An agent architecture for managing data resources in a grid environment

María S. Pérez [a], Alberto Sánchez [a], Jemal H. Abawajy [b], Víctor Robles [a], José M. Peña [a]

[a] *DATSI. FI. Universidad Politécnica de Madrid. Spain,* {*mperez,ascampos,vrobles,jmpena*}*@fi.upm.es*

[b] *Deakin University, Victoria, Australia, jemal@deakin.edu.au*

Abstract

Agent paradigm has been sucessfully used in a large number of diverse fields and initiatives. One of this initiative is the definition of MAPFS, a parallel file system based on a multiagent architecture. The use of a multiagent system implies *coordination* and *cooperation* among its agents. MAPFS is intended for clusters of workstations, where the agent technology is applied in order to provide a communication model between agents with different roles. The adaptation of MAPFS to a grid environment is named MAPFS-Grid.

This paper describes the conceptual agent framework and the communication model used in the design and development of MAPFS-Grid, which provides the management of data resources in a grid environment.

Key words: Multiagent system, cooperation, parallel file system, data grid.

1 Introduction

A large number of application of the agent technology have been made in diverse areas, such as business [19], electric management [18,7], control [2,3], networks [11] or in general, industrial applications [16,17].

Agent technology provides several concepts, which allow analysts to design applications in a way close to the human thought. Furthermore, agents provide applications with useful features for dealing with complex and dynamic environments.

Nevertheless, there exist important differences between *system programming* and agent technology. The main differences are:

- Agent paradigm interact to the system at a higher level than system programming.
- The efficiency is a very strict requirement in the case of the system programming. Agent technology introduces an abstraction layer and, thus, it involves a lost of efficiency.

Nevertheless, these disadvantages can be avoided, since the agent paradigm differ clearly agent theory, which provides the concepts, and agents architectures, which provides concrete solutions and implementations.

MAPFS is a successful application of the agent theory in the development of a parallel file system [25]. The same philosophy is applied to the design of MAPFS-Grid [26], the adaptation of this parallel file system to grids.

This paper describes the use of the agent theory as conceptual framework in the design and development of MAPFS-Grid as well as the MAPFS-Grid cooperation model.

The outline of this paper is as follows. Section 2 introduces the MAPFS-Grid system and describes the related work. Section 3 describes the generic structure of an agent in MAPFS-Grid. Section 4 analyzes the MAPFS cooperation model and describes the communication features of MAPFS-Grid. Section 5 shows the implementation and evaluation of MAPFS, in order to measure the influence of the agents in the management of data resources. Finally, Section 6 summarizes our conclusions and suggests further future work.

2 Problem Statement and Related Work

2.1 MAPFS-Grid overview

MAPFS-Grid [26] provides a grid-like interface to a parallel file system based on clusters, that is, MAPFS [25]. MAPFS (*Multi Agent Parallel File System*) has been developed at the Universidad Politécnica de Madrid in 2003. The main contribution of MAPFS is the conceptual use of agents to provide applications with new properties, with the aim of increasing their adaptation to dynamic and complex environments.

MAPFS is intended to use in a cluster of workstations, transfering in parallel among all the cluster nodes. On the other hand, MAPFS-Grid allows heterogeneous servers connected by means of a wide-area network to be used as data repositories, by storing data in a parallel way through all the clusters and individual nodes which make up the grid.

Figure 1 shows the overview of the MAPFS-Grid system. This system provides two levels of parallelism:

(1) The high level provides parallelism among the set of clusters and nodes of a grid, that is, *inter-cluster parallelism*.
(2) The low level provides parallelism among the set of nodes of each cluster, that is, *intra-cluster parallelism*. This is made through the *Parallel Data Access Service (PDAS)*, which allows parallel I/O operation to be made in a cluster environment, providing access to the MAPFS file system.

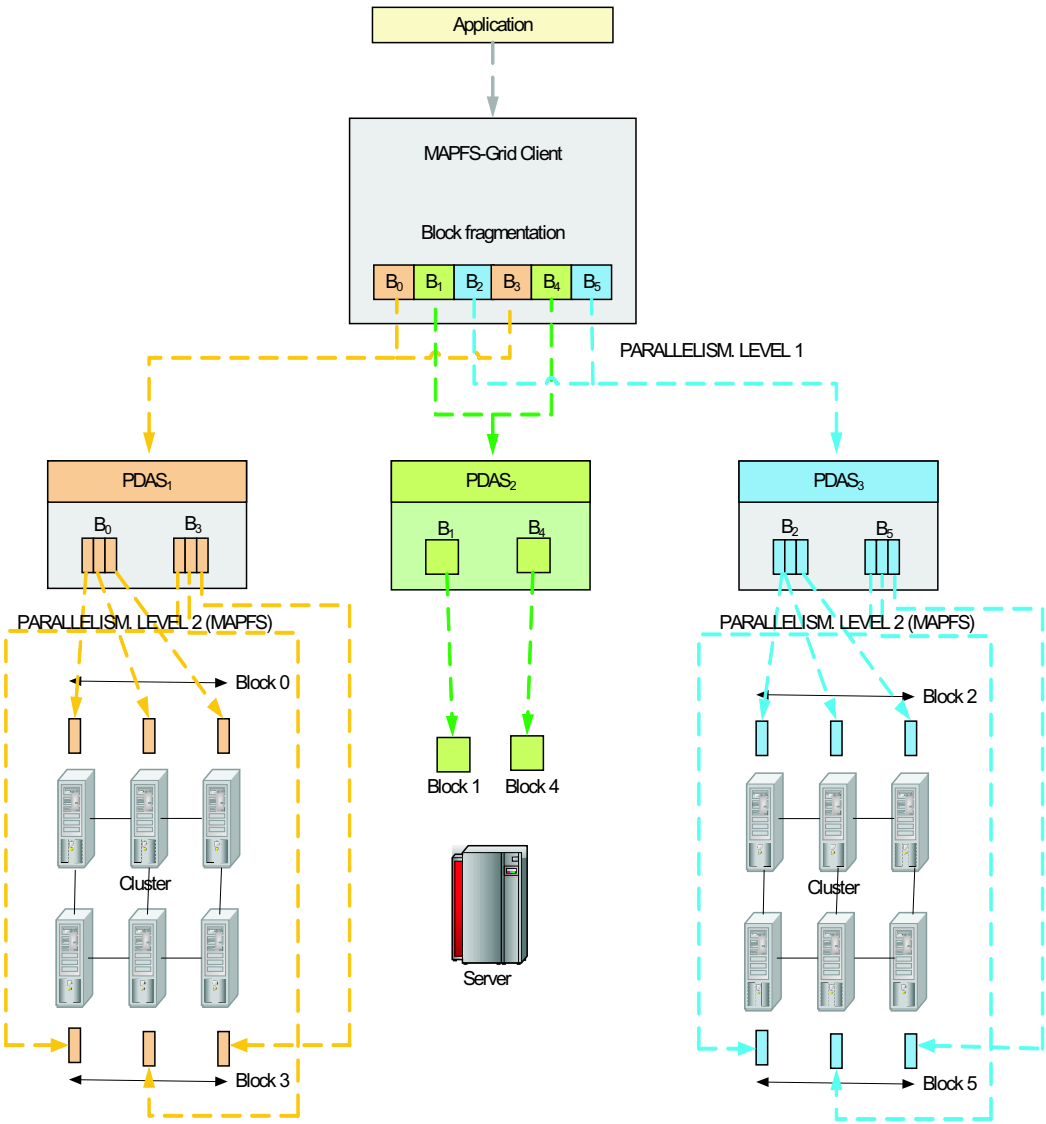

Fig. 1. MAPFS-Grid Overview

On the other hand, MAPFS is based on a multiagent architecture, named MAPFS_MAS, which provides support to the main subsystem (MAPFS_FS) in three different areas:

- Access to the information: This feature is the main task of MAPFS_MAS. Data is stored in I/O nodes (a set of disks distributed among several server nodes). Two different kind of agents are used for providing this capability: *Extractor agents* are responsible for invoking parallel I/O operations and *distributor agents* distribute the workload to extractor agents.
- Caching service: MAPFS takes advantage of the temporal and spatial locality of data stored in servers. A cache has a copy of the most recently used data in a storage device, which is faster than the original storage device. However, by using a cache causes an important coherence problem. Inside MAPFS_MAS, there is a set of agents which manage this feature. These agents are named *cache agents*. They are responsible for using a cache coherence protocol and control data transfer between both storage devices.
- I/O optimizations: MAPFS takes advantage of different I/O optimizations techniques, such as caching and prefetching or use of hints. The use of the agents methodology in this area makes flexible such I/O optimizations. For this proposal, *hints agents* are used.

Files are stored finally in several servers, which constitute the server-side of the underlying architecture. The grouping of servers from a logical point of view in MAPFS is denominated *storage group* [27]. The adaptation of this concept to a grid environment is explained in [28].

As we mentioned previously, the use of a multiagent system implies coordination among their agents. The main goal of the agents cooperation is the interaction among such agents for achieving a common objective in a distributed system.

2.2 Related Work

Nowadays, most of the frameworks are influenced by their environment. Thus, the environment conditions affect their performance in a dynamic way. For this reason, the use of the agent technology is being widely used, since this paradigm adapts to changing and dynamic environments. The agent paradigm is usually implemented on distributed systems.

The parallel execution of different processes allows a cluster of workstation to improve its performance. Agents are also useful for achieving this goal, since agents may run different tasks, which constitute partial solutions of the global aim.

Other intrinsic characteristic of agents is the cooperation. The principal goal of the cooperation among the agents is the interaction of such agents in order to achieve a common objective in the distributed space.

The agents cooperation can be made through a set of steps [15]:

- It is necessary to provide every agent *goals*, that is, descriptions of the desired state of the agents "world" or environment.
- Every agent must make a set of *actions* in order to modify their state. Also, it must build *plans* that contain precise instructions for achieving the goals or objectives.
- Every agent must have planned a set of *events*.
- In accordance with the planning, the agent must run the plan.
- The *cooperation* is achieved using *shared plans*, that is, making the planning in a shared way.

In a complex system, the interaction of several agents is required and, thus, a mechanism of communication between agents is necessary. For achieving agents communication and interoperability, it is necessary to use:

- A common language;
- common ideas about the knowledge agents interchange;
- capacity for interchanging this information.

For standardizing this way of communication, a common or standard language is used. KQML (Knowledge Query Manipulation Language) [6,1,8], is one of the most known agent communication languages. This language is composed of a set of messages, known as *performatives*, which are used for specifying agent communication elements. In [21], Labrou and Finin widely describe the KQML reserved performatives. Some of them are used in MAPFS.

The idea of using agents to access data is not an innovating idea. Nowadays, a great number of agents platforms are widely deployed for accessing web databases. The web popularity has created the need for developing Web Distributed Database Management Systems (DBMSs), obtaining simple data distribution, concurrency control and reliability. However, DBMSs offer limited flexibility, scalability, and robustness. Some suggestions propose the use of agents to solve this problem [29,24].

With respect to file accesses, several approaches have been made. Two paradigmatic approaches are the following:

- MESSENGERS [4] is a system based on agents used for the development and deployment of distributed applications from mobile agents, called *messengers*. This system is composed of a set of daemons distributed in every node an used for managing received agents, supervising their execution and planning where agents must be sent.

  Several aspects related to the system performance are adddressed in [13]. Some of them are load balancing, agent code optimization, and availability and efficient sharing of available resources.

  By analyzing the operation levels of an application, the data access performance influences in the system performance because I/O system is the bottleneck of most systems due to its access speed, that is notably smaller than the

memory access and the CPU speed. Improvements focused on improving data accesses will improve largely the system global performance .

The use of agents allows MESSENGERS to use a non-shared local file system, called LDFS [12], which is used for achieving the following goals:

· Local data access, which allows the access time to be reduced.

· Avoiding a bottleneck in the central server.

- DIAMOnDS [30] stands for Distributed Agents for MObile and Dynamic Services, a system built under Java/Jini. This system is composed of a client module that accesses data from a remote file system, where an agent is responsible for managing this interaction.

Other research projects about agent systems for accesing files have been developed. Nevertheless, there are not agent systems focused on the development of parallel file systems features. MAPFS constitutes a new approach of this kind of systems. MAPFS-Grid is its extension to a grid environment. The advantages of the application of Software Agents and MAS to Grid computing have been also considered in some papers [9,5]. Agents are characterised by their excellent negotiation abilities, and as Grid systems embrace service-oriented computing, a bigger emphasis is put on trading services between users and providers.

3 Generic structure of an agent in MAPFS-Grid

Agents provide a set of very interesting properties. Some of these characteristics are autonomy, reactivity and proactivity, which makes the system flexible for adapting to changing environments. According to Jenning et al. [20], the situation of the agent and the flexibility are additional agents features.

On the other hand, as is described in the previous section, there are different types of agents. Therefore, it is necessary to identify the role of every agent in the system. This method have been already identified and used in some agent architectures, such as MADKIT architecture [14]. This architecture defines the AGR model (*Agent-Group-Role*), in which the role or task of an agent constitutes one of the key concept. This role is the abstract representation of a function or service provided by the agent. Analogously, in MAPFS-Grid the role is used for setting the specific function of an agent.

Definition 1 *In MAPFS-Grid, an agent is defined in a formal way as the following tuple:*

$$< \texttt{Ag\_Id}, \texttt{Group}, \texttt{Role}, \texttt{Int\_Net} >$$

where:

- `Ag_Id:` *Agent identification, which is used in order to identify every agent of the system.*
- `Group:` *Storage group which the agent belongs to.*
- `Role:` *This field represents the kind of agent, taking values in the following domain:* `[Cache, Distributor, Extractor, Hint]`. *This domain can be increased with other values, if other kind of service must be implemented.*
- `Int_Net:` *This field represents the interaction network of an agent with other agents of its storage group. This network can be implemented as a vector or relations between the agent* `Ag_Id` *and the rest of agents of the same storage group.*

Agents cooperate in MAPFS-Grid in order to provide the overall functionality. The MAPFS-Grid cooperation model is described in the next section.

4 MAPFS-Grid Cooperation Model

For modeling agents cooperation, several related concepts have been formalized:

- Firstly, every agent must know its *goals*, that is, descriptions of the desired state of the agents "world" or environment. The goals depend on the kind of agent: extractor agents are subordinate to the distributor agents and they do not depend on the environment. However, distributor agents goals are completely dependent on the environment and are the most similar to the whole system goal. These goals correspond to the user requests. Cache agents goals correspond to the desires or requests of extractor agents. Every request of the information made by extractor agents is solved by cache agents. If data is not available in the cache structure, cache agents try to get data. Hints agents are only activated when optimization techniques are used in the MAPFS file system. Formally, agents goals can be notated and described in this way:
  · g_{da}**:** distributor agents goals
  · g_{ea}**:** extractor agents goals
  · g_{ca}**:** cache agents goals
  · g_{ha}**:** hints agents goals

$$g_{da}(x) = exists(d, G_y)$$

where x is a distributor agent belonging to any storage group

$S_z/S_z \in G_y$

$\wedge$ d is a concrete item

$\wedge$ *exists* is a predicate that indicates if a item is

available for an user in a storage group

$g_{ea}(x) = serves(x, y)$
where x is a extractor agent belonging to any storage group G_z/y is a distributor agent belonging to the same group
$\wedge$ *serves* is a predicate that indicates if x has satisfied the request of the agent y

$g_{ca}(x) = provides(x, y)$
where x is a cache agent belonging to any storage group G_z/y is a extractor agent belonging to the same group
$\wedge$ *provides* is a predicate that indicates if x has the data item requested by the agent y in the cache structure

$g_{ha}(x) = provides\_hints(x, y)$
where x is a hint agent belonging to any storage group G_z/y is a cache agent belonging to the same group
$\wedge$ *provides_hints* is a predicate that is false only when the agent x cannot get the metadata required by the agent y. Otherwise, the predicate is true

- According to agents goals, plans contain precise instructions or actions for achieving such objectives. Again, actions and plans depend on the concrete kind of agent:
  · p_{da}**:** distributor agents plans
  · p_{ea}**:** extractor agents plans
  · p_{ca}**:** cache agents plans
  · p_{ha}**:** hints agents plans

$p_{da}(x) = if \neg exists(d, G_y) \longrightarrow \forall y$ / $is\_a\_ea(y)$
$\wedge$ y $\in G_y$ then $ask(d, y)$
where x is a distributor agent belonging to any storage group
$S_z/S_z \in G_y$
$\wedge$ d is a concrete item
$\wedge$ *is_a_ea* is a predicate that is true if y is
an extractor agent and false otherwise
$\wedge$ *ask* is a function that generates an event for
asking the retrieval of the item d by the agent y

$p_{ea}(x) = if \neg serves(x, y) \wedge is\_asked(y, d) \longrightarrow$
$\forall z$ / $is\_a\_ca(z)$ then ask(d,z)
where x is a extractor agent belonging to any storage group
G_z/y is a distributor agent belonging to the same group
$\wedge$ d is a concrete item
$\wedge$ *is_a_ca* is a predicate that is true if z is a cache agent
and false otherwise
$\wedge$ *ask* is a function that generates an event for asking
the retrieval of the item d by the agent z

$p_{ca}(x) = if \neg provides(x, y) \wedge is\_asked(y, d) \longrightarrow$
$obtain(d)$
where x is a cache agent belonging to any storage group
G_z/y is a extractor agent belonging to the same group
$\wedge$ *obtain* is a function used for obtaining the data item
from the disk and store it in the cache structure

$p_{ha}(x) = if \neg provides\_hints(x, y)$
$\wedge is\_asked(y, h) \longrightarrow obtain(h)$
where x is a hint agent belonging to any storage
group G_z/y is a cache agent belonging to the same group
$\wedge$ *obtain* is a function used for obtaining metadata
and providing it to agent y

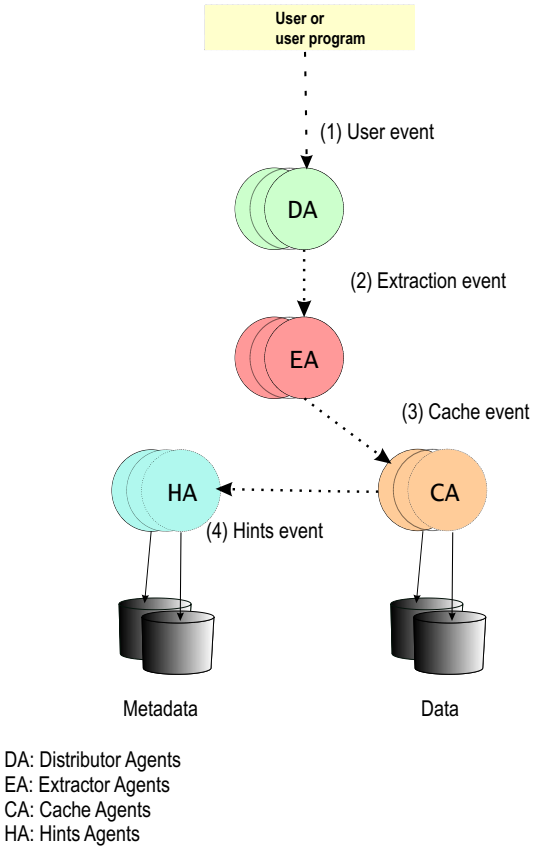

Fig. 2. Events tree in MAPFS-Grid

The function *obtain* corresponds to a read/write operation of the parallel file system.

There are two kinds of events: (i) events originated by the user or by the user applications and (ii) events originated by agents. The first kind of event is the original source of event, because only when a user or a user application make a I/O request, the *events tree* is initiated. Such tree is depicted in Figure 2.

- Every agent must have planned a set of *events*, which must be managed As can be seen in Figure 2, the order in which events are generated is the following one: (i) firstly, an user or user program makes an I/O request. This one generates an user event, which is caught by one distributor agent; (ii) this one generates an extraction event, directed towards an extractor agent, which is responsible for obtaining/storing data; (iii) usually, the extractor agent looks up the data item in the cache, *delegating* to a cache agent for this task; the cache agent must both return data to the extractor agent and store them in the cache structure; (iv) if the system uses some optimization technique, it is necessary to use hints agents. In this case, hints agents are responsible for obtaining metadata from disks.
- According to the planning, the agent must run the plan. The planning model is *event-driven*. If an event is generated and the premises are true, the correspond actions are executed, modifying the system state.
- The *cooperation* is achieved using *shared plans*, that is, making the planning

in a shared way. In this case, the cooperation is achieved through two different schemas: (i) there are replicas of all the agents; these agents must coordinate their efforts with the aim of satisfying the system goals; e.g. the cache structure must be divided into sections and each cache agent must manage one section; the distributor agent is responsible for distributing the work. This planning is denominated *intra-planning*. (ii) Every storage group must interoperate with the rest of the storage groups in order to plan the system. This planning is named *inter-planning*.

According to the MAPFS-Grid cooperation model, a set of performatives has been defined. In order to define MAPFS-Grid performatives, several sets of elements are defined for a concrete storage group:

DA: Set of distributor agents
EA: Set of extractor agents
CA: Set of cache agents
HA: Set of hints agents

Next, these performatives are described.

When an element `d` is requested, a distributor agent is responsible of asking data to several extractor agents. Let `x` be a distributor agent of a storage group G_x. Figure 3(a) includes the KQML performative of the distributor agent.

If the extractor agent has the element `d`, then such agent does the performative of Figure 3(b), indicating that the data item `d` is available in the storage group G_x.

On the other hand, if the extractor agent has not the element `d`, that is, the element is not in the cache structure, the extractor agent does the performative of Figure 4, asking required data to all the cache agents.

The predicate `ask(d,z)` in the cache agent `z` involves the execution of the MAPFS function `obtain(d)` (read operation).

Next, the cache agent sends information about the finish of the operation to the distributor agent, through the extractor agent, indicating that the element `d` is available in the storage group G_x. This process is made by means of the performative of Figure 5.

Thus, the cycle is closed. Nevertheless, the cache structure has a maximum number of entries, which must be replaced by other elements with a concrete replace policy. When the entry is invalidated, the cache agent `z` sends the performative represented in Figure 6.

Cache agents use metadata provided by hints agents, sending the performative of Figure 7(a). In this way, metainformation identified by *h* is required.

Step 1

$x \in DA$

$y \in EA$

(ask-if

:sender x

:receiver y

:reply-with id_da

:language Prolog

:ontology MAPFS-Grid

:content "exists(d,G_x)")

(a) ***Performative for the data request from a distributor agent to an extractor agent.***

Step 2.1

(tell

:sender y

:receiver x

:in-reply-to id_da

:reply-with id_ea

:language Prolog

:ontology MAPFS-Grid

:content "exists(d,G_x)")

(b) ***Response performative from an extractor agent to a distributor agent, if the agent has the required data.***

Fig. 3. Performatives related to a distributor agent

A hint agent build the required metainformation, sending it to the cache agent by means of the performative of Figure 7(b).

Step 2.2

$z \in CA$

(achieve
:sender y
:receiver z
:in-reply-to id_da
:reply-with id_ea
:language Prolog
:ontology MAPFS-Grid
:content "`ask(d,z)`")

Fig. 4. Performative for the data request from an extractor agent to a cache agent

Step 3.1

(forward
:from z
:to x
:sender z
:receiver y
:reply-with id_ca
:language KQML
:ontology kqml-ontology
:content (achieve
:sender z
:receiver x
:in-reply-to id_ea
:reply-with id_ca
:language Prolog
:ontology MAPFS-Grid
:content "`exists(d,`G_x`)`")

Fig. 5. Response performative from a cache agent to a distributor agent, once data are obtained

Invalidation

(forward
:from z
:to x
:sender z
:receiver y
:reply-with id_ca'
:language KQML
:ontology kqml-ontology
:content (unachieve
:sender z
:receiver x
:in-reply-to id_ea
:reply-with id_ca'
:language Prolog
:ontology MAPFS-Grid
:content "exists(d,G_x)")

Fig. 6. Performative of invalidation of data in the cache

5 Implementation and evaluation

Agents are useful in the design of a complex system, and, concretely in the design of a parallel file system, as we have shown in previous sections. Nevertheless, it is necessary to validate this paradigm within this field, evaluating the increase of the performance of the implementation of MAPFS and its multiagent subsystem. MAPFS-Grid provides a grid-like interface to this parallel file system, allowing applications to take advantage of two levels of parallelism. MAPFS-Grid has been implemented by using Globus Toolkit 4 [10], which is based on the WSRF specification [22].

The enhancement provided by agents is given at the second level of parallelism, that is, the parallelism provided by MAPFS. Thus, it is important to evaluate how agents are implemented and affect to this parallel file system.

The implementation of the MAPFS multiagent subsystem is based on MPI technology, mainly for the following reasons:

Step 3.2

$u \in HA$

(achieve

:sender z

:receiver u

:reply-with id_ca”

:language Prolog

:ontology MAPFS-Grid

:content “ask(h,v)”)

(a) ***Performative for the hint request from an cache agent to a hint agent.***

Step 4.1

(tell

:sender u

:receiver z

:in-reply-to id_ca”

:reply-with id_ha

:language Prolog

:ontology MAPFS-Grid

:content “exists(h,G_x)”)

(b) ***Response performative from a hint agent to a cache agent, once hints are obtained.***

Fig. 7. Performatives related to a hint agent

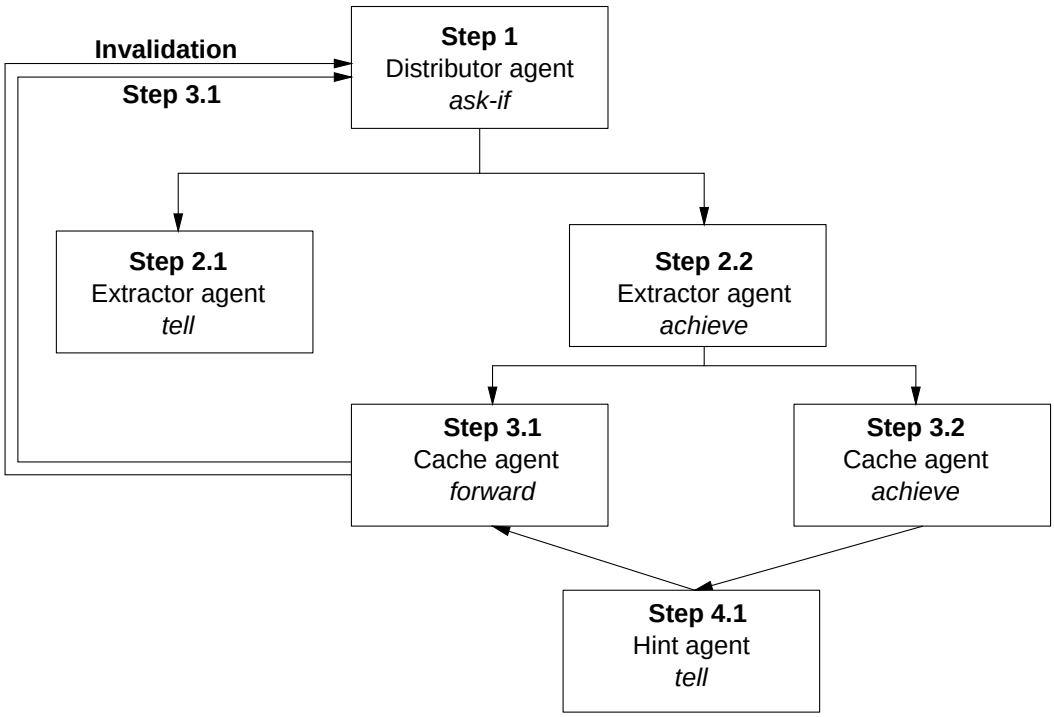

Fig. 8. Control flow of system performatives

(1) MPI is an standard message-passing interface, which allows system agents 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 a dynamic management of processes.
(5) Finally, MPI provides operations for modifying the communication topologies.

These features of MPI are used for providing the following properties of agents:

- *Autonomy:* MPI is able to create dynamically independent and autonomous processes with communication capacities.
- *Reactivity:* Agents react to the environment or changes in other processes by means of a MPI message. In fact, KQML performatives are translated to MPI messages by MAPFS, as it is described below.
- *Proactivity:* The study of the behaviour of the system by agents provides the capability of taking decisions in advance.

KQML defines an abstraction for transport for agent communication. KQML can be implemented with different solutions. MPI is a good choice, since this technology fulfill the requirements of KQML performatives. For instance, the translation of the `ask-one` performative into a MPI message corresponds to the `MPI_Send` primitive. In the same way, all the MAPFS-Grid performatives can be translated.

It is important to emphasize that MPI only solves the communication problem. The semantic is provided by the messages content, by means of MPI structures and the processing of such message by the receiver agent.

A MAPFS multiagent subsystem responsible for prefetching data has been implemented and evaluated. This evaluation has demonstrated that the use of a multiagent subsystem provides a average speedup close to 4, as is shown in Figure 9. The speedup depends on the access block size.

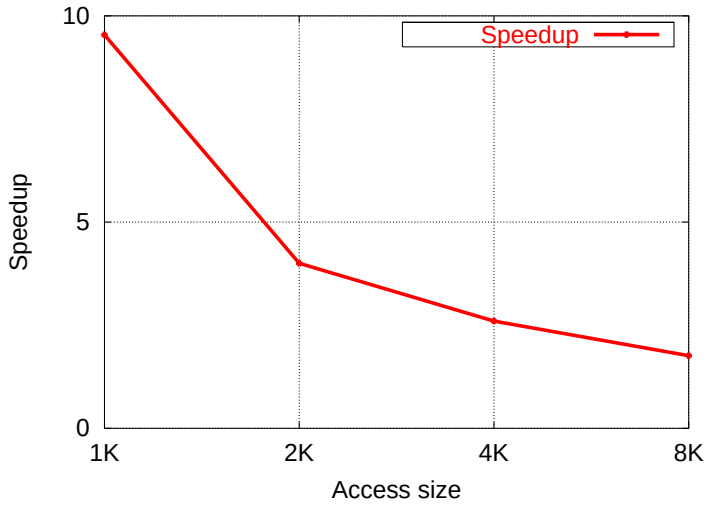

Fig. 9. Evaluation of the use of agents in MAPFS-Grid

6 Conclusions and Future Work

The agent paradigm has been traditionally linked to the Artificial Intelligence field. Additionally, agents have been successfully used in several domains.

The novelty of our proposal is the use of a multiagent subsystem in a parallel file system, which is used by a grid framework for providing access to data resources, that is, MAPFS-Grid. As we have evaluated in this paper, the I/O architecture takes advantage of the agent theory. This paper also describes the agent coordination and communication features of MAPFS-Grid, an I/O architecture for grids.

It is important to distinguish between the conceptual model of MAPFS-Grid and its implementation. The model is based on the agent theory, but the implementation is solved by means of technologies closer to the system programming, as MPI and Globus.

MAPFS-Grid provides two levels of parallelism. In this work, the agents are deployed in the second level of parallelism, that is, at file system level. Currently, we are working on adding agents to the first level of parallelism, in order to improve the performance of the global architecture.

References

[1] ARPA knowledge sharing initiative. specification of the KQML agent-communication language. *External Interfaces Working Grop working paper*, 1993.

[2] E. M. Atkins, E. H. Durfee, and K. G. Shin. Autonomous flight with CIRCA-II. In *Autonomous Agents'99 Workshop on Autonomy Control Software*, May 1999.

[3] C. P. Azevedo, B. Feiju, and M. Costa. Control centres evolve with agent technology. *IEEE Computer Applications in Power*, 13(3):48–53, 2000.

[4] Lubomir Bic, Munehiro Fukuda, and Michael B. Dillencourt. Distributed programming using autonomous agents. *IEEE Computer*, 29(8):55–61, 1996.

[5] D.G. Cameron, R. Carvajal-Schiaffino, C. Nicholson, K. Stockinger, F. Zini, A.P. Millar, and L. Serafini. Analysis of an agent-based grid optimisation strategy. *Software Agents and Grid Computing*, 2004.

[6] H. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, and G. Wiederhold. An overview of KQML: A knowledge query and manipulation language. Technical report, Computer Science Department. Stanford University, April 1992.

[7] D. Cockburn and J. Nick. *ARCHON: A distributed artificial intelligence system for industrial applications*, pages 319–344. 1996. Article belonging to [23].

[8] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. *"Software Agents", MIT Press. Cambridge*, 1997.

[9] Ian Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn - why grid and agents need each other. In *Proc. 3rd Int. Conf. on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2004.

[10] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[11] M. Garijo, A. Cáncer, and J. J. Sánchez. A multiagent system for cooperative network-fault management. In *Proceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology, PAAM-96*, pages 279–294, April 1996.

[12] E. Gendelman, L. Bic, and M. Dillencourt. Ldfs: A fault-tolerant local disk-based file system for mobile agents.

[13] Eugene Gendelman, Lubomir F. Bic, and Michael B. Dillencourt. Fast file access for fast agents. *Proceedings of the 5th International Conference, MA 2001.*, 2240:88–102, 2001.

[14] Olivier Gutknecht and Jacques Ferber. The MADKIT agent platform architecture. *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, pages 48–55, March 2001.

[15] H. Haugeneder and D. Steiner. Cooperating agents: Concepts and applications. In *Proceedings of the Agent Software Seminar. London, England. Unicom Seminars Ltd*, pages 80–106, 1995.

[16] Staffan Hägg. Agent technology in industrial applications. In *Proceedings of the Australia-Pacific Forum on Intelligent Processing and Manufacturing of Materials (IPMM'97)*, 1997.

[17] N. R. Jennings. Using GRATE to build cooperating agents for industrial control. In *Proceedings of the IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real Time Control*, pages 691–696, 1992.

[18] N. R. Jennings, J. M. Corera, L. Laresgoiti, E. H. Mamdani, F. Perriollat, P. Skarek, and L. Z. Varga. Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control. *IEEE Expert*, 1995.

[19] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, M. E. Wiegand, C. Voudouris, J. L. Alty, T. Miah, and E. H. Mamdani. ADEPT: Managing business processes using intelligent agents. In *Proceedings of the BCS Expert Systems 96 Conference, Cambridge, UK*, pages 5–23, 1996.

[20] 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.

[21] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, Baltimore, MD 21250, 1997.

[22] OASIS. Ws-ResourceFramework. 2005.

[23] G. M. O'Hare and N. R. Jennings, editors. *Foundations of Distributed Artificial Intelligence*. John Wiley and Sons, 1996.

[24] Stavros Papastavrou, George Samaras, and Evaggelia Pitoura. Mobile agents for world wide web distributed database access. *Knowledge and Data Engineering*, 12(5):802–820, 2000.

[25] María S. Pérez, Jesús Carretero, Félix García, José M. Peña, and Víctor Robles. MAPFS: A flexible multiagent parallel file system for clusters. *Future Generation Comp. Syst.*, 22(5), 2006.

[26] María S. Pérez, Jesús Carretero, Félix García, José M. Peña Sánchez, and Victor Robles. MAPFS-Grid: A flexible architecture for data-intensive grid applications. In F. Fernández Rivera, Marian Bubak, A. Gómez Tato, and Ramon Doallo, editors, *European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 111–118. Springer, 2003.

[27] María S. Pérez, Alberto Sánchez, José Manuel Peña, and Víctor Robles. A new formalism for dynamic reconfiguration of data servers in a cluster. *J. Parallel Distrib. Comput.*, 65(10):1134–1145, 2005.

[28] María S. Pérez, Alberto Sánchez, Víctor Robles, José M. Pe na, and Francisco Rosales. Grid Storage Groups: A bridge between data-based clusters and data grid architectures. *Journal of Parallel and Distributed Computing Practices. Special Issue on Grid Computing Infrastructures and Applications*, 2006.

[29] K. Segun, A. Hurson, V. Desai, A. Spink, and L. Miller. Transaction management in a mobile data access system. *Annual Review of Scalable Computing*, 3:85–147, 2001.

[30] Aamir Shafi, Umer Farooq, Saad Kiani, Maria Riaz, Anjum Shehzad, Arshad Ali, Iosif Legrand, and Harvey Newman. DIAMOnDS - DIstributed Agents for MObile and Dynamic Services. In *Proceedings of the 2003 Conference for Computing in High Energy and Nuclear Physics (CHEP03)*, La Jolla (California), March 24-28 2003.