve high-performance simulations on distributed-memory cluster systems [6][7]. Nevertheless, as the system under study becomes more and more complicated, the complexity of the model grows significantly. As a result, the simulation tends to be increasingly time-consuming and requires resources that cannot be satisfied by any single site alone. Constructing a common framework to hook together geographically distributed resources in collaborative simulation of large-scale and highly complex models starts to gain momentum in the research society.

Grid computing offers a new paradigm for resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organizations. In a Grid environment, functionalities of various applications are exposed as web services that can be accessed and consumed by other web services without human intervention in a platform-independent manner. In order to integrate DEVS-based simulation services with other services such as visualization and GIS (Geographical Information Systems) services in a Grid environment, the CD++ toolkit was redesigned to support Web-based simulation using standard Web Service (WS) technologies. The resulting distributed simulator, known as DCD++, allows for multi-user concurrent simulation over both commodity Internet connection and dedicated point-to-point fiber links [8][9]. Preliminary experimentations have been conducted to evaluate the capability of DCD++ between two cities in Canada. In this paper, we perform further performance analysis that involves distributed simulation of different models across the Atlantic Ocean between Canada and France. The objective of our study is two fold. First, we want to test the concept of transparent sharing of computing power, data, models, and experiments in DEVS-based simulations on a global scale. Secondly, we want to gain insight into the potential bottlenecks, particularly the communication overhead due to limited bandwidth and high latency of the

Internet connection, and the effect they may have on the system performance.

The rest of the paper is organized as follows. Section 2 introduces the P-DEVS and Cell-DEVS formalisms and provides the necessary background knowledge on CD++ and commonly used WS technologies. It also gives a brief survey on the existing DEVS-based toolkits intended for parallel and distributed simulation (PADS). Section 3 discusses the architecture and main features of the DCD++ simulator. Section 4 describes the experimental environment and the metrics we used to investigate the simulation system. Section 5 presents a detailed performance analysis. And Section 6 closes the paper with conclusion and future work.

2. BACKGROUND

In a discrete-event simulation, the system being simulated changes state only at discrete points in time, upon the occurrence of events. Based on dynamic systems theory, the P-DEVS formalism [2] describes a system as a composition of behavioral (atomic) and structural (coupled) components. A P-DEVS atomic model is defined as:

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

At any given time, an atomic model is in some state $s \in S$. Without the occurrence of external events, it remains in state s for a period of time of ta(s), known as the lifetime of state s. When the lifetime expires, the atomic model outputs value $\lambda(s) \in Y$, and changes to a new state given by the internal transition function $\delta_{int}(s)$. A P-DEVS model uses a bag of inputs (X^b) to support the execution of multiple concurrent events. If one or more external events $x \in X$ occur before the expiration of ta(s), the model transfers to a state determined by the external transition function $\delta_{ext}(s,e,X^b)$, combining multiple transitions into a single one. A confluent transition function δ_{con} is defined to determine the next state in the case of collisions when a model receives external events at the same time of its internal transition.

P-DEVS has a well-defined concept of system modularity and component coupling to form composite models. A P-DEVS coupled model is formally defined as:

$$N = <X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC>.$$

The sets of input and output events are defined by X and Y respectively. D is a set of indices for the components of a coupled model and, for each $d \in D$, M_d is a basic P-DEVS model (atomic or coupled). The external input coupling (EIC) specifies the connections between external and component inputs, while the external output coupling (EOC) describes the connections between component and external outputs. The connections between the components themselves are defined by the internal coupling (IC).

Cell-DEVS formalism [3] is an extension of the traditional CA theory to improve execution efficiency and precision of the simulated model. It describes n-dimensional cell spaces as discrete-event DEVS coupled models, where each cell is represented as a DEVS atomic model. Further, it defines timing constructions for each cell, allowing explicit timing specification, asynchronous model execution, and seamless integration with other types of models. A Cell-DEVS atomic model is formally defined as:

$$C = <X, Y, I, S, \theta, N, delay, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D>.$$

Each cell has a modular interface (I) consisting of a fixed number of ports connected to its neighboring cells. The future state of a cell is computed by the local transition function (τ) based on the cell's current state and input values. State changes are propagated only after a delay given by the delay function (d). Each cell also has the computing apparatus (δ_{int}, δ_{ext}, and λ) as defined in P-DEVS atomic models. Cells are coupled by the neighborhood relationship to form a cell space, which can then be integrated with other DEVS and Cell-DEVS models. A cell space is formally defined as a Cell-DEVS coupled model:

$$GCC = <X_{list}, Y_{list}, I, X, Y, \eta, \{t_1, \ldots, t_n\}, N, C, B, Z>.$$

The cell space (C) comprises a fixed-sized n-dimensional array of cells, and the relative position between a cell and its neighbors is defined by the neighborhood set (N). B specifies the border of the cell space, which can be wrapped (i.e., all cells have exactly the same behavior) or non-warped (i.e., the border cells have a different behavior from others in the cell space). The translation function (Z) defines the input/output coupling between the cells.

The CD++ environment [5] provides a set of simulation engines to execute DEVS and Cell-DEVS models on different platforms. It decouples the modeling and simulation concepts by providing two separate frameworks. A *modeling framework* is defined as a hierarchy of classes that allow users to specify the behavior of atomic and coupled models. For each DEVS atomic model, users need to implement the various functions as required by the P-DEVS formalism in a C++ class, which is then incorporated into the modeling hierarchy during compilation. For DEVS coupled models and Cell-DEVS models, users can specify the coupling information and other attributes of cell spaces in a text-based configuration file using a built-in specification language. In addition, CD++ provides a *simulation framework* that creates an executive entity for each component in the modeling hierarchy to implement the abstract simulator that is responsible for executing the simulation in line with the formalisms [10]. These executive entities are specialized into two categories, namely *simulators* and *coordinators*. Simulators are associated with atomic models to trigger the output and state transition functions, while coordinators are attached to coupled models to keep track of the simulation time and to relay messages between their child simulators and parent coordinators. In parallel conservative simulations, a special root coordinator is employed as a central controller to handle the advance of simulation time and to communicate between the simulated model and the surrounding environment. In order to run parallel and distributed

simulations, the model is decomposed into several partitions (as specified in a user-supplied partition file), each executed by a separated process running on a distinct processor. Traditionally, users submit the model definition and partition files to the CD++ simulator via command-line arguments. At the end of a simulation, the execution results are recorded in output and log files that can be used for visualization and debugging purposes.

The simulation is carried out in a message-driven fashion. CD++ messages fall into two classes: *content messages* include the external message (X, t) and output message (Y, t) that encode the actual data transmitted between the models, while *control messages* include the initialization message (I, t), collect message (@, t), internal message (*, t), and done message (D, t) that are used internally by the simulator to control the simulation. Each message represents an event with an associated timestamp that indicates the simulated virtual time of the event.

The emergence of WS technologies has triggered a major paradigm shift in distributed computing. Various standards and techniques have been proposed to facilitate the construction of a new platform on which a set of network-accessible operations and their associated resources are abstracted as platform-independent machine-consumable services based on the Service Oriented Architecture (SOA) principles. To provide DEVS-based simulation services in Grid environment, a distributed simulator called DCD++ has been developed that uses a flexible wrapper to expose CD++ functionalities as web services based on standard WS technologies [8]. The backbone of the DCD++ simulator is the XML [11] and XML-Schema [12] techniques that encapsulate customized simulation data in a machine-processable format. The public interface of the simulation service is specified in WSDL [13], a general purpose XML-based language for describing web services, protocol bindings and other deployment details. The various partitions involved in the simulation communicate with each other by exchanging SOAP messages [14] that can be transmitted over different transport protocols.

Many DEVS-based toolkits have been developed for PADS based on different middleware technologies. However, few of them can achieve large-scale distributed simulation on a global scale in a platform-independent manner. A non-comprehensive list of existing toolkits is given below.

- DEVS/CORBA [15] is a runtime infrastructure based on CORBA middleware that supports distributed simulation of DEVS models. It can be embedded in a larger network-centric environment to provide a combination of graphical process modeling, discrete-event simulation, animation, activity-based costing, and optimization functions.
- DEVS/HLA [16] is an HLA-compliant M&S environment implemented in C++ that supports high-level model construction. It simplifies the programming effort required to establish and participate in an HLA federation.
- DEVSCluster [17] is a CORBA-based, multi-threaded distributed simulator implemented in Visual C++. It transforms a hierarchical DEVS model into a non-hierarchical one to ease the synchronization of the distributed simulation.
- DEVS/Grid [18] is an M&S framework implemented using Java and Globus for the Grid environment. It includes a set of fully automated simulation facilities, including cost-based hierarchical model partitioning, dynamic coupling restructuring, automatic model deployment, and M&S naming and directory service.
- DEVS/P2P [19] is a P-DEVS based M&S framework implemented on top of Peer-to-Peer communication infrastructure. It uses a customized DEVS simulation protocol to achieve decentralized inter-node communication. Simulators are synchronized by themselves without involving a coordinator.
- DEVS/RMI [20] is a DEVS-based system that provides a dynamic and re-configurable runtime infrastructure for handling load balancing and fault tolerance in distributed simulations. It reduces the overhead associated with common middleware solutions by using the native support of Java RMI to synchronize local and remote simulators.

3. WEB-BASED SIMULATION IN DCD++

The DCD++ simulator relies on the proper functioning of a web service wrapper that interacts with the CD++ simulation engine and exposes its functionality to remote web service clients. The wrapper consists of two main components, namely a *web service component* implemented in Java and a *simulation component* realized in C++. The former component deals with web service related activities such as user authentication, session management, and parsing simulation requests from the clients. It is deployed in an Axis SOAP engine [21], which in turn runs in an Apache Tomcat application server [22]. On the other hand, the latter component is responsible for accessing and manipulating the internal objects and data structures in the simulation engine. A separate workspace is created for each user session in the simulation component, reducing potential resource contention and allowing for multiple user sessions running independently with increased parallelism. Both components communicate with each other through message queues maintained in the Linux kernel. The communication is handled by a proxy object that is implemented as a shared C++ library plugged into the Java Virtual Machine (JVM) through the Java Native Interface (JNI) [23].

A skeleton of the DCD++ software architecture is shown in Figure 1. The proxy object creates two message queues in the Linux kernel for each user session to implement a bidirectional communication channel between the web service component and the corresponding

simulation component. Figure 2 gives a close look at the message queues maintained by the proxy.

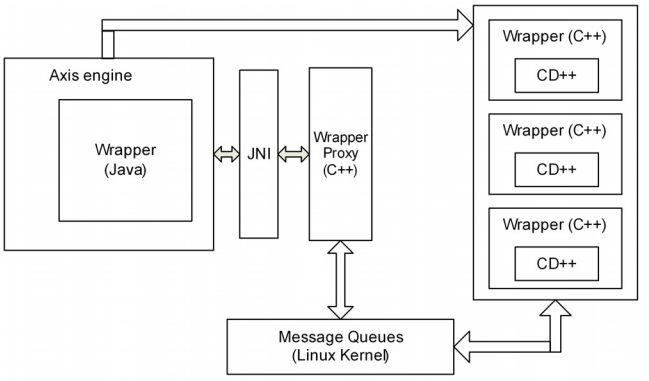


Figure 1. DCD++ software architecture [8]

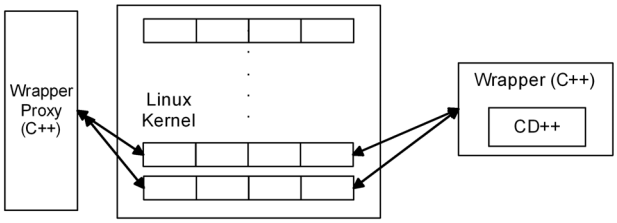


Figure 2. Message queues created for each user session [9]

Furthermore, the wrapper is multi-threaded to improve system performance. For each user session, the wrapper creates two Java threads as well as two Linux POSIX threads. On the web service component side, one Java thread responds to runtime client requests such as simulation monitoring operations, while the other thread records the client operations into a session log file. On the simulation component side, a POSIX thread executes the simulation engine, whereas the other listens on the message queue for events coming from the proxy object.

In this paper, we will not elaborate on the implementation of the simulation service. Interested readers can refer to [8] and [9] for more detailed discussion. Instead, we give a brief summary of the major functionalities provided by the DCD++ simulation service, which can be classified into four categories as follows.

1. Session management

- *User authentication*: verify user credentials against a password file stored locally on the server, and initialize a new session for each successful login.
- *Session logoff*: terminate the user session and reclaim the resources.

2. Configuration

- *setMAFile*: allow users to submit the model definition file for DEVS coupled models and Cell-DEVS models.
- *setDEVSModel*: allow users to submit the C++ header (.h) and implementation (.cpp) files for each DEVS atomic model.
- *setEventFile*: allow users to specify the external input events to be executed during the simulation.
- *setSupportFile*: allow users to configure other supporting files (e.g., the initial cell values in a Cell-DEVS model) that are required by the simulation engine.
- *setExecutionTime*: allow users to specify the end time of the simulation.
- *enableParingInfo*: turn on the simulator parameter to generate information for debugging purpose.

3. Monitoring and control

- *startSimulationService*: allow users to start the simulation process.
- *isSimRunning*: allow users to probe the current status of the simulation.
- *getCurrentSimulationTime*: allow users to monitor the progress of the simulation at runtime.
- *insertExternalEvent*: allow users to dynamically add additional external events to the simulation during runtime.
- *killSimulation*: allow users to terminate the simulation prematurely.

4. Logging and data retrieving

- *retrieveLogFile*: allow users to retrieve the log file generated during the simulation.
- *retrieveOutputFile*: allow users to retrieve the output file that contains the events sent from the simulated model to the surrounding environment.
- *retrieveParsingInfoFile*: allow users to retrieve the files that contain debugging information.
- *retrieveSessionLogFile*: allow users to retrieve the log file that contains information about the operations performed during the current session.

Figure 3 depicts a typical interaction between a WS client and the DCD++ simulation service via SOAP messages across the Internet.

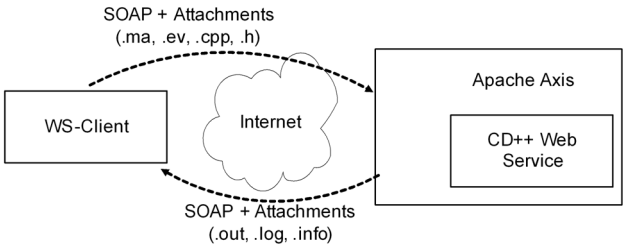


Figure 3. Invocation of DCD++ web service

First, the client retrieves the WSDL document that defines the interface of the simulation service in terms of execution parameters and return values. It can then invoke the web service methods through dynamically generated SOAP messages transmitted over HTTP protocol, which is one of the most widely used protocols on the Internet. The model definition and configuration files (e.g., the C++ header and implementation files, external event file, partition file, etc.) are transmitted to the simulation service as SOAP attachments. When the simulation finishes, the

client retrieves the simulation results (e.g., the output, log and debugging files) again as SOAP attachments. Since the SOAP attachments contain potentially very large documents, especially for the log files that record all the events executed during the simulation, the communication overhead constitutes a major bottleneck in the system, as we will see later in the performance analysis.

To reduce the communication overhead, DCD++ adopts a master/slave structure of coordinators [6]. As a result, when a coupled model is partitioned onto multiple nodes, a coordinator is created on each of them to execute the portion mapped on that specific node. The coordinator on the first node involved in the partition is the master, while all the other coordinators are slaves. The master coordinator is deemed as the immediate parent of the slaves residing on remote nodes. Figure 4 illustrates the master/slave structure on two nodes.

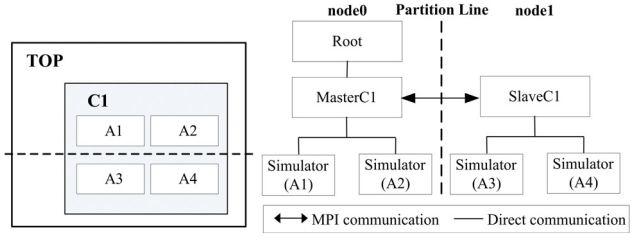


Figure 4. Master/slave coordinator structure in DCD++

Suppose the coupled model `C1` is partitioned onto two nodes and each portion has two atomic models. Two coordinators are created for `C1`: a master (`MasterC1`) on node0 and a slave (`SlaveC1`) on node1. One major advantage of this arrangement is that it can reduce the number of inter-node SOAP messages exchanged in the simulation. For example, if simulator `A3` sends a message to `A4`, then the slave coordinator `SlaveC1` can directly route the message to the local destination without incurring SOAP messages transmitted between the two nodes. Due to the high cost of SOAP messaging, this structure can significantly reduce the communication overhead and improve the simulation performance.

4. EXPERIMENTAL ENVIRONMENT

Along with the tremendous benefits WS technologies bring to the DCD++ distributed simulator, however, this approach also comes with extra overhead. As mentioned earlier, a major part of the overhead is devoted to the prohibitive communication cost associated with SOAP messaging between different compute nodes. Besides, the communication between the two components of the WS wrapper through Linux message queues contributes to this overhead as well. Figure 5 illustrates the composition of a communication path linking two remote compute nodes. The SOAP messages can be transmitted over both commodity Internet connections and high-speed point-to-point fiber links such as UCLP (User Controlled Light Path) [24].

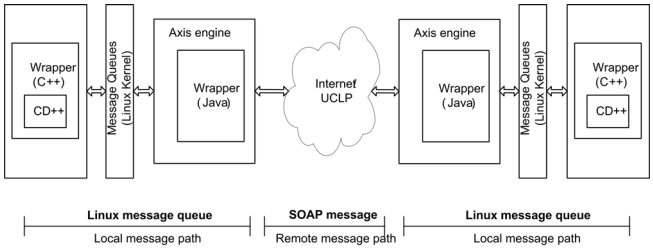


Figure 5. Communication path in DCD++ [8]

In order to test the performance of DCD++ simulation service on a global scale, we conducted experiments using different models executed on two machines across the Atlantic Ocean. One machine is located in the Advanced Real-Time Simulation (ARS) Laboratory at Carleton University, Ottawa, Canada; while the other located in the Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS) at Blaise Pascal University, Clermont-Ferrand, France. The machines were connected over commodity Internet connections involving 19 hops in the route with an average round-trip time (RRT) of 136 ms. Furthermore, the machines used in the experiments are not identical (Intel PIV CPU @ 3.2 GHz with 512 MB DRAM, and Intel PIV CPU @ 1.8 GHz with 256 MB DRAM respectively).

The communication infrastructure used in our experiments was provided by:

- CANARIE[1]: a Canadian non-profit collaboration between business and government to coordinate improved Internet access and network connectivity throughout Canada.
- RENATER[2]: the French educational and research network.

DANTE[3] plans, builds and operates advanced networks for research and education. It is owned by European NRENs (National Research and Education Networks), and works in partnership with them and in cooperation with the European Commission. DANTE provides the data communications infrastructure essential to the development of the global research community. DANTE ensures the connectivity between the Canadian and the French research networks.

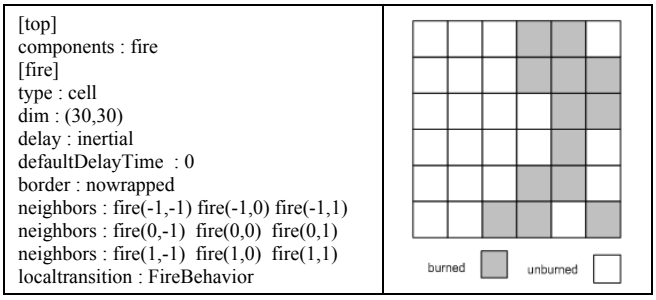
Two models were tested in the experiments, including a 30×30 Cell-DEVS model that simulates propagation of wildfire in forest [25], and a mix sand-pile model consisting of a DEVS particle generator and a 10×10 Cell-DEVS model representing the sand-pile formation [26]. Figure 6 and Figure 7 illustrate the model definition and partition scheme applied on two machines for these models respectively. Notice that we employed very simple partition schemes for the sake of convenience. Using different

[1] CANARIE: http://www.canarie.ca

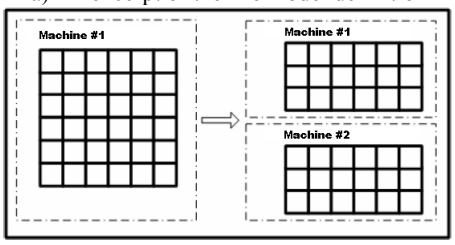
[2] RENATER: "REseau NAtional de télécommunications pour la Technologie l'Enseignement et la Recherche". http://www.renater.fr/

[3] DANTE: "Delivery of Advanced Network Technology to Europe". http://www.dante.net/

partition schemes may have a considerable impact on the communication pattern and thus simulation performance, which is an open topic for research in itself and is beyond the scope of this paper.

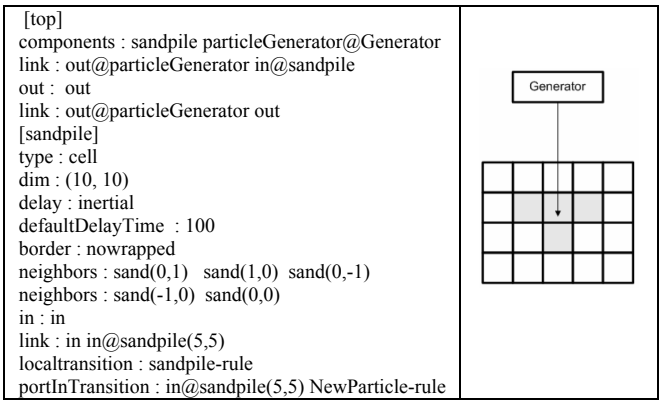


a) An excerpt of the fire model definition

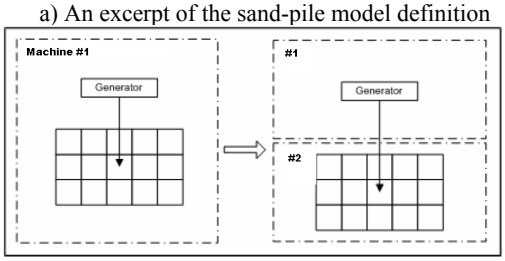


b) Partition of the fire model on two machines

Figure 6. The fire model and the partition scheme



a) An excerpt of the sand-pile model definition



b) Partition of the sand-pile model on two machines

Figure 7. The sand-pile model and the partition scheme

Each model was executed multiple times to achieve a confidence interval of 95%, and the standard deviations of the various metrics were calculated. We collected the following metrics during the experiments:

- The initialization time to start the web service;
- The simulation time to execute the model;
- The total execution time that is the sum of both initialization and simulation time;
- The average time to transmit a SOAP message between the two machines;
- The average time to pass a message through the Linux kernel message queues.

5. PERFORMANCE ANALYSIS

Two types of experiments were carried out in our study. First, the models were executed using a single machine, which allows us to examine the pure computation overhead without considering the communication cost. Then, the models were divided into two partitions as described in the previous section and the metrics were collected for the distributed simulation. Combining the date from these experiments allows us to gain more insight into the communication overhead. Table 1 shows the execution results for the fire model on one machine.

Table 1. Execution of the fire model on one machine

Ottawa Canada	***Avg.***	***Std. Dev.***	***C. I. 95%***
Init. Time (ms)	101.602	3.197	[100.200, 103.003]
Simulation Time (s)	2.891	0.011	[2.886, 2.896]
Total Exec. Time (s)	2.993	0.012	[2.987, 2.998]
Clermont-Ferrand France	***Avg.***	***Std. Dev.***	***C. I. 95%***
Init. Time (ms)	189.049	8.064	[185.515, 192.583]
Simulation Time (s)	4.177	0.528	[3.945, 4.408]
Total Exec. Time (s)	4.366	0.529	[4.134, 4.598]

As expected, the simulation is more than 200 times slower when the fire model is executed using the simulation services across the Atlantic Ocean, as shown in Table 2. This is primarily due to the communication overhead of SOAP messaging. A single transmission of SOAP message takes an average of 8.24 second, which is equivalent to about 60 times of the RRT. As we mentioned in Section 3, the SOAP attachments contain potentially large documents such as model definition and simulation log files with sizes far greater than the TCP window, resulting in multiple rounds of transmission for each SOAP message. When compared with the overhead of SOAP messaging, the communication cost of Linux kernel messaging is relatively minor. Furthermore, kernel messaging has a much smaller variance than SOAP messaging because the transmission time of SOAP messages depends on the current status of the ever-fluctuating Internet traffic.

Table 2. Execution of the fire model over Internet

Fire model	***Avg.***	***Std. Dev.***	***C. I. 95%***
Kernel Msg. (ms)	1.015	0.673	[0.720, 1.310]
SOAP Msg. (ms)	8240.111	1244.400	[7694.738, 785.484]
Init. Time (ms)	99.801	1.138	[99.303, 100.300]
Simulation Time (s)	895.274	84.420	[858.276, 932.272]
Total Exec. Time (s)	895.374	84.420	[858.376, 932.372]

The initialization time is also longer in the distributed simulation. Notice that even though only half of the model is initialized on each machine, the initialization still takes a comparable time as in the single-machine case, mainly because extra components need to be initialized in the simulation web service (e.g., the wrapper and proxy objects). However, this does not yet constitute a bottleneck in the simulation.

The sand-pile model was tested in the same way. The execution results on one machine are shown in Table 3.

Table 3. Execution of the sand-pile model on one machine

Ottawa Canada	***Avg.***	***Std. Dev.***	***C. I. 95%***
Init. Time (ms)	22.625	2.445	[21.553, 23.696]
Simulation Time (s)	0.066	0.002	[0.065, 0.067]
Total Exec. Time (s)	0.089	0.001	[0.088, 0.089]
Clermont-Ferrand France	***Avg.***	***Std. Dev.***	***C. I. 95%***
Init. Time (ms)	36.211	4.913	[34.058, 38.364]
Simulation Time (s)	0.119	0.003	[0.118, 0.121]
Total Exec. Time (s)	0.156	0.004	[0.154, 0.157]

As shown in Table 4, the simulation performance is now more than 1000 times worse when the model is executed over the Internet. Since the model is partitioned in a way that the DEVS particle generator and the Cell-DEVS model are located on remote machines, more SOAP messages are transmitted during the simulation to send the constantly generated events to the other side of the Atlantic Ocean. This illustrates the significant impact a partition scheme can have on the system performance. Again, it demonstrates that long-distance SOAP messaging is the primary bottleneck in the simulation system.

Table 4. Execution of the sand-pile model over Internet

Sand-pile model	***Avg.***	***Std. Dev.***	***C. I. 95%***
Kernel Msg. (ms)	1.522	3.023	[0.198, 2.847]
SOAP Msg. (ms)	7653.817	148.520	[7588.726, 718.907]
Init. Time (ms)	21.720	1.732	[20.961, 22.479]
Simulation Time (s)	132.714	0.956	[132.295, 133.133]
Total Exec. Time (s)	132.735	0.956	[132.316, 133.155]

The initialization time is smaller in this case since a much smaller cell space is used in the send-pile model. While it is trivial when compared with the long execution time in the distributed simulation, the initialization time is non-negligible in the single-machine simulation and constitutes a 23.21% of the total execution time.

6. CONCLUSION

This paper addresses the issue of distributed DEVS-based simulation using standard WS technologies in the Grid environment. The CD++ toolkit has been redesigned to expose the simulation functionalities as machine-consumable web services, allowing for transparent sharing of computing power, data, models, and experiments in heterogeneous environment on a global scale. Also, the simulation services can be easily integrated with other services such as visualization, network management, and geographic information services in a larger system.

We have conducted experiments to investigate the simulation performance across the Atlantic Ocean over commodity Internet connections. The execution results show that SOAP messaging constitutes a major bottleneck in the distributed simulation system. Based on the results of this case study, we identify the following areas as the focus of our future research:

- XML compression techniques for SOAP messages. The verbose textual format for structured simulation data and the bulky SOAP attachments results in excessive communication and processing overhead, degrading system performance to a certain extent. Various XML-aware compression algorithms have been proposed in the literature (e.g., [27], [28], and [29]), which will be investigated in the context of DEVS-based distribution simulation web services in our future research.
- New message aggregation mechanisms. Although using the master/slave coordinator structure reduces the number of inter-node SOAP messaging, we will consider employing additional aggregation mechanisms to transmit the content of multiple messages within one batch transmission to further reducing the communication overhead.
- Proxy replication techniques. In the DCD++ simulator, the proxy object is implemented as a shared library for handling all client requests. This architecture may lead to a software bottleneck when the load of the system surges. Although multi-threading has been used to improve responsiveness of the system, replication techniques might need to be used in order to share load among multiple proxy entities if the simulation service is to be used by many users on the global scale [30].
- CD++ middleware. One objective of our research is to allow the simulation service to be easily integrated with other services in larger systems. In this case, the simulation service also acts as a client that requests other services such as GIS, resulting in a layered architecture where the CD++ toolkit is utilized as a middleware software package.

References

[1] Zeigler, B.P.; H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press, San Diego, CA.

[2] Chow, A.C., and B.P. Zeigler. 1994. "Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism". In Proceedings of the 26th Winter Simulation Conference, Orlando, FL, 716-722.

[3] Wainer, G., and N. Giambiasi. 2002. "N-dimensional Cell-DEVS Models". Discrete Event Dynamic Systems, 12, no. 2, (April): 135-157.
[4] Wolfram, S. 2002. *A New Kind of Science.* Wolfram Media Inc., Champaign, IL.
[5] Wainer, G. 2002. "CD++: A Toolkit to Develop DEVS Models". Software: Practice and Experience, 32, no. 13, (September): 1261-1306.
[6] Troccoli, A. and G. Wainer. 2003. "Implementing parallel Cell-DEVS". In Proceedings of the 36th IEEE Annual Simulation Symposium (ANSS'03), Orlando, FL, 273-280.
[7] Liu, Q. and G. Wainer. 2007. "Parallel Environment for DEVS and Cell-DEVS Models". SIMULATION: Transactions of the Society for Modeling and Simulation International, 83, no. 6, (June): 449-471.
[8] Madhoun, R. and G. Wainer. 2007. "Studying the Impact of Web-Services Implementation of Distributed Simulation of DEVS and Cell-DEVS Models". In Proceedings of the 2007 DEVS Integrative M&S Symposium (DEVS'07), Norfolk, VA.
[9] Madhoun, R., B. Feng, and G. Wainer. 2007. "On the Creation of Distributed Simulation Web-Services in CD++". In Proceedings of the 14th AI, Simulation and Planning in High Autonomy Systems (AIS 2007), Buenos Aires, Argentina.
[10] Chow, A.C., B.P. Zeigler, and D.H. Kim. 1994. "Abstract Simulator for the Parallel DEVS Formalism". In Proceedings of the 5th IEEE Annual Conference on AI, Simulation and Planning in High Autonomy Systems, 157-163.
[11] Bray, T., et al. 2004. *Extensible Markup Language, XML 1.0 Third Edition*. http://www.w3.org/TR/2004/REC-xml-20040204/.
[12] Fallside, D.C. and P. Walmsley. 2004. *XML Schema: Primer 2nd Edition*. http://www.w3.org/TR/ xmlschema-0/.
[13] Christensen, E., et al. 2001. *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl.
[14] Mitra, N. and Y. Lafon. 2007. *SOAP Version 1.2: Primer 2nd Edition*. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.
[15] Zeigler, B.P., D. Kim, and S. Buckley. 1999. "Distributed Supply Chain Simulation in a DEVS/CORBA Execution Environment". In Proceedings of the 31st Winter Simulation Conference, Phoenix, AZ, 1333-1340.
[16] Zeigler, B.P., and H.S. Sarjoughian. 1999. "Support for Hierarchical Modular Component-based Model Construction in DEVS/HLA". In Proceedings of the 1999 Spring Simulation Interoperability Workshop, Orlando, FL.
[17] Kim, K. and W. Kang. 2004. "CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One". In Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2004), Assisi, Italy, LNCS 3046: 167-176.
[18] Seo, C., S. Park, B. Kim, S. Cheon, and B.P. Zeigler. 2004. "Implementation of Distributed High-performance DEVS Simulation Framework in the Grid Computing Environment". In Proceedings of the 2004 Advanced Simulation Technologies Conference – High-Performance Computing Symposium (ASTC'04), Arlington, VA.
[19] Cheon, S., C. Seo, S. Park, and B.P. Zeigler. 2004. "Design and Implementation of Distributed DEVS Simulation in a Peer to Peer Network System". In Proceedings of the 2004 Advanced Simulation Technologies Conference – Design, Analysis, and Simulation of Distributed Systems (ASTC'04), Arlington, VA.
[20] Zhang, M., B.P. Zeigler, and P. Hammonds. 2006. "DEVS/RMI – An Auto-adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies". In Proceedings of the 2006 DEVS Integrative M&S Symposium (DEVS'06), Huntsville, AL.
[21] Axis Development Team. 2007. *Axis User's Guide, Version 1.2*. http://ws.apache.org/axis/java/user-guide.html.
[22] Apache Software Foundation. 2007. *Apache Tomcat.* http://tomcat.apache.org/tomcat-6.0-doc/introduction.html.
[23] Liang, S. 1999. *The Java Native Interface – Programmer's Guide and Specification*. Addison-Wesley, Boston, MA.
[24] St Arnaud, B., J. Wu, and B. Kalali. 2003. "Customer Controlled and Managed Optical Networks". Journal of Lightwave Technology, 21, no. 11, (November): 2804-2810.
[25] Ameghino, J., A. Troccoli, and G. Wainer. 2001. "Models of Complex Physical Systems Using Cell-DEVS". In Proceedings of the 34th IEEE/SCS Annual Simulation Symposium (ANSS'01), Seattle, WA, 266-273.
[26] Saadawi, H. and G. Wainer. 2003. "Modeling a Sand Pile Application Using Cell-DEVS". In Proceedings of the 2003 Summer Computer Simulation Conference, Montreal, QC, Canada.
[27] Ericsson, M. 2007. "The Effects of XML Compression on SOAP". Journal World Wide Web, 10, no. 3, (September): 279-307.
[28] Lifke, H. and D. Suciu. 2000. "XMill: An Efficient Compressor for XML Data". In Proceedings of the 2000 ACM International Conference on Management of Data (SIGMOD), Dallas, TX, 153-164.
[29] Tolani, P. and J. R. Haritsa. 2002. "Xgrind: A Query-friendly XML Compressor". In Proceedings of the 18th International Conference on Data Engineering (ICDE), Washington, DC, 225-234.
[30] Lazowska, E. D. et al. 1984. *Quantitative System Performance – Computer System Analysis Using Queueing Network Models*. Prentice Hall, Upper Saddle River, NJ.