Cooperation Model of a Multiagent Parallel File System for Clusters

María S. Pérez, Alberto Sánchez, Víctor Robles and José M. Peña
Facultad de Informática
Universidad Politécnica de Madrid, Spain
e-mail: {mperez,ascampos,vrobles,jmpena}@fi.upm.es

Jemal Abawajy
School of Computer
Carleton University. Ottawa, Canada
e-mail: abawjem@scs.carleton.ca

Abstract

MAPFS is a parallel file system integrated with a multiagent system responsible for the information retrieval. One of the fields where the agents can be very useful is precisely in the development of information recovery systems. The usage of a multiagent system implies coordination among the agents that belong to such system. The main goal of the agents cooperation is the interaction among them for achieving a common objective in a distributed system. Thus, a communication framework must be provided. This paper shows the MAPFS cooperation model and its communication framework, emphasizing its relation with the whole system.

1. Introduction

MAPFS is a MultiAgent Parallel File Systems for clusters and offers a file system interface that includes traditional, advanced, collective, caching operations and hints [13]. MAPFS consists of two subsystems with two clearly defined tasks: (i) MAPFS_FS, which provides the parallel file system functionality and (ii) MAPFS_MAS, responsible for the information retrieval. MAPFS_MAS is an independent subsystem, 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 are stored in I/O nodes (a set of disks distributed among several server nodes).

- 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, 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 (described above) and usage of hints. The use of the agents methodology in this area makes flexible the use of such I/O optimizations. For this proposal, *hints agents* are used.

With the aim of achieving these goals, MAPFS_MAS is constituted by a set of agents which interact among them, that is, a *multiagent system* (MAS). In this case, agents collaborate in order to provide the features mentioned above. Agents must be reconfigured because of the dynamic and changing environment in which they coexist. These agents adapt their behavior depending on the response of the medium and their own learning. In this paper the MAPFS cooperation model is shown, emphasizing its relation with the whole system.

This paper, whose main purpose is to describe the coordination and communication features of MAPFS, is organized as follows. Section 2 presents an overview of cooperative systems. Section 3 shows the MAPFS architecture. Section 4 analyzes the MAPFS cooperation model and describes the communication features of MAPFS. Finally, section 5 summarizes our conclusions and outlines the future work.

2 Background

There is not a precise definition of an *agent*. In fact, there are many definitions of agents. Franklin and Graesser define an agent, using its feature of *autonomy* [7]: "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future". Wooldridge and Jennings focused the definition of an agent on its properties [17]: "an agent is used to denote a hardware or software-based computer system that enjoys the following properties:

- autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- reactivity: agents perceive the environment and respond in a timely fashion to changes that occur in it;
- pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative."

This paper focuses on the social ability property of an agent.

Russell and Norvig define an agent in three steps [15], ordered by increasing complexity:

- Generic agents: "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."
- Rational agents: "An ideal rational agent should do whatever is expected to maximize its performance measure, on the basis of the evidence provided by the percepts' sequence and whatever built-in knowledge the agent has."
- Autonomous agents: "An agent is autonomous to the extent that its actions and choices depend on its own experience, rather than on knowledge of the environment that has been built-in by the designer."

Other authors emphasize several optional features of the agents such as mobility or intelligence in order to define them [16],[10].

On the other hand, the main goal of the cooperation among agents is the interaction of such agents in order to achieve a common objective in the distributed space. The two main branchs in the design of methods and architectures related to this problem are: (i) Multi-agent systems (MAS) and (ii) Distributed Problem Solving (DPS) [4]. Both fields overlap their objectives and give them feedback, although only the MAS field is related to the topic of agents, and therefore, we focus our work on this one.

Agents cooperation can be made by means of a set of steps [9]:

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

One important aspect of the cooperation is the *communication* among agents. For obtaining the communication and interoperability, it is necessary to use:

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

For standardizing this way of communication, a common or standard language is used. In this sense, KSE (Knowledge Sharing Effort) has several research lines [1], [2], [8].

There are specific agent languages, oriented to communication of agents. KQML (Knowledge Query Manipulation Language) [3], [2], [5], 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 [11], Labrou and Finin widely describe the KQML reserved performatives. Some of them are used in MAPFS.

Some researchers have built generic multi-agent system architectures, which can be used in different domains. Particularly, Flores and Wijngaards describes a generic MAS architecture for dynamic collaboration in an open environment [6]. This work, like ours, considers the agent as a social entity and emphasizes the importance of the social factor in the agents interaction.

3. MAPFS Architecture

As we mentioned previously, MAPFS consist of two subsystems: MAPFS_FS and MAPFS_MAS.

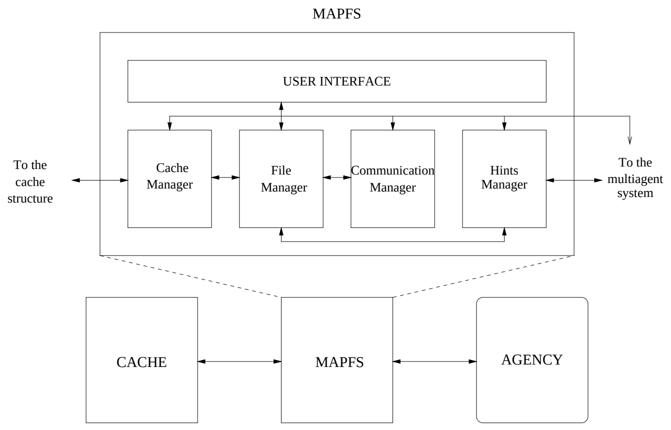

Figure 1. MAPFS Structure

MAPFS_MAS is the subsystem responsible for the information retrieval. Files are stored in 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* [14].

On the other hand, MAPFS has to provide both an user interface and connection capacity through a network in a distributed system. These task are responsibility of two different modules in the architecture: the *user interface* and the *communication manager*, respectively.

Besides, the file system uses hints that provides optimization techniques to the whole system. This functionality is implemented by the *hint manager*, which communicates with an *agency*. In such way, the agency provides autonomy to the system. For this reason, hints agents are used in MAPFS.

Finally, MAPFS manages different caches of the file system. The responsible module is the *cache manager*. This manager communicates with both a cache structure and the multiagent system. For this aim, there are cache agents in MAPFS.

In order to assemble all the file system functionality and build the file model, the *file manager* is used. This subsystem constitutes the central core of the file system, and therefore, the main module of MAPFS_FS.

Figure 1 shows the MAPFS architecture, where the component modules are shown.

The multiagent system consists of a set of agents, with the following features:

- Agents management must be *transparent*, in the same way an ideal distributed system is.

- The *mobility* makes possible the code transfer between two different nodes within a system. This transfer can be very useful in an information recovery system, and particularly in MAPFS. Mobile agents are useful in three general areas. One is disconnected computing, such as laptops and PDAs. The second is dynamic deployment of software. The third category is information retrieval systems, that is, applications where the agent can be sent to the data source and carry out the process of filtering data. MAPFS can take advantage of this last feature. In fact, mobile agents have a number of key features desirable for a network system [12]:

  – Network load reduction: The network traffic in a distributed system is a “bottleneck”. With the usage of mobile agents, this network load can be avoided, since this kind of agents is based on the idea of migrating computations to data rather than data to computations.
  – Network latency decrease: The use of the network increases its latency. Mobile agents overcome network latency due to their local execution.
  – Asynchronous and autonomous execution: Mobile agents can operate asynchronously and autonomously respect to their parent process.
  – Dynamic adaptation: Mobile agents have the ability to perceive their environment and react to changes.
  – Heterogeneity: Mobile agents often depend on their execution environment, but they are usually computer independent.
  – Robustness and fault tolerance: Mobile agents can be dispatched to another host in the network when the source host fails.

  All this features are desirable for the information retrieval in a distributed system.

- MAPFS uses 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 the responsible for the information retrieval in the MAPFS file subsystem.
  – Distributor agents: They distribute the workload to extractor agents. These agents are placed at a higher level in the agents hierarchy.
  – Cache agents: They are associated with one or more extractor agents, caching their data. This relation is susceptible to modify. These agents are responsible for the following tasks: (i) synchronization between the cache structure and the storage device and (ii) cache coherence.

- Hints agents: They are used in order to increase the performance of the I/O system, by means of the definition of hints about the usage and layout of such system.

- As we mentioned previously, the usage of a multiagent system implies *coordination* among the system agents.

4. MAPFS Cooperation Model

The dynamic behavior of MAPFS agents is originated in two sources:

1. The dynamic and changing environment in which they coexist.

2. The messages sent and received by agents with the aim of collaborating in the information retrieval.

Therefore, MAPFS agents adapt their behavior depending on the response of the environment and their partners agents.

4.1 MAPFS Cooperation Features

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 are not available in the cache structure, cache agents aim to get data. Hints agents are only activated when optimization techniques are used in MAPFS. 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 server $S_z / S_z \in G_y$
$\wedge$ 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 server
$S_z / S_z \in G_y$
$\wedge$ 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

The function *obtain* and *obtain_duplicate()* belong to the MAPFS_FS and are parallel file system read/write operations.

- Every agent must have planned a set of *events*, which must be managed by such agents. 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.

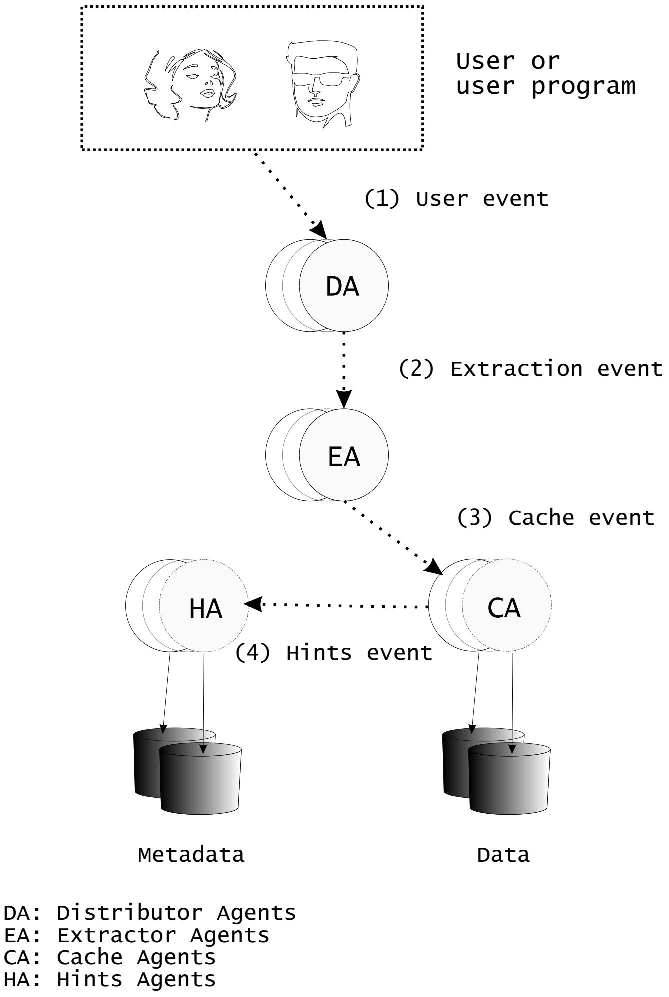

Figure 2. Events tree in MAPFS

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*.

4.2 Putting it together

In this section, we describe how the cooperation model fits into the MAPFS architecture, analyzing the contents of every module:

- User Interface: This module provides access to the MAPFS functionality, offering the MAPFS interface, that is, the set of parallel I/O operations. Therefore, the cooperation model is not related to this module.

- Communication Manager: This module is responsible for three tasks: (i) transference of information in MAPFS; (ii) communication between the cache manager and the file storage server with the aim of updating the cache and (iii) distribution of the agents in the MAPFS system. This module is the distribution channel of the agents belonging to all the MAS.

- Cache Manager: This module is responsible for modeling the cache coherence protocol, which is implemented by cache agents.

- Hints Manager: This module provides capacity for using different optimization techniques by means of hints. This feature is implemented by hints agents.

- File Manager: This module implements the MAPFS_FS subsystem functionality. Extractor agents make requests to this module.

- Agency: It is the software container where the different agents are created and develop their work. Therefore, the communication manager, the cache manager, the hints manager and the file manager are linked to the agency.

Goals and plans must be implemented in the respective modules:

- Distributor agents goals and plans are situated in the Communication Manager.

- Extractor agents goals and plans are situated in the File Manager.

- Cache agents goals and plans are situated in the Cache Manager.

- Hints agents goals and plans are situated in the Hints Manager.

In order to use a standard communication language, we have used KQML as the format of the messages interchanged by MAPFS agents. KQML is a language and an associated protocol to support high level communication among several agents. The standard gives us a large set of primitives to be used to build KQML messages. Since KQML can be used to obtain information from any agent, it seems appropriate to consider it as a good candidate to communicate all the MAPFS agents.

As an example, Figure 3 shows the KQML response performative from a hint agent to a cache agent, once hints are obtained.

```
(tell
   :sender       x
   :receiver      y
   :in-reply-to   id_ca
   :reply-with   id_ha
   :language    Prolog
   :ontology    MAPFS
   :content     "exists(h,Gx)")
```

Figure 3. Response performative from a hint agent to a cache agent, once hints ar e obtained

5. Conclusions and Future Work

This paper describes the coordination and communication features of MAPFS. First of all, an overview of cooperative systems is presented. Next, this paper describes the

MAPFS architecture. As can be seen, MAPFS consists of two subsystems: MAPFS_FS and MAPFS_MAS. This paper focuses on the second subsystems. Within this subsystem, the MAPFS cooperation model is analyzed, describing the social features of the agents used in MAPFS_MAS and showing a sample KQML performative used by hints agents in MAPFS.

The main contribution of this work is the usage of the agent technology in the implementation of a parallel file system, so that MAPFS benefits from the advantages of this technology and its coordination and communication model.

As future work, we will evaluate the performance of MAPFS_MAS and the overload of MAPFS performatives in the system. Also, MAPFS_MAS can be enhanced, by means of the addition of mobility.

References

[1] American National Standard. Knowledge Interchange Format. *Draft Proposed American National Standard (dpANS), NCITS.T2/98-004*, 1998.

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

[3] 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.

[4] E. Durfee and J. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. In *Proceedings of the Thirteenth International Distributed Artificial Intelligent Workshop*, 1994.

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

[6] Roberto A. Flores and Niek J.E. Wijngaards. Primitive interaction protocol for agents in a dynamic environment. In *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, October 1999.

[7] 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.

[8] T. R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220, 1993.

[9] 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.

[10] Michael Knapik and Jay Johnson. *Developing Intelligent Agents for Distributed Systems*. Computing McGraw-Hill, 1998.

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

[12] D. B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents With Aglets*. Addison-Wesley Pub Co, August 1998.

[13] 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.

[14] María S. Pérez, Alberto Sánchez, José M. Peña, Víctor Robles, Jesús Carretero, and Félix García. Storage groups: A new approach for providing dynamic reconfiguration in data-based clusters. In *2004 IASTED Conference on Parallel and Distributed Computing and Networks (PDCN 2004)*, February 2004.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ. Prentice-Hall, 1994.

[16] Tony White and Bernard Pagurek. Emergent behavior and mobile agents. 1999.

[17] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 1995.