

# A Mobility-Aware File System for Partially Connected Operation

Dane Dwyer

Vaduvur Bharghavan

Coordinated Science Laboratory  
University of Illinois at Urbana-Champaign  
(dwyer, bharghav)@crhc.uiuc.edu

## Abstract

*The advent of affordable off-the-shelf wide-area wireless networking solutions for portable computers will result in partial (or intermittent) connectivity becoming the common networking mode for mobile users. This paper presents the design of PFS, a mobility-aware file system specially designed for partially connected operation.*

*PFS supports the extreme modes of full connection and disconnection gracefully, but unlike other mobile file systems, it also provides an interface for mobility-aware applications to direct the file system in its caching and consistency decisions in order to fully exploit intermittent connectivity. Using PFS, it is possible for an application to maintain consistency on only the critical portions of its data files. Since PFS provides adaptation at the file system level, even unaware applications can ‘act’ mobile-aware as a result of the transparent adaptation provided by PFS.*

## 1 Introduction

Due to the increasing number of commercially available local and wide area wireless networking technologies (e.g. Ardis, RAM, CDPD, Metricom, RangeLan2, Wavelan), mobility and network connectivity are no longer mutually exclusive. Since we expect partial (or intermittent) connectivity at bandwidths anywhere from Kbps

to Mbps to be the most common networking mode, there is a critical need for efficient data access mechanisms over variable quality of service networks [1].

A mobility-aware network file system is an important component of any mobile computing environment. Since the storage capacity of most portable computers is much less than the typical file server (or even desktop computer for that matter), there will always be some subset of a user’s files present on the portable. Hand copying of files between the portable and server leads to inconsistencies and inefficiency when important files are missing during disconnection. Also, since the portable spends a lot of time on the road, files may be lost due to breakdown or even theft. Having access to remote files while on the road, as long as it is done in an efficient manner, is a great convenience for the mobile user.

State-of-the-art mobile file systems typically assume two extreme modes of operation – full network connection to a high bandwidth wired network (when the portable is docked to a network access point), or disconnection [4][6][11][13]. When in fully connected mode, the file system *hoards* (or predictively caches) files which the user needs during disconnected operation. When disconnected, the file system reads from, and writes to, the hoarded copy of the file. Two problems arise as a consequence: (a) a critical file may not have been hoarded and may stall the work of the user in

disconnected mode, and (b) files may be inconsistent upon reconnection due to concurrent writes to the hoarded copy and the backbone file system copy. Neither of these problems can be prevented (or even solved satisfactorily) because communication is precluded when disconnected. Given that wireless wide area communication is becoming possible from almost anywhere, the imposition of disconnection as the common untethered mode will be artificial. While recent work has recognized the possible benefits of partial connection [5][7][9][10][12], there exists no file system which provides application-directed adaptation to support partial connectivity as the common mode of operation.

This paper describes the design and implementation of the PRAYER [1] file system (PFS), a mobility-aware file system which optimizes for partial connectivity, and provides a generic interface for application-directed adaptation to varying network quality of service (QoS). The advantages of providing generic support for adaptation in the file system are:

- *Transparency for unaware applications* – Since the file system handles the adaptation on behalf of the application, it can provide mechanisms for mobility support that are transparent to the application.
- *Transparency of consistency mechanisms* – In PFS, applications provide the consistency *policy* to the file system. The mechanisms used to enforce the policy are completely transparent to the application.
- *Reuse* – The traditional approach of writing ‘mobility-aware’ applications can be expensive, since each application has its own special adaptation routines (although one could argue for adaptation *libraries*, but these have limitations, too). For applications which primarily operate on data files, a mobility-aware file system, with simple direction from the application, can provide a fine-grained caching and consistency protocol tailored to each application.

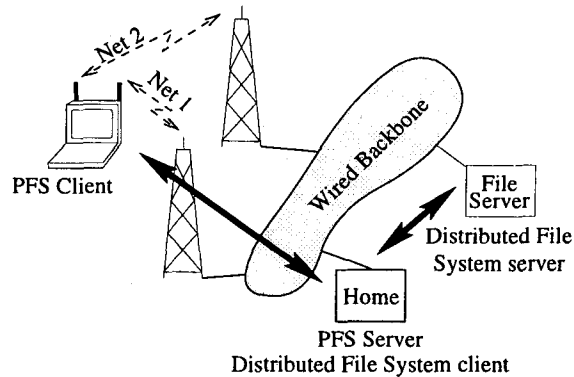


Figure 1: Network Model for PFS.

Two disadvantages of providing adaptation support within the file system are increased complexity and limitations on the types of adaptation possible. In the case of mobile users, the performance of the remote file system is most affected by the speed with which data can be sent across the wireless link. Additional computation on the server and client sides can usually be done for free if it means less data will be transferred over a slow link.

Clearly, there is only so much a file system can know about the data files it stores. Because of this, the file system cannot provide the optimum adaptation for all types of applications. In these cases, specialized mobility-aware applications should be used. However, we feel that a wide range of applications could make exclusive use of the file system for their adaptation needs.

In Section 2, we describe our overall mobile computing model, and the role of the file system in this model. Section 3 describes the techniques used by PFS to support adaptation, Section 4 gives a few examples of how PFS can be used to minimize wireless network traffic, and in Section 5 we give some concluding remarks.

## 2 Model

Figure 1 shows the networking model for PFS. The significant entities are the portable com-

puter (PFS client), the home<sup>1</sup> computer (PFS server), and the file server. We use NFS as the backbone network file system, but any distributed file system could be used. The home computer mounts its shared directories from the file server. The portable computer then caches files from the home computer. Irrespective of the network file system on the backbone, the goal of PFS is to keep files consistent between the home and the portable. This model is considerably different from contemporary disconnected file system models (e.g. Coda or Disconnected AFS), where the portable client interacts directly with the server. We have a three tier model, with client-server interactions between the server and the home, and between the home and the portable. This means PFS can support any number of different backbone file systems, but also precludes end-to-end consistency control. The extra level of indirection enables us to implement efficient application and QoS-dependent adaptive caching and consistency policies, resulting in potentially major time and cost savings during partial connection, while incurring a tolerable overhead during full connection. It also restricts the problem of concurrent read/write access between clients and the backbone file system. The three tier model with an intermediate node for data filtering is an increasingly common technique for providing resource adaptation [3][8].

Based on the above model, PFS provides the following services to applications:

- *Support for a wide range of connectivity.* Hoarding for disconnected operation, and variable-granularity caching and consistency for low-bandwidth operation.
- *Intelligent use of wireless bandwidth.* The necessary or modified portions of an applications files are transparently transferred across the wireless network.
- *Support for application-directed caching and consistency policy.* PFS provides an interface for applications to impose a structure on their files, and then require the file system to keep parts of the file (records or certain fields of all records) consistent with the backbone file system. Depending on the available network quality of service and cost of network access, the application may dynamically change its consistency policy (e.g. move from whole-file consistency to consistency of critical fields in each record).

One key element of any mobile computing environment which supports dynamic adaptation to varying QoS networks is recognizing when to adapt. PFS relies on the existence of an underlying system for notification of QoS changes. When an application using PFS is notified of a QoS change, it can decide whether to modify its current caching and consistency policies.

In the next section we describe how PFS takes advantage of the three tier model to provide variable-granularity caching and consistency control to applications.

### 3 Design of PFS

A block diagram of PFS is shown in Figure 2. The entire system has been designed to operate as an application, rather than as part of the system kernel. The PFS client listens for remote read and write requests from applications at a well known Unix domain socket, and forwards them to the PFS server based on the level of file consistency desired by the application, and the current connectivity of the mobile user. Reads and writes are then cached within the mobile user's local file system. This makes PFS very portable to Unix like operating systems, and portable with slight modifications to Windows 95 (no Unix domain sockets). A detailed explanation of PFS is beyond the scope of this paper, but can be found in [2].

---

<sup>1</sup>The 'home' computer is basically a machine which is designated as the mobile user's PFS server (e.g. the mobile users desktop workstation).

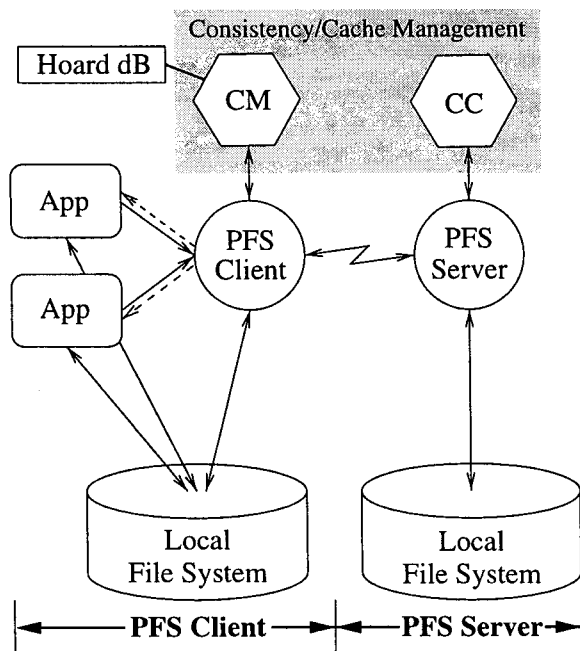


Figure 2: Block diagram of PFS.

Adaptation support in PFS comes in two forms – low-level (at the level of individual reads and writes), and high-level (whole-file and record level consistency control). The PFS client and server provide support for low-level adaptation, and special consistency control processes at the client and server handle the high-level adaptation using the low-level capabilities of PFS.

### 3.1 Low-Level Adaptation Support

To support adaptation to varying network quality of service, PFS provides two I/O functions – `pread()`, and `pwrite()`. As can be seen in Tables 1 and 2, these two functions have two arguments in addition to those found in ordinary `read()` and `write()` system calls.

The `lvl` argument allows the application to specify the importance of network access on an individual read or write basis. For reads, this can be `READ_CONSISTENT`, or `READ_LOCAL`. Consistent reads are always sent to the server, and local reads are always taken from the local file cache. For writes, the consistency level

Table 1: The functionality of `pread()`.

<code>pread(fd, buf, len, lvl, fb)</code>
<b>Operation:</b> The <code>pread()</code> call reads <i>len</i> bytes from file descriptor <i>fd</i> into buffer <i>buf</i> . The <i>lvl</i> option specifies the consistency level of the read operation. For consistent reads, the <i>fb</i> parameter specifies the fallback behavior to use when the server cannot be reached.
<b>Level = <code>READ_CONSISTENT</code></b> <b>Fallback Behavior:</b> ABORT: If unable to reach server, return an error. BLOCK: Block until server can again be reached. LOCAL: If unable to reach server, read from local copy and return.
<b>Level = <code>READ_LOCAL</code>:</b> Read from the local copy and return.

can be `WRITE_THROUGH`, `WRITE_BACK`, or `WRITE_LOCAL`. Write-through operations go directly to the server, write-back operations pass through an outgoing queue, and local writes just go directly to the cache.

The *fb* argument specifies what PFS should do in the event the server cannot be reached when necessary. This fallback behavior gives an application the ability to trade file consistency for performance during intermittent connectivity. Not every application needs every byte of data in their files to be accessible. In most instances, using the previously cached data is good enough.

To support access to remote files by applications which are unaware of PFS, we have built a shared library which *maps ordinary `read()` and `write()` system calls into equivalent calls to `pread()` and `pwrite()`*. The default values for consistency level and fallback behavior are specified using environment variables, and the library is force loaded at run time using the

LD\_PRELOAD capability present on most Unix operating systems.

### 3.2 High-Level Adaptation Support

Sitting on top of the PFS client and server processes are the consistency manager (CM) and the consistency checker (CC) respectively. These two processes work together to support file hoarding and application-directed partial file consistency through the `pconsistency()` [2] call. Table 3 details the functionality of `pconsistency()`.

Table 2: The functionality of `pwrite()`.

<code>pwrite(fd, buf, len, lvl, fb)</code>
<b>Operation:</b> The <code>pwrite()</code> call writes <i>len</i> bytes from buffer <i>buf</i> to file descriptor <i>fd</i> . The <i>lvl</i> option specifies the consistency level of the write operation. For consistent writes, the <i>fb</i> parameter specifies the fallback behavior to use when the server cannot be reached.
<b>Level = WRITE_THROUGH</b> <b>Fallback Behavior:</b> ABORT: Write to server or fail. If remote write succeeds, write to local cache and return. BLOCK: Block until able to write to server. If remote write succeeds, write to local cache and return. LOCAL: Write to server. Regardless of results of remote write, write to local cache and return.
<b>Level = WRITE_BACK:</b> Write to the cache copy, queue write for sending to server, and return.
<b>Level = WRITE_LOCAL:</b> Write to the cache copy and return.

The consistency management features of PFS make use of data file semantics to support fine grained caching and consistency. File format

“templates” are used to describe the structure of an application’s data files. A template can define fixed length records and fields by specifying record and field lengths in bytes. For variable length records, templates can specify record and field boundaries using simple string matching tokens, or complex regular expressions. The reader is referred to [2] for a more detailed description of templates.

Using this structure, it is possible for the consistency manager and checker to maintain consistency on only the blocks or fields of an application’s critical data files, thus saving communication bandwidth and improving performance over slow wireless links.

### 3.3 Conflict Detection and Resolution

When files cached on the portable are modified, the file system can either send updates to the server immediately (for write-through requests), or make the changes locally. When files are shared, there is the possibility for two completely different versions of the same file to exist – one on the portable, and one on the server. PFS will always detect this kind of write/write conflict and report it. PFS takes the position that the server copy is always ‘right,’ and will only notify the PFS client side of the conflict.

Currently PFS does not resolve conflicts. However, we envisage a more intelligent future design which will not only detect, but also resolve conflicts, based on an approach similar to Bayou [14].

In the next section, we give several examples of how PFS can be used to exploit partial connectivity over low-bandwidth wireless links.

## 4 Using PFS for Adaptation

In this section we present a few examples of how PFS can make use of data file semantics to provide adaptation during periods of limited network connectivity.

Table 3: The functionality of `pconsistency()`.

<code>pconsistency(fn, tn, cf, fb)</code>
<b>Operation:</b> The <code>pconsistency()</code> call ensures that the consistency fields ( <i>cf</i> ) as defined in template <i>tn</i> for file <i>fn</i> are consistent with the server. If the server cannot be reached, then the fallback behavior ( <i>fb</i> ) is followed.
<b>Fallback Behavior:</b> NOTIFY: Notify the user of a failure to maintain consistency with the server. LOCAL: Silently fail until reconnection, when consistency is again achieved.

#### 4.1 Simple Databases

Many simple database applications such as calendar managers, phone directories, etc. store their data in structured or semi-structured flat files. In the case of a calendar manager, there could potentially be updates being made at the server (by the secretary), and also at the portable (by the person traveling). This type of file should be kept consistent with the server as much as possible to prevent conflicts.

With PFS, the low-level record based structure of the database file can be described using a template, and the database residing on the portable will be kept consistent with the server at the record level. If one record corresponds to one hour during the day, then maintaining consistency requires that information corresponding to individual appointments be sent across the wireless link. Such small and infrequent updates require very little network communication and can be performed by PFS whether the calendar manager application is running or not.

#### 4.2 Mail Spool File

Email messages within a user's mail spool file

can also be thought of as records in a data file. In fact, the individual fields of the header (i.e. Subject, From, etc.), the message body, and message attachments can be described as fields within each record using a PFS template.

When remotely connected, consistency can be maintained on a few select fields in the header (such as From: and Subject:) to minimize network communication. To read the body or attachments of any particular email message, consistent `pread()` calls can be used.

#### 4.3 Audio Files

PFS can also be used to reduce the amount of data traffic created when playing audio files. For example, by defining a fixed length template, just the first few blocks of an audio file can be fetched and played on the mobile. We've also found that with certain audio types (AIFF and WAV) it is possible to play a subset of records and get a 'gist' of the entire audio clip.

### 5 Conclusions

In this paper we have presented the design of a mobile file system specifically targeted towards partially connected operation. Through three new library calls (`pread()`, `pwrite()`, and `pconsistency()`), PFS supports application-directed partial-file caching and consistency control. Mobility-aware applications provide the file system with semantic information on the structure of their data files, and can then require that the file system keep parts of the file (certain records or certain fields of all records) consistent with the PFS server. Depending on the available network quality of service and cost of network access, the application may dynamically change its consistency policy (i.e. from whole-file consistency to consistency of critical fields in each record). One big advantage of providing adaptation capabilities directly within the file system is that even unaware applications can be supported

by transparently mapping standard `read()` and `write()` system calls into PFS calls.

## References

- [1] Vaduvur Bharghavan. Challenges and solutions to adaptive computing and seamless mobility over heterogeneous wireless networks. *Int. Journal of Wireless Personal Networking*, to appear 1997.
- [2] Dane Dwyer and Vaduvur Bharghavan. Mobility-aware file systems for partially connected operation. *TIMELY group technical report* <http://timely.crhc.uiuc.edu/publications.html>, November 1996.
- [3] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to network and client variation via on-demand, dynamic distillation. In *Seventh Symposium on Architectural Support for Programming Languages and Operating Systems*, October 1996.
- [4] John S. Heidemann, Thomas W. Page, Richard G. Guy, and Gerald J. Popek. Primarily disconnected operation: Experiences with ficus. In *Proceedings of the Second Workshop on Management of Replicated Data*, November 1992.
- [5] P. Honeyman and L. B. Huston. Communications and consistency in mobile file systems. *IEEE Personal Communications*, December 1995.
- [6] L. B. Huston and P. Honeyman. Disconnected operation for AFS. Technical Report 93-3, Center for Information Technology Integration, June 1993.
- [7] L. B. Huston and P. Honeyman. Partially connected operation. *Computing Systems*, 8(4):365–379, 1995.
- [8] Anthony D. Joseph, Alan F. deLespinasse, Joshua A. Tauber, David K. Gifford, and M. Frans Kaashoek. Rover: A toolkit for mobile information access. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995.
- [9] L. Mummert and M. Satyanarayanan. Large granularity cache coherence for intermittent connectivity. In *Proceedings of the 1994 Summer USENIX Conference*, June 1994.
- [10] Lily B. Mummert, Maria R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995.
- [11] Michael N. Nelson and Yousef A. Khalidi. Generic support for caching and disconnected operation. In *Proceedings of the Fourth IEEE Workshop on Workstation Operating Systems*, pages 61–65, October 1993.
- [12] Brian D. Noble, Morgan Price, and M. Satyanarayanan. A programming interface for application-aware adaptation in mobile computing. *Computing Systems*, 8(4):345–363, 1995.
- [13] M. Satyanarayanan, James J. Kistler, and Lily B. Mummert. Experience with disconnected operation in a mobile computing environment. In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, August 1993.
- [14] Douglas B. Terry, Marvin M. Theimer, Karin Peterson, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995.