Adaptive File Cache Management for Mobile Computing*

Jiangmei Mei Rick Bunt
Dept. of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan
Canada S7N 5A9
E-mail: {jim373, bunt}@cs.usask.ca

July 12, 2002

Abstract

File service is a fundamental user requirement, and this applies to mobile users as well as stationary ones. Indeed, mobile users have every reason to expect file access performance comparable to what they would receive in a traditional LAN-based environment, although delivering this is challenged by the constantly changing mobile environment.

Caching files at clients is commonly used in file systems to support mobility. By performing file system operations on cached copies of files mobile users can reduce their need to access files across the network. Effective cache management strategies will limit use of the network to a small number of read misses and periodic reintegration of changes (updates), and thus can shield the mobile user from changes in bandwidth that will have a significant impact on performance.

This paper investigates adaptive approaches to file cache management in mobile environments. Our research addresses cache management strategies that are able to adapt their behaviour in response to changes in bandwidth in order to maintain acceptable performance. Our focus in this paper is on adaptive approaches to file updates and adaptive adjustment of the caching unit. The results of a trace-driven simulation study of our adaptive approaches are presented and demonstrate that adaptive approaches to file cache management can improve file system performance for mobile users at acceptable cost.

1 Introduction

Mobile computing technology allows users to access data and information services anywhere, anytime. By freeing computer users from the constraints of physical location, it allows seamless movement from one place to another.

Mobile computing provides convenience for users, but this doesn't come without challenges. Variations in the network environment (specifically the speed and reliability of the connection) and the resource limitations of mobile devices may make it difficult to achieve performance comparable to that experienced in a stationary LAN environment.

In a mobile environment, users connect to the network through a range of mobile devices, such as laptops, notebooks, palmtops, or PDAs, at different times and at different places, through either wired or wireless connections. From time to time they may be strongly connected to the server through a fast and reliable link, weakly connected over a slow link, or even disconnected when the client has no connection to the server. A file system for mobile users must provide file services through these different types of connectivity, while at the same time keeping the performance impact of mobility to a minimum.

File systems that support mobile users, such as Coda [10, 12] and AFS [7, 8], rely on optimistic file caching to alleviate the impact of mobility. Cache management in such a file system addresses issues such as hoarding files to prepare for mobility, servicing file system requests in cache where possible when mobile, retrieving

*This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and TR*Labs*.

requested files from the home file server where necessary, and propagating updates to the home file server for timely reintegration. In a mobile environment, the bandwidth of the connection to the home file server, however, can change suddenly and dramatically. Since bandwidth has a large impact on the time to complete these file operations, a sudden reduction in bandwidth can have a significant impact on performance. For this reason, cache management strategies that assume that the same bandwidth is available throughout the connection (as is the case in a conventional LAN-based distributed file system) may not perform well in a dynamically changing mobile environment. Cache management in a mobile environment must react dynamically to changes in resource availability – to make effective use of increased resource, and to keep the impact of decreases to a minimum. Froese and Bunt [6] showed that different policies and parameters affect file cache performance in a mobile environment, as does the bandwidth of the connection. The goal of our research is to examine the extent to which cache management in a mobile environment can be designed to adapt to such changes in resource availability, either by dynamically altering management policies or by adjusting parameters in response to changing conditions.

Previous work on file system support for mobile computing has focused mainly on providing functionality to mobile users. While policies have been proposed to manage certain aspects of the file cache, such as write backs in AFS and Coda, adapting cache management to changes in the network conditions has rarely been addressed[1]. Our research is an extension of some earlier work on file cache management for mobile computing [4, 5, 6]. We used the same system model, performance metrics, file system traces and simulator to carry out a simulation-based evaluation of the performance benefits of adaptive file cache management in a dynamically changing mobile environment.

The remainder of this paper is organized as follows. Section 2 discusses issues in file caching for mobile clients. Section 3 discusses the need for adaptation in file cache management and describes how cache management can be made to adapt to changes in the mobile environment. Section 4 presents results from our trace-driven simulation experiments. Section 5 summarizes the results and provides conclusions.

2 File Caching for Mobile Clients

Optimistic caching of file replicas at clients is a technique commonly used to improve performance in distributed file systems. Keeping copies of files at the client computers not only reduces latency in servicing file requests, but also avoids traffic on the network. Caching is effective because most clients' file references exhibit locality properties: a file referenced once is likely to be accessed multiple times in the near future, and most files are read and written sequentially.

In a distributed file system that contains mobile computers, caching plays an even more important role than in a traditional distributed system in providing mobile users with data availability and performance. Data availability is a major issue in mobile computing. If required data is not available locally at the client, the user's work must be suspended or halted until the requested data can be retrieved from a remote source. When the user is mobile he/she may connect to the file server through different communication methods at different places (at different times), and the connections may have quite different characteristics. Much of the time, there may be only a "weak" connection to the home file server over a slow and perhaps error-prone link, and sometimes the client may be totally disconnected. This means that each access to the file server may require substantial time and performance degrades significantly. Maintaining copies of a mobile client's actively used files in his/her local cache allows the client to continue his/her work even while disconnected from the file server, and improves the performance when weakly connected since retrieval over the network is less frequent. As a by-product, caching can also mask failures at the server and in the network if the data requested can be obtained locally.

As in any distributed file system, cache management in a file system supporting mobility deals with issues such as deciding which file or data block to fetch, where to store the incoming data, what to replace if necessary to make room, and when to write updates back to the file server. Effective policies attempt to predict future file references, so that the cache can provide fast response to future requests, and well-designed policies can greatly improve cache performance. For mobile computing, a cache design might include additional elements such as a hoarding policy (for preloading files prior to going mobile) and a prefetching

[1] An exception is the work of Mummert [12], but her adaptation addressed adjusting only the size of the update reintegration "chunk" in response to changes in the bandwidth of the connection.

policy (to attempt to reduce cache misses even further). Other design issues include the size of the cache in a resource-limited environment, the choice of caching unit (whole-file caching or block caching), the mechanism for maintaining cache consistency (to keep cache content up-to-date), and techniques for log file optimization. Each of these decisions affects file system performance and must be made carefully.

While mobile, a user's read requests are given higher priority to access the network because read misses result in suspending the user's work [5]. Writes are recorded in a log file for later propagation to the file server. Delaying write backs allows for consolidation of overlapping requests, but longer reintegration time increases the possibility of write-write conflicts and loss of data, as well as the size of the log file.

3 Adaptive File Cache Management

Strategies used to manage the cache in non-mobile file systems may not be effective in mobile environments [4, 17, 18] because a mobile environment is significantly different. Mobile computing is characterized by limited resources at the client and volatile network connections. Designs for mobility must cope with both limitations and varying conditions [15, 2]. Therefore, the cache management strategies for traditional distributed file systems should be modified or redesigned to adapt to changes.

3.1 The Need for Adaptation

Compared to desktop computing devices, mobile devices are resource-poor and less secure [13, 16]. Mobile devices are smaller in size, lighter in weight, and lower in power consumption than desktop computers. These requirements facilitate mobility and portability but pose significant restrictions on CPU speed, memory size, disk space, and battery life. Data stored on mobile devices is more vulnerable, since portable devices have higher risk of physical damage, loss or theft than desktop computers [3, 16]. It is necessary to cope with these limitations in the design of support software.

Mobile clients may also experience considerable variations in connectivity, such as changes in bandwidth and error rate. As a mobile user changes locations, he/she must rely on the communication technology available at his/her current location to connect to a home server. Available bandwidths can vary by orders of magnitude – from kilobits-per-second to megabits-per-second. Depending on the technology available, bandwidth and quality of service may vary dramatically and unpredictably because of communication noise, congestion, or environmental interference, and disconnections may be frequent. Such wide variations in the mobile environment make any form of static planning or optimization of network operations impossible [1]. Clearly mobile systems must deal internally with variations if good performance is to be achieved. The solution is to provide the ability to adapt to these variations.

As much as possible, a mobile user should receive acceptable service while experiencing variations in resource availability. To achieve this, mobile systems and applications should react to changes dynamically – to take advantage of increased resource availability when that happens, and to avoid the negative impact of decreases when they happen. Being able to adapt successfully to changes in resource availability is widely recognized as a central support issue for mobile computing [1, 3, 9, 14, 13, 16, 19].

3.2 How to Adapt

In a file system supporting mobility, cache management should adapt dynamically to changes in the mobile environment, because those changes dramatically affect the performance of many aspects of cache management.

In this paper, our focus is on two elements of file cache management – write-back policies (policies for file system reintegration) and the size of the caching unit (the amount of data to be brought into the cache in a block caching implementation). The changing factor of interest is the connection bandwidth. To make writing back adaptive to bandwidth changes, it is proposed to change the aggressiveness of write backs when a change in resource levels occurs – either by using a different policy or by adjusting parameters to change the policy's behaviour. Froese [4] analysed both aggressive write-back policies, such as *idle write back*, and conservative write-back policies, such as *delayed write back*. If delayed write-back is implemented, the parameters can be changed to allow the policy to perform more aggressively or less aggressively. Aggressive write backs reduce reintegration time but place more demands on the network. The amount of data brought

into the cache can also be changed dynamically as the bandwidth changes. By affecting the data transfer time and the miss rate for subsequent file system operations, this will affect the user's waiting time.

4 The Experiments

This section describes the experimental setup for our trace-driven simulation study of adaptive file cache management, including the mobile environment assumed, the simulation model, and the traces used to drive the simulation. Simulation results from experiments with adaptive write back and adaptive block caching are then presented.

4.1 Definition of the Environment

The mobile environment for this work is defined as follows. After a period of time for hoarding, a mobile user is away from home with his/her mobile device and will perform work at several locations (at different times). While mobile, he/she connects to the home file server over a range of different connection bandwidths and performs work that involves file system activities.

The system model for this research, shown in Figure 1, is an extension of Froese's model [4] for a weakly-connected mobile environment. The mobile client contacts a single logical home file server for file system requests. While the file server is always connected to the internet through a fast and reliable link, the mobile client is either disconnected or weakly connected to the internet when mobile. Although a mobile client may sometimes have a strong connection to the internet while mobile, this case is not considered for this study. The performance of adaptive cache management is investigated only when the network conditions are less than ideal.

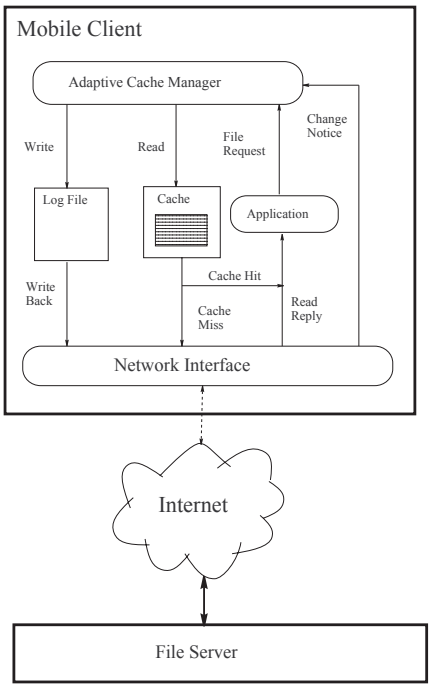

Figure 1: The System Model

In the mobile client, the file cache contains copies of files or file blocks that the user either hoarded prior to going mobile or referenced recently. The log file records the user's file updates. These updates are eventually propagated to the file server according to the write-back policy in effect. There is a mechanism in the network interface to detect changes in the network conditions, including bandwidth, latency, error

rate, etc., and inform the cache manager if a significant change occurs. When notified of any change in the environment, the adaptive cache manager adapts to the changes by modifying the behaviour of cache management strategies.

4.2 The Traces

Detailed traces were collected of user file system activity in the Distributed Systems Performance Laboratory in the Department of Computer Science at the University of Saskatchewan. Each trace records an individual client's real day-to-day file system activities in a typical academic research environment running Unix. Each trace record contains information about the particular request for service, including the type of the request (read or write), the time of the request, the user ID, the inode number, the file offset, and the file size. Full detail on the trace collection, the post-processing that was done, and the meaning of individual fields is given in [4].

The characteristics of the traces we selected for this study are given in Table 1. Each trace represents the file system activity for a single user over a seven-day period. The number of *active hours* in each trace is computed by subtracting the amount of "idle" time in the trace from the time between the first and last references in the trace. *Total Requests* is the number of file system events in the trace, *Unique Files* is the number of different files referenced, and *Unique Bytes* is the total size of those files. *Read Transfers* and *Write Transfers* are the sum of the sizes of all read and write requests in the trace, respectively, and *Write Percentage* is the percentage of all read and write requests that are writes. Finally, *Intervals* is the number of inter-event times (inter-event time is the time between two consecutive events) that have the length of time indicated. Intervals affect assumptions relating to disconnections. If an interval is longer than fifteen minutes and shorter than one hour, it is assumed that the client is experiencing an unexpected disconnection. If the interval is more than one hour, it is considered to be an expected disconnection.

The three traces selected for these experiments represent the range of workloads that a mobile client may expect to deal with in a typical academic environment [6]. Trace 1 has a moderate amount of file system activity and a read-to-write ratio of about 2:1. Trace 2 has many more requests and a high percentage of writes (a read-to-write ratio of about 1:2). This trace was selected as a good "stress test" for update propagation strategies. Trace 3 has both the fewest requests and the lowest percentage of writes (a read-to-write ratio of about 3:1).

Table 1: Trace Characteristics

Trace		1	2	3
Active Hours		12.9	7.0	8.5
Total Requests		21013	53860	17690
Unique Files		383	1000	443
Unique Bytes (MB)		17	18	22
Read Transfers (MB)		4596	4718	3723
Write Transfers (MB)		4	35	2
Write Requests (%)		38.7	67.9	25.2
Intervals	15-60 (min)	10	6	9
	> 60 (min)	11	5	16

4.3 The Simulation Model

Prior to going mobile, the client is strongly connected to the file server and hoards files into the cache to prepare for mobile operation. While mobile, the mobile client experiences a number of connections to the file server, with possibly different bandwidths over the course of a session. For this study we considered six bandwidths: 10 Mbps, 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps. Bandwidth of 10 Mbps is used when the mobile client is strongly connected, and the other five bandwidths are used while mobile. Network bandwidths 10 Mbps, 2 Mbps and 64 kbps correspond to the nominal speeds of Ethernet, WaveLan, and

ISDN, respectively, while bandwidths of 512 kbps, 9.6 kbps and 2 kbps represent other bandwidths a mobile user may experience[2]. Whenever a disconnection (expected or unexpected) happens, a new bandwidth for the next connection is chosen randomly from the five available: 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps.

Trace events are read and processed sequentially. When a read event is encountered, the client cache is searched for the requested data (files or data blocks). If the data is in the cache, the cache is updated, no network access is required. Otherwise, the requested data will be transferred from the file server over the weak connection. The newly transferred data will be put in the cache. Cache replacement may be required if there is insufficient room for the new data. When a write event is encountered, the log file is examined to find if the new update overlaps any current log entry. If it does not, the new update is recorded in the log. Otherwise, all the overlapping log entries are accommodated into this new update and removed from the log file. The combined new update is then recorded in the log file. When appropriate the file updates are propagated to the file server as determined by the write-back policy implemented. The oldest entry is always written back first.

The simulator checks the intervals between any two consecutive events to determine if a disconnection occurs. If an expected disconnection occurs, all file updates in the log file are written back right before the disconnection. Otherwise, all log entries will have to remain in the log until the connection resumes. The next connection may have a different bandwidth. The cache management aspects that are adaptive will be adjusted to accomodate this bandwidth. A 24-hour hoarding period is applied for all the experiments, and LRU cache replacement is used both when hoarding and when mobile. Froese[4] found these choices to provide good performance for mobile client operation for the seven-day traces considered.

4.4 The Performance Metrics

Our work has several goals: to reduce the time required to service a client's read and write requests, to reduce the update reintegration time, to limit network accesses, and to make effective use of the cache space at the mobile client. The following performance measures are used in the evaluation of our approaches to cache management.

Time expansion reflects the additional cost to the user of mobile operation, in other words the added time needed to process file system operations remotely. It is computed by dividing the total time required to process all the trace events in the simulated mobile environment by the time required to process the same trace were the user strongly connected. It is affected mainly by two factors: the read service time (the time spent transferring file requests resulting from cache misses over the network) and write interference time (the time read service is suspended waiting for write backs to complete). Time expansion measures how much the mobile environment "slows down" a user's work with the current implementation of cache management.

Update reintegration time reflects the risk associated with delaying the reintegration of file updates at the server. It's the time between a write event at the mobile client and the propogation of the update to the file server for reintegration, and is measured by the time the update remains in the log file. As mentioned earlier, the probability of update conflicts or data loss or damage due to loss or theft of the mobile device or system failures increases as reintegration time increases. The longer the update remains in the mobile client's log file, the greater the risk of these problems.

The *size of the log file* reflects the cost of mobile operation from a different perspective – how much of the client's precious disk space is used for keeping the user's file updates.

4.5 Results for Adaptive Write Back

The merits of adaptive write back are investigated through a simple modification to the *delayed write back* policy. In delayed write back each update is held in the log for a minimum delay time A, after which time it is eligible to be written back to the home file server. The adaptation is realized by adjusting the policy parameter A at different bandwidth levels. The delay time determines the behaviour (the aggressiveness) of the delayed write back policy. With a shorter delay time write backs are performed more aggressively

[2] Bandwidth to the user is affected not only by factors such as the rated capability of the link but also by factors such as congestion in the network, the need to retransmit dropped packets, and so on.

because the updates become eligible for writing back sooner, while a longer delay time makes the policy more conservative.

The performance of the adaptive approach is compared with that of the non-adaptive approach. In the non-adaptive approach, a fixed value of 15 minutes[3] is used for the delay time A, regardless of the available bandwidth. For the adaptive approach, whenever the available bandwidth increases, the delay time A is *decreased* so that write backs are performed more aggressively to take advantage of the increased resource. The impact of the write back traffic on reads should be small since the write backs can be completed quickly when high bandwidth is available. Whenever the available bandwidth decreases, the delay time is *increased* to give read requests higher priority to access the network. This forces the updates to stay in the log for a longer time.

The delay time for each bandwidth is predetermined for this study. For bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps and 2 kbps, the delay time is set to 0.5 min, 0.5 min, 5 min, 10 min and 15 min, respectively.

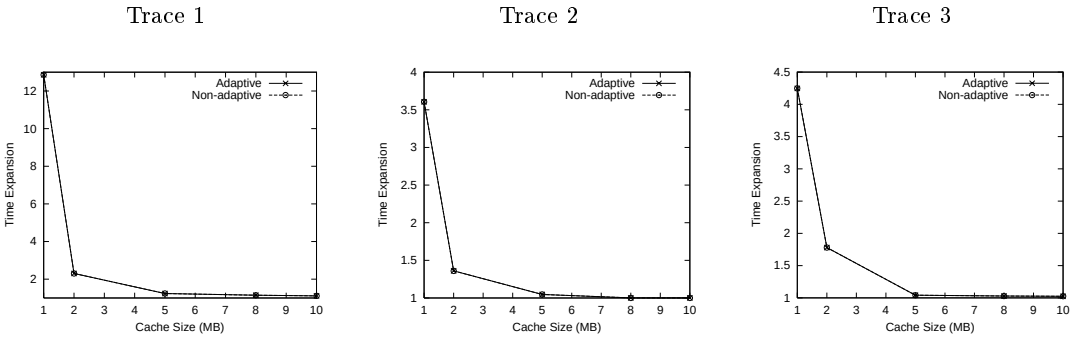

Figure 2: Time Expansion

Figure 2 shows the time expansion for the three traces with regular (non-adaptive) write back and adaptive write back, for different cache sizes at the mobile client[4]. Clearly the size of the cache has a large impact on time expansion for both strategies, since a larger cache can contain more of the user's files and the read service time is reduced with a lower miss ratio. The different approaches to write-back have no significant impact on time expansion, suggesting that propogating updates more aggressively will not degrade the time to service read requests. Because the more aggressive write backs are performed at higher bandwidth, and thus can be completed more quickly, they have little impact on the time to service the reads.

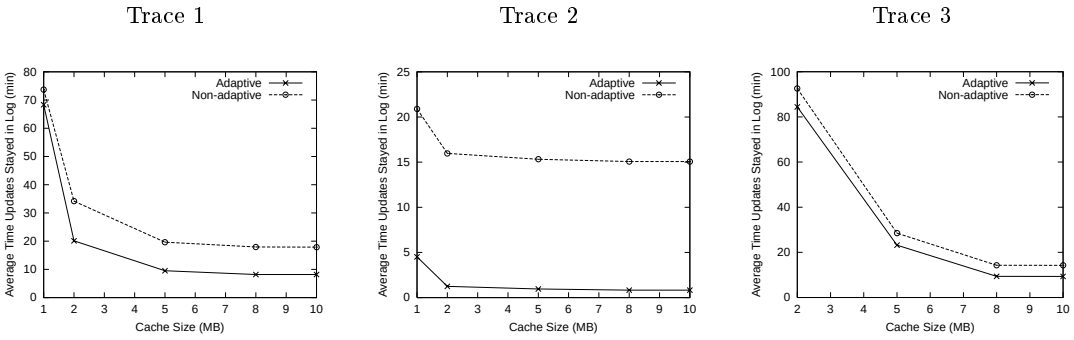

Figure 3: Time Entries Remain in Log File

The primary effect of this adaptation is to reduce the time updates remain in the log file. Figure 3 shows

[3] This value was suggested by Froese [4] based on his experiments with delayed write back.

[4] Although small by today's standards for desktop or even laptop devices, our values for cache size are consistent with the demands presented in the traces and permit comparison with our earlier work. These values are not small, however, in the context of hand-held devices.

how long log entries remain in the log file before they are propagated to the file server, again as a function of cache size at the mobile client. Adaptive write back results in shorter average times than non-adaptive write back. The reduction is particularly striking for the trace with the largest percentage of write requests (Trace 2). When the cache size is large (larger than 2 MB) the reduction is more than 50% for both Trace 1 and Trace 2, and around 20% for Trace 3 (that has the smallest percentage of write requests).

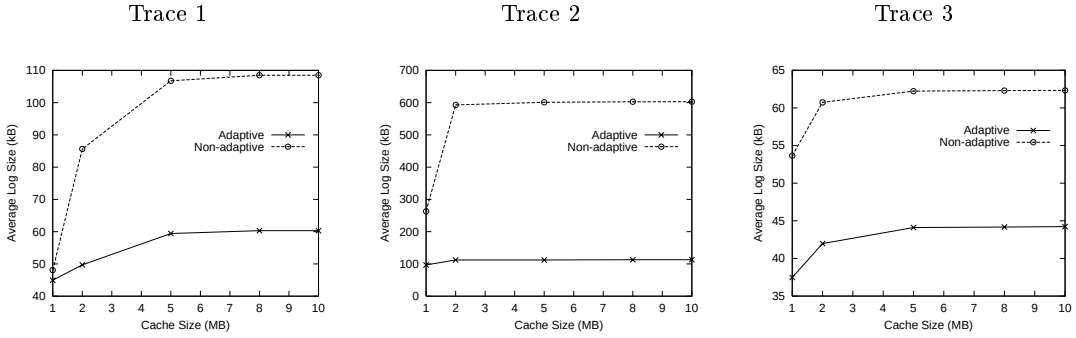

Figure 4: Size of the Log File

Once an update is written back to the file server, it no longer needs to be kept in the log file. Since our adaptive approach writes updates back to the file server more aggressively, it results in smaller log files. Figure 4 shows the impact of different write-back approaches on the average log size. The reduction in average log size with adaptive write backs is large when the cache size is large (larger than 2 MB), close to 50% for Trace 1 and Trace 2 and 30% for Trace 3.

4.6 Results for Adaptive Block Caching

Block caching has several advantages over whole-file caching. Since blocks are smaller than files, block transfers complete more quickly than file transfers and the user can resume work sooner. Block caching is also less demanding on client cache space. On the other hand, if many of a file's blocks are referenced, the necessity of repeated block transfers can offset these advantages. Froese's study [4, 6] showed that block caching becomes more advantageous when bandwidth is limited. He also showed that the size of the block and the bandwidth both impact cache performance. Since a mobile user's connection may experience different bandwidths, adjusting the size of the cache transfer unit dynamically may improve cache performance. The performance of adaptive block caching is studied in this section, and compared to non-adaptive block caching in which a fixed amount of data is brought in to the cache on each miss, regardless of changes in connection bandwidth.

In order to examine the performance impact of adaptive block caching without worrying about the effect of adaptive write back, non-adaptive delayed write back was used throughout[5] with the delay time set to 15 minutes.

For non-adaptive block caching, a block size of 4 kB is used for all transfers. For adaptive block caching, the size of the transfer unit will be increased when the bandwidth increases. The reasoning is that when bandwidth is high, bringing more data will have little impact on the read service time, but will reduce subsequent misses if the additional data fetched is eventually referenced. When available bandwidth decreases, the size of the transfer unit is reduced. This will reduce the read service time at low bandwidth and allows the user to resume working more quickly. The block size at different bandwidths is predetermined for this experiment. The transfer block size is 16 kB, 8 kB, 4 kB, 2 kB, and 1 kB for bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, 2 kbps, respectively.

The impact of adaptive block caching on time expansion is shown in Figure 5. Adaptation results in reduced time expansion over non-adaptive block caching, and the improvement is particularly striking at small cache sizes. In fact, the positive impact on the mobile user is actually even better than what Figure

[5] Subsequent experiments showed that adaptive write back conflicts very little with adaptive block caching, but these results are not reported in this paper. See [11] for details.

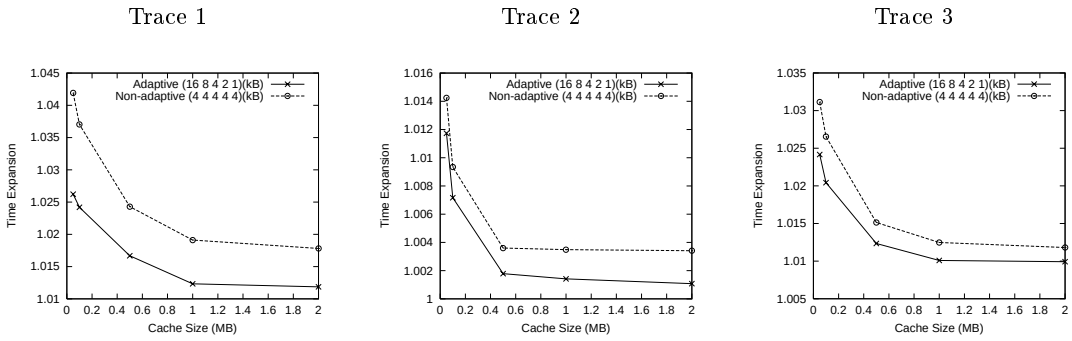

Figure 5: Time Expansion

5 suggests because the "idle time" in each trace is included in the calculation of the time expansion. The actual active time for each trace is much less than the length of the trace, as shown in Table 1. When only the active hours are considered, the time to service actual requests is reduced by 12.5%, 4%, and 5.5% for Trace 1, Trace 2, Trace 3, respectively, at a cache size of 1 MB. With less cache at the client, the reduction is even larger.

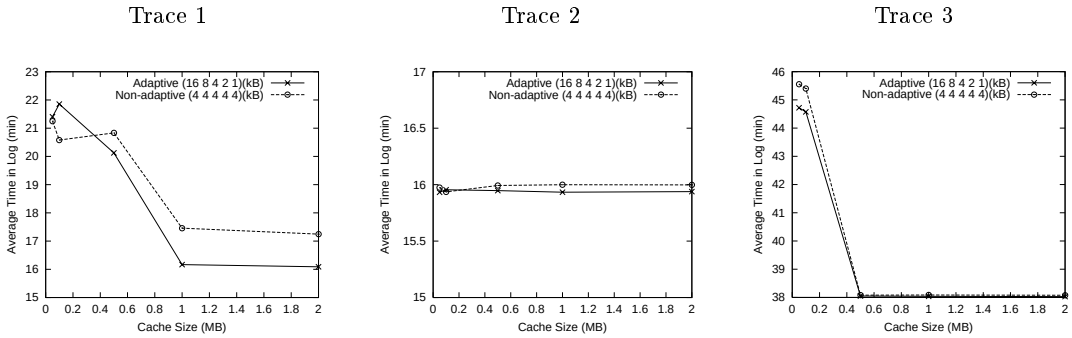

Figure 6: Time Updates Remain in the Log File

The time updates remain in the log file is determined mainly by the delay time parameter in the write-back policy. When the same delay time is used, adaptive block caching has insignificant impact on the time updates remain in the log. This is confirmed by Figure 6. Although the non-adaptive approach has a slightly longer time in log, this is because it takes longer to bring 4 kB blocks into the cache at low bandwidths (as compared to the smaller blocks the adaptive approach uses at these bandwidths), and so some updates will remain in the log file waiting for the read service to complete even though they are eligible to be written back.

Finally, Figures 7 and 8 provide some insight into the extra cost associated with adaptive block caching. Figure 7 shows that adaptive block caching requires almost the same space for the log file, and so the impact on client resources is acceptable. Figure 8 shows that adaptive block caching adds only a small amount of extra traffic to the network. These results suggest that the benefits of adaptive block caching can be provided at little or no extra cost.

5 Conclusions

The ability to adapt dynamically to environmental changes has been identified as an important success factor for mobile computing. We are experimenting with adaptive approaches to file cache management, seeking to understand the extent to which adaptation can provide improved performance for mobile users. Our focus is on two application areas – adaptive write back and adaptive block caching.

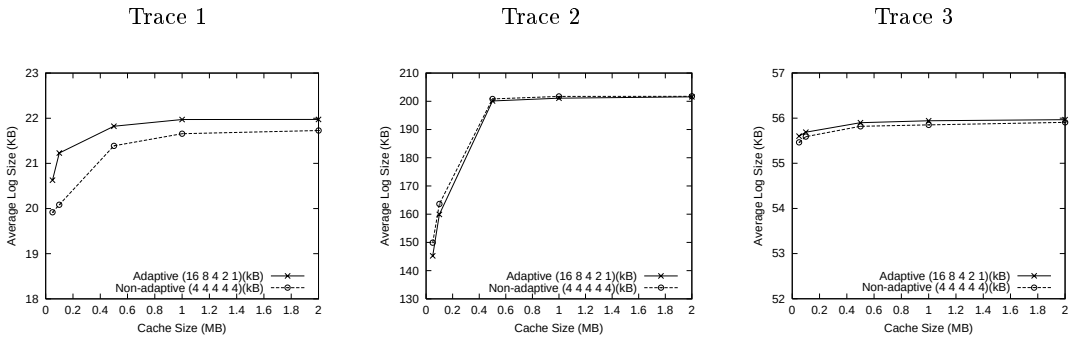

Figure 7: Size of the Log File

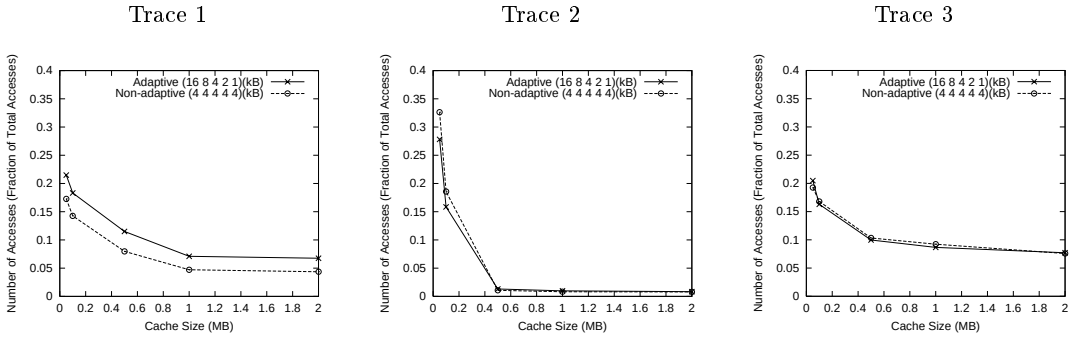

Figure 8: Number of Network Accesses

Results from our simulation experiments suggest that adaptation can be beneficial when appropriate adaptation strategies are applied in the constantly changing mobile environment. Our approaches to adaptive write back and adaptive block caching were successful in improving performance over the non-adaptive approaches. Adaptive write back reduced the update reintegration time for all traces and for all cache sizes considered and, as a result, reduced the demand on the mobile client's disk space for keeping file updates. It does this without negatively impacting the time expansion. Adaptive block caching can also improve performance by dynamically adjusting the amount of data to be fetched as available bandwidth changes. The reduction in time expansion, particularly at smaller cache sizes, is important when the resources at the mobile client are very scarce – as would be the case, for example, when the user's mobile device is a palmtop or PDA.

There may, of course, be negative aspects to adaptation. In adaptive write back, for example, more updates are propagated to the file server and this increases the network load, and adaptive block caching may increase the frequency of accessing the network for demand read requests. Our simulation results, however, suggest that these negative impacts remain at acceptable levels.

On balance our adaptive approaches offer performance benefits with very little extra cost. Although this particular study is somewhat limited in its scope it does add to the growing list of successful adaptive approaches and provides support for further work in this important area.

References

[1] B. Badrinath, A. Fox, L. Kleinrock, G. Popek, P. Reiher, and M. Satyanarayanan. A conceptual framework for network and client adaptation. *ACM Mobile Networks and Applications*, 5:221–231, December 2000.

[2] D. Barbará and T. Imieliński. Sleepers and workaholics: Caching strategies in mobile environments. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Minneapolis, MN, May 1994.

[3] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, April 1994.

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

[5] K. Froese and R. Bunt. Scheduling write backs for weakly-connected mobile clients. In *Proceedings of the Tenth International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 219–230, Palma de Mallorca, Spain, September 1998.

[6] K. Froese and R. Bunt. Cache management for mobile file service. *The Computer Journal*, 42(6):442–454, 1999.

[7] L. Huston and P. Honeyman. Disconnected operation for AFS. In *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium*, pages 1–10, Cambridge, MA, August 1993.

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

[9] J. Jing, A. Helal, and A. Elmagarmid. Client-server computing in mobile environments. *ACM Computing Surveys*, 31(2):117–156, June 1999.

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

[11] J. Mei. Adaptive file cache management for mobile computing. Master's thesis, Department of Computer Science, University of Saskatchewan, Saskatoon, SK, in preparation.

[12] L. Mummert, M. 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, CO, December 1995.

[13] B. Noble. *Application-aware adaptation.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2000.

[14] B. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in Odyssey. *ACM Mobile Networks and Applications*, 4(4):245–254, 1999.

[15] A. Rakotonirainy. Trends and future of mobile computing. In *Proceedings of the Tenth International Workshop on Database and Expert Systems Applications*, pages 136–140, Florence, Italy, September 1999.

[16] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing*, pages 1–7, Philadelphia, PA, May 1996.

[17] M. Satyanarayanan, J. Kistler, L. Mummert, M. Ebling, P. Kumar, and Q. Lu. Experience with disconnected operation in a mobile computing environment. In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pages 11–28, Boston, MA, 1993.

[18] C. Tait, H. Lei, S. Acharya, and H. Chang. Intelligent file hoarding for mobile computers. In *Proceedings of the First International Conference on Mobile Computing and Networking*, pages 119–125, Berkeley, CA, November 1995.

[19] M. Weiser. Hot topics: Ubiquitous computing. *IEEE Computer*, 26(10):71–72, October 1993.