Simulator for Fault Tolerance in Large Scale Distributed Systems

Adrian Boteanu[*], Ciprian Dobre[*], Florin Pop[*], Valentin Cristea[*]
[*] *University POLITEHNICA of Bucharest, Romania*
E-mails: adrian.boteanu@cti.pub.ro, {ciprian.dobre, florin.pop, valentin.cristea}@cs.pub.ro

Abstract

We present a simulation model designed for the evaluation of fault tolerance solutions working 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 different experiments. The model can be used in conjunction with key performance metrics, being able to easily pinpoint areas of failures within the simulated environments.

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

1. Introduction

Modeling and simulation were long time seen 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.

Together with the extension of the application domains, new requirements have emerged for large scale distributed systems; among these requirements, fault tolerance is 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 [5]. 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 middleware. 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 within various simulated component. The model provides realistic observation of failed components and provides a configurable interface that allows the user to integrate failures in many different scenarios. By adding such 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 ([6]-[10]) can 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, so that the user has a better understanding of the processes involved. The component prone to failures generates at various moments, according to statistical probability distribution, a change of state. A supervisor component periodically interrogates, using a heartbeat approach (the default approach), 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. In Section 2 we present related work in the domain of simulating failures in large scale distributed systems experiments. Section 3 presents the architecture of the model and details about the implementation of the components that integrate fault tolerant mechanisms. We next 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 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 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 designed specifically for testing 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.

These simulators were developed for evaluating particular classes of experiments. Some were designed for evaluating scheduling solutions. Others were developed for data transfer and replication technologies. 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.

These simulators lack the support for generic evaluation of fault-tolerance in distributed systems. A fault occurring in such systems could lead to abnormal behavior of any of the system's components. We argue that a correct evaluation of fault-tolerance in distributed systems should consider 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 [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.

3. The MONARC simulator

MONARC is built around 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.

In order to provide a realistic simulation, all the components of a distributed 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 (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.

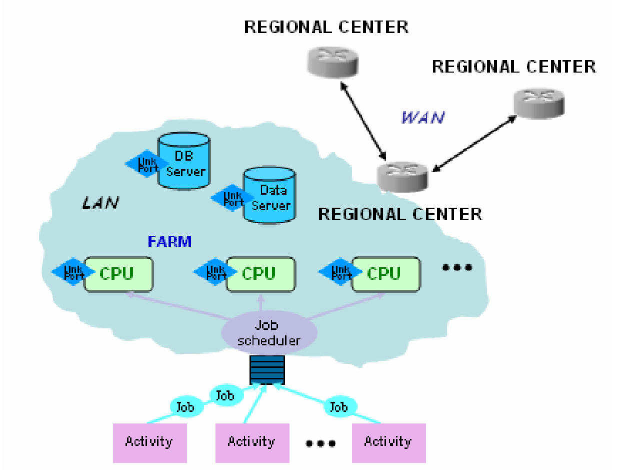

Fig. 1. The Regional center model adopted in MONARC.

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 layer contains the core of the simulator (called the "simulation engine"). If follows a layer responsible for modeling 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 represent different types of jobs, job schedulers with specific scheduling algorithms and politics 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 has over other existing simulators 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.

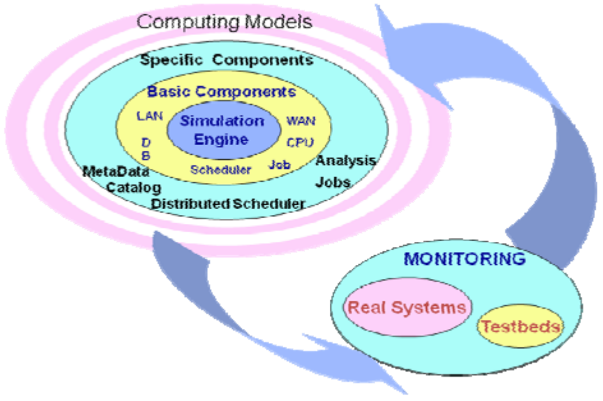

Fig. 2. The layers of the MONARC simulator.

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 LHC experiments at CERN, in Switzerland. A series of scheduling simulation experiments were presented in [11], and [12].

In [13] we presented a first extension to the model designed to simulate faults occurring in distributed systems using MONARC. In this we extend the presented work with results obtained after implementing the fault tolerance model in MONARC. 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, that cover a complete set of resilience-related characteristics, in its generic sense.

4. A Simulation Model for Fault Tolerance

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.

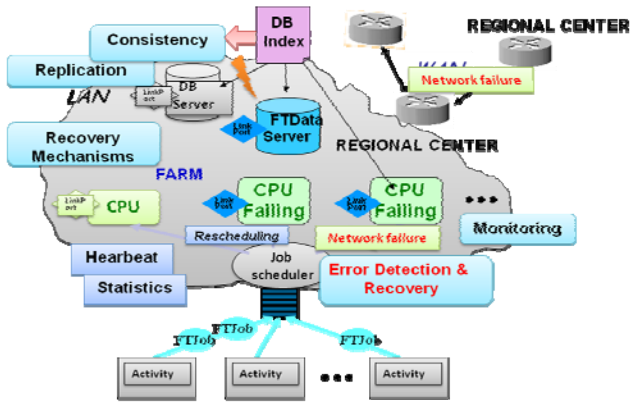

Fig. 3. The Regional Center model.

The model extends all of MONARC's layers. At each layer we added several functionalities and different levels of fault-tolerance model abstractions. Figure 3 presents the components of the extended model. As presented, each component has a supervisor and/or belongs to a bigger 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 a Regional Center, the scheduler automatically distributes it to an appropriate processing unit. This extended architecture is presented in Figure 3.

The extended model includes components and methods necessary to implement fault tolerance in the processing, networking as well as database layers. For example, we added specific components 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 Figure 4.

The extended simulation model is constructed around previously existing simulated components. This allows the simulation of hybrid systems, in which failing components can coexist with traditional components of the MONARC's original model. This is possible because the fault injection mechanism is "hidden" from the other components that the affected one is communicating with.

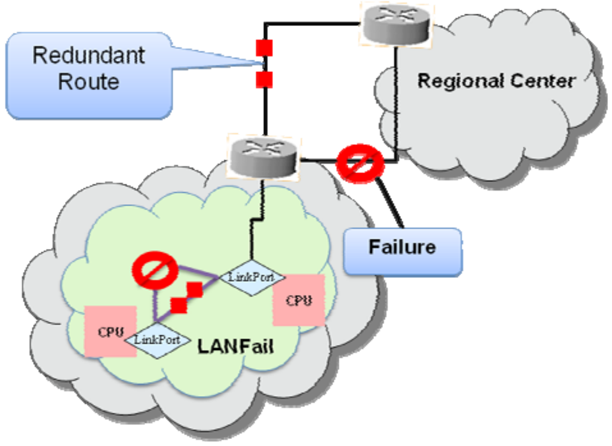

Fig. 4. Network model with local networks and wide area networks.

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 in a simulation experiment, can send messages to the processing unit that it wants to use before sending the actual task. If it fails to answer in due time (according to the detection algorithm specified by the user in the experiment), the simulation can assume the component failed. The decision of considering a processor as "failed" is based on the weighted average of the number of positive answers vs. negative answers. This can result, for example, in the decision of the scheduler to not schedule any jobs on that particular processing unit and all the jobs previously executing there are automatically rescheduled on other processing units. Rescheduling a jobs takes place in two steps: the first is to stop the job by halting the it's thread and add it to the pending jobs list; second step is to find a working processor, assign it the job and restart the job's thread. Some custom operations may be performed on the rescheduled jobs, such as resetting the work done. Such an extensible approach allows the evaluation of various fault detection solutions, as well as fault recovery ones (such as this re-scheduling solution). The first step and the optional custom operations are accomplished by the supervisor. From the schedulers point of view, there is no difference between a new job and a halted/reset job.

The observed fault occurrence schema is configurable via the metrics for calculating processor availability. Taken into consideration are the impact of a new answer over the estimated state, minimum expected consecutive positive answers, maximum tolerable number of negative answers and period of assessment (by default, all answers are considered).

Also the supervisor will not interrogate a faulted processor for some configurable time, a feature useful for simulating scenarios in which processor interrogation is costly.

The Scheduler default MONARC algorithm is to pick the first usable processor for a task, however it is able to integrate custom (re)scheduling algorithms, can also use a registry index, attached to the monitoring process, to be notified when a particular processing unit recovers from transient failures. Periodical failures can easily lead to a situation where a unit would not be suitable for executing long-time jobs. The scheduler can also use the monitor to discover the processing units currently available in the system. The monitor can also question processing units periodically and dynamically adjust the list of still available processing units. Also, new processing units can be dynamically added in the system and registered to the monitor, so that to be used by the scheduler.

The failure mode, either permanent or transient, is decided within the CPU class via configurable statistical means and 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. Therefore, every component, either a network entity, a processing unit, a database server, they all are capable of simulating failures. The user can instruct the experiment how these components fail (possible according to some probability distribution).

Figures 5 and 6 present the described components added to the original model in MONARC in order to implement the fault tolerance capabilities.

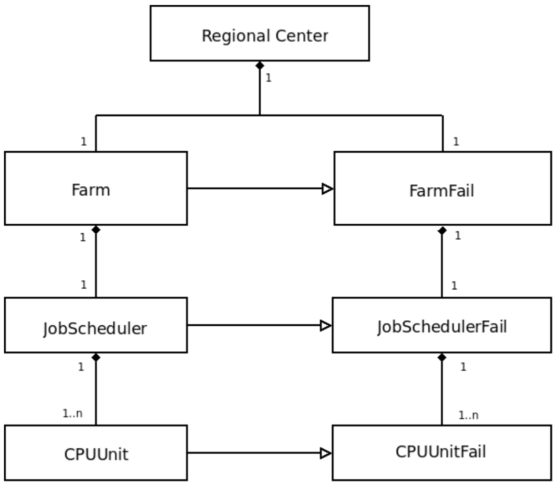

Fig. 5. Classes added within the Regional Center to implement the failure model.

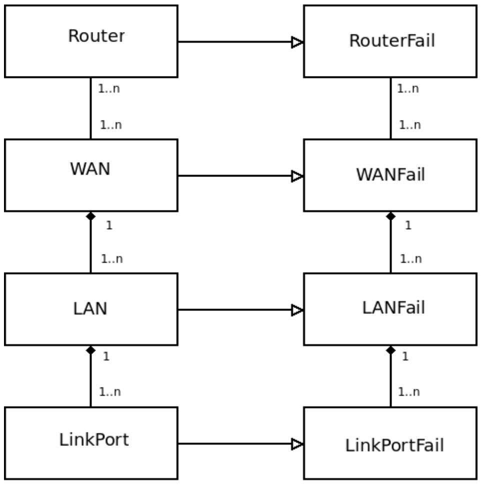

Fig. 6. Classes added within the network model to implement the failure model.

The extent to which a fault affects the performance of the regional center can be use as an indication on how the user can be further alerted. For example, if all processors fail or no new tasks can be run then in the experiment the user can specify 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 using the provided configuration file. If some parameters are not defined then the experiment considers default ones. Some of the available options are probabilities for permanent failure, for transient failure, for recovery from transient failure.

5. Model validation and simulation results

In order to analyze the validity and performance of the fault tolerance simulation model we conducted several 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 failing 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 considered the situation when the objective is to guarantee that a given number of tasks can be processed, without taking into account delays caused by failed processing units. If no processing unit is working at a given moment, the test fails.

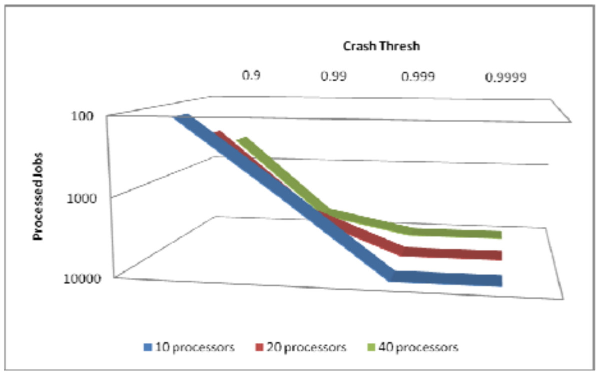

Fig. 7. Results obtained for the permanent failure experiment.

The experiment consisted of a number of jobs being sent for processing. The job scheduler is responsible with finding a suitable processing unit for each of these jobs. The results in figure 7 were obtained for different cases, consisting of 10 (blue), 20 (red), and 40 (green) processors involved and a fixed number of jobs being sent to execution (10,000). We were particularly interested in the capability of the job scheduler to mask failures by re-scheduling jobs when processing unit crash. The *CrashThresh* parameter expresses the probability of the processing units to experiment crash (permanent) failures (the value ranges from 0 to 1). For particular values of these parameters (Fig. 7) the job scheduler gets into a state where, because of too many crashes, there are no more processing units capable of executing jobs. In this situation the scheduler is no longer able to mask failure and, therefore, the user sees a lower number of processing jobs successfully executed (vertical axis).

Table 1. Results for transient failures.

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

Other experiments considered a set of 10 processing units able to experience transient failures. In these experiments we varied the probability of processors to experience failures (the transient thresh parameter) and the probability of processors to recovery from failures (the alive thresh parameter). Table 1 shows results obtained when varying such parameters, showing a bottleneck for the number of jobs that are successfully executed. In this case 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.

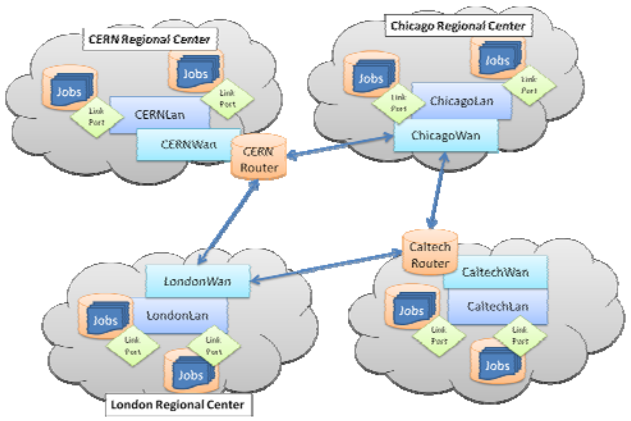

Fig. 8. Network topology for network failure scenarios.

These experiments reveal the importance of taking repairing actions in case of faulty resources. If no permanent faults occur, and transient faults occur in a reasonable range, a task will still be completed, independently of the batch size, because the processing units are repaired faster than they break down. On the other hand, permanent failure rates have a limit, depending on the failure probability and batch size. For very large batches (between 10,000 and 100,000 jobs), by increasing the relatively small number of processing units (between 50 and 100), the rate of permanent failures remains constant.

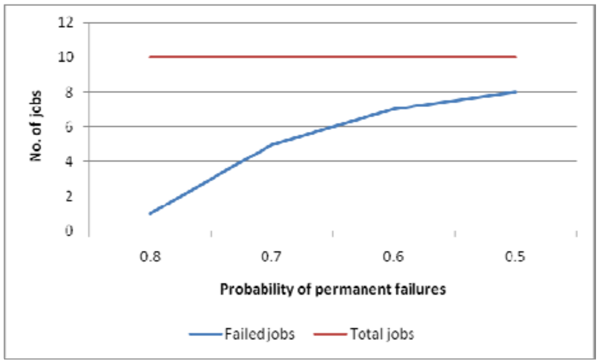

Fig. 9. Results obtained for different probabilities of links to experience permanent failures.

Since tasks conserve the work done when stopped, the time is proportional with the average percentage of failed processors if all tasks are completed.

Efficiency is calculated by dividing the ideal completion time to the actual completion time if failures occur By default, jobs are not reset when rescheduled, resulting in efficiency values proportional to the average number of working processors. If jobs are reset when rescheduled, efficiency is much more correlated to MTBF; if a processor can never finish by itself a job, no jobs will be completed, resulting in 0 efficiency.

The second set of experiments shows the relation between redundant network links and link reliability. The goal was 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 these scenarios both connections (link ports) and routers are able to fail. In the experiments each job is responsible for sending one packet in the network.

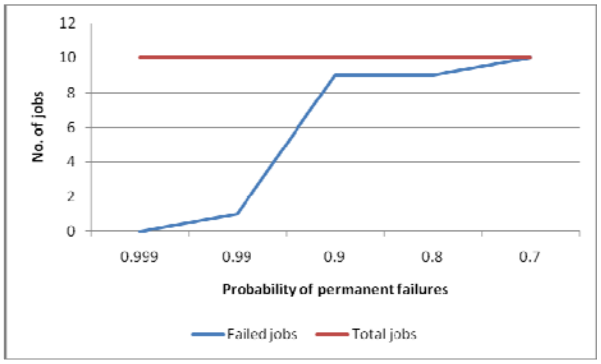

Fig. 10. Results obtained for different probabilities of routers to experience permanent failures.

Figure 8 shows the experiment's network topology. *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.

Figure 9 shows results obtained for the case when the network links can experience permanent failures. We considered a number of 10 jobs that are sending messages. We then varied the probability of a link to experience permanent failures. The vertical axis shows the number of jobs that were able to complete their tasks of transferring the data.

Figure 10 shows the simulation results obtained for a series of experiments that considered that routers can permanently fail. Because the scenario consists of only two routers, if one of them fails permanently all packets afterward are not delivered.

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

5. Conclusion and future work

Fault tolerance represents an important property of modern distributed systems. Because of their characteristics, such systems are more likely to experience failures then traditional systems. Failures are diverse and can occur in various components, from processing units to network links or data storage entities.

The importance of coping with failures is denoted by the large number of research results currently developed in this context. Traditionally, the preferred way of evaluating fault tolerance solutions for distributed systems consists in implementing them in real-world systems. However, this approach is costly and cannot lead to predictions on how the fault tolerance solution behaves under different sets of parameters.

In this we present an alternative solution to the problem of evaluating fault tolerance technologies for distributed systems, using modeling and simulation. We present a simulation model that includes the mechanisms and components necessary to construct a wide range of simulation experiments for evaluating fault tolerance. The model includes capabilities to simulate transient or permanent failures, occurring in processing units, in network components or data storing components. It also includes the capabilities to test various fault detection algorithms, using for example heartbeat monitoring of distributed resources. It also includes capabilities to experience with various fault recovery solutions. For example, the experiment can include a job scheduler capable of rescheduling jobs when processing units experience failures. It also includes the capability to replicate components .

The model was implemented in MONARC, a generic simulator that can model a wide range of distributed systems. The simulation components were extended to support the fault tolerance capability. The result is a easy to use simulator, flexible enough to allow various scenarios.

We presented several experimental results proving the capability to test fault tolerance solutions. We presented situations where processing units, as well as networking components, can fail and, as a result, a job scheduler masks faults by rescheduling jobs, if possible, on alternative nodes.

Future developments include the addition of further capabilities for modeling reactive and proactive recovery solutions. The simulator will further be used to evaluate a number of fault tolerance solutions for large scale distributed systems.

Acknowledgments

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.

6. 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 defended 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 Paper, 2007.

[13] F. Cristian, Understanding Fault-Tolerant Distirbuted Systems, 1993

[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