A SIMULATION MODEL FOR FAULT TOLERANCE EVALUATION

Adrian BOTEANU[1], Ciprian DOBRE[2]

Această lucrare prezintă un model de simulare pentru evaluarea soluțiilor de asigurare a toleranței la defecte în sistemele distribuite de mari dimensiuni. Modelul extinde simulatorul MONARC prin adăugarea de noi funcționalități pentru evaluarea toleranței la defecte. Modelul descrie defecte ce pot apărea în astfel de sisteme şi include mecanisme pentru detecția si corecția acestora. În cadrul lucrării este prezentată şi o implementare pilot a modelului, împreună cu rezultatele testelor de evaluare. Au fost implementate atât defecte permanente cât şi tranziente ce pot apărea în cazul unităților de procesare, componentelor de rețea sau a bazelor de date. Modelul poate fi uşor extins, permițând adăugarea de noi clase de defecte i tehnologii aferente, în funcție de experimentul vizat. Modelul poate fi folosit pentru evaluarea performanțelor unor soluții de toleranță la defecte pentru sisteme distribuite, pretându-se identificării rapide a punctelor sau ariilor vulnerabile din sistemul simulat.

In this paper we present a simulation model designed to evaluate fault tolerance solutions in large scale distributed systems. This model extends the MONARC simulation model with new capabilities for fault tolerance simulation. The model includes failure behavior and capabilities to detect and react to faults. We also present an implementation of this model in MONARC, together with specific evaluation results. The model's implementation considers permanent and transient failures occurring within processing units, network components, as well as databases. The model is easily extendable, allowing the additions of new failure models as required by user experiments. The model can be used in conjunction with key performance metrics, being able to easily pinpoint the likely points or areas of failures in the simulated environments.

Keywords: fault tolerance, distributed systems, performance analysis, simulation model, faults

1. Introduction

Modeling and simulation were seen for a 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

[1] Eng., Faculty of Automatics and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: adiboteanu@gmail.com
[2] Lect., Faculty of Automatics and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: ciprian.dobre@cs.pub.ro

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.

Together with the extension of the application domains, new requirements have emerged for large scale distributed systems; among these requirements, fault tolerance and resilience in face of possible failures occurring in such systems, are needed by more and more modern distributed applications, not only by the critical ones.

The discrete event simulation offers a flexible and powerful method to evaluate solutions designed for large scale distributed systems, without the time and effort necessary to implement them in real-world. MONARC is a complex simulator designed around this paradigm, in which discrete events trigger the advance of the simulation. Its model offers the means of simulating a large range of scenarios, ranging from networking protocols to scheduling and distributed applications running on top of the middle-ware. That is, as long as the simulation experiments exclude the possibility of component failures. However, modern requirements of distributed systems, such as fault tolerance, should also be supported and analyzed using the simulation model. The adoption of a fault tolerance model was also imperative to completely and accurately model the behavior of a real distributed system, having an imperfect nature, as in the case of Internet based systems like GRIDs.

In this paper we present an extension to the MONARC simulation model that allows the analysis of failure-dependent experiments, where faults can occur in any simulated component. The model provides realistic observation of failed components and provides a configurable interface for the user. By adding these capabilities, the simulation scenarios can include evaluation of failure detection approaches, as well as replication or consistency solutions designed for large scale distributed systems. The model can be used for both reactive and proactive types of situations and recovery solutions in the presence of faults.

The main capabilities of the failure simulation model are flexibility and compatibility. Existing MONARC simulation experiments must easily be adapted to evaluate failure behavior. The model also includes a wide range of possible defects, from permanent crashes occurring in various components to transient or Byzantine errors. The user is also presented with an interface to easily allow other simulated mechanisms to be included.

In the model the mechanisms for fault injection, fault detection and possible failure recovery techniques are completely separated, to have a better understanding of the entire processes involved. The component that is prone to failures generates at various moments, according to a statistical probability distribution, a change of state. A supervisor component periodically interrogates, using a heartbeat approach (the default one), it's available resources. It then

evaluates the functioning state of the component by the number of consecutive heartbeat answers of the same type, negative or positive.

The rest of this paper is structures as follows. We next present related work in the domain of simulating failures in large scale distributed systems experiments. We next present the architecture of the model and we present details about the implementation of the various components. Next we present results and a detailed analysis of the capabilities of the proposed solution. The final section presents conclusions and future work.

2. Related work

SimGrid [1] is a simulation toolkit that provides core functionality 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 scheduling algorithms and generates correct and accurate simulation results. *GridSim* [2] is a grid simulation toolkit developed to investigate effective resource allocation techniques based on computational economy. *OptorSim* [3] 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* [4] 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 fault tolerance 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. Our model aims to provide the means to evaluate a wide-range of solutions for fault tolerance in case of large scale distributed systems.

The simulation model provided by MONARC is more generic that others, as demonstrated in [5]. 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. MONARC offers ample customization possibilities, thus enabling us to integrate our model while preserving the interface. Also, because of this feature, our own model can incorporate custom failure, recovery and rescheduling algorithms that the user may need for a particular scenario.

3. A Simulation Model for Fault Tolerance

MONARC is built based on a process oriented approach for discrete event simulations, which is well suited to describe concurrent running programs,

network traffic as well as stochastic arrival patterns, specific for such type of simulations. 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, as demonstrated in [6], because of the considered optimizations, 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 or with thousands of network transfers happening simultaneously.

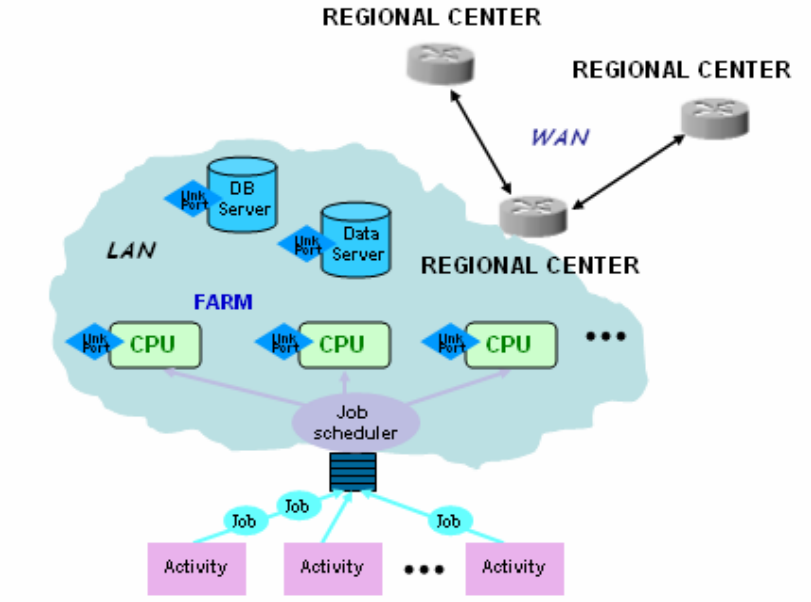

Fig. 1. The Regional center model being adopted in MONARC, [16]

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 its 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 (Fig. 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 Fig. 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, an aspect demonstrated in [10]. This is useful for example when designing experiments that are meant to experiment new conditions starting from existing real distributed infrastructures.

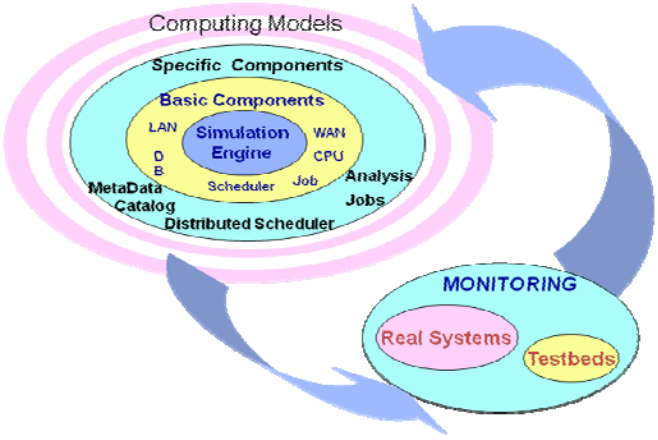

Fig. 2. The layers of MONARC, [16]

Using this structure it is possible to build a wide variety 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 [11], presenting important results for the future LHC experiments, which 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 [11], and [12].

In [13] we presented a first extension to the model designed to simulating faults occurring in distributed systems using MONARC. In this we extend the presented work, with results about the implementation of the proposed model in MONARC, its evaluation, but we also describe additional mechanisms (for modeling different types of failures, at hardware and software levels, together with their occurrences and mechanisms for detection, as well as recovery and masking mechanisms) to cover a complete set of resilience-related characteristics, in its generic sense.

The characteristics of large scale distributed systems make the problem of assuring fault tolerance 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 reliability and make it more vulnerable to various faults occurring in different nodes of the systems because of the heterogeneous possible accesses. 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.

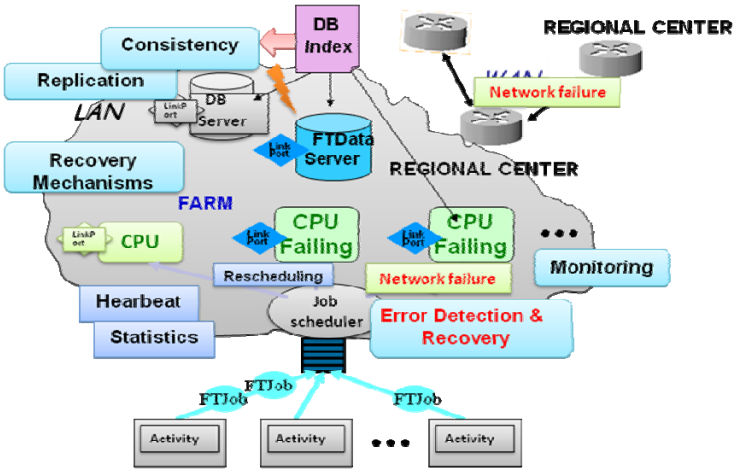

Fig. 3. The Regional Center model

The model extended each of the MONARC's three layers. At each level we implemented different functionalities and different levels of fault-tolerance model abstractions. The Fig. 3 presents the components of the extended model. As presented, each component has a supervisor and/or belongs to a larger system

(also modeled as a larger component). For example, the processing units can be grouped (along with other elements, such as a scheduler) in farms that are associated to Regional Centers. When a task is received by one such Regional Center, the scheduler automatically distributes it to an appropriate processing unit. This Regional Center extended architecture is presented in Fig. 3.

The extended model include components and methods necessary to implement fault tolerance in the processing, networking as well as database layers. For example, we added specific component capable to model faults occurring in local network and in wide area network, as well as the mechanisms to implement fault recovery protocols in the communication layer. The components susceptible to failure are network links (for local networks) and routers (for wide area networks) with their respective supervisors LANs and WANs, as presented in Fig. 4.

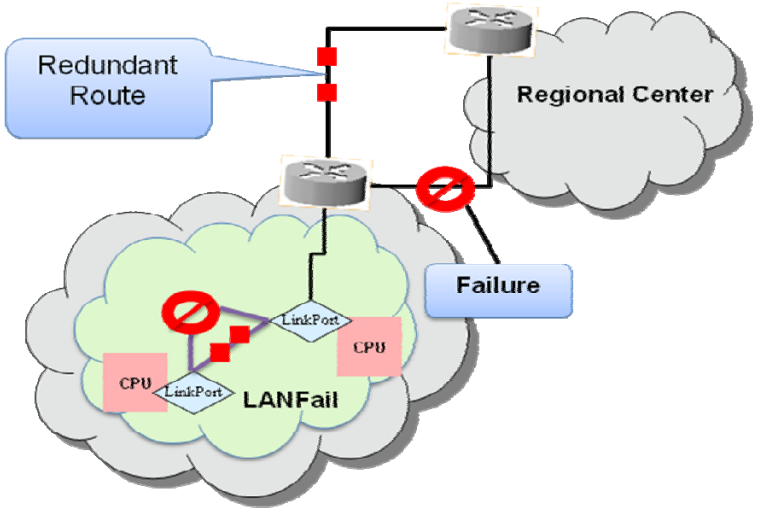

Fig. 4. Network model with local networks and wide area networks

The extended simulation model is constructed based on the basic components, but also uses the modified components from the layer composed of specific components (see Fig. 2). The model allows the simulation of hybrid systems, in which failing components can coexist with traditional components of the MONARC's model. This is possible because the fault injection mechanism is "hidden" from the other components that the affected one is communicating with.

Within the model, a fault can be detected by a supervisor of the affected component through heartbeat interrogations. For example, the Supervisor component, having a monitoring role, can send messages to the processing unit it wants to use before sending the actual task. If it fails to answer in due time (according to the detection algorithm being used by the user), the simulation can assume the component failed. This leads to no further tasks being scheduled on

that particular processing unit and its tasks are rescheduled on other processing units from the same regional center. The Scheduler can also use a registry index, attached to the monitoring task, to find out when a particular processing unit recovered from transient failures. Such transient failures can be attributed to periodical failures, so a unit in this condition would not be suitable for executing long-time jobs. The scheduler uses the monitor also to find out what processing units are currently available in the system. The monitor can question processing units periodically and dynamically adjust the list of still available processing units. Also, new processing units can dynamically enter the system and register to the monitor, so that to be used in the scheduling process.

The failure mode, permanent or transient, is decided within the CPU class and it is not visible from the outside (for example, by the Scheduler). The same applies for all network and database components, also including the failure model in their implementation. Fig. 5 shows the components that were added to the original model in MONARC.

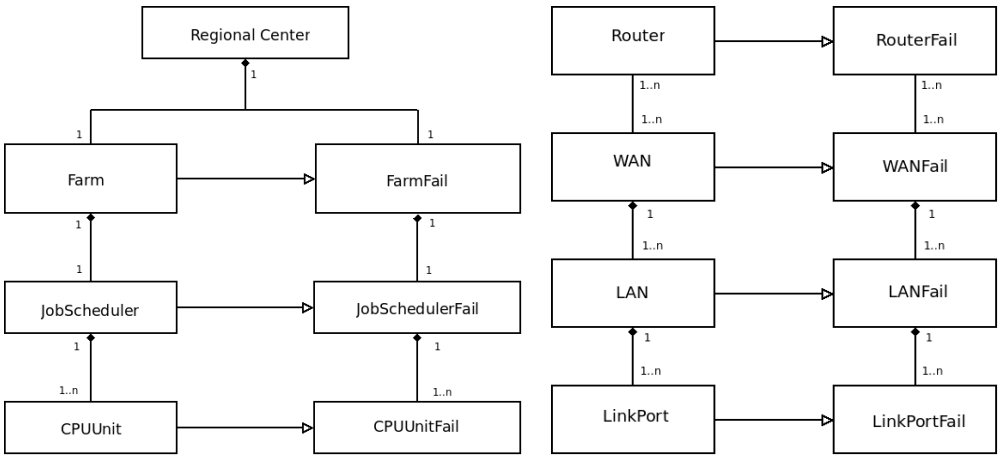

Fig. 5. Classes added to implement the failure model and their relation with the original MONARC classes

The extent to which a fault affects the performance of the regional center can be use as an indication on how the user could be alerted. For example, if all processors fail or no new tasks can be run then the experiment could state that an alert should be raised. This can be used to indicate an area in the system that is more likely to be insufficiently reliable.

The user can also modify various parameters through configuration files. The format of the configuration file is “parameter_name value”. If some parameters are not defined then default values are used. Some of the available

options are probabilities for permanent failure, for transient failure, for recovery from transient failure. Also the average number of consecutive heartbeat answers, negative or positive, which determine the Scheduler to change the observed working state of the processing unit can also be modified.

4. Model validation and simulation results

In order to analyze the validity and performance of the fault tolerance simulation model implemented in MONARC we conducted a number of simulation experiments. We present the results of two such experiments: one designed to test the failure model in case of processing nodes and another in case of the various network components.

The first experiment analyzed how the number of processing units is related to the reliability in processing a batch of tasks. In the experiment we envisioned a situation where the objective is to be guarantee that a given number of tasks can be processed, without taking into account delays caused by failed CPUs. If no CPU is working at a given moment, the test fails.

The test revealed the importance of repairing faulty resources. If no permanent faults occur, and transient faults occur in a reasonable range, the given task will be completed, independently of the given batch size, because the processing units are repaired faster than they break down. On the other hand, permanent failure rates have a limit, dependent on the failure probability and batch size. For very large batches (10,000-100,000), by increasing the relatively small number of CPUs (50-100) the rate of permanent failures stays constant. When correlating the real world failures with their simulation equivalents, the total number of available processing units should be taken into account rather than the physical devices: a permanent failure in the simulation should signify a permanent decrease of the total number of working CPUs. Even if a computer's hardware fails permanently in the real world, replacing it would restore the total number of working computers, so it would be appropriate to interpret this event as a transient failure.

Tables 1, 2 and 3 show test results for permanent failures (crash) alone. Tables 4 and 5 show the effects transient failures in relation to the CPU number.
The values *CrashThresh* and *TransientThresh* in the following tables represent thresholds above which the value of the random variable that models each probability leads to a failure manifestation; in these tests the random variables have uniform distribution in the interval [0,1]. *ChrashThresh* corresponds to permanent failures and *TransientThresh* to transient failures.

Table 1

10 CPUs with 10,000 Jobs, only permanent failures

Jobs	CPUs	CrashThresh	Avg. Failed CPUS	Processed
10000	10	0.9	10	110
10000	10	0.99	10	1008
10000	10	0.99 9	7	10000
10000	10	0.99 99	1	10000

Table 2

20 CPUs with 10,000 Jobs, only permanent failures

Jobs	CPUs	CrashThresh	Avg. Failed CPUs	Processed
10000	20	0.9	20	245
10000	20	0.99	20	3010
10000	20	0.99 9	6	10000
10000	20	0.99 99	1	10000

Table 3

40 CPUs with 10,000 Jobs, only permanent failures

Jobs	CPUs	CrashThresh	Avg. Failed CPUs	Processed
10000	40	0.9	40	409
10000	40	0.99	40	5365
10000	40	0.99 9	13	10000
10000	40	0.99 99	1	10000

Table 4

10 CPUs, 10000 Jobs, transient failures

Jobs	CPUs	TransientThresh	Avg. Failed CPUs	Processed
10000	10	0.5	7	4931
10000	10	0.6	3	10000
10000	10	0.7	1	10000

Table 5

40 CPUs, 10000 Jobs, transient failures

Jobs	CPUs	TransientThresh	Avg. Failed CPUs	Processed
10000	40	0.4	40	3166
10000	40	0.5	11	10000
10000	40	0.6	2	10000
10000	40	0.7	1	10000

For tables 4 and 5, the job scheduler had an optimistic approach: it considers that CPUs are failed if they don't answer for one heartbeat and that they are repaired if one positive answer is received. The following tables, 6 and 7, show how the consecutive heartbeat answer counts influence the scheduler. These parameters, also configurable, add another level of reliability evaluation. In some systems, tasks may be given only to high uptime units. *ResponseTimeThresh* is the number of consecutive negative answers after which the scheduler marks the CPU failed. *AliveThresh* is the number of positive answers after which the scheduler marks the CPU working.

Table 6

ResponseTimeThresh=2, AliveThresh=1, 10000 Jobs

Jobs	CPUs	TransientThresh	Avg. Failed CPUs	Processed
10000	10	0.3	4	10000
10000	10	0.4	4	10000
10000	10	0.5	2	10000
10000	10	0.6	2	10000
10000	10	0.7	1	10000

Table 7

ResponseTimeThresh=1, AliveThresh=2, 10000 Jobs

Jobs	CPUs	TransientThresh	Avg. Failed CPUs	Processed
10000	10	0.3	10	23
10000	10	0.4	10	53
10000	10	0.5	8	411
10000	10	0.6	5	6183
10000	10	0.7	2	10000

The second scenario shows the relation between redundant network links and link reliability. The goal is to send a given number of packets, without considering delays. TCP was chosen for the transport protocol to test the resend mechanism for failed connections. In this scenario both connections (link ports) and routers can fail. In this experiment each Job sends one packet in the network.

Fig. 6 shows the network topology being used. *Cern LAN* sends packets to *Caltech LAN*. Packets are routed by *Cern Router* through the two possible paths towards *Caltech Router* in respect to network load.

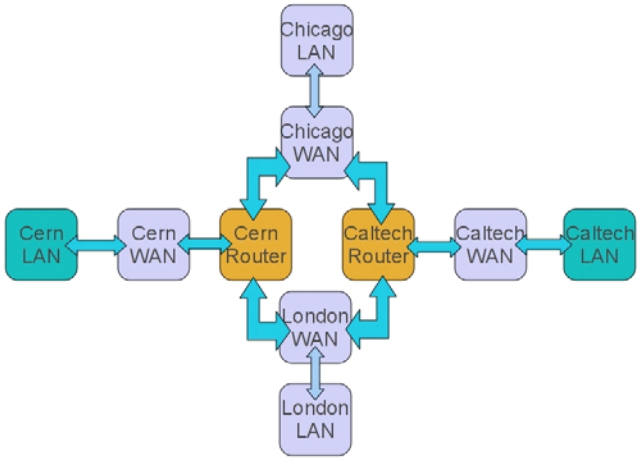

Fig. 6. Network topology for network failure scenarios.

Table 8

LinkPortFail tests, 10 Jobs, permanent and transient failures

Jobs	Link Crash Thresh	Link Transient Thresh	Average Loss
10	0.8	0.8	1
10	0.7	0.7	5
10	0.6	0.6	7
10	0.5	0.5	8

Tables 8 shows test results for link port failures. Since the transport protocol used is TCP, permanent crashes are sent again, thus the average number loss is very much like in transient failure only situations.

Table 9 shows the simulation results for router failures alone, both permanent and transient. The given values are from numerous simulations. Because there are only two routers and no backup ones, if one of them fails permanently all packets afterwards are not delivered.

Table 9

RouterFail tests, 10 Jobs, permanent and transient failures

Jobs	Router Crash Thresh	Router Trans Thresh	Average Loss
10	0.99 9	0.99 9	0
10	0.99	0.99	1
10	0.9	0.9	9
10	0.8	0.8	9
10	0.7	0.7	10

Routers are not influenced by the chosen transport protocol, so there are no alternate means of assuring higher reliability other than intrinsic router reliability or redundancy.

5. Conclusions

Modern distributed systems are more and more seen as the most likely solution to the ever increasing computational needs, despite they are more prone to partial failure. Failures should not be limited to actual malfunctioning; a shutdown of a computer in a "@home" type of network is enough to disturb the process. However, these events are predictable and failure resistant algorithms exist. Given these circumstances, simulating failures in a distributed system can give good predictions over the actual behavior.

The framework developed using MONARC gives users a simple interface to use, by adding some parameters to the configuration files. All existing configuration files supported can be modified to enable the failure model. The simulation model is easy to understand, but flexible enough to allow various scenarios. MONARC's versatility is not reduced. The two fault types implemented, permanent and transient, are the most common and the input values can be adapted to measured values in real life systems.

In this paper we presented an extension to the MONARC simulation model that allows the analysis of failure-dependent experiments, where faults can occur in any simulated component. The model provides realistic observation of failed components and provides a configurable interface for the user. By adding these capabilities, the simulation scenarios can include evaluation of failure detection approaches, as well as replication or consistency solutions designed for large scale distributed systems. The model can be used for both reactive and proactive types of situations and recovery solutions in the presence of faults.

Future development could include realistic routing protocols by which routers make decisions. It may be used in verifying that a protocol is adequate for a given topology, estimating parameters such as convergence speed and network overhead.

Acknowledgement

The research presented in this paper is supported by national project "DEPSYS – Models and Techniques for ensuring reliability, safety, availability and security of Large Scale Distributes Systems", Project "CNCSIS-IDEI" ID: 1710.

REFERENCES

[1] *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

[2] *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

[3] *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

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

[5] *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

[6] *C. Dobre*, "Advanced techniques for modeling and simulation of Grid systems", PhD Thesis publicly sustained at University POLITEHNICA of Bucharest, January 2008

[7] *C. Dobre, C. Stratan, V. Cristea*, "Realistic simulation of large scale distributed systems using monitoring", in Proc. Of the 7th International Symposium on Parallel and Distributed Computing (ISPDC 2008), Krakow, Poland, July 2008

[8] *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

[9] *F. Pop, C. Dobre, G. Godza, V. Cristea*, "A Simulation Model for Grid Scheduling Analysis and Optimization", Parelec , 2006

[10] *C. Dobre, F. Pop, V. Cristea*, "A Simulation Framework for Dependable Distributed Systems", First International Workshop on Simulation and Modelling in Emergent Computational Systems (SMECS-2008), Portland, USA, 2008

[11] *C. Dobre, V. Cristea*, Advanced techniques for modelling and simulation of Grid systems, 2008

[12] *N. Naik*, Simulating Proactive Fault Detection in Distributed Systems, Proposal for Capstone Project, 2007

[13] *F. Cristian*, Understanding Fault-Tolerant Distirbuted Systems, Communications of the ACM, vol. 34, feb 1991

[14] *K. Neocleous, M. D. Dikaiakos, P. Fragopoulou, E. Markatos*, GRID RELIABILITY: A STUDY OF FAILURES ON THE EGEE INFRASTRUCTURE, Proposal Paper, 2006

[15] *A. Avizienis, J.C. Laprie, B. Randell*, Dependability and it's threats: A Taxonomy

[16] Official MONARC page http://monarc.cacr.caltech.edu:8081/www_monarc/monarc.htm