Network-Aware Adaptation Techniques for Mobile File Systems

Benjamin Atkin
NEC Laboratories America
atkin@nec-labs.com

Kenneth P. Birman
Cornell University
ken@cs.cornell.edu

Abstract

Wireless networks present unusual challenges for mobile file system clients, since they are characterised by unpredictable connectivity and widely-varying bandwidth. The traditional approach to adapting network communication to these conditions is to write back file updates asynchronously when bandwidth is low. Unfortunately, this can lead to underutilisation of bandwidth and inconsistencies between clients. We describe a new mobile file system, MAFS, that supports graceful degradation of file system performance as bandwidth is reduced, as well as rapid propagation of essential file updates. MAFS is able to achieve 10-20% improvements in execution time for real-life file system traces featuring read-write contention.

1 Introduction

Distributed file systems are a common feature of large computing environments, since they simplify sharing data between users, and can provide scalable and highly available file access [4]. However, supporting mobile clients requires coping with the atypical patterns of connectivity that characterise them. While a desktop client is well-connected to a file server under most circumstances, a mobile client frequently lacks the bandwidth to perform all its file operations in a timely fashion. Mobile file systems typically assume that a client is strongly-connected like a desktop host, or weakly-connected and should limit its bandwidth consumption to a minimum [5, 9]. If bandwidth lies between these extremes, assuming weak connectivity can be too conservative, since it delays sending updates to the server in order to aggregate modifications.

This paper examines the effectiveness of MAFS, a mobile file system that propagates file modifications asynchronously at all bandwidth levels. Rather than delaying writes, MAFS uses RPC priorities to reduce interference between read and write traffic at low bandwidth. To ensure that file modifications are rapidly propagated to the clients that need them, MAFS also incorporates a new invalidation-based update propagation scheme. Unlike previous mobile file systems, MAFS uses techniques that are oblivious to the exact bandwidth level, and can therefore react to bandwidth variations in a fine-grained manner. With real-life file system traffic featuring high read-write contention, MAFS is able to achieve improvements in execution time of up to 10-20% at both low and high bandwidths.

The authors were supported in part by DARPA under AFRL grant RADC F30602-99-1-0532, and by AFOSR under MURI grant F49620-02-1-0233, with additional support from Microsoft Research and from the Intel Corporation.

2 Motivation

Mobile access to shared data is complicated by an unpredictable computing environment: the network or a particular destination may be unavailable, or the throughput may be substandard, as shown in Figure 1. This graph shows results from packet-pair measurements of available bandwidth between a mobile host on a wireless network, and a wired host near the base station. As the mobile host moves, factors such as the distance to the base station and local interference cause the host's network card to switch to higher-redundancy, lower-bandwidth encodings. Such switching causes available bandwidth to oscillate greatly, even when the mobile host is stationary. If it is to ensure that clients' file operations are executed in a timely way, a file system must adapt to this variation.

Existing systems tailored to low-bandwidth clients differentiate between types of file system communication, so that bandwidth can be devoted to important, user-visible operations [5, 6, 9]. In particular, Coda [9] writes back changes to files asynchronously, and Little Work [5] assigns lower priorities to asynchronous operations at the IP level to reduce interference with foreground operations.

However, adaptation by deferred transmission of file updates has the disadvantage of increasing the delay before updates are applied at the file server, and therefore reduces the degree of consistency between clients' cached copies. For its own benefit, a low-bandwidth client may decide to delay sending a file's update to the file server, but this decision may also affect other clients that would like to read the file. Optimistic concurrency control and reconciliation of conflicting updates are typically used to resolve inconsistencies [9, 11]. When bandwidth is very low, this can be an acceptable price to pay for the ability to continue accessing a file server, but if bandwidth is less constrained, better inter-client consistency is achievable. Coda-like file systems therefore switch between a low-bandwidth, asynchronous-writes mode and a synchronous-writes mode, according to the available bandwidth. However, in a wireless network, variations in bandwidth can occur without the user's knowledge, so that changing modes creates unexpected incon-

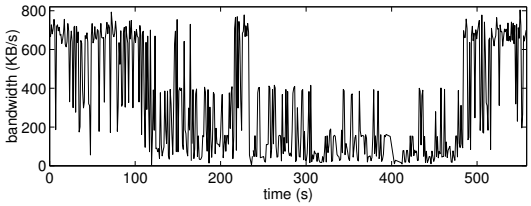


Figure 1: Time series of wireless bandwidth.

sistencies [2]. Adaptation based on low- and high-bandwidth modes can be ill-suited to situations where bandwidth is not severely constrained, but insufficient for a client to ignore it when deciding what to send over the network.

3 File system overview

MAFS (Adaptive Mobile File System) is a distributed file system designed to support efficient access to a remote file server by mobile clients that must cope with variations in available bandwidth. The MAFS design and terminology are similar to the Andrew File System [4] and Coda [9].

3.1 File access model

MAFS clients use *whole-file caching*: when a file is accessed for the first time, a client fetches the entire file from the file server and caches it. MAFS only sends the server the contents of a modified file when it is closed by an application: this is referred to as *writeback-on-close*. Directory operations cache directory contents and apply changes locally, as well as making an RPC to apply the changes to the server's copy. Whole file caching is effective if a client's connectivity is uncertain, since the client can always use cached copies of files instead of incrementally fetching them from the server. MAFS supports this type of *disconnected operation* [9], but not to the extent of automatic reconciliation of update conflicts [11]. On the other hand, block-level caching reduces the delay incurred when an application opens a file, and, as has been shown in the Low-Bandwidth File System [10], it is possible to use a content-based division of files into blocks as the basis for reducing client-server network traffic. However, reducing client-server traffic does not eliminate the fundamental problem of contention for insufficient bandwidth.

3.2 Inter-client cache consistency

When a client fetches a file, the file server grants it permission to cache the file for a limited period, and adds it to a list of clients that cache the file. If the client modifies and then closes the file, it transmits the new contents to the server, which makes a *callback* RPC to any other clients on the list. A client that receives a callback RPC discards its cached copy of the file. However, if an application has the file open when its client receives the callback, the file is discarded once it is closed. When several clients concurrently modify a file, the final contents depend on the client that closed it last. A client can lock a file to synchronise accesses: the server grants the client a lease [3] that is renewed each time the client communicates with the file server.

3.3 Adaptive Remote Procedure Call

MAFS uses Adaptive Remote Procedure Call for client-server communication. Adaptive RPC is based on our earlier work in network-aware communication adaptation [1], and differs from a typical RPC system in allowing applications to control how concurrent RPCs are transmitted, and special handling for failures due to insufficient bandwidth. Adaptive RPC requests and replies can contain an arbitrary amount of data. A sender also attaches a priority and timeout to the send operation. Like a Rover Queued RPC [6], an Adaptive RPC can be asynchronous, so that an application need not block waiting for the result: instead, the library makes an upcall when the reply arrives.

Since an application can perform multiple RPCs concurrently, Adaptive RPC schedules their transmission: this corresponds to allocating bandwidth among the competing RPCs. Attaching priorities to RPCs allows applications to control this scheduling policy. A programmer divides RPCs into classes based on the importance of their results to the user, and then assigns priorities to the classes. The library schedules RPCs based on priorities whenever there is insufficient bandwidth to transmit competing RPCs without a noticeable delay. RPCs from higher-priority classes are performed first, and RPCs of equal priority are performed in parallel. This ensures that the application adapts itself to the available bandwidth gracefully, since lower bandwidth translates into longer delays for lower-priority RPCs. RPC timeouts allow the application to prevent low-priority RPCs being silently starved. Using priorities allows a programmer to write an adaptive application without having to take account of the actual bandwidth or current mix of RPCs at runtime, and avoid having to specify thresholds at which it should switch communication modes.

An RPC whose results are urgently required should be assigned the highest priority, particularly if the RPC contains out-of-band communication. Larger, but still important RPCs can use intermediate levels, while the lowest levels are useful for RPCs that can be arbitrarily delayed, such as speculative activities like prefetching and transferring archival data. If the initial assumption regarding the correct priority level for an RPC proves incorrect, a call to the library can be made to assign a new priority.

3.4 Implementation

MAFS is implemented in C on FreeBSD. The client is a user-level process that stores cached files in a local filesystem. The server also stores its copies of files in a local filesystem. File system operations from applications are redirected to user level through a kernel module at the client.

CONTROL (highest)	fetch file attributes, callback, pull file update
FETCH	fetch file data, store file data (if forced)
PREFETCH	prefetch file data
METADATA	lock a file, invalidate file, most metadata RPCs
STORE	store file data, unlink file

Table 1: Priorities for MAFS remote procedure calls.

Remote procedure calls between a client and the file server are divided into several types depending on their function. RPCs to fetch and store data are self-explanatory. Metadata operations include fetching and setting file attributes, and directory operations such as creating and unlinking files. Control RPCs include locking files and the server's callback to invalidate a client's cached copy of a file.

4 Communication adaptation

To reduce its network communication when bandwidth is low, a mobile file system client can automatically adapt its communication strategy to the available bandwidth. Sometimes applications transfer a large volume of data that the user is unlikely to require immediately, consuming bandwidth that can be used for important tasks. For instance, consider an application that fetches images from a file server, processes each in turn, displays the resulting image, and writes it to the server. If the user wants to see the processed images, but no-one else wants to immediately read them, writing the output back will interfere with fetching the next image, and slow down the application.

Interference due to write traffic is often solved by writing back updates asynchronously: the application in our example can start reading another image without waiting for the previous output to be sent to the file server. Asynchronous writeback allows I/O and CPU processing to be overlapped, reducing execution time and utilising bandwidth more efficiently. However, if bandwidth is low, contention arises when files are being fetched at the same time as updates are written back. This contention can be mitigated by prioritising file fetch RPCs above writeback RPCs to ensure that they will be preferentially allocated bandwidth.

In this section, we assess the effectiveness of asynchronous writeback and RPC priorities in MAFS under different levels of bandwidth availability. In particular, we examine the degree to which a file system client that avoids switching modes in response to bandwidth changes is able to adapt to both insufficient bandwidth, and conditions under which bandwidth is plentiful.

4.1 Asynchronous writeback

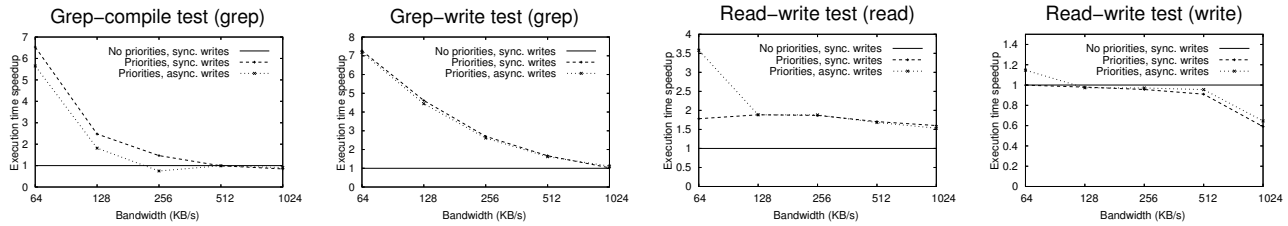
MAFS asynchronous writeback is based on similar mechanisms found in many mobile file systems [5, 9]. Rather than making an RPC when an application performs a metadata update or file write, the operation is logged and replayed to the file server after a delay. This scheme reduces bandwidth utilisation because some logged operations may be superceded by later ones [5], such as deleting a modified file. Such optimisations can be effective at low bandwidth, when there is a natural delay, but at high bandwidth, an artificial delay in writing back updates introduces inconsistencies between the client and the file server. This can be acceptable at low bandwidths, when the user may be grateful to be able to use the file system at all, but should be avoided when bandwidth is unconstrained [2].

MAFS avoids the need for modes by using asynchronous writeback at all bandwidth levels, and incorporates a new update propagation algorithm to reduce the possibility of inconsistencies (see section 5). As new operations are added to the tail of the log, the client flushes operations serially from the head of the log. Client-server traffic consists of a variety of foreground RPCs for control operations and fetching file data, and a stream of background RPCs for logged operations. When bandwidth is high, replayed logged operations complete quickly, with little extra delay. When bandwidth is low, logged operations are delayed in proportion to the foreground RPC traffic and the available bandwidth.

4.2 RPC priorities

MAFS uses priorities to reduce contention between foreground activities and deferrable background activities: Adaptive RPC preferentially allocates bandwidth to foreground RPCs. Unlike Little Work [5], which assigns a lower priority to writeback in low-bandwidth mode, MAFS has a finer-grained differentiation between RPCs, and uses priorities at all bandwidths. This allows control over bandwidth allocation at the level of individual RPCs, without requiring that an MAFS client is aware of the precise bandwidth.

When choosing priorities, automatic assignment and fine granularity are preferable, to avoid the need for user intervention and provide the maximum degree of differentiation among priority levels. Scheduling RPCs based on priorities is only effective if concurrent RPCs usually end up with different priorities, but processes are too coarse-grained for this purpose. File-based priorities provide some more detail, but the importance of a file can be hard to determine automatically [8], and files can be too numerous for the user to manually assign priorities. RPCs are more numerous, but priorities can be automatically assigned to them according to the operation the RPC corresponds to, as shown in Table 1. Small RPCs, or RPCs that the user has to wait for, have high priority; large RPCs, or RPCs whose results can be delayed, such as writing back data or prefetching, have lower priority. Prefetching is an example of *speculative communication*: performing a low-priority RPC whose results can improve performance if bandwidth is high, but can be safely omitted if bandwidth is low. For most RPCs, the initial priority is never modified, but the file server sometimes requests an increase in the priority of an RPC to transmit data.



(a) Grep test (b) Read-write test

Figure 2: Workloads with contention between priority levels. The grep workload consists of validating cached files.

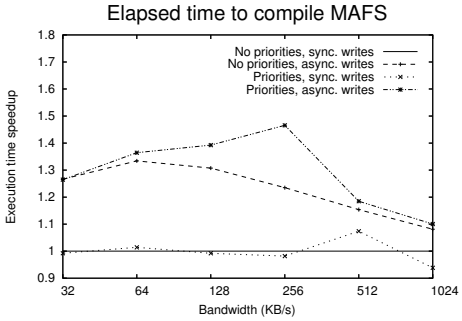


Figure 3: Compiling MAFS on top of MAFS.

	Mostly Writes	*Mostly Reads*	*Heavy Traffic*
duration (s)	106	31	26
distinct processes	15	47	30
distinct files	276	726	227
total of file sizes (MB)	16.51	12.18	17.65
read traffic (MB)	9.17	10.11	17.61
write traffic (MB)	32.39	2.35	5.42
average b/w (KB/s)	401.5	411.6	907.0

Table 2: NTFS trace parameters. Traffic numbers are for synchronous writeback.

4.3 Experimental evaluation

Two questions are of particular interest in evaluating the performance of MAFS communication adaptation:

(i) Do priorities improve performance by reducing RPC contention?

(ii) Is it possible to combine the benefit of asynchronous writeback at low bandwidth with acceptable performance at higher bandwidths?

We compare MAFS to alternative approaches in two sets of experiments: microbenchmarks to measure execution time speedup for simple workloads, and traces of actual Windows NT file system (NTFS) traffic. The NTFS traces were gathered in the Cornell University Computer Science Department, and contain access to local and remote file systems by clients in a local-area network [12].

Microbenchmarks

Our first microbenchmark compiles MAFS from 1.20 MB of source code stored in an MAFS filesystem, storing the 7.21 MB of output in the same filesystem. Figure 3 compares the execution time speedup for the benchmark under differing asynchronous writeback and priority schemes, as bandwidth is varied.

The dominant feature of Figure 3 is that asynchronous writeback is beneficial at all bandwidths until 1 MB/s. There is less improvement at 32 KB/s, where throughput is so low that control traffic and the delay in fetching files become dominating factors. At 1 MB/s, bandwidth is high enough to eliminate differences between writeback schemes. At intermediate bandwidths, asynchronous writeback is clearly beneficial, and priorities are advantageous in reducing contention between reading and writing, which is not possible when synchronous writeback is used.

The second microbenchmark evaluates a workload that contains explicit contention between different types of RPC traffic. In Figure 2(a), one process performs a grep on a set of cached files that need to be validated before they can be opened (via a small, high-priority RPC). Another process either writes data to files rapidly (Grep-write), or compiles MAFS (Grep-compile). In Figure 2(b), one process reads files at the same time as another is writing files. Figure 2(a) shows that priorities are beneficial for the small validation RPCs when the background traffic is heavy. With the sporadic background traffic of compiling MAFS, improvements are confined to low bandwidth levels. Figure 2(b) demonstrates that priorities can improve higher-priority read performance with only a small overhead for writes.

To summarise, these microbenchmarks show that asynchronous writeback improves performance even at comparatively high bandwidths, and priorities are effective in mitigating contention between different classes of RPCS.

Macrobenchmarks

We evaluated MAFS at a larger scale using the NTFS-derived traces summarised in Table 2. Although the original execution times of these traces were short on Windows NT, they execute slowly on MAFS due to high bandwidth requirements.

Figure 4 shows execution times under four combinations of writeback scheme and priorities. Asynchronous writeback

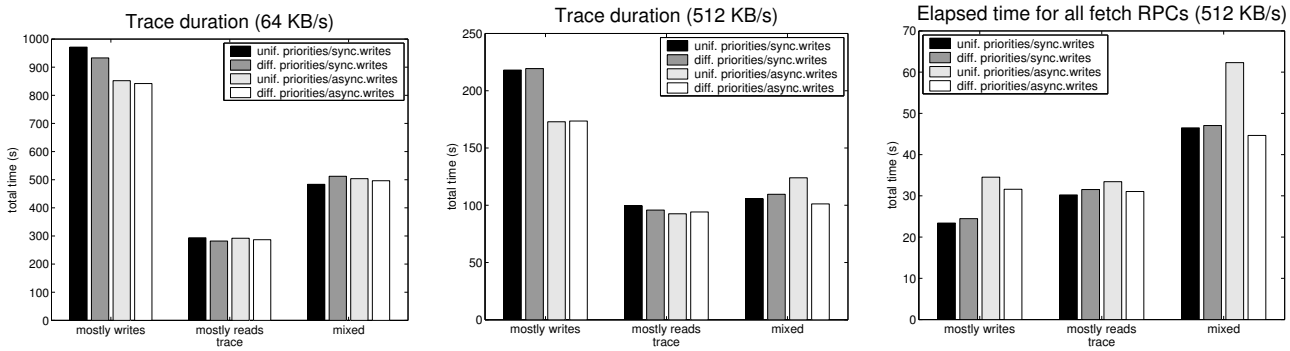


Figure 4: NTFS-derived benchmarks. Trace duration for asynchronous writes is until completion of the last read.

is beneficial in the Mostly Writes trace, which has high read-write contention. At 512 KB/s bandwidth, it is less effective than synchronous writeback, due to increased contention, but this effect is mitigated by using priorities (this is clearer in the graph for time spent on fetch RPCs). At the timescales in the NTFS traces, the improvements are less dramatic than in the microbenchmarks, but they demonstrate that MAFS can improve the performance of large-scale mixed workloads.

Analysis

Both experiments confirm the benefits of asynchronous write-back, even at bandwidths where a typical mobile file system performs all RPCs synchronously. Asynchronous writeback avoids the need to switch operation into a distinct low-bandwidth mode, and choosing a bandwidth threshold at which to switch. When used by themselves, priorities do not always result in improved performance, since they are only effective if concurrent RPCs have different priorities. However, they reduce user-visible delay and contention that is introduced by asynchronous writeback.

5 Update propagation

Using asynchronous writeback at all bandwidths delays sending updates to the file server. In this section, we evaluate the effectiveness of an update propagation scheme to reduce this delay. MAFS allows a client to delay transmitting updates, but the file server forces file updates to be written back when another client must read an up-to-date copy of the file.

5.1 Origin of inconsistencies

Since asynchronous writeback decouples modifying a file from notifying the server that a change has occurred, it can generate inconsistencies between cached copies. Figure 5 illustrates the potential for inconsistency: during the *writeback window*, another client accessing a cached copy, or fetching the file from the file server, will not read up-to-date data. From the server's perspective, there is no inconsistency, since it is unaware of the new update. However, from a global perspective, the second

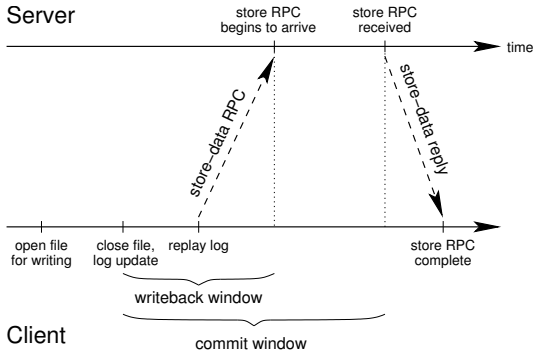


Figure 5: Timeline of a file update. Time advances from left to right.

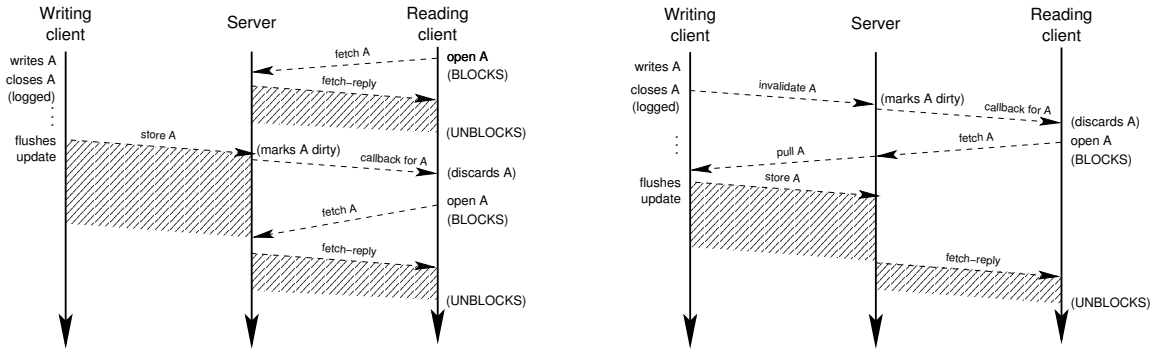
client will access stale data. Due to network latency, the write-back window can never be eliminated, but adding an additional delay before writing back the update increases the scope for inconsistency. Figure 6(a) illustrates how this inconsistency can arise.

In a Coda-like file system such as MAFS, a different type of inconsistency is introduced between a client and the server when a file is modified, since the change is hidden from the server until the file is closed. For the purposes of this investigation we assume that the open-close interval for a file is small relative to the network latency and writeback delay. The update propagation techniques we describe can be applied equally well to individual file writes as to writeback-on-close.

5.2 Techniques for update propagation

Although Coda-like file systems can generate inconsistencies between clients, they were designed to permit a client to function at low bandwidth, rather than for rapid update propagation. Since it is impractical to lock files if clients are permitted to modify the filesystem while they are disconnected, Coda supports stronger consistency through optimistic replication [11].

An alternative approach is to allow a client to use asynchronous writeback, but require that it alerts the file server when a file is modified, by sending an invalidation RPC. This informs the server that the update exists before the new file contents ar-



(a) Pure asynchronous writeback

(b) Asynchronous writeback with invalidations

Figure 6: Asynchronous writeback. A client's update is logged when the file is closed. While it is in the log, other clients see the server's stale version. An invalidation RPC allows the server to invalidate other clients' cached copies.

rive, so that it can prevent inconsistencies by inhibiting access to the file by other clients, as shown in Figure 6(b). Invalidations are used in Fluid Replication [7] to allow clients to avoid sending data across a wide-area network: instead, the server only asks the client for a file's data if another client requests it.

5.3 Selective invalidation with reader pull

MAFS incorporates SIRP, a new algorithm for maintaining inter-client consistency, which combines asynchronous writeback with invalidations and expedited transmission of updates for files that other clients are attempting to read.

Selective invalidation. Using an invalidation RPC to alert the file server to the existence of a new update improves cache consistency, but consumes additional bandwidth. If writeback traffic is low enough for the server to start receiving an update immediately after it receives the invalidation, the invalidation is superfluous. SIRP avoids this overhead by performing *selective* invalidation: when a client adds an update to the writeback queue, it only sends an invalidation if the queue is not empty. If the queue is empty, the invalidation is piggybacked onto the update.

Reader pull. When the server receives an invalidation from a client, it makes callbacks to all the other clients that cache the file, to tell them to discard their copies. If several clients modify the same file, modifications are serialised in the order of their invalidations. Normally, the client that made the update only transmits it when it reaches the head of the writeback queue. If another client attempts to fetch the file during the update's writeback window, the server blocks that client until the update has arrived. The server also makes a *pull* RPC to the client that modified the file, instructing it to expedite sending the update. When it receives the pull RPC, the client begins sending back the update at the same priority as an RPC to fetch file data, so that it will be preferentially allocated bandwidth. If the update was already being written back, the client increases its priority.

In principle, a client that modifies a file could save bandwidth by not sending it to the file server at all, unless the server pulls it to supply it to another client. MAFS clients push updates to the server in the background, to reduce the delay incurred when fetching an invalidated file: pushing updates can result in the server having received some, or all of the update by the time another client accesses it.

The effect of selective invalidation and reader pull is that SIRP behaves similarly to synchronous writeback if a client concurrently fetches a file, but behaves like asynchronous writeback when there are no concurrent fetches. Like synchronous writeback, SIRP sends an RPC to the server as soon as an application closes a modified file, but it can defer transmitting the actual contents until they are needed.

5.4 Experimental evaluation

We conclude this section with an experiment that compares the effectiveness of SIRP to three alternatives. Synchronous writeback transmits an update as soon as a file is closed, and asynchronous writeback puts the update in a queue and transmits it as soon as it reaches the front of the queue. We also compare SIRP against a policy we refer to as SIRP-C, which only differs from SIRP in performing compulsory invalidations: every update results in an invalidation RPC to the server. Two questions are of particular interest in this comparison: First, how stale are the files readers read? Second, how is the performance of readers and writers affected by stronger consistency?

Experimental setup

Experiments were conducted in a network of five hosts: one file server, one writer client that was responsible for modifying a collection of files, and three reader clients that only read the files. The bandwidth between the writer client and the server was set to 32, 128 or 512 KB/s, and the reader client-to-server bandwidth was always set to 1 MB/s.

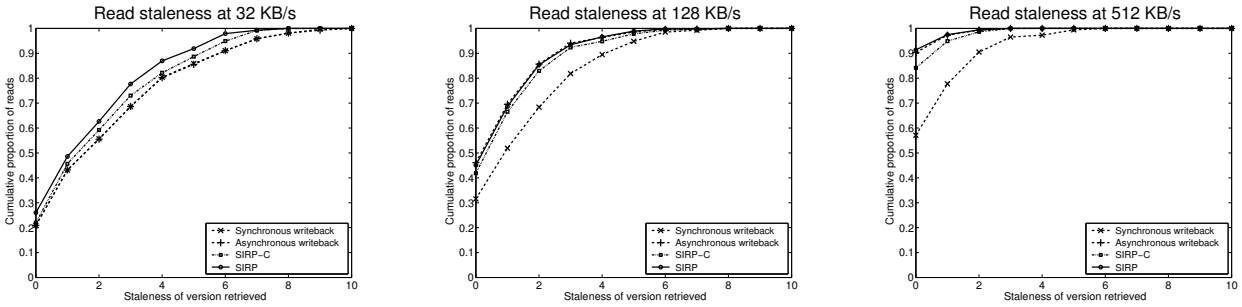


Figure 7: Staleness of reader file accesses. Cumulative distributions for the staleness of all accesses to files by the three readers are shown. Higher curves represent less staleness.

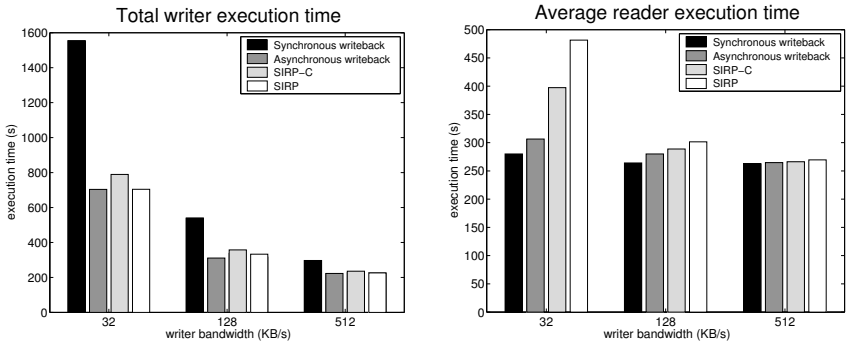


Figure 8: Execution times for concurrent access trace. Reader execution times are averages for the three readers.

The files shared between the clients were divided into 20 file sets of 8 files each. File lengths were randomised, with an average length of 128 KB, to prevent the clients falling into lockstep in the course of fetching and writing back the files. Clients performed 50 file set operations, consisting of selecting a random file set and performing a sequence of reads or writes on files in it. The writer performed a file set operation of 8-20 accesses every 2-10 seconds, with each access being equally likely to open a file for reading or writing. Readers performed a file set operation of 2-20 reads every 1-5 seconds. The first 5 file sets were treated as "hot", meaning that 90% of the file set operations were directed to those file sets.

Read staleness

Comparing update propagation schemes requires a criterion for measuring the staleness of file reads. We identified updates to files by associating a version number with each file, and incrementing it every time the file was modified. Reads were labelled with the version number of the file at the time the read occurred. The staleness of a particular read was determined according to an ideal version number derived from executing the experiment with all participants running on a single host. In a real execution, the difference between the version number a read returns and the optimal version number determines how stale the read is. Figure 7 shows cumulative distributions for the staleness of reads at different writer-server bandwidths. Improved consistency results in fewer stale reads, and this is reflected by a curve that is higher on the left side of the graph.

Overall, higher bandwidth results in less staleness, since writes can be sent to the file server faster. At 32 KB/s bandwidth, SIRP is most effective at reducing staleness: though many reads return out-of-date file contents when compared to the optimal version, 5% more SIRP reads are up-to-date, compared to synchronous or asynchronous writeback. Allowing higher degrees of staleness, 9% more reads performed with SIRP are within 3 versions of the optimal. With this bandwidth level, synchronous and asynchronous writeback coincide in performance, since they are constrained by the bandwidth bottleneck and send updates in the same order. By suppressing unnecessary invalidations, SIRP reduces its bandwidth usage and achieves a small improvement over SIRP-C, since devoting less bandwidth to invalidations results in data reaching the server faster.

At higher bandwidths, asynchronous writeback performs as well as SIRP, but surprisingly, synchronous writeback continues to underperform. This is because the progress of writers using asynchronous writeback schemes is less constrained by the bandwidth, and they can overlap computation and fetching file contents with writeback. Rather than simply being a self-interested optimisation by writers to improve their own performance, asynchronous writeback therefore benefits both writers and readers.

Consistency maintenance cost

The overhead of the update propagation schemes can be compared by referring to the reader and writer execution times, as

shown in Figure 8 (reader execution time is the average for all three readers).

For the writer, the reduced staleness achievable by SIRP has little or no cost compared to asynchronous writeback with no invalidations. Since the writer is up to 14% slower when using SIRP-C compared to SIRP, selective invalidation is clearly beneficial. For readers, SIRP has the highest average execution time, but this is because it provides the best consistency of all the schemes. If a reader reads more up-to-date file versions, then it transfers more data: in fact, the reader execution time for each case is proportional to the amount of data transferred between the reader and server, though lack of space precludes showing this in a graph.

Summary

This experiment demonstrates that SIRP is preferable to asynchronous writeback at low bandwidth, and adds little additional overhead. At higher bandwidths, the difference between asynchronous schemes is minimal, but any scheme improves over synchronous writeback. For the same reasons that it improves performance, asynchronous writeback reduces staleness, and SIRP makes it an acceptable choice at low bandwidth.

6 Conclusion

This paper has described MAFS, a new file system for mobile clients that is tailored for wireless networks by incorporating automatic adaptation to the available bandwidth. MAFS differs from previous designs in making use of asynchronous writeback at all bandwidth levels, rather than switching from synchronous to asynchronous writeback when bandwidth is insufficient. RPC priorities and a new update propagation algorithm, SIRP, reduce a client's contention for wireless bandwidth, and permit a degree of consistency that is equivalent to instantaneous propagation of updates. Experiments demonstrate that these techniques allow MAFS to achieve performance that is at least equal to, and in most cases superior to that achievable by conventional file system designs that switch between low- and high-bandwidth modes according to thresholds. MAFS is therefore able to make efficient use of the network and provide predictable file system semantics, regardless of the available bandwidth.

Acknowledgements

We thank Robbert van Renesse, Werner Vogels, Emin Gün Sirer, Paul Francis, Rimon Barr and Stephen Rago for comments regarding this work.

References

[1] B. Atkin and K. P. Birman. Evaluation of an adaptive transport protocol. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Infocom 2003)*, volume 3, pages 2323–2333, San Francisco, California, Apr. 2003.

[2] M. R. Ebling, B. E. John, and M. Satyanarayanan. The importance of translucence in mobile computing systems. *ACM Transactions on Computer-Human Interaction*, 9(1):42–67, Mar. 2002.

[3] C. Gray and D. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the Twelth Symposium on Operating Systems Principles*, pages 202–210, 1989.

[4] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, Feb. 1988.

[5] L. B. Huston and P. Honeyman. Partially connected operation. *Computing Systems*, 8(4):365–379, 1995.

[6] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek. Mobile computing with the Rover Toolkit. *IEEE Transactions on Computers*, 46(3):337–352, Mar. 1997.

[7] M. Kim, L. P. Cox, and B. D. Noble. Safety, visibility, and performance in a wide-area file system. In *Proceedings of the First USENIX Conference on File and Storage Technologies*, Monterey, California, Jan. 2002.

[8] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 264–275, Saint Malo, France, Oct. 1997.

[9] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 143–155, Copper Mountain Resort, Colorado, Dec. 1995.

[10] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pages 174–187, Lake Louise, Alberta, Oct. 2001.

[11] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–188, Copper Mountain Resort, Colorado, Dec. 1995.

[12] W. Vogels. File system usage in Windows NT 4.0. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, pages 93–109, Kiawah Island, South Carolina, Dec. 1999.