On Building a File System for Mobile Environments Using Generic Services

F. André
IRISA Research Institute
Campus de Beaulieu
35042 Rennes Cedex, FRANCE

M.T. Segarra
IRISA Research Institute
Campus de Beaulieu
35042 Rennes Cedex, FRANCE

Abstract

The highly dynamic nature of mobile environments (portable computers using wireless networks) makes them dramatically different from desktop contexts. Applications should adapt their behavior to current environment conditions to provide good performance to their users. The MolèNE project aims at providing a systematic approach to build such applications. It offers developers a set of generic services that can be customized to each application and adapted to environment conditions depending on applicaitons requirements. Using these services, programmers concentrate on application functionalities and makes the development and maintenance of code simpler. Several experiences are being performed to validate the generic nature of the services. In this paper, we describe one of these experiences, the design and implementation of MFS, a NFS version 2.0 protocol compatible file system that deals with disconnections of mobile clients. MFS allows users to access NFS data regardless of network condition, eliminates the need for mobile NFS clients to manage data consistency, thus reducing battery consumption due to network utilization, and allows a per data block consistency management.

1 Introduction

Recent technology advances in wireless networks have provided portable computers with wireless links to access available resources and services in a distributed environment. Although this trend allows users to work connected to the network while *on the move* - what we term mobile computing in our context -, it also introduces many new technical challenges [4]. The inherent changing characteristics of wireless networks and portable computers render mobile environments much more dynamic than fixed ones. Building a mobile application requires skilled programmers as it needs to adapt to current execution conditions to achieve good performance.

The MolèNE system provides programmers with a set of generic services that can be customized to different applications and which behavior adapts to execution conditions depending on application requirements. These services include data caching, consistency, and functionality relocation. Developers of mobile applications may use MolèNE services to manage adaptation to the environment so that they can concentrate on the implementation of application functionalities. In order to identify and validate the generic nature of MolèNE services, two experiences are being performed. The first one concerns transactional applications on the Internet [1]. The second one concerns file system applications and is the subject of this paper. Traditional distributed file systems are not suitable for mobile applications as they make the assumption that clients[1] and servers communicate using a fast and reliable link. However, wireless networks offer poor performance compared to wired ones and, even worse, while disconnected, clients cannot access data. This paper presents the design and implementation of MFS (MolèNE File System), a NFS version 2.0 protocol compatible file system that allows mobile users to work on NFS objects despite periods of disconnection and also reduces the network traffic issued by mobile NFS clients. A process referred as proxy is executed on the fixed network so that the mobile computer is notified about modifications on NFS objects by other users. On the other hand, requests issued by NFS clients are executed locally as far as the concerned data are available on the mobile device. MolèNE services are responsible for managing requests to be sent through the network. While the mobile computer is connected, theses requests are sent to the proxy. Oth-

[1] We refer as client the piece of software that communicates with a server to offer a service. We call user, the physical human that uses the mobile computer to carry out some work.

erwise, and depending on their type, they are stored in a log until the network becomes available again. A prototype has been implemented that supports user operation during disconnections, and a per data block conflict detection and resolution so that all the updated data are preserved. Moreover, network traffic issued by mobile clients is reduced thus reducing battery consumption.

The remainder of the paper is organized as follows: next section overviews the NFS protocol and introduces the reader to the design of MFS. Section 3 and 4 describe in detail components of the mobile client and the proxy respectively. Section 5 compares our system with other work in the mobile computing and distributed file systems communities. Finally, conclusion and future work are given in section 6.

2 NFS Overview and MFS Design

From NFS main features and disadvantages when used in a mobile environment, we define the main goals that guided MFS design.

2.1 NFS Overview

NFS is a distributed file system that allows users to access files stored in remote servers transparently. Clients access files contents by blocks of a fixed well-known size and cache the accessed blocks to gain from data locality thus reducing network traffic. Due to the stateless nature of NFS servers, cached data consistency is ensured by clients. When accessed, a NFS object[2] is cached by the client and a validity timeout is associated with it letting the client directly access the cached object. Upon timeout expiration, its access is preceded by a request to the server asking for the attributes of the object[3]. If the copy on the server is more recent than that on the mobile device, the cached blocks are invalidated. These verification requests result in unnecessary network traffic if, as is the case in most UNIX environments, data sharing is infrequent [6].

As sending data through the network is a battery consuming operation in mobile environments, NFS requests issued by clients should be minimized. On the other hand, disconnected clients are not allowed to access NFS objects as consistency cannot be verified. Finally, as permanent storage is not used to cache NFS objects, powering off and on the mobile computer requires the cache to be entirely reloaded.

2.2 MFS Design Overview

There are three goals that guided the design of MFS:

- Mobile clients should be able to use standard NFS servers in order to avoid changes on existing infrastructures.
- Data verification traffic may imply lots of requests issued by mobile clients in order to validate cached data. It consumes the precious resource of battery power and, thus it needs to be minimized.
- Due to the highly variable nature of wireless links, data block size used to access NFS objects should be adapted to avoid applications performance degradation and to better utilize the memory of mobile computers: small block sizes should be used on poor quality links while large ones when the quality improves.

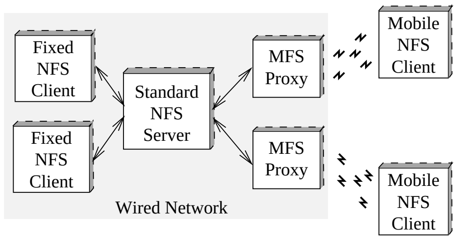

Figure 1: Interaction among NFS servers, MFS proxies, and NFS clients

In order to achieve these goals, a proxy is executed on the wired network, acting as an interface between the mobile computer and the standard NFS server. This is shown in Figure 1. In this figure, Fixed NFS Client modules refer to standard NFS clients that are executed in a wired-connected station and Mobile NFS Client modules are their counterpart on mobile computers with added functionalities to implement the previously mentioned goals.

The standard NFS server is used by Fixed NFS Client modules and proxies (MFS Proxy modules on Figure 1). These proxies work as standard clients for NFS servers as well as servers for mobile computers. NFS traffic between a Mobile NFS Client module and the NFS server is managed by the MFS Proxy, and thus it is able to trace mobile computer cache contents. Keeping this information allows the proxy to

---

[2]We name NFS object, either a file or a directory.

[3]Object attributes refers to the meta information about the object and this includes the last modification time (mtime) on the server.

perform validity checks of the cached data by sending periodic NFS requests to the server. The frequency of verifications corresponds to the validity timeout for NFS objects so that the same consistency guarantees as the standard NFS protocol are ensured. Verifications may be reduced or even avoided when the mobile computer is disconnected.

If cached data are no longer valid, the proxy sends an invalidation request to the mobile computer. Therefore, Mobile NFS Client modules are no longer responsible for detecting conflicts with data on the NFS server. Network traffic issued by mobile clients is thus reduced if, as is the case in most UNIX environments, data sharing is infrequent.

Finally, the third goal of the MFS design is achieved by allowing data transfers between our Mobile Client module and the proxy being performed in a Standard NFS Client module-independent block size. Therefore, this size may be adapted to the current characteristics of the utilized network.

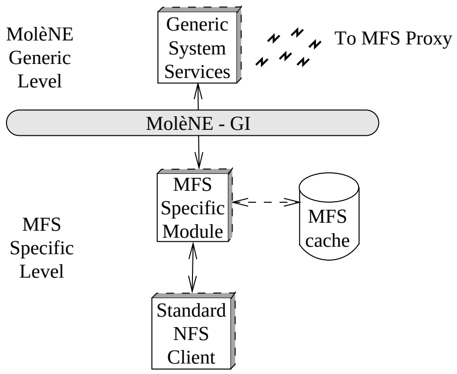

Figure 2: Mobile NFS Client Components

3 Mobile NFS Client Module

The Mobile NFS Client module in Figure 1 runs in the mobile computer and enables mobile users to access NFS objects. It is composed of a Standard NFS Client module, a MFS Specific module, and MolèNE services as depicted in Figure 2. Modules are logically subdivided into two levels and those in different levels communicate through the MolèNE-GI (MolèNE Generic Interface). The MolèNE Generic Level contains modules offering MolèNE services that may be used by several applications to cope with mobile computing related problems. Currently these services include a Proxy Manager Service, and a Mobile Communications Service (MCS). The former manages location of proxies executed on the fixed network on behalf of mobile computers. The latter takes into account network conditions in order to send requests issued by clients.

3.1 Components Description

The MFS Specific Level contains modules that know the semantic of the NFS distributed file system and uses MolèNE services to deal with disconnections from the network.

MolèNE-GI is a set of library functions that may be used by applications to exploit MolèNE services. Existing functions in MolèNE-GI include the configuration of services required by applications, and sending and receiving information coming from the services.

At the beginning of its execution, the MFS Specific module configures the MCS to use its log facility: network traffic issued while disconnected is stored in the log and sent through the network later. Requests submitted to the MCS are encapsulated in a generic message indicating, among other information, if the request is logable or not. Non logable requests report an error to the Standard NFS Client module if the network is not available. Logable requests, on the other hand, are stored in the log while disconnection and a well-performed operation response is issued to Standard NFS Client module, thus letting the user continue his work despite disconnections.

The MFS Specific module plays two roles for the Standard NFS Client in the mobile computer. The first one is that of a cache manager. In order to allow a mobile client to access NFS objects while being disconnected and to reduce network traffic while connected, the MFS Specific module caches information into the local disk as it arrives from the network. This information concerns not only NFS objects contents but also their attributes. The second role is that of a "NFS server". The Standard NFS Client module issues NFS requests in response to user operations on NFS objects. These requests are captured by the MFS Specific module that serves them if required information is locally available or submits them to the MolèNE-GI in order to obtain it, thus, acting as a NFS server for the Standard NFS Client module.

3.2 Interaction Among Components

In MFS, the send facility of the MolèNE-GI is used by the MFS Specific Module to send two kinds of requests to the MCS:

- Requests generated by clients because of mobile user operations on NFS objects.

- Requests generated by the MFS Specific Module itself when some information has been deleted from cache due to cache replacement algorithm.

3.2.1 Standard NFS Client Module Requests

When the mobile user issues an operation on a NFS object, it is received by the Standard NFS Client module which in turn creates a NFS request intended to be sent to the corresponding server. The request is captured by the MFS Specific module that performs different actions depending on the type of the request. Requests implying the search of information from the server (read requests) are not submitted to the network if required information is locally available. A NFS response is built and directly sent to the Standard NFS Client module. Otherwise, they are labeled as non logable requests and submitted to the proxy through the MCS. If network is not available, the MFS Specific module reports an error to the Standard NFS Client module as mentioned in the previous paragraph.

Requests that are intended to modify information on the NFS server (write requests) are submitted to the proxy through the MCS. They are tagged as logable requests. These operations are executed locally for available data and integrated into the server upon reconnection.

3.2.2 MFS Specific Module Requests

The second kind of requests are those generated by the MFS Specific module itself while managing cache contents, and, concretely, when the cache replacement algorithm is executed. In this case, cache contents have changed and the MFS Proxy should be advised to avoid unnecessary information checks.

Before submitting these requests to the MCS, they are stamped as logable requests. If the mobile computer is disconnected, the MFS Proxy does not verify validity of the data cached on the mobile computer and thus, it is unimportant that its cache contents is not harmonized with that of the MFS cache. However, when the mobile computer reconnects, logable requests are sent to the MFS Proxy for reintegration which will involve harmonization of caches.

4 MFS Proxy

The MFS Proxy runs in a fixed station and acts as a stateful server for the mobile computer while being able to operate with standard NFS servers. In order to achieve this, it stores on disk information (object data and attributes) that is currently present in the MFS cache and verifies its validity as previously mentioned.

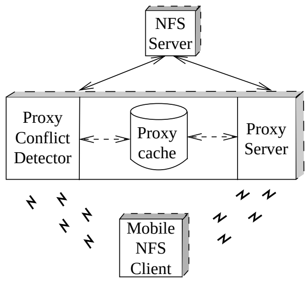

Figure 3: MFS Proxy Components

It is logically divided into two modules as depicted in Figure 3. The Proxy Server is responsible for serving requests coming from the mobile computer and managing the Proxy cache contents depending on known contents of the MFS cache.

The Proxy Conflict Detector interacts with the NFS server to validate data currently stored on the Proxy cache by performing periodic checks. A conflict between data cached on the mobile computer and the server arises when a third part modifies the copy on the server. Data concerning the modified object become stale on the mobile computer and it is the role of the MFS Proxy to detect and report it to the mobile computer. In order to achieve this, the Proxy Server caches not only data accessed by the mobile computer but also their attributes so that they always correspond to the attributes the object had on the server the last time the mobile computer asked for an operation on it.

The Proxy Conflict Detector works on the cached object attributes to detect conflicts. Figure 4 shows how conflicts are detected for a NFS object `O1`. At `t0`, the mobile computer performs an operation on the object and the Proxy Server caches attributes on the Proxy cache. At the next verification period (`t1`), `O1` attributes are requested by the Proxy Conflict Detector. Nobody has modified the copy on the server, thus no conflict is detected. Between `t1` and `t3`, a third part modified `O1` resulting on an increased mtime for it on the server. When at `t3` the Proxy Conflict Detector checks for `O1` attributes, it will realize that they are older than those on the server and will assume this as a conflict.

Operations issued by a standard NFS client may involve either object attributes only or object data-and-attributes. Therefore, there are two kinds of conflicts:

- Attributes only conflicts: object attributes differ between the mobile computer cache and the

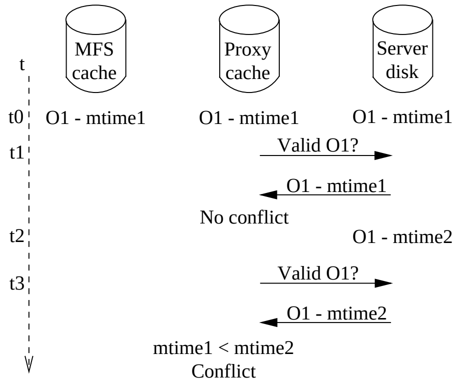

Figure 4: Conflict Detection Mechanism

server but object contents remain unchanged. It may appear when access rights of an object are modified on the server by another client.

- Data-and-attributes conflicts: attributes and object contents on the mobile computer are different from those on the server. This appears when there is a modification of an object contents on the server by another client (it also implies automatic modification of its attributes).

Data-and-attributes conflicts are reported as blocks invalidation requests. NFS objects are accessed by standard NFS clients in a block fashion, i.e. they ask for the access to an amount of blocks of a well-known size. Therefore, mobile NFS clients need not to store all the contents of an object in their cache but only a part of it. A data-and-attributes conflict only appears if a third part modifies on the server one of the blocks that are currently stored in the MFS cache. As the Proxy cache contains blocks stored in the mobile computer, it is easy for the Proxy Conflict Detector to discover if the detected conflict should be reported to the mobile computer or not. Continuing with the previously example of object `O1`, Figure 5 shows the case in which a conflict does not cause invalidations on the mobile computer. Let us assume that the mobile computer has blocks 1 and 2 stored in the MFS cache and that `mtime1` changed to `mtime2` when block 0 of `O1` was modified. Once the Proxy Conflict Detector discovers that the `mtime` attribute has changed on the server, it demands contents of blocks 1 and 2. If there is no difference between them and the copy on the cache (as is the case in the example), no invalidation should be sent to the mobile computer. If the modified block were block 1 or 2 or both, an invalidation message concerning the block(s) would be reported to the mobile computer. Other blocks of the object cached on the mobile computer remain available to the user, thus reducing transfers through the wireless link.

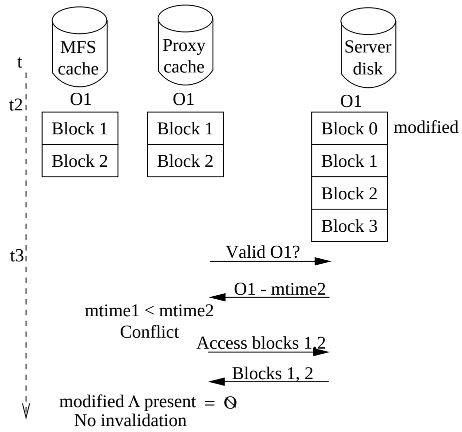

Figure 5: Invalidation Mechanism

5 Related Work

A lot of attention is being recently paid on the mobile computing paradigm in general, and on mobile distributed file systems in particular. Some mobile approaches aim at providing a particular functionality such as automatic hoarding for file systems [7] or data consistency for databases [2].

Other systems provide developers with basic mechanisms to build adaptive mobile applications by providing adaptation mechanisms such as the Relocatable Dynamic Objects of Rover [5] or work conditions discovery mechanisms [10]. Developers use these mechanisms to implement their own adaptation strategy.

A more advanced solution would allow applications to be aware of their environment and would propose ready-to-use adaptation strategies. This approache is provided by Odyssey [9]. A set of ready-to-use policies to allow applications to access information depending on resource availability are provided to developers. However, Odyssey uses the file system interface and extends it to allow applications to use its services which implies a limited framework to exploit existing policies and to create new ones. MolèNE also follows this approach. However, it is designed so as to allow easy modification of existing services and addition of new ones. Its flexibility and extensibility are important features in a mobile environment.

Work on distributed file systems for mobile environments was pioneered by Coda [6]. Coda modified the

AFS distributed file system to allow mobile clients to access files in spite of being disconnections and deals with the partially connected mode of operation. However, although NFS stays the most frequently used distributed file system, few attention has been paid to mobile users. Moreover, existing approaches ([8], [3]) propose a middleware that acts as a standard NFS client when connected to the network and allows users to access files residing on the portable memory during disconnection. Our MFS file system differs from both systems in three main features: 1) the use of generic services to manage disconnections avoids their explicit consideration into the file management code which simplifies application development, 2) the reduction of traffic issued by mobile clients, and 3) the use of a bandwidth-adapted data block size.

6 Summary

In this paper we have presented the design and implementation of the MFS file system that takes into account the dynamic nature and poor performance of wireless networks in its connected operation and allows users to work while disconnected and all this without modification of existing infrastructures. MFS uses generic services to deal with the periods of disconnection making the file management related code independent of different execution environments.

A first prototype of MFS has been already developed on top of the Linux operating system. When connected, requests from the mobile device are executed by the MFS proxy and invalidations coming from it are also taken into account in the mobile computer cache. When disconnected, the user is able to perform his work depending on the availability of the required data in the mobile computer.

MFS has allowed us to identify several generic services that may be used by mobile applications. Currently these services are customized and accessed by applications using several library functions. Customizing generic functions can easily be done by using the inheritance and polymorphism mechanisms of object oriented languages. Therefore, we are working on developing an object-oriented MolèNE system that profits from these features to offer its services. A mobile transactional application for the internet has been yet developed in an ad-hoc manner. It will be adapted to gain from MolèNE services in order to validate our ideas.

References

[1] F. André and E. Saint Pol, A Middleware for Transactional Internet Applications on Mobile Networks. *In Proc. of the 1998 PDPTA*, Las Vegas Hilton, Las Vegas, USA, 1998.

[2] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch, The Bayou Architecture: Support for Data Sharing among Mobile Users, *Workshop on Mobile Computing Systems and Applications*, Sta. Cruz, CA, USA, 1994.

[3] M.E. Fiuczynski, and D. Grove, A Programming Methodology for Disconnected Operation, http://www.cs.washington.edu/research/ /mobicomp/postscript/grovemef.ps, 1994.

[4] T. Imielinski and H.F. Korth, *Mobile Computing*, Kluwer Academic Publishers, 1996.

[5] A.D. Joseph, J.A. Tauber, and M.F. Kaashoek, Mobile Computing with the Rover Toolkit, *IEEE Transactions on Computers*, Vol. 46, Num. 3, pp. 337-352, March 1997.

[6] J.J. Kistler and M. Satyanarayanan, Disconnected Operation in the Coda File System, *ACM Transactions on Computer Systems*, Vol. 10, Num. 1, pp. 3-25, February 1992.

[7] G.H. Kuenning and G.J. Popek, Automated Hoarding for Mobile Computers, *In Proc. of the 16th SOSP*, St. Malo, France, 1997.

[8] J.C.S. Lui, O.K.Y. So, and T.S. Tam, NFS/M: An Open Platform Mobile File System, *In Proc. of the 18th ICDCS*, Amsterdam, The Netherlands, 1998.

[9] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker, Agile Application-Aware Adaptation for Mobility, *In Proc. of the 16th SOSP*, St. Malo, France, 1997.

[10] G. Welling and B.R. Badrinath, An Architecture for Exporting Environment Awareness to Mobile Computing Applications, *IEEE Transactions on Software Engineering*, Vol. 24, Num. 5, 1998.