A Simulation Framework for Dependable Distributed Systems

Ciprian Dobre Florin Pop Valentin Cristea

Faculty of Automatics and Computer Science, University "*Politehnica*" of Bucharest, Romania
Emails: {ciprian.dobre, florin.pop, valentin.cristea}@cs.pub.ro

Abstract

The use of discrete-event simulators in the design and development of distributed systems is appealing due to their efficiency and scalability. Their core abstractions of process and event map neatly to the components and interactions of modern-day distributed systems and allow designing realistic simulation scenarios. MONARC, a multi-threaded, process oriented simulation framework designed for modeling large scale distributed systems, allows the realistic simulation of a wide-range of distributed system technologies, with respect to their specific components and characteristics. In this paper we present an innovative solution to the problem of evaluating the dependability characteristic of distributed systems. Our solution is based on several proposed extensions to the simulation model of the MONARC simulation framework. These extensions refer to fault tolerance and system orchestration mechanisms being added in order to asses the reliability and availability of distributed systems. The extended simulation model includes the necessary components to describe various actual failure situations and provides the mechanisms to evaluate different strategies for replication and redundancy procedures, as well as security enforcement mechanisms.

Keywords

Dependable Distributed Systems, Grid Computing, Fault tolerance, Simulation, Performance Analysis.

1. Introduction

Nowadays there is an increasing interest in large scale distributed systems, in both academic and industrial environments. Today more than ever distributed systems represent the preferred instruments for developing a wide range of new applications. Due to the offered technological opportunities, the domains of usage of Grid computing in particular have been extending during the past years from scientific to commercial applications.

Together with the extension of the application domains, new requirements have emerged for large scale distributed systems; among these requirements, reliability, safety, availability, security and maintainability, in other words dependability [1], are needed by more and more modern distributed applications, not only by the critical ones. However, building dependable distributed systems is one of the most challenging research activities. It considers several aspects like control of complexity, early defect detection, rigorous translation, ease of simulation, model checking, productivity gains for teams, wide applicability [1]. It means collecting informal requirements, requirements translation, build requirement behavior trees, making requirements integration and integrated behaviour tree, and then make simulation, verification and implementation [2].

In this paper we present a solution to evaluating the correctness and performance of various dependability-related technologies for distributed systems using the formalism provided by the modeling and simulation domain. Simulation is a powerful method to perform system analysis, even when the involved system does not physically exist or if it is very expensive to use it. Our proposed solution is based on an extension proposal of the simulation model being provided by MONARC, a generic simulation framework designed for modeling large scale distributed systems. The rest of this paper is structured as follows. Section 2 presents related work to the problem of analyzing dependable distributed systems using simulation. The next section presents the MONARC architecture and the simulation model. Sections 4 and 5 present the proposed solutions for simulating dependable distributed systems. Finally, in Section 6 we present some conclusions and future work.

2. Related Work

SimGrid [7] is a simulation toolkit that provides core functionalities for the evaluation of scheduling algorithms in distributed applications in a heterogeneous, computational Grid environment. It aims at providing the right model and level of abstraction for studying Grid-based

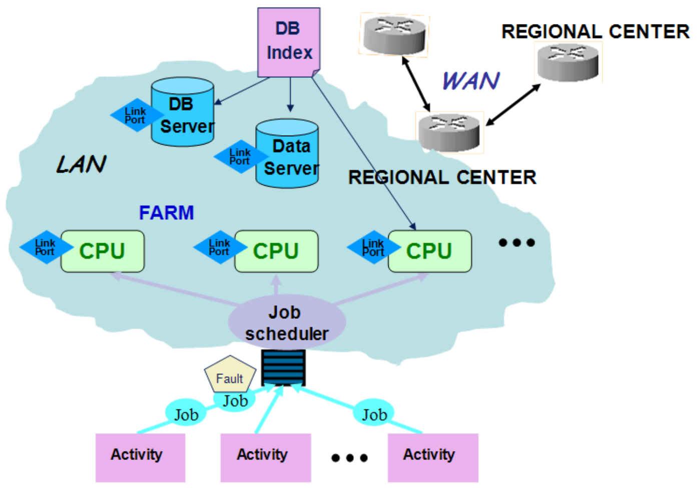

Figure 1. The Regional center model.

scheduling algorithms and generates correct and accurate simulation results. GridSim [8] is a grid simulation toolkit developed to investigate effective resource allocation techniques based on computational economy. OptorSim [9] is a Data Grid simulator project designed specifically for testing various optimization technologies to access data in Grid environments. OptorSim adopts a Grid structure based on a simplification of the architecture proposed by the EU DataGrid project. ChicagoSim [10] is a simulator designed to investigate scheduling strategies in conjunction with data location. It is designed to investigate scheduling strategies in conjunction with data location.

None of these projects present general solutions to modeling dependability technologies for large scale distributed systems. They tend to focus on providing evaluation methods for the traditional research in this domain, which up until recently targeted the development of functional infrastructures. However, lately, the importance of dependable distributed systems was widely recognized and this is demonstrated by the large number of research projects initiated in this domain. Our solution aims to provide the means to evaluate a wide-range of solutions for dependability in case of large scale distributed systems.

Another issue is related to the generic evaluation of dependable distributed systems. A fault occurring in such systems could lead to abnormal behavior of any of the system's components. For this reason we argue that a correct evaluation of dependability in distributed systems should provide a complete state of the entire distributed system. Because of the complexity of the Grid systems, involving many resources and many jobs being concurrently executed in heterogeneous environments, there are not many simulation tools to address the general problem of Grid computing. The simulation instruments tend to narrow the range of simulation scenarios to specific subjects, such as scheduling or data replication.

The simulation model provided by MONARC is more generic that others, as demonstrated in [11]. It is able to describe various actual distributed system technologies, and provides the mechanisms to describe concurrent network traffic, to evaluate different strategies in data replication, and to analyze job scheduling procedures.

2.1. MONARC Architecture

MONARC 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 simulations [3]. 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. However, due to various optimizations considered, the threaded implementation of the simulator can be used to experiment with scenarios consisting of thousands of processing nodes executing a large number of concurrent jobs, as demonstrated in [12].

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 (Figure 1), which contains a site 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. Any regional center can dynamically instantiate a set of users or activity objects, which are used to generate data processing jobs based on different simulation scenarios. Inside a regional center different job scheduling policies may be used to distribute jobs to corresponding processing nodes. One of the strengths of MONARC is that it can be easily extended, even by users, and this is made possible by its layered structure. The first two layers contain the core of the simulator (called the "simulation engine") and models for the basic components of a distributed system (CPU units, jobs, databases, networks, job schedulers etc.); these are the fixed parts on top of which some particular components (specific for the simulated systems) can be built. The particular components can be different types of jobs, job schedulers with specific scheduling algorithms or database servers that support data replication. The diagram in Figure 2 presents the MONARC layers and the way they interact with a monitoring system. In fact, one other advantage that MONARC have over other existing simulation instruments covering the same domain is that the modeling experiments can use real-world data collected by a monitoring instrument such as MonALISA. This is useful for example when designing experiments that are meant to experiment new conditions starting from existing real distributed infrastructures.

Using this structure it is possible to build a wide range of models, from the very centralized to the distributed system models, with an almost arbitrary level of complexity (multiple regional centers, each with different hardware configuration and possibly different sets of replicated data). The maturity of the simulation model was demonstrated in previous work. For example, a number of data replications experiments were conducted in [3], presenting important results for the future LHC experiments, which will produce more than 1 PB of data per experiment and year, data that needs to be then processed. A series of scheduling simulation experiments were presented in [3], [4] and [5]. In [5] we successfully evaluated a simple distributed scheduler algorithm, proving the optimal values for the network bandwidth or for the number of CPUs per regional centre. In [4] the simulation model was used to conduct a series of simulation experiments to compare a number of different scheduling algorithms.

Probably the most extensive simulation scenario is the one described in [6]. The experiment tested the behavior of the tier architecture envisioned by the two largest LHC experiments, CMS and ATLAS. The simulation study described several major activities, concentrating on the data transfer on WAN between the T0 at CERN and a number of several T1 regional centers.

The experiment simulated a number of physics data production specific activities (the RAW data production, Production and DST distribution, the re-production and the new DST distribution and the detector analysis activity). We simulated the described activities alone and then combined. 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.

2.2. Simulation of Failure Models in MONARC

The characteristics of large scale distributed systems make the problem of assuring dependability a difficult issue because of several aspects. A first aspect is the geographical distribution of resources and users that implies frequent remote operations and data transfers; these lead to a decrease in the system's safety and reliability and make it more vulnerable from the security point of view. Another problem is the volatility of the resources, which are usually available only for limited periods of time; the system must ensure the correct and complete execution of the applications even in situations such as when the resources are introduced and removed dynamically, or when they are damaged.

The management of distributed systems is also complicated by the constraints that the applications and owners of the resources impose; in many cases there are conflicts between these constraints - for example, an application needs a long execution time and performs database operations, while the owner of the machine on which the application could run only makes it available in a restricted time interval and does not allow database operations. Solving these issues still represents a research domain and, although numerous projects have obtained significant results, no complete solution has yet been found to integrate all requirements involved in obtaining a dependable system. The means to achieve dependability are fault prevention, fault tolerance, fault removal and fault forecasting [1]. Among these fault tolerance is the most difficult to achieve because of the complexity of the distributed systems. We consider that the simulation model of MONARC is adequate to be augmented as presented in this paper to testing various methods, techniques and technologies related to the different phases of fault tolerance achievement in distributed systems: error and fault detection, recovery and fault masking (redundancy) [2]. Figure 3 presents the components of the proposed dependable modeling layer.

The first extension to the simulation model relates to modeling faults appearing inside the modeled distributed system. In a distributed system failures can be produced at hardware level (abnormalities in the functionality of hard-

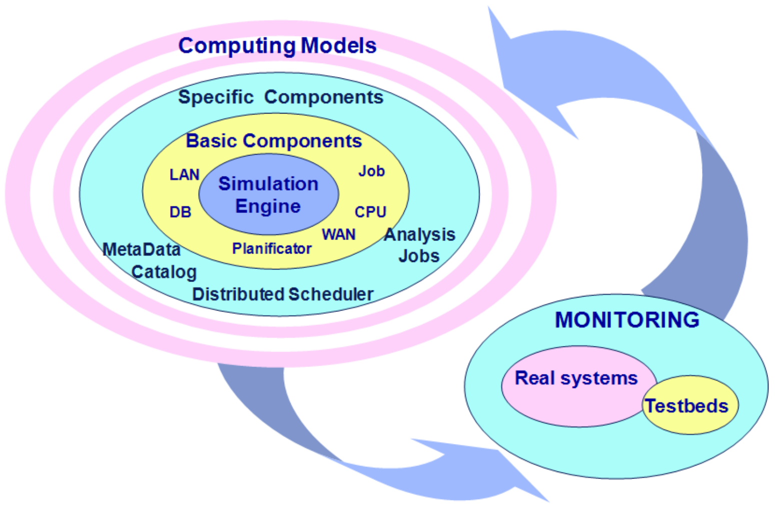

Figure 2. The layers of MONARC.

ware components) or software level (the middleware or application deviating from their normal functionality or delivery of services). We propose extending the simulation model to account for both hardware, as well as software failures, modeling their occurrences and detection, as well as recovery and masking (redundancy) mechanisms [1].

At hardware level different distributed components can be modeled as failing: the processing unit, the network connectivity as well as the storage devices. At software level we consider the faults occurring in a middleware component (the scheduler behavior could be erroneous, the database server could return wrong values, etc.) or in the higher-level distributed application (for example the jobs could fail to return correct results). For all the modeled components being considered by the simulation model we propose adding the mechanisms to inject faults.

The injection of faults will affect primarily the behavior of the components. In this way we will be able to model all types of faults: crash faults, omission faults, time faults, as well as Byzantine faults [2]. In order to cover all possible faults, we propose the use of two mechanisms, as follows.

In the first approach the user will input into the simulation model values for the MTTF (mean time to failure) parameter in case of the various components involved in a specific simulation experiment. In the simulation this parameter represents the basis for simulating the haphazardness stimuli coming from external and internal sources of influence that affect the characteristics of the modeled components and is seen as a probability measure of fault occurrence that is supposedly computed for each component prior to its deployment in the system. For modeling the fault injection mechanisms we will use the MTTF together with some mathematical probability distributions (the simulation model already uses distributions such as binomial, Poisson, Gaussian, standard uniform, exponential to implement the stochastic behavior of distributed systems). In this way, at random intervals of time, a component will experience faulty behavior (failures), as well as possible recovery. Regarding the failures, a component will experience complete crash (the component will not recover anymore and will not be accessible by modeled entities), omissions (the network model will deliver only partial messages for example) or delays (the component will experience modeled delays).

The second proposed approach considers a completely random occurrence of fault events, without the user specifying any value. This is useful in modeling the most disruptive faults, the Byzantine failures that occur arbitrary in the simulated system, such as in the case of transient hardware faults for example: they appear, do the damage, and then apparently just go away, leaving behind no obvious reason for their activation in the first place [13]. For these types of errors the simulation model will allow the resuming of the normal behavior of the affected component, as it is usually enough to allow successful completion of its execution.

This is also useful for example in modeling the intervention of the human operator restarting or replacing a faulty component. For modeling this behavior the user will input into the simulation model a MTTR (mean time to repair) value that, together with a probability distribution, represents the basis of a triggering mechanism that allows the modeling of a resume operation event. In addition, in order to model a wider range of possible, a second approach to resuming will consider that the timestamp of the resuming event can also be based on a completely random decision.

A different faulty situation that needs to be addressed is represented by users or programs trying to gain access to secured components. For modeling security technolo-

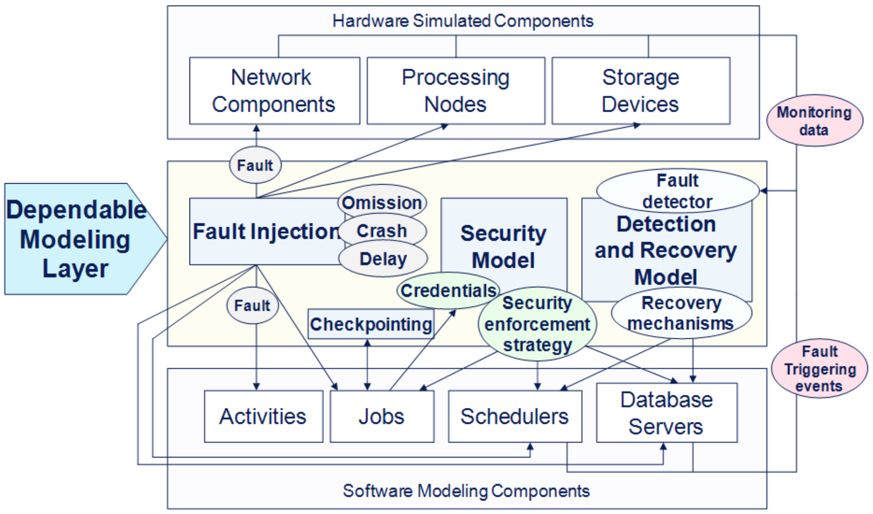

Figure 3. The proposed dependable modeling layer and its components.

gies we propose adding an extra layer that injects security problems. This layer includes the mechanisms that are used to construct a special job that communicates with various components in the attempt to gain access to restricted data or evaluate security technologies (single authentication, credentials protection, interoperability with local security solutions, etc.).

2.3. Simulation of Dependable Distributed System Technologies using MONARC

The fault injection mechanisms presented above will be used together with various fault detection and recovery mechanisms. For that we propose the addition of an external monitoring component in the simulation model. The monitoring component will be responsible for receiving data of interest (such as fault occurrence triggered events), for taking actions to update the state of the distributed system being modeled and possible inform interested components of the events occurrences.

For the purpose of the current proposal one of the roles of this component will be to keep track of the resource faults in the system and generate appropriate controlling actions. For example, as described before, based on various fault injection mechanisms, we could trigger the generation of a crash of a processing unit. The trigger will translate in the generation of a special simulation event that will be received by the monitoring component. Upon receive of such event the monitoring unit will remove the processing unit from the modeled system (it will no longer be visible and all its ongoing tasks will be forced to stop - the state update action) and inform all interested components of the occurrence of the event. In this approach a scheduler that is interesting in monitoring the state of the processing unit on which it deployed a task for execution would register with the monitoring component and, upon triggering of the crash event, will be informed of the failure so that to take the appropriate correction actions). In this model an omission fault example is the simulation of a crushed network link, where the monitoring unit will be responsible with generating corresponding interrupt events for the still-running tasks that were using that modeled link.

For modeling timing faults the monitoring unit will also play an important role. In the simulation model the boundaries of the action (start of the execution of a task, termination of a task, etc.) are modeled using simulation events. When the monitoring unit receives a timing fault triggering event it will simply modify termination events so that to be triggered at a later time in the future. In this way we simply change the default behavior so that a task will not end when it was supposed to end, but sometimes later in the future. A modeled fault-tolerant software component could then use timing checks (deadlines) to look for deviations from the acceptable behavior. For example, the scheduler will also implement a fault-tolerant mechanism, according to which whenever a new job is submitted the scheduler will also produce a special simulation event that will be triggered when the timeout occurs (where by timeout we mean an amount of time dependant on the user specification that will be triggered if the job fails to return results in due time). The scheduler will then be interrupted by one of two actions: either the job finishes or the timeout event occurs (which one happens faster in the simulation timeline). The same mechanisms will be implemented to the network simulation model, a job being informed if a transfer failed to finished in a specified amount of time (possible due to network congestion for example) in order to consider taking appropriate measures (such as canceling the transfer for example).

We state that the described extension to MONARC's simulation model will be useful for testing both reactive and proactive fault tolerance existing techniques [14]. In case of the reactive fault tolerant systems the fault monitoring systems is informed after the detection of the failure in order to start the corresponding recovery processes. This is in concordance with the presented functionality of the proposed modeled monitoring component. The predictive (proactive) fault tolerant systems predict a fault before its occurrence. Predictive systems are based on the assumption that faults do show some disruptive effect on the performance of the system [15]. Much effort has been put recently into developing proactive fault tolerant systems [16] [17] [18]. In our approach the user could evaluate the performance of the distributed system in the presence of various predictive fault tolerance techniques by augmenting the monitoring component with various prediction algorithms.

In order to allow the evaluation of a wide-range of dependability technologies, the simulation model will also include the necessary mechanisms to allow the modeling of check-pointing or logging of the system's state. These mechanisms will be implemented based on the simulation events and the state of the objects used to simulate the modeled components. For that the job interface will provide a method that, when called, will result in the saving of the serialized objects as well as the state of the simulation on the local disk storage. This will be useful in experiments dealing with both static and dynamic check-pointing strategies.

The simulation model will also include the evaluation of various replication and redundancy mechanisms. The replication provides mechanisms for using multiple identical instances of the same system or subsystems and choosing the result based on quorum. The simulation model already allows the simulation of DAG distributed activities. This construction is also useful in modeling the replication of the jobs, where the same job would be executed on multiple processing units and another job will be used to receive the outputs and select the correct result. Or another approach could be to model special voter jobs acting at the input of the replicated jobs and selecting the correct results (as the one described in [14]). We propose augmenting the simulation model with these approaches as well.

The redundancy is another characteristic that is important to be considered when evaluating dependable distributed system strategies. The fault-tolerance techniques applicable to software, either single version of multi-version, are based on their vast majority on the use of redundancy, on the use of resources beyond the minimum needed to deliver the specified services [18]. For example, a strategy to achieve reliability could be that when a storage component fails the system would select a different replica of that storage still functioning and query it for the data of interest. This is already possible in MONARC, as demonstrated by the experiments described in [3]. In the experiments we describe how the simulation model already deals with replicating database storages. We propose to add replication mechanism to in case of the simulated jobs and scheduler as well.

The simulation of security technologies also plays an important part in assessing dependable systems. We propose adding mechanisms to evaluate various security procedures that are essential in dependable distributed systems. These mechanisms refer to providing availability for authorized users only, confidentiality and integrity. The networking model being provided by MONARC can be easily extended to provide the modeling of all these aspects. The message being transferred among simulated entities can carry inside a wide-range of information. This can be used to implement various authorization enforcement mechanisms, such as for example certificates that are provided as input to the modeling experiments and that are automatically verified upon the receipt of the message. We propose extending the network and job models to incorporate several such security technologies.

We also propose adding an extra layer to the simulation model, a security layer that could be used to simulate various security-related procedures. The security layer will provide mechanisms to encrypt the data that could be used in conjunction with the storage components for example. It will also include various security protocols (such as SSL and TLS for example, or group key exchange protocols) that could be used of-the-shelf by users trying to run their own experiments in the presence of various security mechanisms.

2.4. Conclusions

As society increasingly becomes dependent of distributed systems (Grid, P2P, network-based), it is becoming more and more imperative to engineer solutions to achieve reasonable levels of dependability for such systems. Simulation plays an important part in the building process of dependable distributed systems. MONARC is a generic simulation framework designed for modeling a wide range of distributed systems technologies, with respect to their specific components and characteristics. Its simulation model incorporates the necessary components to model various modern-day distributed systems technologies, providing the mechanisms to describe concurrent network traffic and to evaluate different strategies in data replication or in the job scheduling procedures.

In this paper we described a solution to simulating dependable distributed systems using MONARC. The solution is able to model failures in distributed systems at hardware level (abnormalities in the functionality of hardware components) or software level (the middleware or application deviating from their normal functionality or delivery of services). We aim at extending MONARC to account for types of failures (at hardware and software levels), modeling their occurrences and detection, as well as recovery and masking (redundancy) mechanisms.

References

[1] A. Avizienis, J. C. Laprie, B. Randell, “Fundamental Concepts of Dependability”, Research Report No 1145, LAAS-CNRS, April 2001.

[2] A. Avizienis, V. Magnus, J. C. Laprie, B. Randell, “Fundamental Concepts of Dependability”, ISW-2000, Cambridge, MA, 2000.

[3] I. C. Legrand, H. Newman, C. Dobre, C. Stratan, ”MONARC Simulation Framework”, International Workshop on Advanced Computing and Analysis Techniques in Physics Research, Tsukuba, Japan, 2003.

[4] Florin Pop, Ciprian Dobre, Gavril Godza, Valentin Cristea, “A Simulation Model for Grid Scheduling Analysis and Optimization”, Proceedings of International Symposium on Parallel Computing in Electrical Engineering (PARELEC’06), pp. 133-138, September 13-17, 2006, Bialystok, Poland, Published by IEEE Computer Society, ISBN: 0-7695-2554-7

[5] C. Dobre, C. Stratan, “MONARC Simulation Framework”, RoEduNet International Conference, Timisoara, Romania, 2004.

[6] 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, Bucharest, Romania, 2005.

[7] H. Casanova, A. Legrand, M. Quinson, ”SimGrid: a Generic Framework for Large-Scale Distributed Experimentations”, Proc. of the 10th IEEE International Conference on Computer Modelling and Simulation (UKSIM/EUROSIM’08), 2008.

[8] R. Buyya, M. Murshed, ”GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing”, The Journal of Concurrency and Computation: Practice and Experience (CCPE), Volume 14, 2002.

[9] W. Venters, et al, ”Studying the usability of Grids, ethongraphic research of the UK particle physics community”, UK e-Science All Hands Conference, Nottingham, 2007.

[10] K. Ranganathan, I. Foster, ”Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications”, Int. Symposium of High Performance Distributed Computing, Edinburgh, Scotland, 2002.

[11] C. Dobre, V. Cristea, ”A Simulation Model for Large Scale Distributed Systems”, Proc. of the 4th International Conference on Innovations in Information Technology, Dubai, United Arab Emirates, November 2007.

[12] C. Dobre, C. Stratan, ”MONARC Simulation Framework”, in Proc. of the 3rd Edition of RoEduNet International Conference, Timisoara, Romania, 2004.

[13] J. Gray, ”Why Do Computers Stop and What Can Be Done About It?”, Proc. of the Fifth Symposium On Reliability in Distributed Software and Database Systems, 1986.

[14] N. Naik, ”Simulating Proactive Fault Detection in Distributed Systems”, Proposal Draft for Capstone Project, 2007.

[15] T. A. Dumitras, P. Narsimhan, ”Fault-Tolerant Middleware and the Magical 1%”, ACM/IFIP/USENIX Conference on Middleware, Grenhole, France, 2005.

[16] L. Yawei, Z. Lan, ”Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing”, Proc. of the sixth IEEE International Symposium on Cluster Computing and Grid, 2006.

[17] P. Narasimhan, R. Rajkumar, G. Thaker, P. Lardieri, ”A Versatile, Proactive Dependability Approach to Handling Unanticipated Events in Distributed Systems”, Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005.

[18] W. Torres-Pomales, ”Software Fault Tolerance: A Tutorial”, Langley Research Center, Hampton, Virginia, NASA/TM-2000-210616, October 2000.