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.

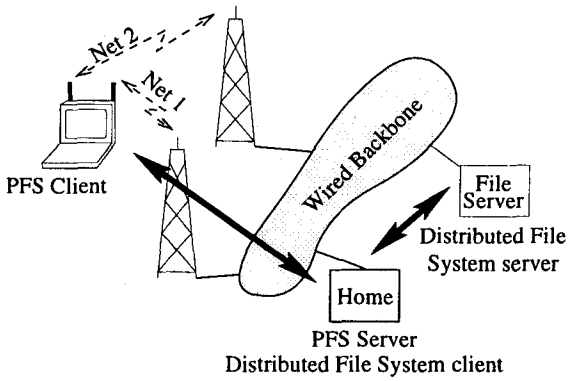

Figure 1: Network Model for PFS.

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[1] 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].

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

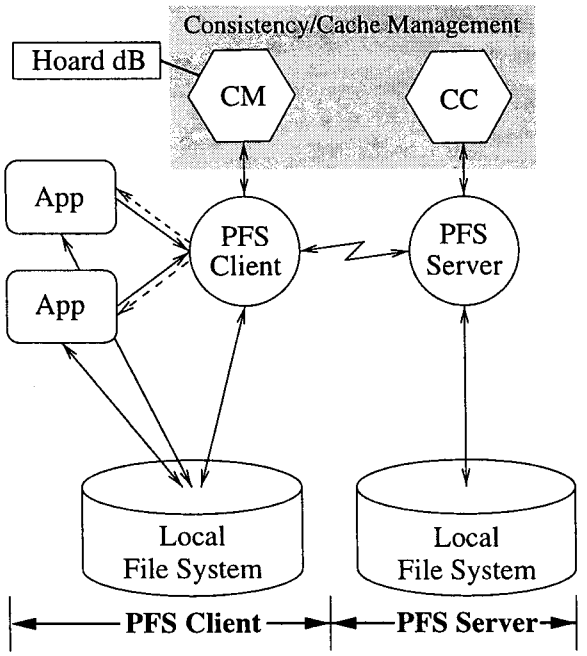

Figure 2: Block diagram of PFS.

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

`pread(`*fd*, *buf*, *len*, *lvl*, *fb*`)`
Operation: The `pread()` call reads *len* bytes from file descriptor *fd* into buffer *buf*. The *lvl* option specifies the consistency level of the read operation. For consistent reads, the *fb* parameter specifies the fallback behavior to use when the server cannot be reached.
Level = READ_CONSISTENT
Fallback Behavior:
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.
Level = READ_LOCAL: Read from the local copy and return.

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 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()`.

`pwrite(`*fd*, *buf*, *len*, *lvl*, *fb*`)`
Operation: The `pwrite()` call writes *len* bytes from buffer *buf* to file descriptor *fd*. The *lvl* option specifies the consistency level of the write operation. For consistent writes, the *fb* parameter specifies the fallback behavior to use when the server cannot be reached.
Level = WRITE_THROUGH **Fallback Behavior:** 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.
Level = WRITE_BACK: Write to the cache copy, queue write for sending to server, and return.
Level = WRITE_LOCAL: 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.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.