Pre-print version
W. Fornaciari, V. Piuri, A. Prestileo, V. Zaccaria, "An agent-based approach to full interoperability and allocation transparency in distributed file systems", in Proc. of the 2001 ACM-IEEE International Workshop on Mobile Agents for Telecommunication Applications (MATA 2001), Montreal, QC, Canada, pp. 153-162, August, 2001

An Agent-based Approach to Full Interoperability and Allocation Transparency in Distributed File Systems

William Fornaciari[§] Vincenzo Piuri[‡§] Andrea Prestileo[†] Vittorio Zaccaria[§]

[§]Politecnico di Milano
Dip. di Elettronica e Informazione
Milano, ITALY 20133
{fornacia,piuri,zaccaria}@elet.polimi.it

[‡]University of Milan
Department of Information Technologies
Crema, ITALY 26013
piuri@dti.unimi.it

[†] Cluster Reply Milano
Milano, ITALY 20139
a.prestileo@reply.it

Abstract. Modern distributed file system realizations offer only partially resource location transparency, resource location independence, fault tolerance, load balancing, heterogeneity, self-configuration, and simplified user access. Traditional portability techniques developed in these systems become unsuited in highly dynamic environments.
To solve these problems within a homogeneous framework we studied and experimented the use of static and mobile agents in a portable environment. In this paper we describe the philosophy, the structure, and the prototype realization of the Agent-based Distributed File System (ADFS). The main properties of this innovative distributed file system are resource location transparency, resource location independence, self-configuration, and heterogeneity of the underlying hardware and operating system architectures.

1 Introduction

The main goal of a distributed operating system is to provide a uniform resource view in a collection of interacting, loosely-coupled computers [1]. The distributed system user must always be able to perceive the same view of the system as well as the same logical and physical resources, independently from his network access point. This is useful, for example, within a Virtual Private Network where users

may not be aware of the physical position of the resources. In this perspective, a Distributed File System (DFS) plays a fundamental role by creating a unique logical view of the file system resources. The result is a virtual composition named the Distributed Directory Tree (DDT).

A DFS must present various transparency properties to applications. The most important properties are the *resource location transparency* and the *resource location independence* [1]. The resource location transparency guarantees that the distributed pathname of a resource does not offer any information about its physical position. The resource location independence ensures the immutability of the resource name even if the physical position is changed.

Traditional mechanisms realizing transparency are usually based on static resource-position binding information, created when a new component unit of the file system is added. For example, to mount the remote file system onto a client directory in NFS [2, 3], the network administrator must create a suitable entry in the mounting configuration file that is used to setup the system tables during bootstrap. If the exported file system location is modified in the network or in the local file-system, information have to be updated manually in every computer of the network to achieve correct distributed system operation.

In the Andrew File System (AFS) [4], the resource-position bindings are partially stored in client's directories by means of a mapping between the file-names and the identifiers of the atomic portions of the distributed file-system (called *volumes*). In this case server availability changes can cause incoherence in server-side and client-side mappings that must be manually fixed by the network administrator. The Coda file system [5] improves AFS functionality by adding replication, fault tolerance, and disconnected operation features. Although it represents a substantial step towards solving the server availability problem for opened files, the fully automatic reconfiguration is not yet achieved. This property is useful, for example, when new computers are added to the system.

In the Locus file system [6, 7], a path-traversal mechanism and a globally-replicated mounting table are used by each client to map a resource pathname onto the managing site. Although this globally-replicated mounting table hinders scaling to large and dynamic networks, location and replication transparency goals are significantly reached.

In Sprite [8] distributed resources are accessed via prefix tables [9] stored by clients. The prefix table is in fact only an hint table [10]. When performing a lookup, if the selected hint is not correct the client issues a broadcast query to know which of the servers actually contains the desired resource. The hint table is updated with the results of the query. Even if the Sprite's adaptation mechanism is a significant step towards dynamic reconfigurability, the large use of broadcast messaging makes this DFS unsuitable for large-scale environments.

The xFS distributed file system [11] realizes the theoretical resource location transparency and independence. These characteristics are achieved by implementing the mapping from a resource name to the storage servers through indirections. Although the design of this distributed file system is very com-

plex, it represents one of the first complete solutions to the problem of location transparency and independence.

In Microsoft Windows NT's Dfs [12], clients contain a reference to a DFS root server that hosts the upper portion of the DDT. The root server contains a partition knowledge table (PKT) mapping the logical DFS namespace onto a set of servers that physically contain the resources. This client dependence from the root host and the junction points is based on the use of explicit server names to create DDT, leading to low location transparency and independence as well as to reduced operating system heterogeneity. Nowadays networks are becoming highly dynamic environments, presenting challenging problems such as disconnected or weakly connected operations. In these environments the use of static traditional distributed mechanisms is often unsuited.

In our research we experimented the mobile agent technology [13–15] to replace static binding with a straightforward dynamic introspection. Results of our studies and experiments were design and prototype implementation of Agent-based Distributed File System (ADFS). In this system we exploit agent's ability to explore the network with a minimal set of information to dynamically create mappings between resource-name and position. Besides, ADFS can actively and autonomously assimilate new clients and servers with minimal administration effort when computers are replaced, removed or added. Efforts are confined to the new computers, while in traditional DFS a significant and wide configuration effort is usually required. The global auto-configuration ability with only localized initial configuration can also be exploited as a mechanism to realize hot plug-in (or hot-swap) servers for fault tolerance. On the other hand, since we adopted an highly portable and interpreted mobile code written in Java [16] to realize ADFS, we achieved heterogeneity, interoperability, and portability in a very straightforward way by overlapping the DFS to the local file system.

Due to space limits, this paper focus on configuration and lookup operations only, although a complete prototype of the system has been implemented. This paper is organized as follows. Section 2 describes the basic requirements of our system. Section 3 analyzes the system architecture, while Section 4 presents the implementation and some experimental results. Section 5 concludes the paper envisioning current ADFS research directions.

2 Basic system issues

In ADFS the Distributed Directory Tree (or DDT) is a logical name space, i.e., a set of distributed pathnames, whose structure is virtually overlapped on the physical locations of the distributed system resources (fig. 1).

Pathnames contained in the DDT are directly connected to the resources that they represent. However, to provide resource location transparency, pathnames do not include any information concerning this mapping.

To show how mapping is actually performed, let us introduce some basic concepts. A Distributed Partial Sub-Tree (DPST) is a portion of the DDT composed at least by a root directory. Two DPST are said not overlapping if one

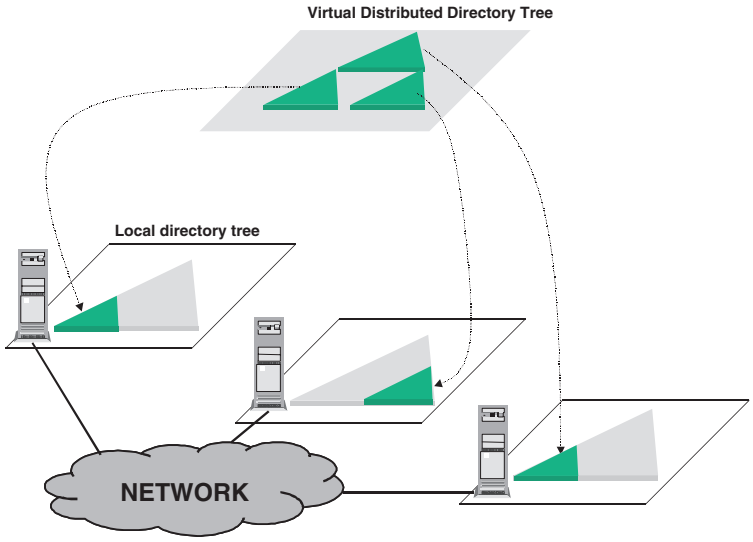

Fig. 1. A virtual DDT mapped on physical locations

does not contain any node or leaf of the other. In ADFS, the DDT is decomposed in a set of not overlapping DPST and each DPST is implemented by means of a Sub-Tree (or ST) resident in a specific computer.

Fig. 2 depicts a typical view of a DDT. As shown in this figure, an empty root is allowed for being not implemented. Besides, local trees can have hidden (i.e., not shared) resources. When a computer implementing a DPST is turned on, the virtual pathnames that it defines are automatically active, even if parent DPSTs are not active. This is due to the fact that each computer knows the relative position of its own DPST in the DDT. The basic idea underlying our approach consists of performing the file lookup operation in the distributed environment by using a mobile agent (*lookup agent*) automatically created by the client application. Such an agent navigates among networked computers and inspects all the DPSTs eventually contained. When the agent finds the desired resource on a computer, it notifies the client application that asked for the lookup of the resource network address.

Since lookup is not based on a-priori information contained in the pathname, the ADFS architecture offers implicitly resource location transparency. Also resource location independence is guaranteed since moving the DPST implementation between two computers does not imply any change in its pathnames. ADFS has been designed, realized and tested on a hierarchical network model based on a structure containing nodes and sub-networks. Two sub-networks are connected by at least a low-bandwidth link between nodes. Links between sub-

networks may be either physical or logical. In the first case, the link connects physically one node in each sub-network. In the second case the link consists of complex network path through which the sub-networks can exchange information. In both cases the connected sub-networks are called *adjacent*.

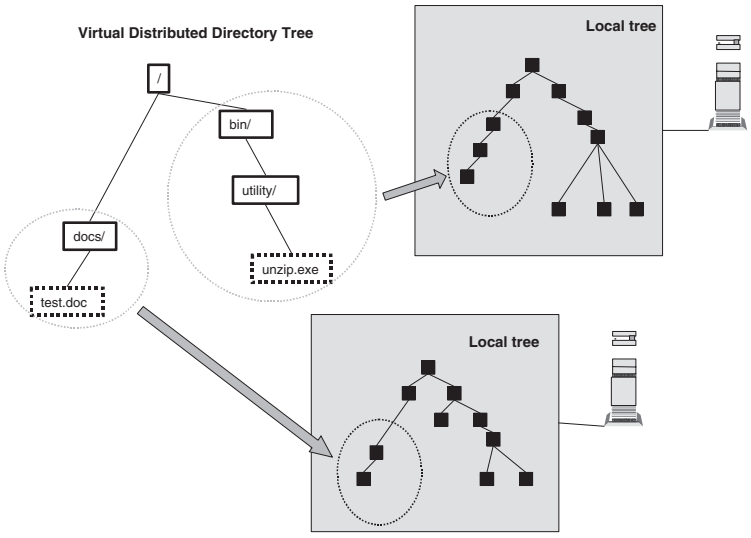

Fig. 2. A DDT implemented by two computers

An agent located in a sub-network determines the next sub-network to reach only on the basis of its knowledge and the adjacent sub-networks. The agent scope is thus dynamic because it varies with the sub-network-relative position of the agent. While logical proximity information is critical to determine the inter-sub-network route, the mobile agent needs more specific information to build its intra-sub-network route. This information is based on the activity status of the nodes in the given sub-network and is updated by ADFS self-configuration system.

3 The system architecture and operation

The distributed system architecture supporting ADFS is composed by an heterogeneous set of computers. Each computer can behave as client, server, or both. ADFS transparently realizes cooperation and interoperability in an innovative way by means of mobile agents.

Each computer of the distributed system contains a computational environment (called *location*), as shown in fig. 3. The location is composed by a set of system processes and by a set of system modules that provide basic DFS functionalities:

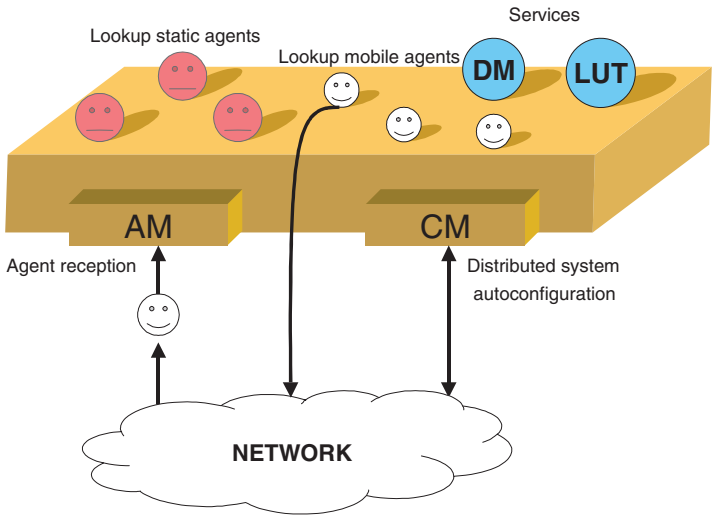

Fig. 3. A typical location

- Lookup Static Agents: processes that manage the lookup requests for a particular application.
- Lookup Mobile Agents: processes that embody the migrating lookup requests made by a particular application.
- Directory Manager Service (DM): used by mobile agents to inspect local resources.
- Look up Table (LUT) Service: used by mobile agents to obtain information about active nodes in the network.
- Agent Manager (AM) : used by mobile agents to be accepted in the location.
- Configuration Manager (CM): process that manages the auto-configuration of LUTs.

Let us describe how the system works starting from the lookup system call performed by the application.

The interface of the location towards the application processes (the location API) consists of a set of inter-process calls. To perform a lookup operation, an application calls the lookup procedure. This procedure activates a local lookup static agent associated to that application. This static agent is directed to enhance the performance of the lookup operation. In fact, the static agent is assigned to all process instances of a specific application and furnished of a simple cache of the previous lookup results. In this way, all the users working with that application obtain reduced response time with respect to a common operating system cache.

To realize the lookup operation the static agent creates a lookup mobile agent that inspects all locations of the distributed system until it finds a computer

holding the given desired resource. Actual lookup mobile agent's route is hierarchically built on top of the logical view of the network. The mobile agent visits exhaustively all nodes of the sub-network in which is created. Then, the agent moves to every adjacent sub-network and repeats the exhaustive exploration of the nodes within each of them.

Mobile agents perform the navigation by using information about the active nodes in the local and the nearby sub-networks. A node is active if it is connected to the network and contains a running location. This information, (i.e., the locally reachable nodes) is contained in the navigation Look Up Table (LUT) of the current location. The LUT is simply constituted by a mapping from a sub-network prefix to the relative set of active nodes. The domain of this mapping, (i.e., the set of local and adjacent sub-network prefixes) is fundamentally static and can be locally configured when the computer is added to distributed system. Conversely, information about active nodes is dynamic and updated by the Configuration Manager.

In each location visited by the mobile agent, the local file system resources are scanned by looking into the Directory Manager (DM) of the location itself. The DM provides an abstraction layer that transforms the local exported sub-trees into their respective distributed partial sub-trees. This is done by maintaining, for each DPST stored in the location, the pair composed by the DPST root and the local ST root of the exported ST. Interaction between the mobile agents and DM is very straightforward: by means of inter-process communication, the mobile agent asks the DM about the local existence of a given distributed resource. The DM checks if at least one of the locally contained DPST roots is a prefix of the given pathname. In the negative case, the mobile agent does not find the desired resource locally. Otherwise, the pathname is transformed into the corresponding local one by substituting the root of the local ST to the matching prefix. A local file-system lookup is then executed and the result is returned to the mobile agent.

When the desired resource is found or the whole distributed system has been unsuccessfully visited, the mobile agent sends a message with the search result back to its parent lookup static agent. The static agent provides this information to the application process that asked for.

With a certain frequency, the mobile agent sends a Check Point Message (CPM) to the static agent. CPMs are a form of asynchronous information transfer (from the mobile agent to the static agent) about the state of the mobile agent. The static agent is allowed to generate several agents for a single lookup when it does not receive CPMs from a given mobile agent within a predefined maximum time. This is done by using the information of the last received CPM and the search is restarted from the last point where the dead mobile agent gave his last vital sign. The robustness of this approach is proportional to the granularity of check pointing but a very fine granularity could compromise the time efficiency of the entire system.

Correct location management implies security and networking issues. To such purposes two additional entities are available in each location: the Agent Man-

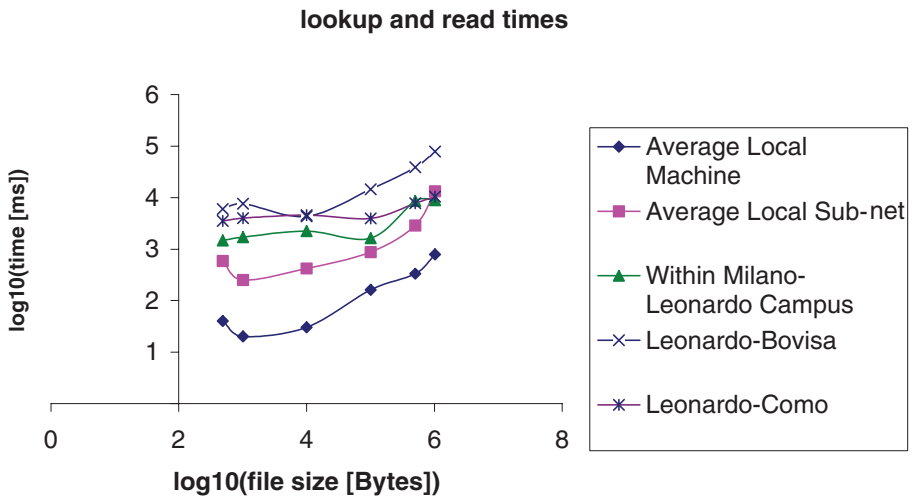

Fig. 4. Read times for various file sizes

ager (AM) and the Configuration Manager (CM). The AM is the agent activator through which mobile agents can request to be accepted and activated in the location. The AM receives the mobile agent's state and code and verifies the access permissions. If the mobile agent passes the verification, the AM activates it by starting the corresponding thread.

The Configuration Manager (CM) is the process through which the LUT auto-configuration is realized. The CM listens to the network channel for configuration messages sent by other locations and consequently modifies the LUT. Besides when a new computer is activated, its CM broadcasts an activation message to all the active computers. The CMs of these nodes receive the activation message, update their local LUTs, and answer with their activity state. At the end of this automatic configuration process all the active nodes have been configured to reflect the actual state of the network.

4 Implementation and Experimental Results

ADFS was implemented in Java (JDK 1.1.5), on IBM compatible computers running Windows NT. It was also ported on PCs running Windows 95 and Linux. The prototype system was extensively tested in a real geographical network composed by 12 PCs. Computers were distributed on four LANs at Politecnico di Milano namely two in the Milano-Leonardo Campus, one in the Bovisa Campus and one in the Como Campus. Bovisa-Leonardo Campuses were connected by a link at 128Kbit/sec while Leonardo-Como link was of 2Mbit/sec.

Fig. 4 shows the average times of a read operation that include the time to perform a lookup within the network. As can be seen, read operation times of little size files (<10KB) are conditioned mostly by the lookup time, which

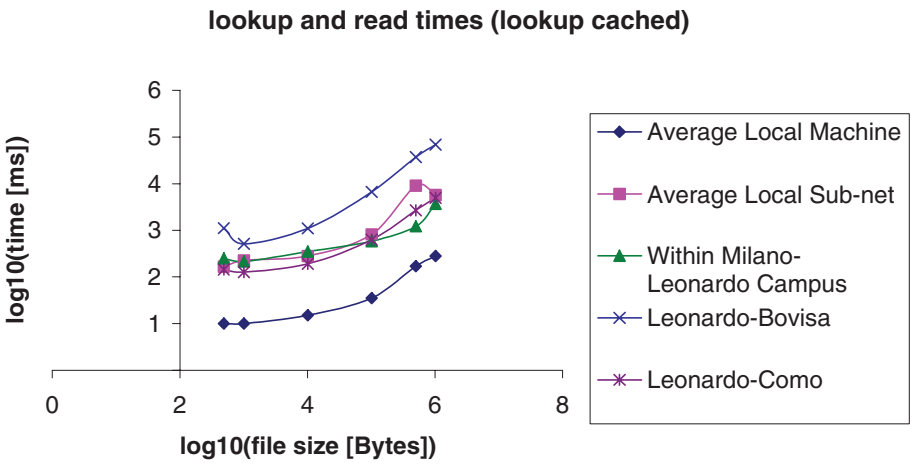

Fig. 5. lookup and read times (lookup results cached by lookup static agents)

depends on the path that the lookup agent follows. For files of greater size, the read time is comparable with a read operation performed on a NFS file system since the static agent use a similar direct access mechanism for reading file blocks. For files of growing size the times become linearly dependent on the file size.

Fig. 5 shows the average times of a read operation when the caching of the lookup results is enabled, i.e., the lookup static agent does not always generate a new lookup agent but tries to use the results of the previous lookup operations. In the case that a lookup history for a specific file does not exists or it is not correct a lookup mobile agent is created. This feature exploits the file and directory access locality proper of a typical user behavior and, comparing Fig. 5 with Fig. 4, it reduces greatly the read time of files of little size.

5 Conclusions

In this paper we presented ADFS, an innovative prototype of a distributed file system based on mobile agents. The underlying ideas and the fundamental characteristics were discussed. Resource location transparency and independence through the distributed system architecture have been achieved in a very straightforward way by means of a dynamic introspection based on mobile agents. System adaptivity was obtained easily by self-configuration for very dynamic environments. Portability and heterogeneous interoperability were also provided implicitly by the use of the Java language. Our research is now focused on the issues concerning performance and fault tolerance.

References

1. G. Coloris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, Reading, MA., 1994.
2. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network file system. In *USENIX 1985 Summer Conference proceedings*, pages 119–130, 1985.
3. Sun Microsystems, Inc. NFS: Network file system protocol specification. *Internet Request for Comments*, (1094), 1989.
4. J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(4):51–81, 1988.
5. M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.
6. Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The LOCUS distributed operating systems. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles (SOSP)*, volume 17, pages 49–70, 1983.
7. G. Popek and B. Walker. The locus distributed system, 1985.
8. J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The sprite network operating system. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 21(2), 1988.
9. B. Welch and J. Ousterhout. Prefix tables: A simple mechanism for locating files in a distributed system. In *Proceedings of the 6th International Conference on Distributed Computing Systems (ICDCS)*, pages 184–189, Washington, DC, 1986. IEEE Computer Society.
10. E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–374, 1990.
11. T. Anderson, M. Dahlin, J. Neefe, D. Patterson, and R. Wang. Serverless network file systems performance. *ACM Transactions on Computer Systems*, 14(1):41–79, 1996.
12. Microsoft. Distributed file system: A logical view of physical storage, 1999.
13. K. Rothermel and F. Hohl. *Mobile Agents (MA '98)*. Number 1477. Lecture Notes on Computer Science, Springer, Berlin; Heidelberg, 1998.
14. C. Harrison, D. Chess, and A. Kershenbaum. Mobile agents: Are they a good idea? Technical report, IBM Research Division, T.J.Watson Research Center, http://www.research.ibm.com/massdist/mobag.ps, 1995.
15. Davis Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, 2(5):34–49, 1995.
16. Ken Arnold and James Gosling. *The Java Programming Language*. Addison-Wesley, Reading, MA, 1998.