# Scheduling Write Backs for Weakly-Connected Mobile Clients

*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. File access is a fundamental requirement of any computing environment, and mobile users expect their file service needs to be met just like any other user. 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.

One important cache management decision is how (and when) modifications that have been made to cached files at a weakly-connect client can and should be propagated back to the file server. It is desirable to write back any changes made to files as soon as possible in order to reduce the likelihood of update conflicts occurring due to sharing of files under an optimistic replication scheme. Conflicting with the benefits of eagerly performing write backs is the need for reads to have priority for network usage. Both of these factors must be considered when choosing a write-back scheduling policy for a mobile client.

This paper presents the results of a trace-driven simulation study of two write-back scheduling policies: a simple statistical policy and a more explicit "reader preference" policy. Our results indicate that even at low bandwidths, even a simple write-back policy can reintegrate changes to cached files with the home file system in an acceptable period of time, and can do so with very little interference with read traffic. This performance can be achieved with quite reasonably sized caches at the mobile client.

# 1 Introduction

The ability to work from any location has rapidly become a requirement for many users, and systems and network designers must respond to the challenge of supporting mobile computing. Our interest is in providing file system support to mobile users where only a low-bandwidth[1] (or *weak*) network connection to a file server (and other resources provided by a local-area network) is available. The high latency and/or unreliability of a weak network connection means that the mobile client cannot rely exclusively on the network for file system services. File caching at the client appears to offer the best solution for maintaining file system performance in a weakly-connected mobile environment.

The use of client file caching to support mobility originally arose in the context of disconnected operation, where the mobile client has no network connection at all. While operating in disconnected mode the client relies on cached copies of optimistically replicated files, all cache misses are fatal (there is no way to retrieve a requested file which is not present at the client), and reintegration with the home file system cannot occur until reconnection takes place (increasing the likelihood of conflicts when both the cached and permanent copies of a file are updated independently). These issues become less problematic when the mobile client has even a weak network connection. Cache misses can be serviced, albeit slowly, and file updates can be sent back to the file server much more quickly, reducing the chance of update conflicts occurring.

File systems with support for weakly-connected operation have been described by Mummert (Coda) [8] and Huston (AFS) [6]. These systems are based on the same caching techniques which are used to support disconnected operation – whole-file caching of optimistically replicated files. Prior to "going mobile" the client's cache gathers files for use while weakly-connected. (This process is called *hoarding* in the Coda file system [10].) While mobile, any updates made to cached files are stored in a *log file*. The weak network connection is used to retrieve any requested files which are not present in the client's cache, and can also be used to send log file entries back to the file server. (This is called *trickle reintegration* in Coda and *background replay* in AFS.) Regardless of the specific file system in use, client caching is the central component of these systems. Our interest is in the performance implications of design choices for the mobile client file cache – specifically factors such as cache size, cache management policies, and write-back scheduling.

As with much of the current work in mobility, the client caching techniques currently used in systems which support weakly-connected operation have been developed with *functionality* as their primary goal. However, the cache is also the key to achieving acceptable file system *performance*. We feel that users will be willing to sacrifice some performance for the advantages of mobility, but that a large degradation will be unacceptable. If acceptable file system performance is to be achieved while mobile, appropriate caching decisions must be made.

One important cache management decision is how (and when) modifications that have been stored in the client's log file can and should be propagated back to the file server. Two conflicting goals complicate this problem. First, it is desirable to write back the contents of

[1] "Low bandwidth" may arise from a low-technology connection, from network congestion, from distance from the home network, or from a combination of all three.

the log file as soon as possible in order to reduce the likelihood of update conflicts occurring due to sharing of files under an optimistic replication scheme [6, 8]. As well, performing write backs promptly helps reduce the size of the log file, which is desirable for freeing local client resources (disk space), as well as for reducing reintegration time once the client regains a strong connection [2]. Write backs also help protect file modifications from loss due to failure, damage or theft of the mobile client [8].

Conflicting with the benefits of eagerly performing write backs is the need for reads to have priority for network usage. Any time a file must be transfered from a server (in response to a read miss), it is likely that both the application which requested the file and the user will be stalled until the read has been completed. Therefore it is desirable that reads start (and complete) as soon as possible, and performing write backs could easily interfere with this goal, particularly if "whole-file" caching is being used. An advantage of delaying (and thus a disadvantage of eagerly performing) write backs is the fact that it is common for several modifications to be made to a file over a short period of time, and the number of actual I/O operations which need to be performed can be reduced by delaying the write back. Huston and Honeyman found that 70% of operations stored in a log file could be eliminated by delaying write backs [6].

We use trace-driven simulation to investigate the performance of two write-back scheduling policies: a simple statistical policy and a more explicit "reader preference" policy. Our results indicate that even at low bandwidths, even a simple write-back policy, can reintegrate changes to cached files with the home file system in an acceptable period of time, and can do so with very little interference with read traffic. This performance can be achieved with quite reasonably sized caches at the mobile client.

The remainder of this paper is organised as follows. Section 2 describes the experimental setup for this study. Results are presented in Section 3, and conclusions are given Section 4.

# 2 Experimental Setup

The approach taken in this study was to gather detailed traces of real file system activity from users performing "everyday" work, and to use these traces to drive simulations of a mobile client's file cache. We begin with a description of the type of mobile environment envisioned by this work. The techniques used to collect the traces are described next, followed by a discussion of the simulation model. Some notes on the simulation methodology conclude the section.

## 2.1 The Mobile Environment

We envision a style of location-independent computing that is based on current LAN-based client-server distributed systems. Each mobile user is assumed to originate from such a LAN, which is called his/her "home" location. This is where the home 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. Apart from his/her home location (say, the workplace), a user may work from several other locations from time to time, such as his/her place of residence, a hotel room, or another laboratory.

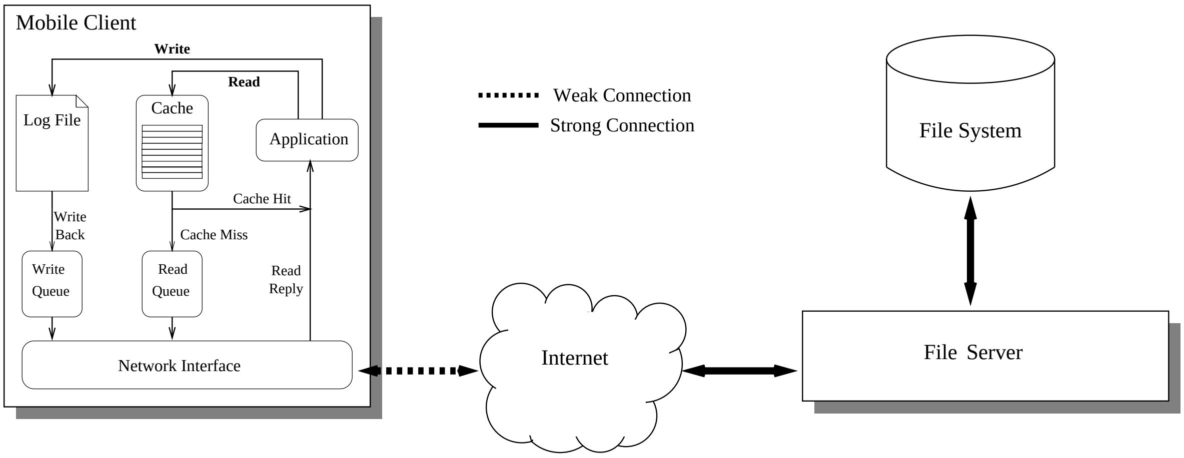


Figure 1: System Model

A key assumption about the environment is that while the user may experience a wide range in available bandwidth between various working locations, the bandwidth at a given location is fairly constant. For example, the user may have access to a 10 Mb/sec Ethernet at the office, but only a 28.8 kb/sec modem when working from home or on a business trip, and may have no network connection at all while in transit.

The type of work being performed is assumed to be that found in a typical academic/research environment: such things as text editing, compiling programs from source code, document preparation and viewing, and information browsing (such as using a World Wide Web client). We focus on this type of workload despite the arguments that this is *not* the type of work which users will likely do in a mobile environment [7]. While this argument has merit, the fact that it is impossible to predict accurately what future workloads from mobile environments will look like makes the use of current workloads a reasonable choice. We feel that stronger results can be obtained by using a workload model based on current data rather than some speculative model of future user behaviour.

Figure 1 shows the system model used in this work. The client file cache manager prepares for going mobile by hoarding (whole) files while the client is strongly connected. Optimistic replication [1] and whole-file caching are assumed. Whenever the client attempts to read from a file, the cache is searched for the file. If the file is present in the cache no network access is needed. If the file is not in the cache, then the read request is placed in the read queue where it waits for access to the network. When the network becomes available, the request is sent to the 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, and is placed in the write queue. The write may later be sent to the file server over the weak connection for file system reintegration. All network traffic passes through a network interface which arbitrates access to the client's network connection.

## 2.2 Collection of Workload Traces

In order to obtain the traces of file system activity, modifications were made to the HP-UX 9.05 operating system kernel to record all `read()` and `write()` system calls. Information

Table 1: System Calls and Parameters Recorded

| System Calls Recorded | Parameters Recorded |
|---|---|
| `read()`<br>`write()` | time, process ID, user ID, error status, vnode pointer, file type, device, inode, file size, file offset, transfer size, file name |

Table 2: Trace Characteristics

| Trace | Active Time (Hours) | Total Requests | Unique Files | Unique Bytes (MB) | Read Transfers (MB) | Write Transfers (MB) | Write Requests (% of total) |
|---|---|---|---|---|---|---|---|
| Trace 1 | 12.9 | 21013 | 383 | 17 | 4596 | 4 | 37.8 |
| Trace 2 | 7.0 | 53860 | 1000 | 18 | 4718 | 35 | 67.9 |

such as the device and inode numbers of files, user ID's, and file sizes was recorded. A complete list of the parameters recorded is given in Table 1. `Unlink()` system calls are not included in these traces. Since inodes are reused by the file system, it is possible for a file to be deleted and a new file created with the same device and inode number as the deleted file. Our simulator would not recognise the new file as being distinct from the original file; but the effect on our results is assumed to be small, since earlier traces gathered with `unlink()` calls showed that fewer than 1% of events were `unlink()` events.

Tracing was done in the Distributed Systems Research Laboratory in the Computer Science Department at the University of Saskatchewan. A series of traces was collected from a set of four HP Series 700 workstations by capturing all file system activity over a seven-day period. This data was broken into seven day-long segments, and were combined into a set of four week-long, continuous traces, each from a single workstation. From these four traces, the activity of single users was isolated and stored in separate trace files. These individual-user traces from each workstation were then combined (maintaining the time stamps on each event), resulting in traces which recorded the file activity of a single user (on all four workstations) over a period of one week. In the interests of space we limit this analysis to two of these week-long single-user traces, the characteristics of which are summarised in Table 2. "Active Time" is the duration of the trace (24 hours times seven days) minus any periods of inactivity greater than 15 minutes in length. "Total Requests" is the sum of the number of read and write requests in the trace. "Unique Files" is the number of files which are either read from or written to in the trace, counting each distinct file only once. "Unique Bytes" is the total size of all unique files seen in the trace. "Read Transfers" and "Write Transfers" are the total amount of data read and written, and "Write Requests" indicates the fraction of requests which are writes. Trace 1 was chosen because it is typical of the traces collected, and has characteristics similar to those of traces used in other studies (such as [1]). Trace 2 was chosen because it has an unusually high level of write activity – roughly two-thirds of all events are writes. This trace is included as a "stress test" of the write-back scheduling policies considered.

## 2.3 Simulation Model

A trace-driven simulator was written to examine performance issues relating to management of a mobile client's file cache. The simulator reads and processes each read/write event in the input trace file sequentially. The client is considered to be strongly connected (with a 10 Mb/sec network link) at first, and prepares for weak connection by operating in a "hoarding" mode for some specified period of simulation time. The client cache (which is of a fixed size) begins empty, but stores copies of all files referenced while in hoarding mode. If there is insufficient space in the cache to store a file, files are removed (according to the replacement policy currently in use) until there is enough space in the cache for the new file. After some period of time the client becomes weakly connected, at which time hoarding activity is halted. The client remains weakly connected for the remainder of the simulation.

When a read event occurs, the client cache is searched for the requested file. If the file is present in the cache (a hit), no action is required. Otherwise, a read miss has occurred and the file is transfered from a file server to the client over the weak connection, and the file is placed in the cache (potentially replacing other files if the cache is full). Since reads are assumed to be blocking events, no other activity occurs at the client.

When a write event occurs, the log file is updated. For the purposes of simulation the log file stores only the content of write operations and a time stamp. After each write the log file is searched for an entry that overlaps the current write. An entry is said to *overlap* if it is from the same file and any portion of the data in the entry is from the same segment of the file as the new entry. All overlapping log file entries are deleted, and the new entry is added to the log file. No simulation time is consumed processing a write. All writes are assumed to be non-blocking.

A write back can be performed at any time, as determined by the write-back policy in use (LRU). Regardless of policy, the oldest entry in the log file is always written back first.

The simulation runs until all trace events have been processed. It is possible (and common) for there to be entries remaining in the log file when the simulation completes.

## 2.4 Simulation Parameters

A series of initial experiments was performed to determine reasonable values for various simulation parameters, which were then fixed for the remainder of the study. The client is considered to be strongly connected for an initial period of 24 hours (the hoarding phase), during which the cache is "heated" in preparation for subsequent mobile activity (as is done in [5]). While mobile, LRU replacement is used regardless of the type of connection and size of cache.

## 2.5 Performance Metrics

As noted in [9], there is a need to develop metrics for effectively evaluating caching strategies for mobile environments, since the constraints imposed by the mobile environment make the resource/performance trade-offs pertaining to caching significantly different from those in a LAN environment. The metrics used in this study aim to characterise important relationships between resource consumption and performance when managing a mobile client's file cache.

One important measure of performance is the amount of time required to service all I/O requests in a reference trace. This quantity consists of two components:

- **read service time** – the amount of time spent transferring files requested by applications that resulted in cache misses
- **write interference time** – the amount of time spent waiting for write backs to complete when the read queue is not empty

For this study, the total (simulation) time required to process all events is reported as a "time expansion" value. This is computed by dividing the total time simulation time by the time required to process the same trace while strongly connected. The resulting ratio provides an indication of the "slowdown" resulting from the weakly connected environment. Write interference time is also of interest since it directly measures the amount by which the decisions of a given write-back policy extend the total time required to service all requests in a trace.

The size of the mobile client's log file is also measured. The maximum size of the log file indicates a minimum level of available disk capacity needed at the client (in addition to that required for the file cache and other local files) to operate in weakly-connected mode for the duration of the trace.

The period of time from when a file is modified until it is written back is of interest since it is during this period that update conflicts can occur or changes can be lost due to some sort of failure. The less time an update spends in the log file, the less chance there is of such problems occurring.

Finally, since the reason for delaying write backs is to reduce the number which need to be performed, the fraction of log file entries which are overwritten by subsequent overlapping write activity is examined. If the frequency of log file overwrites is low, the disadvantages of delaying writes outweigh the advantages.

# 3 The Experiments

## 3.1 Policies

Our study focuses on the problem of accommodating both reads and writes over a low bandwidth connection. Since the demand placed on a mobile client's network connection will often exceed the bandwidth available, it is necessary to implement policies which control access to the network. In this study, there are two sources of network traffic – the read queue and the write queue (see Section 2.1). Requests in the read queue are given access to the network immediately upon the network becoming idle. While weakly connected, requests in the write queue (i.e. write backs) are given access according to one of the following write-back scheduling policies[2]:

[2]Additional polices are considered in [3]

- **No Write Back** – perform no write backs while weakly connected
- **Lotto** – hold a lottery (with the write queue receiving $W$ tickets and the read queue receiving $R$ tickets[3]) after each transfer (read or write) has completed, or if the network has been idle for some fixed period $I$ (set to 0.1 seconds for this study[4]); write back the oldest log file entry in its entirety whenever the write queue wins a lottery
- **Delayed Write Back with Preemption** (DWB-P) – begin writing back the oldest log file entry if it has been in the log file for at least some fixed aging period $A$ (set to 15 minutes for this study) and if the network is idle; if a read request occurs before the write back has completed, halt the write back and resume from the same point when the network becomes idle

Each of these policies attempts to perform writes whenever possible while giving priority to read traffic. No Write Back is included simply to provide a basis for comparison. (At the other extreme is the Write Through policy, in which writes are performed immediately with the application blocking until the write has been completed. We do not examine Write Through, since its appropriateness is doubtful even in a strongly-connected environment, and it is even more poorly suited for a mobile client.) Delayed Write Back with Preemption gives priority to reads by performing writes only when the network is otherwise idle. A write back in progress will be halted if a read request is received, and will be resumed from the same point at a later time. Lotto is based on the policy described in [11] (and used in AFS [6]), with modifications to work on a whole-file (rather than packet) basis. Lotto attempts to detect idleness using fixed timer-based prediction [4].

## 3.2 Results

Figures 2 and 3 show the time expansion experienced for each write-back policy for client cache sizes of 10 MB and 20 MB respectively. Very little difference in performance is seen between the various policies. With Trace 1 there is essentially no difference in performance, and there is only a very small difference with Trace 2. Regardless of write-back policy, with a relatively small client cache of 10 MB, the time expansion values are all small enough (less than 15%) that a user should be able to work effectively even with a very low-bandwidth connection of 9.6 kb/sec[5]. Increasing the cache size to 20 MB reduces the upper bound on time expansion to 5%.

Results for all remaining experiments are presented only for a client cache size of 20 MB, a size that is not unreasonable by today's standards.

The fraction of total run time attributable to write interference time for the Lotto policy can be seen in Figure 4. As expected, interference time is reduced by increasing bandwidth, but even at low bandwidths very little write interference is observed. By design the Delayed Write Back with Preemption policy does not experience write interference.

[3]The number of read tickets *(R)* is set 11, and the number of write tickets *(W)* is set to 1. These values were used since they maintain the same ratio of tickets for write backs to other traffic used in [6].

[4]Based on preliminary experiments to determine reasonable values.

[5]This represents a "net" (i.e. effective) bandwidth value, ignoring any bandwidth lost to congestion, routing delays, retransmissions, etc.

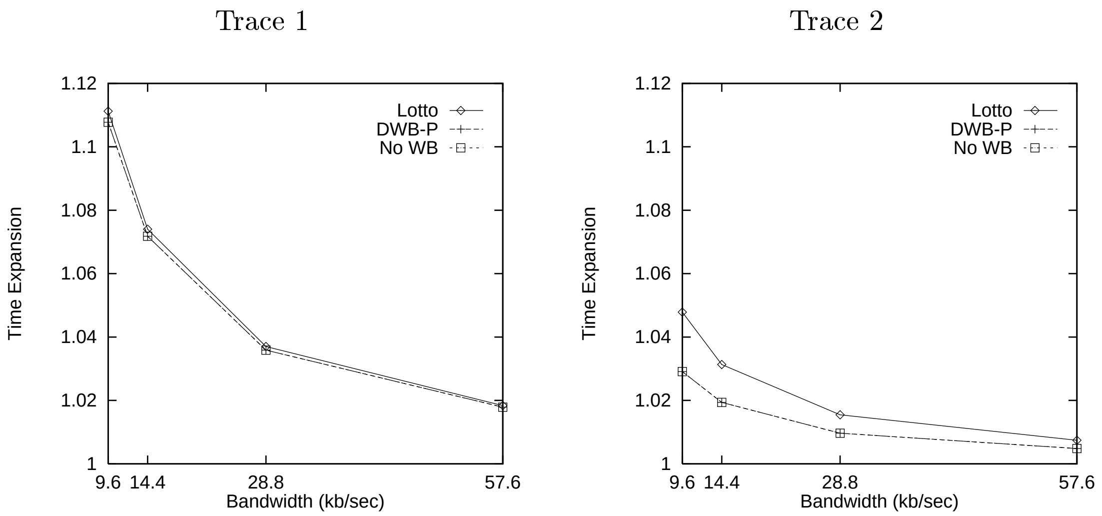


Figure 2: Time Expansion with a 10 MB Cache

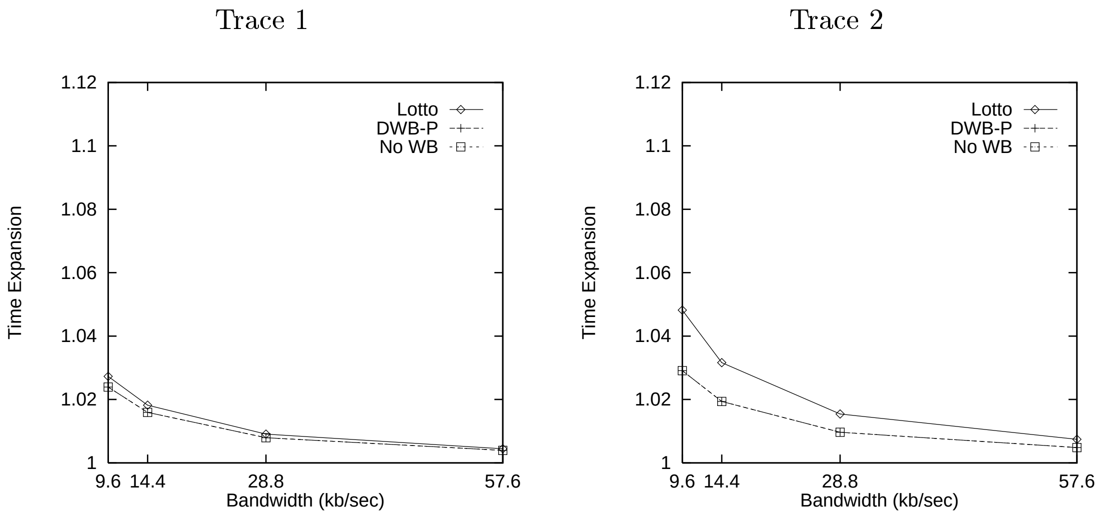


Figure 3: Time Expansion with a 20 MB Cache

Figure 5 demonstrates the cost of write-back policies, as reflected by the maximum size of the log file at the client. Bandwidth has almost no effect on the size of the log file for any of the write-back policies. The log file size of the No Write Back policy is provided to show the extent to which each policy reduces the size of the log file. For both traces, the maximum size of the log file is under 2 MB. This is particularly surprising for Trace 2, given that 35 MB of write activity is present in that trace.

The small log file sizes observed for all policies are due in part to the large number of log file entries which are overwritten while in the log file, as seen in Figure 6. Although the values shown for Trace 2 are in line with those reported in [3], they are much larger than was observed for Trace 1, where approximately 45%-55% of log file entries were overwritten. This is likely due to the higher percentage of write requests present in Trace 2 (see Table 2).

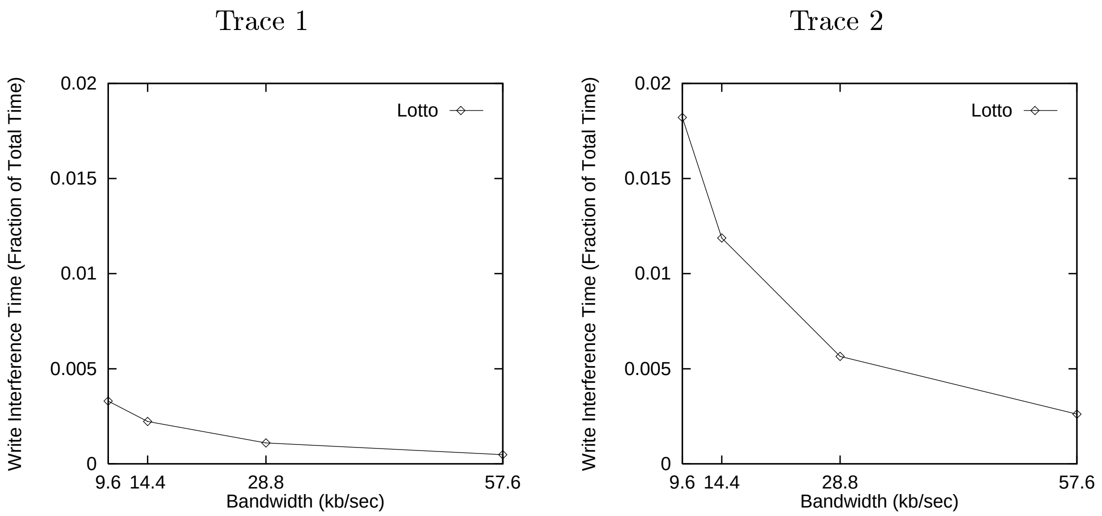


Figure 4: Write Interference Time with the Lotto Policy (20 MB cache)

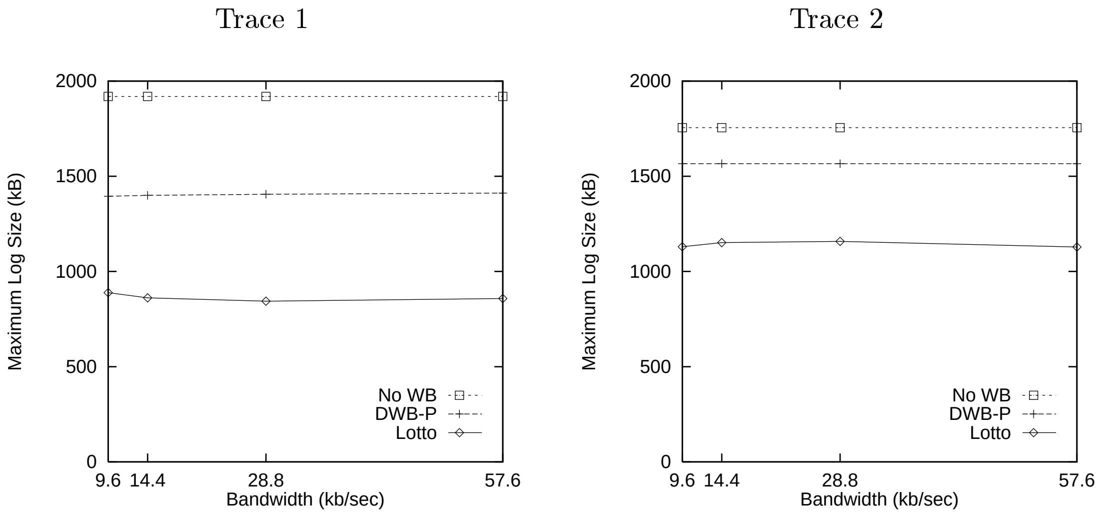


Figure 5: Maximum Log File Size (20 MB cache)

Again, the difference in performance between the various write-back policies is small.

Finally, Figure 7 shows the average period of time log file entries remain in the log file. The time in log file is reduced as bandwidth increases, since it is possible to perform more write backs. This reduction is relatively small in most cases. As is expected, use of a more "aggressive" write back-policy (such as Lotto) results in log file entries being reintegrated more quickly than with a more "conservative" write-back policy (such as Delayed Write Back), with the amount of difference depending on the number of writes which are performed. For Trace 1, the difference between the two policies is small, but in Trace 2, where many more writes are performed, the difference is significant. In all cases, the average time a log file entry remains in the log file is quite short, ranging from 15 to 35 minutes (depending on network bandwidth), even with a conservative write-back policy. We deem this to be

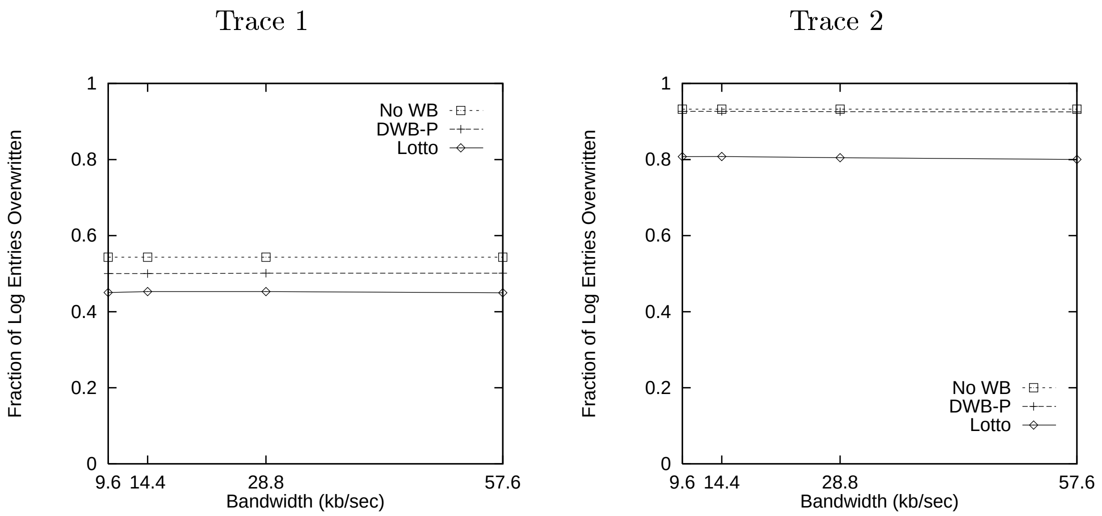


Figure 6: Fraction of Log Entries Overwritten (20 MB cache)

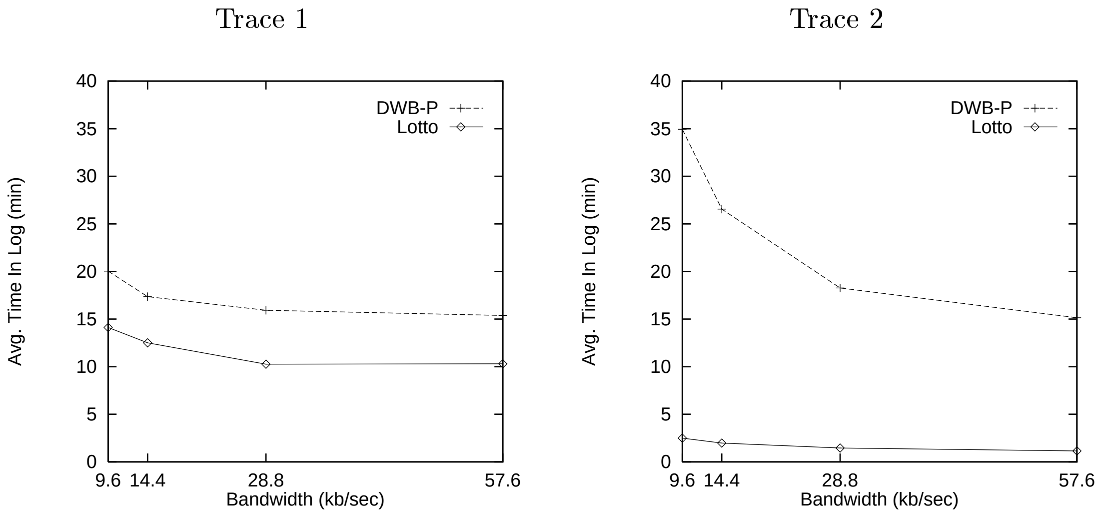


Figure 7: Average Time Entries Remain in Log File (20 MB cache)

an acceptable period of time since it is likely within the length of a user's working session, which means that the majority of changes made to files will likely be propagated to the server before he/she is finished working.

# 4 Conclusions

The goal of this study was to investigate resource/performance trade-offs related to managing a mobile client's file cache in a weakly-connected environment. The effects of write-back scheduling policy on client cache performance were studied in detail.

Our simulation results show that it is possible to provide quite good file system support in a weakly-connected environment, servicing read requests in a timely manner while at the same time providing acceptable write service. Even when bandwidth is low, there is enough unused network capacity that even a conservative write-back policy, such as Delayed Write Back, results in writes being reintegrated with the home file system in an acceptable period of time – less than 30 minutes in most cases. This is accomplished with very little interference with read traffic, increasing the total time required to perform all read operations by less than 5% in most cases. This level of performance can be achieved even at a resource-poor client. For the traces examined in this study, a 20 MB file cache was found to be large enough to support this type of activity. While there are definitely performance benefits to be gained from higher-bandwidth connections, even a 9.6 kb/sec network link proved to be enough to provide acceptable levels of performance.

The performance benefits of delaying write backs were found to be quite substantial. By ensuring that log file entries remain in the log file for even 15 minutes, the number of entries in the log file can be dramatically reduced by overlapping writes, in turn significantly reducing the number of write backs which need to be performed.

Interestingly, the Delayed Write Back with Preemption policy did not perform significantly better than Lotto. This is encouraging since it shows that it is not necessary for write-back policies to be complex in order to achieve acceptable levels of performance.

File access is an important requirement of a mobile computing environment. The presence of a weak network connection provides a mobile client with an opportunity to have the same file access functionality as a strongly connected client. Although such functionality comes with an attendant cost (in both resource needs and performance), our study suggests that these costs should be acceptable, even when simple write-back scheduling policies are used.

# Acknowledgements

This research was funded, in part, by the Natural Sciences and Engineering Research Council of Canada (NSERC), through a Postgraduate Scholarship and Research Grant OGP0003707. We are also grateful for support from Telecommunications Research Laboratories (TR*Labs*), IBM Canada's Centre for Advanced Studies (CAS), and from Hewlett Packard.

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

[3] Froese, K. File cache management for mobile computing. Master's thesis, Dept. of Computer Science, University of Saskatchewan, Saskatoon, SK, 1997.

[4] Golding, R., Bosch, P., Staelin, C., Sullivan, T., and Wilkes, J. Idleness is not sloth. In *Proc. of the USENIX Conf.* (New Orleans, LA, Jan. 1995), pp. 201–212.

[5] 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.

[6] Huston, L., and Honeyman, P. Partially connected operation. In *Proc. Second USENIX Symposium on Mobile and Location-Independent Computing* (Ann Arbor, MI, April 1995), pp. 91–97.

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

[8] 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.

[9] Satyanarayanan, M. Fundamental challenges of mobile computing. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996.

[10] 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.

[11] 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.