# MAPFS: A Flexible Multiagent Parallel File System for Clusters

María S. Pérez [a], Jesús Carretero [b], Félix García [b], José M. Peña [a] and Víctor Robles [a]

[a] *Department of Computer Architecture and Technology, Technical University of Madrid, Madrid, Spain, {mperez, jmpena, vrobles}@fi.upm.es*

[b] *Departament of Computer Science. University Carlos III of Madrid. Spain, {jcarrete, fgarcia}@arcos.inf.uc3m.es*

**Abstract**

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, making the I/O phase a bottleneck in the computing systems, due to its poor performance. In this sense, parallel I/O is becoming one of the major topics in the area of high-performance systems. Existing data-intensive GCA have been used in several domains, such as high energy physics, climate modeling, biology or visualization. Since the I/O problem has not been solved in this kind of applications, new approaches are required in this case. This paper presents MAPFS, a multiagent architecture, whose goal is to allow applications to access data in a cluster of workstations in an efficient and flexible fashion, providing formalisms for modifying the topology of the storage system, specifying different data access patterns and selecting additional functionalities.

*Key words:* Parallel I/O, Cluster Computing, Multiagent Architectures, Access Patterns

## 1 Introduction

Nowadays, there is a growing interest in the development of high-performance I/O systems, because the I/O phase has become a bottleneck in the computing systems due to its poor performance. In fact, one of the major goals of high-performance computing is to provide an efficient access to data, making parallel I/O one of the most relevant issues in this field.

Currently, there are different parallel file systems, such as Galley [25], Parallel File System (PFS) [9] and Portable Parallel File System (PPFS) [16], which offer high-performance services to access resources. Many of these systems have been widely developed for parallel machines and are not suitable for clusters of workstations. Nevertheless, there is an increasing trend in parallel computing towards the usage of clusters, mainly because of their prices and their ease of integration. In this sense, the Parallel Virtual File System (PVFS) [3] is used in dedicated clusters of workstations. On the other hand, parallel file system optimizations provide improved I/O operations. The usage of hints related to different aspects of data distribution and access patterns allows parallel file systems to increase the performance of these operations. Processes such as caching or prefetching are useful approaches used in addition with these last ones [28], [2]. The *agent technology* [10], [15] is a suitable framework for integrating these functions in the storage system, because of its adaptability to different domains and its capability to achieve process autonomy.

This paper presents MAPFS, a multiagent architecture, whose goal is to allow applications to access data in a cluster of workstations in an efficient and flexible fashion, providing formalisms for modifying the topology of the storage system, specifying different data access patterns and selecting additional functionalities. 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. Section 3 presents MAPFS as a suitable infrastructure for this kind of applications in a cluster environment. This section describes MAPFS architecture, showing the different modules in which the system is divided into. Additionally, it shows the formalism of storage groups, which is used for the dynamic management of servers. Finally, in this section the way in which MAPFS allows applications to define different access patterns is also described. Section 4 shows the results obtained for the evaluation of applications using MAPFS. With this aim, we have evaluated the different features of MAPFS. Furthermore, a comparison between MAPFS and PVFS, another parallel file system for clusters, is analysed. Finally, section 5 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 I/O access requirements, also known as data-intensive applications or I/O-intensive applications, demands new I/O solutions. Examples of data-intensive and Grand Challenge applications [12] include data mining systems, data warehousing [17], high energy physics applications [34], and satellite data processing [1]. These applications may require access to data sources distributed among different nodes. Moreover, typical data-intensive applications require access to terabyte size datasets, which must be processed in an efficient way, in order to increase the performance of the applications executed on the cluster. Furthermore, data-intensive applications are very different depending on the kind of functional requirements and access patterns. It is critical for I/O system to be flexible enough to match these demands. The usage of hints, caching and prefetching policies or different data distribution configurations are optional features, which can reduce latency and increase I/O operations performance.

Respect to the underlying architecture, clusters are characterized to be modified dynamically. Operations such as addition of new nodes or elimination of existing nodes are typical in a cluster environment or in distributed systems, in general. In fact, it is desirable that the services in a cluster allow complex software systems to be built in a “plug-and-play” fashion. Therefore, we need a tool or formalism for the dynamic reconfiguration of the storage nodes. This paper describes a parallel I/O architecture for increasing the performance of data-intensive applications in a flexible way, providing formalisms for modifying the topology of the storage system and selecting additional functionalities..

## *2.2 Related Work*

Although I/O systems have traditionally tackled the I/O phase in the same way, independent of the applications domain and their access patterns, some studies [22], [26], [7] have demonstrated that a higher control of the user applications over the I/O system can increase their performance. Some of the features for increasing the control of the applications are the usage of access patterns, I/O caching and prefetching or the usage of hints.

The performance of I/O accesses depends on two related aspects: data layout in the files, which is named *storage pattern*, and the distribution of data through different nodes, that is, *access pattern*. In those cases in which these two different patterns are not equal, the efficiency of I/O accesses can be decreased, since every node must access in an independent way to data, resulting in a great number of small I/O accesses. Therefore, a suitable data placement is extremely important in the increase of the performance of I/O operations. Most file systems provide transparency to user applications, hiding details about the data distribution or data layout. Some file system such as nCUBE [6] or Vesta [5] provide programmer a higher control over the layout.

Panda [33], [4] hides physical details of the I/O systems to the applications, defining transparent schemas known as *Panda schemas*. PPFS defines a set of access patterns when a parallel file is opened [8]. Madhyastha and Reed [21] use HMM (*Hidden Markov Models*), and neural networks for classifying access pattern within a file. MPI-IO uses *MPI datatypes* to describe data layout both in memory and in file, in such a way that it is possible to specify non-contiguous access [36], [37].

On the other hand, if the behaviour of an algorithm is known, I/O requirements could be achieved in advance before the algorithm actually needs the data. An optimal usage of computational resources, like I/O operations or disk caches is also a key factor for high-performance algorithms. With the usage of data access knowledge both accurate prefetching and resource management could be achieved. For using these features, it is necessary to have a cache structure, where the I/O data items are stored in memory. The effectiveness of the cache is determined largely by the replacement policy. A typical cache replacement policy is *Least Recently Used (LRU)*. This policy assumes that data items used recently are likely to be used again and therefore, on every release moves the item to the tail of the free list. The item at the front of the list is the oldest inactive item and the first candidate to leave such list. Other alternatives widely used are the FIFO policy, which replaces the oldest item and MRU policy, which replaces the most recently used item. Historical information is often used for both file caching and prefetching. In fact, the LRU policy is this kind of system, where the history consists of the recent accesses to the data. In the case of prefetching, the most used approach is sequential readahead. All these policies are used in conventional systems. Nevertheless, these policies are not suitable for all the different access patterns of applications. A more general scheme resorts to using hints about applications with the aim of increasing of the caching and prefetching phases, since hints provide information used to decrease the cache faults and prefetch the most probable used data in the next executions. In other words, the more information has been obtained, the less uncertainty in predicting future accesses and, therefore, better prefetching and caching results.

Finally, hints are widely used for increasing the system performance in general in computing systems. In 1983, Lampson described its usage in operating systems such as Alto or Pilot, networks such as Arpanet or Ethernet and in languages such as Smalltalk [20]. In the context of the parallel file systems, hints are structures known and built by the I/O system, which are used for improving the read and write routines performance. Hints are usually used as historical information for optimizing caching and prefetching techniques. For example, hints can be used in the prefetching phase for deciding how many blocks are read in advance. OSF/1 system prefetchs up to 64 data blocks when large sequential requests are detected [27]. Other works detect more complex access patterns for non-sequential prefetching [19]. In fact, hints allow I/O systems to take a more active role, since with the usage of this information, they can be reconfigurated in order to increase the performance of applications. There exists a great number of works which infer future accesses based on past accesses [18], [35], [13]. In [19], historical information is used in order to predict the I/O workload and, thus, enhance the prefetching stage. In [28] two different kinds of hints are defined, *disclosing hints* and *advising hints*. The first kind of hints describe the necessary knowledge about the behavior of the application. Advising hints give recommendations about resources management. Hints can also be specified by means of the MPI-IO interface [37]. `MPI_Info` structures are used for specifying pairs `(key,value)`, which provide additional information about I/O operations. There is a set of reserved keys, related to the access patterns or data layout over I/O devices.

## 3 Proposed Approach

The previous section states the need of defining a flexible framework for tackling the I/O problems of data-intensive applications.

MAPFS [30] is a multiagent architecture, which provides flexibility in different aspects:

- System topology configuration: Ability to change system topology, setting the I/O nodes and their relationships. This feature is achieved through the usage of *storage groups*, which are described in section 3.2.
- Access pattern specification: Although MAPFS is a general purpose I/O system, it can be configured in order to adapt to different I/O access patterns [32].
- There are different reasons to allow some functionalities (such as caching or prefetching) to run in parallel on different nodes and even on the data servers. Moving executions to data servers may reduce network latency and traffic. Because of its properties, such as autonomy, proactivity and reactivity, the agent technology is useful in this case. MAPFS is composed of a multiagent subsystem, which is responsible for performing several independent tasks.

These features allow MAPFS to enhance the three main levels of an integrated I/O system:

(1) Server-side, through the definition of the storage groups formalism.
(2) Client-side, through the implementation of different functionalities, made by a multiagent subsystem.
(3) Applications, which may configure MAPFS through the usage of control user structures in order to provide hints that increase the I/O operations performance.

## *3.1 MAPFS Architecture*

MAPFS is based on a client-server architecture that uses general-purpose servers. In the first prototype, NFS servers are used [11]. NFS [23], [24] has been ported to different operating systems and machine platforms and it is widely used by many servers worldwide. Data is 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.

Additional multiagent subsystems, providing several functionalities, are executed on different nodes. Every storage group is associated to a multiagent subsystem. These multiagent subsystems use an agent hierarchy, which solves the information retrieval problem in a transparent and efficient way. The taxonomy of agents used in MAPFS is composed of:

- Extractor agents: They are responsible for information retrieval, invoking parallel I/O operations.
- Distributor agents: They distribute the workload to extractor agents. These agents are placed at the highest level of the agents hierarchy.
- Caching and prefetching agents: They are associated with one or more extractor agents, caching or prefetching their data.
- Hints agents: They must study applications access patterns to build hints improving data access.
- Fault tolerance agents: They must provide data availability. In order to achieve this goal, these agents must use duplicate data.

Figure 1 represents the relation among these agents. The taxonomy of agents can be extended to provide additional functionalities. The most usual configuration is to run these subsystems on data servers, helping to reduce network traffic. In this case, a major requirement is to install a technology that supports distributed execution of agents. MAPFS provides this optional functionality.

It is possible to distinguish two different aspects of the MAPFS architecture:

(1) Aspects related to the intrinsic features of a parallel file system.
(2) Aspects related to data distribution and processing of these data. The usage of agents is suitable in this field.

These two aspects are mapped into the MAPFS architecture through two subsystems: file subsystem, called MAPFS_FS and a multiagent subsystem, called MAPFS_MAS [31]. The first one is responsible for the modeling of the file and the final I/O operations. The second one contains the hierarchy of agents, represented in Figure 1. There exists a multiagent for each storage

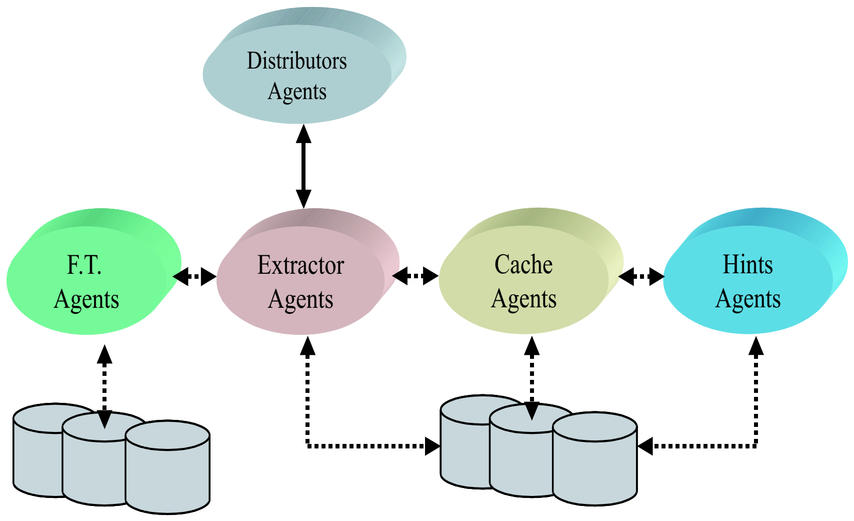


Fig. 1. Taxonomy of agents used in MAPFS

group (see section 3.2). Both subsystems are used together in order to acquire data in an efficient and transparent manner. Figure 2 represents these two subsystems.

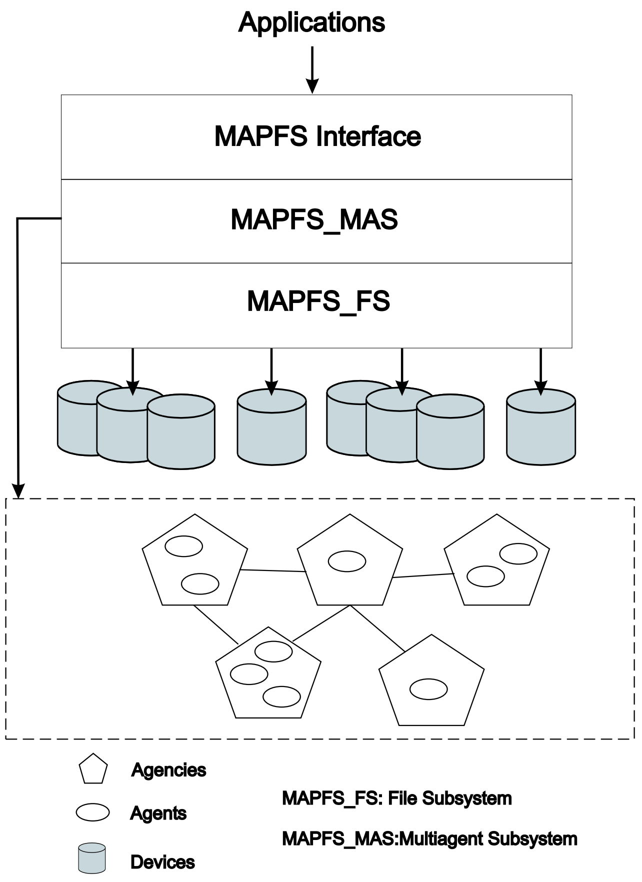


Fig. 2. MAPFS Subsystems

From an architectonic point of view, MAPFS is divided in several modules, which have different tasks. MAPFS has to provide both an user interface and connection capacity through a network in a distributed system. These tasks are responsible for two different modules in the architecture, the **user interface** and the **communication manager**, respectively. Besides, the file system must use a redundancy scheme that provides fault tolerance to the whole system. This feature, which is established at file level, is implemented by the **fault tolerance manager**. This manager communicates with an **agency**[1]. The agency provides autonomy to the system in order to achieve the fault tolerance feature. Finally, MAPFS manages the different caches of the file system. The module responsible is the **cache manager**. This manager communicates with both a cache structure and the multiagent system. In order to assemble all the file system functionality and build the file model, the **file manager** is used, which constitutes the central core of the file system.

In Figure 3 the MAPFS architecture is depicted, where the component modules are shown.

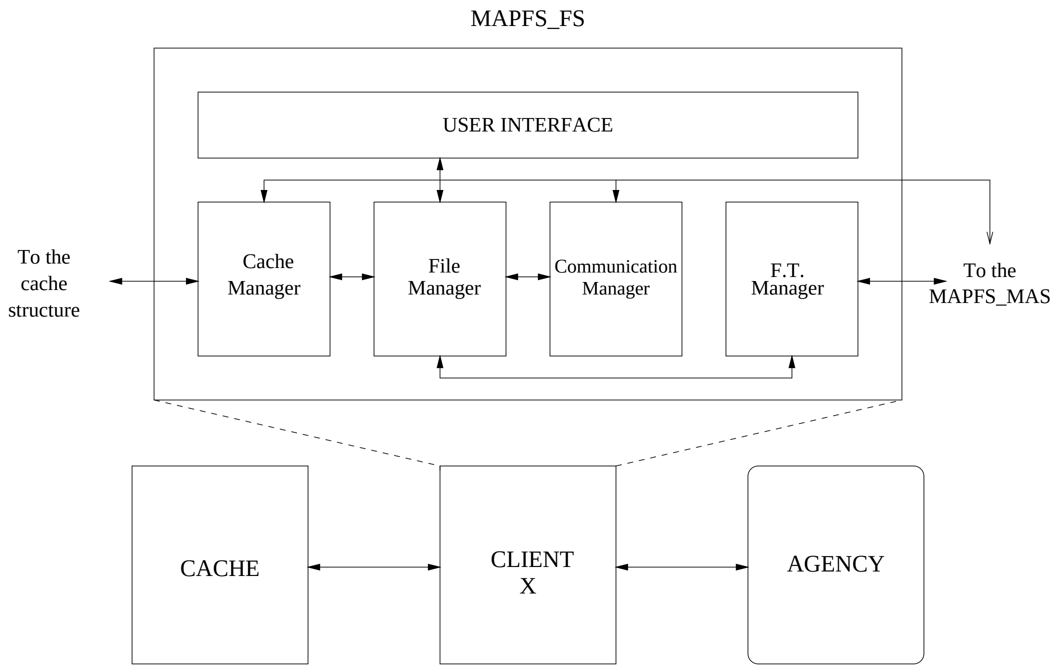


Fig. 3. MAPFS Structure

The user interface is a module that provides access to the MAPFS functionality. The main objective of the user interface is to isolate the applications that use MAPFS from the MAPFS implementation. The communication manager is used for managing the communication between the MAPFS system and the file storage server. The communication manager has three goals:

(1) Transfer information and meta-information in the MAPFS system.

[1] An agency is a software container where several agents develop their work

(2) Manage the communication between the cache manager and the file storage server in order to update the cache.
(3) Distribute agents in the MAPFS system.

For this reason, the communication manager is linked to the file manager, the cache manager and the multiagent system.

The fault tolerance manager provides fault tolerance to the file system. The goal of this module is to guarantee the data availability. For managing this feature, *fault tolerance agents* from the multiagent system are used. Because of that, the fault tolerance manager has a connection link with the multiagent system. The fault tolerance manager must also communicate with the file manager, because the fault tolerance characteristic is implemented on a per-file basis.

Furthermore, MAPFS takes advantage of the temporal and spatial locality of data stored in the servers. Using a cache in the system improves its performance, but it causes an important coherence problem. Again, there are a set of agents that manage this feature. These agents are named *cache agents*. They are responsible for using a cache coherence protocol and control data transfer between the storage devices. The cache manager is linked to the file manager, the communication manager and the multiagent system.

Finally, the file manager is responsible for the modeling of the file and, therefore, contains the implementation of the MAPFS interface. This module communicates with all other modules in order to provide access to the main functionalities.

## *3.2 Storage Groups*

The concept of grouping is fundamental in every aspect of life. Edwin P. Hubble, who is considered the founder of observational cosmology, said in the thirties that the best place for searching for a galaxy is the next to another one, describing the concept of galaxy grouping. As in real life, computer science has a significant number of groupings, e.g. process group or user group.

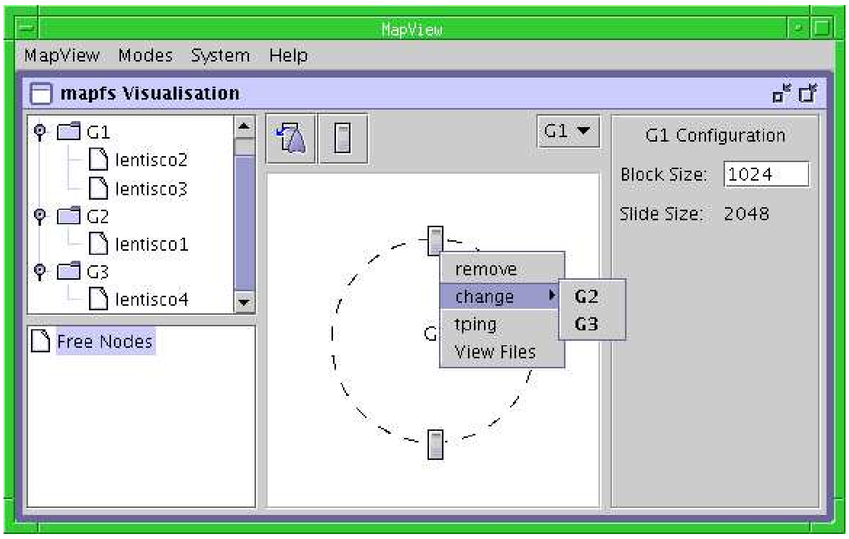
A *storage group* is defined in MAPFS as a set of servers clustered as groups. These groups take the role of data repositories and can be built by applying several policies to optimize the accesses to all the storage groups. Some significant policies are:

- 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.

The main advantages of storage groups are:

(1) Logical abstraction of the concept of storage server: As a partition is a logical 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 usage of storage groups provides dynamic management of servers through the MAPFS interface (see Figure 4).
(3) Efficiency of the storage operations: The 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, according to 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.



G1 Configuration
Block Size: 1024
Slide Size: 2048

Fig. 4. MAPFS interface showing the system topology

The system topology can be changed dynamically. In this case, data must be reconstructed, because the system must map files stored in the servers of the previous topology to the new configuration. Data reconstruction degrades the performance of the I/O system.

In order to avoid data reconstruction, MAPFS defines two types of storage groups, main storage groups and secondary groups, which form a lattice structure between them. Secondary groups are used for storing new data, avoiding the migration of the files data. In fact, this approach postpones data reconstruction until the system runs a de-fragmentation operation, which is used for deleting secondary groups and simplifying the storage system description.

### *3.3 Access Pattern Specification*

There are a huge number of applications using parallel file systems. These applications have very different requirements and data access patterns. Therefore, it is desirable that the underlying file system allows these applications to provide information about the layout of the data used by them. This information is given as hints, which are used for improving read and write operations performance. For example, storage systems using hints may provide greater performance because they use this information to decrease cache faults and to prefetch the data most likely to be used in later executions.

MAPFS uses hints to access data. They can be obtained in two ways:

(1) Given by the user, that is, the user application provides to I/O system the necessary specifications.
(2) Built by the multiagent subsystem. If this option is selected, the multiagent system must analyze the access pattern of the applications in order to build hints, improving data access.

If hints are provided by the user application, it is necessary for the system to provide syntactic rules for setting the system parameters, which configure the I/O system. On the other hand, if the multiagent subsystem creates the hints, it is also necessary to store them in a predefined way. In any case, lexical and syntactic rules must be introduced in the system.

The system is configured through several operations, which are independent of the I/O operations, although these last ones use the former operations. The configuration operations are divided into:

- Hints Setting Operations: Operations for establishing the hints of the system. Therefore, they can set and get the values of the different fields of the hints.
- Control User Operations: Higher level operations that can be used directly by the user applications to manage system performance.

Figure 5 shows the three subsets of the Storage System API. As can be seen in the Figure, there are three ways of accessing the Hints Setting Module:

(1) I/O operations may ask for hints values and even modify them.
(2) Control user operations may modify the hints. This is the normal way

for the user applications to interact with hints.

(3) To make the system flexible, the Hints Setting Module may be accessed directly through the Hints Setting API. The multiagent system may build and modify hints through this interface.

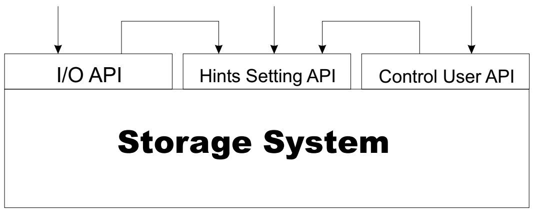


Fig. 5. Storage System API

The MAPFS I/O API has been described in [30]. Hint setting operations are the following ones:

- *Mapfs_Hints * mapHintsNew(int block_ident)*: This operation creates a new hint structure for the block with identifier *block_ident*.
- *void mapHintsFree(Mapfs_Hints *hint)*: This operation releases a hint structure.
- *int mapHintsSet(Mapfs_Hints *hint, int code_field, void * value)*: This operation modifies a field of the hint structure with a value. The operation returns 0 if successful and -1 otherwise.
- *void * mapHintsGet(Mapfs_Hints *hint, int code_field)*: This operation returns the value of a concrete field of the hint structure. If the field is not defined, *mapHintsGet()* returns NULL.

Hints in MAPFS are built based on the concept of attribute. An attribute can take different values, which provides useful information for the I/O operations. The control user operations have a similar structure:

- *Mapfs_CtrlUser * mapCtrlUserNew(Mapfs_Tuples *tupl)*: This operation creates a new Control User structure for a set of tuples represented by *tupl*.
- *void mapCtrlUserFree(Mapfs_CtrlUser *ctrlUser)*: This operation releases a Control User structure.
- *int mapCtrlUserSet(Mapfs_CtrlUser *ctrlUser, int code_field, void *expr)*: This operation modifies a field of the Control User structure with an expression. The operation returns 0 if successful and -1 otherwise.
- *void *mapCtrlUserGet(Mapfs_CtrlUser *ctrlUser, int code_field)*: This operation returns the value of a concrete field of the Control User structure. If the field is not defined, *mapCtrlUserGet()* returns NULL.

Control User operations are higher level operation using expressions instead of simple attributes, which are translated to hints by the MAPFS system. These expressions are formed by boolean operations applied to attributes. An example is shown in Section 4.1.

## 4 Performance Analysis

### *4.1 MAPFS Evaluation*

In order to validate our implementation, it is necessary to evaluate its performance. Experiments were run on a cluster of four Athlon 650MHz nodes, each with 256 MB of RAM memory, attached to a Gigabit Ethernet network. This cluster constitutes our trial storage group.

Our experiment consists of four processes running a multiplication of two matrices, where the matrices are stored in the cluster, using MAPFS as underlying platform. The resultant matrix is also stored in a distributed fashion.

Assume a matrix multiplication of two matrices:

$$A[M, N] * B[N, P] = C[M, P]$$

The rows of the matrices A and B are stored in row order in two different files.

With a traditional product algorithm:

```
for (i=0; i<M; i++)
        for (j=0; j<P; j++)
                for (k=0; k<N; k++)
                        C[i][j]+=A[i][k]*B[k][j];
```

We assume that a row may be larger than the cache (we are considering large size matrices). It is not suitable to prefetch the complete row associated to a concrete element because:

- Every time a row is finished (that is, the index "`i`" is changed, after using the row P times), it is appropriate to prefetch elements of row "`i+1`".
- If the row is larger than the cache, it is better to prefetch a window of `K` values, which corresponds to the `K` following elements with respect to the considered element, but in a circular way (`mod N`). `K` may depend on different factors, such as cache size, system load or even it may be dynamic.

In this case, it is possible to use "row prefetching", with circular window `K` (`mod N`), a cycle of P repetitions per row, sequential advance of the rows and without cycle of repetition per matrix. That is:

```
$read(A[i,j]) \Longrightarrow$
  If it is not the last cycle (number of accesses $A[i,j] < P$) :
    • *Prefetching* from $A[i, j+1]$ to $A[i, (j+K)\%N]$
  If it is the last cycle (==P) $\Longrightarrow$ (advance to the following row):
    • *Prefetching* of the K following elements, both from the current row,
      if there are enough elements, and the first ones from the row "$i+1$",
      if it is necessary and $i < M$
```

Therefore, the control user structures provided by the applications are the configuration parameters of the prefetching process. These are:

- Type: It may be per row or per column.
- Window type: It may be circular or not.
- Window size: Number of elements to read in every cycle of prefetching.
- Cycle per row or per column: It specifies if there exists cycle per row or per column or not. This last case is indicated with a value 0.
- Cycle per matrix: It specifies if there exists cycle per matrix or not. This last case is indicated with a value 0.

The control user structure provided by the application follows this expression:

```
Type=Row AND Window_Type=Circular AND Window_Size=K
   AND Cycle_Per_Row=P AND Cycle_Per_Matrix=0
```

Therefore, the sequence of configuration operations made by the application is:

```
(1) Mapfs_CtrlUser *mapCtrlUser = mapCtrlUserNew(data);
(2) mapCtrlUserSet(mapCtrlUser, position, "Type=row
    AND Window_Type=Circular AND Window_Size=K
    AND Cycle_Per_Row=P AND Cycle_Per_Matrix=0");
```

This control user structure is translated to hints by the multiagent subsystem. Hints are used as data tags for processing the elements in an efficient manner, prefetching data provided by the corresponding rule. This experiment was compared to another one, which consists of multiplying the same matrices stored on the local disk through the usage of a traditional I/O system. The size of the matrices was 100 MB. Figure 6 shows the execution times of both applications.

Figure 7 shows the speedup of the MAPFS solution versus local solution, varying the access size used in the I/O operations. As can be seen, the speedup

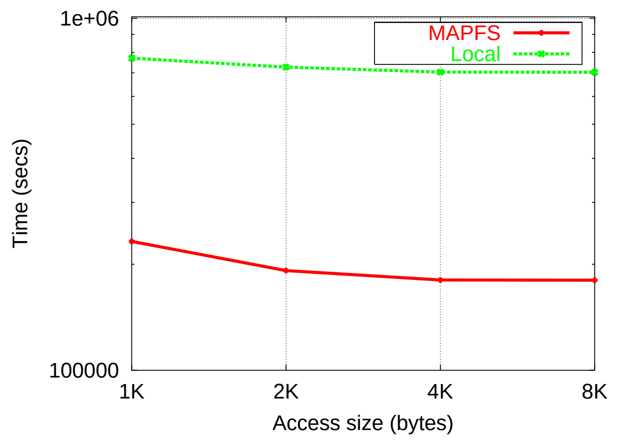


Fig. 6. Comparison of MAPFS and local application

is very close to 4, the number of nodes, which is the maximum speedup, limited by the "Amdahl's law".

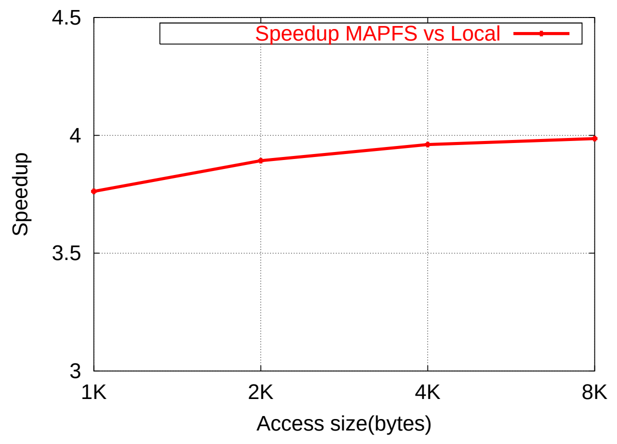


Fig. 7. Speedup of the MAPFS solution on a cluster of 4 nodes versus Local solution

### *4.2 Comparison with PVFS*

The best way of evaluating MAPFS is to compare the performance of an application using this parallel file system and another one. PVFS [3], [14] has been chosen due to the following reasons:

(1) It is a parallel file system developed for Linux clusters.
(2) It provides high bandwidth for concurrent read/write operations from multiple processes or threads to a parallel file.
(3) It is robust and scalable.
(4) It gives support for multiple APIs, including a native PVFS API, the UNIX/POSIX API and MPI-IO API. We have used the first one for implementing the application.

Again, we have compared the multiplication of two matrices distributed among different nodes both in MAPFS and PVFS. Figure 8 shows the execution time of the application using both parallel file systems. As can be seen in the Figure, the difference is significant, although it decreases as the access size is increased. It is important to emphasize the fact that PVFS is a high-performance parallel file system.

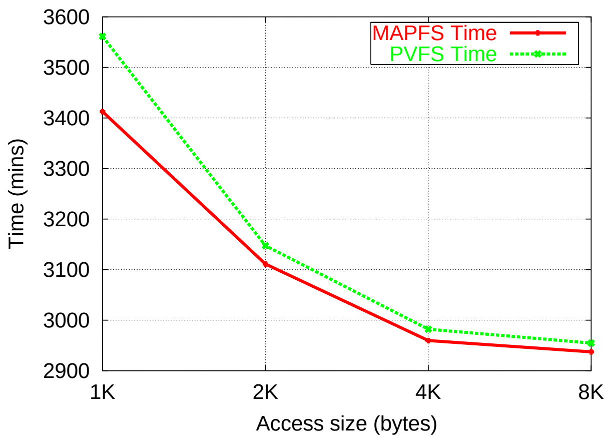


Fig. 8. Execution time of multiplication of two matrices with MAPFS and PVFS

## 5 Conclusions and Future Work

In this work we have presented MAPFS, a new multiagent architecture for high performance I/O in clusters. MAPFS provides the following properties: (i) system topology configuration; (ii) access patterns specifications by applications, and (iii) usage of a multiagent subsystem to support specific functionalities, such as caching and prefetching.

MAPFS has been compared to both a local solution and PVFS, a high performance parallel file system developed for Linux clusters, concluding that MAPFS provides a speedup very close to the maximum, improving upon the results of PVFS.

For future work, it would be interesting to evaluate the performance of the system with the usage of several storage groups and other different applications. Furthermore, the growing proliferation of data grids in heterogeneous and geographically distributed environments provides a new framework in which MAPFS can offer the parallel I/O features described in this paper. In this sense, [29] constitutes a proposal for the usage of MAPFS in data grids. Nevertheless, there are a great number of open research issues and work for finalizing this solution. One such issue is the adaptation of the concept of storage group to a grid environment.

## References

[1] Anurag Acharya, Mustafa Uysal, Robert Bennett, Assaf Mendelson, Michael D. Beynon, Jeff Hollingsworth, Joel Saltz, and Alan Sussman. Tuning the performance of I/O-intensive parallel applications. In *Proceedings of the Fourth ACM Workshop on I/O in Parallel and Distributed Systems*, May 1996.

[2] Pei Cao, Edward W. Felten, and Kai Li. Implementation and performance of application-controlled file caching. In *Operating Systems Design and Implementation*, pages 165–177, 1994.

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

[4] Yong E. Cho. *Efficient resource utilization for parallel I/O in cluster environments.* PhD thesis, University of Illinois at Urbana-Champaign, 1999.

[5] P. Corbett, D. Feitelson, J. Prost, and Johnson S. Parallel access to files in the Vesta file system. In *Proc. of the 15th. Int. Symp. on Operating Systems Principles*, pages 472–481. ACM, 1993.

[6] E. DeBenedictis and J. M. del Rosario. nCUBE parallel I/O software. In *Eleventh Annual IEEE International Phoenix Conference on Computers and Communications (IPCCC)*, pages 117–124, Apr 1992.

[7] J.M. del Rosario and A.N. Choudhary. High-performance I/O for massively parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, 1994.

[8] C. Elford, C. Kuszmaul, J. Huber, and T. Madhyastha. Scenarios for the Portable Parallel File System, 1993.

[9] R. Esser and R. Knecht. Intel Paragon XP/S— architecture and software environment. In *Proceedings of Supercomputer '93, Lecture Notes in Computer Science, Mannheim, Germany*, 1993.

[10] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Springer-Verlag*, 1996.

[11] Félix García, Alejandro Calderón, María S. Pérez, and Luis M. Sánchez. Evaluating Expand: A parallel file system using NFS servers. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, July 2002.

[12] Grand Challenging Applications. *http://www.mcs.anl.gov/projects/grand-challenges.*

[13] Jim Griffioen and Randy Appleton. Reducing file system latency using a predictive approach. In *USENIX Summer*, pages 197–207, 1994.

[14] W. B. Ligon III and R. B. Ross. An overview of the parallel virtual file system. In *Proceedings of the 1999 Extreme Linux Workshop*, June 1999.

[15] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems Journal*, 1(1):7–38, 1998.

[16] James V. Huber Jr., Christopher L. Elford, Daniel A. Reed, Andrew A. Chien, and David S. Blumenthal. PPFS: A high performance portable parallel file system. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 385–394, Jul 1995.

[17] R. Kimball. *The Data Warehouse Toolkit.* John Wiley and Sons, Inc., 1996.

[18] K. Korner. Intelligent caching for remote file service. In *Proceedings of the 10th Intl. Conf. on Distributed Computing Systems*, pages 220–226, May 1990.

[19] David Kotz and Carla Schlatter Ellis. Practical prefetching techniques for parallel file systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 182–189. IEEE Computer Society Press, 1991.

[20] B. W. Lampson. Hints for computer system design. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles (SOSP)*, volume 17, pages 33–48, 1983.

[21] Tara M. Madhyastha and Daniel A. Reed. Input/Output access pattern classification using hidden Markov models. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 57–67, San Jose, CA, 1997. ACM Press.

[22] Ethan L. Miller and Randy H. Katz. Input/output behavior of supercomputer applications. In *Proceedings of Supercomputing '91*, pages 567–576, November 1991.

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

[24] Network Working Group. *NFS Version 3 Protocol Specification*, June 1995. RFC 1813.

[25] Nils Nieuwejaar and David Kotz. The Galley parallel file system. In *Proceedings of the 10th ACM International Conference on Supercomputing*, pages 374–381. ACM Press, May 1996.

[26] Barbara K. Pasquale and George C. Polyzos. A static analysis of I/O characteristics of scientific applications in a production workload. In *Proceedings of Supercomputing '93*, pages 388–397, 1993.

[27] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. Informed prefetching and caching. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pages 224–244. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[28] R. Hugo Patterson, Garth A. Gibson, and M. Satyanarayanan. A status report on research in transparent informed prefetching. *ACM Operating Systems Review*, 27(2):21–34, 1993.

[29] María S. Pérez, Jesús Carretero, Félix García, José M. Peña, and Víctor Robles. MAPFS-Grid: A flexible architecture for data-intensive grid applications. *Grid Computing (Lecture Notes in Computer Science)*, 2004.

[30] María S. Pérez, Félix García, and Jesús Carretero. A new multiagent based architecture for high performance I/O in clusters. *2001 International Conference on Parallel Processing Workshops*, September 2001.

[31] María S. Pérez, Félix García, and Jesús Carretero. MAPFS_MAS: A model of interaction among information retrieval agents. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, May 2002.

[32] María S. Pérez, Ramón A. Pons, Félix García, Jesús Carretero, and Víctor Robles. A proposal for I/O access profiles in parallel data mining algorithms. In *3rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, June 2002.

[33] K. Seamons. *Panda: Fast Access to Persistent Arrays Using High Level Interfaces and Server Directed Input/Output.* PhD thesis, University of Illinois at Urbana-Champaign, 1996.

[34] A. Shoshani, L. Bernardo, H. Nordberg, D. Rotem, and A. Sim. Storage management for high energy physics applications. *Computing in High Energy Physics*, 1998.

[35] C. D. Tait and D. Duchamp. Detection and exploitation of file working sets. In *Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS)*, pages 2–9, Washington, DC, 1991. IEEE Computer Society.

[36] Rajeev Thakur, William Gropp, and Ewing Lusk. A case for using MPI's derived datatypes to improve I/O performance. In *Submitted as an extended abstract to SC98: High Performance Networking and Computing.* Argonne National Laboratory, May 1998.

[37] Rajeev Thakur, William Gropp, and Ewing Lusk. Optimizing noncontiguous accesses in MPI-IO. *Parallel Computing*, 28(1):83–105, 2002.