Issues In File Cache Management for Mobile Computing

Kevin W. Froese
Richard B. Bunt

Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

{froese, bunt}@cs.usask.ca

Abstract

The emerging demand for mobile computing has created a need for improved file system support for mobile clients. Current file systems with support for mobility provide availability through file replicas cached at the client. These systems manage client file caches in the same manner as file caches in traditional LAN-based distributed systems. However, mobile clients may experience a wide range of network connections which vary greatly in terms of bandwidth, error rates, and expense of use, and these diverse environments add to the complexity of the cache management task. The impact of mobility on cache management has yet to be thoroughly investigated.

This paper examines some of the issues which arise when file caching is performed at mobile clients, and examines the potential of various cache design decisions and cache management techniques to address these issues. Among the issues considered are caching unit, replacement policy, write backs, prefetching, hoarding, and adaptivity. Particular emphasis is placed on the differences between file caching in a traditional LAN environment and in a mobile environment.

Keywords: mobile computing, file caching, cache management

1 Introduction

The continual improvement in the capabilities and portability of notebook computers and hand-held personal digital assistants (PDA's) has given rise to a new style of computer use: that of *mobile computing*. Whereas traditional "stationary" computing consists of users performing work at fixed-location workstations, mobile computing sees the computer accompanying the user, for use in many different locations. Mobile computing allows for *location-independent* work. This method of operation is typically characterized by three actions on the part of the user: disconnection from a network, movement to a new location (along with the mobile client), followed by reconnection to the network and renewed access to data [1]. The goal of system support for location-independent computing is to attempt to make the changes in the computing environment transparent to the user (or at least to the application program), thus allowing the user to work in the same manner and with the same productivity from any location.

In the absence of file system support for location-independent operation, mobile users manually maintain and update separate copies of files at both their personal computer and at a file server [12]. Currently, mobile file systems attempt to remove this burden from the user by employing automatic file replication management techniques [9]. Central to these techniques is the caching of files at client machines. It has been noted that:

> Caching issues are beginning to predominate the open research topics in this area [mobile computing]. In between connected and disconnected states, there are many states of expensive, intermittent and unreliable connections. Adapting caching to these varying situations is a necessity [2].

Since the mobile environment differs significantly from stationary environments (in terms of

[1] It is also possible for a user to resume working at a new location without first establishing a network connection (this is known as "disconnected operation" [4].) We treat this as a special case of the more general problem.

available bandwidth, work performed by users, etc.), it is unclear whether file cache management techniques which are appropriate for fixed file systems will provide the best performance for mobile file systems. This paper examines some of the issues which arise when file caching is performed at mobile clients, and examines the potential of various cache design decisions and cache management techniques to address these issues. Section 2 sets the context by outlining the style of mobile computing considered in this paper. Section 3 describes file caching issues in the context of mobile computing. Section 4 outlines cache management techniques which address these issues. A summary and conclusions are presented in Section 5.

2 Network Connections for Mobile Computing

The needs of the mobile user will determine the type and quality of services which the "mobile-aware" operating system will need to provide. We envision several key characteristics of the mobile user which will influence the design of mobile operating systems. First, the mobile user will require access to the same set of resources and services regardless of his or her working location. As well, users will desire seamless mobility; that is, no special actions should be required of the user when working in different locations, and the user should not be aware of actions taken by the operating system to accommodate mobility. Finally, users will be willing to trade performance for mobility. While a certain level of performance must be maintained, it will not be necessary to satisfy the requests of a mobile user as promptly as those of a user in a LAN environment.

Our focus in this research is on file service, in particular location-independent access to the "home" file system by extending the current LAN-based client-server distributed systems to embrace mobile clients. The "home" location of a user is likely a place of work, a school, etc., and is where the file server resides. While there may be several file servers at the home location, such a configuration can be thought of as a single logical file server without loss of generality. In addition to this location, a mobile user may have other locations from which he/she works. Common examples include a residence or a hotel room. Note that the home location is not necessarily the location where a user performs most of his/her work. Rather, it should be thought of as the (physical) permanent home for a user's files.

A key assumption about this environment is that network connectivity of some sort is required for file system access. Depending on resources available at the point of connection, the characteristics of the connection may vary. In particular, a user may experience a wide range of available bandwidths between various working locations, but the bandwidth at a given location is fairly constant. For example, a user may have access to a 10 Mb/s Ethernet at an office, but only a 28.8 kb/s modem when working from home, and may have no network connection at all while working on a business trip. Our goal is to provide file access across all forms of connection in as transparent a fashion as possible. Effective client caching will be fundamental step in achieving this goal.

We can classify a client's connection status to a file server at any point in time as being in one of three categories:

- *strongly connected* – The client is connected to a file server over a fast and reliable link, such as an Ethernet.
- *disconnected* – The client has no network connection to a file server.
- *weakly connected* – The client is connected to a file server, but only over a slow and possibly error-prone or expensive link [14], such as a modem connection (either wired or cellular), or a radio link.

A mobile client is likely to experience each of these types of connection over some period of time. We assume however that only one type of connection will be experienced during any given working session. It should be noted that while a weakly connected client may temporarily become disconnected due to a network failure, the client is still considered to be operating in a weakly connected environment. A truly disconnected client has no network connection to a file server at all during a session.

While strongly connected, the mobile client operates much as it would were it in a traditional LAN environment[2]. However, it also uses periods of strong connection to prepare for future periods of weak connection or disconnection (this is discussed in greater detail below).

While operating in a disconnected environment, the mobile client must rely totally on its file cache to satisfy all its file system requests (reads and writes). For this approach to be useful, the cache must contain (prior to disconnection) all the files on which the user will want to work while disconnected, and thus whole-file caching, as opposed to file block caching, is used when supporting disconnected operation. Various approaches to "pre-loading" the cache have been advocated, including attempting to keep a user-defined set of files in the cache [16], and applying long-term predictive caching techniques [5, 10]. While operating in disconnected mode, a client can perform any operation on these files that could be performed while connected, but any attempt to access any file not in its cache (i.e. a cache miss) is a fatal error. The user must choose either to work on something else, or to wait until reconnected to a file server in order to retrieve the file.

While disconnected, all of the client's file system operations are logged. Upon reconnection with a network, the log file is replayed, which has the effect of propagating to the file server all file system changes that were made. In this way, the final state of the file system is the same as if the changes were made while connected.

For disconnected operation to be practical, the file system must provide optimistic replication of files. That is, multiple copies of files are allowed to exist and be updated independently, but attempts are made to detect and resolve update conflicts should any occur [1]. Studies have shown that update conflicts occur only rarely [9, 11, 16], and that optimistic replication is a good strategy

[2] We assume that routing support is provided to mobile clients by lower level network services (e.g. Mobile IP [8]).

for a disconnected environment.

Weakly connected operation is similar to disconnected operation, but with a few important extensions. A key advantage is that cache misses are no longer fatal. If the requested file is not in the cache, then the read request is sent to a file server over the weak connection, and the file is then transfered to the client (albeit slowly). When a client issues a write, the operation is saved in the log file but the write may later be sent to a file server over the weak connection to reduce the likelihood of update conflicts occurring. This might be of great benefit in environments where file sharing is common.

File caching in strongly connected environments is well understood, having been used in distributed systems for many years. The use of file caching to permit disconnected operation is a much more recent innovation. While many file caching issues have been addressed (for example by the Coda file system [9]), current systems still have limitations. The management of file caches in weakly connected environments has received less attention, and it is in this environment where many interesting issues arise. Ultimately, good management strategies will be those which provide transparent operability across all connection environments. The next section will review some of the issues which arise when file caching is performed in a mobile environment, with emphasis on the effect of different connection types on these issues.

3 File Caching Issues in Mobile Computing

File caching in a mobile environment presents many of the same challenges which arise in traditional client-server environments, as well as some new ones. However, even caching issues which are understood in LAN environments must be re-examined due to the circumstances of the mobile environment. Mobile users are willing to trade performance for mobility, but they want seamless mobility (i.e. transparent operation across varying connection types). Some of the most prominent design issues surrounding file caching in mobile computing are discussed in this section.

Caching Unit

The caching unit refers to the unit of data transfer between the file server and the cache. File caching is typically done either on a block basis or an a file basis. File caches in LAN environments generally use the disk block as the unit of caching. These are usually small, around 512 bytes in size, although some file systems, such as AFS [6], use much larger disk blocks. Block caching is a well understood process and performs well in the LAN environment since the transfer is very efficient. Block caching is not practical for disconnected operation, however, since caching only part of a file is of little use to applications. Thus, entire files have been used as the unit of caching for disconnected operation where transfers are done once, as part of the pre-loading phase. When operating in a weakly connected environment, however, the question must be revisited.

Operating in a limited bandwidth environment means that the number and size of data transfers should be minimised. The use of small blocks (such as in a strongly connected environment) reduces the size of transfers but increases the number. This may overwhelm the connection since bandwidth is limited. Larger blocks may simplify replacement policies (discussed below), while reducing the number of transfers to an acceptable level. However, latency increases. Further investigation of the implications of these techniques is necessary.

Replacement Policies

Whenever an item must be brought into an already full cache, some other item must be removed to create the necessary space. In existing client-server LAN environments, client file caches typically utilize a Least Recently Used (LRU) replacement policy to make make room for an incoming block. LRU is known to perform well for client file caches in this particular environment [1, 20]. Most current research into mobile file systems has assumed that the LRU replacement policy will continue to be suitable for a mobile client. While there are exceptions to this prevailing opinion, such as [6], even these researchers have not yet investigated alternatives.

There are several factors which make the replacement decision appropriate to a mobile environment different from that appropriate to a LAN environment. First, since a mobile client is so dependent on the contents of its file cache, caches will likely be significantly larger than in a desktop workstation. This in turn means that replacement decisions will need to be made regarding files which have not been referenced for a much greater period of time. It is also possible that users may perform different work in a mobile environment than when they are strongly connected, and this could result in different patterns of file system requests.

There are also new considerations which complicate the replacement decision in a mobile environment. One of the most pressing of these is the fact that if whole file caching is being used, replacement policies can perform significantly differently than when block caching is used. For example, if an LRU policy is used with whole file caching, it is possible for the hit rate achieved by a smaller cache to exceed that of a larger cache. As well, the use of file caching could affect the degree of locality present in file reference patterns. Finally, the *number* of files which can be cached is a function of the *sizes* of the files cached. Better performance may be achieved if the replacement decision considers the sizes of the files present in the client cache.

Write Back Policies

A write back is the propagation of file system mutating operations back to a file server. In this way, the results of a write at a client are reflected in the file system rather than just stored in a log file. A cache management system's write back policy determines when and how write backs occur.

How and when to perform write backs is more of an issue in a mobile environment than in a LAN environment, where bandwidth is always available to quickly propagate file modifications to a filer server. Two conflicting goals complicate write backs in the mobile environment. First, the potential for update conflicts to occur due to optimistic file replication motivates prompt write backs to minimize the chances of such conflicts. However, many file updates are later negated by subsequent updates. For example, a file may be created, written to, then deleted, removing any need for previous updates to the file to be sent to a file server. This means that the longer write backs are delayed, the fewer write backs will actually need to be performed [3]. Further complicating the write back issue is the fact that in a limited bandwidth environment, it may not be desirable to consume bandwidth with write backs if this has too great of a negative impact on the performance of other (e.g. read request and replies) network usage.

For all of the difficulties associated with write backs in mobile environment, there are several advantages to performing them. First, once a write has been sent to a file server, it no longer needs to be stored in the client's log file. This has the advantage of keeping the log file small, which can be important if disk space is limited at the client (which it usually is). As well, if writes are propagated to a file server while weakly connected, there are fewer events that need to be replayed upon reintegration with the network. Thus, the time needed for reintegration will be shorter. Finally, performing write backs allows a file server to commit file modifications to disk, reducing the chance of update conflicts.

Prefetching

In a traditional (fetch on demand) cache management scheme, the first reference to a file will always result in a miss, since the file can be in the cache only if it was previously referenced. In a traditional distributed file system setting, this first miss is of little concern, but in a location-independent environment this can have a large impact on performance. For example, should a weakly connected user wish to run an application which reads from many separate data files, none of which are cached, each file would have to be sent to the client from the file server as it was needed by the application. If the data files were large enough that the cache could not hold them all, each invocation of the application would take a significant amount of time. However, if the file system is able to estimate future references (based, perhaps, on past references), it could begin to send those files to the client *before* they are requested. In this way, the high latency of file accesses could be hidden. Prefetching is *essential* in a disconnected environment, and is appealing in a weakly connected environment, not because there is insufficient bandwidth to meet demand but because there is insufficient bandwidth *at any instant.*

The ability of prefetching to hide the latency of a slow network connection makes it extremely attractive when viewed from the perspective of mobile computing, particularly during periods of weak connection. However, the ability of any prefetching mechanism to successfully anticipate future needs is limited, and this creates several problems. Not only is there the potential for bandwidth to be wasted due to poor prefetching decisions (which occurs any time a prefetched file is replaced in the cache before it is referenced), but there is also the problem of how prefetched files should be accommodated by the replacement policy. If prefetched files are managed in the same way as demand-fetched files, the problem of choosing which resident file to replace for a prefetched file arises. If prefetched files are treated in a special way, the cache's replacement policy must become much more complex. While these issues have prevented prefetching from gaining much acceptance in traditional distributed environments, prefetching may be more appealing in mobile environments.

Hoarding

A fundamental part of client file caching in a mobile environment is to attempt to ensure that files are cached before they are needed by a weakly connected or disconnected user. Hoarding is often used to achieve this goal. The process of hoarding can be viewed as a mass pre-load of a client's file cache prior to mobile activity. Hoarding is similar in concept to prefetching. It is concerned with the loading of the file cache while a user is strongly connected, in anticipation of a future disconnection from the network. The term comes from the Coda file system [16], which has developed a mechanism for user-aided gathering of files at a mobile client. The goal of hoarding is to provide file availability over a longer period than prefetching is concerned with. For this paper, the term hoarding will be used to refer not only to the specific techniques used in Coda, but more generally to any method of "heating" a mobile client's file cache while strongly connected.

What makes hoarding a challenge is the fact that the files chosen for caching must serve the file system needs of the mobile user for an extended period of time (perhaps for several days or more). It is obviously difficult to predict a user's needs for such periods of time, and doing so may require storing large amounts of information about the user's past file reference history. Because of this, it may be necessary to involve the user in the hoarding process, as the Coda file system does. However, this raises the further issue of how to strike a balance between effectiveness of hoarding and intrusiveness to the user.

Adaptivity

The ability of cache management policies to adapt to changes in the operating environment will likely be critical to achieving good file cache performance for mobile clients. Not only can the available bandwidth vary with location, but also because the work that a user performs may change as well. Thus the request stream seen by the file cache may have time varying characteristics. Since it may be difficult to know how best to adapt to various working environments, it may be necessary to solicit advice from the user on how various situations should be handled. This could be especially important in a weakly connected environment, where a poor cache management decision could result in a much greater drop in performance than if the client was strongly connected [14]. It is important to note that in order for cache management policies to be adaptive, there must be some mechanism in place for the operating system to monitor and detect changes in the characteristics of the current network connection and inform the cache of these changes.

4 File Cache Management for Mobile Computing

The management of a client's file cache in a distributed system is a process that is fairly well understood. However, cache management in a location-independent environment is a significantly different problem, and management policies must be examined in this new context if performance expectations are to be met. A successful cache management mechanism has several attributes. It must be robust, being able to cope with a wide range of connection types and qualities. As well, cache management should be adaptive in nature, capable of modifying its behaviour in response to changes in the operating environment. Whenever possible, the caching mechanism should be able to hide its actions from higher levels, such as applications and the user. Good performance, in terms of metrics such as file availability and latency, is always desirable. Finally, a good user interface should allow the user some control over certain aspects of cache management. Some thoughts on the design and management of client file caches in a location-independent environment follow.

Caching Unit

The choice of caching unit for a mobile client's file cache is significantly more complex than the same decision for a stationary workstation. In a LAN environment, the use of small disk blocks is appropriate since this leads to a reduction in cached bytes which are never referenced, and network speeds are high enough that performing many block transfers does not introduce unreasonable delay. This approach is not appropriate for mobile environments, where a client's network connection may be slow or non-existent. Two possible ways of addressing this may be to use variable-sized blocks, or to cache groups of files as a single unit.

The variable block size design attempts to capture the advantages of block caching, while recognizing that small block sizes are impractical in low-bandwidth situations. The basic idea is to cache large blocks when the client has a low-bandwidth connection, and to cache small blocks in high-bandwidth situations. When a client is strongly connected, the size of block cached is the same as in a LAN environment, while whole file caching is used when the client is disconnected. When the client has a weak connection, the size of block cached falls somewhere between these two limits. The adaptiveness of this approach makes it an attractive option.

Another possible approach is to use groups of files as the unit of caching, such as in done in the Seer predictive caching mechanism [10]. The idea is to place semantically related files in the same group, and then to cache these files as a single unit. While the identity of which individual files are cached is still maintained for the purposes of satisfying file system requests, entire groups are either cached or replaced as a single entity. One of the drawbacks of this approach is that it does not consider the type of connection possessed by the client, and transferring groups of files in a low bandwidth environment may not be practical.

Replacement Policy

The design of a replacement policy for clients in a LAN environment is simplified by the fact that client file caches are relatively simple, and therefore simple replacement policies, such as LRU, perform well. LRU is an easy replacement policy to implement, and it results in an acceptably high cache hit rate, which in turn helps improve overall file system performance at the client. In the LAN environment no single cache miss will have any significant impact on performance at the client so the use of a more complicated replacement policy in order to achieve a slightly higher hit rate may not be warranted. These same conditions do not hold in a mobile environment, and this makes the choice of a replacement policy for a mobile client's file cache much more difficult.

In the mobile environment, all cache misses do *not* have the same performance impact. For a disconnected client, *any* cache miss will have a significant negative impact on the ability of the user to perform useful work, since being unable to access any one file could prevent him/her from completing the task. In a weakly connected environment, a cache miss on a large file will stall the user for much longer than a miss on a small file. These factors mean that a simple policy such as LRU may not perform adequately in a mobile environment.

There are several known approaches which could prove be useful in managing replacement in a mobile client's file cache. *Frequency Based Replacement* [15], with its combination of frequency and recency information, may help keep critical, often used files in the cache. As well, this policy could be made adaptive by allowing certain parameters to change as the operating environment changes. Another possibility is to allow certain files to be exempt from consideration for replacement, effectively "pinning" these files in the cache. The choice of caching unit could also have an effect on the choice of replacement policy. For example, if file groups are the unit of caching, then the choice of which group to replace in the cache could consider such factors as the total size of the files in the group, how often the user accesses the group, or observed patterns regarding which groups are accessed shortly after others.

Write Back Policy

Write back policies in LAN environments are strongly tied to the need to maintain file consistency throughout the file system. In many distributed file systems, a "write-on-close" policy is used, where all updates to a cached file are sent to the file server as soon as the file is closed. As well, the file system ensures that cached copies of files which become modified are invalidated, preventing the use of stale data.

The use of optimistic replication in the mobile environment significantly changes the design of a write back policy for client caches. The fact that it is not *necessary* to propagate changes to cached files immediately allows for write backs to be delayed for longer periods of time. This is important since it allows for more flexible scheduling of network activity, which becomes more critical as available bandwidth varies with time.

In the Coda file system, the Venus cache manager employs a "trickle reintegration" mechanism [14]. This is a delayed write back mechanism which becomes active only when there is bandwidth available. AFS [6] uses a similar mechanism, with three separate network queues (one for interactive traffic, one for other network traffic, and one for replay traffic) and the use of lottery scheduling (a proportional-share randomized resource allocation algorithm [19]) to arbitrate network access between the queues. While these policies "naturally" adapt to the amount of bandwidth available, they may not perform well in an environment with a high degree of file sharing, due to the low priority placed on performing write backs.

Other policies are also possible. For example, it may be possible to achieve some of the performance benefits of write backs without consuming the same amount of bandwidth [7]. By simply sending a small notification to the file server that a file has been modified, the server may be able to help reduce conflicts (especially if the server is allowed to initiate write backs [18, 17]). The performance benefit of using write notifications would depend on the amount of file sharing which occurs in the mobile environment.

Prefetching Policy

The availability of a fast network, combined with the difficulties of predicting future file system requests, has prevented prefetching from producing any significant performance improvements in LAN-based distributed file systems. To date, few mobile file systems have implemented any prefetching mechanism, and research in this area is sparse.

Lei and Duchamp [13] have proposed a file-based prefetching policy for use with mobile clients. The approach taken is to store file reference patterns in tree structures, and then attempt to match current references with a stored pattern. As soon as a match is made, the remaining files in the pattern are prefetched. Once files have been prefetched, they are treated the same as demand-fetched files, and are subject to LRU replacement. There are several parameters in this algorithm which could be changed dynamically to allow the policy to become adaptive.

It may also be possible to extend a system such as the Seer predictive caching mechanism, designed for use in disconnected operation, to operate as a prefetching system in a weakly connected environment. This could be done by allowing Seer to track file system activity and determine file groupings while weakly connected, much as it does when it is strongly connected, but allowing new file groups to be cached "on the fly".

Hoarding Policy

Hoarding is an action unique to mobile computing. With no history of hoarding in LAN environments to draw upon, hoarding policies have been developed which take significantly different approaches to the problem of pre-loading a mobile client's file cache.

The simplest approach to hoarding is to rely simply on the contents of an LRU cache to contain the files which will be needed in the near future, such as is done in AFS. While simple, this approach likely does not capture the long-term patterns in a user's file system requests and thus may not meet the needs of users who may be mobile for extended periods.

In Coda, the file system keeps track of a user's reference history, and uses this along with a user-provided hint file (called a *hoard database*) to load a client's cache with files that the user will (hopefully) want to use while disconnected. Each file has some priority associated with it (provided by either the user or the system), and these values are used to help determine which files should be cached if not all can be accommodated. The one drawback of this approach is the need for user involvement in the hoarding process.

The Seer predictive caching mechanism for the Ficus file system completely automates the hoarding of files. Seer attempts to identify groups of files are semantically related (i.e. they are related to each other in how and when they are used), and then to cache these file groups as a single unit. That is, if two files are used by the user to perform the same "task", the files should be placed in the same caching group. A task could be such things as document creation (involving text editing, formatting, viewing, etc.), programming (involving source code editing, compiling, linking, executing, etc.), or information browsing (involving the use of a world wide web client and associated helper applications). The motivation for this design is that if a user accesses some files used to perform a given task, he/she is likely to require the other files to complete the task. Similarly, if some files needed to perform a task are *not* cached, then there is little benefit to caching related files. Implementation of such a design requires the tracking of user reference patterns over long periods of time, as well as an algorithm to identify semantically related files based on this reference history.

Adaptivity

The need for cache management algorithms to be able to adapt to changes its operating environment is also unique to mobile computing. Many of the policies discussed above modify their behaviour to better suit the current operating environment. The most important factor likely to change from one location to another is amount of bandwidth provided by the current network connection. The management of write backs and prefetching, and the choice of caching unit all should adapt to the current network characteristics to improve their performance.

As noted above, the unit of caching could be made adaptive by increasing the size of blocks cached as available bandwidth decreases. Write back policies described earlier "naturally" adapt to low-bandwidth situations by performing write backs only when the network is otherwise unused.

A simple way in which a prefetching algorithm could be made adaptive is to adjust the *amount* of data prefetched to match the available bandwidth. That is, if a client is strongly connected, very aggressive prefetching could be performed, since "wasting" some bandwidth by prefetching files which are never referenced is likely of little consequence. However, when the client is weakly connected, the amount of data prefetched could be reduced, to ensure enough bandwidth for other types of requests (such as demand fetched files or blocks).

5 Summary and Conclusions

The wide range of environments a mobile client may experience creates many challenges for file cache management. Strongly connected, weakly connected, and disconnected environments require different approaches to cache management. Robustness across a range of connection types will be a key requirement for the mobile user. While strongly connected operation is a familiar situation, any new cache management policies must continue to perform well in this environment. In some ways the lack of a network connection greatly simplifies cache management when operating in disconnected mode (since no new files can be cached or written back), but this places a much greater burden on the hoarding policy to be able to correctly identify files which will be required in the near future. Weakly connected operation stresses any cache management policy, forcing it to be able to make maximum use out of limited bandwidth.

This paper has examined some of the issues which arise when performing file caching in a mobile environment, and has discussed the role of client cache management in addressing these issues. Most of these issues require much more examination in order to find robust and adaptive solutions. New management policies need to be designed, tested, and implemented if an acceptable level of file system performance is to be achieved for mobile clients.

Ongoing work is seeking to evaluate some of the file cache management approaches discussed in this paper. Simulations are being used to examine how well various cache management policies (both existing and proposed) perform across a range of operating environments. It is hoped that this work will identify successful management policies as well as discover areas where better policies need to be designed.

References

[1] Baker, M., Hartman, J., Kupfer, M., Shirriff, K., and Ousterhout, J. Measurements of a distributed file system. In *Proc. of the Thirteenth ACM Symposium on Operating System Principles* (Pacific Grove, CA, Oct. 1991), pp. 198–212.

[2] Banerji, A. Answers to frequently asked questions for comp.os.research: Part 1. Available at `http://www.maths.tcd.ie/scrg/os-faq/FAQ-1.html`, 1995.

[3] Chen, K., Bunt, R., and Eager, D. Write caching in distributed file systems. In *Proc. Thirteenth International Conference on Distributed Computing Systems* (Vancouver, BC, May 1995), pp. 457–466.

[4] Chikarmane, V. Disconnected operation in a distributed system. In *Proc. Sixth Annual Graduate Symposium on Computer Science* (Dept. of Computer Science, University of Saskatchewan, March 1994), pp. 23–32.

[5] Ebling, M., Mummert, L., and Steere, D. Overcoming the network bottleneck in mobile computing. In *Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, Dec. 1994).

[6] Huston, L., and Honeyman, P. Disconnected operation for AFS. In *Proc. First USENIX Symposium on Mobile and Location-Independent Computing* (Cambridge, MA, April 1993), pp. 1–10.

[7] Huston, L., and Honeyman, P. Distributed systems support for telecomputing. Technical report, Center for Information Technology Integration, University of Michigan, Ann Arbor, MI, 1994.

[8] Johnson, D. Scalable and robust internetwork routing for mobile hosts. In *Proc. Fourteenth International Conference on Distributed Computing Systems* (Poznan, Poland, June 1994), pp. 2–11.

[9] Kistler, J., and Satyanarayanan, M. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10*, 1 (Jan. 1992), 3–23.

[10] Kuenning, G. The design of the Seer predictive caching system. In *Proc. Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, Dec. 1994).

[11] Kuenning, G., Popek, G., and Reiher, P. An analysis of trace data for predictive file caching in mobile computing. In *Proc. 1994 Summer USENIX Conference* (Los Angeles, CA, June 1994), pp. 291–303.

[12] Landay, J., and Kaufmann, T. User interface issues in mobile computing. In *Proc. Fourth Workshop on Workstation Operating Systems* (Napa, CA, Oct. 1993).

[13] Lei, H., and Duchamp, D. Transparent file prefetching. Technical report, Department of Computer Science, Columbia University, New York, NY, 1995.

[14] Mummert, L., Ebling, M., and Satyanarayanan, M. Exploiting weak connectivity for mobile file access. In *Proc. Fifteenth ACM Symposium on Operating Sytems Principles* (Copper Mountain Resort, CO, Dec. 1995), pp. 143–155.

[15] Robinson, J., and Devarakonda, M. Data cache management using frequency-based replacement. In *Proc. 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Boulder, CO, May 1990), pp. 134–142.

[16] Satyanarayanan, M., Kistler, J., Mummert, L., Ebling, M., Kumar, P., and Lu, Q. Experience with disconnected operation in a mobile computing environment. In *Proc. First USENIX Symposium on Mobile and Location-Independent Computing* (Cambridge, MA, April 1993), pp. 11–28.

[17] Tait, C., and Duchamp, D. An efficient variable-consistency replicated file service. In *Proc. USENIX File Systems Workshop* (Ann Arbor, MI, May 1992), pp. 111–126.

[18] Tait, C., and Duchamp, D. An interface to support lazy replicated file service. In *Proc. Second Workshop on Management of Replicated Data* (Monterey, CA, Nov. 1992).

[19] Waldspurger, C., and Weihl, W. Lottery scheduling: Flexible proportional-share resource management. In *Proc. First Symposium on Operating System Design and Implementation* (Monterey, CA, November 1994), pp. 1–11.

[20] Willick, D., Eager, D., and Bunt, R. Disk cache replacement policies for network file servers. In *Proc. Thirteenth International Conference on Distributed Computing Systems* (Pittsburgh, PA, May 1993), pp. 2–11.