File System Support for Weakly Connected Operation

Kevin W. Froese

Supervisor: Rick Bunt

Abstract

With the growth of interest in mobile computing, there has been an interest in providing file system support for clients with low bandwidth network links to file servers. Most techniques for supporting these weak connections have been based upon the disconnected computing paradigm. However, there are significant differences between these two environments. This paper examines the issues that arise in supporting weakly connected operation, with special attention paid to the role of write backs. The results of a small performance study are presented, which shows the potential benefits of delaying write backs. The paper concludes with a discussion of areas where more work is warranted, and with the presentation of ideas for new ways of supporting weakly connected operation.

1 Introduction

In a distributed (client-server) computing environment, there are various types of network connections that a client may possess. A *strong connection* is said to exist when the client is connected to a network over a fast and reliable link, such as an Ethernet. This is the most common type of connection. However, due to either failures or voluntary actions, a client may have no network connection at all, in which case it is said to be *disconnected.* Significant research has done on providing file system availability for disconnected clients [2, 5, 6, 11, 14]. In a mobile computing environment, both of these types of connections may occur, but there is also a third possibility. A client which has access to a file server, but only over a relatively slow and possibly unreliable link is said to be *weakly connected.* Little work has been done on exploiting such connections. It has been recognized that a weak connection can be used to provide access to files residing at a server. However, it is likely that there are also significant performance benefits in allowing the client to write data back to the file server, and it is this which has provided the motivation for this research.

This paper begins with an examination of the work which has been done towards providing support for disconnected operation. This provides a background for a discussion of weakly connected operation. The use of write backs in a weakly connected environment is then examined in detail. The results of a small performance study are presented, followed by a discussion of extensions to previous work, and finally the results and conclusions are presented.

2 Disconnected Operation

Disconnection occurs whenever a client in a distributed system does not have access to a file server. There are two types of disconnection in a mobile environment: *voluntary* and *involuntary* [14]. Involuntary disconnection occurs when a fault severs a client's communication link with a file server. In traditional (non-mobile) distributed systems this is the only type of disconnection which can occur. Voluntary disconnection occurs when a client is physically separated from a network, usually for the purpose of continuing work in new location. File systems such as Coda [6] and AFS [5] have been designed to allow disconnected clients to continue to work in a manner as similar as possible to fully connected operation.

2.1 File System Support

File system support for disconnected operation centres around the idea of maintaining a file cache at the client, so that when disconnection occurs the client can continue to operate on these files. (It is important to note that whole-file caching, as opposed to file block caching, must be used when supporting disconnected operation.) For this approach to be useful, the cache must contain (prior to disconnection) the files which a user will want to work on in the near future. This is accomplished through the caching of recently used files, by attempting to keep a user-defined set of files in the cache [14], and through long-term predictive caching techniques [3, 7]. While operating in disconnected mode, a client can perform any operation on these files that could be performed while connected. However, if a client tries to access any file not in its cache (a cache miss), an error occurs. The client must either choose to work on something else, or must wait until reconnected to a file server in order to retrieve the file.

While disconnected all file system operations which a client performs are recorded in a *log file*. This log is used to ensure that the work done while disconnected can eventually be stored at a file server. Upon reconnection with a network, the log file is replayed, which has the effect of propagating all file system changes which were made to the file server. In this way, the final state of the file system is the same as if the changes were made while connected. Log files are usually optimized in order to save both disk space at the client and to decrease the time needed for reintegration. This is accomplished through recognition of which operations actually affect the final state of the file system. Operations which do not change the state of the file system (such as reads) are called *non-mutating*, and are not recorded in the log file. *Mutating* operations are logged, but only if no subsequent mutating action "cancels" the operation. For example, if a file is created then removed, the final state of the file system has not changed, and therefore neither the creation nor the removal need to be logged.

2.2 Consistency Issues

A file system's *replication strategy* determines what type of operations are allowed on replicas of files. A *pessimistic* strategy allows only one copy of a file to be updated at a time, and ensures that all copies are then modified to reflect these changes. An *optimistic* strategy allows replicas to be modified independently, but still attempts to detect and resolve conflicts when they occur [2]. An optimistic replication strategy must be used by a file system which allows disconnected clients to modify files. Such a strategy allows for the possibility that a cached copy of a file may be modified at the same time that a copy residing on a file server is changed. This is known as a *write-write conflict*. Studies have shown that such conflicts occur only rarely [6, 8, 14], and that optimistic replication is a good strategy for a disconnected environment. However, some write-write conflicts do occur, and these must be resolved. Currently, manual intervention is often required for this, although there has recently been work on the development of *application specific resolvers* [9], which would either partially or totally remove this burden from the user.

3 Weakly Connected Operation

Mobile clients differ from stationary clients in two main ways. First, they tend to be resource-poor compared to desktop workstations. Processing power, memory and disk capacity, and network bandwidth of mobile clients may all be several orders of magnitude less than those of a stationary client [10]. As well, mobile clients may often possess only a weak connection to a file server, or possibly no connection at all. Common types of weak connections include modem links (either wired or cellular, with bandwidths of up to 14.4 Kbits/second), radio links (with bandwidths of up to 19.2 Kbits/second),

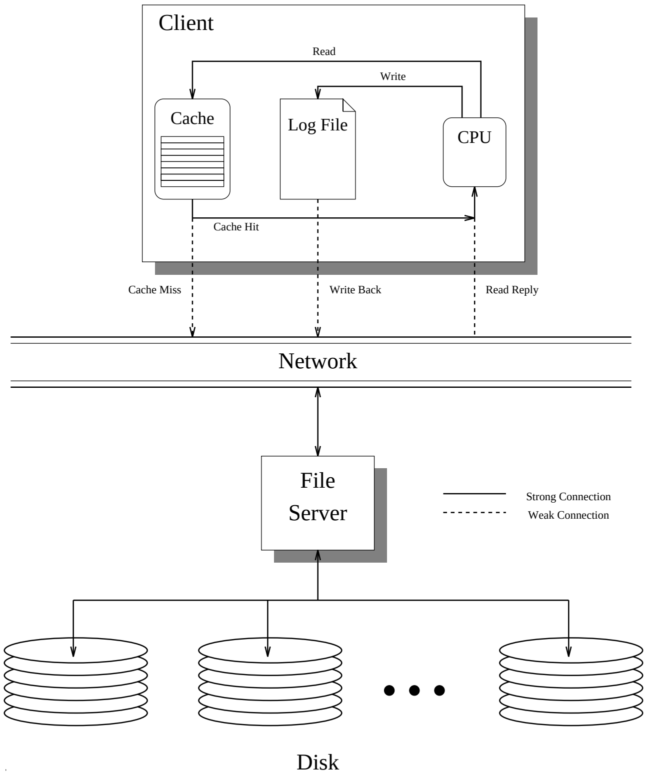

Figure 1: Reads and writes in a weakly connected environment

and wireless local area networks (with bandwidths of up to several Mbits/second). The key feature of these types of connection is that they offer significantly lower bandwidth than a wired LAN. Thus, it would not be practical to support weakly connected clients in the same manner as fully connected clients. While the techniques used for supporting disconnected operation form a useful base towards supporting weakly connected operation, there are significant differences between the two environments which must be addressed by a file system.

Figure 1 shows how reads and writes would be handled in a weakly connected environment. Whenever a client attempts to read from a file, the cache is searched for the file. If the file is present in the cache, it is read into memory and no network requests are needed. If the 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. When a client issues a write, the operation is saved in the log file. The write may later be sent to a file server over the weak connection.

3.1 Servicing Read Misses

The most important advantage of having a weak connection (as opposed to being disconnected) is that cache misses are no longer nearly as critical. If a client needs to access a file which it does not have in its local cache, that file can be retrieved from the file server, even if the latency is rather large. In this respect, a weakly connected client operates much like it is fully connected. However, the more files a client needs to fetch from a file server, the worse the perceived performance will be for the user. This is because current systems use whole-file caching, and therefore the entire file must be downloaded before the read is completed. All successive reads to the file are then satisfied at the local cache. While other approaches are conceivable, such as downloading only the actual file blocks needed by the client (to reduce latency), the performance impact of such designs has not been investigated. Therefore, it is still very important that the local cache contains as many of the files which will be needed in the near future as possible.

3.2 Dealing with Writes

Another aspect of weakly connected operation is that file system mutating operations can be propagated back to the file server. There are both advantages and disadvantages to performing write backs, and there are many issues involved in determining the performance impact of using write backs. The use of write backs in weakly connected environments has not been extensively studied in the literature. The issues surrounding the implementation of a write back mechanism will be discussed in detail later.

3.3 Resource Conservation

The overriding principle which must be followed when providing file system support for weakly connected operation is that of bandwidth conservation, since this is likely the most precious resource at a mobile client [3]. It may often be necessary to use more of other resources, such as disk space and CPU time, in order to keep bandwidth requirements to a minimum [4]. The servicing of cache misses should be given the highest priority on bandwidth, since a user may be effectively blocked from working until a file is received from a file server. All other functions requiring bandwidth (such as write backs) should be deferred until there are no cache misses to be serviced.

4 Dealing with Writes

In the context of weakly connected operation, 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. There are several reasons why the use of write backs is desirable. First, once a write has been sent to a file server, it no longer needs to be stored in the log file. This has the advantage of keeping log files 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, the quicker a file server is able to commit the changes a weakly connected client makes to files, the less chance there is of a write-write conflict occurring in that file. Since resolving such conflicts can be difficult, reducing the probability of them occurring is beneficial.

4.1 Implementation Issues

One of the most important issues in implementing a write back scheme is when to actually perform the write back. This problem actually occurs at two different granularities. The first is how often write backs should be performed. For example, should they be done immediately after a write operation, or should they be delayed an hour, or many hours? This is the *scheduling* decision. Once a scheduling decision has been made, there is also the problem of exactly when to initiate a write back. That is, should it be performed as soon as there is available bandwidth, or should some heuristic be used to try to wait until the weak connection is "idle" before initiating the write back? Each of these problems is examined in detail next.

4.1.1 Scheduling

There advantages to both undelayed and delayed write backs. The quicker mutating operations are written back to a file server, the less chance there is of a write-write conflict occurring in the modified file. As well, since writes which have been written back to a file server no longer need to be stored in the log file, quick write backs have the advantage of less disk space being used at the client. However, by delaying write backs for some amount of time there is a chance that a given write may be "cancelled" by a subsequent operation. For example, a temporary file may be created then removed within a short time period. If the write back of the file creation is delayed until after the time the file is removed, no write backs have to be sent at all. Another possibility is that there may be multiple writes to the same file within a short time period. By delaying write backs, it may be possible to send a large group of changes in a single write back, instead of having to perform a separate write back for each modification. This would be advantageous since, due to communication setup and tear-down overhead, fewer longer conversations require less bandwidth than many short conversations [12]. Since bandwidth should be conserved whenever possible, it is likely that delaying write backs would result in better performance at the client as less bandwidth should be required. This in turn keeps more bandwidth available for the servicing of cache misses.

4.1.2 Initiating Write Backs

The second problem of when to write data back to a file server occurs at a much finer time granularity. Given that it has been decided that a write back should be performed at some time, there is the issue of exactly when to initiate the transfer. There are two basic options. The simplest approach is to initiate the write back as soon as there is bandwidth available. (However, if there are both reads and writes waiting for the network, reads would likely be serviced first). Such a system could be implemented through the use of a priority queue for network access, with read traffic given the highest priority [4]. This approach does not take into account that shortly after the write back has begun there may be a cache miss which will require servicing. A variation on this approach is to allow the write backs to be preempted by read requests. While this approach has been mentioned in the literature, it has not been determined whether this approach is feasible or desirable in an actual system.

An alternative to initiating a write back as soon as possible is to wait until the connection is "idle", so as not to interfere with any read requests. The obvious problem with this approach is determining when there will be a period in the future during which no reads will occur. This problem is not simple to solve, and thus may not be practical in an actual system.

It should also be noted that mobile clients will often spin-down their drives in order to save power. This puts yet another condition on when write backs should be initiated. It would be undesirable to spin-up the drive just to transmit a write back. Therefore write back initiation should only occur once the drive is already active.

4.2 Design Proposals

There have been several ideas proposed for the design of file system support for weakly connected operation. Almost all of these include the use of delayed write backs, waiting until the network is free until initiating write backs [2, 4, 14], although some choose not to use write backs at all, and only use the network for servicing read requests [5]. There seems to be little consensus on whether support for the preemption of write backs by reads should be included.

Several techniques have been proposed for reducing the amount of bandwidth required by write backs. The use of batching and compression of write backs has been suggested [14]. This would involve "storing up" writes and then sending multiple updates in a single write back. This batched write back would be compressed before transmission, using one of many available file compression algorithms. This approach trades computation time (both at the client and the file server) for shorter transmission times. The use of *difference-based updates* has also been proposed [10]. Instead up transmitting an entire modified file, this method sends only enough information to indicate what has changed within the file. This strategy could also be used in conjunction with batching and compression.

Replay filtering is a proposed technique for reducing the number of write backs which need to be sent [4]. This method allows the user to specify that some operations should not be written back to a file server. This may be desirable when dealing with application-generated files. For example, if a weakly connected client compiled a program from source code, it would not necessary for the resulting object and application files to be written back to a file server. By just writing back the source files, the program could be compiled by other strongly connected clients themselves if they so desired.

Trace	Duration (hours)	Files Referenced	Total Events	Write Events
1	1.20	60	8424	956
2	1.11	53	6106	916

Table 1: Trace characteristics

It has also been suggested that weakly connected clients should be able to dynamically adapt to varying network performance [14]. For example, if the available bandwidth drops below some value, write backs may be postponed until network performance improves. This may be important if connectivity is intermittent, which may well be the case in a mobile computing environment. However, exact methods for monitoring and quantifying network performance have not been given, nor have specific ideas for how clients should change their behaviour when network performance changes been proposed.

5 Quantitative Analysis

To gain some understanding of the feasibility of using a write back scheme in a weakly connected environment, a small study was undertaken to determine the effect of delay length on the number of write backs which need to be sent to a file server. If the number of write backs decreases rapidly with increased delay length, this would suggest that write backs could likely be implemented with only small bandwidth requirements (if sufficiently long delay lengths are used). If the number of write backs is not greatly affected by delay length, the bandwidth requirements of using write backs may outweigh any potential benefits offered.

While an analysis of the performance benefits to be gained from delaying write backs is an important first step in investigating the use of a write back mechanism, there are several other factors which should also be considered if a full study was to be performed. The rate at which write-write conflicts occur must be measured in order to set limits on how long writes should be delayed. As well, the ratio of read to write activity will have a large impact on bandwidth available for performing write backs. Other factors, such as long-term user file access patterns while weakly connected (i.e. over several days), and the error rates associated with the transmission medium, may also effect the performance of the write back scheme. While this study does not investigate these factors, it provides an indication of whether a more detailed study of this type is warranted.

5.1 Methodology

Trace driven simulation was used for the experiments. Two traces were gathered on non-mobile workstations attached to a (strongly connected) Ethernet LAN, since no facilities were available for tracing actual weakly connected activity. However, the results are still relevant since it is the goal of mobile computing to allow for the same work to be carried out regardless of a client's connectivity status. Thus, the work represented by the traces, text editing and document preparation, is likely typical of that which would be performed in a mobile environment. It should be noted that the study examines only short-term file activity, during which time only a single task is being worked on at the client. Table 1 shows the characteristics of the traces. The traces each represent over an hour of activity, and contain just under one thousand write events.

A separate simulation was performed for each delay length (i.e. the amount of time between a write event occurring and the client sending the write to a file server) considered. Delays ranging from one millisecond to just over 15 minutes were simulated. For a given delay length, the simulator determined how many write backs would have actually be sent had the work been performed on a weakly connected client. To do this, the trace is read event by event, with all non-write and remove events being ignored. Upon reading a write event, the simulator checks if this is the first write to the file. If this is the case, the file is added to a list, which stores a file identifier and the time of the write. If previous writes have occurred to the file, the time of the last write to the file is examined.

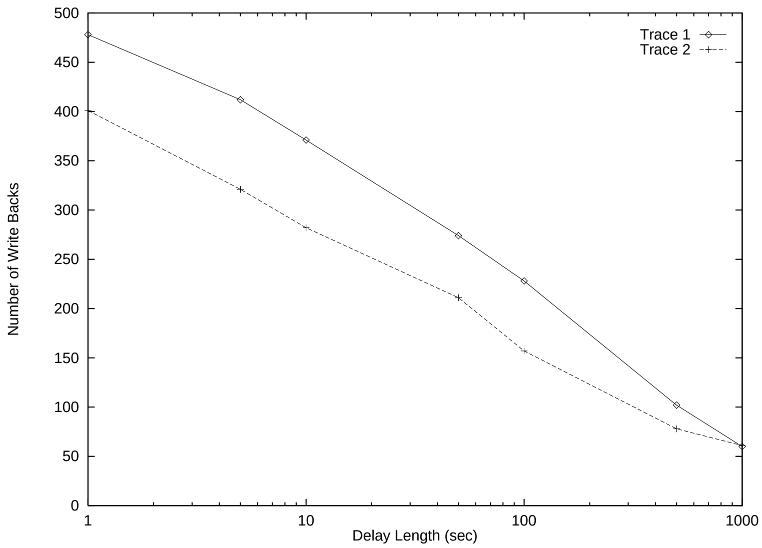

Figure 2: Effect of delay length on number of write backs

If more time has passed than the delay length, then the last write would have been written back, and the write back count is incremented. If the new write is occuring during the time the old write is being delayed, no write back of the old write will occur. The time of the current write then replaces the time of the last write in the file list, and the simulation continues. At the end of the trace, the simulator examines the entire file list, and increments the write back counter for every file which was last written to a length of time ago greater than the delay length.

When a delay length of zero is used, non-delayed write backs are simulated. This means that all updates will be written back. Larger delay times will usually result in fewer write backs. Note that even with very large delay times, there will always be at least one write back for each unique file which is modified.

5.2 Results

The results of the experiments are shown in Figure 2 (note the log scale on the time axis). As the delay length is increased, the number of write backs decreases roughly logarithmically. With delay times as short as 30 seconds, the number of write backs is reduced by roughly 35% to 45% (as compared to not delaying write backs at all). This range seems to fit with other studies which have found that between 65% and 80% of files are destroyed within 30 seconds of being created [1]. Since both short-lived files and more permanent files are measured in this study, it is reasonable to expect that the number of overwrites which occur would be somewhat less than in [1]. When the delay time is 15 minutes, about 85% of writes are avoided. This is higher than has been found in other studies, which reported that between 50% and 75% of write backs could be avoided when delaying writes for an entire day [13]. However, these lower values came from studying traces which represented several days of activity. Since people tend to work on several different sets of files during time periods such as these [8], it is to be expected that the amount of file reuse would be smaller than in this study.

The experiments show that the number of write backs sent by a client can be greatly reduced by implementing a delayed write back policy. However, it is important to note that as delay lengths become very large, the corresponding reduction in the number of write backs becomes small. This means that it is important to balance the increased chance of a write-write conflict occurring with the benefits of delaying write backs.

This result suggests that a write back scheme may be feasible in a weakly connected environment, and that futher, more detailed study would be worthwhile.

6 Extensions

While the foundations of file system support for weakly connected operation have been laid through the adaptation of disconnected operation principles, there is still work to be done to make better use of write backs in weakly connected environments.

6.1 Performance Studies

It has been noted that performance studies to determine the parameters for an efficient delayed write policy need to be done [13]. While initial studies have already indicated that delaying write backs can greatly reduce bandwidth requirements, it remains to be seen at what point a good balance exists between these benefits and the increased probability of write-write conflicts occurring. This is a difficult task for several reasons. First, since there are few, if any, large-scale mobile systems to trace file activity on, studies must almost always be based upon strongly connected file activity. It remains to be seen if the results of these studies will hold in a truly (weakly connected) mobile environment. As well, the rate at which write-write conflicts occur appears to be highly dependent upon the type of work being performed [8]. Finally, the severity of conflicts depends on the amount of work required to resolve them. If current work on application specific resolvers is largely successful, it may be more desirable to set delay lengths quite long.

6.2 Write Notifications

A possible extension to a write back scheme is the addition of *write notifications*. That is, when a file is modified at a weakly connected client, the client could notify the server of the write without actually sending any information on the modification itself. Write backs would still be sent, only at a later time. Such notifications should require very little bandwidth. It would likely be beneficial to delay notifications in the same manner as write backs, only with a much shorter delay interval, possibly on the order of five or ten minutes. This would keep the number of notifications sent small while providing file servers with (nearly) up to date information on files which have been modified. This information could in turn be used by a file server to help maintain file consistency. The use of write notifications would help reduce conflicts at the cost of slightly higher bandwidth usage. Whether this is desirable or not would depend on the frequency of conflicts in a particular working environment.

6.3 Dynamic Scheduling

Instead of scheduling write backs to occur after a fixed delay length or at some other fixed time, it may be possible to dynamically schedule write backs according to user file activity. The ideal time to perform write backs is just after all modifications to a file have been completed. This minimizes both the number of write backs require as well as the probability of write-write conflicts occurring. It may be possible to estimate this condition by using a metric such as *attention-shifts* [8], which examines the set of files referenced by a user over time. Such a scheduling technique could prove to be very valuable in a weakly connected environment.

6.4 User Involvement

Since it is the rate of write-write conflicts which will likely determine how long write backs should be delayed in a weakly connected environment, it may be desirable to receive input from the user to help with setting this interval. If user at a weakly connected client knows that a file being modified is likely to be referenced by another user, he or she could flag the file so that the file system would initiate write backs more quickly for the file in question. Such a technique could be useful for sets of files which are commonly shared among many users, such as shared databases and mailboxes. The advantage of such a scheme would be that the number of conflicts would likely be reduced, without a large increase in bandwidth usage (since most write backs would still be delayed for a significant length of time).

However, one of the goals of file system support for weakly connected operation is to present the user with the same environment while weakly connected as when strongly connected. Relying on users to provide file usage information when weakly connected violates this principle. It may also be the case that the user is not aware of which files are likely to be shared. It is likely that the type of work being performed in the user's environment (e.g. business management, programming, etc.) would have a large effect on wether such a scheme would be practical or not.

7 Conclusions

The adaptation of file system support techniques for disconnected operation should provide a solid base upon which to build support for weakly connected environments. While there have been many proposals for how file systems should support weakly connected operation, it is difficult to determine the relative merits of these proposals due to a lack of experimental systems upon which studies can be performed. This is likely to change in the near future as mobile computing becomes more of a reality.

The use of delayed write backs in weakly connected operation appears to be an effective way to help maintain file consistency while still conserving bandwidth. The greatest need remaining in this area is studies to determine how long write backs should be delayed in order to provide for both file consistency and efficient operation at the client.

File system support for weakly connected operation should provide a means to make mobile computing both efficient and practical in the near future.

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, 1991).

[2] Chikarmane, V. Disconnected operation in a distributed system. In *Proc. Sixth Annual Graduate Symposium on Computer Science* (Saskatoon, SK, 1994), pp. 23–32.

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

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

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

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

[7] Kuenning, G. The design of the seer predictive caching system. In *Workshop on Mobile Computing Systems and Applications* (Santa Cruz, NM, 1994).

[8] Kuenning, G., Popek, G., and Reiher, P. An analysis of trace data for predictive file caching in mobile computing. Technical report, University of California, Los Angeles, CA, 1994.

[9] Kumar, P., and Satyanarayanan, M. Supporting application-specific resolution in an optimistically replicated file system. In *Proc. Fourth IEEE Workshop on Workstation Operating Systems* (Napa, CA, 1993), pp. 66–70.

[10] Marsh, B., Douglis, F., and Cáceres, R. System issues in mobile computing. Technical report, Matsushita Information Technology Laboratory, Princeton, NJ, 1993.

[11] Mazer, M., and Tardo, J. A client-side-only approach to disconnected file access. In *Workshop on Mobile Computing Systems and Applications* (Santa Cruz, NM, 1994).

[12] Mirghafori, N., and Fontaine, A. A design for file access in a mobile environment. In *Workshop on Mobile Computing Systems and Applications* (Santa Cruz, NM, 1994).

[13] Muntz, Honeyman, and Antonelli. Performance implications of delayed-write policy in a multilevel file system cache. Technical report, Center for Information Technology Integration, University of Michigan, Ann Arbor, MI.

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