

**Dependable services, built on group  
communication systems, providing fast  
access to huge volumes of data in  
distributed systems**

PhD Thesis  
Extended abstract

PhD Student  
ing. [Adrian Coleşa](#)

Scientific advisor  
[Prof. dr. ing. Iosif Ignat](#)

[Department of Computer Science](#)



2010

## 1 Introduction

This thesis presents some methods aimed to provide high-availability of remote data managed by a specialized software system. The data management system consists of more services, which cooperate each other to provide high availability of data and fast access to it. Getting this characteristics of data implies the system's services being dependable and also using efficient methods to provide access to data.

We considered the case of huge volumes of data and great number of clients of the data system. The high data availability is a common requirement in such a context. These requirements result, on one hand, from the fact that even normal users hold huge volumes of personal data and, on another hand, from the fact that many local data and applications tend to be moved remotely on dedicated sites managed by specialized companies.

The dependability of a service is its capacity to function according to its specifications in any conditions, even in cases some of its components fail. The dependability is given by other service's characteristics, like fault-tolerance, reliability, scalability, availability etc. Providing such characteristics presumes the use of some redundant components of the service. The most used technique of redundancy is replication. Both data and software components are replicated in order to obtain the needed redundancy. Replication also implies distribution of service components on more different nodes. In order to get a better price-performance ratio, off-the-shelf hardware components are used. Unfortunately, such components have a low degree of dependability. That means that services' dependability must be obtained from the software redundancy. Yet, this is not a simple problem, especially in asynchronous systems, where one cannot consider time limits on message transmissions and process scheduling.

All the above considerations show that providing service dependability is a real requirement, but it is not so simple to obtain it in an asynchronous distributed environment. A lot of research has been done in this direction in the last two decade and many solutions have been proposed in the meantime. Yet, we identified problems not completely or satisfactorily solved. That is why we lead our research in this direction.

Firstly, we noted the even if there are many solutions proposed to the problem of dependable services and high-availability of data, most of them deal with particular aspects related to high-availability and just few of them have a global overview of the problem. Consequently, one of our main purposes was to develop a system able to manage all the problems related to high-availability of data it provides access to. We think our system such that it allows combination of different existing techniques used to provide fast and reliable access to data and high-availability of services composing the system itself. We proposed in Chapter 2 the detailed architecture of such a data access system, composed by more services, each one responsible for different aspects related to data availability, although collaborating each other in order to get global behavior of the system. We also proposed in Chapter 3 a strategy to provide dependability of services, requiring no collaboration from the service itself and totally transparent for the service's clients. Specifically, the type of service we treated was a centralized service, which could have not been modified, still it must be dependable, a characteristics that obviously it has not in its initial centralized implementation just one server. The strategy we developed run the centralized

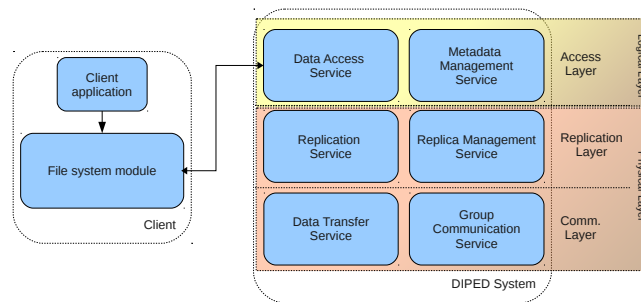


Fig. 1: DIPED's Architecture

service in a distributed manner, replicating its unique server. In Chapter 4 we proposed a way to flexibly organize data in order to find quickly the desired data in huge spaces of data. The data we considered is stored in files and the method we proposed associated metadata to data in order to establish relationships between files, relationships based on which files or sets of files can be found very fast.

## 2 DIPED: providing high-availability of data

This chapter presents the detailed architecture of a data management system, which aims to provide high-availability of data it manages. Its architecture is illustrated in Figure 1.

DIPED is composed by more services. The services are organized on two layers: logical and physical. The services at the logical layer deal with aspects related to data organization and finding. The way they provide high-availability of data is based on convenient, fast and efficient techniques for finding and selecting the desired set of files in a huge file space (file system). The services at the physical layer deal with data replication and replica management, in order to provide fault-tolerance and fast access to files contents. The two-layer organization of our system provides to the system's clients transparency related to the physical location of files and to the fact that they have more replicas. All the aspects mentioned above — efficient organization of files, fast finding of desired files, fast access to their contents, fault-tolerance — contribute to what we call high-availability of data managed by our system.

We described the detailed functionality of each service of DIPED. We also established the way the services interact each other. In fact, two important contributions we introduced result from this aspect.

Firstly, every file transfer, both to clients' node — when provide them with the requested data and storage nodes — when data is replicated, is considered to be a data replication and is managed by the replication service. The replication operations are initiated, although, by other services, depending on the type of transfer: by the data access, in case of transferring data to clients, and by the replica management service, in case of creating new replicas on other replication nodes. This strategy makes possible the involvement of clients, if they accept that, when data is provided to other clients, in order to increase the performance and fault-tolerance of this operation. We called such clients involved clients and designed their architecture.

Secondly, we proposed a replication schema, based on replication patterns associated to each storage (replication) node. The replication patterns indicate the data sets required or accepted to be stored on the corresponding node. They are described in terms of the metadata management service’s language, at the logical level, but are used by the replication service, at the physical level. The schema organized replication nodes in groups that intersects each other such that they form a hierarchy. The hierarchy is used to make the replication operation scalable both when new data is introduced in the system and some degree of redundancy must be provided for it, and when some nodes crashes and the degree of redundancy must be preserved. We proved the schema works correctly, which means that some new data generated or stored on some node will be automatically replicated on any other node whose replication pattern indicate that data as being desired on it. The prove is based on the fact that there always exists a sequence of replication groups that links two different groups, containing the node on which the data is generated and a node interested in that type of data. Any two consecutive nodes in such a sequence contain in their intersection at least one node whose replication pattern matches the data that must be replicated.

We implemented the DIPED system in Grid using the Globus Toolkit 4 data services. Some of the GT4 services matches very well our proposed services, yet the other must be implemented to provide the needed degree of fault-tolerance and to automate the replication operation. We also tested the correct functionality of the system for huge transfers of data.

Ideas and parts of these chapter were published as papers in proceedings of different national and international conferences: [1], [2], [3], [4], [5] and [6].

### 3 Strategies to transparently make a centralized service dependable

We dealt with making dependable an existent centralized service, which could not be modified. Thus, the service implementation consisted in just one server, running on just one physical node, a situation that obviously could not provide dependability of that service. The dependability of a service is given by its fault-tolerance, scalability, availability. So, we tried to make the service behaving like this. The main technique we used to provide all the mentioned properties was replication, i.e. we replicated the unique server of the service on other physical nodes. However, we could not rely on any sort of collaboration from the service’s server itself, but we had to find a way to run the service in a distributed manner totally transparent from the service itself. Also, the client transparency was an important requirement. The the system we developed to provide service’s dependability, were practically independent of and invisible to both the service itself and its clients.

The only possible way to do this was to run the service’s unique server into a virtual machine, which was run in its turn on a physical node, and replicating the virtual machine on several other physical nodes. Figure 2a illustrates this technique. Because of the non-cooperation of the service, the only replication method that we could use was the primary-backup one. The replication is applied continuously in consecutive phases (see Figure 3), each phase consisting of the following operations: running the primary virtual machine for a time  $t_R$ ,

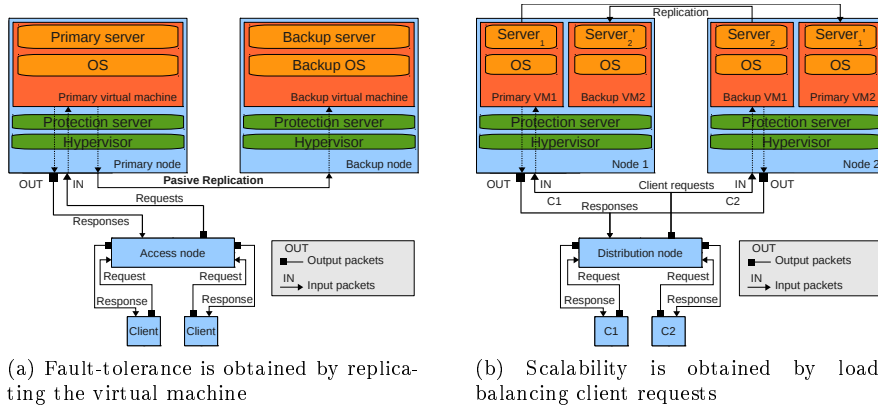


Fig. 2: Making a service dependable using virtualization and replication

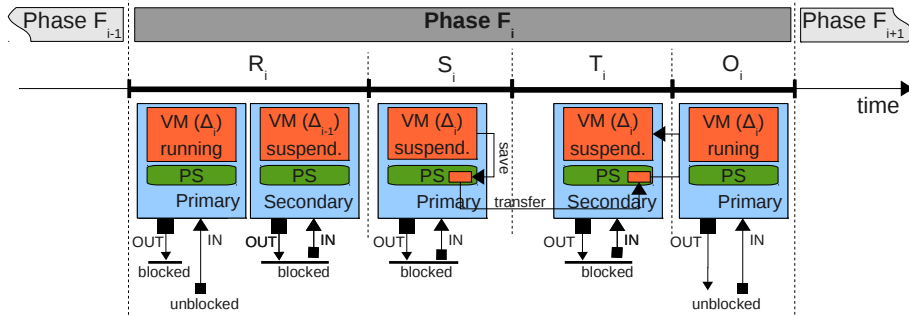


Fig. 3: Operatiile unei faze ale protocolului de replicare

suspending the primary virtual machine and saving locally its state (an operation taking  $t_S$ ), transferring the saved state on backup nodes, thus updating the state of the backup virtual machines (an operation taking  $t_T$ ) and releasing the corresponding output, i.e. network packets, generated by the primary virtual machine during the current phase's running operation. The virtual machine's output is blocked at the beginning of a new phase, in order to make the new state of the primary machine (which becomes different by those of its backups during the running operation) invisible to its clients, until it is replicated on the backup nodes. That way, any time the primary machine crashes, it can be replaced by any one of its backups, totally transparent to the service's clients.

The passive replication strategy did not permit us to balance the load of the service on the many backup virtual machines hosting service's replicas, because only one was active (the primary one). We needed although applying load-balancing in order to make the service scalable to the number of client requests. We succeeded developing a load-balancing strategy, but it is merely applicable for a limited number of service's types, i.e. services whose clients' requests can be handled totally independent on each other. For instance, if each client request is a read-only one (e.g. an HTTP service) or if different clients do not interact or influence directly each other (e.g. a mail service). The load-balancing strategy

Name	Type	Properties		Values
		<i>Author</i>	<i>Year</i>	
Book1	file	Author1	2007	
Book2	file	Author1	2008	
Book3	file	Author1	2009	
Book4	file	Author2	2007	
Book5	file	Author2	2007	
Book6	file	Author2	2007	
Books2007	category		2007	

Tab. 1: File category *Books*

consists in running multiple independent tuples of primary-backup servers of the service placed in different virtual machines. Each tuple handles requests of a set of clients totally disjoint from the sets of clients handled by the other replication tuples. Figure 2b illustrates the load-balancing strategy.

We analyzed and compared two ways of applying the replication: synchronously and asynchronously. In order to do this, we calculated the delays they introduce in the response time to client requests. Based on this comparison we developed an adaptive replication algorithm, which considers the real-time service’s behavior and the capacity of the network links between the primary node and the backup ones. When the links’ capacity degrades, either because the service modifies a great number of memory pages or a network overloading or outage occurs, the adaptive algorithm modifies dynamically the running time interval ( $t_R$ ) of each replication phase, in order to keep the additional delay introduced by the protection system bellow or at least as near as possible the acceptable maximum delay. It takes its decisions detecting the locality-of-modification periods of the running service (and implicitly of the virtual machine) and accounting the number of output network packets and their actual delay.

We implemented and tested our replication strategy over Xen hypervisor.

We also developed, implemented and tested a strategy to dynamically update the implementation of Linux modules during their use. The strategy contributes to the operating system dependability and implicitly to that of the service running on the corresponding operating system.

Ideas and parts of these chapter were published as papers in proceedings of different national and international conferences: [7], [6] and [8].

## 4 Flexible Organization and Fast Finding of Data in File Systems, Based on File Relationships

High-availability of data in case of huge volumes of data (i.e. a great number of files) is also given by the ability of the data management system to provide ways of finding very fast the desired sets of data. This was the reason that conducted our work to develop data organization and finding methods to be used by the metadata management service of the DIPED system, we described in Chapter 2. We integrated the developed methods in a file system, in order to provide transparency and a uniform interface to all user applications.

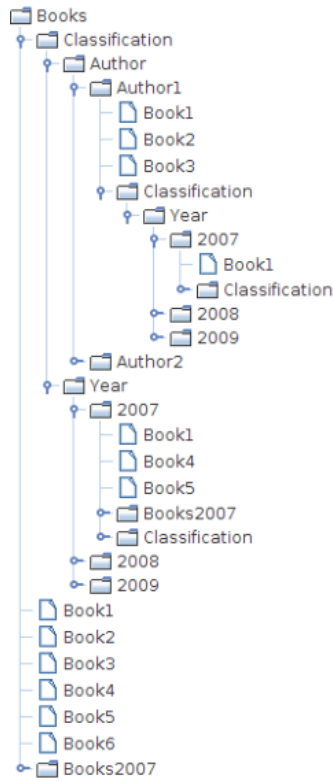


Fig. 4: File tree generated for *Books* category

We based our strategy on the relational database model, which we integrated into a traditional file system, introducing new concepts that map the traditional file system concepts on those specific to relational databases. We associated properties (metadata) to files and used them to establish relationships between files. Based on the files' metadata, we introduced the concepts of file category, class and dependencies. The most developed concept was the file category, which is a container consisting of files with the same set of properties, each file having different values (none, one or more) associated to each property. Table 1 illustrates such a category.

Our system provides the category contents as a file tree, generated based on category properties and the corresponding values associated to the files in that category. Each possible property and value is provided as a virtual folder, the resulting file tree consisting of the paths corresponding to all possible combinations of  $(property, value)$  pairs in the category. Figure 4 illustrates the file tree generated for the *Books* category.

Our system treats equivalently at a category level files and subcategory. This allows us to organize, view and find files in many different ways, based on the set of metadata (classification criteria) that were considered at one moment.

The way the files can be found is based on methods specific to both database model, i.e. interrogation, and traditional file systems, i.e. navigation. We combined both of them to allow refining interrogations by navigating in the file tree corresponding to their result. That way, the user can specify only the properties and values he knows about, the other necessary ones could have being found by looking and navigating in the interrogation's result. We built a simple interrogation language, based on logical formulas. The way the interrogations can be formulated is illustrated in the example below.

```
> cd Books/Classification
> cd 'Author=Author1&Year=2007'

> cd Books/Classification
> cd 'Author=Author1&Year=2007|Year=2008'

> cd Books/Classification
> cd 'Author=Author1!Year=2007'
```

We implemented and tested the proposed system in Linux, like a user space

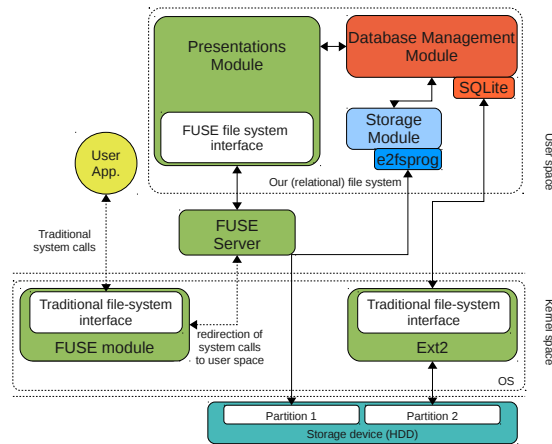


Fig. 5: File system architecture in Linux user space using Fuse

application, using the FUSE mechanisms. The Figure 5 illustrates its architecture, position and interactions with the system.

Ideas and parts of these chapter were published as papers in proceedings of different national and international conferences:[9], [5] and [10].

## 5 Conclusion

The main objective of this thesis was to develop different techniques to provide high-availability of data. We developed and described in chapter 2 DIPED, a data management system to provide this property. We also developed specific methods used by DIPED, which we described in chapters 3 and 4.

The main personal contributions in chapter 2 are:

1. identification and detailed presentation of the main problems related to high-availability of data, needed characteristics of the data management and access system;
2. the detailed and complete architecture of a data management system, called DIPED, which integrate different strategies to provide high-availability of data;
3. the strategy of dealing any data transfer as a data replication operation, dealt with by the replication service in collaboration with metadata management and replica management services;
4. the scalable replication schema, based on replication patterns and the prove of its correct functionality;
5. Grid implementation and testing of DIPED system;

The main personal contributions in chapter 3 are:

1. The identification of the only possible solution to make a centralized service dependable: run the service in a virtual machine and replicate the entire virtual machine.

2. The load-balancing strategy applicable to few types of services run in virtual machines;
3. The mathematical formulas for the delay introduced by our system in the client requests' response time and the comparison between the synchronous and asynchronous replication strategies.
4. The adaptive replication algorithm, which reduce as much as possible the delay introduced in response time.
5. The Xen implementation and testing of the synchronous and asynchronous replication strategies.

The main personal contributions in chapter 4 are:

1. The way the relational database model was integrated into file systems, introducing new concepts like file properties, category, multiple views and mapping them on the traditional folder concept.
2. The uniform way of dealing with files and subcategories as elements of a category, allowing classification of both of them.
3. The transparent integration of the proposed file system in Linux, such that it is compatible with the existent applications (using the traditional file system interface) and accessible to them.

The main future research and development directions are:

1. detailed design and implementation of an adaptive data replication strategy, based on the combination of the data push and pull techniques and the actual characteristics of the links between the replication nodes;
2. the implementation and testing of the adaptive replication algorithm used in the system that provide transparent service dependability;
3. implementation of a copy-on-write technique, to eliminate the saving operation in the replication system;
4. implementation of the replication algorithm for WANs;
5. implementation of all the proposed file relationships;
6. development of a specialized file system interface and browser application to benefits at maximum by the advantages of the relational file system;
7. implementation of the proposed file system inside the operating system.

## Publications

- [1] A. Coleșa, “Probleme de fiabilitate, scalabilitate și disponibilitate a serviciilor în sistemele grid,” in *Volumul 1 al Atelierului de lucru MedioGRID*, Cluj-Napoca, Romania, Decembrie 2005, pp. 169–183. 3
- [2] A. Coleșa and I. Ignat, “Fiabilitatea și scalabilitatea serviciilor de replicare a datelor în sistemele grid,” in *Volumul 2 al Atelierului de lucru MedioGRID*, Cluj-Napoca, Romania, February 2006, pp. 160–180. 3

- [3] A. Colesa, I. Ignat, and R. Opris, “Providing high data availability in mediogrid,” in *Proceedings of The 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '06)*. IEEE Computer Society, 2006, pp. 296–302. [3](#)
- [4] A. Colesa, T. Pop, I. Ignat, and C. Ardelean, “Automatic and reliable distribution of data in grids over globus toolkit,” in *Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC07)*. IEEE Computer Society, 2007, pp. 310–316. [3](#)
- [5] A. Coleșa, V. Cionca, A. Tața, and I. Ignat, “A meta-data enhanced file system,” in *Proceedings of the 3rd IEEE International Conference on Intelligent Computer Communication and Processing (ICCP07)*. IEEE, September 2007, pp. 267–270. [3](#), [7](#)
- [6] A. Colesa, B. Marincas, I. Ignat, and C. Ardelean, “Strategies to transparently make a centralized service highly-available,” in *Proceedings of the 5rd IEEE International Conference on Intelligent Computer Communication and Processing (ICCP09)*. IEEE Computer Society, 2009, pp. 296–302. [3](#), [5](#)
- [7] Y. Balde, A. Coleșa, and I. Ignat, “An indirect hotswapping system for linux kernel modules,” in *Proceedings of the 3rd IEEE International Conference on Intelligent Computer Communication and Processing (ICCP07)*. IEEE, September 2007, pp. 270–276. [5](#)
- [8] A. Coleșa, I. Stan, and I. Ignat, “Transparent fault-tolerance based on asynchronous virtual machine replication,” in *Proceedings of The 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '10)*. IEEE Computer Society, 2010, to appear. [5](#)
- [9] A. Coleșa, I. Ignat, Z. Majo, and V. Cionca, “A meta-data enhanced file system using the classical interface,” in *Proceedings of The 10th International Conference on Applied Mathematics and Computer Science*, May 2006, pp. 100–106. [7](#)
- [10] A. Coldea, A. Coleșa, and I. Ignat, “Orcfs: Organized relationships between components of the file system for efficient file retrieval,” in *Proceedings of The 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '10)*. IEEE Computer Society, 2010, to appear. [7](#)