Cache Management for Mobile File Service

Kevin W. Froese† and Richard B. Bunt

Department of Computer Science, University of Saskatchewan, Saskatoon, SK, Canada, S7N 5A9
Email: kfroese@opentext.com, bunt@cs.usask.ca

File service is a fundamental computing requirement, but one that has been problematic for mobile users. Concerns about the latency associated with transferring large amounts of data over network connections of unknown quality have led to rather ad hoc approaches that provide some functionality, but with uncertain performance. Through a series of trace-driven simulation experiments, we investigate performance issues relating to providing remote file system support to mobile users through optimistic caching at the mobile client. Tradeoffs between resources and performance are explored across a variety of design choices, specifically issues relating to system configuration, policies for file system updates (write backs), and choice of caching unit (whole-file caching or block-based caching). The results of our experiments show that it is possible to provide quite acceptable remote file service to weakly connected mobile clients, even when bandwidth is limited. Reads can be serviced in a timely manner, updates can be committed in an acceptable period of time, and resource requirements at the client are modest.

1. INTRODUCTION

Improvements in the capabilities and portability of notebook computers and hand-held personal digital assistants (PDAs), coupled with increased user expectations for "anywhere, anytime" connectivity, have given rise to increasing demands for *mobile computing*. In this new paradigm the computer accompanies the user, for use in different locations as he/she travels about. This presents significant challenges for operating system and network designers to meet user expectations of functionality and performance. The goal is to make transparent to the user (or at least to the application program) any changes that might occur in the point and type of network connection, thus allowing the user to work in the same manner and with the same productivity from any location.

Prominent among services mobile clients require, wherever they might be located and however they might be connected, is file service. Access to data is a fundamental system requirement and, even when roaming, users want access to the data that resides on their (home) file server. In the absence of automatic file system support for location-independent operation, mobile users must rely on ad hoc methods to maintain and update separate copies of files manually, at both the mobile client and the file server [1]. More sophisticated "mobility-tolerant" file systems attempt to remove this burden from the user by employing automatic file replication management techniques, central to which is the caching of file replicas at client machines. To reduce the reliance on the network, file system "reads" are read from the cache and file system "writes" are logged for later propogation to the file server upon reconnection [2]. Although this allows some measure of operability while mobile, any attempt to access data not cached (a cache miss) might be problematic (even fatal if the mobile client is currently disconnected). As well, delays in the reintegration of file changes with the home file server (*write backs*) can lead to consistency problems. Support for disconnected and weakly connected operation has been studied by researchers at Carnegie Mellon University [3] and the University of Michigan [4], who have shown that even a weak connection can permit enough processing of read misses and write backs to maintain operability while mobile.

Adapting file system support so that it can be made available to mobile clients across the full range of connection possibilities is the overall goal of this

†Presently at Open Text in Waterloo, Ontario, Canada

research, and effective cache management is a fundamental element of any solution. Although file caching has been extensively studied, its application to the mobile environment has not. As Banerji [5] has noted,

> Caching issues are beginning to predominate the open research topics in [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.

Not only is the cache the key to achieving file system *functionality*, it is also the key to achieving acceptable *performance*, an issue that has not been adequately addressed. This paper examines some of the design issues relating to file caching at mobile clients, and reports results from trace-driven simulation experiments investigating the performance/resource tradeoffs associated with factors such as cache size, bandwidth, write-back policy, and caching unit. The results show that quite acceptable file service can be provided to mobile clients, even with modest sized client caches and low bandwidth connections.

The needs of the mobile user will determine the type and quality of services that "mobility-aware" operating systems and networks will need to provide. We envision several key requirements of the mobile user that will influence the design of support for mobile operation. First, the mobile user will require access to a common set of resources and services regardless of his or her working location. Second, mobility must be seamless; this means that 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 *some* performance for mobility (but only some). Although it will not be necessary to satisfy the requests of a mobile user as promptly as those of a user in a fully connected LAN environment for example, performance remains an important concern.

The rest of this paper is organized as follows. Section 2 deals with the different types of connectivity that a mobile client may experience. Section 3 focuses on issues relating to cache management. The elements of the experimental methodology are described in Section 4, and the simulation results are presented in Section 5. Section 6 summarizes the paper and presents conclusions from the study.

2. NETWORK CONNECTIONS FOR MOBILE COMPUTING

We envision a style of location-independent computing that derives from LAN-based client-server distributed systems. Each user is assumed to originate from such a LAN (called his/her *home* location), 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 the home location (say, the workplace), a user may work from several other locations from time to time, such as a place of residence, a hotel room, or an airport, where the type and quality of network connection will vary. At various points in time the mobile client may be *strongly connected* (connected to the file server over a fast and reliable link), *disconnected* (no network connection to the file server), or *weakly connected* (connected to the file server over a slow and possibly error-prone or expensive link [3], such as a modem connection or radio link). For our experiments we assume that only one type of connection will be experienced during any given working session, although this is not a requirement of the approaches we consider. While strongly connected, the mobile client can operate as it would in a traditional LAN environment[1]. The client can also use periods of strong connection to prepare for future periods of weak connection or disconnection (this is discussed in greater detail later). While operating in a disconnected mode, the mobile client must rely totally on its file cache to satisfy all its file system requests (reads and writes). The client can perform any operation on cached files that could be performed while connected, but any attempt to access a file not in its cache (i.e. a cache miss) is, of course, a fatal error. 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 (called *hoarding*) have been advocated, including attempting to keep a user-defined set of files in the cache [7], and applying long-term predictive caching techniques [8, 9]. For weakly connected clients some access to the home file server is available over the network, but the connection bandwidth is a limiting resource to be managed carefully. While weakly connected, the client uses the network only when absolutely necessary,

[1] We assume that routing support is provided by lower level network services (e.g. Mobile IP [6]).

for miss processing or periodic write backs to maintain some level of file system consistency.

Issues relating to disconnected and weakly connected operation are discussed in greater detail in the following subsections.

2.1. Disconnected Operation

Mobile clients are forced to operate in disconnected mode when no network connection is available or when there are problems with the connection. This mode has been examined in previous studies [2, 10, 7] and file management techniques based on optimistic caching of whole-file replicas at the client have been proposed. Prior to going mobile, the client's cache gathers files for use while disconnected. This process is called *hoarding* in the Coda file system [2]. While disconnected, all read operations must be processed in cache (misses are fatal errors that probably require termination of the application) and all write operations are logged. Upon reconnection with the home network, the log file is replayed, which has the effect of propagating to the home file server all file system changes that were made while disconnected. In this way, the final state of the file system is the same as if the changes were made while connected, although considerable time may have elapsed since the changes were actually made. Log files can be optimised to save disk space at the client and to decrease the time needed for reintegration [4, 7]. This optimisation is accomplished through recognition of which operations actually affect the final state of the file system. For example, if the same part of a file is updated more than once, only the last update needs to be kept in the log, since the previous changes are "overwritten" by later changes. We call these "overlapping writes".

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 [11]. Some studies have shown that update conflicts occur only rarely [2, 12, 7], and that optimistic replication is an acceptable strategy for a disconnected environment.

2.2. Weakly Connected Operation

Weakly connected operation (also called "partially connected" [13]) 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 the home file server over the network, 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 network to reduce the likelihood of update conflicts occurring. This might be of great benefit in environments where file sharing is common. Techniques for supporting weakly connected operation have been studied by [14, 3, 15]. These are based on the same caching techniques used to support disconnected operation – whole-file caching of optimistically replicated files and logs of changes. As with with disconnected operation, files are hoarded prior to going mobile, and any updates made to files while mobile are logged. The weak connection is used for two purposes: to retrieve any requested files that are not cached (misses) and to send log entries back to the file server when appropriate. This is called *trickle reintegration* in Coda [2] and *background replay* in AFS [10].

3. ISSUES IN CLIENT CACHE MANAGEMENT

File caching in strongly connected environments is well understood, having been used in distributed systems for many years [16, 17, 18], but the use of file caching to permit disconnected or weakly connected operation is a much more recent innovation. While many file caching issues have been addressed, current systems still have limitations. Ultimately, good management strategies for mobile computing will be those that provide transparent operability across all connection environments, with acceptable performance. Effective cache management is the key to both functionality and performance.

In this paper, our focus is on the resource/performance tradeoffs that exist with respect to mobile client file cache management. In this context we examine configuration parameters, caching policies, policies for scheduling write backs, and choice of caching unit.

- **Configuration Parameters** – Two complementary issues are considered here: the amount of cache space at the client and the transmission capacity of the network connection. These are complementary in the sense that one can be traded off for the other: the ability to hold more items in the cache reduces the need to fetch them over the network, and reduces delay in the application to process misses; on the other hand, a faster connection means that items can be fetched more quickly, and so there

is less need to cache as many of them.
The amount of cache space at the client is an important consideration when disk space is limited, as it may be in a laptop, notebook, or palmtop computer. What is allocated to cache is, of course, unavailable for general use.
The capacity of the network connection is determined by a variety of factors, most of which are beyond the user's control. For this study we specify capacity in terms of bandwidth, but we mean more by "bandwidth" than simply the rated capacity of the immediate link to the network. "Low bandwidth" may arise from a low-technology connection, from congestion on the network, from distance from the home network, or from a combination of all three. We do not differentiate among the causes in this first-order investigation.

- **Caching Policies** – There are two basic issues of interest here: the fetch policy that determines which items are loaded into the cache (and when), and the replacement policy that determines which items they will replace, both in regular operation and in the hoarding period. Items can be fetched on demand or prefetched (with or without explicit user advice) in anticipation of use, and can replace previously cached items according to their expectation of future need. Different approaches to all of these come with their own attendant benefits and drawbacks.
- **Write-Back Scheduling** – An important issue is how (and when) file 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 write-back scheduling. First, it is desirable to write back the contents of 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 [13, 3]. As well, performing write backs promptly helps reduce the size of the log file, freeing local client resources (disk space), and the reintegration time. Timely write backs also help protect file modifications from loss due to failure, damage, or theft of the mobile client [3].
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 important that reads start (and complete) as soon as possible. Performing write backs can 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 (overlapping writes), and the number of actual I/O operations which need to be performed can be reduced substantially by delaying the write back. Huston and Honeyman [13] found that as many as 70% of operations stored in a log file could be eliminated by delaying write backs.
We investigate the performance implications and resource requirements of several write-back scheduling policies, including both conservative and aggressive approaches.
- **Choice of Caching Unit** – When a mobile client operates in disconnected mode, whole-file caching is the only practical approach – having only part of a file cached is likely of little value if the rest of the file cannot be accessed. This is not necessarily true when weakly connected, however. In this respect, a weakly connected environment is similar to a strongly connected environment. With even a low-bandwidth network connection, block caching is an option.
Two features of block caching make it an attractive option for weakly connected operation. First, block transfers can complete more quickly than file transfers. The less time a transfer needs to complete, the sooner the user can resume working. This could be particularly advantageous at low bandwidths where file transfers are very time consuming. Additionally, the use of blocks as the caching unit means that it is not necessary to transfer the entire contents of a file to the client if the whole file is not referenced. Again this could be very useful in a low-bandwidth situation, since it allows more bandwidth to remain free for other read or write-back activity.
We compare the performance of block and whole-file caching, for a range of bandwidths.

4. EXPERIMENTAL METHODOLOGY

If mobility is to be truely seamless then the style and type of work performed must be unaffected by location and by type of connection. Consequently, the approach taken in this study was to gather detailed traces of read and write activity from real users performing "everyday" work in a strongly connected LAN environment, and then to use these traces to drive simulations of a mobile client's file cache, with the goal of measuring the performance impact of mobility on normal file system activity. This section begins with a description of the type of mobile environment envisioned. The techniques used to collect the traces are described next, followed by a discussion of the simulation model.

4.1. The Mobile Environment

The type of work done at the mobile client 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). Despite arguments [1] that this is *not* the type of work that mobile users will do, we focus on this type of workload for two reasons, one conceptual and one practical. First, we contend that it is impossible to predict accurately what future workloads from mobile environments will look like. We believe that stronger results are obtained using a workload model based on current data rather than one based on some speculative model of future user behaviour. Second, this is the type of environment in which we work, and so this is the type of workload to which we have ready access.

Figure 1 shows the system model. The mobile client connects to the home file server over the Internet. Prior to leaving home, the client file cache manager prepares for going mobile by hoarding information (either whole files or file blocks) while the client is strongly connected. While mobile, the cache is the key to file system functionality and performance. Whenever the client attempts to read from a file, the cache is searched for the information requested. If the information requested is present in the cache no network access is needed; if not, 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 information requested 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. We consider only weakly connected operation[2] in this investigation. Issues relating to disconnected and weakly connected operation are discussed in greater detail in the following subsections.

[2] Although a weakly connected client may become disconnected temporarily because of a network problem, it is still considered to be operating in a weakly connected environment if network access is available at least part of the time.

4.2. Collection of Workload Traces

Traces of real file system activity from real users were collected in the Distributed Systems Performance Laboratory in the Computer Science Department at the University of Saskatchewan over a period of one week. To obtain the traces, modifications were made to the HP-UX 9.05 operating system kernel to record all `read()` and `write()` system calls. Information 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 (for file deletion) are not included in these particular 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 does not recognise the new file as being distinct from the original file, but the error introduced should be small since traces we gathered with `unlink()` calls showed that fewer than 1% of events were `unlink()` events.

Tracing was done simultaneously on a set of four HP Series 700 workstations over a seven-day period. To accommodate storage limitations, the traces were captured as seven day-long segments, each of which was moved to off-line storage at the end of the day. After all traces were collected, the day-long trace segments were combined into a set of four week-long, continuous traces, each from a single workstation. From these 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 system activity of a single user (on all four workstations) over a period of one week. This paper presents results for two of these week-long single-user traces, some characteristics of which are summarised in Table 2. (Note that "active time" is the total length of the trace minus any periods of inactivity greater than 15 minutes in length.) These particular traces were selected for this paper for the following reasons. Trace 1 is typical of work common in our environment and displays characteristics similar to traces used in other studies (for example, the amount of file system activity and the ratio of reads to writes). Trace 2 has much more write activity and thus serves as a "stress test" for the cache update techniques. Re-

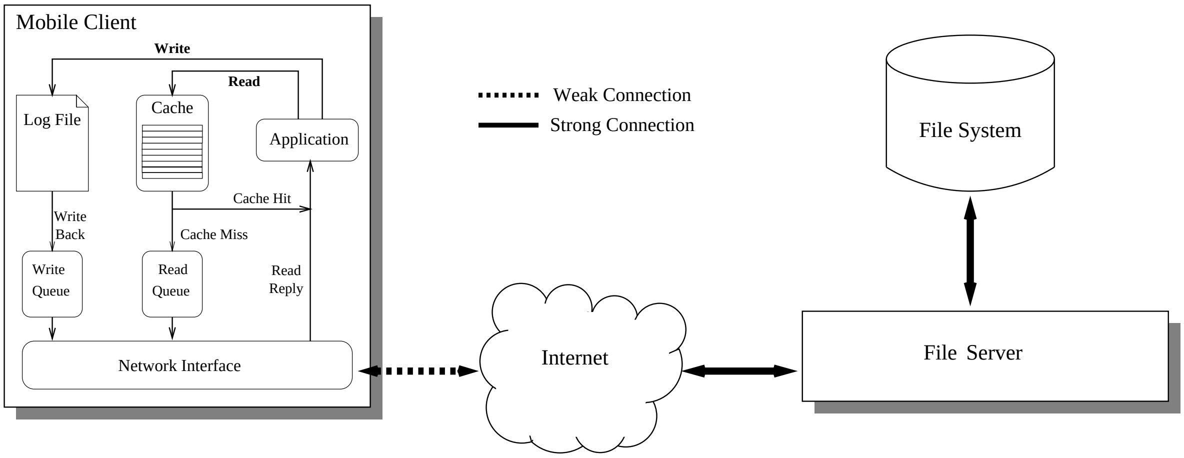

FIGURE 1. System Model

TABLE 1. System Calls and Parameters Recorded

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

sults from the complete set of traces are available in [19].

4.3. Simulation Model

A trace-driven simulator was written to examine performance issues related to managing a mobile client's file cache. The simulator is based on the system model in Figure 1. The client is considered to be strongly connected to the file server at first (with a 10 Mb/sec network link), and prepares for weak connection operating in hoarding mode for some specified period of simulation time (24 hours in the experiments reported here). The client cache (which is of a fixed size) begins empty, but stores copies of all files[3] 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 in use) until there is enough space in the cache for the new file. Once the client becomes weakly connected the hoarding activity is halted. The client remains weakly connected for the remainder of the simulation.

Read/write events in the input trace file are read and processed sequentially. 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 the file server to the client over the network, 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 to the file server can be performed at any time, as determined by the write-back policy in use. Regardless of policy, the oldest entry in the log file is always written back first to ensure timely reintegration.

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

[3] Whole-file caching is assumed for this discussion. Block caching is considered later.

TABLE 1. 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

4.4. Performance Metrics

As noted in [20], we need new metrics to evaluate caching strategies for mobile environments since the constraints imposed by mobility alter the traditional resource/performance tradeoffs pertaining to caching. For example, "miss rate" fails to capture the profoundly different implications of a miss to an 8 kilobyte block of data and a miss to a 1 megabyte file – in the time needed to transfer the data, in the impact of loading it into the mobile client's cache, and in the subsequent write backs that this may induce. The metrics used in this study aim to characterise important relationships between resource consumption and performance when managing a mobile client's file cache.

For this study, we focus primarily on the amount of time required to service all I/O requests in a reference trace. This is determined by several factors, some of them related to configuration parameters, some of them related to management strategies. Our primary performance metric is *time expansion*, which is computed by dividing the total (simulation) time required to process all I/O events in a trace in the mobile environment by the time required to process the same trace while strongly connected. The resulting ratio provides an indication of the "slowdown" resulting from the nature of the mobile connection, and thus the performance impact of mobility. Time expansion is affected by several factors: the number of cache misses, the amount of time spent transferring files requested by applications that resulted in cache misses (*read service time*), and the amount of time read service is suspended waiting for write backs to complete (*write interference time*). Write interference time is of particular interest since it directly measures the extent to 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 also of interest since it is during this period that update conflicts can occur or changes can be lost because of some sort of failure. The less time an update spends in the log file, the less chance there is of such problems arising.

4.5. Caching Policies

Although they are not variables in these experiments, policy decisions in several areas impact our performance results. Throughout the experiments, all items are fetched on demand, and the Least Recently Used (LRU) policy is the basis for all cache replacement decisions[4]. LRU is widely accepted and well-understood, and preliminary experiments showed it to perform well in this application [19]. A decision was also made to fix both the hoarding policy and the hoarding period. It is clear that without successful hoarding, disconnected operation is impossible and weakly connected operation is likely to be very frustrating. A range of hoarding policies were examined in our preliminary experiments [19]; LRU was found to work well in general, and a hoarding period of 24 hours was found to provide an adequate basis for the week-long traces considered.

4.6. Write-Back Policies

Since the demand placed on a mobile client's network connection by file system requests may exceed the bandwidth available, it is necessary to implement policies which control access to the network. In this study, there are two competing sources of network traffic: the read queue and the write queue (see Section 4.1 and Figure 1). Requests in the read queue are given access to the network immediately upon the network becoming idle because application stall time must be minimized. While weakly

[4] Other replacement policies, including Least Frequently Used (LFU), Random, and several size-based policies were examined (see [19] for details). LFU performed well in some cases, and may work even better with longer hoarding periods, but LRU was the most consistent at reducing cache misses.

connected, requests in the write queue (i.e. write backs) are given access according to the write-back policy in effect.

There are many possible policies for scheduling write backs – some of them more aggressive in seeking reintegration, some more conservative. In our terminology *aggressive* policies focus on performing more frequent write backs, with secondary consideration given to the possibility of degrading read performance, while *conservative* policies attempt to avoid interfering with reads, and perform fewer write backs as a result. We consider the following representative policies in this study:

- **Aggressive Policies:**
  - Idle Write Back (Idle WB): write back the oldest log file entry when the network has been idle for some fixed period I.
  - Lottery (Lotto): hold a lottery (with the write queue receiving W tickets and the read queue receiving R tickets) after each transfer (read or write) has completed, or if the network has been idle for some fixed period I; write back the oldest log file entry whenever the write queue wins a lottery.
- **Conservative Policies:**
  - Delayed Write Back (Delayed WB): write back the oldest log file entry if it has been in the log file for at least some fixed aging period A, and if the network has been idle for some fixed period I.
  - Preemptive Write Back (Preemptive WB): begin writing back the oldest log file entry if it has been in the log file for at least some fixed aging period A, and if the network has been idle for some fixed period I; halt the write back if a read request occurs.
- **Baseline Policy:**
  - No Write Back (No WB): perform no write backs at all while weakly connected.

Each of these policies[5] attempts to perform writes whenever possible while giving priority to read traffic. Idle WB gives priority to reads by performing writes only when the network is otherwise idle[6]. Delayed WB attempts to reap the benefits of delaying write backs by explicitly forcing a delay, while Preemptive WB functions like Delayed WB, but gives even greater preference to reads by halting a write back in progress whenever a read request is received, resuming it from the same point at a later time. Lotto is based on the policy described in [22] (and used in AFS [13]), with modifications to work on a whole-file basis where required.

For this study, we classify the Lotto and Idle WB policies as aggressive polices, and the Delayed WB and Preemptive WB policies as conservative. These classifications arise from the fact that Lotto and Idle WB have fewer restrictions on when a write back can be initiated, and therefore more write backs are performed. Delayed WB and Preemptive WB are by comparison conservative, since they require that more conditions be met before a write back is performed.

Several parameters govern the behaviour of our write-back policies, and preliminary experiments were done to determine reasonable values. Results presented here are for idle time I=0.1 seconds and aging time A=15 minutes. For the Lotto policy, ticket ratios were set to match those used in [13]: the number of read tickets *(R)* was 11, and the number of write tickets *(W)* was 1. The "aggressiveness" of Lotto is easily modified by changing the ratio of read and write tickets.

4.7. Caching Unit

When operating in a disconnected environment, whole-file caching is the only practical approach; having only part of a file cached is likely of little value if the rest of the file cannot be accessed. This is not necessarily true in a weakly connected environment, however. In this respect, a weakly connected environment is similar to a strongly connected one. We assume whole-file caching (initially) since it has been successfully employed in systems such as Coda [2, 3, 7].

Two features of block caching make it an attractive prospect in weakly connected environments. First, since block transfers can complete more quickly than file transfers, the time to process misses is reduced. This is appealing in low-bandwidth environments. Second, blocks are more easily accommodated within the client caches than are whole files; the fact that blocks are identically

[5] We include the No WB policy, in which no updates are done while mobile, simply to provide a basis for comparison. It provides an upper bound on resources needed at the mobile client and a lower bound on contention for the network. At the opposite extreme is the Write Through policy, in which updates are performed immediately, with the application blocking until the update has been completed. We do not examine Write Through, since its appropriateness is questionable even in a strongly connected environment, and it is even more poorly suited for a mobile client.

[6] Idleness is detected in all policies using fixed timer-based prediction [21].

sized makes for easier placement, and the fact that they are much smaller eases the demand for cache space at the client[7]. On the other hand, the frequency of misses is almost certain to increase, since repeated accesses to the same file may well refer to different blocks, some of which will not be cached. In a low-bandwidth environment any added use of the connection must be considered very carefully.

A set of experiments was performed to investigate the feasibility of block caching in a weakly connected environment. The experiments use the same format as is employed for file caching: a 24-hour hoarding period followed by 6 days of weakly connected operation. In preparing for going mobile, hoarding is done on a whole-file basis, as in the other experiments, but block caching is used once the client becomes weakly connected.

5. SIMULATION RESULTS

We present our results under categories identified in Section 3: in Section 5.1 we examine the resource/performance tradeoffs associated with fundamental configuration parameters, in Section 5.2 we consider the performance impact of policies for write-back scheduling, and in Section 5.3 we examine the feasibility of block caching with weakly connected mobile clients.

5.1. Configuration Parameters

The first two factors considered are the basic configuration parameters: the size of the client's file cache and the bandwidth of the client's network connection. Increasing either of these resources will (up to a point) improve file system performance for the client. A larger cache reduces the number of read misses, and more bandwidth means that those misses that do occur can be serviced more quickly. Figure 2 shows exactly this: the time expansion surface[8] is affected by both cache size and bandwidth. The size of the client's cache is particularly important. When the client's cache is small and bandwidth is low there are many read misses, stall time is high, and time expansion is very high. Clearly a 1 MB cache yields quite unacceptable performance across all bandwidths considered for the traces used in this study. There is a clear tradeoff between cache size and bandwidth – increasing either will reduce time expansion, but if either resource is in short supply, acceptable performance (as measured by time expansion) can be still be achieved by increasing the other.

The combinations of cache size and bandwidth which are needed to achieve specific time expansion values are shown in the contour lines in Figure 3. The numbers beside each line are time expansion values for that contour. When the cache is small (less than 10 MB in our simulations), many misses occur, and very high bandwidth is necessary to maintain acceptable expansion times. When the cache is larger (10 MB or more for these traces), fewer misses occur and less bandwidth is needed to achieve the same level of performance. While the actual cache sizes and bandwidths which are needed to keep time expansion low will obviously be workload dependent, the qualitative relationship between cache size and bandwidth seen here should change little.

5.2. Write-Back Policy

The management of file updates also impacts file system performance at a weakly connected mobile client. Again, tradeoffs arise when managing this aspect of a client's file operations. As more (uncommitted) writes are performed, the log file will grow accordingly. As well, the longer the updates remain at the client, the greater is the chance that update conflicts or data loss can occur before reintegration. By writing back the contents of the log file to the server in a timely fashion, the chance of such problems is reduced, and the log file should remain small. At the same time, however, it is important that write backs not steal too much bandwidth from the servicing of read requests (write interference), which would result in further stalling the user's work. The write-back policy employed by the mobile client must attempt to balance these tradeoffs.

The main advantage of an aggressive write-back policy is that the time until updates are reintegrated with the home file server should be relatively small. Figures 4 and 5 show that this is indeed the case. For our traces, the Idle WB policy (the most aggressive policy we examine) can reintegrate file updates in 20 minutes or less (often much less) across a range of cache sizes and bandwidths. The more conservative policies (Delayed WB, Preemptive WB) often take more than five times longer to reintegrate updates (but then provide increased opportunities for overlapping writes).

Another reason to perform write backs aggressively is to keep the log file small. This is particularly important when disk space at the client is

[7] This is precisely the reason block caching is preferred over whole-file caching for strongly connected clients.

[8] The No WB update policy was used for these runs since it creates no write interference.

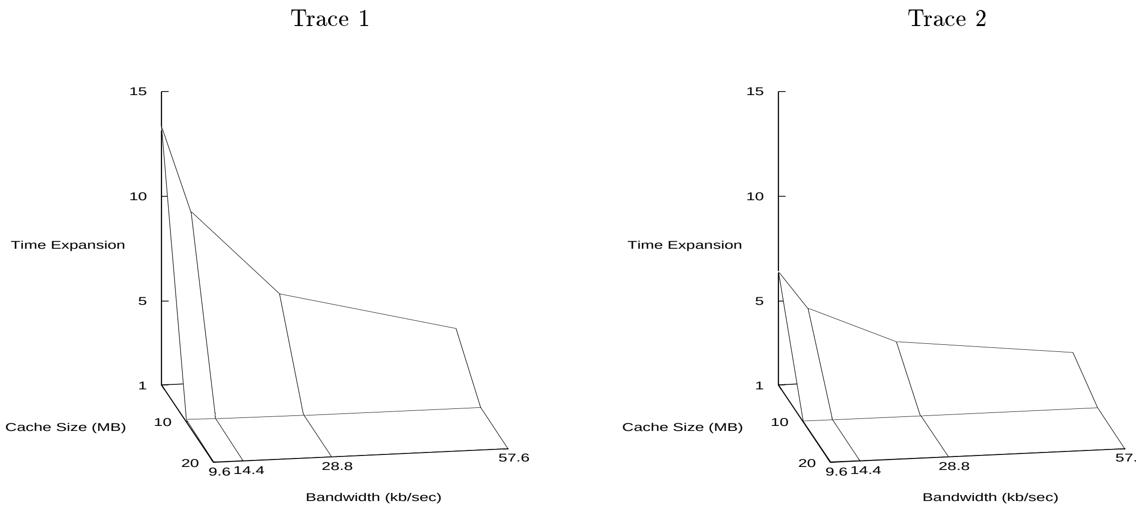

FIGURE 2. The effect of cache size and bandwidth on time expansion (no write backs)

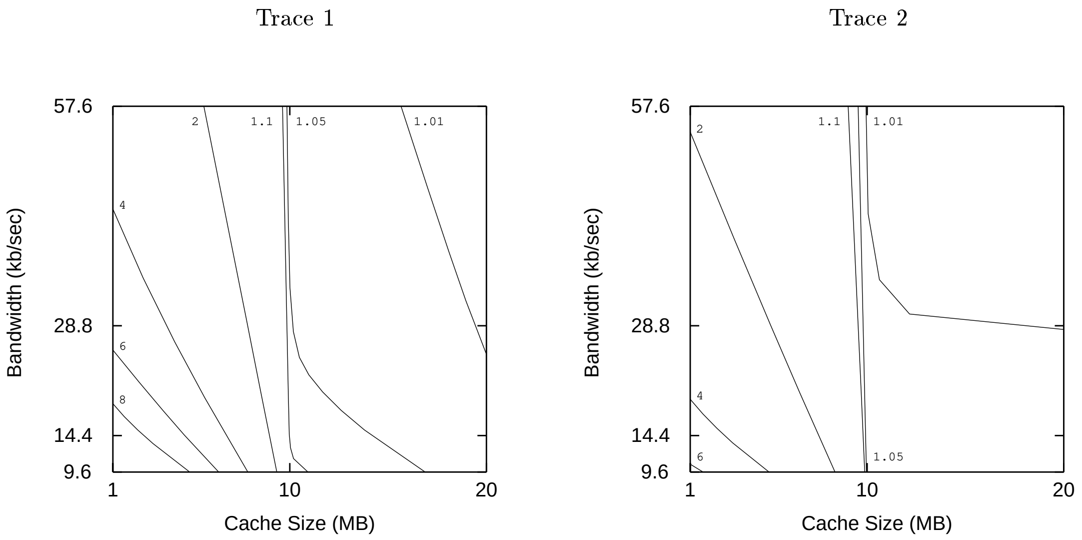

FIGURE 3. Time expansion contours for bandwidth and cache size (no write backs)

limited. Figures 6 and 7 show the maximum size attained by the log file in our simulations. Interestingly, the log file sizes are very similar for both traces presented, even though much more write activity is present in Trace 2. This indicates that the writes overlap much more in Trace 2, keeping the number of log entries small. The ability of aggressive write-back policies to reduce the size of the log file is clear. While the 25% to 50% reduction in log size observed for our traces is not dramatic given that the log never exceeds 2 MB, such a reduction could indeed be valuable under other workloads, especially considering that this saving is consistent across cache sizes and bandwidths.

It is not surprising that write-back policies can be designed that keep both the time until reintegration and the size of the log file small. However, it is less clear whether or not the use of such policies will still allow cache misses to be serviced in a timely manner. Figures 8 and 9 show the fraction of total time attributable to write interference for the various write-back policies. While conservative policies often experience considerably less write interference than aggressive policies, write interference time never exceeded two percent of total time for our traces, even for aggressive write-back policies. Not surprisingly, write interference increases as more writes are performed at the client, as seen by the higher levels of interference for Trace 2 compared to Trace 1. Considering that writes account

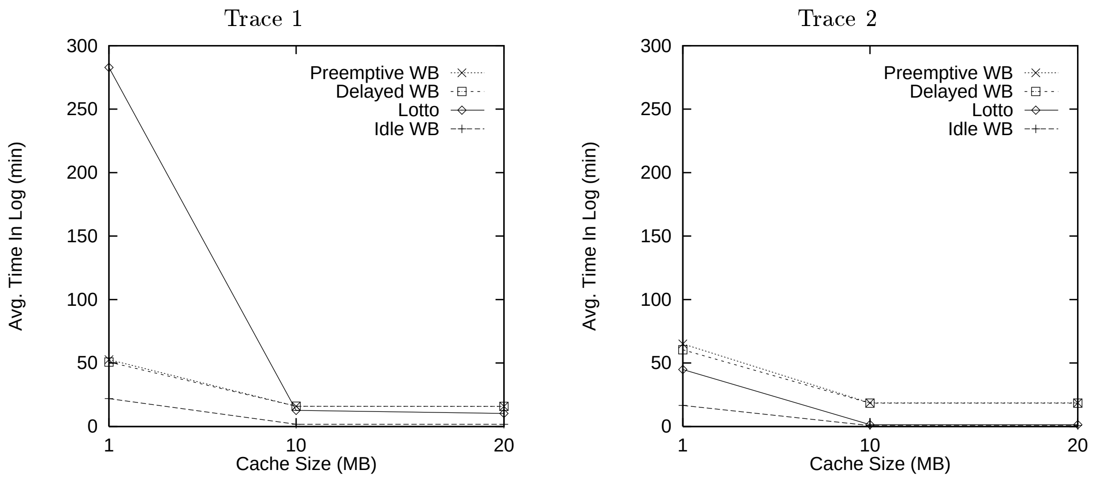

FIGURE 4. The effect of cache size on average time until reintegration (b/w = 28.8 kb/sec)

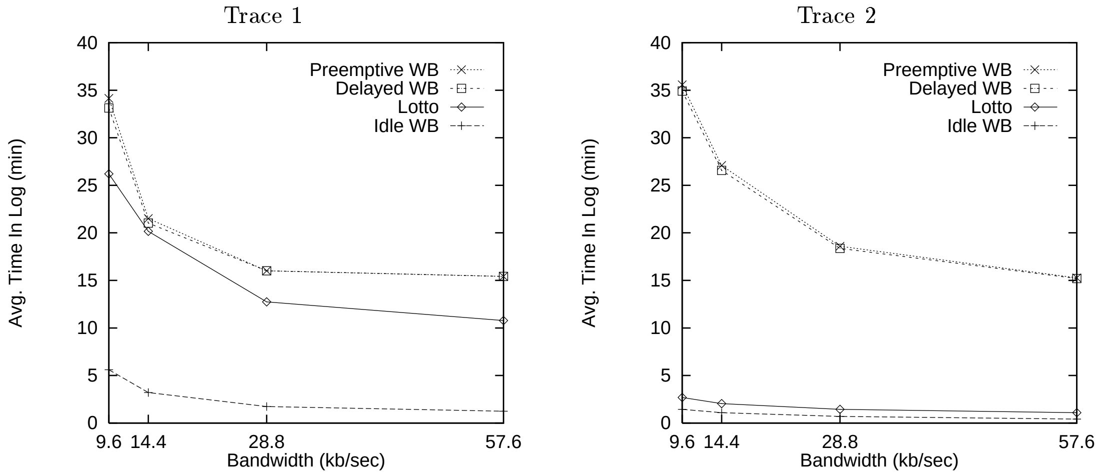

FIGURE 5. The effect of bandwidth on average time until reintegration (10 MB cache)

for over two-thirds of the requests in Trace 2, it seems unlikely that most workloads would ever experience any significant amount of write interference, even in low bandwidth environments.

Finally, the effect of write-back policy on time expansion is shown in Figure 10. Without much write activity (Trace 1), there is little difference between policies, but with more activity (Trace 2), the aggressive policies have slightly more impact than the conservative ones. Even with quite limited bandwidth (of 14.4 kb/sec) and these very simple policies, however, time expansion never exceeds 8%.

5.3. Block Caching Performance

Since block caching appears to offer some potential performance benefits for weakly connected operation (see Section 4.7), a preliminary comparison of block caching and whole-file caching was carried out. A client cache of 10 MB was used, with network connections ranging in speed from 9.6 to 57.6 kb/sec. A block size of 4 kB is used, and various representative write-back policies are considered, modified to operate on either a block or a whole-file basis.

Figure 11 shows the time expansion values experienced by block and whole-file caches for various policies. The use of block caching has a significant impact on time expansion; in fact, the effect of block caching is greater than the effect of choice of write-back policy. For both traces, block caching results in less time expansion than whole-file caching, regardless of the write-back policy in use. The differ-

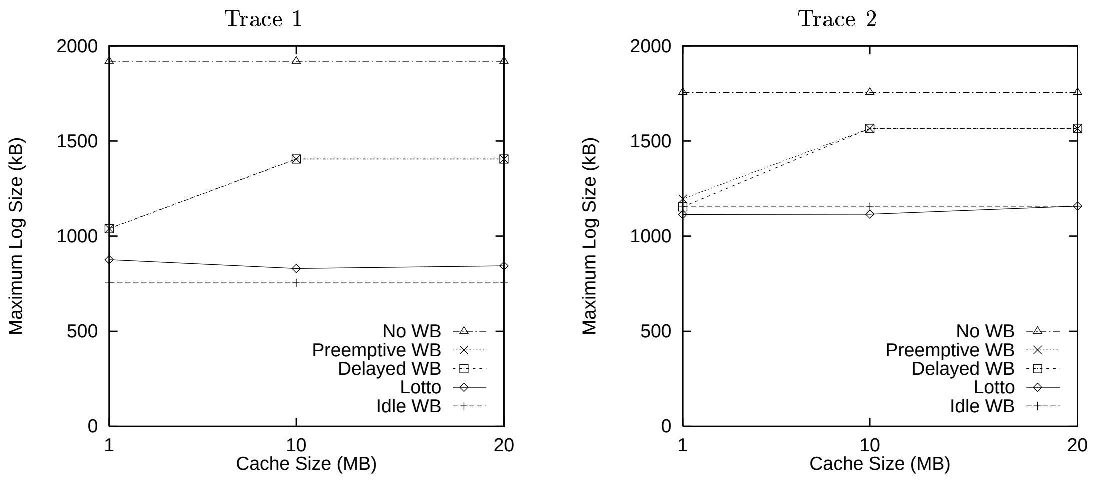

FIGURE 6. The effect of cache size on maximum log file size (b/w = 28.8 kb/sec)

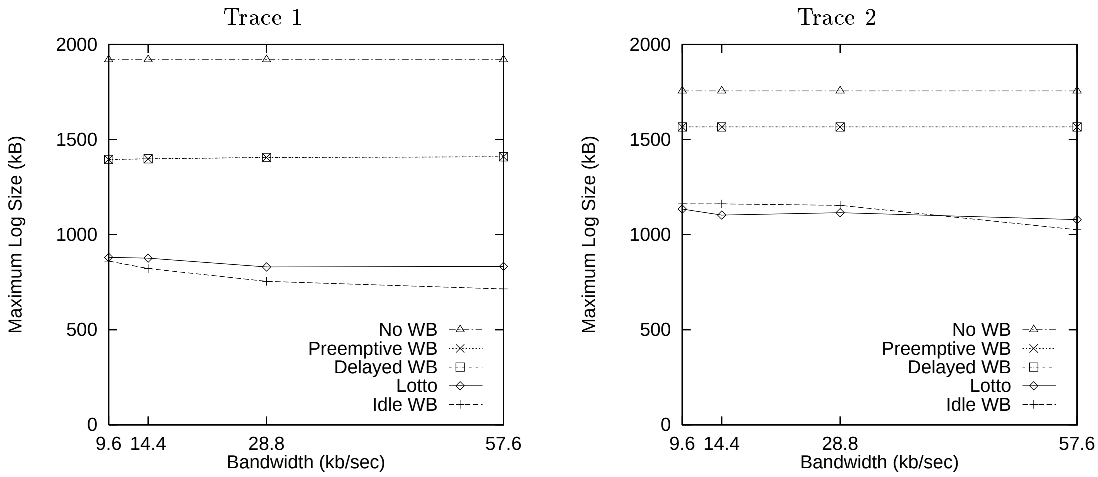

FIGURE 7. The effect of bandwidth on maximum log file size (10 MB cache)

ence becomes less as the connection bandwidth increases, suggesting that block caching may be even more desirable when bandwidth is low, contrary to expectations.

Figure 12 shows the impact of block caching on write interference. With an aggressive write-back policy such as Idle WB, block caching may result in much less interference, particularly at low bandwidths. For conservative policies such as Delayed WB, the difference is much less, and may (as with Trace 2) even be in the other direction.

Some other results (not shown) point to additional benefits of block caching in this situation. As expected, fewer bytes are written back (more than 50% fewer in many cases) and the size of the log file is reduced significantly (35-60% on average). We also saw reductions in the time to reintegration. Details of these results can be found in [19].

In summary, the presence of a weak connection presents an opportunity for block caching that is not feasible when disconnected. Our simulations found this to be beneficial. Time expansion, log size, and time in log are all reduced when block caching replaces whole-file caching. Write interference is reduced with aggressive write-back policies (especially in low bandwidth situations), but may increase slightly with conservative ones. Further advantages could come from "tuning" write-back policies for better performance when block-caching is used – something that was not done in this study. Block caching is definitely a direction that should be explored further.

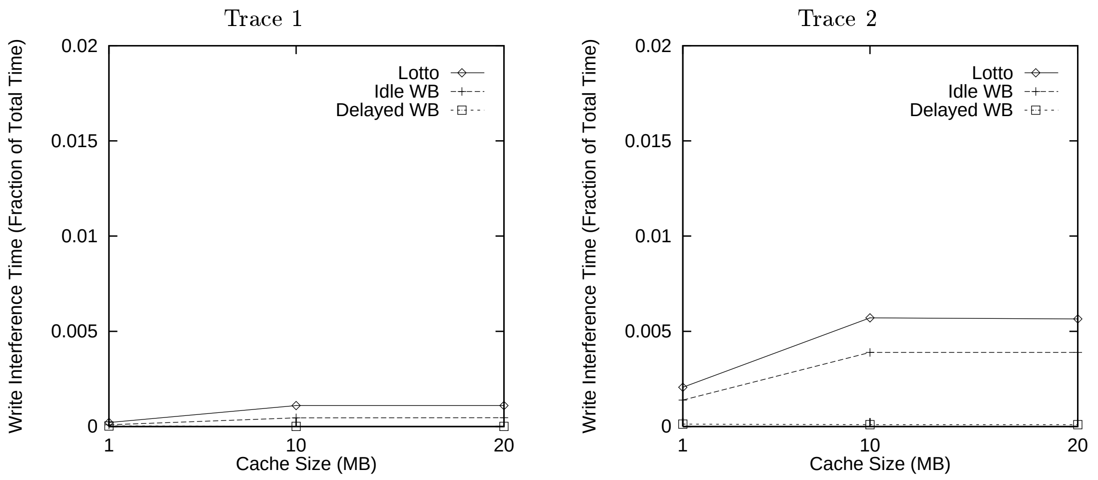

FIGURE 8. The effect of cache size on write interference time (b/w = 28.8 kb/sec)

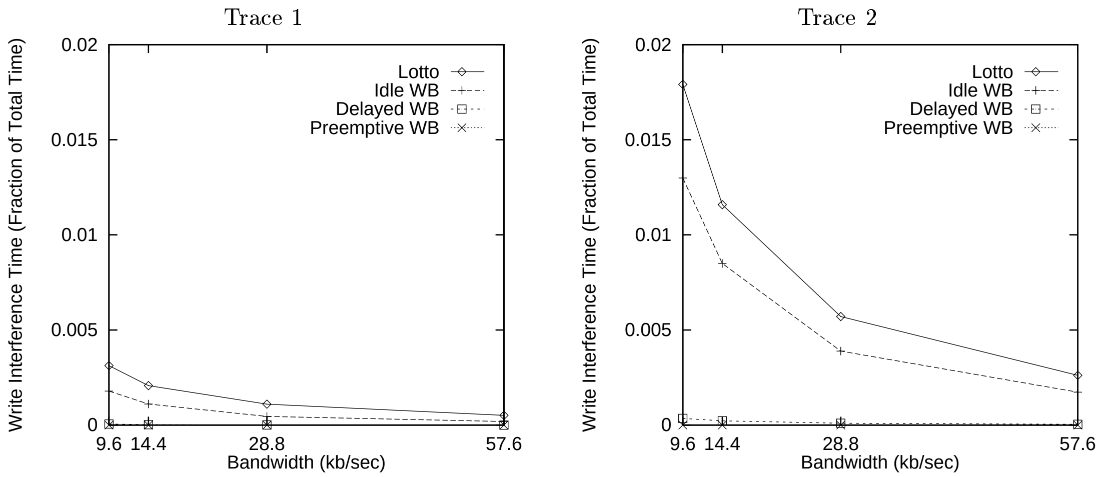

FIGURE 9. The effect of bandwidth on write interference time (10 MB cache)

6. SUMMARY AND CONCLUSIONS

The purpose of this study was to investigate the extent to which effective management of a mobile client's file cache can provide remote file system support with acceptable performance. It is clear that tradeoffs exist between resources available and performance. This study explores the dimensions of these tradeoffs.

Our simulation results show that it is possible to provide quite acceptable file system support in a weakly connected environment, servicing read requests in a timely manner while at the same time providing good update 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 2% for the workloads we examined. This level of performance can be achieved even at a modestly resourced client. For the traces used in this study, a 10 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 14.4 kb/sec network link proved to be enough to provide acceptable levels of performance. Simple policies work quite well.

The performance benefits of delaying write backs

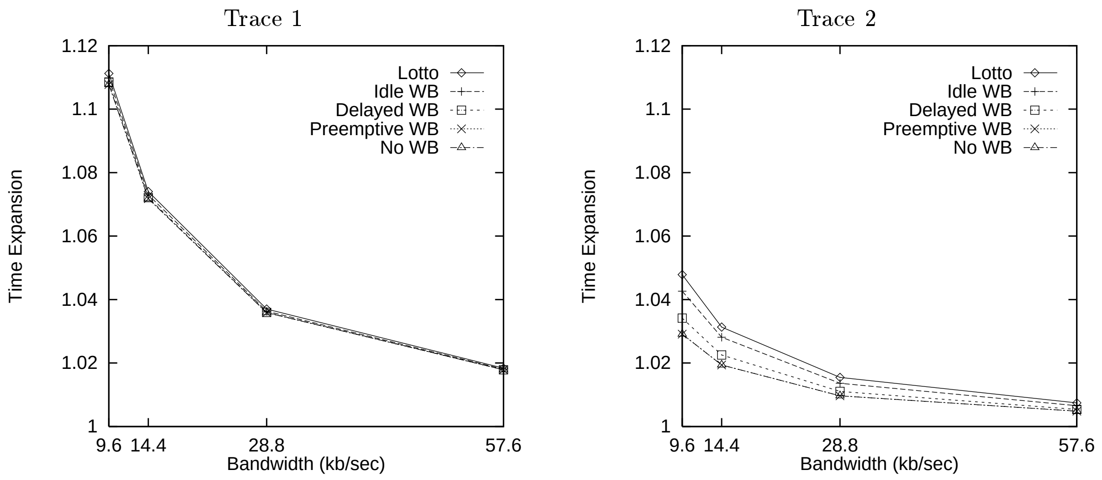

FIGURE 10. The effect of write-back policy on time expansion (10 MB cache)

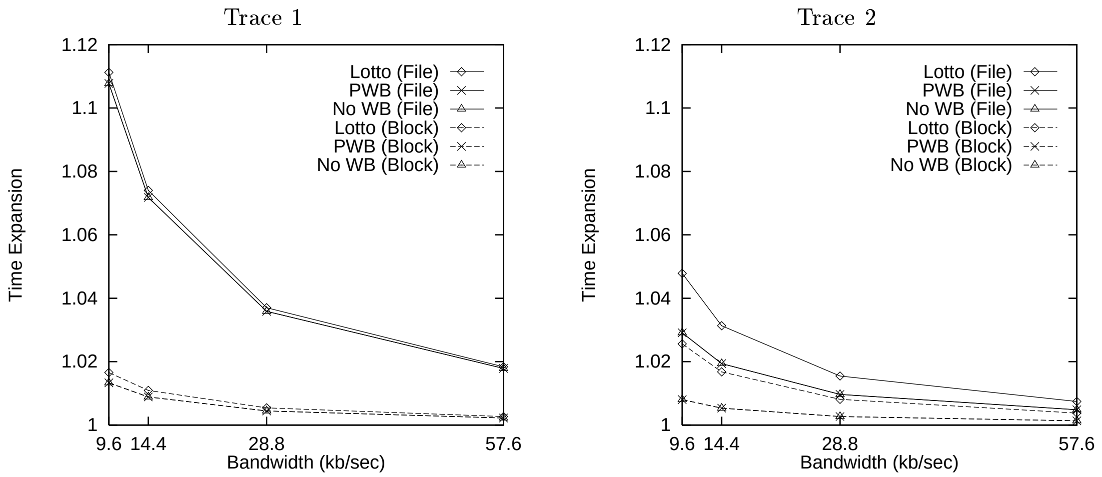

FIGURE 11. The effect of caching unit on time expansion

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. In terms of managing write backs, our results show that it is not necessary for write-back policies to be complex in order to achieve acceptable levels of performance.

Several advantages to block caching are apparent from this study. Time expansion, write interference time (for aggressive write-back policies), log file size, and time entries remain in the log file are all reduced when block caching is used. The results of this study indicate that there are definite advantages to performing block caching in a weakly connected environment. Whole-file caching is required to support disconnected operation, but designing a cache manager to adapt to a weak connection by switching to block caching when possible could prove beneficial. It could also be possible to allow the cache manager to change block size as bandwidth changes. Further investigation is required to examine issues such as how to cope with returning to a disconnected, whole-file mode of operation after performing block caching while weakly connected.

Mobile computing is here now, and although support is still somewhat limited, what is in place is sufficient to provide file system functionality to mobile clients with acceptable performance through well-managed caches. Functionality and performance

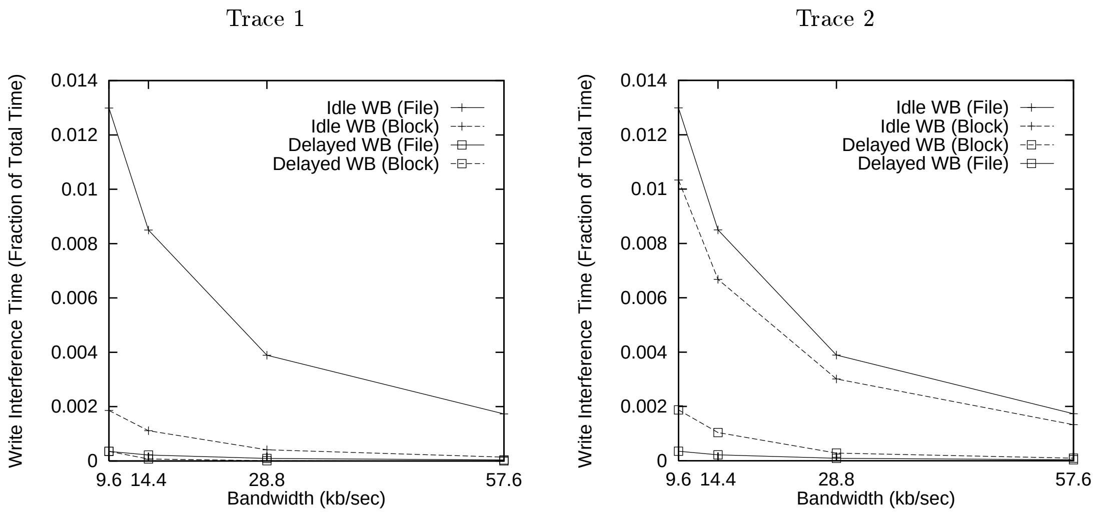

FIGURE 12. The effect of caching unit on write interference time

are equally important to widespread user acceptance, and efforts must proceed in both directions.

ACKNOWLEDGEMENTS

Financial support for this research came from the Natural Sciences and Engineering Research Council of Canada (NSERC) through Research Grant OGP0003707 and a Postgraduate Scholarship, and from Telecommunications Research Laboratories (TR*Labs*) in Saskatoon. Some of the equipment we used was donated by Hewlett Packard. We were fortunate to have the support of our colleagues in the *DISCUS* research group at the University of Saskatchewan during our experiments, and we particularly thank Greg Oster for his ongoing technical assistance. Finally, we gratefully acknowledge the helpful comments of the anonymous referees who reviewed our paper.

REFERENCES

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

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

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

[4] L. Huston and P. Honeyman. Peephole log optimization. In *Proc. Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, Dec. 1994.

[5] A. Banerji. 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.

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

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

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

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

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

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

[12] G. Kuenning, G. Popek, and P. Reiher. An analysis of trace data for predictive file caching in mobile computing. In *Proc. 1994 Summer USENIX*

Conference, pages 291–303, Los Angeles, CA, June 1994.

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

[14] D. Terry, M Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proc. 15th ACM Symposium on Operating Sytems Principles*, Dec. 1995.

[15] D. Dwyer and V. Bharghavan. A mobility-aware file system for partially connected operation. *ACM Operating Systems Review*, 31(1):24–30, Jan. 1997.

[16] E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–374, Dec. 1990.

[17] J. Howard, L. Kazar, S. Nichos, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, 1988.

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

[19] K. Froese. File cache management for mobile computing. M.Sc. thesis, Dept. of Computer Science, University of Saskatchewan, Saskatoon, Canada, 1997.

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

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

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