New Trends in Large Scale Distributed Systems Simulation

Ciprian Dobre*, Florin Pop*, Valentin Cristea*
*Faculty of Automatics and Computer Science, University Politehnica of Bucharest, Romania
E-mails: {ciprian.dobre, florin.pop, valentin.cristea}@cs.pub.ro

Abstract

In this paper we review current and previous work in the field of modeling and simulation of large scale distributed systems. We propose a new taxonomy to analyze the most representative large scale distributed systems simulators. The taxonomy considers the design characteristics of large scale distributed systems, and also the implementation properties of existing simulators. Based on the proposed taxonomy, we then analyze several simulators designed to model large scale distributed system, emphasize the pros and cons for each of the surveyed instruments, and present our approach in developing simulation tools for large scale distributed systems. Future trends are identified, which are targeted also by the authors in the projects under development.

Keywords: Distributed Systems, Grid Computing, Modeling and Simulation, Performance Analysis.

1. Introduction

Modeling and simulation were seen for long time as viable solutions to develop new algorithms and technologies and to enable the enhancement of large-scale distributed systems, where analytical validations are prohibited by the scale of the encountered problems. The use of discrete-event simulators in the design and development of large scale distributed systems is appealing due to their efficiency and scalability.

In this paper we define the categories of a taxonomy that best describes modeling instruments for Grid systems. The proposed taxonomy is focused on the simulation of Grid systems, hence it introduces categories such as motivation and Grid components that are particularly designed to better categorize the special family of Grid simulation instruments. The proposed taxonomy considers both the design characteristics of the Grid systems being modeled and the implementation properties of simulators designed for such systems.

The taxonomy includes a small and comprehensive set of categories, making it clear and easily understandable. In the same time, the chosen set of categories allows to differentiate between the most important Grid simulation tools, thus providing a broad view of the instruments that a Grid researcher can use. The taxonomy is useful as a methodological support for testing the differences between various simulation tools and in the same time serves as guidelines for developing new tools.

Starting from this taxonomy, we also analyse the most representative simulators for Grid systems. We detail their scope, implementations and applications. We use the categories of the proposed taxonomy to emphasize the pros and cons of each of the surveyed instruments. The analysis highlights their specific characteristics, from the adopted simulation models or internal simulation design to implementation details. In this way we demonstrate that simulation tools cover important aspects of distributed systems, allowing exploration of different areas of parameter space.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 presents the categories of the proposed taxonomy, while in Section 4 we use the taxonomy to analyze several modeling instruments for large scale distributed systems. Section 5 presents future trends in large scale distributed systems simulation. Section 6 is reserved for conclusions.

2. Related work

There are two types of works dealing with the analysis of simulators for large scale distributed systems. We don't consider papers published by various authors in which they highlight the quality of their tools in comparison with different existing simulation frameworks. We found that, in most cases, these papers tend to be misleading and subjective. We consider only those papers published for an objective analysis and compare existing simulation tools.

Sulistio et al. (2004) analyzed various parallel and distributed systems simulators, proposing a taxonomy to categorize between the various existing modeling frameworks. We state that the taxonomy is appropriate for describing any simulator, but is inappropriate to

completely describe simulators aiming at modeling Grid systems. The authors suggest that based on the purpose of the simulation one of the category according to which we can classify various simulator is the presence of physical time. This property indicates whether the simulation of a system encompasses the physical time factor. We argue that this category is obsolete in case of large scale distributed simulators. Time represents an inherent property in case of large scale distributed systems, and simulators designed for such systems always take this factor into consideration. For example, the networking traffic flow is an inherent time-based process, the job processing also.

Another property according to which the simulation tools can be classified is represented by the modeling framework. According to this the simulators are classified into entity-based and event-based. An entity-based modeling framework represents processes to be modeled as entities. Each entity performs its own tasks and communicates with other entities via messaging. In an event-based modeling framework, each task in a modeled process is activated via the arrival of some triggering events. We argue that in reality this category is inconclusive since all simulators that address Grid-related or P2P-related problems use both modeling frameworks. For example, the analyzed simulators use entities to simulate various components such as clusters, processing units, network elements, on which external tasks are running, yet they all use events to trigger the evolution of the simulated scenario being run.

In contrast with our proposed taxonomy, the one in Sulistio (2004) is not covering the scope behind various simulator implementations. Because of the inherent complex structure of the distributed systems many simulators are more focused on just particular areas. Some simulators were designed specifically for evaluating scheduling algorithms, while others target the validation of data replication strategies. The scope behind these simulation tools represents an important aspect that, in our opinion, should also be included in the taxonomy.

Another related paper is Quetier (2005). The authors address some of the problems that we too observed in Sulistio (2004) such as validation and the presenting of the advancements in simulators. However, the authors are not trying to compare the presented simulation tools. Rather, the paper presents only a survey of current trends in Grid simulation. The authors are not sketching any comparisons methods (except for a review of the validation studies carried out in case of the surveyed simulation tools) between the simulation tools.

3. A taxonomy of large scale distributed systems simulators

In this section we introduce the categories of the proposed taxonomy. We categorize the simulators for large based distributed systems based on the adopted simulation model, as well as based on their design and implementation.

According to the adopted simulation model a simulator can be classified according to the *scope*, and *supported model*. The *scope* category refers to the extent and nature of the simulated systems, to the components supported by the simulation model.

A simulator designed for large scale distributed systems can include the components necessary to model Intranet or Internet systems, Web, Grid, Cloud, Farm, or Cluster systems, P2P networks, distributed applications and other types of large scale distributed systems and applications. Many of existing simulators for large scale distributed systems were implemented to model only particular classes of problems. For example, some were implemented to simulate various scheduling algorithms; others were developed to simulate data transfer technologies. Large scale distributed systems are complex by their nature. Because of this reason the simulation models also suffer from various limitations in the adopted scope.

According to this classification a simulator can be used to study data movement optimizations, to investigate scheduling algorithms, to evaluate replication solutions or to study a particular model of a Grid or P2P system. This category considers only the upper most scope. If, for example, we consider the case of a simulator designed specifically to study various scheduling algorithms, we can observe that in order to study such algorithms the simulator must also provide additional support, such as simulated underlying networks or processing nodes. If the underlying Grid components are also simulated then a scientist, with proper mediations, could also evaluate various other scenarios such as different file replication algorithms (assuming the simulator includes the possibility to also simulate data warehouses). But generally such modifications require some amount of work, and, except for the cases when the original developers of a simulator redesigned it at some point to comprehend some different classes of Grid related problems, such developments are virtually non-existent. The majority of the Grid simulators were developed in the context of the validation of the Large Hadron Collider (LHC) experiments in CERN and their proposed running conditions. For this reason we often see as possible motivation the categories identified by the authors of the taxonomy described in Venugopal (2006): Data Transport, Data Replication or Scheduling issues.

This classification is also important because it classifies simulators according to their capabilities to simulate the layers of the distributed architecture. For that we classify simulators based on the components of a distributed system that are supported by the simulation model. There are four types of components: hosts, network, middleware and user applications. It is very

important to have a standardized, complete hierarchy of characteristics that are simulated for all four components of such systems. *Host characteristics* are used to describe the hosts within the distributed environments. Such hosts may contain computing, data storage, and other resources, grouped into single or distributed systems. In a simulation we are interested in the types of host resources capable of being modeled, their organization, as well as the characteristics that are considered for each type of resource. Examples of resource organization in simulation are the "central model" proposed by the Bricks project or the "tier model" proposed by the MONARC project (Dobre&Cristea, 2007). Of interest for this category are aspects such as how different simulators model the load of the computing nodes, the granularity of jobs being processed, or the types of data storage facilities.

The *Network characteristics* category describes the network elements interconnecting hosts within simulated distributed environments. Among network elements there are routers, switches and other devices. This category considers both the types of network devices and the communication protocols that are considered by the various simulation models. The infrastructure communication protocols refers to lower-level protocols such as TCP, UDP, etc. as well as higher-level application protocols such as FTP, NFS, etc.. This category also considers the granularity of the simulation. The simulation of the network can model in detail the flow of each packet through the network, a time consuming operation that leads to better output results, or it can model only the flows of packets going from one end to another in the network. The *middleware characteristics* are used to describe the middleware layer within the simulated environments. Amongst simulated middleware elements this category describes components such as schedulers and various security enforcement components. This category analyses how the middleware system schedules the jobs for execution inside a Grid system, for example. The *user applications* characteristics are used to describe the user applications within the large scale distributed environment.

In classifying different simulators based on their capability to model various distributed resources it is important to consider the ability to easily incorporate components dynamically defined during simulation runtime, for example by the user constructing the scenario experiment. This capability is crucial because it's almost impossible to provide the users with a complete set of predefined components to support all possible simulation scenarios. The vast majority of simulation tools provide this capability, but there are also exceptions (Bricks for example).

According to the *supported model* we can classify simulators based on their *behavior* and *time base*. The *behavior* category classifies simulators based on how the simulation proceeds. A *deterministic* simulation has no random events occurring, so repeating the same simulation will always return the same simulation results. In contrast, a *probabilistic* simulation has random events occurring. *Time base* specifies the values that the simulation time can contain. In a *discrete* simulation time has values within a finite range, but in a *continuous* simulation time has values within a finite range.

According to their implementation simulators for large scale distributed systems can be categorized based on several observations. Each simulator exploits a *simulation engine* to implement and execute the simulation model. According to their simulation engine simulators can be classified based on their *execution* implementation or based on their *mechanics*.

A simulator advances the simulation based on the *mechanics* defined in the simulation engine. The existing literature divides the types of possible mechanic designs into three categories: the *continuous*, *discrete-event* and *hybrid* categories. In a continuous simulation state changes occur continuously across time. In a discrete-event simulation (DES) state changes only occur at specific time intervals. Finally, a hybrid simulation comprises both continuous and discrete-event simulations. In case of modeling distributed systems the continuous category identifies the particular case of emulators. While there are a number of particular good emulator projects for large scale distributed systems (MicroGrid, Grid eXplorer, etc.), we focus our analysis presented in the next section on simulators.

We also can classify furthermore discrete-event simulation based on how simulation proceeds. A discrete-event simulation adopts a queuing system where queues of events wait to be activated. A *trace-driven* DES proceeds by reading in a set of events that are collected independently from another environment and are suitable for modeling a system that has executed before in another environment. A *time-driven* DES advances by fixed time increments and is useful for modeling events that occur at regular time intervals. An *event-driven* DES advances by irregular time increments and is useful for modeling events that may occur at any time. An event-driven DES is more efficient than a time-driven DES since it does not step through regular time intervals when no event occurs.

Based on their execution, we can classify simulators for large scale distributed systems in *centralized* and *distributed*. Here execution refers to the capability of the simulators to make use of the underlying architectures. The authors in Sulistio (2004) see two modes of execution: serial and parallel. We argue that a better taxonomy should classify the simulators into centralized and distributed. Shared-memory parallel systems, comprising multiple processors, are becoming more and more accessible as home computing stations. In fact, a pure serial simulation execution, which would make use of only a single processor, can not be a reality when

addressing the problem of simulating large scale distributed systems, which are highly complex and in which multiple tasks/jobs are inherently being simultaneous processed. Modern simulators make use of at least the threading mechanisms provided by the underlying operating system; they use every processor existing on the underlying computing station. The idea of addressing instead of parallel simulation the term distributed simulation comes back from the early (Misra, 1986) paper.

In this classification the simulators designed to use only a single computing unit, no matter if the underlying processing architecture provides multiple processing cores, are called centralized simulators. The second category comprises the simulators designed to make use of multiple processor units, running on different architectures and dispersed around a larger area. There are no pure distributed simulators for modeling large scale distributed systems. The reason for this is that, despite over two decades of research, the technology of distributed simulations has not significantly impressed the general simulation community (Fujimoto, 1993). Considerable efforts and expertise are still required to develop efficient simulation programs.

Of course, simulators also different between each others on aspects such as the queuing structures adopted in the design of the simulation engine for managing the event lists or the mapping of simulation jobs on physical threads or processes. These aspects, considered by the designers of the simulators, are important because they greatly influence the performance runtime and the capability to model systems consisting of many resources. A system using an O(1) structure for the event list will behave better than another one using an O(log n) queuing structure. The time needed to run a complex simulation experiment can be quite huge when using the second queuing structure solutions, but the implementation of queuing solutions from the first category can take quite long and requires more time researching valuable results. Finding the best suitable queuing structure to be used for the simulation of large scale systems still represents a hot subject today. There is not a single unanimity accepted queuing structure that performs best when modeling distributed systems, they all tend to behave different depending on various parameters.

One other aspect to consider is the mapping of the simulation jobs on the underlying threads or processes. Reusing threads, using advanced mapping schemes in which multiple jobs can be simulated running in the same thread context, or any other aspect considered in this direction can yield higher simulation performances. This category considers the optimizations adopted in the design of the simulation engine to either improve the running performances or allow for advanced simulation models to be executed.

According to the *simulation model specification* simulators for large scale distributed systems can incorporate *specialized languages*, *general programming language and specialized library routines* or some other *visual* components (such as, for example, specific visual model components used in a drug-and-drop style for model construction).

A good design environment facilitates easy learning and fast usage. A language provides a set of defined constructs for the user to design simulation models. A library provides a set of routines to be used with a supporting programming language. A library-based simulation tool normally gives the user more flexibility in creating and controlling the simulation. An experienced user of the supporting programming language may fine-tune and optimize the simulation by exploiting certain libraries. A language-based simulation tool usually hides low-level implementation details from the user and thus provides less flexibility. Therefore, a language-based tool needs to provide a complete set of well-known constructs to ensure it supports the required level of flexibility. On the other hand, a language-based tool is often easier to learn and use since it is more high-level compared with a library-based tool.

Based on their *input data*, simulators can be further classified as including *input data generators* or as accepting *data sets collected by monitoring.* For example, MONARC 2 accepts both types of input (the monitoring data format is the one produced by MonALISA), while ChicagoSim accepts only input data generators.

The *user interface* determines how the user interacts with the simulator. Accordingly, we can classify simulators as having a *textual* or a *graphical output.* A visual user interface is preferred over a non-visual interface because graphical displays enable better interaction and they are easier to use and understand. A visual design interface allows the user to create a simulation model easier and faster compared with a non-visual interface, but the simulator that do provide this facility generally are restricting the types of simulation components that can be inserted in the modeling scene. Using a design interface the user can build the simulation model by dragging and dropping simulation objects and configuring the attributes and values (using forms for example). In contrast, a typical non-visual design interface requires the user to write programming code which requires more time and effort, but also extendibility support for the model. Examples of simulators providing visual design interfaces are GridSim and MONARC 2.

A visual execution interface provides a better representation of the simulation process. The user can more easily observe and analyze the simulation experiment. *Animations* provide a good visualization and display the flow of the simulation. Graphs give the graphical version of statistical data captured from the

simulation. Without a visual execution interface the user encounters difficulties in analyzing and understanding the simulation results based on huge amounts of statistics and events captured. The visual interface can also include *interactivity* features, such as allowing the user to stop, suspend, resume, restart, change parameters or query the results database while the simulation is running.

The visual output analyzer is probably the most important graphical tool a simulator could have. Generally a simulation generates huge amounts of data. The data is difficult to be analyzed using a pure text format. Based on their visual analyzing support, there are two categories to classify simulators. The *plots* are the usual instruments used to represent the output data of the simulation in a graphical format that is more accessible to the end-user. Some tools provide high-level capabilities, being able to not only represent the data but also to analyze it and provide it in a modified and more meaningful way to the end-user. This category includes instruments such as 2D plots (bar graphs, scatter plots, contour maps) and 3D plots (such as surface rendering). The second category includes *analysis* of the original output results of the simulation, with possible comparison between different sets of results, often from different simulation runs. A simulation instrument that offers more visual capabilities to the end-user to better analyze the results of the simulation scenario is generally preferred by scientists.

Simulators can also be classified based on their capability to offer validation results. This classification refers to the process of assuring that the conceptual model accurately represents the behavior of a real system. According to this classification we can distinguish between simulators that provide appropriate validation tests or not. For those simulators providing validation proofs, we can further differentiate simulators as proving validation results based on mathematical comparison or based on comparison between simulation model and real-world testbed systems. Validation in this case represents a measure of the reliability offered to the end-user running different modeling scenarios. Validation is essentially a statistical problem because the number of performable experiments is limited and in general the magnitude of tolerable errors depends on the type of obtained results. To this date only a few simulators present validation studies (e.g. Bricks, MONARC and SimGrid).

4. A critical analysis of simulation tools for large scale systems

Using the categories of the proposed taxonomy, we present an analysis of the properties of six representative simulators for large scale distributed systems. This study presents an evaluation of most relevant work in the domain of distributed systems simulation, and tests the capabilities of the presented taxonomy to correctly investigate properties of the analyzed simulators.

Bricks was among the first simulation projects developed to investigate different resource scheduling issues using modeling techniques. The Bricks simulation framework allows the simulation of various behaviors: resource scheduling algorithms, programming modules for scheduling, network topology of clients and servers in global computing systems, and processing schemes for networks and servers. In its latest versions Bricks was extended, in order to evaluate the performance of various Data Grid application scenarios, with replica and disk management simulation capabilities. Bricks uses a model which the authors call the “central model”. In this simulation model it is assumed that all the jobs are processed at a single site. In contrast with the model, MONARC also proposed another simulation model, called the “tier model”, in which jobs are processed according to their hierarchical levels.

OptorSim is a Data Grid simulator project initially developed by a team of researchers working on the WorkPackage 2 of the European DataGrid project, which was responsible for replica management and optimization, and the emphasis is on this area. The objective of OptorSim is to investigate the stability and transient behavior of replication optimization methods. OptorSim adopts a Grid structure based on a simplification of the architecture proposed by the EU DataGrid project. According to this model the Grid consists of several sites, each of which may provide resources for submitted jobs. Given a Grid topology and resources, a set of jobs to be executed and an optimization strategy as input, OptorSim runs a number of Grid jobs on the simulated Grid. It provides a set of measurements which can be used to quantify the effectiveness of the optimization strategy under the considered conditions.

SimGrid is a simulation toolkit that provides core functionalities for the evaluation of scheduling algorithms in distributed applications in a heterogeneous, computational distributed environment. SimGrid aims at providing the right model and level of abstraction for studying scheduling algorithms and generates correct and accurate simulation results. In its current form SimGrid can be used to simulate a single or multiple scheduling entities and timeshared systems operating in a Grid computing environment or to simulate distributed applications in the context of resource scheduling. SimGrid describes scheduling algorithms in terms of agent entities that make scheduling decisions. These agents interact by sending and receiving events via communication channels. SimGrid can be used to simulate compile time and running scheduling algorithms. In the first category, all scheduling decisions are taken before the execution. In the second category some decision are taken during the execution. In accordance with out proposed taxonomy,

SimGrid does not provide any of the system support facilities as discussed in the taxonomy. A validation of SimGrid was presented in its very first paper (Casanova, 2001). The validation consisted in comparing the results of the simulator with the ones obtained analytically on a mathematically tractable scheduling problem.

GridSim is a simulator developed by researchers from the Gridbus project to investigate effective resource allocation techniques based on computational economy. It allows simulation of entities in parallel and distributed computing systems-users, applications, resources, and resource brokers (schedulers) for design and evaluation of scheduling problems. It provides a comprehensive facility for creating different classes of heterogeneous resources that can be aggregated using resource brokers for solving compute and data intensive applications. GridSim supports modeling of heterogeneous computing resources (both time and space shared) from individual PCs to clusters, and various application domains from biomedical science to high energy physics. The focus is very much on scheduling and resource brokering. The GridSim toolkit can be used for modeling and simulation of application scheduling on various classes of parallel and distributed computing systems such as clusters, Grids, and P2P networks. GridSim focuses on Grid economy, where the scheduling involves the notions of producers (resource owners), consumers (end-users) and brokers discovering and allocating resources to users. Its design considers the existence of several brokers, which in SimGrid was introduced only since SimGrid2 (the Agents). GridSim is mainly used to study cost-time optimization algorithms for scheduling task farming applications on heterogeneous Grids, considering economy based distributed resource management, dealing with deadline and budget constraints. In some sense, GridSim is a higher-level simulator compared with SimGrid, which is basically designed to investigate interactions and interferences between scheduling decisions taken by distributed brokers.

ChicagoSim, developed by the University of Chicago, is a simulator developed by a team of researchers from the University of Chicago designed to investigate scheduling strategies in conjunction with data location. It is a modular and extensible discrete event Data Grid simulator built on top of the C-based simulation language Parsec. It is designed to investigate scheduling strategies in conjunction with data location. Its architecture includes a configurable number of schedulers rather than one Resource Broker, for example. It also allows for data replication but with a “push” model in which, when a site contains a popular data file, it will replicate it to remote sites, rather than the “pull” model used in OptorSim. A distributed system in ChicagoSim is modeled as a collection of sites. Each site has a certain number of processors of equal capacity and limited storage.

The final simulator analyzed is MONARC 2. Its simulation model is based on the characteristics of the LHC physics experiments, and is organized in the form of a hierarchy of different sites that are grouped into levels called tiers, mostly based on their resources. MONARC 2 is built based on a process oriented approach for discrete event simulation, which is well suited to describe concurrent running programs, network traffic as well as all the stochastic arrival patterns, specific for such type of simulation. Threaded objects or "Active Objects" (having an execution thread, program counter, stack...) allow a natural way to map the specific behavior of distributed data processing into the simulation program. In order to provide a realistic simulation, all the components of the system and their interactions were abstracted. The chosen model is equivalent to the simulated system in all the important aspects. A first set of components was created for describing the physical resources of the distributed system under simulation. The largest one is the regional center, which contains a farm of processing nodes (CPU units), database servers and mass storage units, as well as one or more local and wide area networks. Another set of components model the behavior of the applications and their interaction with users. Such components are the “Users” or “Activity” objects which are used to generate data processing jobs based on different scenarios. The job is another basic component, simulated with the aid of an active object, and scheduled for execution on a CPU unit by a “Job Scheduler” object.

Table1. Design comparison of surveyed Grid simulation projects.

No.	Simulation tool	Scope	Time base	Simulated components
1	**Bricks**	Resource scheduling in Grid systems	Discrete	Client-Server components organized in a central model; Servers and networking elements modeled as queuing systems; Scheduling Unit as the central simulation component.
2	**OptorSim**	Resource scheduling; Data replication strategies	Discrete	Grid sites composed of Computing Elements and Storage Elements; Computing Elements run one job at a time; Complex network model but lack routing, data transport, packetization; GSs with modeled Resource Broker, Replica Manager and Replica Optimiser.
3	**SimGrid**	Resource scheduling	Discrete	Scheduling tasks; Resource objects: modeled hosts and network links; Grid model can be obtained from traces (ENV and NWS are supported).
4	**GridSim**	Resource scheduling; Simplistic data replication	Discrete	The modeling systems is composed of users, brokers and resources; Both Computational and Data Grids are supported by the simulation model; Networking model takes into consideration QoS, background traffic; Well suited for algorithms designed for Nimrod-G?
5	**EDGSIM**	Resource Scheduling	Continuous	Jobs submitted using appropriate User Interface; Resource Broker programmed with various Scheduling algorithms; Replica Catalog mapping logical and physical file names; Compute Element models the computation Resource of the Grid; The network model is simple, without considering low-level functionality.
6	**ChicagoSim**	Resource scheduling; Data replication strategies	Discrete	Three modeled components: the site, the network and the driver; Replica management is carried out at local level; The modeled Grid includes any number of external schedulers, whilst the managing of the files is done locally by a dataset scheduler; Support for modeling various scheduling algorithms and various replica methods.
7	**MONARC 2**	Generic Grid simulator	Discrete	Processing units, Data Storage, farms, networking, HEP components; Support for the modeling of scheduling algorithms, replica management, networking procedures, etc. Strong support for modeling generic Grid architectures.

No.	Simulation tool	Execution	User interface	Validation
1	**Bricks**	Centralized, event driven	Textual output, designed to be used with external tools	Performed by replacing the Predictor with NWS
2	**OptorSim**	Centralized, event and time driven	Graphical user interface, animations	Degenerate tests, fixed values, internal validity
3	**SimGrid**	Centralized, event driven	Graphical user interface, result analysis capabilities	Fixed values
4	**GridSim**	Centralized and distributed, event driven	Graphical user interface built on top of the simulator	N/A
5	**EDGSIM**	Centralized	Graphical user interface, drag-drop capability to construct scenarios	N/A
6	**ChicagoSim**	Centralized, event driven	Textual output, designed to be used with external tools	N/A
7	**MONARC 2**	Centralized, event driven	Graphical user interface, animations	Model Validation Monitored resource utilization vs simulated observations Queuing theory tests

The critical analysis of the simulators reveals the motivations, principles, implementations and applications of the instruments. The analysis (see Table 1), based on the categories of the taxonomy presented in the previous section, describes the differences between the tools in terms of modeling, implementation and design. The evaluation of the analyzed simulators was mainly based on three criteria: (1) the ability to handle basic Grid functionalities; (2) the ability to schedule compute- and/or data-intensive jobs; and (3) the underlying network infrastructure.

The analysis highlights the specific characteristics of each analyzed simulator, from the simulation model to the internal properties to its implementation. There are advantages and disadvantages with each of the simulators. Interesting enough, even if many of them attack similar problems, being driven by comparable motivations, the simulators give a complementary approach to each others, allowing exploration of different areas of parameter space. Although the use of a particular simulator depends very much on the scope of the simulation being conducted and the skills of the user, they all cover important aspects of distributed systems, allowing exploration of different areas of parameter space.

5. Future trends

Large scale distributed systems are today regarded as the solution to developing increasingly large computing applications, designed to answer many of the problems of humanity. Commercially, large scale distributed systems are also becoming more and more appealing and this is reflected in the increasingly interest in the development of such systems coming from major industry players.

The development of solutions designed for large scale distributed systems, either applications running on top of them or technologies designed to help them, is facilitated by the use of adequate simulation instruments. However, today many of the simulators existing today are too focused on specific technologies, lacking the capability to model generic distributed systems. They do not include all the components and characteristics specific to such systems, leaving the user with the problem of implementing its own solutions on top simulation model. This translates into time and effort and is a reason why many prefer to implement a newly designed technology directly in real-world and evaluate its behavior at runtime.

In the future we believe this lack of generality in simulation model will be increasingly reduced, as designers start to invest more effort into providing more complete modeling solutions. Simulators such as MONARC 2 and ChicagoSim already started this trend. Users already see the potential of such simulators. As a consequence, MONARC 2 was already used to evaluate the specific behavior of the LHC experiments (Legrand, *et al*, 2005), providing valuable information without the need to implement the system in real-world. The experiment tested the behavior of the Tier architecture envisioned by the two largest LHC experiments, CMS and ATLAS. The obtained results indicated the role of using a data replication agent for the intelligent transferring of the produced data. The obtained results also showed that the existing capacity of 2.5 Gbps was not sufficient and, in fact, not far afterwards the link was upgraded to a current 30 Gbps. Other simulators, such as GridSim, SimGrid, OptorSim and many others also are well underway to extend their simulation models to be more generically and address a wider set of possible simulation experiments.

Another problem with existing simulators for large scale distributed systems consists in the lack of evaluation results. A scientist wanting to use a simulator to evaluate a specific technology needs to have increased confidence in the obtained results. He needs evidence that the obtained results also are valid in real-world. Many of the existing simulators designed for large scale distributed systems do not present confidence because they lack proof of their validity. This is due to the nature of the large scale distributed systems, for which

analytical models to be compared against the simulation models are hard to design. However, evaluation proof can be obtained in several ways. For example, a well-design simulator must present comparisons between experiments modeling small distributed systems against equivalent real-world testbeds. The comparison between the results obtained in simulation experiments and the monitored parameters of a real-world testbed should be made at least for the networking protocols, for the computing nodes and the storage facilities. If this simplified form of evaluation is conducted for each of the simulated component a general conclusion can be drawn, with higher confidence, for the entire simulation model. Another mechanism designed to facilitate the evaluation of the simulation models consists in the use of queuing theory. The formalism provided by the queuing models is important for the definition and validation of the simulation stochastic models. They provide an analytical model to the problem of testing the randomness introduced by various mathematical distributions. For example, in the simulation of network traffic pattern, queuing models are generally used to describe traffic generation, flows of the transmission and many intrusive problems related with the communication systems.

Another trend relates to the need to model very large distributed systems, consisting of a great number of resources. Many of today's simulators lack the capability to simulate large distributed systems because their simulation engines are limited to the physical resources of the workstations where the experiments are being executed. Today many researchers are interesting in finding solutions to facilitate the simulation of large scale distributed systems. The simulation engine can be optimized, in order to facilitate the evaluation of large scale distributed systems experiments, by using advanced priority queuing structures for the simulation events, by optimizing the way in which simulated entities are being scheduled in simulation for execution, by using various simplifications mechanisms or by using the underlying physical distributed resources of clusters of nodes.

With the advent of large scale distributed systems, today more than ever scientists are looking into simulation as the possible today to answer faster many of the faced problems. However, in order to be useful, a simulator must include solutions to be generic, to present evaluation capabilities and allow scalability.

6. Conclusions

In this we presented a comparison study of the most important simulation projects involved in the modeling of distributed systems. We showed how they relate and differ, their advantages and disadvantages. We demonstrated that, although the use of a particular simulation instrument depends very much on the scope of the simulation being conducted and the skills of the user, they all cover important aspects of distributed systems, allowing exploration of different areas of parameter space.

The critical analysis is based on the categories of our proposed taxonomy for comparing simulators for large scale distributed systems. We needed a standardized taxonomic system and, after analyzing the existing work on this subject, we concluded that all existing taxonomies are either too generic or do not consider important characteristics, therefore are not able to sustain a correct comparison. The proposed taxonomy is particularly focused on the simulation of such systems, hence it introduces categories such as motivation and specific components that are particularly designed to proper categorize the special family of Grid and/or P2P simulation instruments.

7. References

[1] A. Sulistio, C. S. Yeo, R. Buyya, "*A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools*", Software – Practice and Experience, 2004.

[2] B. Quetier, F. Capello, "*A survey of Grid research tools: simulators, emulators and real life platforms*", IMACS Survey, 2005.

[3] S. Venugopal, R. Buyya, K. Ramamohanaro, "*A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing*", ACM Computing Surveys, 38(1), 1-53, 2006.

[4] C. Dobre, V. Cristea, "*A Simulation Model for Large Scale Distributed Systems*", In Proc. of the 4th International Conference on Innovations in Information Technology, Dubai, November 2007.

[5] J. Misra, "*Distributed discrete-event simulation*", ACM Computing Survey, 18(1), 39-65, 1986.

[6] R.M. Fujimoto, "*Parallel discrete event simulation: Will the field survive?*", ORSA Journal on Computing, 5(3), 213-230, 1993.

[7] H. Casanova, "*Simgrid: a toolkit for the simulation of application scheduling*", The IEEE Intl. Symp. on Cluster Computing and the Grid (CCGrid'01), pp. 430-437, IEEE/ACM, 2001.

[8] I.C. Legrand, C. Dobre, R. Voicu, C. Stratan, C. Cirstoiu, L. Musat, "*A Simulation Study for T0/T1 Data Replication and Production Activities*", The 15th International Conference on Control Systems and Computer Science, 2005.