Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

ELSEVIER

J. Parallel Distrib. Comput. 65 (2005) 1134–1145

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

A new formalism for dynamic reconfiguration of data servers in a cluster

María S. Pérez*, Alberto Sánchez, José M. Peña, Víctor Robles

Department of Computer Architecture and Technology, Politechnical University of Madrid, Madrid, Spain,

Available online 20 June 2005

Abstract

The use of parallel file systems constitutes a high-performance solution to the problem known as I/O crisis in parallel or distributed environments. In the last years, clusters have become one of the most cheap and flexible frameworks for the deployment of parallel and distributed applications. Both parallel file systems and clusters have been successfully used in several scenarios, where it is possible to share and access data in an efficient way. In fact, clusters provide a huge number of advantages to this kind of systems, the wide availability of tools integrated with them being one of the most important. Nevertheless, clusters and, in general, high-availability distributed systems are characterized to be dynamically modified. Operations such as the addition or elimination of nodes are typical in a cluster environment. Therefore, it is necessary to use new approaches for the dynamic reconfiguration of the nodes that belong to a cluster. This paper describes a mathematical formalism for achieving high-performance and dynamic reconfiguration of data-based clusters with service maintenance.

Keywords: Dynamic reconfiguration; Clusters; Parallel I/O; Data servers; Distributed systems

1. Introduction

Clusters are rapidly becoming a standard platform for high performance and reliable computing. The main reasons are their low cost, high performance, and the flexibility of their off-the-shelf hardware components.

In the last years, a huge number of software tools and applications have been developed for being used in clusters of workstations. One significant example of these applications is the development of parallel file systems oriented to clusters, such as parallel virtual file system (PVFS) [4] or multiagent parallel file system (MAPFS) [12]. These systems combine the advantages of parallel file systems and clusters.

MAPFS is a multiagent file system that provides an efficient access to data stored in the server-side of cluster architecture. MAPFS is based on a client module able to interact with different traditional or distributed servers, providing them parallel I/O features. It presents several advantages:

- MAPFS is a multiagent-based architecture for high-performance I/O in clusters.
- It is easily integratable with conventional distributed systems, since MAPFS is based on these kinds of systems. MAPFS also makes the coexistence of distributed and traditional partitions possible.
- The building of the parallel file system is simpler, because it is based on an existing file server, whose performance has been widely tested. This approach is different from most of the current parallel file system, which are built from scratch, both client and server sides. This last feature makes systems more difficult to integrate in distributed environments.
- MAPFS allows applications to access in a parallel way to both data of different files and data of the same file, which reduces the bottleneck that constitutes the access to conventional servers.
- MAPFS improves the use of system resources, because data distribution among different servers leads to a better load balancing.

* Corresponding author.
E-mail addresses: mperez@fi.upm.es (M.S. Pérez), ascampos@fi.upm.es (A. Sánchez), jmpena@fi.upm.es (J.M. Peña), vrobles@fi.upm.es (V. Robles).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.04.018

- MAPFS fits the heterogeneous nature of a distributed system, because it can use servers with different architectures and operating systems.

Nevertheless, clusters are usually dynamic environments, characterized by the use of operations that modify their configuration. For this reason, MAPFS needs to use a new approach for solving the problem of data management in a changing environment, where clusters are reconfigured dynamically and, therefore, data servers are modified. This approach is the definition of the *storage groups*. This paper describes the concept of storage group and its deployment within MAPFS file system. The main requirement is that MAPFS must keep its service during the use of reconfiguration operations.

The outline of this paper is as follows. Section 2 describes the problems of data intensive applications, which we need to address by means of a flexible I/O architecture. Concretely, we analyze the needs of reconfiguration of the applications and the problem of keeping the data service during the phase of the dynamic changes. In this section, we also describe other approaches used for the dynamic reconfiguration. Section 3 presents the formalism of storage groups, which is used for the dynamic management of servers in MAPFS. Section 4 shows the results obtained for the evaluation of the use of storage groups, analyzing different aspects related to them. Section 5 presents the advantages and differences of storage groups versus other approaches. Finally, Section 6 summarizes our conclusions and suggests further future work.

2. Problem statement and related work

2.1. Data-intensive applications and their I/O needs

The emergence of applications with greater processing and speedup requirements, such as *grand challenge applications* (GCA), involves new computing and I/O needs. Many of these applications require access to huge data repositories and other I/O sources, being the I/O phase a bottleneck in the computing systems, due to its poor performance. Existing data-intensive GCA have been used in several domains, such as physics [8], climate modeling [9], biology [15] or visualization [5]. The I/O problem is not solved in these kinds of applications. New approaches are required in this scene.

On the other hand, the use of clusters constitutes one of the most successful and stable distributed solutions. In fact, clusters have become a cheap and flexible solution to the deployment of parallel and distributed computing. Furthermore, a huge number of systems and applications have been developed for being used in this kind of infrastructures. These applications need tools for managing and configuring properly the cluster, and concretely, the data servers. One of the most important problems in this scenario is providing service maintenance during the changes in the hardware and software infrastructure. MAPFS offers different tools for giving this service.

2.2. Related work

With respect to approaches used for the dynamic reconfiguration of servers, there are different alternatives similar or related to the MAPFS storage groups, defined in Section 3.1. Two different I/O architectures are analyzed here. The xFS file system [1] defines the concept of stripe group as subsets of storage servers and the GFS file system [13] defines network storage pool (NSP) as a set of physically shared devices.

A stripe group in xFS is a set of storage servers. xFS distributes data through all the storage servers, implementing a software RAID. xFS uses a striping system, based on logs, in a similar fashion as Zebra [7]. The main goal of stripe groups is reducing the problem of the scalability, which appears when the stripe is made over a huge number of disks. Grouping servers, data distribution is made without losing performance. In fact, Zebra, which does not use grouping, is limited by the maximum number of servers that can be used properly. Like MAPFS, xFS provides dynamic reconfiguration of servers, in the case a node leaves or joins to the architecture. For managing the way in which stripe groups works, xFS uses a structure named *stripe group map*, which stores information about every stripe group. Moreover, xFS defines two kind of groups, current groups and obsolete groups. When a server leaves or joins the system, xFS modifies the map, in such way that every active server belongs exactly to one of the current storage groups. If this reconfiguration changes the ownership of a concrete group, xFS does not delete the old entry of the group. Instead of this, xFS marks such entry as "obsolete". Clients only write in current groups, although can read from both current and obsolete groups. Thus, there is no data transference from obsolete groups to current groups. On the other hand, in systems based on logs, a *cleaner process* is responsible for eliminating obsolete entries. This process transfers data from obsolete groups to current groups along the time. When this process moves the last item from an obsolete group, xFS deletes its entry of the stripe group map.

As we mentioned previously, a NSP or pool in GFS is a set of physically shared devices. Subpools are divisions of the NSP according to the features of the devices. These features are the latency and the bandwidth. A subpool of high bandwidth devices contains devices attached to clients with one or more high bandwidth links. Meanwhile, a subpool of low latency devices is composed of solid state devices. A GFS implementation can exploit several performance features, using different subpools. For instance, GFS can locate often referenced files in low latency subpools and large files in high bandwidth subpools. The other choice is locating data and metadata in different subpools, data in high bandwidth subpools and metadata in low latency subpools, in order to increase the performance of I/O operations. Additionally, *resource groups* (RG) are defined in GFS as groups that distribute system resources through a NSP. There are multiple RGs per device. RGs make the location of files in different subpools easier. Advanced users or some specific applications can exploit the parallelism through the file transference between RGs. File migration can be used for making a good load balancing between different devices.

3. Proposed approach

3.1. Storage groups

The concept of grouping is fundamental in every aspect of the life. Edwin P. Hubble, who is considered the founder of the observational cosmology, said in the 1930s that the best place for searching for a galaxy is next to another one, describing the concept of galaxy grouping. Like in real life, computer science has a significant number of groupings, such as process group or user group, which are used for representing sets of objects from the computing field.

A *storage group* is defined in MAPFS as a set of servers clustered as groups providing data storage capabilities. These groups take the role of data repositories and can be built by applying several policies, trying to optimize the access to all the storage groups.

A file is said to be associated to a storage group if file data are distributed among the servers belonging to such storage group.

Providing dynamism to the servers management is one of the main goals of the definition of storage groups, in such a way that we can add and modify dynamically servers to existing or new groups. In fact, the main advantages of storage groups are:

(1) Logic abstraction of the concept of storage server: As a partition is a logic abstraction of the physical disk, the storage group is also a logical abstraction of the storage server concept.
(2) Dynamic management of servers: As we mentioned previously, the use of storage groups provides dynamic management of servers through the MAPFS interface.
(3) Efficiency of the storage operations: Policies used in the system provide a way of increasing the global efficiency of the system.
(4) Load balancing: It is possible to use a concrete storage group in order to optimize system load balancing, depending on the load of the remaining storage groups.
(5) Transparent migration: In addition to the MAPFS interface, the system can change the distribution of storage groups in a transparent way in order to optimize different aspects related to the system performance.

In a first step, storage groups are considered as a *partition* of all the servers of the system.

Let be a set of servers $\mathbf{S} = \bigcup_{i,j} S_{i(j)}$. We define the set $\mathbf{G} = \{G_1, G_2, \ldots, G_n\}$ of storage groups. Every group G_i contains a set of servers $S_{i(j)}$, that is,

$$\begin{aligned} G_1 &= \{S_{1(1)}, \ldots, S_{1(m)}\} \\ G_2 &= \{S_{2(1)}, \ldots, S_{2(r)}\}, \\ &\vdots \\ G_n &= \{S_{n(1)}, \ldots, S_{n(l)}\}. \end{aligned} \tag{1}$$

The function used for building this association between servers and storage groups is called *grouping function* and is written as $\lambda_G : \mathbf{S} \rightarrow \mathbf{G}$. In short:

$$\lambda_G(S_i) = G_j \iff S_i \in G_j,$$

where λ_G must be defined in the domain of $\mathbf{S}$.

Let R_G be a relation defined over the cartesian product of the elements of $\mathbf{S}$, which is called *grouping relation*. The relation R_G between servers S_i and S_j is defined as

$$S_i R_G S_j \iff \exists t / S_i \in G_t \wedge S_j \in G_t.$$

Proposition 1. *If the grouping function is an injective application, then the relation R_G is an equivalence relation, which defines the partition of Eq. (1), that is, R_G is a partition of the set of servers $\mathbf{S}$. Therefore, this relation has reflexive, commutative and transitive properties.*

3.1.1. Limitations of the definition of storage groups as partition

Defining storage groups as a partition of the set of all the servers has as main advantage the simplicity of this kind of relation, because a server only belongs to a single group. Thus, the estimation and optimization of different parameters related to the storage groups is very simple. Furthermore, it is possible to achieve load balance in the system, because a storage group is completely independent of the rest of storage groups and thus the load can be easily calculated. If the structure of the storage groups is flexible, the optimization of such parameters could be a NP-hard problem.

Nevertheless, these kinds of storage groups do not represent the dynamism of the system correctly and thus they are not suitable for representing situations in which several servers may join to existing groups or change its group. In this case, it is necessary to specify what will happen with files stored in such a server. Because of this, it is necessary to extend this model with a new formalism that provides more flexibility to the operations of storage groups and servers.

In order to define this new model, we are going to analyze different scenarios:

(1) Creation of a storage group from empty servers: In this case, the creation of a group is previous to the creation

of files or directories in its servers. Therefore, the group is configured before the distribution of the information.

(2) Creation of a storage group from non-empty servers: A storage group is composed of servers with files. To redistribute this information among the components of a new group is a very inefficient choice. For this reason, it is necessary to deal with files distributed among different number of servers within the same storage group.
(3) Joining a server to an existing storage group: This case constitutes a generalization of the previous scenario. As in the previous case, the redistribution is not a valid option, because it is very inefficient.
(4) Creation of a storage group from two existing groups: This case is a generalization again of the previous scenario. Different groups with different distributions are joined in a new group. The redistribution is not used either.
(5) Elimination of a server from a storage group: In order to keep the stored information, it is necessary to redistribute the information among the rest of the components of the storage group. It is advisable to make this operation in low-load hours.
(6) Elimination of a storage group: The information of a storage group must be redistributed in other group. This operation must also be made in low-load hours.

Next, all these situations are going to be analyzed.

3.1.2. Creation of a storage group from empty servers

The data distribution over a new storage group is made through a set of nodes initially empty, and therefore, all the files are distributed in a homogeneous way among such nodes. This situation does not involve any problem in a dynamic environment.

3.1.3. Creation of a storage group from non-empty servers

This section deals with the creation of a storage group from servers that contains files, distributed or not. The difference between the previous case and this one is that in this last case, at first sight, it is necessary to *reconstruct* the information if files must be homogeneously distributed among all the servers belonging to their storage group. The problem of the reconstruction of information is dealt in the following section, because it affects all the situations in which the topology is changed. We will see that the reconstruction is not a feasible solution in most of the situations.

3.1.4. Reconstruction of the information

One of the most important aspects related to the distribution of the information is the way in which the reconstruction of the information is performed when a node is added to the initial topology or a new storage group is created from non-empty servers.

The first alternative is to reconstruct the information *by brute force*, which consists of the following steps:

(1) read all the files,
(2) write the data blocks in new files taking into account the new topology,
(3) delete old files.

This option is very inefficient. Because the addition of nodes is feasible in high-performance environments, the change of topology is a very usual operation. This fact together with the facilities offered by clusters for reconfiguration [3] implies that this technique is not a suitable solution.

On the other hand, when the topology is changed, the I/O system needs to know these changes. The advantage of redistributing the information is that these changes are transparent for the I/O system.

Storage groups provide an intermediate solution, in such way that the data redistribution is only made when a node is deleted from a existing storage group. To avoid the redistribution, when a node is included in the topology, a new storage group is created, including all nodes plus the new node. When a node disappears from the topology, the information must be redistributed among the new topology. Nevertheless, this reconstruction is made within the storage group.

Therefore, storage groups simplify the reconstruction in two different ways:

(1) Reconstruction operations are limited to a single storage group.
(2) In principle, the reconstruction is only made when a node is deleted.

Nevertheless, avoiding the reconstruction in the addition operation has a collateral effect: if the topology is often changed, a great number of storage groups are created with small differences between them. To tackle this disadvantage, the system provides storage groups defragmentation, with the same goal than the disk defragmentation [14], avoiding the fragmentation of data among lots of groups. This operation is implemented by MAPFS and is named `mapGroupDefragm()`. This operation will be described in depth in Section 3.2.1.

Therefore, in order to create a storage group from non-empty servers, avoiding the information reconstruction, we will create as many group as different files distributions plus a group with all the servers, named *main group*.

Definition 1. A file distribution is the list of servers $(S_1, S_2, \ldots, S_n)$ in which data are homogeneously distributed or sliced.

Definition 2. A main group that belongs to a set of servers is a storage group that includes all the servers of such a set and has the same distribution for all the files that stores.

Typically, if we join n servers without common files, it is necessary to create $n+1$ storage groups, one per server and other more, the main group, which is initially empty.

The main group is visible by the applications; the rest of the groups are used exclusively for avoiding the system degradation, as we described previously. For this reason, the rest of the groups are named *invisible groups* or *secondary groups*.

Definition 3. An invisible group or secondary group is a storage group that is not a main group of a set of servers.

Observation 1. The main group that belongs to a set of servers is a superset of all the storage groups which include some of the servers.

3.1.5. *Joining a server to an existing storage group*

To add a new node or server is a usual scenario in clusters. In this situation, such a server may be added to an existing storage group. There are two cases:

(1) the server is empty, that is, without files;
(2) the server is not empty and, therefore, it is necessary to give access to its files.

In the first case, we have to create a new storage group, the main group, containing all the servers, as we described previously. The original storage group is kept for the management of the files. The main group will be initially empty.

In the second case, we must create two new storage groups, a storage group for storing the files of the new server and the main group, initially empty. Also, the original storage group is kept as in the previous case.

3.1.6. *Creation of a storage group from two existing groups*

Let G_x and G_y be two main storage groups. We want to create a third storage group from these two groups, named *union group* G_z. The union group G_z will be formed by all the servers of the original groups G_x and G_y, keeping the existing files in the original groups. Every operation in the new group is made over all the servers, except if the operations are deployed over files existing in the original groups. This behavior is transparent for the user applications using the system, because for such applications there is only a storage group G_z, after the union, that is, the new main group.

It is important to note that the union of two storage groups is made over main groups. However, the invisible groups associated to the main groups do not disappear when the union is made. They become invisible groups of the union group. Furthermore, the main groups G_x and G_y become invisible groups of the main group G_z, that is

$$\begin{aligned}&G_x \cup G_y = G_z,\\ &G_z \text{ is main group},\\ &G_x \text{ and } G_y \text{ become invisible groups of } G_z,\\ &\text{If } G_w \text{ is invisible group of } G_x \text{ or } G_y\\ &\quad\Longrightarrow G_w \text{ is invisible group of } G_z.\end{aligned}$$

Proposition 2. *Every server belongs only to a single main group.*

Definition 4. The *main grouping relation* is the relation of ownership of a server r to a main storage group. This relation is written as R_{GP}. This relation is defined as

$$S_i R_{GP} S_j \Longleftrightarrow S_i \in G_x \wedge S_j \in G_x$$
$$\text{being } G_x \text{ a main group.}$$

Proposition 3. *The main groups of a system constitute a partition of all the servers of such system.*

The application that builds the association between servers and main storage groups is named *main grouping function.* This function is written as $\lambda_{GP}(S_i)$. In short

$$\lambda_{GP}(S_i) = G_j \Longleftrightarrow S_i \in G_j \wedge G_j \text{ is a main group.}$$

Proposition 4. *The main grouping function is an injective function.*

There exists an order relation between secondary groups and their main storage group. This order relation is a relation of "ownership" between storage groups. This relation is called *group ownership relation.* Moreover, every main group together with its secondary groups compose a *lattice*.

Definition 5. Let P_G be a relation defined over all the storage groups, named group ownership relation. The relation P_G between two storage groups G_i and G_j is defined as

$$G_i P_G G_j \Longleftrightarrow G_i = \{S_{i(1)}, \ldots, S_{i(m)}\} \subseteq G_j = \{S_{j(1)}, \ldots, S_{j(r)}\}.$$

Proposition 5. *The group ownership relation P_G is a partial order relation.*

Definition 6. We define the null storage group or simply null group ($\emptyset$) as a storage group with no servers. By definition, every group includes the null storage group.

Definition 7. The main lattice of a main group is the lattice formed by the main group, its secondary groups and the null group and whose order relation is the group ownership relation P_G. The set constituted by the main group GP_i, its secondary groups and the null group is written as $\sum GP_i$ and the main lattice $(\sum GP_i, P_G)$.

Definition 8. A main lattice is defined as a tuple $(\sum GP_i, \vee, \wedge)$, where the disjunction operation $(\vee)$ and conjunction operation $(\wedge)$ consists in

$$\begin{aligned} &\forall S_x, S_y \in \sum GP_i,\\ &S_x \vee S_y = sup\{S_x, S_y\},\\ &S_x \wedge S_y = inf\{S_x, S_y\}. \end{aligned}$$

In the case of the main lattices, the disjunction and conjunction are the same as the union of sets $(\cup)$ and intersection of sets $(\cap)$, respectively.

Observation 2. Every partition of the system is a lattice of the set of servers that belongs to such partition. The set of all the lattices is called *lattice partition*.

Example 1.

- Scenario 1: [1]

$$\begin{aligned} G_1 &= \{S_1, S_2\},\\ G_2 &= \{S_3, S_4\},\\ G_3' &= \{S_5, S_6\}\\ G_4' &= \{S_1, S_2, S_3, S_4\},\\ G_5' &= \{S_7, S_8\},\\ G_6' &= \{S_9, S_{10}\}. \end{aligned}$$

Fig. 1 shows the lattice partition in the scenario 1.

- Scenario 2: The groups G_5 and G_6 join together:

$$\begin{aligned} G_1 &= \{S_1, S_2\},\\ G_2 &= \{S_3, S_4\},\\ G_3' &= \{S_5, S_6\},\\ G_4' &= \{S_1, S_2, S_3, S_4\},\\ G_5 &= \{S_7, S_8\},\\ G_6 &= \{S_9, S_{10}\},\\ G_7' &= \{S_7, S_8, S_9, S_{10}\}. \end{aligned}$$

Fig. 2 shows the lattice partition of the scenario 2.

3.1.7. Elimination of a server from a storage group

The elimination operation requires the redistribution of the data, in order to avoid the loss of the information of the eliminated element. As we mentioned previously, these kinds of operations must not be made often because they have a high cost. For this reason, it is advisable to make them in low-load hours.

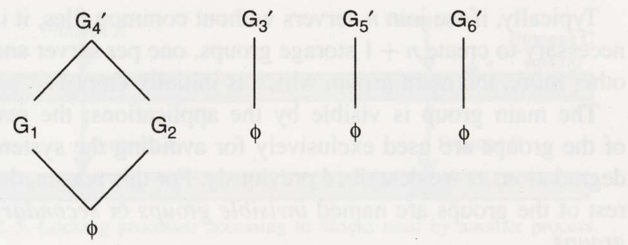


Fig. 1. Lattice partition corresponding to the scenario 1.

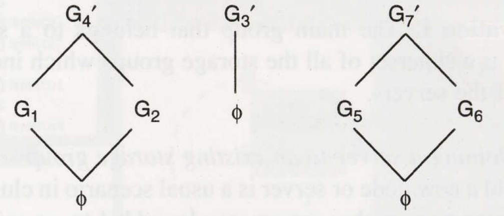


Fig. 2. Lattice partition corresponding to the scenario 2.

In the case of the elimination of a server from a storage group, files are redistributed among the rest of the components of such storage group. Furthermore, the main storage group has been modified when the server is eliminated. It is possible that the eliminated server belongs to some secondary group associated to the main group. In this case, it is necessary to redistribute the information of such secondary groups in the new main storage group, deleting them.

Example 2.

- Scenario 3: The server S_1 is eliminated. This fact implies that the secondary group G_1 disappears. After that, the server S_1 (empty) joins to the group G_7.

$$\begin{aligned} G_2 &= \{S_3, S_4\},\\ G_3' &= \{S_5, S_6\},\\ G_4' &= \{S_2, S_3, S_4\},\\ G_5 &= \{S_7, S_8\},\\ G_6 &= \{S_9, S_{10}\},\\ G_7 &= \{S_7, S_8, S_9, S_{10}\},\\ G_8' &= \{S_1, S_7, S_8, S_9, S_{10}\}. \end{aligned}$$

Fig. 3 shows the lattice partition of the scenario 3.

3.1.8. Elimination of a storage group

This operation is made over main storage groups. These secondary groups associated to the eliminated main storage group also must be eliminated. Therefore, the information of the main storage group and the secondary groups must be redistributed in a different main storage group. The selection of the storage group in which the information is going

[1] The main groups are represented by the symbol ′.

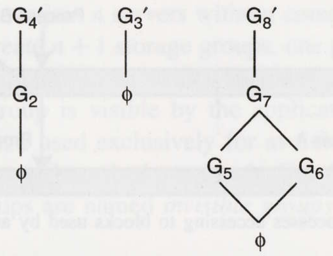


Fig. 3. Lattice partition corresponding to the scenario 3.

to redistribute must be calculated according to the system policies.

3.1.9. Storage groups policies

Storage groups can be built applying several policies, trying to optimize the access to all the storage groups. Some significant policies are:

- Grouping by server proximity: Storage groups are built based on the physical distribution of the data servers. Storage groups are composed of servers in close proximity to each other. This policy optimizes the queries addressed to a storage group because of the similar latency of messages sent to servers.
- Grouping by server similarity: Storage groups are composed of servers with similar processing capacity. This policy classifies storage groups in different categories, depending on their computational and I/O power.

3.2. Redistribution of the information in MAPFS

The creation of secondary groups involves new challenges, since the need of providing dynamic reconfiguration maintaining the performance. Some new approaches have been added to MAPFS for tackling this issue. Next sections describe in depth these proposals.

3.2.1. MAPFS defragmentation

One of the problems that must be addressed by MAPFS is the storage groups defragmentation, which is used for eliminating secondary storage groups. The defragmentation is implemented on a per-file basis, since a file is situated in a specific storage group, secondary or main. This task can be made through different alternatives.

The first choice consists of three stages, (i) read the file, which is placed at a secondary group; (ii) write the file in a temporal and sequential file; and (iii) write the temporal file in the main group. That is, it is similar to the brute force technique.

The main advantage of this alternative is its simplicity. Nevertheless, as we will see above, this solution has not a good performance.

A second proposal is reading slices from the file in the old topology and writing them in the new topology. In this case, it is necessary to change the topology between both operations. Therefore, the redistribution time follows this expression:

$$T = (T_{\text{topologyChange}} + T_{\text{transference}}) \times N,$$

where N is equal to the number of slices. We will analyze this choice above.

Finally, the third choice is based on the use of selective read operations. These operations are used for reading data of the servers that belong to the secondary group, writing them in the main group and thus, achieving the defragmentation. These operations are made without changing the topology.

In case of elimination of a server from a storage group, it will be necessary to redistribute information between the rest of servers belonging to such group. Therefore, we will have to read files from all the servers and make a selective write through the new topology, which does not include the eliminated server.

3.2.2. Big writes

Unlike the problem known as *small write problem* [7], which appears for instance in RAID systems and xFS, we have to deal with the opposite problem in MAPFS, named *big write problem.* Such problem is due to the use of secondary storage groups, which do not take advantage of all the servers of the respective main storage group.

As we have mentioned previously, the defragmentation operation is used with the aim of increasing the performance of future read and write operations. Nevertheless, due to its high cost, this operation is only made in specific situations.

The other choice is allowing applications to write files in the main group, taking advantage of a higher parallelism. In the case of new files, this is the usual scenario, since applications only see the main storage group. However, if a file has already been created and stored in a secondary group, the solution is not so obvious. There are two choices, writing in the secondary group or transferring files to the main storage group for increasing the performance of future accesses.

If a file is going to be widely changed, the best choice is writing the file in the main storage group, since there are few data items that must be redistributed. This kind of write is called *big write*. The waste of redistribution is very low (new data is not redistributed). Moreover, this solution allows us to take advantage of the improvement in later accesses (higher parallelism). Nevertheless, it is impossible to know a priori the amount of data that the user is going to write, because the write call is used with a small-size a buffer and can be invoked a huge number of times. For this reason, the user must decide whether or not a big write operation must be used, according to the overload of this operation and its advantages.

For making the use of this advanced operation easier, MAPFS offers two operations, `mapInitBigWrite` and

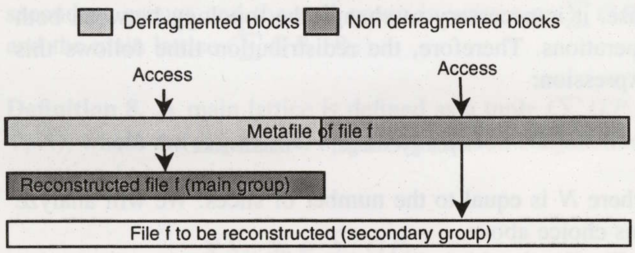

Defragmented blocks
Non defragmented blocks
Access
Access
Metafile of file f
Reconstructed file f (main group)
File f to be reconstructed (secondary group)

Fig. 4. Service maintenance during a redistribution operation.

mapFinishBigWrite, which set the boundaries in the code, in such a way that all the I/O operations between them, redistribute a file over the main storage group, if such a file is stored in a secondary storage group. Therefore, the first write operation made just after invoking mapInitBigWrite redistribute the beginning of the file until the offset of the write operation in the main storage group. The rest of write operations allow MAPFS to write in the main storage group. When the mapFinishBigWrite operation is made, the rest of data of the file are redistributed on the main storage group.

3.3. *Service maintenance*

Data redistribution is an expensive task, which affects the system performance. Since this operation is necessary, because it increases the performance of later accesses (defragmentation), or due to the capacity of the system of eliminating a server (redistribution), the system must provide service during the execution of such task.

For getting the service maintenance during these administration tasks, the original file must not be eliminated until the redistribution operation has been completed. Then, at this moment the system has two copies or views of the same file in different topologies, and each copy has a different name created by the system. The access to the different views is made through both names.

As it can be seen in Fig. 4, during the redistribution task, a file mapping must be made, in such way that if other process accesses to slices that have been written by the defragmentator, the system uses the new view. If such process accesses to slices that have not been written yet, the system uses the old view. The metafile represents the file view during the defragmentation process.

On the other hand, it is very important that the system provides service to different processes that access to the same file in a point of time. This aspect is established by means of the coutilization semantic. Since MAPFS is based on UNIX, MAPFS offers a coutilization semantic very similar to the UNIX semantic. Thus, there exists an only file view. For achieving an only image of the files, MAPFS locks processes accessing to blocks used by other process. In this sense, the defragmentator is considered as another process. This lock operation is made in a per-block basis, avoiding to reduce the parallelism. Fig. 5 shows this approach.

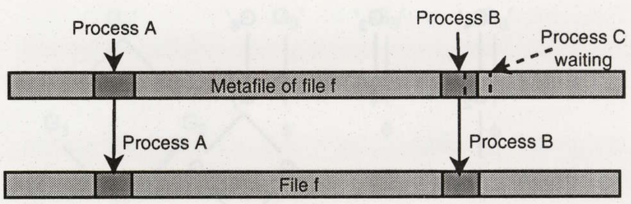
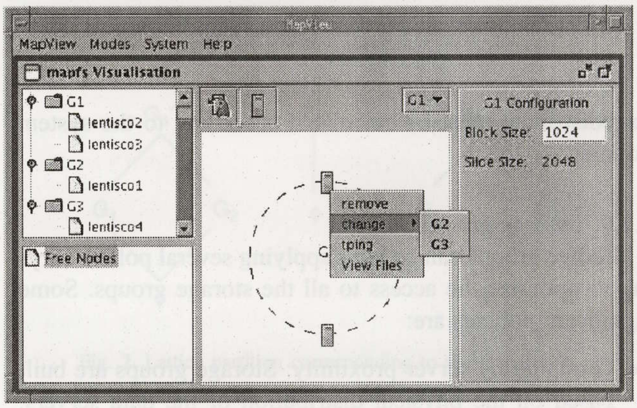


Fig. 5. Locking processes accessing to blocks used by another process.



G1 Configuration
Block Size: 1024
Slice Size: 2048

Fig. 6. MAPFS interface showing the system topology.

3.4. *Framework architecture and implementation*

Storage groups are defined within the MAPFS system. MAPFS is based on a clientserver architecture using general purpose servers, providing all the MAPFS management tasks as specialized clients. In the first prototype, we use network file system (NFS) server. NFS [11] has been ported to different operating systems and machine platforms and is widely used by many servers worldwide. Data are distributed through the servers belonging to a storage group, using a stripe unit.

On the client side, it is necessary to install a MAPFS client, which provides a parallel I/O interface to the servers, through the use of MPI [10,6] and the master–slave paradigm [2]. MPI has been chosen for the following reasons:

(1) MPI is an standard message-passing interface, which allows different processes to communicate among them by means of messages.
(2) Message-passing paradigm is useful for synchronizing processes.
(3) MPI is widely used in clusters of workstations.
(4) It provides a suitable framework for parallel applications and dynamic management of processes.
(5) Finally, MPI provides operations for modifying the communication topologies.

As repository to manage storage groups, a group database is used. Such a database stores information about the groups, their properties and their composition.

Furthermore, a graphical interface has been implemented with the aim of increasing the interaction between the user and the storage system. Fig. 6 shows the implemented interface.

4. Performance analysis

The MAPFS module in charge of managing and configuring storage groups is named MAPFS_GM (MAPFS Group Management). This section shows the analysis of the test results about the MAPFS_GM performance in a dynamic environment. By means of this analysis, we can extract some interesting conclusions that assert our previous proposals.

Systems evaluations are often made in unusual conditions, mainly due to two different reasons: (i) systems are evaluated through simulations and (ii) the test environment is different from the deployment environment. In order to evaluate our implementation, we have tested it in a real environment, supporting a normal workload.

Our work environment is a cluster, which is constituted by nodes Intel Xeon 2.40 GHz, with 1 GB of RAM memory and a 2 Gigabit network.

Our experiments have been developed in order to measure the following parameters: (i) computational load added by the storage groups management in the MAPFS system and (ii) performance of defragmentation and *big write* operations.

First, we have studied the time of the write operations to measure the computational load added by the storage groups management in the MAPFS system. We have compared the write time in the MAPFS system without group management, and the write time with group management (MAPFS_GM).

Fig. 7 shows this comparison. As it can be seen, the difference between the write time is minimal. This suggests that both write times in MAPFS were correlated to the write time in MAPFS_GM, adding a small load, less than 300 ms.

Regarding MAPFS performance, it was anticipated that the storage groups management implies load to the MAPFS system. Results showed that this load is very small. It was unlikely that in some write operations, the access time of MAPFS_GM is smaller than the access time of MAPFS without groups management. One reason could be that the small load of group management affects the total time less than the load of the nodes and the synchronization between processes.

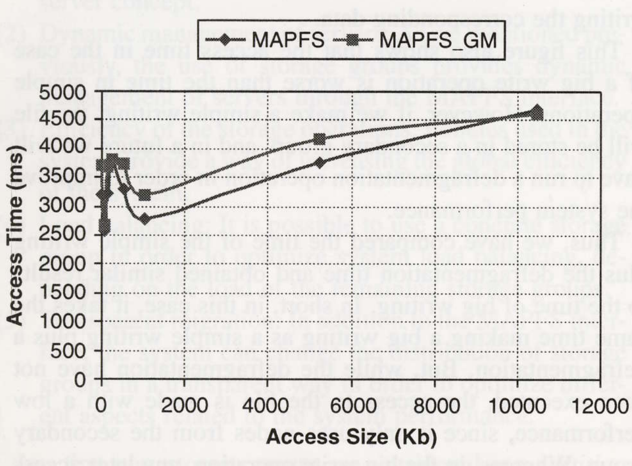


Fig. 7. Computational load added by the storage groups management.

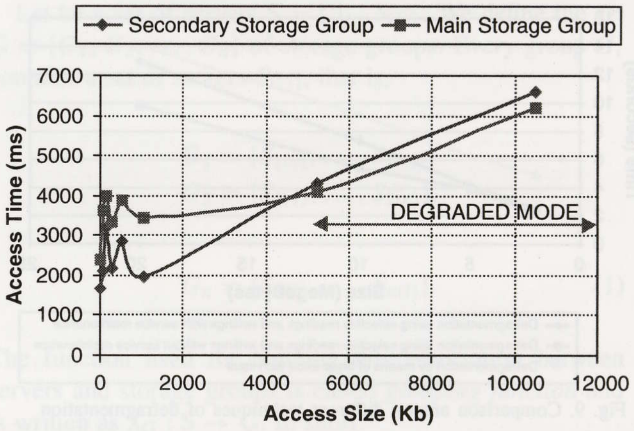


Fig. 8. Comparison between the performance of a secondary group and a main group (Degraded mode).

Furthermore, we must take into account that accesses to secondary groups are less efficient than accesses to main storage groups, because the last ones allow applications to take advantage of the highest parallelism of the system (the maximum number of storage nodes). This implies that the changes in the topology lead the system to a degraded mode by the use of old files (those files that are stored in secondary groups).

With the aim of measuring the decrease of the performance in the *degraded mode*, we have evaluated the differences between the read time of a file in a secondary group and its read time in the main group, which contains a server or storage node more. Fig. 8 shows this comparison. As it can be seen, before starting the degraded mode the read time in a secondary group is better than the time in a main group. Nevertheless, when the system is in a degraded mode the request time in a secondary group is worse.

The theory led us to infer that the degraded mode is achieved from a determined access size, because before this point, the accesses are small and the higher parallelism of the main storage group is not significant. It should be emphasized that the results shown in Fig. 8 are for a difference of an only node. Thus, in the case of the main group having more than one node of difference with respect to the secondary group, the decrease of the performance in the degraded mode would be more significant.

On the other hand, with the aim of measuring the system performance with respect to the defragmentation and big write operations, we have built a work environment formed by a secondary group with 2 nodes, which belong to a main group with 4 nodes.

First, we have focused on the performance analysis of the defragmentation operation. In Section 3.2.1 we have presented three different proposals to make the system defragmentation. In the proposal based in the changing topology

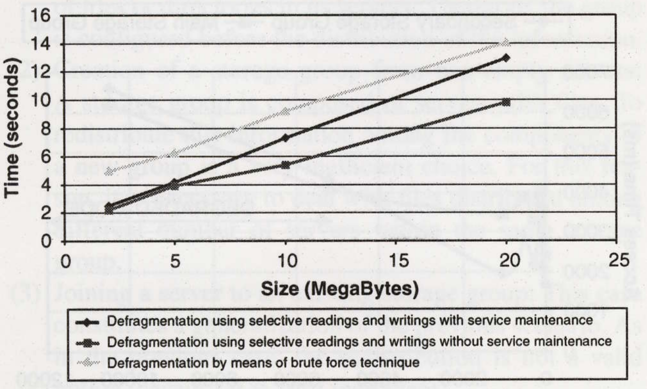


Fig. 9. Comparison among different techniques of defragmentation.

every slice of information is necessary to know the certain time that takes a changing topology. We did some tests to measure this time and we obtained that is around 1.7 s. As it is observed, this time is too high, because this operation requires killing all old MPI process and creating them on the new topology. In conclusion, if an operation defragmentation has a lot of slices and we change the topology every slice, the time of changing topology will significantly affect the total time. Thus, this solution cannot be acceptable.

The rest of proposals are compared in Fig. 9. In this figure, the time differences among the defragmentation using the brute force technique and two versions of the technique based on selective readings and writings, with and without service maintenance, can be seen. The time difference between both versions shows the maintenance cost that allows one to respond to data requests while the defragmentation is running.

Every I/O system must satisfy the user requests at any moment. Thus, every new operation included in the MAPFS file system must keep this feature and be transparent to the final user.

After this, we can compare the results obtained by the defragmentation based in the brute force technique and the defragmentation using selective readings and writings. It is necessary to emphasize that the implementation of brute force do not guarantee the service maintenance. The results show that the operation time in the second alternative is better. Thus, it can be concluded that the approach based in selective readings and writings could be the best choice to do the system defragmentation.

Then, with the aim of improving the system performance, we have defined in Section 3.2.2 a new operation to do the big write operations. Fig. 10 shows the time of the first write operation starting from a certain offset. We must take into account that this operation redistributes the information before the indicated offset on the main group that contains to the secondary group where the file is stored, and then, it makes the corresponding write operation in the new topology. The results show that the operation time is related to the size of the redistribution (offset) and there is a minimum time corresponding to the processes synchronization.

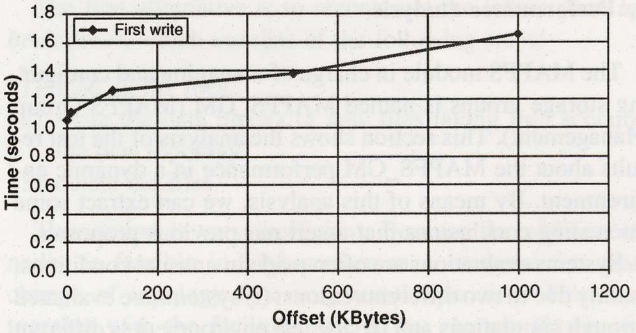


Fig. 10. Time of the first writing in a big write operation with a certain offset.

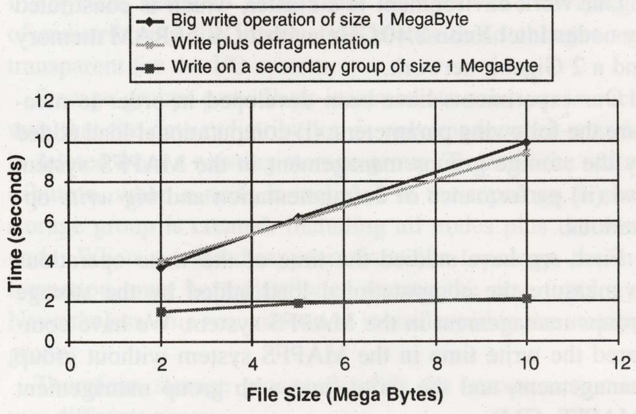


Fig. 11. Comparison between big writing operation and writing on a secondary group of size 1 MB.

These results are not decisive to take any decision about the advisability of using or not the big write operation. Fig. 11 shows a comparison between big and simple write operations on a secondary group of size 1 MB. It can be observed that the time of simple write is not correlated to the file size because only the indicated data are written. Meanwhile, the time of the big write operation is associated with the file size in which is written, because it is necessary to redistribute the rest of the file on the main group as well as writing the corresponding data.

This figure also shows that the access time in the case of a big write operation is worse than the time in simple operations. However, if we make a simple writing, the file will be stored in a secondary group, and in a future we will have to run a defragmentation operation in order to improve the system performance.

Thus, we have compared the time of the simple writing plus the defragmentation time and obtained similar results to the time of big writing. In short, in this case, it takes the same time making a big writing as a simple writing plus a defragmentation. But, while the defragmentation have not been executed, the access to the file is made with a low performance, since it only uses nodes from the secondary group. Whereas, in the big write operation, any later access to the file is made with better performance.

This discussion is supported by means of Fig. 8. This figure showed the time differences between file accesses on different types of storage groups. These results show that the access time on a main group is much better than the access time on a secondary group in the degraded mode.

Thus, all operations that redistribute files from secondary groups in the corresponding main group, may increase the performance in future accesses. This means that is convenient to use the big write operation but only when the size of the write operation is large enough comparing to the size of the file in which we are writing. This decision must be taken by an advanced user that understands the benefits of the big write operation and knows the size to write. We are currently working on providing self-management characteristics to MAPFS in order to maximize the transparency of this operation.

5. Discussion

This section shows a comparison between our proposal, storage groups, and the proposal of xFS and GFS.

Firstly, the concepts of current and obsolete groups in xFS are similar to secondary and main storage groups in MAPFS, although obsolete groups are read-only groups versus secondary groups, which are read-and-write storage groups. However, the main differences between xFS and MAPFS are:

(1) MAPFS provides several grouping policies for optimizing different parameters.
(2) MAPFS defines a formalism, based on mathematical concepts such as partitions and lattices.
(3) xFS uses a cleaner process for deleting obsolete groups. MAPFS uses defragmentation operations and big writes for this task.

With respect to GFS, the way in which subpools are formed is similar to the grouping by server similarity policy in MAPFS. However, MAPFS provides different policies for building storage groups. Storage groups in MAPFS are built for storage servers. In GFS, pools are used for grouping physical devices. Finally, GFS does not implement a dynamic reconfiguration of the storage devices.

6. Conclusions and future work

This paper has widely described storage groups and the formalism for building them. This concept allows MAPFS system to define a logical abstraction of the concept of storage server, providing a dynamic management of such servers. Moreover, storage groups allow applications uses the data servers during the reconfiguration phase.

With respect to the evaluation, the system performance has been evaluated through the analysis of several storage groups, measuring the effects of using storage groups in a dynamic environment. We conclude that the storage groups management hardly decreases the performance of the I/O architecture, providing a flexible and powerful dynamic reconfiguration.

As future work, we are developing the extension to MAPFS to an autonomic system for providing self-management of the storage groups.

References

[1] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, R. Wang, Serverless network file systems, ACM Trans. Comput. Systems 14 (1) (February 1996) 41–79.

[2] O. Beaumont, A. Legrand, Y. Robert, The master–slave paradigm with heterogeneous processors, CLUSTER 2001, IEEE Computer Society Press, Silver Spring, MD, 2001, pp. 419–426.

[3] R. Buyya (Ed.), High Performance Cluster Computing: Architectures and Systems, vol. 1, Prentice-Hall, Englewood Cliffs, NJ. 1999.

[4] P.H. Carns, W.B. Ligon III, R.B. Ross, R. Thakur, PVFS: A parallel file system for linux clusters, in: Proceedings of the Fourth Annual Linux Showcase and Conference, October 2000, pp. 317–327.

[5] L.A. Freitag, R.M. Loy, Adaptive, multiresolution visualization of large data sets using a distributed memory octree, in: Proceedings of SC99: High Performance Networking and Computing, Portland, OR, November 1999, ACM Press, New York, IEEE Computer Society Press, Silver Spring, MD.

[6] W. Gropp, E. Lusk, N. Doss, A. Skjellum, High-performance, portable implementation of the MPI message passing interface standard, Parallel Comput. 22 (6) (1996) 789–828.

[7] J.H. Hartman, J.K. Ousterhout, The Zebra striped network file system, in: H. Jin, T. Cortes, R. Buyya (Eds.), High Performance Mass Storage and Parallel I/O: Technologies and Applications, IEEE Computer Society Press, Silver Spring, MD, Wiley, New York, 2001, pp. 309–329.

[8] K. Holtman, Object level physics data replication in the grid, in: Proceedings of ACAT'2000, October 2000, pp. 244–246.

[9] P.M. Lyster, K. Ekers, J. Guo, M. Harber, D. Lamich, J.W. Larson, R. Lucchesi, R. Rood, S. Schubert, W. Sawyer, M. Sienkiewicz, A. da Silva, J. Stobie, L.L. Takacs, R. Todling, J. Zero, Parallel computing at the NASA data assimilation office (DAO), San Jose CA, IEEE Computer Society Press, Silver Spring, MD, November 1997.

[10] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, May 1994.

[11] Network Working Group, NFS: Network File System Protocol Specification, March 1989, RFC 1094.

[12] M.S. Pérez, F. García, J. Carretero, A new multiagent based architecture for high performance i/o in clusters, in: ICPP Workshops, IEEE Computer Society, Silver Spring, MD, 2001, pp. 201–206.

[13] S.R. Soltis, T.M. Ruwart, M.T. O'Keefe. The global file system, in: Proceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, September 1996.

[14] S. Widen, C. Chris, Disk defragmentation for windows NT/2000–hidden gold for the enterprise, Technical Report IDC white paper, IDC, 2000.

[15] S.J. Young, G. Y Fan, D. Hessler, S. Lamont, T.T. Elvins, M. Hadida, G. Hanyzewski, J.W. Durkin, P. Hubbard, G. Kindlmann, E. Wong, D. Greenberg, S. Karin, M.H. Ellisman, Implementing a collaboratory for microscopic digital anatomy, Supercomput. Appl. High Perform. Comput. 10 (2/3) (1996) 170–181.

María S. Pérez received the MS degree in Computer Science in 1998 at the Universidad Politécnica de Madrid, and the PhD degree in Computer Science in 2003 at the same university. From 1998 she is an associate professor at the Universidad Politécnica de Madrid. During 2003 she was post-doc at NRC in Ottawa. Her research interest includes high-performance and grid computing, parallel I/O and data mining. She is

coauthor of 4 books and she has published more than 40 articles in journals and conferences.

Alberto Sánchez received the MS degree in Computer Science in 2003 at the Universidad Politécnica de Madrid. He is currently a PhD student at the same university. He has done a predoctoral stay at CERN for 3 months. His research interest includes high-performance and grid computing, parallel I/O and autonomic computing. He is coauthor of 14 articles in journals and conferences.

José M. Peña received his PhD in Computer Science at the Universidad Politécnica de Madrid. He is an Associate Professor at the same University. He has been working during the last six years in high performance data analysis and parallel data mining. He has publications in data mining and machine learning techniques as well as distributed and parallel data processing. He is also member of the Intelligent Data Analysis Council and member of the editorial board in other specialized journals.

Víctor Robles received the MS degree in Computer Science in 1998 at the Universidad Politécnica de Madrid, and the PhD degree in Computer Science in 2003 at the same university. From 1999 he is an associate professor at the Universidad Politécnica de Madrid. During 2004 he was post-doc at Harvard Medical School. His research interest includes optimization, data mining, grid and cluster computing and bioinformatics. He is coauthor of 4 books and he has published more than 40 articles in journals and conferences.