Design and Evaluation of an Agent-Based Communication Model for a Parallel File System

María S. Pérez[1], Alberto Sánchez[1], Jemal Abawajy[2], Víctor Robles[1], and José M. Peña[1]

[1] DATSI. FI. Universidad Politécnica de Madrid. Spain
[2] School of Computer. Carleton University. Ottawa, Canada

Abstract. Agent paradigm has become one of the most important topics appeared and widely developed in computing systems in the last decade. This paradigm is being sucessfully used in a large number of fields. MAPFS is a multiagent parallel file system, which takes advantage of the semantic concept of agents in order to increase its modularity and performance. This paper shows how to use agent theory as conceptual framework in the design and development of MAPFS. MAPFS implementation is based on nearer technologies to system programming, although its design makes usage of the abstraction of a multiagent system.

Keywords: Agent, multiagent system, parallel file system, prefetching.

1 Introduction

Agent technology constitutes a new computing paradigm. Despite agents are very related to the Distributed Artificial Intelligence (DIA) area [2], some works have demonstrated that the agent technology can be used in fields totally different to the DIA. In the last decade, a large number of applications have appeared in business [6], electric management [11], control [3], or industrial applications in general [10].

Agent technology provides several concepts, which allow programmers to analyze and design applications in a way close to the natural language. Furthermore, agents give applications a set of useful features for tackling complex and dynamic environments.

On the other hand, there are important differences between *system programming* and agent technology. Firstly, agent paradigm interacts with the system at a higher level than system programming. Furthremore, 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.

This paper shows how to use agent theory as conceptual framework in the design and development of a parallel multiagent system, called MAPFS (MultiAgent Parallel File System), using nearer technologies to system programming, but taking advantage of the semantic concepts of agents.

The outline of this paper is as follows. Section 2 presents the overview of MAPFS, focusing on the multiagent subsystem of such file system, and describes the related work.

A. Laganà et al. (Eds.): ICCSA 2004, LNCS 3044, pp. 87–96, 2004.

Section 3 describes our proposal for the generic structure of an agent in MAPFS. Section 4 shows the implementation and evaluation of MAPFS, in order to measure the influence of the agents in the parallel file system. Finally, Section 5 summarizes our conclusions and suggests further future work.

2 Problem Statement and Related Work

2.1 MAPFS Overview

MAPFS is a multiagent parallel file system for clusters, which provides a file system interface that includes traditional, advanced, collective, caching, and hints operations [14]. 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. In order to provide data to MAPFS_FS, MAPFS_MAS is constituted by a set of agents which interact among them, that is, a *multiagent system* (MAS). The use of a MAS implies coordination among their agents. The cooperation model of MAPFS is defined in [15]. 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. 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: (i) Extractor agents, responsible for information retrieval; (ii) distributor agents, which distribute the workload to extractor agents; (iii) caching and prefetching agents, associated with one or more extractor agents, caching or prefetching their data; and (iv) hints agents, which must study applications access patterns to build hints for improving data access.

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 named *storage group*.

2.2 Related Work

Nowadays, most of the frameworks are influenced by their environment, so that the environment conditions affect their performance in a dynamic way. For this reason, the usage 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.

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: (i) A common language; (ii) common ideas about the knowledge agents interchange; and (iii) 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]. This paper focuses on the ability of agents for communicating among them and developing a specific task within MAPFS.

As it is shown in [13], the usage of agents simplifies the distribution and optimizes the messages interchange among them. The idea of using agents to access data is not

an innovating idea. Nowadays, a great number of agents platforms are widely deployed for accesing web databases. Different access methods are used, such as JDBC. The web popularity has created the need for developing Web Distributed Database Management Systems (DBMS), obtaining simple data distribution, concurrency control and reliability. However, DBMS offer limited flexibility, scalability, and robustness. Some suggestions propose the use of agents to solve this problem [16]. With respect to file accesses, several approaches have been made. Two paradigmatic approaches are described next.

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 features are defined in [8] in order to measure the system performance. Some of them are load balancing, agent code optimization and availability and efficient sharing of available resources.

DIAMOnDS [17] 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 of a remote file system, where an agent is responsible of 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.

3 Generic Structure of an Agent in MAPFS

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. Furthermore, an aditional characteristic very related to agents and useful in the case of the MAPFS system is the intelligence. Intelligent agents usually take decisions in the system. In this context, MAPFS agents are responsible for building hints dynamically, modifying them according to the acquired knowledge in the process of analysis of data patterns.

On the other hand, as is described in the previous section, there are different kind 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 [9]. 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 the role is used for setting the specific function of an agent.

Definition 1 *In MAPFS, 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.*

A key aspect of a multiagent system is the communication among agents. There are specific agent languages, oriented to communication of agents. KQML (Knowledge Query Manipulation Language) [7], 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 [12], Labrou and Finin widely describe the KQML reserved performatives. Some of them are used in MAPFS.

According to the MAPFS cooperation model, a set of performatives has been defined. In order to define MAPFS 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 section defines KQML performatives for the interaction among agents.

3.1 MAPFS Performatives

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 1(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 1(b), indicating that the data item `d` is available in the storage group G_x.

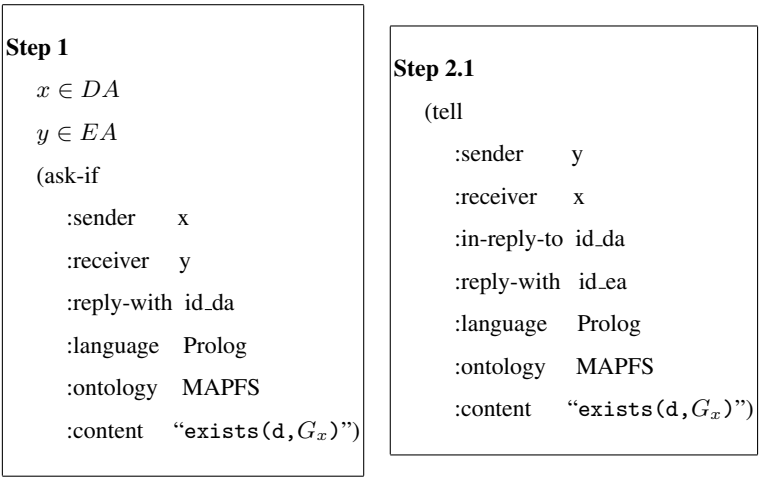
On the other hand, if the extractor agent has not the element `d`, that is, the element is not in the cache structure, the extractor agent does the performative of Figure 2, asking required data to all the cache agents. The predicate `ask(d,z)` in the cache agent `z` involves the execution of the MAPFS function `obtain(d)` (read operation).

Next, the cache agent sends information about the completion 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 corresponds to the performative of Figure 3. 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 4.

Cache agents use metadata provided by hints agents, sending the performative of Figure 5(a). In this way, metainformation identified by *h* is required. A hint agent build the required metainformation, sending it to the cache agent by means of the performative of Figure 5(b). Figure 6 shows the control flow of system performatives.

3.2 Cache Agent. A Sample Agent

A cache agent is a sample MAPFS agent. In order to increase the efficiency of the I/O system, cache agents make prefetching and caching tasks. Prefetching is used for



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

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

Fig. 1. Performatives related to a distributor agent

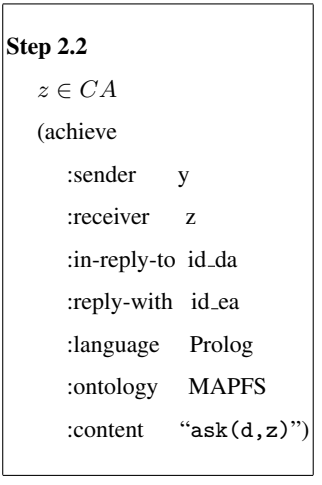


Fig. 2. Performative for the data request from an extractor agent to a cache agent

increasing the performance of read operations, since it is possible to read in advance data used in posterior operations. Caching is used for increasing the performance of read and write operations, due to the locality of data and the possibility of delayed write. The

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
:content "exists(d,G_x)")

Fig. 3. Response performative from a cache agent to a distributor agent, once data are obtained

definition of a cache agent, according to the previous definition of a generic agent is the following:

Definition 2 *A cache agent of a storage group G_x is defined as the following tuple:*

$$< \texttt{C\_Ag\_Id}, \texttt{G}_x, \texttt{Cache}, \texttt{C\_Int\_Net} >$$

where C_Int_Net *represents the interaction network of the cache agent with other agents linked to it, as a vector of relation between cache agent and the rest of agents that belong to the same storage group. According to Figure 6, cache agents are related with other cache agents, with extractor and hint agents.*

The relation between a cache agent and an extractor agent is derived directly from the performative of Figure 2. Therefore, a cache agent is associated unless to an extractor agent. Furthermore, there may be a relation between different cache agents, with the aim of providing a caching service to a single extractor agent. Cache agents are related to hint agents, as we can see in the performative of Figure 5(a).

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
:content "`exists(d,`G_x`)`")

Fig. 4. Performative of invalidation of data in the cache

4 MAPFS Agents 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.

The implementation of the multiagent subsystem is based on MPI technology. MPI provides a framework for deploying agents and their main features. MPI is able to create dynamically independent and autonomous processes with communication capacities. Additionally, agents can 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.

KQML defines an abstraction for transport for agent communication and can be implemented with different solutions. MPI is a good choice, since this technology fulfill the requirements of KQML performatives. The translation of the most relevant KQML performatives into MPI messages is shown in Table 1.

Step 3.2

$u \in HA$

(achieve

:sender z

:receiver u

:reply-with id_ca"

:language Prolog

:ontology MAPFS

: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

:content "`exists(h,`G_x`)`")

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

Fig. 5. Performatives related to a hint agent

Table 1. Translation of KQML performatives into MPI messages

KQML Performative	MPI Messages
`ask-one`	`MPI_Send` `MPI_Recv` a single basic element
`ask-all`	`MPI_Send` `MPI_Recv` a vector of elements
`tell`	`MPI_Send`
`untell`	`MPI_Send`
`deny`	`MPI_Send`
`insert`	`MPI_Send`
`uninsert`	`MPI_Send`
`achieve`	`MPI_Send`
`unachieve`	`MPI_Send`
`error`	`MPI_Send`
`sorry`	`MPI_Send`
`forward`	`MPI_Send` Interpretation of the included performative

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

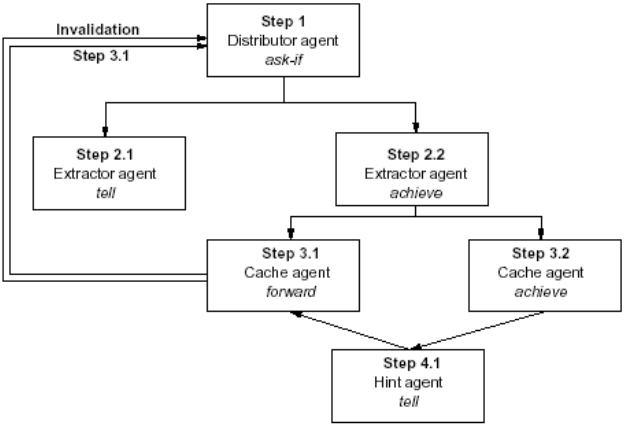


Fig. 6. Control flow of system performatives

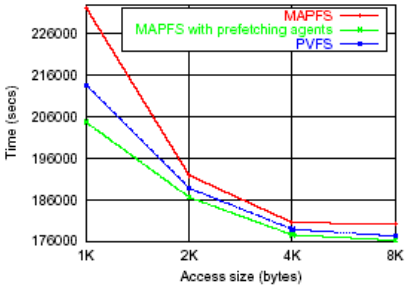


Fig. 7. Comparison of a scientific application in PVFS and MAPFS

A MAPFS multiagent subsystem responsible for prefetching has been implemented and evaluated. If we compare PVFS (Parallel Virtual File System) [5] and MAPFS (Figure 7), we conclude that the usage of agents is a flexible way of increasing the performance, improving the efficiency of PVFS by means of a multiagent subsystem oriented to prefetch probably used data in next executions.

5 Conclusions and Future Work

MAPFS is a multiagent parallel file system, whose design is based on agent theory. Several multiagent subsystems are implemented in MAPFS for tackling different aspects related to its performance. This paper shows the evaluation of a multiagent subsystem used for prefetching probably used data in next executions, concluding that MAPFS system can improve its efficiency in a flexible way by means of the usage of these mul-

tiagent subsystems. This paper also describes how to translate conceptual abstractions based on agents in implementations of close technologies to system programming. In the case of MAPFS, a implementation based on MPI has been used.

As future work, we are developing new multiagent subsystems for improving other aspects of the MAPFS system. Additionally, we have implemented MAPFS-Grid as a version of MAPFS for grid environments. We need to adapt these multiagent subsystems to this new version. Nevertheless, in this kind of system we have to face with different problems, related to the heterogeneity and geographical distribution of grids.

References

1. American National Standard. Knowledge Interchange Format. *Draft Proposed American National Standard (dpANS), NCITS.T2/98-004*, 1998.
2. N. M. Avouris and L. Gasser. *Distributed Artificial Intelligence: Theory and Praxis. Volume 5 of Computer and Information Science*. Kluwer Academic Publisher, Boston, MA, 1992.
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. 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.
6. N. R. Jennings et al. ADEPT: Managing business processes using intelligent agents. In *Proceedings of the BCS Expert Systems 96 Conference, Cambridge, UK*, pages 5-23, 1996.
7. Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. *"Software Agents", MIT Press. Cambridge*, 1997.
8. 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.
9. Olivier Gutknecht and Jacques Ferber. The MADKIT agent platform architecture. *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, March 2001.
10. Staffan Hägg. Agent technology in industrial applications. In *Proceedings of the Australia-Pacific Forum on Intelligent Processing and Manufacturing of Materials (IPMM'97)*, 1997.
11. 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.
12. Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, Baltimore, MD 21250, 1997.
13. E. Pitoura. Transaction-Based Coordination of Software Agents. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA)*, 1998.
14. María S. Pérez, Félix García, and Jesús Carretero. A new multiagent based architecture for high performance I/O in clusters. In *Proceedings of ICCP'01*, September 2001.
15. María S. Pérez, Félix García, and Jes´us Carretero. MAPFS MAS: A model of interaction among information retrieval agents. In *2nd IEEE/ACM CCGrid 2002*, May 2002.
16. 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.
17. Aamir et al. Shafi. DIAMOnDS - DIstributed Agents for MObile and Dynamic Services. In *Proceedings of the CHEP03*, March 24-28 2003.