Simulation model and instrument to evaluate replication technologies

Bogdan Eremia*, Ciprian Dobre*, Florin Pop*, Alexandru Costan*, Valentin Cristea*
*University POLITEHNICA of Bucharest, Romania
E-mails: bogdan.eremia@cti.pub.ro, {ciprian.dobre, florin.pop, alexandru.costan, valentin.cristea}@cs.pub.ro

Abstract

Fault tolerance in distributed systems relies heavily on some form of replication. Replication can also be used to reduce the access latency and the bandwidth consumption in large scale distributed systems. However, in case of large volumes of data, the replica placing strategy and the consistency algorithms become key factors for the performance of the data replication strategy. We present a simulation model designed to realistically evaluate replication solutions for large scale distributed systems. This model was, implemented in the MONARC simulator, allowing the evaluation of various replication strategies. In this context, we present a scalable architecture designed to facilitate the adoption of data replication strategies in large scale distributed systems. The solution combines a hybrid replication model with a strategy to maintain data consistency while preserving fault tolerance. We also present evaluation results obtained using the MONARC simulator.

1. Introduction

Large scale distributed systems are widely used for their capability to provide efficient access to geographically dispersed computational and data resources through extensive networks and between different organizational entities. In such systems replication is widely used for providing efficient access to data being distributed on a large scale. As demonstrated in [1], replication can in fact lead to an increase in the availability of data. It can also reduce bandwidth consumption, can increase fault tolerance and improve scalability in such systems.

In this we present a simulation model, together with its implementation in the MONARC simulator, designed for evaluating replication solutions. It includes components for modeling faults, data storing architectures, communication protocols, failure detection solutions, consistency protocols, all in the context of a generic model, capable of simulating a wide range of architectures for distributed systems.

To illustrate the capabilities of this simulation model, we also present details about the implementation of a replication architecture designed to combine both the capabilities of partial with total replication strategies. The replication architecture consists of both a tree and a ring topology. The tree topology allow a multi-level stratification of replicas and a sequenced way of bringing data from its source to the destination entities (organizations) interested in processing. To define the data flow, the replication is optimized for read access (most data are written once - or modified by the issuing source - and read many times). Such a solution is suitable for lends itself to large-scale systems architecture that incorporate growing data volumes.

Using the proposed simulation model we analyze the properties of the proposed replication architecture by implementing it in the MONARC simulator. Its generic model allowed the evaluation of the replication architecture under various conditions. In this we demonstrate how the extendable simulation model was augmented with components specific to the proposed replication architecture. The implementation of this particular architecture in MONARC involved developing a module for the specific data replication algorithm, a module for the data consistency protocol and the extension of other components to enable integration of the new added structures.

Using the proposed simulation model, we were able to show several capabilities of the replication architecture. We present results showing that the solution not only leads to the balancing of the load to the databases available in the distributed systems, but also successfully tolerate failures.

The rest of this paper is structured as follows. Section 2 presents related work. In Section 3 we present the simulation model that allows the evaluation of a wide range of replication technologies. Section 4

presents, as a case study, details about the extension of this model with components specific to a proposed replication solution. Its implementation is detailed in Section 5. In Section 6 we present evaluation results and in Section 7 we give conclusions and present future work.

2. Related work

OptorSim [2] 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. Given a replication algorithm and a Grid configuration as an input, it runs various activities over its resources.

GridSim, besides a solution for detecting defects occurred inside a Grid architecture [3], comes with a data Grid extension [4]. This extension, *DataGrid*, implements a data replication model and a mechanism of retrieving data based on a data catalog. The hierarchical proposed model uses root nodes that hold mappings between files and lists of leaf nodes that contain them and leaf nodes that also have a catalog which holds the mappings between files and the local resources list were are replicated.

GridNet [5] can model from tree to ring topologies. In particular, for replication experiments, it considers three types of nodes: a root node (main data source), intermediate nodes (or cache nodes) with larger replication capacity and client nodes (leafs) with smaller replication capacity. The *GridNet* model assumes that between nodes located on the same level there is a ring structure to ensure fault tolerance, structure where member nodes are consistent: if one of them is no longer available, the answer will be given by another node in the ring storing the same set of data. *ChicagoSim* [6] 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 particular classes of experiments. They all support, to some extent, the simulation of data transfer and replication technologies. However, they do not present general models that allow the evaluation of replication in the wider context of different architectures encountered in case of 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 provides the means to evaluate a wide-range of solutions for replication technologies, while still being able to consider characteristics of a wide range of distributed systems architecture [7].

We argue that a correct evaluation of replication technologies 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. The simulation model provided by MONARC is more generic that others, as demonstrated in [7]. 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. A Model for Replication Strategies

The model for evaluating replication solutions was implemented in the MONARC simulator. MONARC is a simulator built around 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 [8]. The simulator is optimized such that to support experiments consisting of thousands of processing nodes executing a large number of concurrent jobs, as demonstrated in [8].

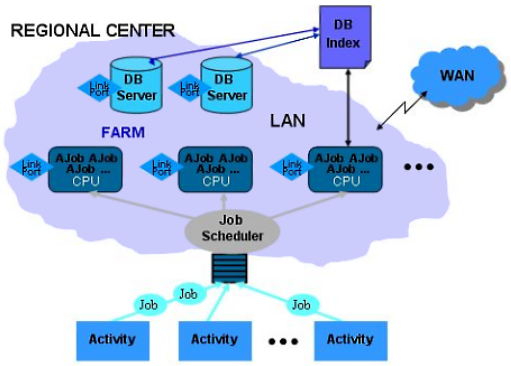

Fig. 1. The Regional center model adopted in MONARC.

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 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 (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.

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 layer contains the simulation engine. Above it there is a layer responsible for modeling the basic components of a distributed system (processing units, jobs, databases, networks, job schedulers, etc.). These are the fixed parts on top of which some particular components (specific for particular types of simulated systems) are built. These particular components can be various types of jobs, job schedulers implementing different scheduling algorithms or database servers that support data replication. 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).

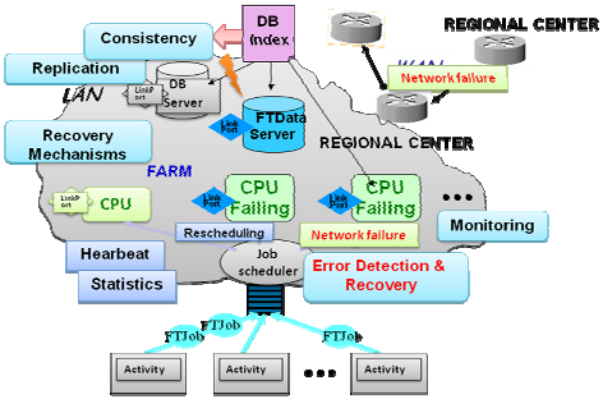

Fig. 2. The extended Regional Center model.

As demonstrated in [1], replication is widely used in distributed systems. It provides capabilities such as fault recovery or load balancing. For evaluating replication technologies designed for distributed systems MONARC was also extended accordingly. For that we adopted a model that is able to describe a wide range of replication data technologies. This model includes components that monitor and implement various algorithms for detecting when failures occur. It includes the capability to simulate permanent or transient failures in processing units, networking or storing devices. It includes components for assuring consistency among replicas. It includes components for emulating the characteristic of various storing infrastructures. The model can be used to model, in this sense, a wide range of fault tolerance solutions for large scale distributed systems.

The model extends all of MONARC's layers. In each layer we added new components or updated the already existing ones. Figure 2 presents the components of the extended model.

In this model 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.

The model includes components and methods necessary to implement various replication strategies. It includes a component for modeling faults and another one for modeling various fault detection strategies. The data model includes a wide range of components, capable of simulating database servers or data containers. This model is able to simulate the behavior of file systems or various database management systems, relational or of other types. The model includes a wide range of components for modeling networking entities and corresponding communication protocols. It includes a component responsible with the adoption of various consistency protocols. All these components are able to experience transient or permanent faults.

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. For example, the fault injection mechanism is "hidden" from the other components that the affected one is communicating with.

4. A Hybrid Replication Extension

Based on the model for replication technologies, we experimented with several components and extensions for replication technologies. We focused on introducing mechanisms to allow for the evaluation of various replication architectures and consistency protocols.

In particular, we present details about a data replication architecture that combines the advantages of hierarchical and total replication solutions. This was implemented in the simulation and, as a case study, we present details about the kind of components that were extended in the simulation model and how we

implemented the replication architecture and conducted evaluation tests on it.

In *hierarchical replication* the data accesses coming from a client (*data_client*) to a data resource (*data_master*) are intermediated, due to low bandwidth, by an intermediate data resource (*data_cache*). This middle component replicates the data closer to the client. When the client accesses again the data, it is served by the intermediate resource with a much higher transfer rate. The hierarchical replication is appropriate for increasing data availability and the transfer speeds.

Another approach, *total replication*, is more appropriate for dealing with fault tolerance. It consists of duplicating data by creating complete copies of the same data source. When an error occurs and a current data source cannot serve the data, another one takes its place and handles further requests. This replication model can be constructed based on a ring or interconnected topology. All entities might have the same set of data and when a resource changes a specific record, the new value is propagated to the other entities.

By combining the advantages of both these replication schemas, we propose a *hybrid replication architecture*. In this architecture data replication is accomplished using an extended tree-like topology. Each data source can be replaced by a ring or interconnected structure of nodes in which each node has the same set of data.

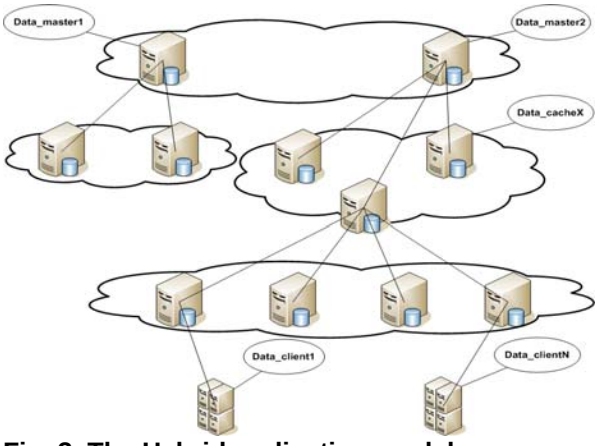

Fig. 3. The Hybrid replication model.

The proposed replication solution also includes a strategy for hierarchical replication, a *data consistency algorithm*, a *failure detection system* and, a *schema for data propagation*.

The *hierarchical replication strategy* is based on a LRU (Last Recently Used) model. In this approach a *data_client* can request data through an intermediate node (*data_cache*). The *data_cache* can immediately serve the data, or if the data is not available locally it will further initiate a transfer request from a superior node. The *data_cache* also saves data for future requests. If the local space is insufficient, it deletes an entry based on the last recently used criteria.

The *data consistency* is maintained using an algorithm based on the leader election paradigm for reaching consensus using Paxos [9]. In this approach consensus is achieved in two phases. In the first phase the leader tries to impose itself to be recognized as a leader. In the second phase it proposes a resolution for solving the consensus problem. It then receives replies from all the entities involved and, depending on the replies, it imposes a decision that the others will fallow (Fig. 4). If meanwhile an error occurs, the flow it reinitiated with another round number.

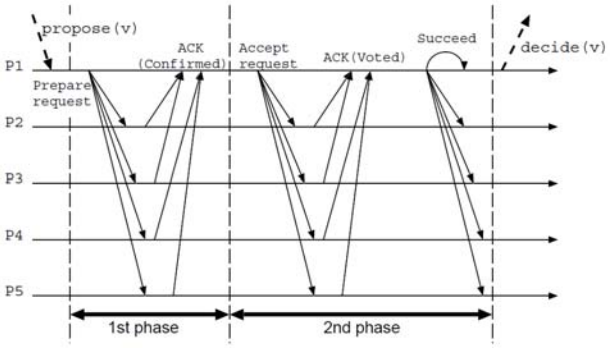

Fig. 4. Phases in a round in the Paxos algorithm.

A necessary condition for implementing the Paxos algorithm is to know the status of the system regardless of any errors that may occur [10]. Thus, to establish consensus, an important parameter is the number of entities that participate in the voting decision. If one participant experience failure, it must not be consider in the voting decision. In this case the number of data resources participating to consensus decreases by one unit. Similarly, when the entity becomes (eventually) available again, it should be reintroduced between the participants in the voting process.

The *failure detection* is implemented using the failure detector based on a heartbeat communication proposed by Chen, Toueg, and Aguilera [11]. Each entity regularly sends messages to other entities to announce their availability. If an entity fails to sends such messages, after a time period it is considered suspect for failure. Similarly, when messages are received again, it is no longer considered suspect.

When the data belonging to a node is updated, the node initiates the *data propagation* schema. The propagation influence all the data resources interested

in the new value. For each descendant node a list of unique identifiers is kept with all records served. Only those descendants maintaining replicas of the updated data will receive propagation messages with the new values. When such a node receives an update message, it can respond with a negative value, meaning that the descending node no longer holds that data and the parent node should remove it's identifier from the list associated to that descendant. A case for this situation is when a node takes a delete action for a data based on a LRU policy.

5. Implementation

The replication model was further implemented in the MONARC simulator. The implementation was made by extending the existing data model [8] and adding new structures. In the model the *DatabaseEntity*, extended by the *DatabaseServer* and *MassStorage* entities, represents a basic modeling of databases. We extended it with the *NodeDBServer* entity (Fig. 5). To implement the replication model we also extended the basic *DContainter*, the atomic unit object which emulates a database file containing a set of objects of a certain type. Also the *MessageData* object was extended. This object represents a message that is used when transferring data between a database job and a database entity.

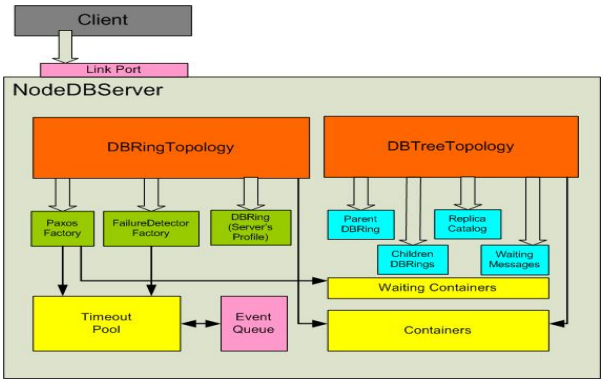

Fig. 5. Components of a NodeDBServer.

In Fig. 5, the *LinkPort* allows communication with other entities, whether ordinary clients or other databases. The *DBRingTopology* maintains horizontal replication logic, and must ensure data consistency of the database with the others situated on the same level.

The *PaxosFactory* maintains a structure with all processes involved in synchronization. It implements the consensus logic and is responsible with treating all messages exchanged in this scope. When a synchronization process ends, it gives a response regarding the new version and the value of the container that was the subject for consensus.

FailureDetectorFactory is a module responsible with the discovering failures in the databases members of the ring. It also handles the sending of heartbeat messages to the other entities. At any time the module provides the status of any database member and the total number of entities considered available; such information is necessary to the synchronization module (the *PaxosFactory* component).

The *DBRing* maintains the profile of the servers and is responsible for the communication schema. The *ParentDBRing* component maintains the profiles of the servers considered parent and *ChildrenDBRings* of the ones considered child nodes. The vertical replication logic is maintained by the *DBTreeTopology* component. Also, this component handles the data consistency algorithm, with the communication between parent and child nodes to receive or send updates of data.

The *ReplicaCatalog* provides a mapping of containers involved in the vertical replication. This component maintains for each child node the identifiers of the containers that have been locally replicated. This data is used to filter the nodes involved in the updating/consistency process.

The *WaitingMessages* component implements a queue of messages currently treated. When a read request cannot be resolved locally, that request is queued and another read request is addressed to a server in the parent ring. When the answer to that request is received, it is correlated to the queued request and is passed back to the first caller.

The *TimeoutPool* implements a queue of *Timeout* objects. When a component of *NodeDBServer* needs to set up a notification, it adds in this queue a *Timeout* object. This component implement, therefore, a notification mechanism that handles interrupt timeouts in the simulation. Components of the *NodeDBServer* that use this pool are *PaxosFactory* and *FailureDetectorFactory*.

The role of the *Containers* is to maintain all records (data chunks) of a database. The *WaitingContainers* maintains all records (containers) that are waiting to follow a synchronization process and then to be eventually written. Consider, for example, the case when a container is in a synchronization process. Meanwhile another application is writing another value and updates the same container. The new container is placed in the queue waiting its turn to be again synchronized. If another application wants to update the same container, the last value from the queue is replaced by the newcomer. When the current synchronization process is completed, the mediated

container is written to disk, and the one from the queue is passed to a new synchronization process.

The *EventQueue* maintains a queue of discrete events that will "happen" in future. This is inherited from class *Task* and extended by *DatabaseEntity*. The queue of events is used to implement the *TimeoutPool* component, which is not based on creating new events but on the arrival times of the existing events: when an event is treated, before that all timeouts are checked to verify if some expired at the moment of its arrival.

In this architecture the messages are sent and received either synchronously or asynchronously. The user can influence several quality of service (QoS) parameters: message loss probability, average message delay, upper bound on detection time, upper bound on average mistake duration, lower bound on average mistake recurrence time.

6. Experimental Results

In order to evaluate the performance of the proposed replication solution we conducted several experiments, starting from the MONARC implementation. We were interested in the characteristics of the replication solution and how it affects the performance of the distributed system where it is used.

6.1. Performance evaluation of the hierarchical replication

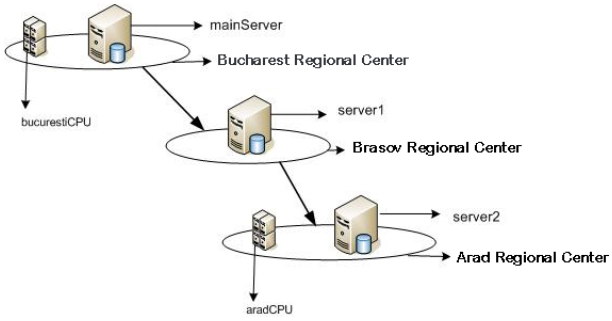

Fig. 6. The Topology used in the Hierarchical Replication Experiment.

The hierarchical replication experiment is based on the topology presented in Fig. 6. The experiment involved three modeled regional centers: Bucharest, Brasov and Arad. It also involved two activities. In Bucharest in the experiment 10 concurrent jobs write containers, each of 500 MB. In Arad the experiment involved the execution of two sets of 50 jobs, one for each processing unit. Each job sequential reads the 10 containers several times. The main storing location of the containers is Bucharest, but in Brasov and Arad data storing servers are also available.

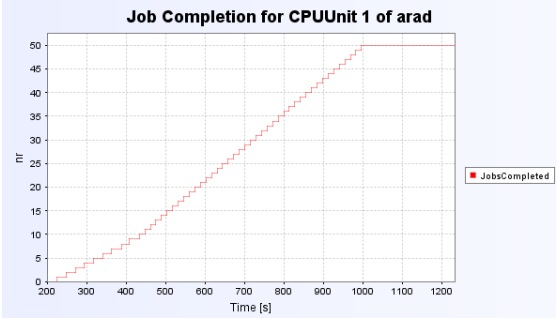

Fig. 7. Number of finished jobs as reported by the processing unit in Arad.

Fig. 7 presents the results obtained for the number of jobs successfully processed on the first processing unit available in the Arad regional center.

In this case we were interested in the functionality of the hybrid replication solution. In this case, the requests were addressed to *server2*, which in turn forwarded them to *server1* in the Brasov regional center. This server forwarded requests further to the *mainServer* located in Bucharest, where the containers are originally kept. This behavior is observed in the first section of the graph. Then, as data is replicated in the local caches of *server1* and *server2*, the results show that the serving time decreases. This shows an improvement in the performance of the system due to this hierarchical approach.

6.2. Performance evaluation of the hybrid replication

The hybrid replication experiment involved four regional centers: Bucharest (containing the main database), Brasov (with the serverCache1, serverCache2 and serverCache3 databases), Arad (with the server1 database) and Iasi (with server3). The database hierarchy is represented in Fig. 8. The servers in Brasov are connected in a ring topology, and between these servers we have total replication. The other servers, connected using the hierarchical approach, involve partial replication.

With this topology we executed two experiments. In the first experiment each processing unit in Bucharest runs jobs that write 10 containers (of 20 MB each) in the *mainServer* database. In Arad and Iasi we have jobs reading 5 containers each, 10 times in a row, from the *mainServer*. In the second experiment the jobs in Arad will request data from the local *server1* database, and the jobs in Iasi from the *server2* database.

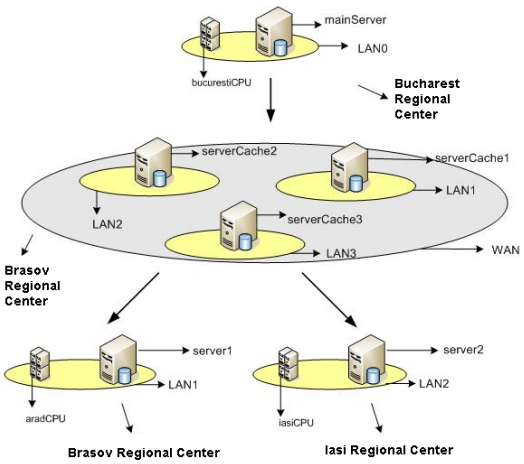

Fig. 8. The hybrid replication experiment.

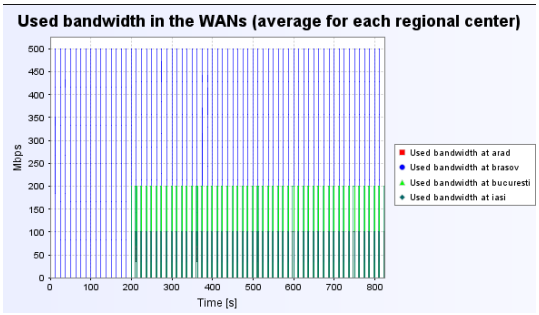

Fig. 9. The throughput in the Regional Centers in the first experiment.

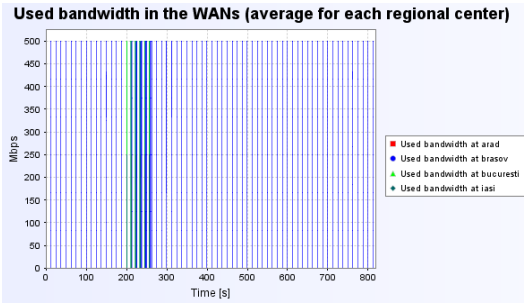

Fig. 10. The throughput in the Regional Centers in the second experiment.

Because the second experiment involves data replication, all successive requests for containers will not be served by the server in Bucharest. Figures 9 and 10 present a comparative representation of results obtained for the activities involved in networking in regional centers in the first and second experiments. In this case the replication solution brings the data where they are most needed (notice the decrease in the traffic involved in the Bucharest regional center), but also lead to an improvement in the performance of the distributed system (the average throughput is lower in the second experiment).

6.3. The capability of the replication solution to handle faults

The fault tolerance experiment involved three Regional Centers: Bucharest (with the *mainServer* database), Brasov (with *serverCache1*, *serverCache2* and *serverCache3*) and Arad (with *server1* database). The hierarchy of the regional centers is presented in Fig. 11. The servers in Brasov are connected in a ring topology and use total replication. In the other cases the experiment involved partial replication.

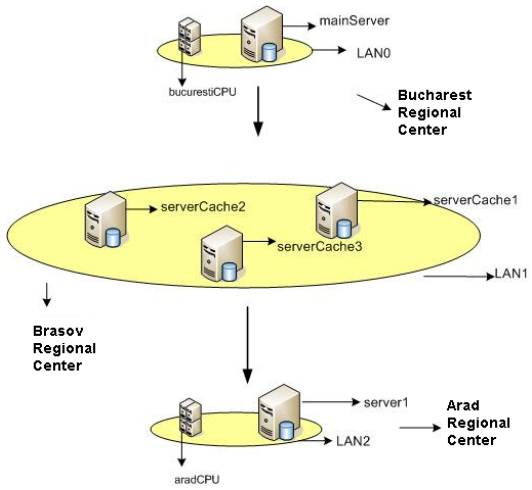

Fig. 11. The topology used in the fault tolerance simulation experiment.

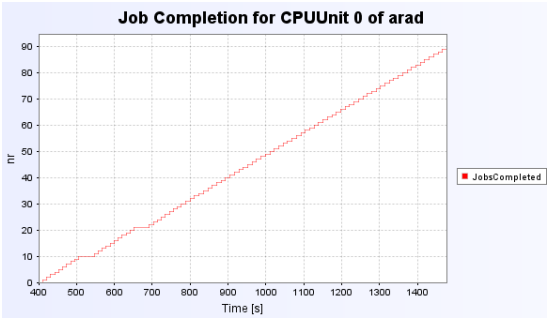

Fig. 12.The number of finished jobs in cpu0Arad.

In this experiment the processing unit in Bucharest writes 30 containers of 20 MB each on the *mainServer* database. The jobs in Arad start with a delay (400 seconds) and read these containers. In this experiment we have two errors. Starting with the simulation time 500, the *serverCache1* database in Brasov stops responding to all requests. Starting with the simulation

time 650, the *serverCache2* database stops also responding to all requests.

In Fig. 12 we notice two regions on the graph where the reading jobs last longer than usual. These two regions correspond to the two moments when *serverCache1* and *serverCache2* stop responding.

In this case *server1* in Arad waits for a longer time period the response from the "parent" database server (*serverCache2* in this case). When the response fails to arrive the request is reassigned to the *serverCache2* database, which in turn becomes the new "parent" and handles all successive requests. The same thing happens when *serverCache*2 fails. This experiment demonstrates the utility of the total replication. The system can tolerate faults by redirecting requests to available alive copies of the data.

7. Conclusions and future work

We presented a replication architecture that combines both partial and total replication strategies to optimize the data access, keep replicas consistent and tolerate faults. This architecture was evaluated using a generic simulation model and the capabilities of the MONARC simulator. The experiment showed that by using the proposed replication architecture for handling data in distributed environment the access times decrease as the data is pulled closer to where it is needed. Also, the throughput is redistributed, and the networking load is balanced among the available server. The main database servers become less loaded and they can handle, as a result, higher data loads. The caching nodes can, in real world, be located on nodes with higher bandwidth capabilities or, if possible, as close as possible to the requester. This leads to data being faster served to jobs. Also, as the requests are served by more than one database server, this approach also leads to scalability and fault tolerance.

As future work we intend to experiment with adaptive failure detection algorithms or experiment with various rules to access containers. Also, we intend to experiment with further consistency algorithms and make extensive comparisons between our proposed replication architecture and other approaches in the context of existing large scale distributed systems such as the ones used in the LHC experiments in CERN.

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.

10. References

[1] H. Lamehamedi, Z. Shentu, B. Szymanski, "Simulation of Dynamic Data Replication Strategies in Data Grids", *12th Heterogeneous Computing Workshop*, Nisa, France, April 2003.

[2] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, F. Zini, "Evaluating Scheduling and Replica Optimisation Strategies in OptorSim", *International Conference on Grid Computing*, Phoenix, Arizona, SUA, November 2003.

[3] A. Caminero, A. Sulistio, B. Caminero, C. Carrion, R. Buyya, "Extending GridSim with an Architecture for Failure Detection", *The 13th International Conference on Parallel and Distributed Systems*, Hsinchu, Taiwan, December 2007.

[4] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, R. Buyya, "A toolkit for modelling and simulating Data Grids: An extension to GridSim", *Concurrency and Computation: Practice & Experience*, 20(13), September 2008.

[5] H. Lamehamedi, Z. Shentu, B. Szymanski, "Simulation of Dynamic Data Replication Strategies in Data Grids", *12th Heterogeneous Computing Workshop*, Nisa, France, 2003.

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

[7] 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.

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

[9] N. Hayashibara, P. Urbán, T. Katayama, A. Schiper, "Performance Comparison between the Paxos and Chandra-Toueg Consensus Algorithms", *Technical Report* IC-2002-61, 2002.

[10] Y. J. Song, R. van Renesse, F. B. Schneider, D. Dolev, "The Building Blocks of Consensus", *The 9th International Conference on Distributed Computing and Networking*, January 2008.

[11] W. Chen, S. Toueg, M. K. Aguilera, "On the Quality of Service of Failure Detectors", *IEEE Transactions on Computers*, Vol. 51, pp. 561-580, 2002.