ADAPTIVE FILE CACHE MANAGEMENT FOR MOBILE COMPUTING

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

For the Degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Jiangmei Mei

Spring 2003

Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

Abstract

File service is a fundamental user requirement, and this applies to mobile users as well as stationary ones. Mobile users 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.

Mobile clients face wide variations in resource availability such as bandwidth when accessing remote data. These variations have a large impact on providing performance to mobile users, and require that systems and applications for mobile users be adaptive. Caching files at clients is commonly used in file systems to support mobility. To improve file system performance, file cache management should react to the changes dynamically to take advantage of increased resource availability and to minimize the impact of decreases accordingly. Being able to adapt successfully to changes in resource availability is widely recognized as central to the support of mobile computing.

This thesis addresses adaptive approaches to file cache management in mobile environments. The research deals with cache management strategies that are able to adapt their behaviour in response to changes in bandwidth in order to maintain acceptable performance. The focus is on adaptive approaches to file updates and adaptive adjustment of the caching unit – adaptive write back and adaptive block caching, respectively. The results of a trace-driven simulation study of adaptive approaches are presented and demonstrate that adaptive approaches to file cache management can improve file system performance for mobile users at acceptable cost. The thesis also explores the conflicts between adaptive write back and adaptive block caching. The simulation results show that the conflicts are insignificant.

Acknowledgements

This thesis would not have been possible without the support of many people during my Master's study. I would like to take this opportunity to express my appreciation.

First of all, my supervisor, Professor Rick Bunt, for his constant support, guidance and encouragement, and for helping me keep this work in perspective.

Thanks also go to my thesis committee members, Professor Tony Kusalik, Professor Kevin Schneider, and Professor David Klymyshyn (External), for their valuable comments and suggestions.

Funding from Professor Rick Bunt, Telecommunications Research Laboratories (TR*Labs*), and the University of Saskatchewan (Graduate Teaching Fellowship) made it possible to pursue my graduate education.

Thanks to Greg Oster, Jan Thompson and other people in the department and Telecommunications Research Laboratories (TR*Labs*) for providing necessary assistance on numerous occasions.

Finally, I would like to express my special thanks to my family. My parents have been always supportive in my education and always there when I need them. Special thanks to my husband Bin for his incredible support and encouragement during the program. Thanks go to my son David, who is young but also supportive.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AFS	Andrew File System
ARDIS	Advanced Radio Data Information Service
CDPD	Cellular Digital Packet Data
CML	Client modify log (from the Coda file system)
CPU	Central Processing Unit
HDB	Hoarding database (from the Coda file system)
ISDN	Integrated Services Digital Network
LAN	Local Area Network
LFU	Least Frequently Used (replacement strategy)
LRU	Least Recently Used (replacement strategy)
kB	kilobytes (1 kilobyte = 1024 bytes)
kbps	kilobits per second (1 kilobit = 1000 bits)
MB	MegaBytes (1 Megabyte = 2^{20} bytes)
Mbps	Megabits per second (1 Megabit = 10^6 bits)
NFS	Network File System
PDA	Personal Digital Assistant
PFS	PRAYER File System

Chapter 1

Introduction

Advances in portable computing devices and networking technologies have made possible a new paradigm of computing: *mobile computing*. Mobile computing allows computer users to access data and information services anywhere and anytime, independent of their physical location and movement behaviour.

While mobile computing provides great convenience for users who want to work out of their home offices or on the road, there are difficulties with mobile computing. Mobile users expect comparable quality of service as they would receive in traditional stationary computing, but variations in network environment and the limitations of mobile devices may make this difficult to achieve. In a stationary computing environment, all computers are connected together by Local Area Networks (LANs) with 10 Mbps or higher bandwidth; however, in a mobile computing environment, the nature of the connection to the network may change. Mobile devices may connect to the network at different times and at different places, through either wired or wireless connection that is available at their current locations. Sometimes, there may be no connection available at all, and the mobile devices have to be operated in a disconnected mode.

The limitations of mobile devices pose challenges to mobile computing. Mobile devices that support the mobility of the users, including laptops, notebooks, palmtops, PDAs, all have different capacities in CPU speed, memory size, and disk space. For ease of carrying, they are smaller in size and lighter in weight than stationary desktop computers [43]; hence they typically have limited CPU speed, main memory capacity, disk space, and battery life.

Users working in a mobile environment expect to use their mobile devices in the same manner as their stationary computers without being aware of the varying network conditions or the constraints of the mobile devices. They demand both mobility and performance without having to trade one for the other. To meet mobile users' requirements, even with advances in hardware (mobile devices and networks), the current system software for stationary computers needs to be modified or even redesigned for mobile devices [25, 40, 43].

A file system that supports mobility is one of the most important components in mobile computing [6, 9, 18, 40]. Such a file system should provide mobile users with functionality and performance comparable to that of a Local Area Network (LAN) environment, including continued read and write accesses and timely update reintegration with a file server [9]. To ensure mobile users' read and write accesses, caching is usually employed at the mobile client to increase the data availability and to improve performance [9, 23, 18, 30, 44]. Caching of frequently used files can also reduce the latency of file access, since some of the files in the local cache are used instead of being transferred remotely every time. When the client is disconnected from the server, caching is the only way to provide file services [45].

To provide support at the system level, the cache should be managed carefully. Cache management in a file system for mobile computing provides functions including hoarding files to prepare for mobility, servicing file system requests when mobile, and propagating updates to the home file server for reintegration [9, 29]. Different cache policies are used to manage the cache, including the hoarding policy, the replacement policy, the prefetching policy, and the write-back policy. The hoarding policy decides which file(s) should be brought into the cache prior to going mobile. The replacement policy determines which file(s) should be evicted to make room for newly brought-in file(s) when the cache is full. The prefetching policy predicts the future file reference(s) and fetches the corresponding file(s) into the cache when mobile to reduce response time. The write-back policy manages the propagation of file updates back to the home file server. Each of these policies directly affects file system performance and needs to be designed efficiently for mobile systems.

Mobile users require mobility and performance. A file system for mobile users must provide the users with the functionality they require while keeping the performance impact of mobility to a minimum. In a mobile environment, resources available to a mobile client, such as bandwidth, can change dynamically and dramatically. To improve file system performance, cache management should take advantage of increased resource availability and react to decreases accordingly. Thus, cache management should be designed to adapt to changes in available resources. This can be achieved through dynamically altering management policies or adjusting parameters at different available resource levels. As shown by Froese [9], different policies and parameters, as well as bandwidth, impact cache performance. Therefore, dynamically altering policies and/or resetting parameters in response to available resources may improve cache performance.

Previous work on file system support for mobile computing has focused mainly on providing functionality to mobile users. Cache management techniques used for traditional computing were implemented to achieve some level of performance. Froese and Bunt [11, 9] thoroughly investigated the performance of file cache management for mobile clients. However, adapting file cache management to changes in the mobile environment has rarely been addressed and the performance impact has not been evaluated. In Mummert [29], the adaptation was implemented only in the reintegration of updates in which the size of the reintegration chunk (holding the updates ready to be propagated to the file server) was adaptive to changes in connection bandwidth.

This research addresses issues in adaptive file cache management for mobile users, specifically approaches to cache management that can adapt to changes in the speed of the connection. An environment is assumed in which a mobile client experiences different types of connections with different bandwidths over the course of a session. The specific research objectives are:

1. to identify opportunities for adaptive cache management.

2. to experiment with adaptive strategies for propagation of file updates to the home file server.

3. to investigate the impact of dynamic adjustment of caching unit.

Froese's system model and performance metrics, as well as his simulator were used to carry out a simulation-based study to evaluate the performance benefits and negative impact of these approaches.

The remainder of this thesis is organized as follows. Chapter 2 provides background information for mobile computing. Chapter 3 discusses file system support for mobile computing and issues in file cache management for mobile computing. Existing file systems that support mobility are also introduced in this Chapter with emphasis on their file cache management strategies. Chapter 4 describes issues in adaptive file cache management and the simulation design for performance evaluation. The results of the performance study are presented in Chapter 5 and Chapter 6. Finally, Chapter 7 summarizes the thesis research and discusses possible areas for future research.

Chapter 2

Mobile Computing

This chapter starts by providing background information about mobile computing and the definition of mobile computing used in this work. The requirements of mobile users and the communication technologies available for mobile users are presented. Then the constraints in mobile devices and network conditions are described. Finally, the need for adaptation in the mobile environment is discussed.

2.1 Mobile Computing

Different directions have been taken to support mobile computing: location-independent computing [32], nomadic computing [20, 24, 38, 26], ubiquitous computing [50, 51], etc. Location-independent computing allows mobile users, with their portable devices, to work at different locations (at different times) without being aware of the location changes. Nomadic computing attempts to make the computing environment independent of users' characteristics for nomadicity, including the location the users are at, whether or not they are in motion, and the computing platform they are using. Ubiquitous computing seeks to make computers available throughout the physical environment and effectively invisible to users. In ubiquitous computing, computers may be embedded into things such as clothes, walls, pencils, etc., and all the computers are interconnected through wireless networks. Each person can be served by many computers everywhere in the world.

For this research, mobile computing is defined as follows. A mobile user performs work on his/her mobile computer at different locations, such as office, home, hotel, branch office, conference room, etc. At different places, the user may connect his/her computer to the home file server using whatever communication method is available at his/her current location. Each of these connections may have a different type and quality of service. Regardless of his/her location and the network conditions, the user expects to have the same working environment, with comparable functionality and performance achievable in a stationary computing environment.

2.2 Requirements of Mobile Users

Mobile users require functionality and performance comparable to what they would receive in a traditional computing system. Functionality includes network connectivity, data availability, access to services, etc.

- *Network Connectivity.* Network connectivity provides a means for mobile users to transfer data to/from home servers. Although some systems maintain a cache to support operations without network connection, cache misses are inevitable and they suspend or even halt users' work. On the other hand, keeping cache contents up to date and propagating file updates need network connections; otherwise, stale data returned to mobile users may cause problems. For example, an out-of-date schedule is of limited value.

- *Data Availability.* Data availability is an important issue in mobile computing. The work a mobile user is performing may be suspended if required data is not available.

- *Access to services.* Mobile users would like to have access to the variety of services and resources being enjoyed by users in a traditional computing environment, such as file services, database services, mail services, internet services, etc.

Performance expectations include reasonable throughput, reasonable response time, transparent mobility, etc.

- *Throughput.* This is the measure of how much work is performed. Even in the presence of low quality of service in network conditions, mobile users expect to perform the same amount of work as they do in a traditional computing environment.

- *Response Time.* The response time is the time from the submission of a request until the response begins to be received. In a mobile environment, the response time can reflect the users' satisfaction with the network. Long response time is distracting to users and is usually not tolerated.

- *Transparent Mobility.* Mobile users require that their movement from one location in the system to another make as little difference as possible to the way they interact with the system. Transparency relieves users from the need to be constantly aware of the details of their computing environment, thus allowing them to focus on their real work.

2.3 Communication Technologies

The capability of mobile clients to access data and information anywhere and anytime is supported by advanced communication technologies. The following are some methods used to connect mobile devices to the rest of the world [8, 20, 42, 37, 22]:

- *Wired or Wireless Modems.* Modem connectivity provides global coverage via dial-up telephone networks with relatively low bandwidth, typically 9.6 kbps for wireless modems, and ranging from 9.6 kbps to 128 kbps for wired modems.

- *Ethernet LANs.* An Ethernet LAN is a fixed, wired network. It provides bandwidth ranging from 10 Mbps to 1000 Mbps.

- *Wireless LANs.* A wireless LAN is a traditional LAN (e.g., Ethernet) extended with a wireless interface to serve portable computers capable of wireless access. Wireless LANs provide low cost connectivity within a very small geographical area (a few kilometers). Examples include NCR's WaveLan with a bandwidth of 2 Mbps, Motorola's ALTAIR with a bandwidth of 3.3 Mbps, InfraLan with bandwidth of 4 Mbps, and IEEE 802.11b with a bandwidth of 11 Mbps.

- *Wide Area Wireless Networks.* These are special mobile radio networks offered by private service providers. The infrastructure provides a wide area coverage for low bandwidth data services. Examples include Cellular Digital Packet Data (CDPD) with a bandwidth up to 19.2 kbps, Mobitex with a bandwidth around 12.5 kbps, and Advanced Radio Data Information Service (ARDIS) operating at a rate of 19.2 kbps.

- *Satellite Services.* Satellites are interconnected via microwave links and form a global network in space with linkage to the ground. Satellite services provide a wide coverage but high latency and very expensive connectivity with a limited bandwidth. Motorola's Iridium offers connectivity at a bandwidth of 2.4 kbps.

- *ISDN.* Integrated Services Digital Network (ISDN) is an international communication standard for sending voice, video, and data over digital telephone lines or normal telephone wires. ISDN offers connectivity with a bandwidth of 64 kbps.

2.4 Constraints

Mobile computing suffers from many constraints. The constraints are due mainly to the limitations of mobile devices and the variations in the network conditions.

2.4.1 Mobile Devices

There are generally two constraints on mobile devices compared to desktop computing devices – they are resource-poor and they are less secure [36, 43].

For ease of carrying, mobile devices are required to be 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. These resource constraints are inherent to mobile devices and technological progress will not eliminate the difference between mobile devices and desktop computers. As the technology for increasing the resources for mobile devices is improved, the same advance may have been developed on desktop computers. Thus, mobile devices will always suffer from limited local resources.

Mobile devices also suffer from limited security. Portable devices accompanying mobile users when traveling have a higher risk of physical damage, loss, theft, and unauthorized access than desktop computers [8, 43].

Resource and security constraints imply that mobile users cannot rely entirely on their mobile devices for data supply and data storage. Data should be kept primarily on servers. Data of interest may need to be accessed remotely from servers, and important updates should be transmitted to servers as soon as possible for stronger security.

2.4.2 Network Conditions

It is impossible to provide a mobile client with a certain type and quality of service network connection at all places the mobile user travels. A mobile client may experience considerable variations in network conditions, such as changes in bandwidth and quality of service. As a mobile user changes locations, he/she has to use different communication technologies available at his/her current location to connect to other computers, such as a home server. At different places or different times, a mobile user may choose to use different types of connection for reasons such as service cost, data rate, etc. As was shown in Section 2.3, the bandwidths of the communication technologies that are available for mobile clients vary from a few kbps (a modem connection) to hundreds of Mbps (in a hard-wired environment). Besides the bandwidth, other factors, such as error rate, latency, and service cost, may also change over time.

The bandwidth and quality of service in a certain connection may vary dramatically and unpredictably because of the existence of communication noise and congestion. This phenomenon becomes more evident in wireless communications since they are more susceptible to environmental interference, and disconnections may be frequent. Although wireless communication is not necessary for mobile computing, it is a main component. When the user is in motion (walking or in a moving vehicle) and wants to communicate with others, wireless communication is the only option. Meanwhile, when the number of users sharing a cell (radio coverage area) changes, the actual bandwidth available to each user changes as well. As Weiser [51] pointed out, when users are sharing a radio cell, the network bandwidth is divided among the users.

A mobile user may find himself/herself at a place where there is no network connection available at all. Under this circumstance, the mobile user will have to work in a disconnected environment.

2.5 The Need for Adaptation

The constraints in the mobile devices and the network conditions bring challenges to mobile systems and applications. In the constantly changing mobile environment, mobile users will face problems such as high latency of information access and slow update reintegration.

While experiencing wide variations in resource availability, mobile users should receive acceptable performance as much as possible. To achieve this, mobile systems must be able to detect their transmission environment and exploit knowledge about their current situation [22]. Mobile systems and applications should react to changes dynamically to take advantage of increased resource availability and to avoid negative impact of decreases accordingly. Such a reaction, adapting to changes in resource availability, is widely recognized as central to the support of mobile computing [1, 8, 21, 35, 33, 43, 50].

It should be noted that although hardware advances in mobile devices and communication technologies for mobile computing may improve the performance for mobile users, for example, developing mobile devices with large memory and communication technology with high bandwidth, the problems for mobile users would still remain. As the hardware technology for mobile users improves, more advanced technology may have already been applied for stationary ones. The performance difference received by mobile users and stationary users still exists. Another method to deal with the low bandwidth and high latency problems in a mobile environment is to provide context-aware applications. The context usually means the user's personal, current time, current location, etc. The applications modify their behavior depending on the current context to retrieve related information. An example for such a system is a mobile tourist guide like *The Guide Project* [5]. However, this method is efficient only for location-dependent information access.

Adaptation can be defined as the trading of resources, either for other resources or for some measures of accessed data [33]. For example, when local disk space becomes scarce, mobile users may want to prevent local data from overflow by consuming bandwidth to write locally stored data to a server. This is the tradeoff between bandwidth and disk space. In a video player, for example, when the bandwidth is low, mobile users may want to sacrifice the quality to some level to keep the real time viewing. This is the tradeoff between bandwidth and quality of display.

2.5.1 Models of Adaptation

Three adaptation approaches have been proposed [33, 44]: *laissez-faire* adaptation, *application-aware* adaptation, and *application-transparent* adaptation.

In laissez-faire adaptation, applications are solely responsible for coping with the consequences of mobility. The advantage is that no system support is required. The disadvantage is that there is no single point of resource control, and each application should make its own decision of how to adapt to changes in the environment.

Application-aware adaptation supports collaborative adaptation between the applications and the system. This approach allows both the application and the system to be aware of the mobility and react collaboratively. The system is aware of the resource level, notifies the application when the resource level changes and enforces resource allocation decisions. When notified of the resource changes, the application independently decides how to adapt to the changes. Application-aware adaptation supports application diversity and can perform well especially when the applications involve multimedia data such as videos and maps. The Odyssey system [34, 35], developed at Carnegie Mellon University, implements this approach.

In application-transparent adaptation, the system is totally responsible for adapting to changes in the mobile environment. The system hides the differences between stationary and mobile environments for applications and makes mobility transparent to applications. Such an approach provides a central point of resource control and makes adaptation at a single place. Applications working in a stationary environment can also work well in a mobile environment without any modification.

Examples of mobile file systems implementing an application-transparent approach include Coda [23, 29] and an experimental mobile file system at the University of Michigan [15, 17, 19]. Although both file systems maintain a file cache on a client's local disk to alleviate the impact of changes in the mobile environment, adapting cache management to changes in resource availability was not investigated.

2.6 Summary

This chapter presents some background information on which the research is based. User requirements in the mobile environment are identified. Users require functionality and performance such as data availability, access to services, reasonable response time, etc. Constraints on mobile devices and network conditions are identified. These constraints make adaption to resource availability a necessity for meeting user requirements.

Chapter 3

File Systems for Mobile Computing

A distributed file system is essential for a computer with limited storage [15]. A file system for mobile clients must support the user's operation at different types of connectivity. This chapter introduces file system support for mobile computing in a strongly-connected, disconnected, or weakly-connected environment. A discussion of the main issues in file cache management follows. Finally, some existing file systems that provide support for mobile computing, such as Coda [45], Ficus [13], PFS [6], and the file system at the University of Michigan [17, 19], are introduced, with emphasis on their file cache management strategies.

3.1 File System Support for Mobile computing

From the early days of mobile computing, researchers have noted the need to extend distributed file systems to the environment of mobile computing [15, 14, 17, 27], and some success has been achieved. Some file systems that support mobile computing were actually built on top of existing distributed file systems. For example, the experimental mobile file system at the University of Michigan [17, 19] was built on top of the Andrew File System (AFS) [16].

However, a distributed file system that supports mobile devices is different from a traditional distributed file system. In a traditional distributed file system, the computers have substantial computing power, memory, and power sources. They are connected together through fast and reliable wired network links. The locations, connections, and bandwidths among them do not change, the bandwidth and the memory are not scarce resources. When a file system contains mobile devices, the nature of connection to the network may change from time to time. Mobile devices usually have limited local resources, and they may connect to the file server at different times and different places. The network conditions, including bandwidth, error rate, latency, etc., vary considerably and disconnections are frequent. Therefore, the file systems that work well in stationary distributed file systems need to be modified or even redesigned in order to support mobile computing.

As discussed in Section 2.4.2, a mobile client may experience wide variations in their network conditions. A mobile client's connection with a file server can be placed in one of three broad categories [9, 23, 29].

- *Strongly connected.* The client is connected to a file server over a fast and reliable link, such as an Ethernet.
- *Weakly connected.* The client is connected to a file server over a slow link, such as a modem connection or a radio link.
- *Disconnected.* The client has no network connection to a file server.

A file system for mobile computing needs to be able to deal with these three types of network connections. The following subsections discuss file system support in these three environments.

3.1.1 Strongly-Connected Operation

When a mobile client is connected to a home file server over a fast and reliable link (e.g. a 10 Mbps Ethernet link), the client is said to be strongly connected or fully connected. When strongly connected, the mobile client is much like a client in a traditional distributed file system. However, the file system for the mobile client has additional work to do.

Prior to going mobile, when the client is strongly connected, the file system should be able to successfully hoard files into the client cache to prepare for mobility.

After operating in a disconnected and/or weakly connected environment for a period of time, the mobile client will eventually resume to strong connection to the file server. At this time, the file system has to synchronize the client's cache, propagate file updates that are still in the client update log, resolve conflicts, and return to hoarding state.

3.1.2 Disconnected Operation

In a mobile environment, a client may frequently experience disconnections, either expected or unexpected [43]. An unexpected disconnection (also called an involuntary disconnection) occurs when there is a temporary impediment to communication such as a network or system failure. An expected disconnection (also called a voluntary disconnection) happens when a user deliberately isolates from a network. This may happen when no network connection is available at the current location, or when the user temporarily does not want network support for his/her work.

To support disconnected operation, the file system should be able to ensure the availability of a user's files. This is usually achieved by utilizing automatic file replication management techniques [9, 23]. Central to these techniques is the caching of files at a mobile client. Prior to disconnection, the mobile device should prepare itself by hoarding files into its local cache. To effectively use the limited size of the cache, the cache should contain files that a user will work on in the future. Usually, which files to be cached is determined through the client's reference history (most recently used or most frequently used files) [23, 17], consulting the user for advice (providing a priority list of files) [23, 29], or using automatic caching techniques [13, 45, 48].

When disconnected, a client can rely only on the files available locally in the cache. A client's request for a file that is not in cache (a cache miss) can not be serviced since the file server cannot be contacted. The application from which the cache miss occurs will be suspended until the connection is reestablished and the requested file is transferred to the client.

File updates at the mobile client are recorded in a log file. The propagation of updates to the file server for reintegration has to be suspended until a connection is reestablished. Log optimization (optimizing of the log file) must be implemented to remove the redundant updates and to combine the overlapped file updates. This helps keeping the size of the log file small, therefore, reducing the consumption of scarce local resource for storing file updates. As a result, log optimization also reduces the time for reintegrating updates with the file server at reconnection.

A file system must implement an *optimistic replication strategy*, as opposed to a *pessimistic replication strategy*, to support disconnected operation [17, 23]. A pessimistic strategy allows only one copy of a file to be updated at a time. This is not suitable for a file system to support disconnected operation, since the file server can not be contacted. An optimistic strategy allows all the replicas at different computers to be modified independently. It places data availability above consistency. *Write-write conflict* may happen if the replicas of a file, at different computers, are modified at the same time. Although the consequences can be serious, studies have shown that such conflicts happen only occasionally [17, 23].

3.1.3 Weakly-Connected Operation

Disconnected operation allows a mobile user to continue his/her work when no file server is accessible. However, a mobile user working in a disconnected mode suffers from many limitations. For example, cache misses, which are inevitable [19, 29], suspend the user's work; file updates recorded in the log file cannot be propagated to the file server for reintegration, which not only consumes the local resource in the mobile computer, but also causes possible write-write conflicts, and risks damage or loss of important data.

Lying between strongly-connected and disconnected operations is a mode of operation called weakly-connected or partially-connected operation. If a mobile client is connected to a file server over a slow, expensive, and intermittent link, the client is said to be weakly connected. There are a range of weak connections a mobile client may experience, in terms of connect bandwidth, error rate, and latency. For example, the mobile client may connect to the file server through a wireless LAN (2 Mbps), a modem link (wired or wireless, with a bandwidth up to 56 kbps), or an ISDN link (64 kbps).

In a weakly-connected environment, whenever the client issues a read request on a file, the local cache is searched for the file. If the file is already in the cache, then the request is serviced by the local cache. If the file is not in the cache, then a network access to retrieve the file is requested. The file will be transferred to the client over the weak connection. When a client issues a write request, the file update is recorded in the log file, which is the same as in a disconnected environment. Log optimization should still be implemented to preserve its benefits for the mobile user. The update may be later propagated to the file server for integration as determined by the write-back policy implemented.

Having even a weak connection has many advantages. Cache misses can be serviced over the weak connection and are no longer fatal. The mobile user's work being performed is suspended for only a short time until the requested data is transferred to the mobile client. File updates can be propagated to the file server over the weak connection for reintegration in a timely manner. Timely propagating of file updates back to the file server further reduces the size of the log file and the possibility of write-write conflicts.

When mobile, the client may experience a wide range of connection speeds, even during a single session. The file system may take advantage of high bandwidth by using the network more aggressively to improve performance. For example, the file system may fetch data needed in the near future from the server and propagate file updates to the file server more aggressively. When only low bandwidth is available, the file system should attempt to minimize the impact on performance by using the network only when it is necessary. This is the focus of this research.

3.2 File Cache Management for Mobile Computing

Caching techniques have been implemented in traditional distributed file systems to improve performance. Since some files can be obtained locally, caching files at the client reduces network traffic, server load, and the latency of the client's file accesses. It is faster to get a file from the local disk than to obtain it from a remote server. The key issue is to manage the client's cache efficiently to reduce the need to access files across the network. The management of a client's file cache in traditional distributed file systems has been studied and evaluated extensively [2, 16, 31, 41].

Although a mobile computing environment is significantly different from a traditional distributed computing environment, the strategies for cache management in a distributed file system have been used as starting points in building cache management for file systems in mobile computing [19, 29]. However, the same cache management strategies may not be effective for mobile clients [9, 45, 47]. Some assumptions for building cache management strategies in a traditional computing environment may no longer be valid in a mobile environment. In a traditional computing environment, there are fast and reliable links from clients to servers and the client computers have sufficient memory and disk space. This is not the case in a mobile environment. Mobile computing designs must cope with limitations and variations in local resource availability and network conditions [39, 3]. It may thus be necessary for the cache management strategies for traditional distributed file systems to be modified or redesigned to adapt to changes in a mobile environment.

In mobile computing, caching plays a more important role than in traditional computing in providing mobile users with data availability and performance. At most of the time, the file server can not be contacted if the client is disconnected, or can only be reached through a slow and error-prone link when the client is weakly connected. Maintaining copies of a mobile user's files in his/her local cache allows the user to continue to work when the client is disconnected from the file server, and improves the performance when the client is weakly connected. Caching frequently accessed files not only provides quick read/write access and reduces access latency, it also reduces the server load and bandwidth consumption. In most of the file systems supporting mobility, such as Coda, Ficus, AFS, etc., caching is implemented to improve file system performance and data availability.

File cache management is the key to providing acceptable performance. A cache manager uses cache policies to decide which file or data block to fetch, to store, and to write back. Effective cache policies attempt to predict future file references, so that the cache can provide fast response to future requests. Well-designed policies can greatly improve cache performance. In mobile computing, a cache design includes a number of elements, such as a hoarding policy, a prefetching policy, a replacement policy, and a write-back policy. Other issues in cache design include the size of the cache, the choice of caching unit (whole-file caching or block caching), the mechanism for maintaining cache consistency (keep cache content up-to-date), and techniques for log file optimization. Detailed discussion of these issues follows.

3.2.1 Hoarding

Hoarding is a process to bring data (files or data blocks) to a client's local cache prior to going mobile, while the client is still strongly connected to the file server. Hoarding is necessary in mobile computing for preparing for the user's upcoming disconnected operation or weakly-connected operation. The client relies on the cache to satisfy its file requests when disconnected, and to provide quick response when weakly connected. A cache miss on a mobile client can mean a substantial cost in time and money, and may suspend or even halt the user's work if critical information is not in the cache. By transferring data to a mobile client's local cache in advance of going mobile, hoarding can overcome challenges such as low bandwidth, high latency, and frequent disconnections in the mobile environment; hence it improves access performance.

The basic idea of hoarding is to predict the user's future data needs. How to successfully predict the information a user will need is a challenge; otherwise, scarce resources, such as mobile device disk space, are wasted by storing information that the user will never reference. System performance will be decreased if important data is not cached and has to be transferred over a slow and expensive link.

The most commonly used and easily implemented method for hoarding is to use a client's reference history and bring previously referenced data to the cache [17, 19]. The user can also be involved in the hoarding decision by providing a list of priority files. The Coda file system [23, 29] implements a hoarding policy which combines the mobile client's past history and the user's explicit information. In the SEER predictive caching scheme [12], files are hoarded without the user's intervention based on a measure called *semantic distance*. Semantic distance determines how closely related those files are. If one file is referenced and needs to be brought into the cache, then the closely related files are also brought in together. Tait et al. [47] proposed an intelligent file hoarding scheme called *transparent analytical spying*. Transparent analytical spying detects the file working sets for individual applications by analyzing a user's file accesses. File working sets are hoarded in the cache on a Least Recently Used (LRU) basis.

3.2.2 Prefetching

Prefetching is a well-established technique to improve performance in traditional distributed systems. This technique has been utilized in mobile computing to improve access to remote file systems. In mobile computing, prefetching is an ongoing process that seeks to bring soon-to-be-referenced data (files or blocks) into a client's local cache during less than ideal network connections when the link is not busy. Prefetching improves cache performance by reducing response time of the client's file requests when the data requested is in the cache instead of transferring it remotely from the server over a slow link.

Prefetching is different from hoarding, although both strategies attempt to predict a user's future data needs and to bring data to a mobile client's local cache. Prefetching concerns near-future data access, while hoarding seeks to provide availability over a longer period. Hoarding is carried out while the client is strongly connected to the server. The decision of which files or blocks to bring into the cache has little impact on the client's other requests, since the data transfer is fast. However, this is not the case in prefetching. When prefetching, the mobile client and the server are weakly connected and data transfer is slow. Decisions concerning which files or blocks to prefetch and when to retrieve them over the slow and expensive links should be made carefully; otherwise, precious bandwidth would be wasted and demand data fetching, for which the application is suspended, would be delayed. Ideally, prefetching should use only network idle time, should not interfere with demand fetching, and should reduce cache misses.

Previous work on prefetching has relied on analyzing the client's past reference pattern [28, 7]. Prefetching is usually invoked when there is a cache miss. Instead of bringing in only the missed data, more data will be brought in as well, to the extent that this data is then referenced. For example, when a file needs to be brought to the cache, other files that are closely related to this file can be taken in together, or when a data block is demanded, neighbouring blocks can also be brought into the cache together. Prefetching can also be invoked by the user.

There are many other benefits from prefetching, beyond reducing response time of file requests. When more data is brought to the client each time, prefetching can also reduce the number of network accesses and reduce server load. Prior to experiencing poor connection or disconnection, prefetching critical data to the cache in advance may allow the user to continue working in a severe network environment.

Meanwhile, improper prefetching or wrong decisions regarding which files or blocks to prefetch can be harmful. Prefetching may interfere with demand access. When prefetching is in progress, and there is a cache miss from an application, the user's current work has to be suspended until the prefetching is completed. On the other hand, prefetching data to a cache may result in the eviction of other data to make room for the newly prefetched. The evicted data may be subsequently referenced and has to be retrieved from the server. Further, if the prefected data is not referenced in the near future, or it is evicted out of the cache before being referenced, the time and bandwidth are wasted. These problems become severe when the connection bandwidth is low. Therefore, prefetching in mobile systems should be treated more carefully than in traditional distributed file systems to keep the overhead of prefetching low.

The amount of data to prefetch may be dynamically determined to utilize available resources and to reduce communication cost. When high bandwidth is available, performing prefetching more aggressively (fetching more data) may further reduce the response time and latency. When only low bandwidth is available, prefetching should be performed less aggressively to reduce the interference with demand fetching and to allow the suspended application to resume work more quickly.

3.2.3 Replacement

The size of a client's cache is always limited. When the cache is full and new data needs to be brought in, one or more items in the cache must be removed to make room for newly-brought-in data. This is done according to the replacement policy. A good replacement policy will select cached items for removal which will not be subsequently referenced.

Most distributed file systems implement an LRU replacement policy. LRU has proven to perform well in traditional file systems. Existing mobile file systems, such as the Coda file system [23, 29] and the mobile file system at the University of Michigan [17, 19], have implemented an LRU replacement policy. Froese [9] investigated the performance of various cache replacement policies in a simulated mobile environment, including LRU, Least Frequently Used (LFU), Keep Large, and Keep Small. He concluded that LRU performed consistently well.

In a mobile file system, different replacement approaches may need to be developed. Users' advice may need to be sought in order that critical files be exempted from removal. Another approach is to treat files as groups for individual applications, as used in the SEER predictive caching scheme [12].

3.2.4 Write Back

Write backs are the propagation of file updates from a client back to a file server for integration. The write-back policy determines how and when modifications of files are synchronized with the file server. This policy has a critical effect on file system performance and reliability. One of the advantages of a distributed file system is sharing information [7]. If updates are not propagated from a client, stale data may be returned to other clients and effective sharing may not be achieved.

In a mobile environment, when a client is weakly connected, read requests should be given the highest priority to access the network, because read requests suspend the progress of the user's work. Write backs are also desirable and have many benefits. Propagating updates reduces the size of the file update log since updates that are written back no longer need to be kept in the log file. The possibility of write-write conflict is reduced if updates are reintegrated with the server in a timely manner. Finally, in terms of security, write backs reduce the possibility of data loss and damage if the updates are propagated promptly.

Write backs should not interfere with a user's other work, and so write backs should be scheduled intelligently [10]. Traditional distributed file systems have implemented write-back policies such as write-through, delayed write back (Sprite network file system [31]), and write-on-close (Andrew file system [16] and NFS [41]). A write-through policy writes file updates to the server as soon as they happen. A delayed write-back policy writes file updates in a log file at the client and writes them back to the file server some time later. A write-on-close policy writes data back to the server when the file is closed. Write-through and write-on-close may not be successful in mobile file systems, because the server cannot be contacted when the client is disconnected. Also, when the client is weakly connected, write-through produces heavy traffic and will not be efficient [31].

Delayed write back has performance advantages [9, 10, 29] and has been implemented in mobile file systems such as Coda and the file system at the University of Michigan [23, 29, 19]. By delaying writing back, overlapped updates can be accommodated into a single update. Froese [9, 10] evaluated five write-back policies including delayed write back, idle write back, preemptive write back, lotto write back, and no write back in a simulated mobile environment and found that the performance benefits of delaying write backs were substantial.

Having a write-back policy that adapts to environment changes can improve file system performance. For example, in a delayed write-back scheme, the aggressiveness of the policy may be dynamically matched to different resource levels to achieve better performance.

3.2.5 Caching Unit

The caching unit concerns whether the cache entry is a whole file or a data block. In the case of data blocks, it concerns the size of the block. The choice of caching unit can also impact cache performance [9].

Some mobile file systems, such as Coda and Ficus, implement a whole-file caching strategy. When a file is referenced, the whole file is transferred to the client's local cache. Whole-file caching has some advantages. Whole-file caching is the only way to support a disconnected client [9]. After a file is transferred to the cache, no more server interaction is needed until the file is closed. This reduces the load on both the network and the server. Whole-file caching is more likely to provide high availability since entire files will be available, instead of isolated data blocks.

In a mobile environment, when the user's files are large and the size of the cache is limited, whole-file caching causes problems. When files are large, the cache can only contain a small portion of the user's files, and cache replacement may be frequent and more transfers may be needed. On the other hand, transferring a whole file takes a longer time. With the increased latency, the client must wait for a longer time to resume work. If the major part of a file is not referenced in the near future or is not referenced before it is evicted, time, bandwidth, and power are wasted. These problems become more serious when only low bandwidth is available.

Block caching is more flexible than whole-file caching [9, 46]. When a file is referenced, one or more blocks that the client actually needs are transferred to the client's local cache. It takes less time to transfer part of a file than the whole file, and the user is better served by a shorter waiting time. The cache can contain more files in terms of data blocks. It allows the user to work with files larger than the size of local cache. However, the other parts of a file may be referenced subsequently, and this will require additional data fetches.

Froese's study [9] showed that there are several advantages to using block caching. In comparison to whole-file caching, block caching reduces response time, the size of the update log file, and the time the updates remain in the log. In block caching, the size of the block impacts performance. A small block size reduces latency but increases the number of file transfers. The performance with a large block size is likely to be close to that of whole-file caching. One possible solution to this problem is to make block size adaptive to bandwidth changes.

3.2.6 Cache Consistency

Caching files at a client using an optimistic replication strategy introduces cache consistency problems when files are shared. If a file cached at a mobile client is modified in the server by other users, stale data may be returned to the mobile client if the client cache is not updated accordingly. In mobile computing, it is harder to keep cache content up-to-date, since mobile computers may connect to file servers over slow and expensive links and sometimes even disconnect and cannot be contacted.

When a mobile client is disconnected from the file server, the file server cannot be contacted and the cached files must be considered as up-to-date. After the connection is resumed, the files in the client's local cache should be validated and synchronized with the file server. Such a scheme has been implemented in Coda [23, 29]. Both the client cache and the server maintain version stamps for each file. When a file is updated, the server increments the version stamp of that file. When the connection is reestablished, the client presents version stamps of files in the local cache to the server. If the version stamp of a file is not valid, then the file in the client is discarded. Subsequent reference to this file should retrieve it from the server.

In a weakly-connected environment, a mobile client can rely on *callbacks* to maintain cache consistency [29]. When a client caches an object, the server remembers it and promises to notify the client when the object is updated by another client. When notified, the client discards the cached copy of the object. In the Ficus peer-to-peer model [13, 12], whenever an update is made to a replica, Ficus attempts to notify all other replicas of the update. The other replicas then fetch the latest version.

3.2.7 Log Optimization

The update log is used to record file updates when a mobile client is disconnected or weakly-connected. The logged file updates may be propagated to the file server when possible. The size of the log file implies the usage of local resource for storing file updates, and the bandwidth and time needed for the propagation of these updates. Since mobile devices are resource-poor, the client's disk space should be used efficiently. The update log should be kept small to give more space for other uses. The bandwidth is the precious resource for mobile users and needs to be used wisely.

If each file update is logged as it occurs, many log entries will be redundant and the time needed to propagate the updates will be long. Consequently, more bandwidth will be consumed for the propagation of these updates, and more write-write conflicts will happen. To solve this problem, the log file should be optimized to remove redundant or overlapped updates. For example, when a file is created, and a later operation removes the file, these two operations and other operations on this file between the creation and the removal of this file should all be removed from the log file. If a subsequent update overlaps with a previous update, the previous one should be accommodated into the later one and be removed for the log. Huston et al. [18] found that log optimization can remove as many as 75% of the logged operations in a distributed file system. In Chapter 5 and Chapter 6, it will be seen that log optimization does eliminate many log entries; but even if log optimization is implemented and write backs are performed in a timely manner in a weakly connected environment, the update log size is still large.

3.3 Existing Mobile File Systems

This section introduces some existing file systems that support mobile computing, including Coda, PFS, Ficus, and the mobile file system at the University of Michigan. The emphasis is on their cache management strategies in hoarding, write back, prefetching, and methods for cache consistency.

3.3.1 Coda

The Coda file system [23, 29], developed at Carnegie Mellon University, is a descendant of the Andrew File System (AFS) [16]. The goal is to offer clients continued access to data in the face of server and network failure. By employing a cache at each client, Coda supports disconnected and weakly connected operations.

At each client, a cache manager, called Venus, manages a file cache on the local disk. Venus acts as a file system proxy to make the applications work with no modification. To support disconnected operation, Venus operates in one of three states. When strongly connected, Venus operates in a *hoarding state* and caches data into the local cache of the mobile computer to prepare for disconnection. Venus combines implicit (Least Recently Used (LRU)) and explicit (hoarding database (HDB)) information into a priority-based cache management algorithm. When disconnected, Venus enters the *emulating state* and begins logging updates in a client log file called the *client modify log* (CML). File system requests are serviced by the local cache. Update optimizations are implemented to reduce the size of the CML. Cache misses cannot be serviced and are fatal. Upon reconnection, Venus enters the *reintegrating* *state*, synchronizes its cache with the server, propagates updates, and returns to the hoarding state.

Coda has evolved to support weakly-connected operation [29]. When the client is weakly connected, the file updates are logged as when the client is disconnected, but cache misses are no longer fatal and can be serviced over the slow link. *Trickle charging* and *trickle integration* mechanisms are implemented. Trickle charging is a form of prefetching which retrieves user advised data when the network is idle. Trickle reintegration is the process to propagate file updates to the server asynchronously. To preserve the effectiveness of log optimizations, the file updates are required to stay in the CML for a certain time to be eligible for reintegration. The size of the integration chunk, which holds the file updates that are ready to be propagated to the file server for reintegration, is made adaptive to changes in connection bandwidth.

3.3.2 The Mobile File System at the University of Michigan

The mobile file system in the University of Michigan [15, 17, 19] was built on top of the Andrew File System (AFS) [16]. As the first step, disconnected operation for AFS was developed through modifications to the cache manager in the AFS client [17]. The cache manager for the AFS distributed file system is *pessimistic* and maintains strict consistency guarantees. To support disconnected operation, the cache manager was modified to be *optimistic*, allowing mobile users to update files when they cannot communicate with any file server. In order to avoid the inevitable cache misses to suspend a user's work in the disconnected mode, a "fetch-only" mode was added to request needed data from the server. File updates are reflected in the disconnected cache and logged for later propagation when returning to connected operation. After running disconnected for a short period of time, a network connection must be established to propagate modifications back to the file server.

Weakly-connected (called "partially connected") operation for AFS [19] was based on a version of the AFS client that supports disconnected operation. It was an extension of disconnected operation that resolves cache misses and preserves client cache consistency. A mobile client can take advantage of network availability, no matter what the quality it is, to achieve timely propagation of updates. To prevent the write-back traffic from increasing the response time of interactive traffic such as reads, a priority queuing technique is implemented. Three levels of queuing were used in this system, interactive traffic, other network traffic, and replay traffic (i.e. the write-back traffic). To avoid starvation of the low priority queues, lottery scheduling [49] , which offers probabilistic guarantees of fairness and service, was implemented to schedule different types of traffic. *Delayed write back* was implemented to manage the propagation of the file updates.

3.3.3 PRAYER File System

The PRAYER File System (PFS) was developed at the University of Illinois at Urbana-Champaign. It is a mobility-aware file system specifically designed for partially or weakly-connected computers [6]. PFS also supports strongly-connected and disconnected operations.

In the PFS model, the PFS client, used by a mobile user, interacts with the home computer, the PFS server, which is strongly connected to a file server and replicates shared directories from the server. The portable computer (the PFS client) caches files from the home computer (the PFS server). File consistency is kept between this home computer (the PFS server) and the portable computer. The client-server interactions take place between the home computer and the server. This PFS model with intermediate node for filtering data minimizes the problem of concurrent read/write access between clients and the file server.

The PFS client listens and forwards read and write requests to the PFS server, then the reads and writes are cached in the client. When a file cached in the client is updated, the file system can either send the update to the server or change locally. PFS always considers the server copy of a file to be the latest version and can detect any write-write conflict. The PFS does not resolve conflicts; instead, it notifies the PFS clients of the conflict.

3.3.4 Ficus

The Ficus file system, developed at the University of California at Los Angeles, is a replicated file system, which implements an optimistic replication to achieve data availability [13, 12]. Any file and directory can be replicated with identifiers and version vectors to any Ficus host. This optimistic replication allows the clients to work regardless of their location. Ficus is designed to support partitioned operations and primarily disconnected operations, in which some replicas are only occasionally connected to a network. Ficus implements a peer-to-peer model which allows any two replica file holders to share files and propagate updated versions of the files. This peer-to-peer model allows each client to get data from nearly any other peer client, which improves data availability in the case of failures and improves performance. This peer-to-peer model overcomes the sharing problems as in Coda, in which, clients cannot share files with each other; each client has to contact the home server to service cache misses, and updates are only propagated to the file server. Connected and disconnected nodes are not distinguished in Ficus. It considers the peer clients to be dynamically connected to various degrees that are determined by the network bandwidth and latency.

Ficus implements an automated predictive hoarding system called SEER to automatically hoard files for operations in the case of absence or near-absence of the network [25]. SEER enhances the accuracy of its predictions by considering the user's activities to be composed of projects rather than individual files. It calculates *semantic distance* (relationships) between individual files and feeds this information to a replication system that manages the actual transfer of data.

When an update occurs in a site, all other sites that are accessible are notified and the changes are propagated. The consistency in other inaccessible sites is maintained by a background process called *reconciliation*, which runs on each replica periodically. Reconciliation compares a local replica with a corresponding remote replica, and retrieves any missed update.

Chapter 4

Adaptive File Cache Management

As discussed in previous chapters, file systems are essential for the support of mobile computing. File cache management for mobile clients plays a key role in providing acceptable performance for mobile users. Mobile users rely on the network for file system services. The limited bandwidth and the variations in bandwidth severely affect cache performance. Mobile users suffer from problems such as long latency of file access, slow update reintegration, etc.

Existing file systems for supporting mobility deal with these problems by caching more files at the client side and using better hoarding algorithms to make better decisions on which files to cache, trying to avoid or reduce network accesses when the client is weakly connected or disconnected. But cache misses and network accesses cannot be eliminated. The resource-poor mobile devices can contain only a subset of a user's files. The user may not be able to successfully anticipate his/her activities and future accesses. Some types of data such as dynamic data cannot be taken with the user. In the mobile environment, update propagation traffic always gives priority to demand read traffic to access the network. Without any effort, the update reintegration is very slow.

In response to the dynamics in the mobile environment, the systems for mobile computing should adapt their behavior to the variations in the environment, and attempt to improve overall performance by providing mobile users with variable levels of quality. In turn, file cache management for mobile clients needs to be adaptive.

This research investigates adaptive approaches to file cache management in mobile environments and examines the performance implications of adapting file cache management to environment changes. Since the major source of problems that impact file system performance is the dynamics of the network connections [4], the connection bandwidth is the interest for adaptation for this research.

This chapter investigates strategies in file cache management adaptation and provides a simulation-based design for the performance evaluation of adaptive file cache management. Based on the previous study [9], the system model for this research is developed. Finally, the factors that affect cache performance and the metrics for performance evaluation are discussed.

4.1 Introduction

In mobile environments, many factors that affect cache performance are dynamically changing. Network conditions may change dramatically and suddenly in bandwidth, latency, and bit error rate. Mobile users may have different tasks at different locations. Their workloads, read-to-write ratios or reference patterns, and their requirements may change as different tasks are being carried out. Security and economic issues also complicate problems.

If good performance is to be achieved and the user's requirements are to be met, the cache manager should be aware of these changing factors and make cache management elements dynamically respond to the changes. The following are factors that can change.

- *Bandwidth.* The bandwidth available to a mobile client varies from place to place and from time to time when different type of network connection is used. Even during a connection, the bandwidth can fluctuate over time because of network load or environmental interference. The cache manager should take advantage of high bandwidth but use low bandwidth carefully only when it is necessary. For example, when high bandwidth is available, it may be desirable to write back updates to the file server and prefetch data from the server more aggressively.

- *Workload.* The workload, read-to-write ratio, and reference pattern change when the user performs different tasks or works on different projects. There may be heavy read requests for one project and heavy write requests for another. If there are heavy read requests, prefetching of data may be performed more aggressively to avoid frequent suspensions of running applications and to reduce response time. There may be a total change in the client's reference pattern. For this change, the replacement policy may need to change its algorithm to not remove soon-to-be referenced files.

- *Disk space.* The local storage available to a mobile user may change. When the available disk space becomes scarce, the cache manager should act accordingly to avoid data overflow. One possible solution is to write data (file updates or other data) back to the file server. For example, since the update log consumes disk space, the cache manager could write back more aggressively when disk space is in short supply.

- *Cost.* The communication services are charged in different ways, such as per message, per packet, the length of connection time, when the services are used, etc. It is always preferred to keep service cost as low as possible. Writing data to and reading data from the server could be performed more aggressively when the service is cheap and fast, and when the service is expensive, it should be used only if necessary.

For cache management to adapt to changes described above, there should be a mechanism in the operating system to detect any change in the environment and inform the cache manager promptly. Cache manager cannot make any assumption about future resource availability.

When informed of changes, the cache manager can cope with these changes by adjusting its management strategies. The following aspects can be considered.

- *Write back*. The aggressiveness of write backs can be dynamically set. Write backs can effectively adapt to environmental changes such as bandwidth, the workload, and local disk space. When write backs should be performed aggressively, an aggressive write-back policy such as idle write back can be used. If implementing delayed write back, the parameter A, the minimum delay time, which determines the minimum period of time that updates should remain in the log file, can be set to different values dynamically to make the write backs more aggressive (a small value) or less aggressive (a large value).

- *Prefetching*. The amount of data to prefetch each time can be dynamically set to adapt to bandwidth changes. At a higher bandwidth, more data can be prefetched to reduce the response time. Since transferring data is fast, the impact on demand fetch would be small. When the bandwidth is low, prefetching may need to be turned off to reduce latency.

- *Block Caching*. The size of the data block in a block-caching implementation can be dynamically adjusted when there is a change in the available bandwidth. A bigger block can be brought in when there is a higher bandwidth available, and vice versa. This is different from adaptive prefetching, although in both strategies, the amount of data to be brought into the cache is dynamically changing. Prefetching may be performed in background and using network idle time. The performance will be improved when the prefetched data will be subsequently referenced and the prefetch process does not interfere with demand read requests. In adaptive block caching, it concerns the required data. When the size of the block is set to a big value (when the bandwidth is high), the required data is fetched with other neighbouring data that is not demanded. When the size of the block is set to a small value (when the bandwidth is low), if the required data is bigger than the block size, a few network accesses may be needed to transfer all the required data.

- *Replacement*. The approach to replacement can be modified to match changes in the user's workload and reference pattern. Different replacement algorithms may be used to respond to changes in the user's reference pattern to increase cache hit rate. The cache manager need to be able to analyze the user's reference patterns and use alternative replacement policies accordingly.

4.2 System Model

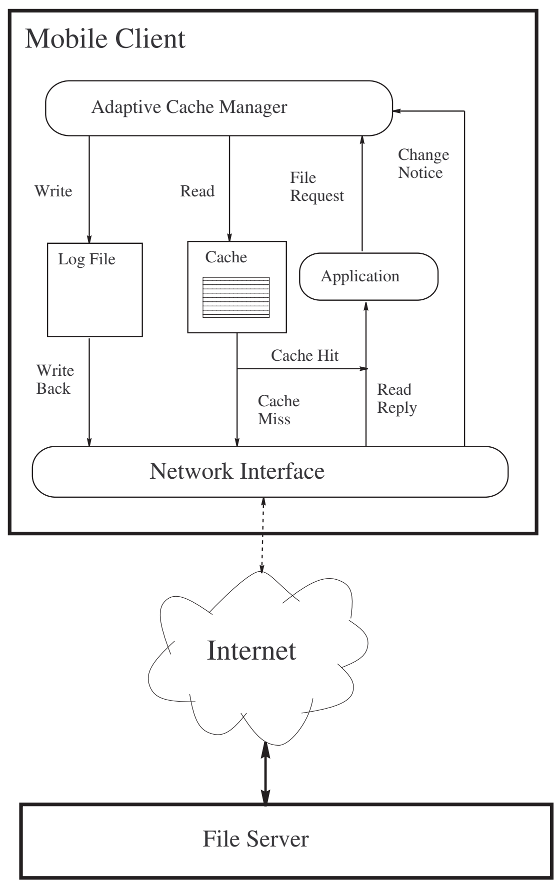


Figure 4.1: System Model

This research extends Froese's study [9, 10, 11]. His system model for a weakly-connected mobile environment is modified for this research. The extended system model is shown in Figure 4.1. The mobile client is assumed to contact a single logical home file server for its file system requests. The functions of the main components and the interactions among them are described below.

- **Mobile Client**. The mobile client consists of a cache, a log file, a network interface, and an adaptive cache manager.

- **File Cache**. The file cache has a fixed size and contains copies of files or file blocks that the mobile client hoarded, referenced, or prefetched. When the client is in mobile, if the requested data is in the cache, the file cache can provide quick responses; otherwise, the file cache requests a network access to retrieve the requested data and the new data is recorded.

- **Log File**. The log file records file updates when the client is weakly connected or disconnected. When log optimization is implemented, it offers a place for removal or overwriting of redundant file updates to reduce the number of file updates and bytes to be written back. Log entries may be propagated to the file server when possible. The log file is maintained in a first-in first-out manner.

- **Network Interface**. The mobile client's interactions with the file server for transferring data pass through the network interface. The network interface is capable of accessing different networks, since the mobile client may use different network access points for connecting to the file server.

- **Adaptive Cache Manager**. The cache manager is in charge of cache management policies, the client's activities with the cache, and the log file. For cache management to be adaptive, there should be some mechanisms in the cache manager to obtain current information about the environment that the cache management needs to adapt to.

  (1) The cache manager should be able to keep track of the mobile client's reference history and analyze the reference pattern, looking for characteristics such as locality.

  (2) The cache manager should be aware of local resource availability, including disk space, battery power, etc.

(3) The cache manager should be able to receive and process any important notification about network conditions.

(4) When informed of any change in the environment, the cache manager adapts to the changes by modifying the behaviour of cache management strategies.

- **File Server**. The file server is the permanent home of a user's files. It serves the mobile client's remote read and write accesses.

- **Internet**. The mobile client connects to the file server through the internet while mobile. 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 sometime possess a strong connection to the internet while mobile, this case is not considered. In this research, the performance of adaptive file cache management is investigated only when the network conditions are less than ideal.

4.3 Issues in Adaptive Cache Management

Bandwidth changes in the mobile client's network connection to the file server are the basis for adaptation in this research. Major questions arising when designing adaptive cache management include:

- What issues should be considered to make adaptation in cache management feasible and effective?

- Which aspects in cache management can be involved in adaptation?

- What strategies should be used for efficient adaptation?

- What level of adaptation should cache management take?

Answers to these questions are provided in the next subsections.

4.3.1 Effective Adaptation

Effective adaptation should:

- improve overall performance for mobile users.
- be aware of the current environment including bandwidth, workload, etc.
- take advantage of increased resource availability.
- minimize the impact of decreased resource availability.
- keep users' specific requirements as priorities.
- avoid interfering with users' other work.
- avoid increasing network and server load.

4.3.2 Implementing Adaptation in File Cache Management

The purpose of making file cache management respond dynamically to the changing environment is to improve cache performance for mobile users. Any cache management aspect whose performance could be affected by a changing mobile environment can be involved in adaptation. As discussed in Section 4.1, adaptation in cache management can be implemented in the way the system handles write backs, replacement, block caching, and prefetching, all of which have significant impact on cache performance.

Since the performance of cache replacement policies is affected more by cache size and the patterns of file requests than by network conditions in a mobile environment, cache replacement is not considered for adaptation in this research. Adaptive prefetching needs to develop an effective strategy on which files to prefetch and when to prefetch, it is left for future study.

The performance of write back and the performance of block caching in using the size of the block (the caching unit) can be severely affected by the currently available bandwidth. As a result, adapting write back and block caching to bandwidth changes are studied for this research.

4.3.3 Strategies for File Cache Management Adaptation

To make a cache management element respond dynamically to changes in resource availability, the aggressiveness of the element in using this resource needs to be changed at different resource levels correspondingly. The aggressiveness of write back can be changed by either using an alternative policy or by adjusting the policy parameters. For example, Froese [9] presents aggressive write-back policies, such as idle write back, and less aggressive write-back policies, such as delayed write back. If delayed write back is implemented and it is required to perform write backs more aggressively, idle write back can be used as an alternative, or the parameters in delayed write back can be reset to allow this policy to perform more aggressively. For adaptive block caching, the size of the data block can be changed (this changes the amount of data to be brought into the cache at each network access) dynamically as the bandwidth changes.

For each of the two approaches of adaptation (using alternative policies or adjusting parameters), there are two directions for adaptation. First, when the bandwidth increases, the cache management can be made to perform more aggressively to take advantage of the increased resource, and vice versa. Second, when the bandwidth decreases, the cache management can be made to perform more aggressively to avoid the impact of decreased resource, and vice versa. For example, the cache manager should take the first direction to take advantage of increased resources by propagating updates to the file server more aggressively when the bandwidth increases. It may take the second direction to reduce the client's network accesses when the bandwidth is low. Such an approach for adaptive block caching may avoid frequent interruptions on the user's application by bringing more data into the cache each time when there is a cache miss, with the prospect that the extra data brought in will be subsequently referenced by the user.

4.3.4 Control and Stability

The goal of any adaptive approach is to implement appropriate changes to parameters or policies. Ideally, changing policies or adjusting parameters in response to resource changes should maximize the benefit of a change and minimize the negative impact. This may require choosing an appropriate alternative policy or resetting policy parameters. Over-adapting or under-adapting must be avoided through appropriate control. For example, when there is a small increase in bandwidth, if write backs are performed much more aggressively than before the increase, the performance of write backs may be improved, but the interference with the client's read requests may also be increased significantly. Likewise, being too cautious may result in inefficient use of available resources. For this research, issues of stability were addressed only very casually.

4.4 Performance Evaluation

4.4.1 The Simulation

A trace-driven simulation is performed to evaluate the performance of approaches to adaptive file cache management.

Definition of the Environment

A general mobile environment is defined for these experiments. After a period time of hoarding, a mobile client is away from home and will perform work at multiple locations (at different times). As the client changes locations, he/she disconnects, and reconnects at a new location. At different locations, he/she may use different types and qualities of communication services with possibly different bandwidths to connect with the file server to perform his/her work. Even at a certain place, different networks may be available and the client may use different network connections (at different times) for reasons such as bandwidth (data rate), network failure, communication service cost, etc. It is assumed that the bandwidth for a given connection is constant, except for possible disconnection resulting from network or system failure. At the present time, since communication is available almost everywhere [6], it is assumed that the mobile client will have a connection to the file server anywhere it is taken, and the user performs work that involves file system requests only when the client is connected. It is also assumed that the client's connections to the file server when mobile are weak connections, even though the client may have strong connections when away from home.

The mobile client experiences frequent disconnections, both expected and unexpected, as discussed in the previous chapters. The expected and unexpected disconnections are determined through the long intervals between two consecutive events in individual traces. Thus, in the interval of any disconnection, the client has no file system requests.

Modelling Client Connections

In previous studies of cache performance in mobile computing [9, 11, 29], a constant network bandwidth was assumed for the mobile client's connection after hoarding. In the real world, a mobile client may experience enormous variations in network bandwidth as she/he changes locations and/or uses different communication technologies. Although a mobile client may experience a wide range of connection speeds, bandwidths 2 kbps, 9.6 kbps, 64 kbps, 512 kbps, 2 Mbps, and 10 Mbps are considered for this research. Bandwidth of 10 Mbps is used when the mobile client is hoarding, and the other five bandwidths are used while mobile. Network bandwidths 10 Mbps, 2 Mbps, 64 kbps correspond to the nominal speeds of Ethernet, WaveLan, and ISDN, respectively. Since a mobile client may experience other or decreased bandwidth capacities in a connection (described in Section 2.4.2), bandwidths of 512 kbps and 9.6 kbps are also included for this simulation. A mobile client may also experience very low bandwidth, so the bandwidth 2 kbps is chosen as the lowest bandwidth that the client may experience.

A randomly generated bandwidth sequence from bandwidths 2 kbps, 9.6 kbps, 64 kbps, 512 kbps, and 2 Mbps will be used for the experiments. Whenever a disconnection (expected or unexpected) happens, the client will have a chance to use a different bandwidth which will be obtained from the bandwidth sequence for the next connection.

The Traces

Three seven-day traces collected by Froese [9] from the Distributed Computing Laboratory in the Department of Computer Science at the University of Saskatchewan are used for this research. Each trace recorded a client's day-to-day file system events in an academic research environment. Each event contains information about a read or write request, including event time, userID, error status, vnode, file type, file size, file offset, transfer size, etc.

The expected and unexpected disconnections considered in these experiments are based on the interval time between two consecutive read or write events. If an interval is longer than fifteen minutes and shorter than an hour, it is assumed that the client is experiencing an unexpected disconnection such as a system or a network failure. If the interval is over one hour, it is considered as an expected disconnection, for example, when the user changes location, he/she deliberately disconnects from the network and connects at a new location. The number of intervals for each case in each trace is shown in the rows *Intervals* of Table 4.1.

The characteristics of the traces are shown in Table 4.1. The number of active hours in each trace was computed by subtracting the amount of "idle" time (the long intervals between two consecutive events, see the description in [9]) in the trace from the time between the first and last references in the trace. Total Requests is the number of events in the trace. Unique Files is the number of different files referenced, and Unique Bytes is the total size of those unique files. Read Transfers and Write Transfers are the sum of the sizes of all read and write requests in the trace respectively. Write Percentage is the percentage of read and write requests that are writes. More detailed information about these traces including trace collection and trace preprocessing can be found in Froese's thesis [9].

The three traces selected for these research represent the range of workloads that a mobile client may expect to deal with in a typical academic environment [11]. Trace 1 has a moderate amount of file system activity and a read-to-write ratio of about 2:1. Trace 2 has a large number of 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 4.1: 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

The Simulator

Froese's simulator is used for this research. This simulator consists of over 10,000 lines of C++ code and is described in detail in [9].

In Froese's simulation model, there were seven processes: *generation of file I/O requests, change in network characteristics, file transfer, cache replacement, hoarding, log file optimization, write backs.* To incorporate the modified system model and to implement adaptive file cache management strategies, some processes were modified or redesigned and new processes were developed. These include *change in network characteristics, write backs, hoarding, and block caching.*

- *Change in Network Characteristics.* The characteristics of the network connection between the mobile client and the home file server include bandwidth, propagation delay, and error rate. In this research, both the propagation delay and the error rate are set to zero. The characteristics of the network change when there is a long interval between any two consecutive events in the trace currently used. The bandwidth for the next connection is randomly chosen from the five bandwidths. A randomly generated bandwidth sequence is used for this purpose.

- *Hoarding.* For adaptive write back implementation, when the client is in hoarding mode (strongly connected), the hoarding is on a whole-file basis. Whenever a file is referenced, the whole file is brought into the client's cache. For adaptive block caching implementation, the hoarding is on block-caching basis. When a block of a file is referenced, this block and its immediate neighbour blocks are brought into the client cache.

- *Adaptive Write backs.* When the client is weakly connected, the file updates are recorded in the log file at the client and have a chance to be written back to the file server as determined by the write-back policy currently implemented. The write-back policy will adapt to bandwidth changes. When there is a change in bandwidth, the behaviour (the aggressiveness) of the write-back policy is changed by resetting policy parameters or using alternative write-back policy.

  The delayed write-back policy and the idle write-back policy are considered in the performance study. In the delayed write back, when parameter A, the minimum delay time, is set to zero, it turns into idle write back.

  Delayed Write Back. Delayed write back writes back the oldest log entry if it has been in the log file for at least the minimum delay time A and if the network has been idle for some period I.

  Idle Write Back. Idle write back writes back the oldest log file entry when the network has been idle for some period I.

- *Adaptive Block Caching.* The amount of data to be brought into the cache is adaptive to bandwidth changes. Depending on the current available bandwidth, the amount of data to be brought in is a multiple or a fraction of the block size used when hoarding.

The simulator reads events from a trace sequentially. When a read occurs, the client cache is searched for the requested data (files or data blocks, depending on the caching unit implemented). If the data is in the cache , i.e., a cache hit, the cache is updated. The request is satisfied immediately by the cache, and no network access is required. Otherwise, i.e., a cache miss, the requested data will be transferred from the file server over the weak connection. The user's application is suspended until the transfer is completed. 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 occurs, 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 file. Otherwise, log optimization is performed and 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. The updates may be later propagated to the file server. How long an update should stay in update log file and when the update is written back depend on the write-back policy currently implemented and the settings of policy parameters. The oldest entry is always written back first.

The simulator checks the intervals between any two consecutive events to determine whether a disconnection occurs. If an expected disconnection occurs, all file updates in the log file are written back right before the disconnection. If an unexpected disconnection occurs, all log entries will have to remain in the log until the connection resumes.

4.4.2 Factors Affecting Performance

Many factors affect cache performance. In this study, cache size, caching unit, workload, and the levels of adaptation were considered.

- *Workload.* The workload is the requests (reads and writes) made to the system by the mobile users. The workload characteristics such as locality, the read-to-write ratio, the file sizes, and the inter-event times can have significant impact on cache performance.

- *Cache Size.* The size of the cache determines the amount of data the cache can contain. Usually, a larger cache means better performance for a given workload.

- *Caching Unit.* Froese's study [9] showed that whole-file caching and block caching impact cache performance differently. In this research, the impact of different sizes of data blocks will be considered in adaptive block caching.

- *Levels of Adaptation.* The levels of adaptation refers to the aggressiveness of a policy at various resource levels. Different levels of adaptation will result in different performance.

4.4.3 Performance Metrics

The research in adapting cache management to bandwidth changes 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 will be used to evaluate the performance of adaptive cache management [9].

- *Time expansion.* Time expansion is the ratio of the time required to process all of the user's file system requests while mobile to the time required to perform the same activity while strongly connected. It reflects the additional cost to the user's mobile operation, e.g., the added time needed to process file system operations remotely. It is a measure of the satisfaction of mobile users with the system implementation and the environment. Two factors mainly affect the time expansion: read service time and write interference time (described below).

- *Log size.* Log size indicates how much of the client's precious disk space is used for keeping the user's file updates. The average log size measures the average usage of disk space for the file updates. Maximum log size specifies a minimum disk space needed for file updates at the client.

- *Time in log.* The update integration time is measured by the time updates remaining in the log at the client. It is the time between a write event at the mobile client and the propagation of the update to the file server for reintegration. The time in log reflects the possibility of update conflicts and data loss or damage. The longer an update remains in the log, the greater the risk of these problems.

- *Write interference time.* Write interference time is the time when the write backs are in progress while read requests are waiting to be served. Write interference time indicates how the write-back policy affects the interactive requests. It is one of the two factors that affect the time expansion.

- *Log overwrites.* The number of log overwrites indicates how many log entries are overwritten by upcoming updates in the delayed write-back policy. Log overwrites imply how much disk space is saved by log optimization and how many network accesses for update propagation are eliminated.

- *Bytes written back.* Bytes written back is the amount of data that is written back to the file server for reintegration. It implies how much bandwidth is consumed by write backs.

- *Network accesses.* It is the number of times that the mobile clients needs to access the network for data transfer. A read access occurs when there is a cache miss. Read accesses implies how many times the user's work is interrupted.

- *Read Service time.* Read Service time is a performance metric. It is the time spent on transferring requested data that is not in the cache from the file server remotely. It directly affects the user's application suspending time and the client's waiting time for requested data. It is the main factor that affects the time expansion.

Chapter 5

Experiments with Adaptive Write Back

Write backs are desirable no matter what quality the network conditions are. Timely propagation of file updates from a mobile client allows users in a distributed file system to share data effectively. Write backs also reduce the size of the update log file, and thus reduce the consumption of scarce local resources in the mobile computers. A shorter update reintegration time implies better data sharing and a smaller probability of write-write conflicts. Cache management in file systems for mobile users should attempt to reduce update reintegration time without affecting the performance of reads. Since the bandwidth for a mobile client changes from time to time and has a large impact on write back performance [9], write backs should be managed to respond to the changes dynamically to achieve better write back performance.

The purpose of adapting write back to changes in bandwidth is to reduce the update reintegration time, which can be measured by the time the update remaining in the log file at the client. A series of experiments was performed to investigate the performance benefits and negative impact from write back approaches that adapt to changes in available bandwidth. A delayed write back policy, which has been implemented in many distributed file systems [41, 31] and has proved to work well for mobile clients [29, 19, 11, 9], is considered in this research. The adaptation is realized by adjusting the policy parameter, the minimum delay time A, at different available bandwidths.

This chapter first investigates the range of reasonable values for the minimum delay time at different bandwidth levels. The results of simulation experiments with adapting write back to changes in available bandwidth are then presented.

5.1 Parameter Settings

In the delayed write back policy, two parameters, the minimum delay time A and the idle threshold I, control the behaviour (the aggressiveness) of the implementation. When the network has been idle for a period time of I, the oldest log entry that has been remaining in the log file for a minimum time A will be written back to the file server for reintegration. The purpose of the idle threshold I is to attempt to ensure that write backs are performed at network idle time and therefore cause little impact on read requests. Although the value for idle threshold has impact on write back performance, it is not included for the adaptation in this research. A fixed value of 0.1 seconds for I, suggested by Froese [9] as an appropriate value for a weakly-connected environment, will be used throughout all the experiments.

Both the parameter minimum delay time A in the delayed write-back policy and the available bandwidth have a large impact on write back performance. The settings for minimum delay time A at different available bandwidths are critical to achieve good write back performance and to avoid impact on read performance.

The minimum delay time A determines the minimum period of time that an update has to stay in the log file before it becomes eligible to be written back, therefore, it has a direct impact on the update reintegration time. This parameter also affect the log entries overwritten which implies how much time and bandwidth will be used for transferring file updates to the file server. A longer delay time likely increases the number of log entries overwritten, but it also increases the update integration time. A shorter delay time reduces the update reintegration time, but may cause more write back traffic and more interference with reads. Appropriate settings for delay time at different bandwidths should provide the benefit of reducing update reintegration time with minimal interference to reads.

When a low bandwidth is available, data transfer is slow, write backs can easily

Table 5.1: Parameter Delay Time A Settings Tested

Parameter	Values
Bandwidth	2000, 512, 64, 9.6, 2 (kbps)
Minimum Delay Time	0, 1, 5, 10, 15, 60 (minutes)
Idle Threshold	0.1 (seconds)
Cache Size	2 (MB)

interfere with reads. To avoid write back traffic interfering with reads, the minimum delay time should be long enough to have a high percentage of log entries overwritten. This reduces the write back traffic and reduces the chance of write backs interfering with reads. But a very long minimum delay time is not recommended, because the update integration time will be increased and the benefits of write backs will be lost. When high bandwidth is available, the data transfer is fast. The possibility of write backs interfering with reads would be small. Write backs can be performed aggressively by using a small minimum delay time to reduce the update reintegration time. From the above discussion, to achieve good write back performance, the minimum delay time should be decreased when the bandwidth increases, and it should be increased when the bandwidth decreases.

An initial set of experiments was performed on each of five bandwidths (2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps) at minimum delay time ranging from 0 minutes to 60 minutes. Then the appropriate values of minimum delay time at different bandwidths were determined. These values were used for subsequent experiments for adaptive write back. Whole-file caching was implemented and a 24-hour hoarding period was applied. The LRU replacement policy was used both when hoarding and while mobile. A cache size of 2 MB was used. The parameters for this set of experiments are summarized in Table 5.1. The effect of the minimum delay time A was investigated by examining its effect on write interference time and log entries overwritten at different bandwidths.

The effect of the minimum delay time A on write interference time across the five bandwidths is shown in Figure 5.1. Clearly, lower bandwidths such as 2 kbps

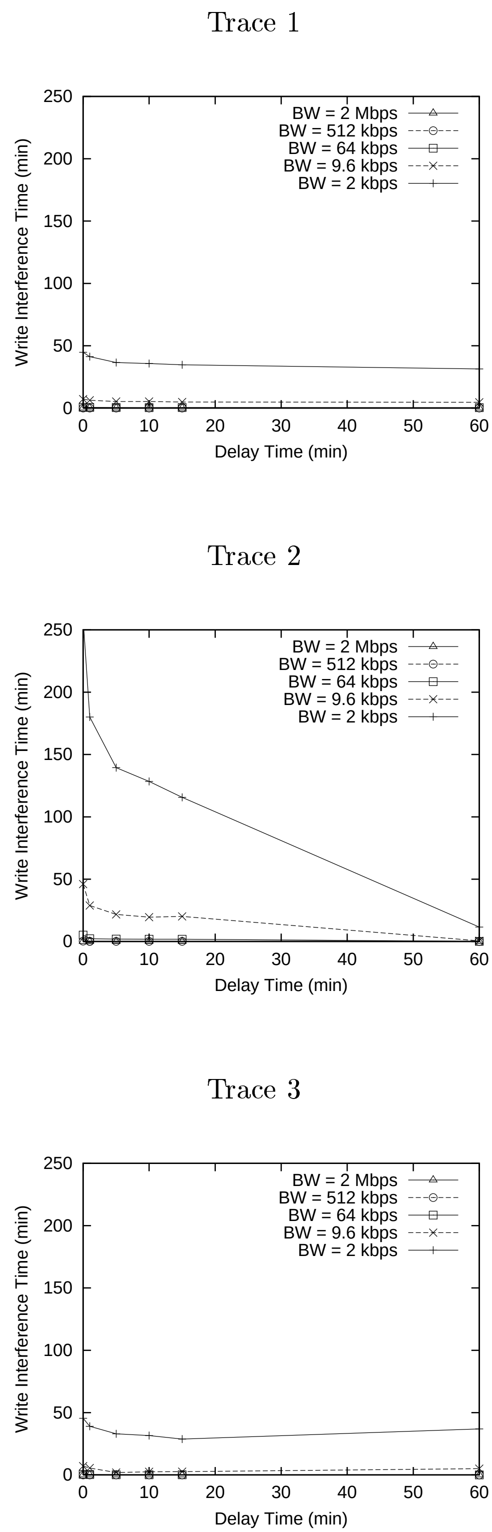
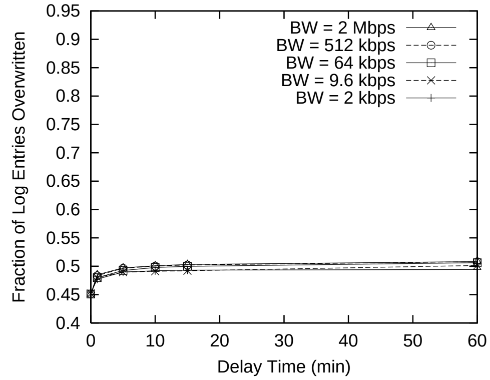
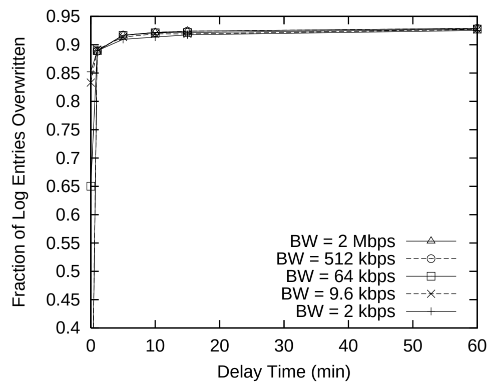


Figure 5.1: Effect of A on Write Interference Time

Trace 1



Trace 2



Trace 3

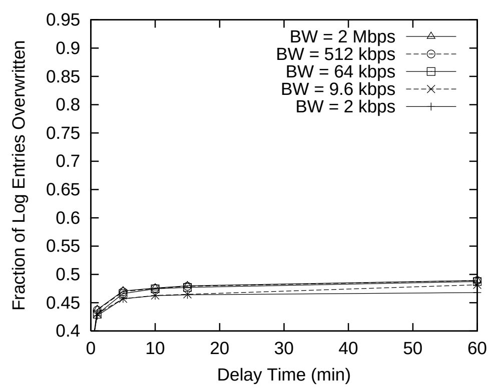


Figure 5.2: Effect of A on Log Entries Overwritten

cause larger write interference time for all three traces. The write interference time decreases when the minimum delay time increases, because a longer delay time allows more log entries to be overwritten by subsequent updates, and fewer file updates need to be written back. The rate of decrease in write interference time becomes slow when the minimum delay time is longer than 15 minutes. For Trace 3, when the delay time is longer than 15 minutes, the write interference time even increases. These results suggest that at low bandwidths, the delay time should have a large value to achieve a small write interference time, but a value larger than 15 minutes is not necessary. These results confirm Froese's suggestion for the minimum delay time in [9, 10]. High bandwidths such as 2 Mbps and 512 kbps cause almost no write interference time, since data transfer is quite fast at those bandwidths and most write backs can be completed during network idle time. Given the fact that a high bandwidth is available, the minimum delay time can be set to a very small value to reduce the average time log entries remain in the log file, and this aggressive write back will have little impact on the demand read requests.

Figure 5.2 shows the effect of the minimum delay time A on log entries overwritten at different bandwidths for the three traces. For each of the five bandwidths, the number of log entries overwritten increases when the minimum delay time increases. With a longer minimum delay time, updates will have to stay in the log file for a longer time and have more chances to be overwritten by subsequent updates. At a certain delay time, a higher bandwidth results in a higher number of log entries overwritten. This is because at a higher bandwidth, the user's read and/or write requests are performed faster; in the same period of time, more updates may come up and these updates may overlap some log entries and overwrite them. For the three traces, when the delay time increases from 0 minutes to 15 minutes, the number of log entries overwritten increases rapidly. The fastest rate of increase is when the delay time increases from 0 to 1 minute. For higher bandwidths, any minimum delay time causes little write interference time, but a 30-second or 1-minute delay time may greatly reduce write traffic.

From the above discussion, at a low bandwidth such as 2 kbps, the minimum delay time for delayed write back should be long but not necessarily longer than 15 minutes to achieve good write back performance, such as a short write interference time and a high number of log entries being overwritten. For a high bandwidth such as 2 Mbps, a very low (even zero) minimum delay time can be set for write back. But for the effect of log optimization, a short delay (as low as 30 seconds or 1 minute) should be used. These experiments suggest that the range of minimum delay time for the five bandwidths considered should fall between 0.5 minutes and 15 minutes. The minimum delay time will be set to 0.5 minutes and 15 minutes for bandwidths of 2 Mbps and 2 kbps, respectively. For other bandwidths, the values for minimum delay time will be between 0.5 minutes and 15 minutes. For these experiments, the minimum delay time for bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps are set to 0.5 minutes, 0.5 minutes, 5 minutes, 10 minutes, and 15 minutes, respectively, as one approach of adaptation.

5.2 Experimental Design

All three seven-day traces were considered for this study. Before going mobile, the client is strongly connected to the file server and hoards files to the client cache using an LRU hoarding policy. A 24-hour hoarding period was chosen since it is long enough for preparing the file cache for mobile [9]. After hoarding, the client will perform work at several locations and have weak connections to the file server. These connections may have different bandwidths randomly chosen from the five bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps. A randomly generated bandwidth sequence was used in these experiments. Caching was on a whole-file basis. While the client is weakly connected, LRU replacement is performed. Cache sizes ranging from 1 MB to 10 MB are used. The simulation parameters for these experiments are summarized in Table 5.2.

The performance of adaptive write back was compared to that of non-adaptive write back. In the non-adaptive write back approach, a fixed value was used for the minimum delay time A at any available bandwidth. In these experiments, a value of

Table 5.2: Factors and Levels Used in Adaptive Write Back Experiments

Factors		Levels
Hoarding Period		24 hours
Hoarding Policy		LRU
Replacement Policy		LRU
Caching Unit		Whole file
Cache Size		1, 2, 5, 8, 10 (MB)
No Adaptation	Write-back Policy	Delayed Write Back
	Strategy	Fixed parameter A
Adaptation-1	Write-back Policy	Delayed Write Back
	Strategy	Adjust A at different bandwidths
	Direction	Decrease A when bandwidth increases
Adaptation-2	Write-back Policy	Delayed Write Back, Idle Write Back
	Strategy	Alternative policy, adjust parameter A
	Direction	Decrease A when bandwidth increases
Adaptation-3	Write-back Policy	Delayed Write Back
	Strategy	Adjust parameter A at different bandwidths
	Direction	Decrease A when bandwidth increases

15 minutes was used. This value was suggested by Froese [9] based on experiments with delayed write backs for mobile clients in a weakly-connected environment. Any value that is bigger or smaller than this could result in either a large average log size or a long write interference time.

Experiments were performed on three approaches to adaptive write backs. The first approach (called adaptation-1) had minimum delay time of 0.5 minutes, 0.5 minutes, 5 minutes, 10 minutes, and 15 minutes for bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps, respectively. This adaptive write back approach was considered as having appropriate settings for minimum delay time at different bandwidths, as discussed in Section 5.1. The second approach (called adaptation-2) implemented a strategy of using idle write back as an alternative by setting the value of A to zero in the delayed write back algorithm, when the highest bandwidth considered (2 Mbps) was available. When any other bandwidth was available, delayed write back was still applied, with a higher bandwidth having a smaller value

Table 5.3: Minimum Delay Time A Settings (min)

Bandwidth (kbps)	2000	512	64	9.6	2
No Adaptation	15	15	15	15	15
Adaptation-1	0.5	0.5	5	10	15
Adaptation-2	0	1	5	15	20
Adaptation-3	1	2	5	8	10

of A, and a lower bandwidth having a bigger value of A. The values of the minimum delay time for bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps were 0 minutes, 1 minute, 5 minutes, 15 minutes, and 20 minutes, respectively. This approach was considered as somehow over-adapted compared to adaptation-1. The third approach was considered as under-adapted. The values of the minimum delay time for bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps were 1 minute, 2 minutes, 5 minutes, 8 minutes, and 10 minutes, respectively. Experiments on adaptation-2 and adaptation-3 were performed to examine the performance of other levels of adaptation which may not perform as well as adaptation-1 in some aspects. For all the experiments, the idle threshold I was a fixed value of 0.1 seconds for reasons discussed before. Table 5.3 shows the values of the minimum delay time A selected for different bandwidths in the three adaptive approaches.

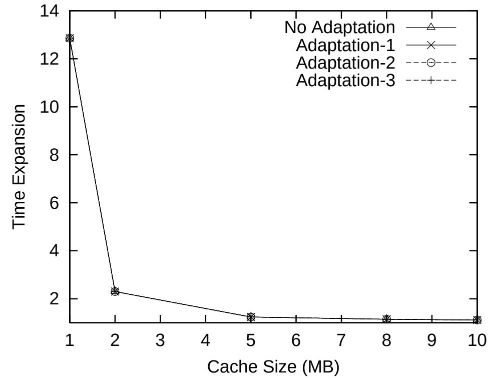
5.3 Simulation Results

Figures 5.3 through 5.9 show the performance of non-adaptive write back and three approaches to adaptive write back on the three traces, for different cache sizes at the mobile client.

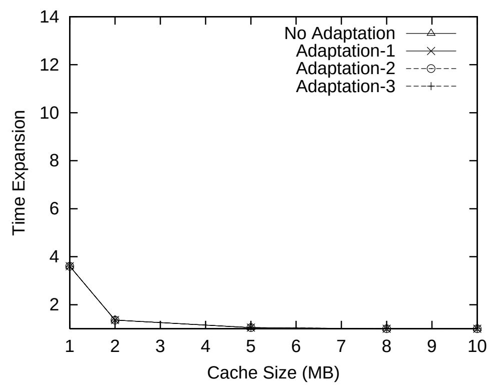
5.3.1 Time Expansion

Time expansion is affected mainly by the bandwidth, the size of the cache, the read service time, and the write interference time. Since the same randomly-generated bandwidth sequence was used for all experiments, the connection bandwidth was not

Trace 1



Trace 2



Trace 3

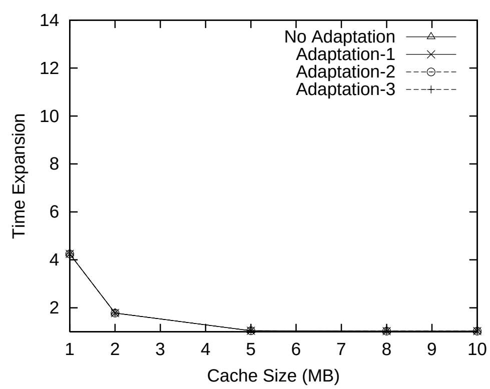
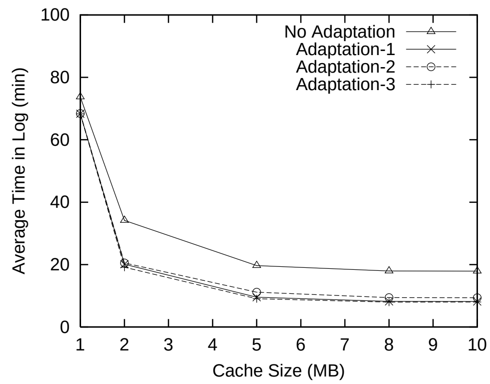
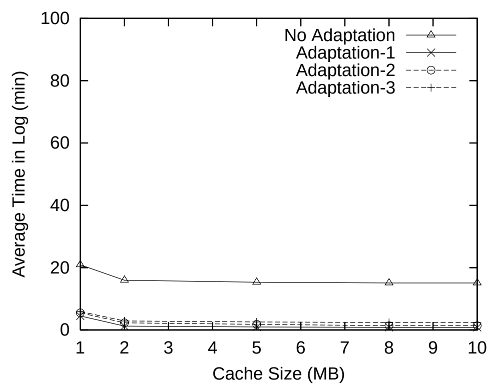


Figure 5.3: Time Expansion

Trace 1



Trace 2



Trace 3

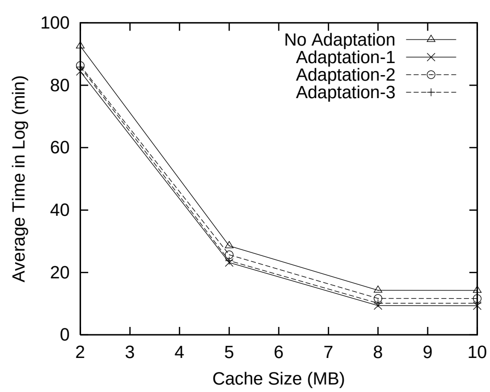
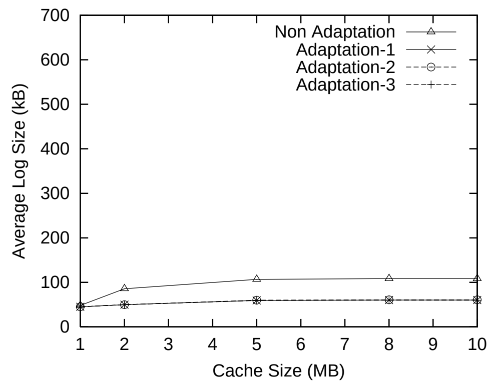
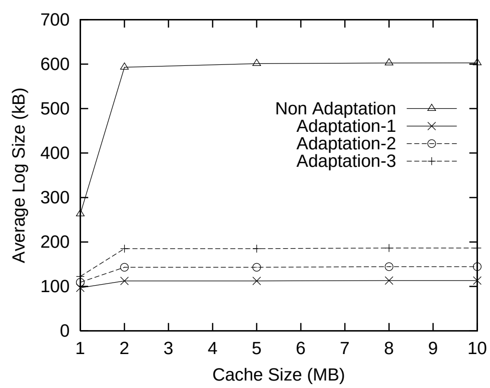


Figure 5.4: Average Time Log Entries Remain in Log File

Trace 1



Trace 2



Trace 3

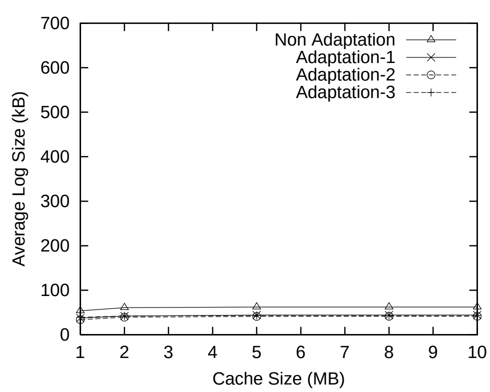


Figure 5.5: Average Log Size

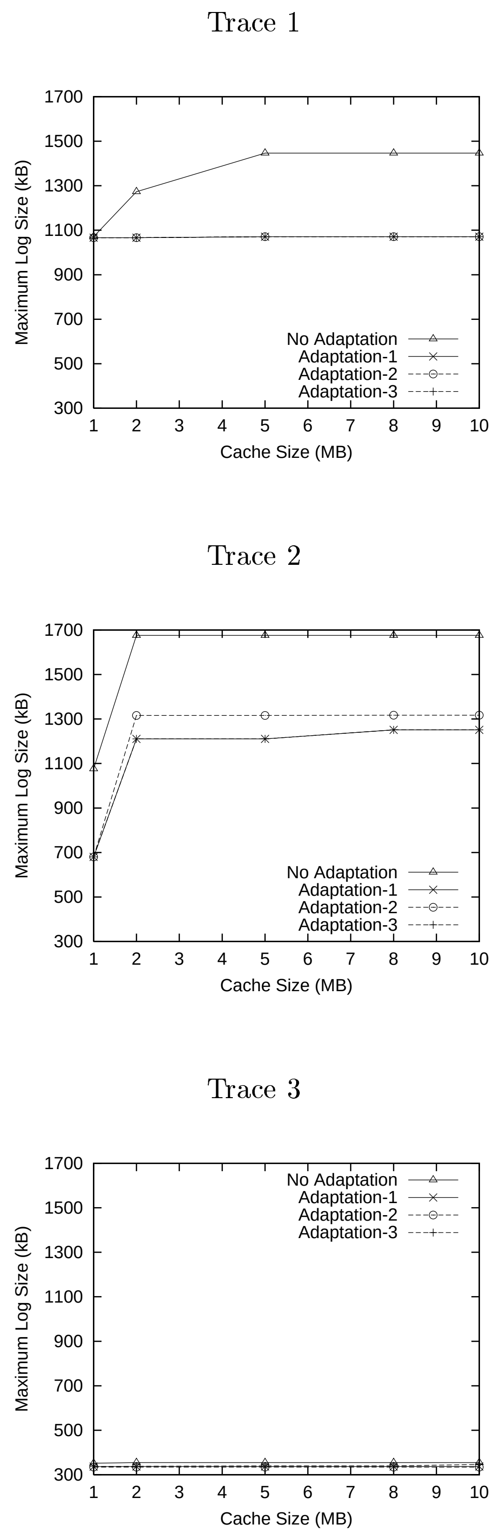
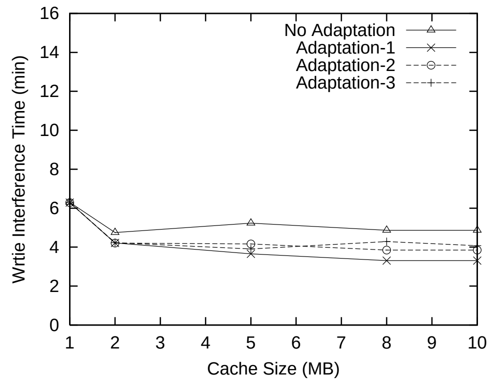
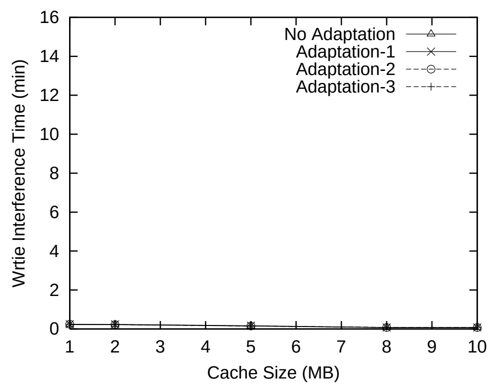


Figure 5.6: Maximun Log Size

Trace 1



Trace 2



Trace 3

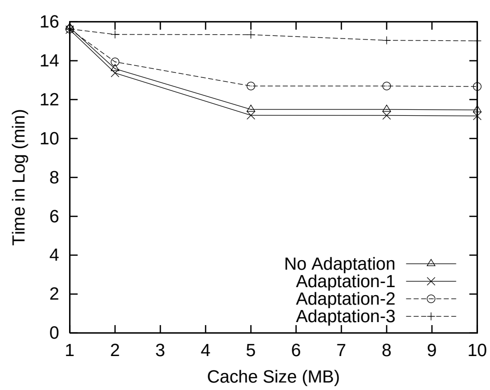


Figure 5.7: Write Interference Time

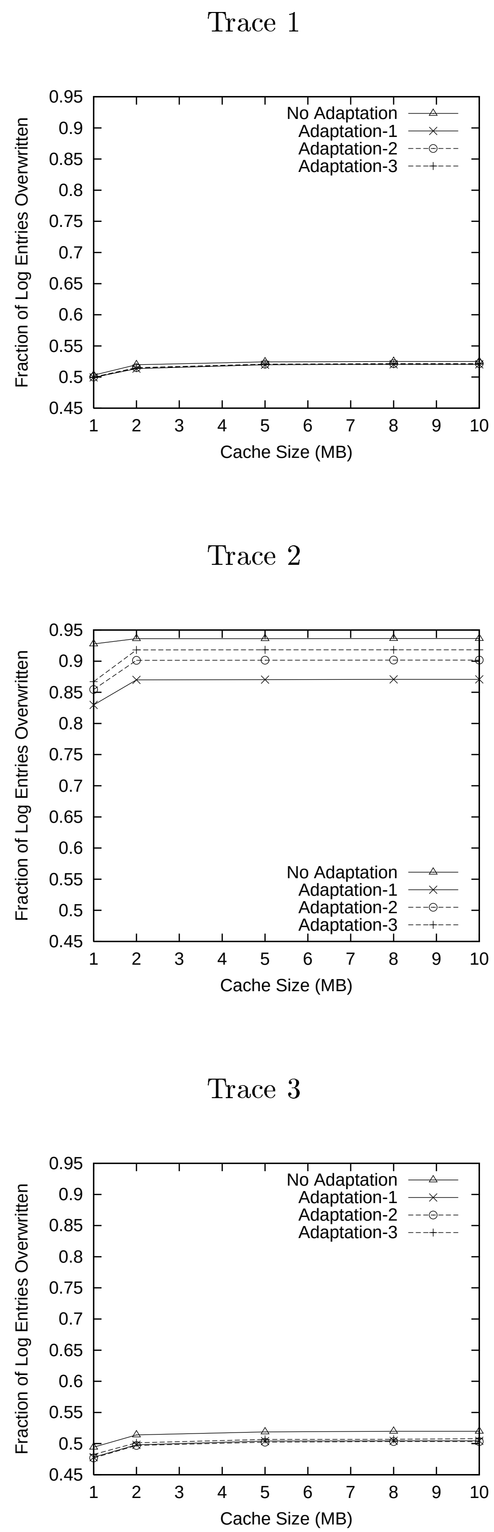


Figure 5.8: Log Entries Overwritten

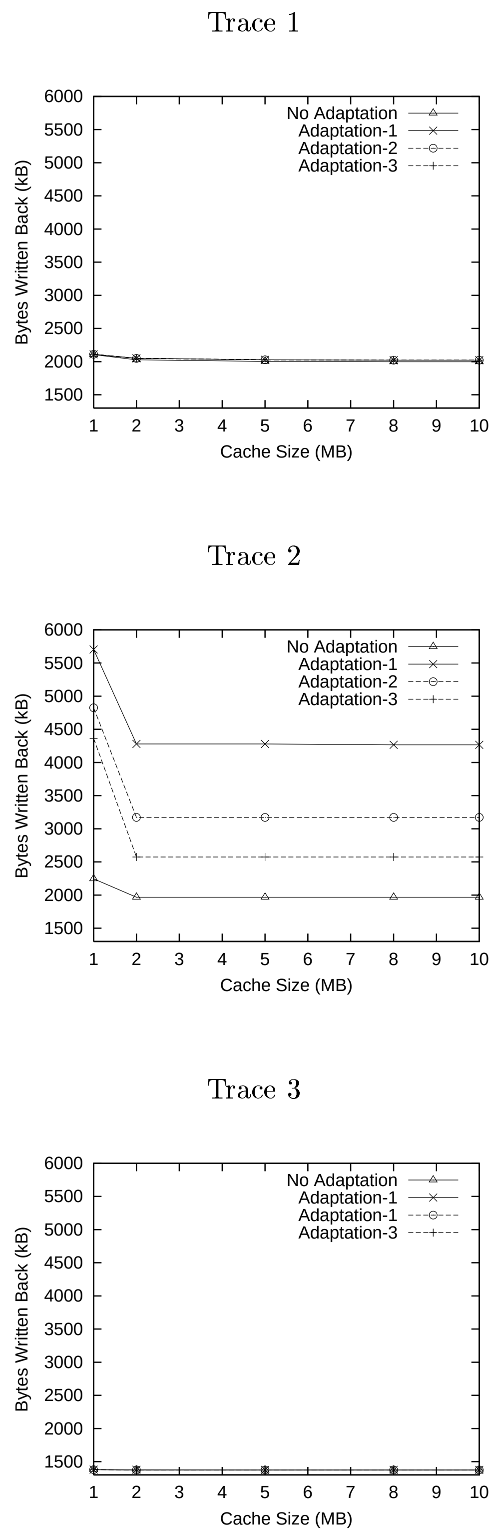


Figure 5.9: Bytes Written Back

a factor. The size of the cache can have a large impact on time expansion, because the size of the cache determines how many of the user's files can be contained in the cache, and thus it affects the miss ratio and the read service time. Write interference suspends the user's application until the write is completed. For these experiments, the performance of different write-back strategies was compared at each given size of the cache.

Figure 5.3 shows the time expansion for the three traces with non-adaptive write back and the three approaches to adaptive write back. For each trace, the size of the cache has a large impact on time expansion for all approaches, as expected. Although the time expansion is different across the three traces at a certain size of the cache (this depends on the trace characteristics, such as the byte transferred, the inter-event time, etc.), the different approaches to write back result in almost no difference in time expansion at each given size of the cache for each trace. This result suggests that propagating updates more aggressively will not degrade the performance of read requests. The writing back of updates is scheduled during network idle time, and in the adaptive write back approaches, the more aggressive write backs are performed only at higher bandwidths, such as 2 Mbps and 512 kbps. Thus the aggressive write backs can be completed more quickly and most can be finished during network idle time, the interference with demand read requests would be small. Therefore, adaptive write back causes little write interference time and has little impact on time expansion.

5.3.2 Average Time in Log

The primary effect of adapting write back is to reduce the time updates remain in the log file. The time of a log entry remaining in the log file is determined mainly by the implemented write-back policy. A longer time an update remains in the log file implies a higher probability of a write-write conflict, and more consumption of the mobile client's local resource. With delayed write back, the time a log entry remains in the log file is determined primarily by the minimum delay time A.

Figure 5.4 shows the average time log entries remain in the log file before they are propagated to the file server. For all three traces, at any size of the cache, the three adaptive write-back approaches result in a shorter average time than the non-adaptive approach, of which adaptation-1 performs the best. When adaptive write backs are implemented, the minimum delay time A is shorter than in non-adaptive write back at high bandwidths, such as 2 Mbps and 512 kbps. This allows file updates in the log to be eligible for writing back earlier and makes the average time shorter. Although adaptation-2 and adaptation-3 are over-adapted and under-adapted compared to adaptation-1, they still result in a shorter reintegration time than non-adaptive approach, since at high bandwidths, the delay time settings for these two adaptive approaches are still much shorter than in the non-adaptive approach. The reduction of update reintegration time by all adaptive approaches is significant for all three traces, with the trace that has the higher percentage of write requests having the more significant reduction. In Trace 1, the reduction is over 50% with a cache size larger than 2 MB. In Trace 2, the reduction is over 80% at any size of the cache. In Trace 3, the reduction is around 20% at cache size of 5 MB. For all the traces, adaptation-1, considered to have appropriate settings for minimum delay time, performs best in reducing the update reintegration time.

5.3.3 Average Log Size

The impact of different write-back approaches on the average size of the update log file is shown in Figure 5.5. Although there are differences in the average size of the log across the three adaptive write-back approaches in each trace, all the approaches to adaptive write back result in a smaller average log size than the non-adaptive approach. When an update is written back, it no longer needs to be kept in the log file. In adaptive write back, some log entries were propagated to the file server shortly after the file updates were recorded in the log file in adaptation-1 and adaptation-3, or immediately in adaptation-2 when a relatively high bandwidth is available. The reduction in average log size by adaptive write backs is over 50% for

Trace 1 and Trace 2, and 30% for Trace 3 when the cache size is larger than 2MB.

5.3.4 Maximum Log Size

Figure 5.6 shows the maximum size reached by the update log file. The same trend as observed in the average log size is also shown here for maximum log size. The maximum log size resulting from adaptive write backs is much smaller than that observed with non-adaptive write backs.

5.3.5 Write Interference Time

Figure 5.7 shows the effect of different write back approaches on write interference time. Overall, the write interference time is quite small for all approaches and for the three traces, even when the cache size is as small as 1 MB. For Trace 2, all approaches have almost the same write interference time, which is less than 1 minute. For Trace 1, all adaptive approaches perform better than non-adaptive write back. Adaptation-1 performs better than adaptation-2 and adaptation-3. For Trace 3, adaptation-1 still presents the best performance. Over-adapted adaptation-2 and under-adapted adaptation-3 perform even worse than the non-adaptive approach.

5.3.6 Log Entries Overwritten

Less aggressive write backs will result in more log entries being overwritten due to the longer minimum delay time requirement for log entries. As shown in Figure 5.8, the non-adaptive write-back approach in which log entries have to stay in the log for at least 15 minutes regardless of bandwidth changes presents the highest percentage of log entries overwritten for all three traces. Although the three adaptive approaches perform differently in the three traces, they all have a smaller percentage of log entries overwritten. For most traces with reasonable percentage of write requests (Trace 1 and Trace 3), the adaptive approaches cause a little fewer log entries overwritten than non-adaptive approach. In Trace 1 and Trace 3, adaptive approaches result in 2% fewer log entries being overwritten than the non-adaptive approach in the worst case. In Trace 2, this is around 6% in the worst case, due to its high percentage of write-to-read ratio. Overwriting fewer log entries will result in more updates needing to be written back and therefore more bandwidth needing to be consumed for update propagation.

5.3.7 Bytes Written Back

Figure 5.9 shows the bytes written back by adaptive and non-adaptive write backs. The non-adaptive approach writes the fewest bytes back to the file server for all three traces, since it has the highest percentage of log entries overwritten. All three adaptive approaches write more bytes back, due to their shorter delay time at higher bandwidths. In fact, the increase in bytes written back is insignificant for traces which have a normal write-to-read ratio (Trace 1 and Trace 3). For Trace 1, this increase is less than 1.5% in the worst case. For Trace 3, which has the least write-to-read ratio, the increase is less than 0.5% in the worst case. With Trace 2 that has high write-to-read ratio, the bytes written back increases in a large amount for all the adaptive approaches. With adaptation-3, this increase is 25%.

5.4 Conclusions

These experiments were performed to investigate the benefits and negative impact of adaptive write backs on cache performance. From the above simulation results, it can be observed that, compared to non-adaptive write backs, adaptive write backs result in:

- No increase in time expansion. The adaptive write back approaches result in essentially the same time expansion as non-adaptive write back.

- Smaller average and maximum log file size. The adaptive approaches to write back reduce the size of the log by more than 30% compared to non-adaptive approach for all traces.

- Less write interference time. Although aggressive, adaptive approaches to write back can slightly reduce write interference time.

- Less average time updates remain in the log file. The adaptive approach to write back achieves its goal of reducing time in log. The reductions are 20% to 80% for the three traces.

- Fewer log entries overwritten. The adaptive approaches to write back reduce the effect of log optimization and cause fewer log entries overwritten, but the impact is only 2% for most traces.

- More bytes written back. The negative impact of adaptive write back is that it increases the bytes written back. But for traces with the percentage of write requests less than 40%, the increase is less than 2%. For the trace with a higher percentage of write requests, this increase could be much higher.

The simulation results show that adaptive write backs can improve performance by properly changing the aggressiveness of the write-back policy at different bandwidths. The improvements in the size of log file and the average time log entries remain in log file are significant. Although the fact that fewer log entries are overwritten may cause more bytes needed to be written back, the increase is small and can be ignored for most traces.

Chapter 6

Experiments with Adaptive Block Caching

Whole-file caching could reduce the number of accesses to the network if the same file is subsequently accessed. Meanwhile, it is the only practical approach when operating in a disconnected environment [9]. However, if the mobile user works only when the client is connected, even with a low bandwidth, block caching is an option.

Block caching has many advantages over whole-file caching. If only part of a file is referenced, block caching provides shorter file access time than whole-file caching. The user can resume work more quickly, because less time is wasted on bringing in the data that is not referenced. The cache can contain more of the user's files in terms of data blocks. With a low-bandwidth connection, block caching becomes more advantageous. Froese and Bunt's research on caching unit [9, 11] showed that block caching did have better performance than whole-file caching in a weakly-connected mobile environment.

Both the size of the block and the bandwidth impact cache performance. The performance of block caching could be further improved if it is possible to adapt to changes in the available bandwidth. The size of the block to be brought into the client's cache can be dynamically reset as the available bandwidth changes. At a lower bandwidth, if a smaller amount of data (only the required data) is brought in, instead of a big block most of which the user may not need, the user's file access time can be further reduced. At a higher bandwidth, more data (or a big data block) can be brought in, and the user's waiting time may only be slightly increased since data transfer is fast.

This chapter investigates the performance of adapting the block size to bandwidth changes in a block caching implementation (adaptive block caching) on a simulation basis. In Section 6.1, appropriate block sizes for different bandwidths are determined through experiments with non-adaptive block caching, in which a fixed block size is used no matter what bandwidth was available. The experimental design for this part of work is provided in Section 6.2. The experiment results are separated in three parts. In the first part, the performance of adaptive block caching is evaluated at different sizes of the cache with non-adaptive write back. In the second part, the performance is examined when the size of the data block varies, again with non-adaptive write back. In the last part, the performance of adaptive block caching is studied with adaptive write back to examine if these two adaptive strategies conflict with each other.

6.1 Parameter Settings

The goal of adapting block size to changes in the connection bandwidth is to reduce the read service time and to allow the user to resume work more quickly than with non-adaptive block caching. At the same time, adaptive block caching should avoid causing negative impact such as increasing the server and network load, i.e., the number of accesses to the network. It is critical to choose appropriate sizes of the data block to be brought into the client cache at different bandwidths, since the amount of data to be brought in over a connection with a certain bandwidth directly affects the read service time and thus the time expansion.

When only low bandwidth is available, the data transfer is slow, and so the block size should be set to a small value to reduce the read service time. But too small a block size may fragment the required data and result in several network accesses. If high bandwidth is available, the block size can be set to a big value. Since the data transfer is fast, the increase in the read service time is negligible. Bringing more data may reduce the network accesses if the data brought in is eventually referenced. But a very big block may result in the eviction of other cache entries which may

Table 6.1: Block Size Settings Tested

Parameter	Values
Bandwidth	2000, 512, 64, 9.6, 2 (kbps)
Block Size	0.5, 1, 2, 4, 8, 16, 32 (kB)
Cache Size	0.5 (MB)

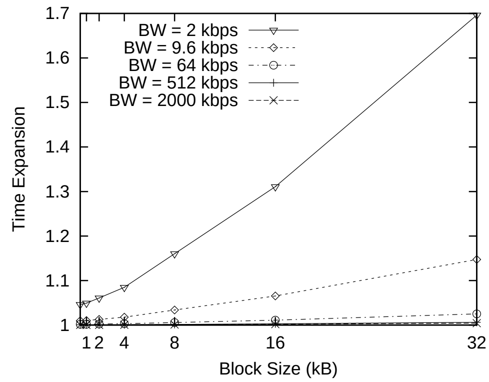
be referenced subsequently. This would increase the number of network accesses. There should be a range of how small the block size is with a low bandwidth and how big the block size is with a high bandwidth.

Experiments were performed on the three seven-day traces at each of the five bandwidths (2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps) at a range of block sizes from 0.5 kB to 32 kB. The appropriate block size for each bandwidth was then suggested. Hoarding was performed for a 24-hour period at the start of the simulation on a block caching basis. A small cache of 0.5 MB was used[1]. The simulation parameters for these experiments are summarized in Table 6.1. Since adaptive block caching should reduce the time expansion and should not increase the number of network accesses, time expansion and the number of network accesses are used as performance metrics.

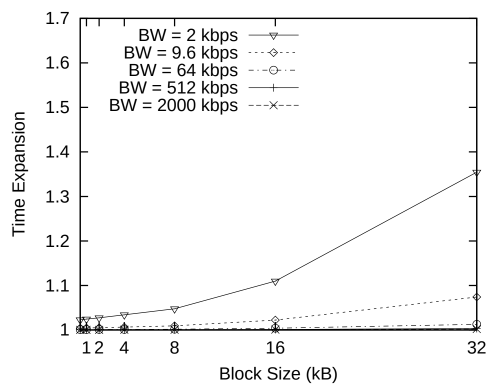
Figure 6.1 shows the time expansion from the five bandwidths as a function of block size for the three traces. For each bandwidth, as the block size increases, the time expansion increases as well, because transferring a bigger block takes a longer time. The rate of increase is fast for low bandwidths, such as 9.6 kbps and 2 kbps. When the bandwidth is higher than 64 kbps, the time expansion is very small at any block size, and so the rate of increase as the block size increases is negligible. As illustrated in Figure 6.1, when only a low bandwidth is available, a small block size should be used to keep the time expansion small. For bandwidth higher than 64 kbps, any big block could be used, but too big a block may result in a increased

[1]This size of the cache is small, but for these a few years old traces it is appropriate. With a larger cache, a very high hit rate could be achieved for all traces. If the network is rarely used, then adapting block caching may not be in effect.

Trace 1



Trace 2



Trace 3

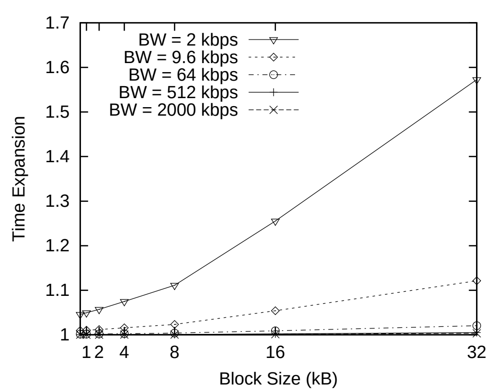


Figure 6.1: Effect of Block Size on Time Expansion

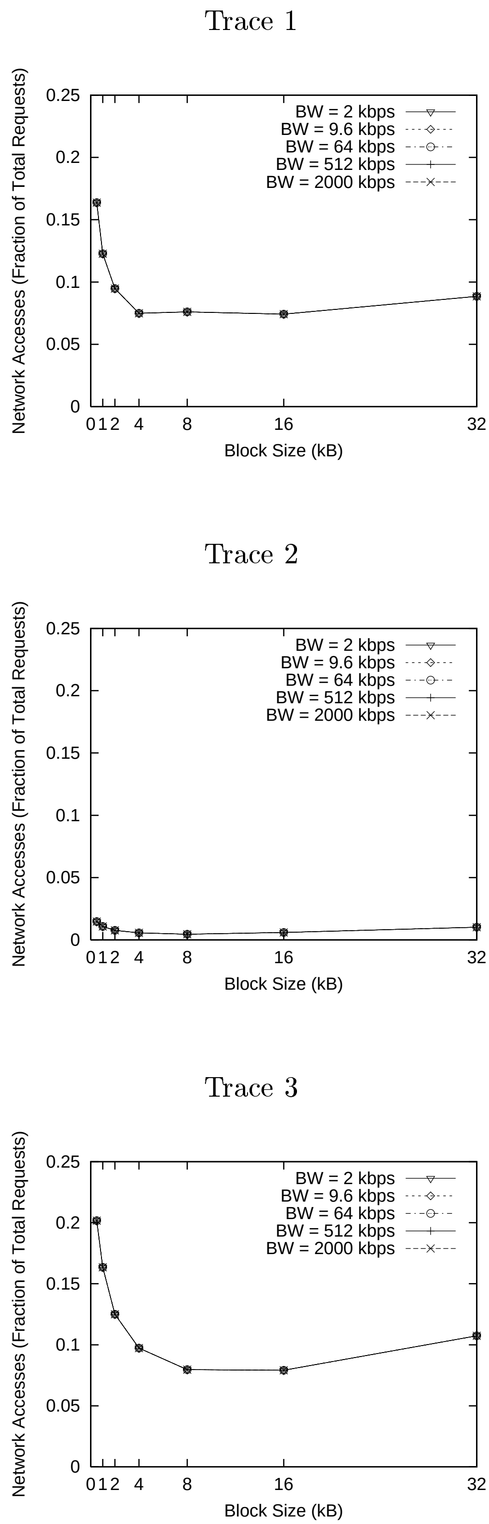


Figure 6.2: Effect of Block Size on the Number of Network Accesses

number of network accesses as discussed above.

Figure 6.2 shows the number of network accesses at different bandwidths as a function of block size. For each trace, the number of network accesses for all five bandwidths shows the same trend. This is because when a certain block size is used, the same set of cache misses will happen for a certain trace regardless of the available bandwidth. But at a lower bandwidth, it takes longer to complete all the user's activities, as shown in Figure 6.1.

In Figure 6.2, the same trend can be observed for the three traces in the number of network accesses when increasing block size. Up to a point, when the block size increases, the number of network accesses decreases. After that point, as the block size increases, the number of network accesses increases as well. When the block size is small, increasing the block size will increase the cache hit rate, and the number of network accesses is reduced as a result. If the block size is too big, however, increasing the block size may result in the eviction of other cache entries because of the limited size of the cache. This may force subsequently referenced data to be kicked out and need to be retrieved again from the server, thus increasing the number of network accesses. For Trace 1 and Trace 3, the change point is at a block size of 16 kB. For Trace 2, this point is at 8 kB. It is also observed that for Trace 2, at any block size even at 16 kB, the number of network accesses is only a very small fraction of the total requests. Then 16 kB will be considered as the change point in the number of network accesses. Thus at a high bandwidth such as 2 Mbps, if it is necessary to bring in a big block to reduce the accesses to the network, the size of the block should not exceed 16 kB.

Considering both the time expansion and the number of network accesses, the appropriate amount of data to be brought in at different available bandwidths is chosen for adaptive block caching. For the lowest bandwidth considered (2 kbps), the block size will be set to 1 kB, and for the highest bandwidth considered (2 Mbps), the block size will be set to 16 kB. For other bandwidths, the block size will be between 1 kB and 16 kB. In the subsequent experiments, for bandwidths of 512 kbps, 64 kbps, and 9.6 kbps, the block size will be set to 8 kB, 4 kB, and 2 kB, respectively, for one option of adaptive block caching.

6.2 Experimental Design

The experimental design for this part of the research followed the same basic design as for the experiments of adaptive write back. The same three seven-day traces were used. Hoarding was performed for a 24-hour period at the start of the simulation on a block caching basis. The block size was the same as that used for non-adaptive block caching. When hoarding, each data block that was requested in the first 24 hours and its immediate neighbour blocks were brought into the client's cache. After hoarding, the client had different bandwidth connections to the file server at different times. As the available bandwidth changes, the size of the block (the amount of data to be brought in) changes as well. The same randomly generated bandwidth sequence was used for this part of the experiments. The LRU replacement policy was implemented when hoarding and while mobile. The size of the cache ranged from 0.05 MB to 2 MB.

The performance of adaptive block caching was compared with that of non-adaptive block caching in which the block size was kept fixed no matter what bandwidth was available. Experiments were performed on three approaches to adaptive block caching. Adaptation-1 used the settings for block size as described in Section 6.1: for bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps, the block size was set to 16 kB, 8 kB, 4 kB, 2 kB, and 1 kB, respectively. This approach to adaptation is considered having appropriate settings. Two other levels of adaptive block caching, adaptation-2 and adaptation-3, which are considered as over-adapted and under-adapted respectively, were used to compare to adaptation-1. The settings of the block size for different approaches to block caching are summarized in Table 5.2.

Table 6.2: Block Size Settings

Bandwidth (kbps)	2000	512	64	9.6	2
Non-Adaptation (kB)	4	4	4	4	4
Adaptation-1 (kB)	16	8	4	2	1
Adaptation-2 (kB)	32	16	8	4	1
Adaptation-3 (kB)	8	4	4	2	2

6.3 Performance with Adaptive Block Caching

6.3.1 Varying Cache Size

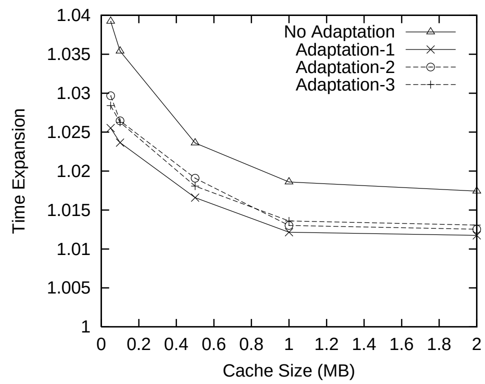
In this section, the performance of adaptive block caching is examined and compared with non-adaptive block caching at various sizes of the cache. In non-adaptive block caching, the block size uses the standard value of 4 kB, which is also the block size used when hoarding. The simulation parameters and levels used for these experiments are summarized in Table 6.3. 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 policy was implemented for all approaches of block caching in this part of experiments.

Figures 6.3 through 6.11 show the performance impact of adaptive block caching at various sizes of the cache.

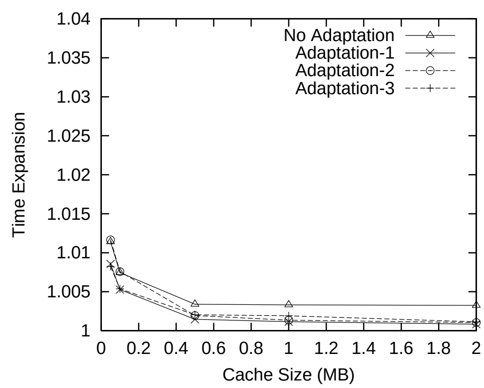
Time Expansion

Figure 6.3 shows the performance impact of adaptive block caching on time expansion. Except for adaptation-2 with Trace 2, all levels of adaptive block caching result in a smaller time expansion than non-adaptive block caching at different cache sizes ranging from 0.05 MB to 2 MB. In adaptive approaches, the amount of data to be brought in dynamically changes as the available bandwidth changes. At a higher bandwidth, more data is brought in, and since data transfer is faster, this will have little impact on increasing the read service time. At a lower bandwidth, less data is brought in compared to the non-adaptive block caching. The read service time is

Trace 1



Trace 2



Trace 3

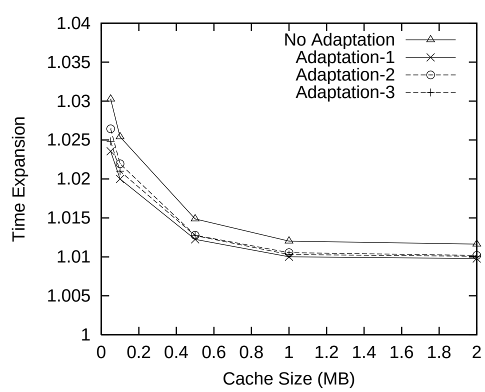


Figure 6.3: Time Expansion

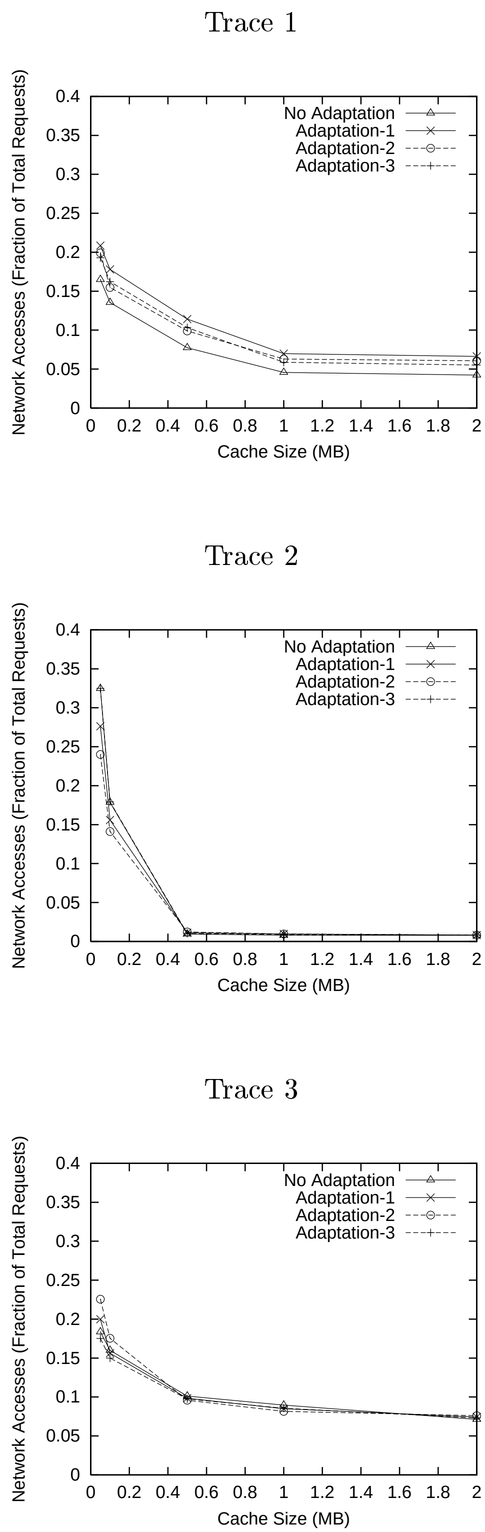
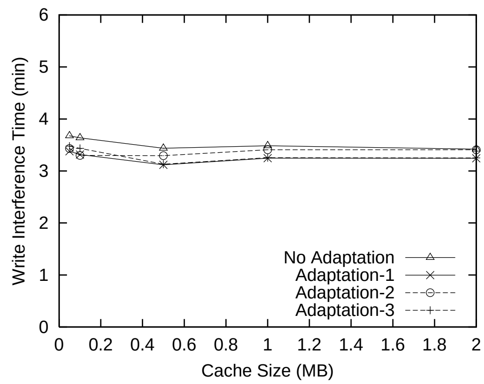
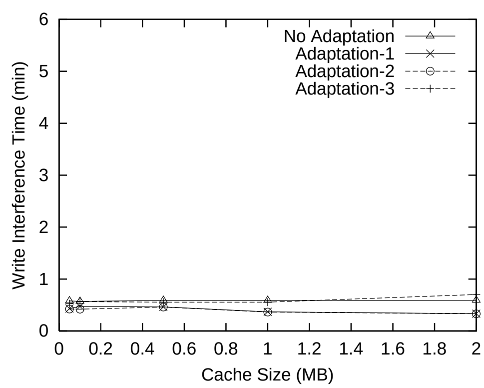


Figure 6.4: Number of Network Accesses

Trace 1



Trace 2



Trace 3

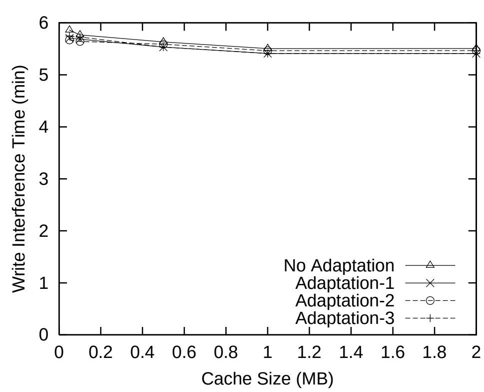
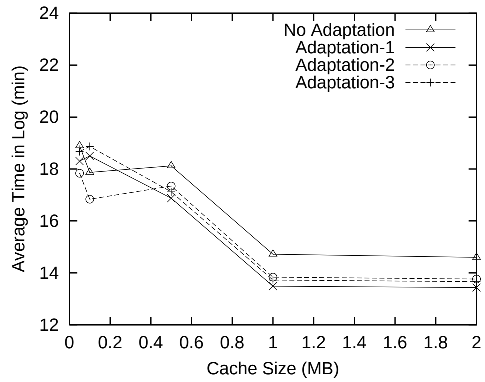
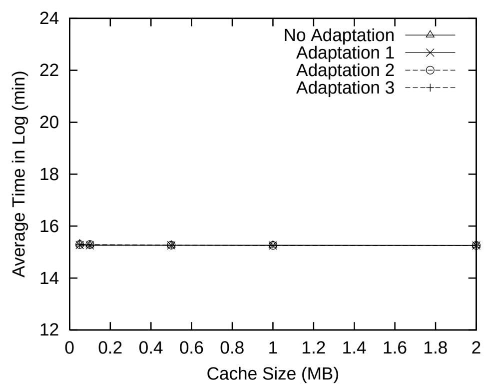


Figure 6.5: Write Interference Time

Trace 1



Trace 2



Trace 3

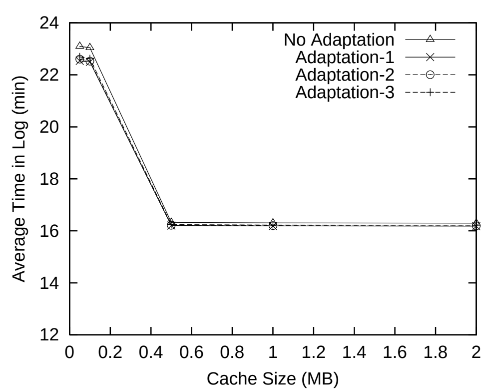
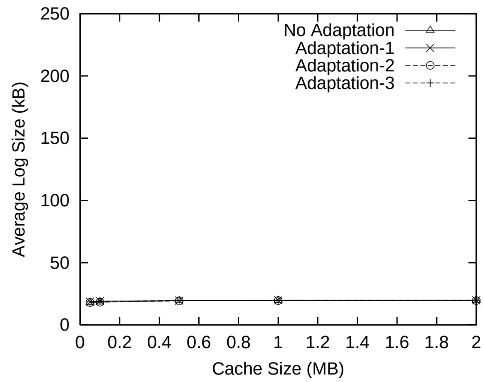
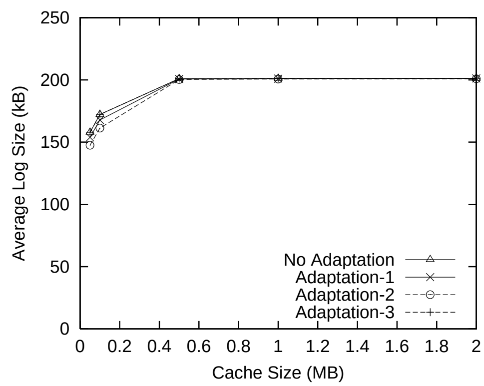


Figure 6.6: Average Time Updates Remain in Log

Trace 1



Trace 2



Trace 3

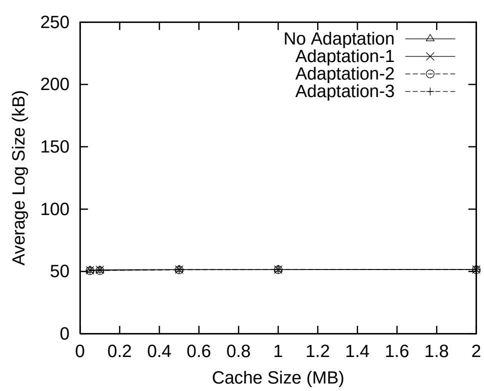
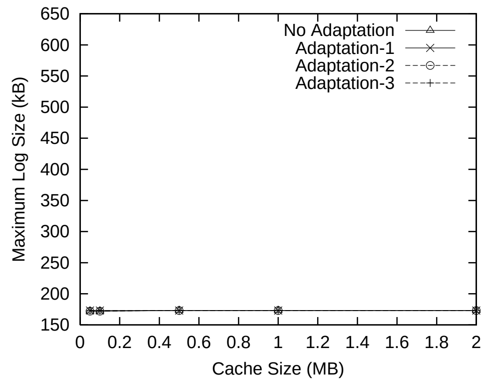
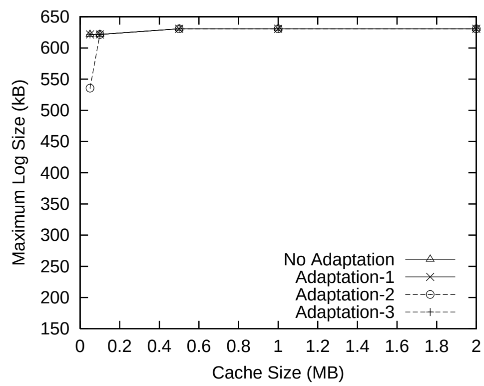


Figure 6.7: Average Log Size

Trace 1



Trace 2



Trace 3

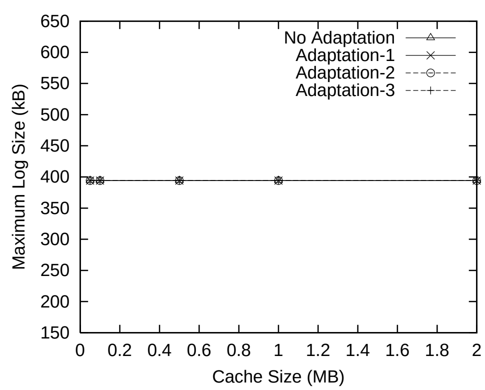
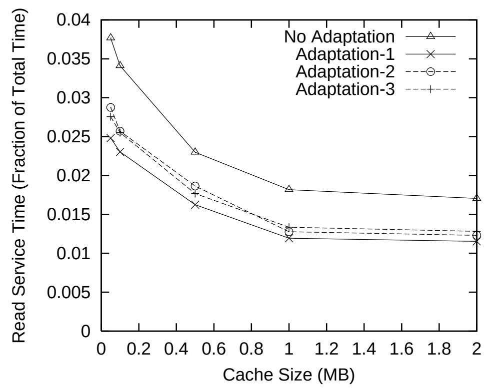
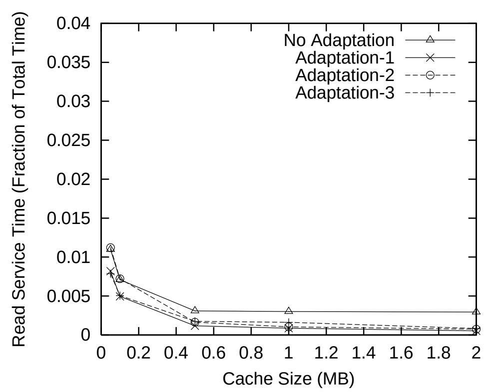


Figure 6.8: Maximum Log Size

Trace 1



Trace 2



Trace 3

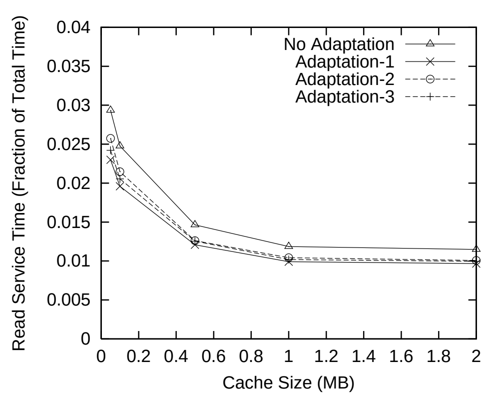


Figure 6.9: Read Service Time

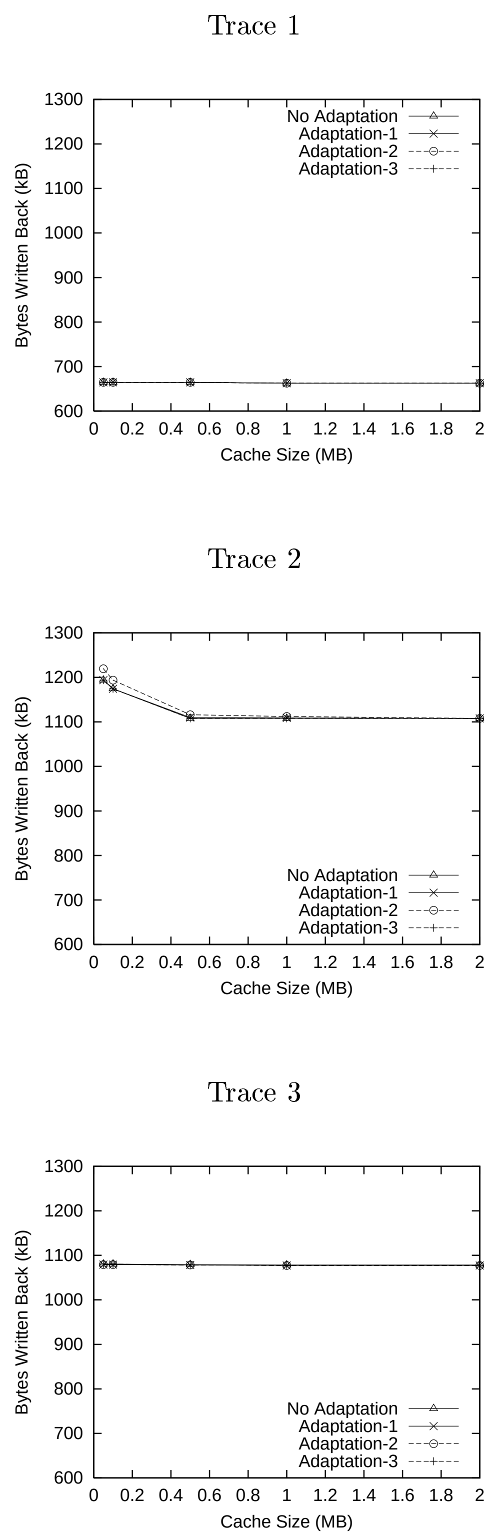
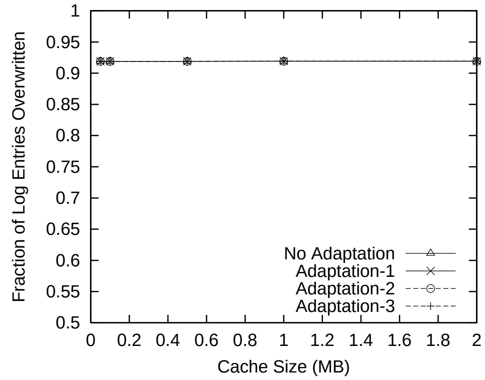
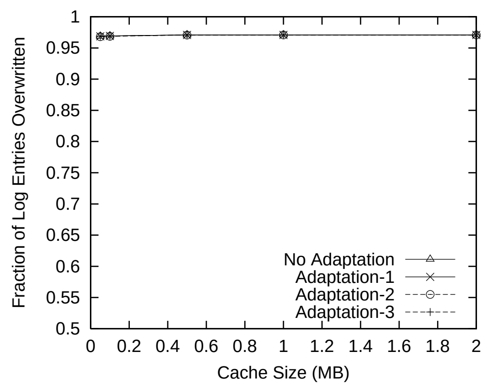


Figure 6.10: Bytes Written Back

Trace 1



Trace 2



Trace 3

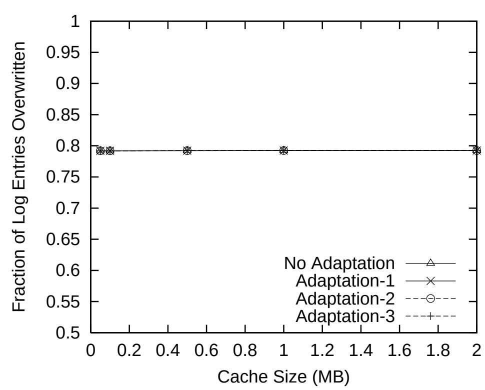
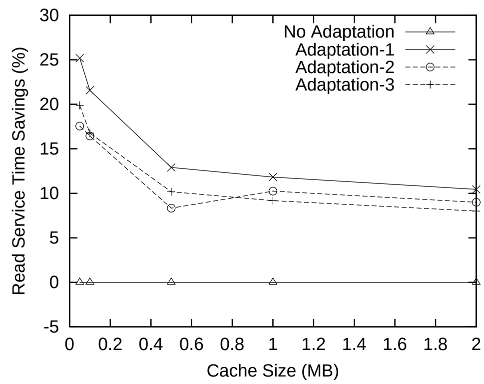
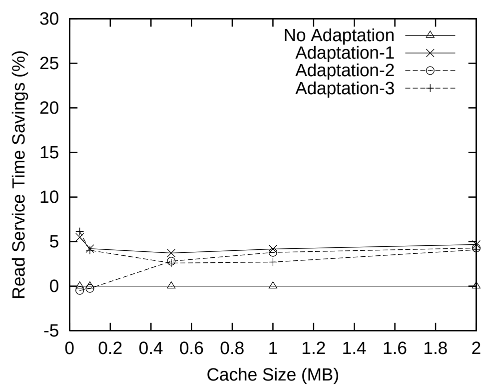


Figure 6.11: Log Entries Overwritten

Trace 1



Trace 2



Trace 3

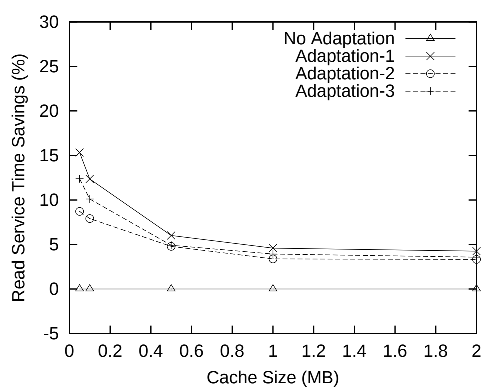


Figure 6.12: Time Saving

Table 6.3: Factors and Levels Used (Varying Cache Size)

Factors	Levels
Hoarding Period	24 hours
Hoarding Policy	LRU
Replacement Policy	LRU
Caching Unit	Block
Cache Size (MB)	0.05, 0.1, 0.5, 1, 2
Write-back Policy	Delayed Write Back (non-adaptive, delay time 15 minutes)
Adaptation Strategy	Change block size when bandwidth changes
Adaptation Direction	Increase size of the block when bandwidth increases

reduced by transferring only the required data. With Trace 2, adaptation-2 results in a slightly longer time expansion than non-adaptive block caching when the cache size is smaller than 0.5 MB. This may result from the over-adapted characteristic of adaptation-2 and the reference behaviour in Trace 2. Adaptation-1 is considered to have the appropriate settings in the sizes of the block, and it results in the lowest time expansion for each of the traces. Adaptation-2 and adaptation-3 are considered to be over-adapted and under-adapted to the bandwidth changes, respectively. Their performance in terms of time expansion is between non-adaptive and adaptation-1. Although there are performance differences among the adaptive approaches, all the adaptive approaches result in performance that is no worse than the non-adaptive block caching.

In Figure 6.3, the reduction in time expansion by adaptive block caching appears small. This is because the "idle time" in each trace was included to calculate the time expansion. The savings in read service time by adaptive approaches of block caching take place only in the user's active time. The actual number of active hours for each trace is much shorter, as shown in Table 4.1. If only the active hours were used to calculate the reduction in the time servicing requests, the percentage reductions at a cache size of 1 MB would be 12%, 4%, and 5% for Trace 1, Trace 2, and Trace 3, respectively. If a smaller cache is used, the reduction is more significant.

The percentage of time savings from trace active time is calculated and shown in Figure 6.12.

Number of Network Accesses

In adaptive block caching, at lower bandwidths, a block smaller than in the non-adaptive approach is brought into the cache each time. The number of network accesses may be increased if the client references big blocks. Although bringing in big blocks at high bandwidths can reduce the number of network accesses, it cannot outperform the increase in network accesses at lower bandwidths, because of the need for more frequent cache replacement. The increase may become more significant when the client has more occasions to use low bandwidth connections such as 2 kbps and 9.6 kbps. But with appropriate settings in the block size in response to the bandwidth changes, this increase may be kept small and avoided.

Figure 6.4 shows the number of network accesses from different approaches of adaptive block caching. For Trace 2 and Trace 3, adaptive approaches result in little increase in the number of network accesses compared to the non-adaptive block caching. As the cache size increases, the difference in the number of network accesses among different approaches is diminished. For Trace 1, the increase is also at an acceptable level. At a cache size of 1 MB, the increase is less than 2.5%.

Write Interference Time

Figure 6.5 shows the write interference time from different approaches of block caching. Overall, the write interference time is fairly small for all three traces and all approaches to block caching. Different approaches to block caching have little impact on write interference time. The same is true for the cache size, due to the same write back policy with the same parameters being used throughout.

Time in Log

The time updates remain in the log file is determined mainly by the minimum delay time in the write-back policy. When the same minimum delay time is used in the write-back policy, at a certain size of the cache, different approaches of block caching should have little impact on the time updates remain in the log. This is confirmed in Figure 6.6. This figure shows that, for each of the three traces, different approaches to block caching result in similar results for the average time updates remain in the log file, at each given size of the cache.

As the cache size increases, the time in log decreases for most traces. With a smaller cache, there are more cache misses and more data transfers. The updates will have to wait for the reads to complete before being written back, even when the updates have stayed enough time in the log. This increases the time updates remain in the log at a smaller cache.

Log Size

Figures 6.7 and 6.8 show the average log size and maximum log size. The difference in the size of the log across different approaches to block caching at each size of the cache is insignificant, since there is little difference in the average time updates remain in the log file. As the cache size increases, the log size increases as well.

Read Service Time

Figure 6.9 shows the read service time for different approaches to block caching for the three traces. The read service time is a better metric than time expansion to reflect the performance of different approaches to block caching. The read service time has the same trend as time expansion for each trace. The read service time and the write interference time are the two factors that affect the time expansion. The small write interference time (see Figure 6.5) and the little difference in the write interference time across the different approaches to block caching make the read service time a main factor affecting the time expansion, and so the read service time has the same trend as the time expansion.

Adaptive approaches to block caching require less time to service the requests than the non-adaptive approach, except for Trace 2 where adaptation-2 results in a slightly higher read service time than non-adaptive block caching when the cache is smaller than 0.5 MB. The reason for this is the same as described in the *Time Expansion* subsection above.

Adaptation-1 reduces the read service time the most for all traces since it is properly adapted to bandwidth changes. Adaptation-2 and adaption-3 are over-adapted and under-adapted, so their performance with respect to read service time is not as good as adaptation-1.

Bytes Written Back

Figure 6.10 shows the bytes written back by the different approaches. The bytes written back is also affected mainly by the write-back policy implemented. As the same write-back policy with the same parameters is applied for all approaches to block caching, there is little difference in the bytes written back across the different approaches at a given cache size.

Log Entries Overwritten

For all approaches to block caching, the non-adaptive version of delayed write back is implemented. The 15-minute minimum delay time keeps file updates remain in the log long enough for them to be overwritten by subsequent updates. This makes the log entries overwritten very similar across different approaches, as shown in Figure 6.11. The size of the cache also has almost no impact on log entries overwritten.

6.3.2 Varying Block Size

In the last section, the performance impact of adaptive block caching was evaluated at different sizes of the cache. In this section, the performance impact of adaptive block caching is examined at different sizes of data block. A 0.5 MB cache is used in these experiments, and the block size ranges from 0.5 kB to 8 kB for non-adaptive block caching. Other parameters settings are the same as in the last section. The same three levels of adaptive block caching implemented in the last section are applied. Figure 6.13 through 6.21 show the performance of different approaches to block caching as the block size varies.

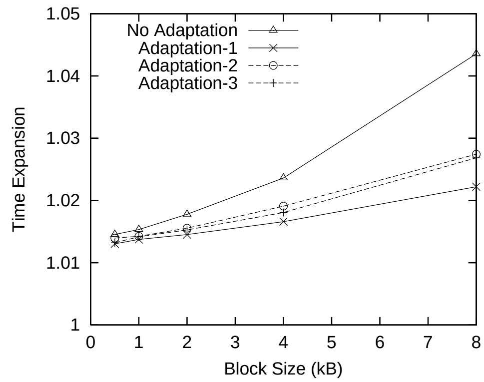
Time Expansion

Figure 6.13 shows the time expansion for adaptive block caching at different block sizes in comparison to non-adaptive block caching. As discussed in the previous section, adaptive approaches to block caching reduce the time expansion. From this figure, for the three traces, all three approaches to adaptive block caching result in a smaller time expansion than non-adaptive block caching. As the block size increases, the time expansion for each approach increases as well, with the non-adaptive approach having the fastest rate of increase. Adaptation-1, considered as having the appropriate settings for block size at different bandwidths, has the lowest rate of increase. The rate of increase in time expansion from the other two approaches to adaptive block caching falls between the non-adaptive block caching and adaptation-1, but is fairly low compared with the non-adaptive approach. This indicates that when a big data block is implemented in the system, it is more beneficial to make block caching adaptive to changes in available bandwidth.

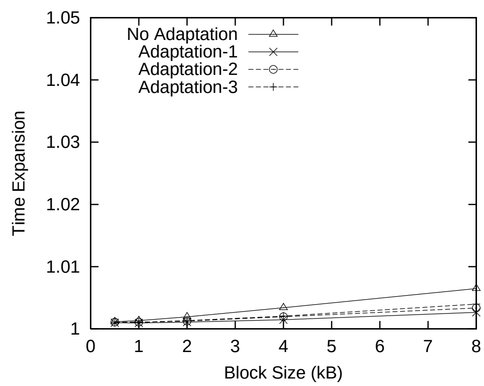
Number of Network Accesses

Figure 6.14 shows the number of network accesses for different approaches to block caching. For Trace 2, since at a cache size of 0.5 MB, the hit rate at any block size can reach a high percentage (around 97%), the block size and the different approaches to block caching have little impact on the number of network accesses. For Trace 1 and Trace 3, when the block size increases, the number of network accesses decreases for all approaches. This is because taking more data to the cache is more likely to increase the hit rate, but this increases the read service time and time expansion,

Trace 1



Trace 2



Trace 3

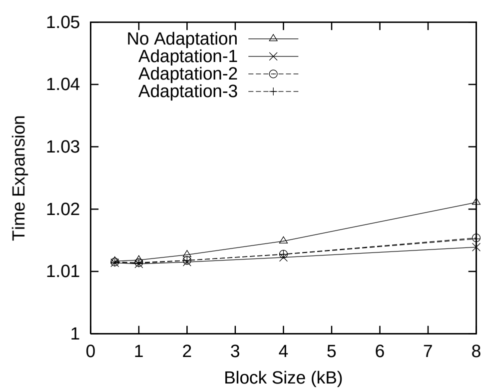


Figure 6.13: Time Expansion

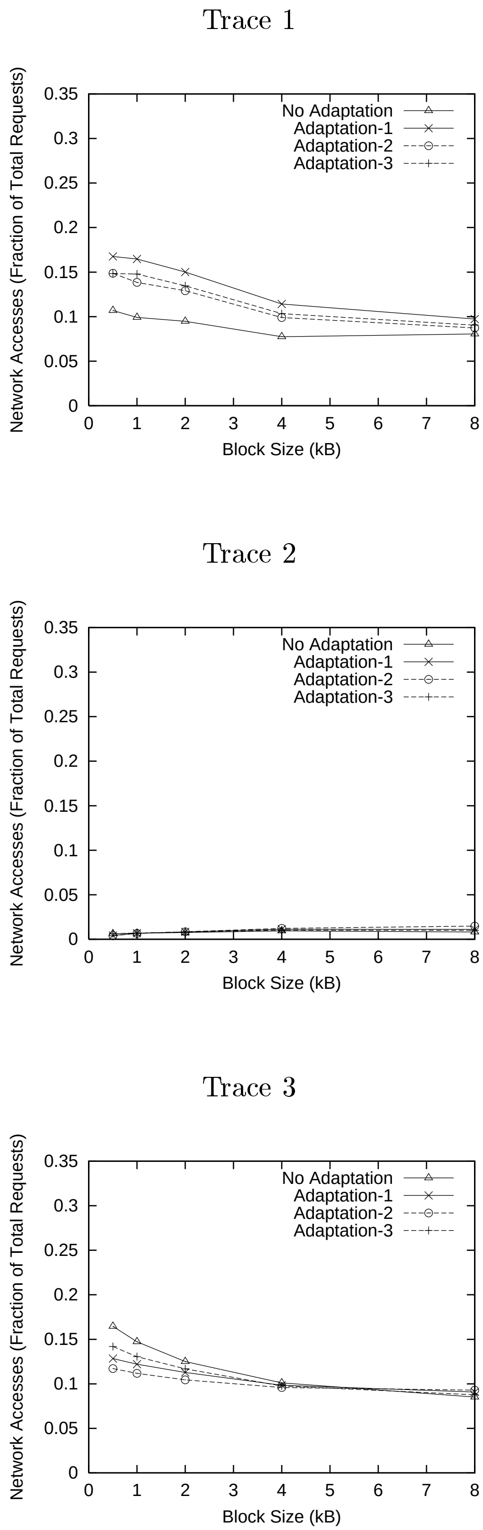
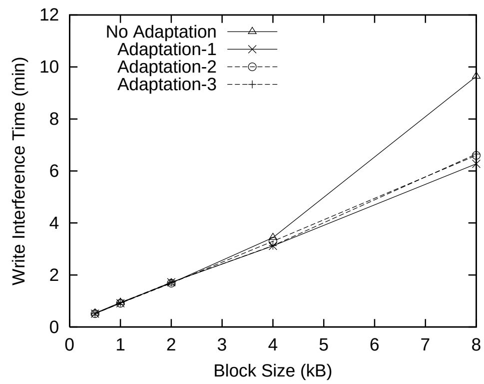
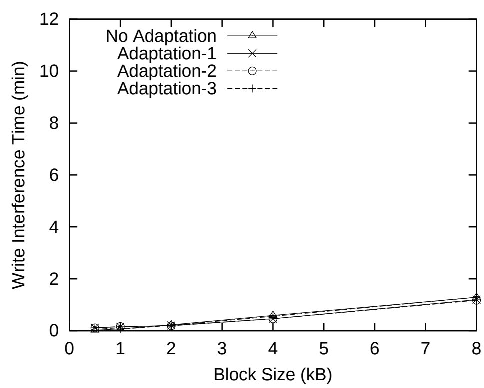


Figure 6.14: Number of Network Accesses

Trace 1



Trace 2



Trace 3

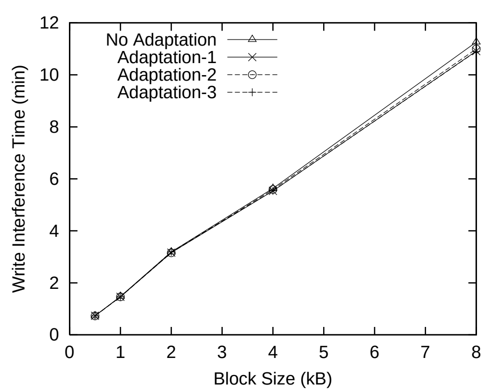


Figure 6.15: Write Interference Time

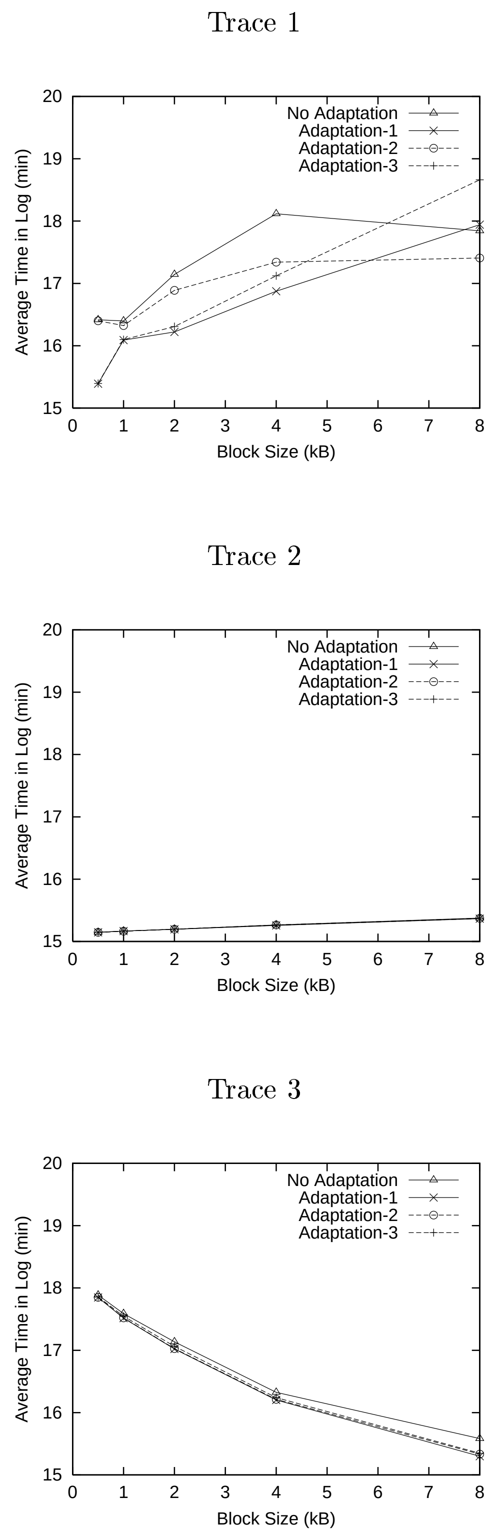
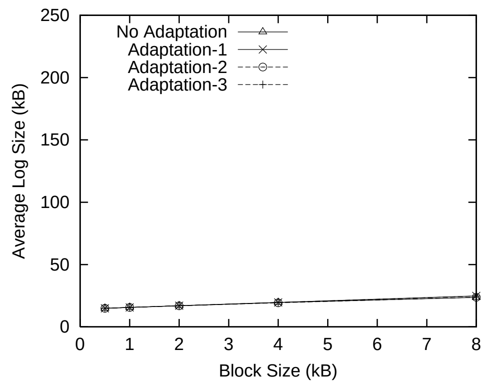
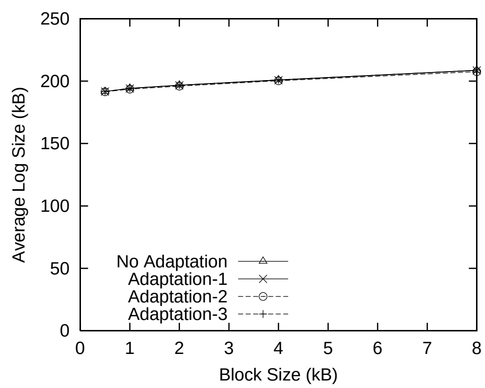


Figure 6.16: Average Time Updates Remain in Log

Trace 1



Trace 2



Trace 3

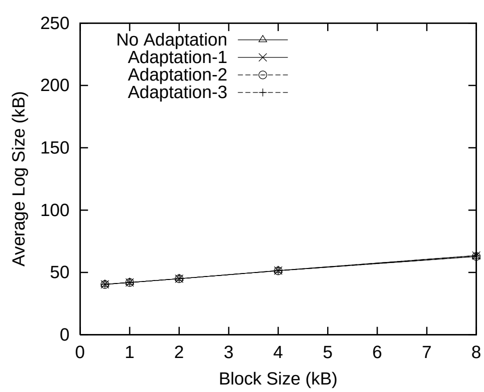
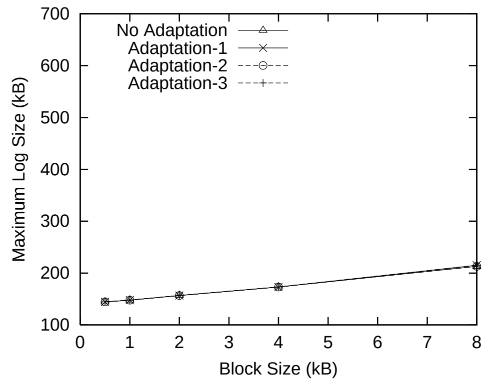
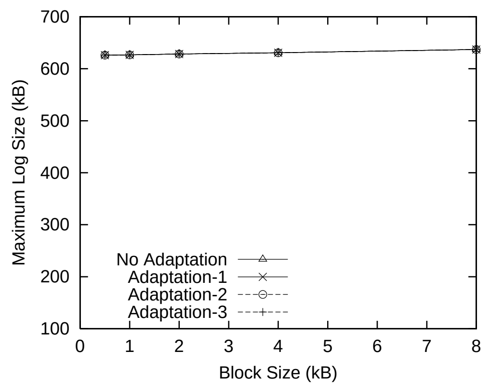


Figure 6.17: Average Log Size

Trace 1



Trace 2



Trace 3

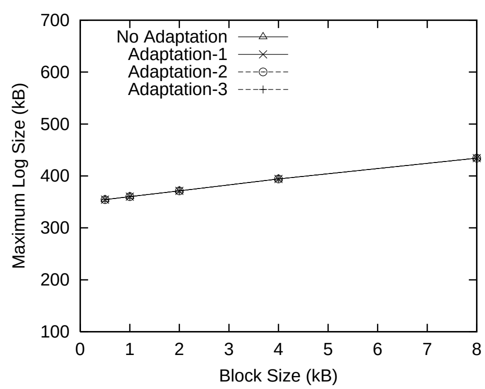
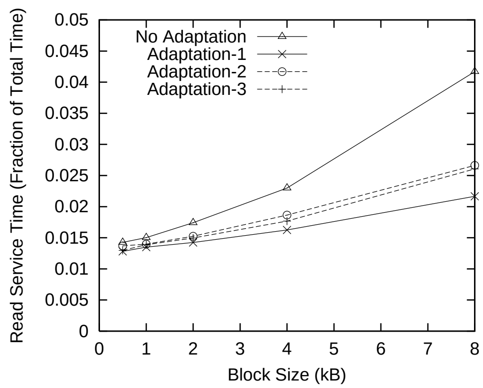
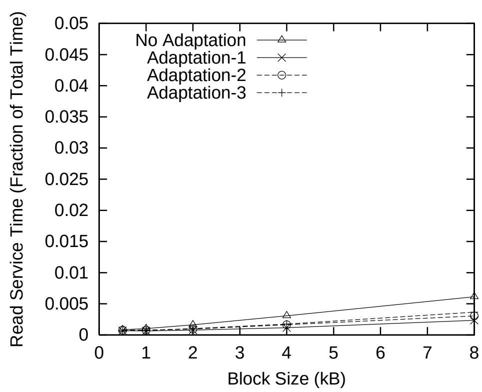


Figure 6.18: Maximum Log Size

Trace 1



Trace 2



Trace 3

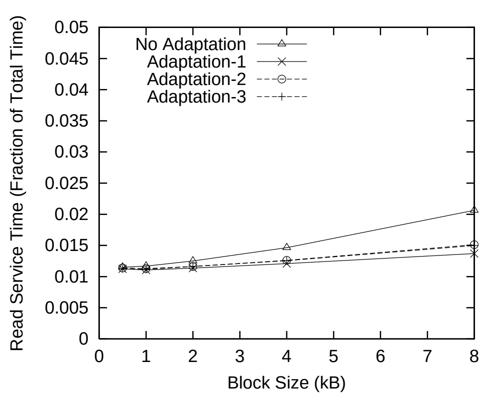


Figure 6.19: Read Service Time

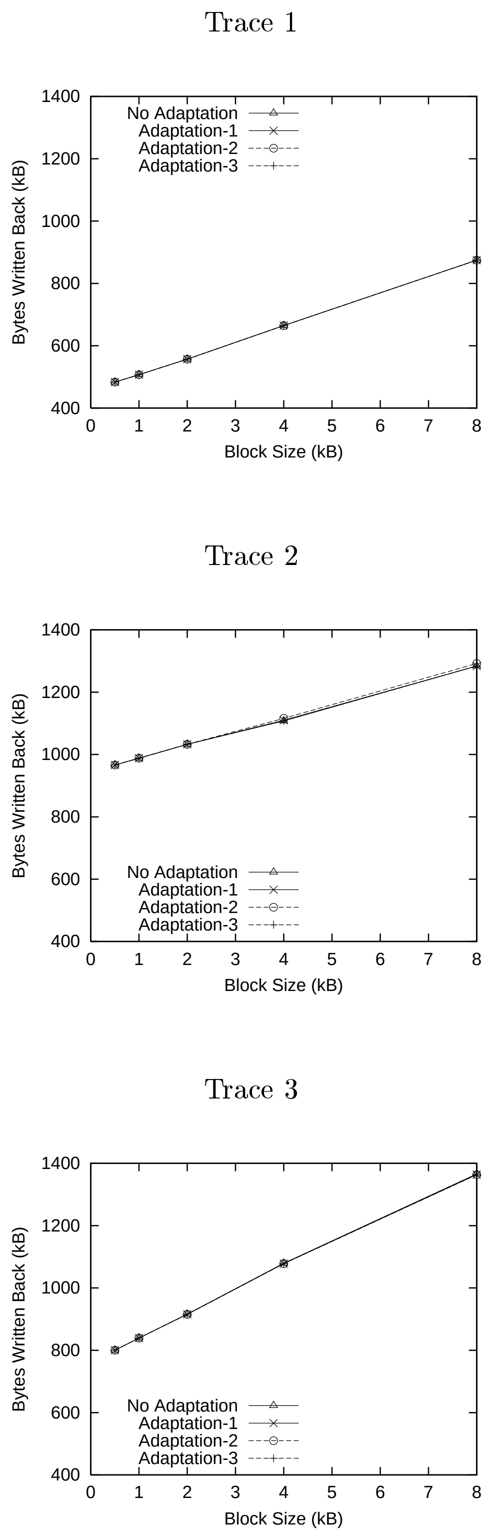
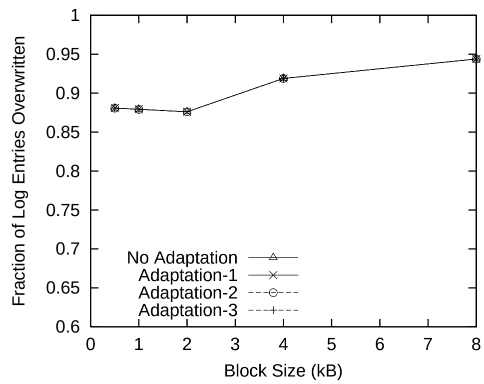
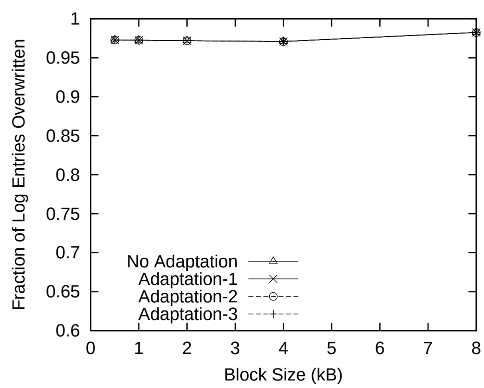


Figure 6.20: Bytes Write Back

Trace 1



Trace 2



Trace 3

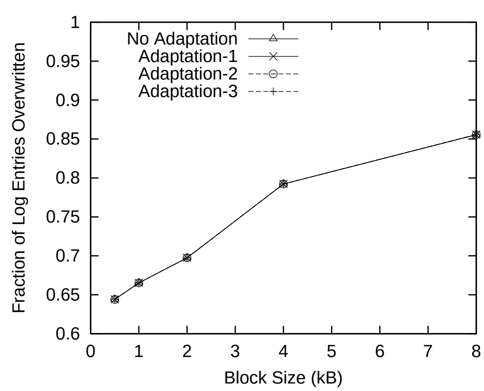


Figure 6.21: Log Entries Overwritten

as shown in Figures 6.19 and 6.13. Different approaches to block caching show different behaviours for Trace 1 and Trace 3 in terms of the number of network accesses. These two traces may behave differently in the amount of data referenced each time. For Trace 1, adaptive approaches result in more network accesses than the non-adaptive approach. There may be many events in this trace that reference big blocks, such as 2 kB or 4 kB. For Trace 3, adaptive approaches result in a smaller number of network accesses than the non-adaptive approach. This may be caused by more frequent cache replacements in the non-adaptive approach. The difference in the number of network accesses between adaptive approaches and the non-adaptive approach of block caching becomes small as the block size increases.

Write Interference Time

Figure 6.15 shows the impact of different approaches to block caching on write interference time as block size increases. When the block size increases, the write interference time increases for all approaches to block caching for the three traces. This is because as the block size increases, the size of the log entries increases as well. Writing back a bigger update is more likely to interfere with the read requests.

Due to the same conservative delayed write back policy is implemented with the same parameters for all approaches to block caching, the different approaches to block caching have little impact on the write interference time at each give block size.

Time in Log

Figure 6.16 shows the average time updates remain in the log file from different approaches to block caching as the block size varies. As discussed previously, the time in log is mainly determined by the minimum delay time in the write-back policy implemented. For all three traces, there is little difference in the average time updates remain in the log across different approaches at each block size, since the 15-minute minimum delay time is applied throughout. As the block size increases, there are changes in the average time in log, but the trends are different for different traces. For Trace 1 and Trace 2, the time in log increases slightly as the block size increases. For Trace 3, the time in log decreases. It may be a little hard to compare the average time in log here. A few factors affect on the average time in log as the block size increases. The trace characteristics, such as the amount of data updated in each write event, the inter-event time, etc., may be responsible for these different trends. The number of log entries are different when different block size is used, as the size of the log entries is related to the block size. The number of log entries that are still in the log when an expected disconnection occurs are different at different block size, and the period of time they have stayed in the log has large effect on the average time in log, because these log entries are written back right before the disconnection without concerning the minimum delay time in the write back policy.

Log Size

Figures 6.17 and 6.18 show the average log size and maximum log size caused by different approaches to block caching at different block sizes. When the block size increases, the size of the log entries increases as well, so the log size increases for all approaches. Different approaches to block caching have little impact on the log size, because there is little difference in the average time updates remain in the log across all the approaches.

Read Service Time

Figure 6.19 shows the read service time from different approaches to block caching. The trends are the same as time expansion for the three traces because the write interference time is small. The non-adaptive approach needs a longer time to service the read requests than adaptive approaches at any block size. As the block size increases, the read service time increases for all approaches, and the non-adaptive approach has the fastest rate of increase. As the block size increases, the difference between adaptive approaches and the non-adaptive approach increases as well. The reason for these observations is the same as described for time expansion.

Bytes Written Back

Different approaches to block caching have little impact on the bytes written back if the same write-back policy is implemented. As shown in Figure 6.20, the bytes written back by different approaches to block caching are the same at any given block size. But the size of the block has an impact on bytes written back. When the block size increases, the size of the log entries increases, so the bytes written back increase for all approaches.

Log Entries Overwritten

Figure 6.21 shows the percentage of log entries overwritten. The different approaches to block caching have no impact on log entries overwritten. The block size, however, has an impact on the log entries overwritten. For Trace 3, the percentage of log entries overwritten increases as the block size increases. This is because when the sizes of log entries (updates) increases, the log entries are more likely to be overwritten by subsequent updates. For Trace 1 and Trace 2 that have higher percentage of write requests than Trace 3, when the block size is big, the percentage of log entries overwritten also increases as the block size increases. But when the block size is small, the percentage of log entries overwritten slightly decreases when the block size increases. This is likely because of the increased read service time (as block size increases) which allows some log entries to become eligible to be written back before they are overwritten.

6.3.3 Adaptive Block Caching in Combination with Adaptive Write Back

The experiments on adaptive block caching described in previous two sections were performed to examine the performance impact of the adaptive block caching without adaptive write back. In this section, experimental results from adaptive block caching in combination with adaptive write back are presented.

Adaptive write back was shown to perform well in Chapter 5. Adaptation-1 was implemented along with adaptive approaches to block caching in a new set of experiments. In this adaptive write back approach, the minimum delay time was set to 0.5 minutes, 0.5 minutes, 5 minutes, 10 minutes, and 15 minutes for bandwidths of 2 Mbps, 512 kbps, 64 kbps, 9.6 kbps, and 2 kbps, respectively. In the non-adaptive approach to block caching, the non-adaptive write back with 15 minutes minimum delay time was still applied regardless of the available bandwidth. The goal of these experiments was to examine if adaptive block caching would conflict with adaptive write back in providing performance benefits to mobile users. All other parameter settings were the same as in Section 6.3.1.

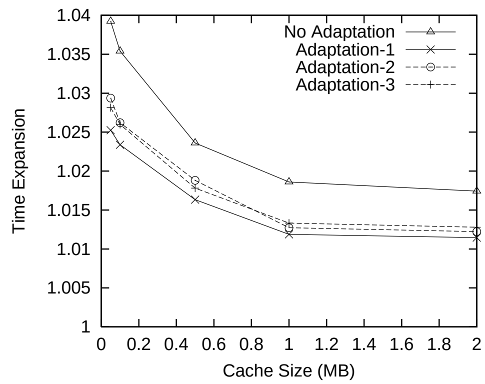
Figures 6.22 through 6.30 show the performance from adaptive approaches to block caching in combination with adaptive write back, with comparison to non-adaptive block caching in combination with non-adaptive write back. From these figures, it is clear that there is little conflict between adaptive block caching and adaptive write back. Users can get all the benefits from adaptive block caching and adaptive write back.

Comparing Figures 6.22 and 6.23 to Figures 6.3 and 6.9 in Section 6.3.1, adaptive block caching in combination with adaptive write back provides the same reduced time expansion and read service time as when combined with non-adaptive write back. In other words, adaptive write back did not degrade the performance benefits from adaptive block caching in providing reduced time expansion and read service time.

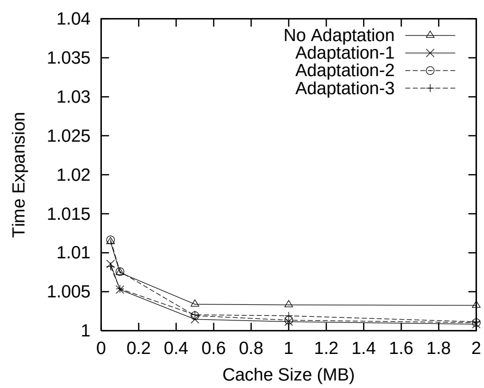
Adaptive write back reduces the average time updates remain in the log file and the size of log file. From Figure 6.26, 6.27 and 6.28, mobile users can still get significantly reduced update reintegration time and log file size for all three traces. Adaptive block caching did not affect the performance benefits from adaptive write back.

When adaptive write back is combined with adaptive block caching, the impact on the write interference time is similar to adaptive block caching with non-adaptive write back (comparing Figure 6.25 to Figure 6.5). The write interference time is very small for all approaches and all traces. There is little difference between adaptive

Trace 1



Trace 2



Trace 3

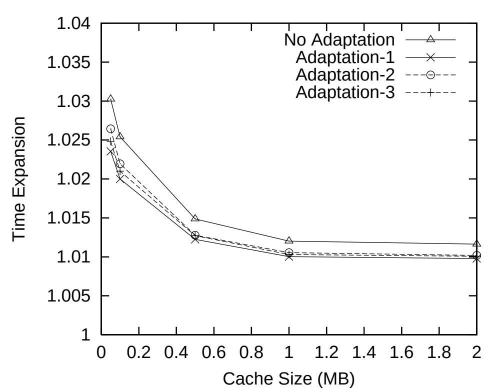
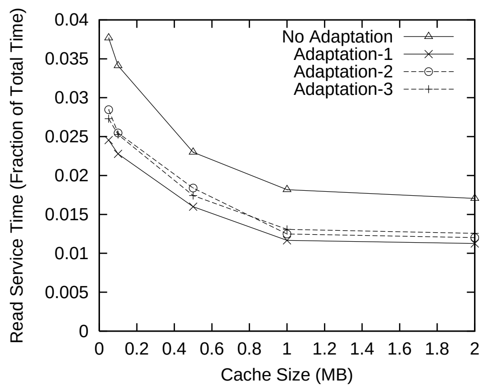
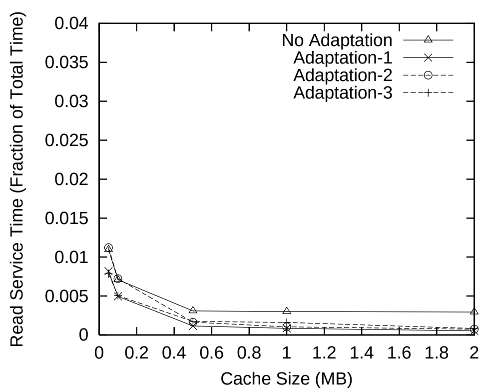


Figure 6.22: Time Expansion

Trace 1



Trace 2



Trace 3

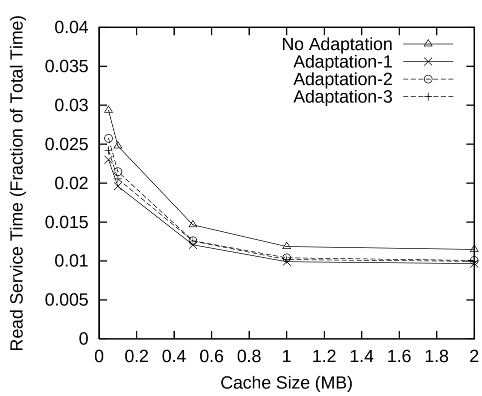


Figure 6.23: Read Service Time

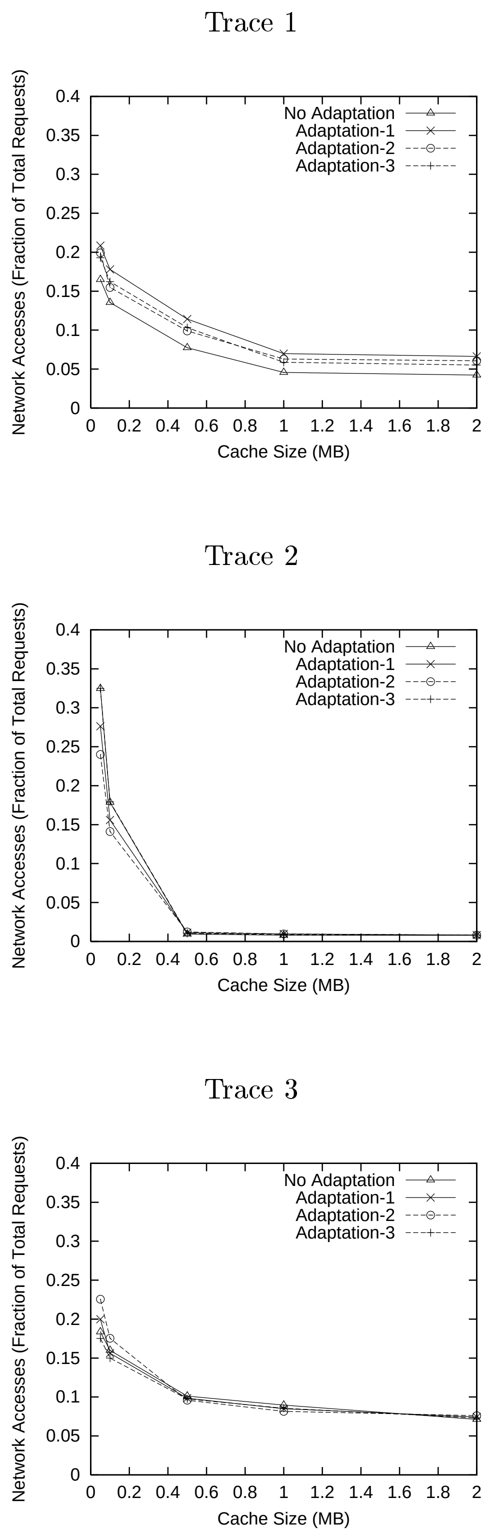


Figure 6.24: Number of Network Accesses

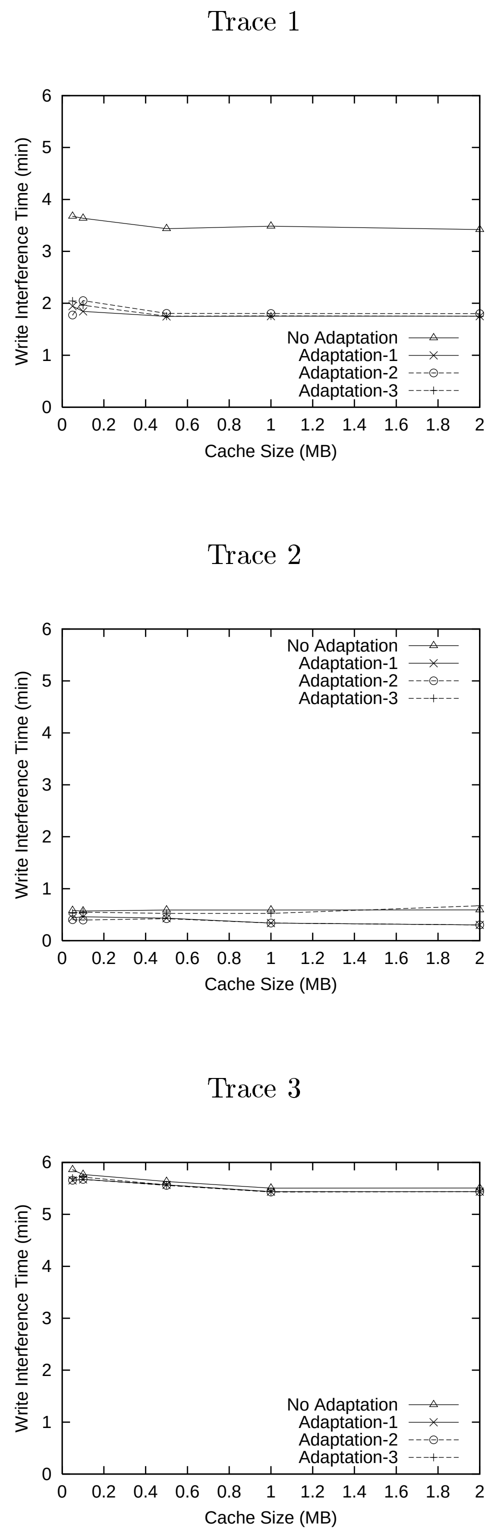
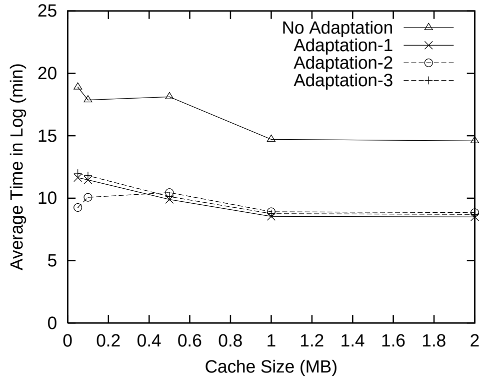
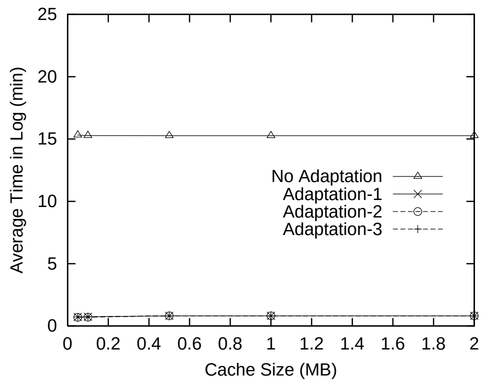


Figure 6.25: Write Interference Time

Trace 1



Trace 2



Trace 3

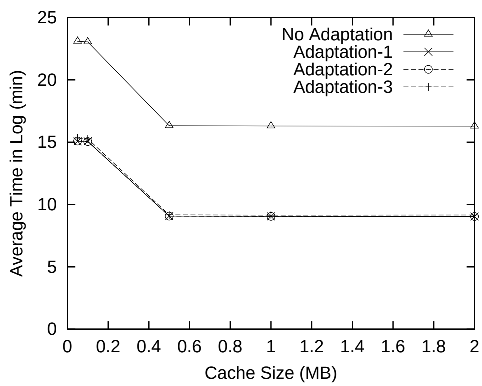
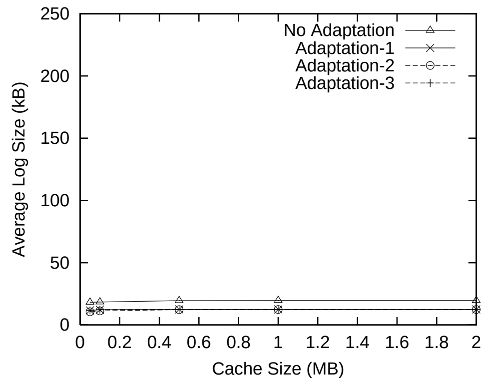
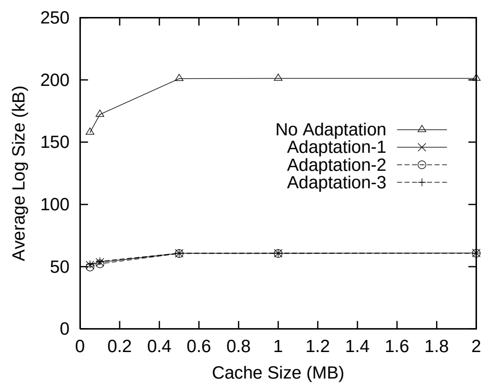


Figure 6.26: Average Time Updates Remain in Log

Trace 1



Trace 2



Trace 3

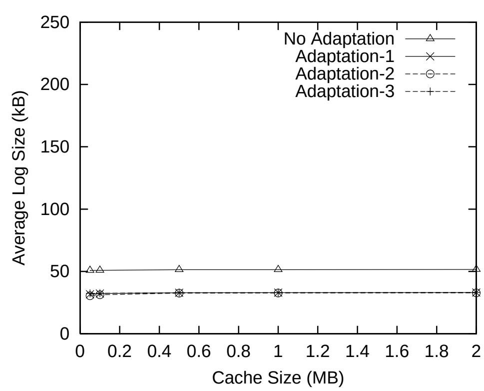
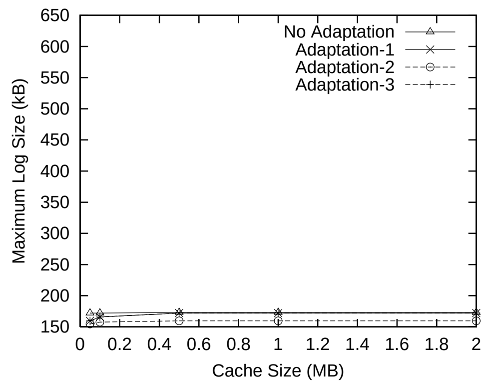
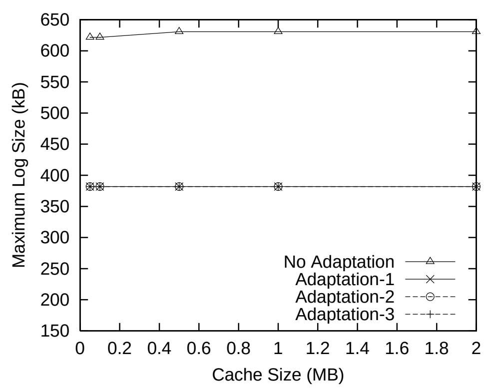


Figure 6.27: Average Log Size

Trace 1



Trace 2



Trace 3

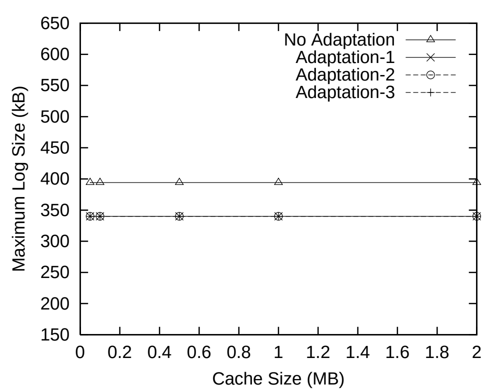
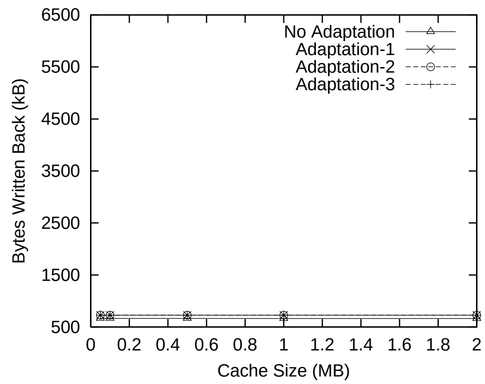
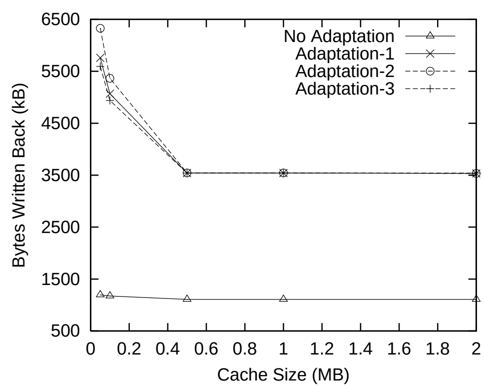


Figure 6.28: Maximum Log Size

Trace 1



Trace 2



Trace 3

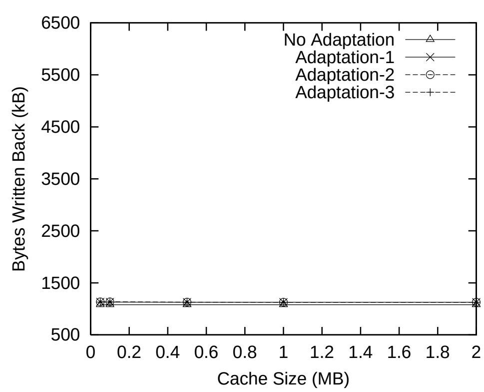
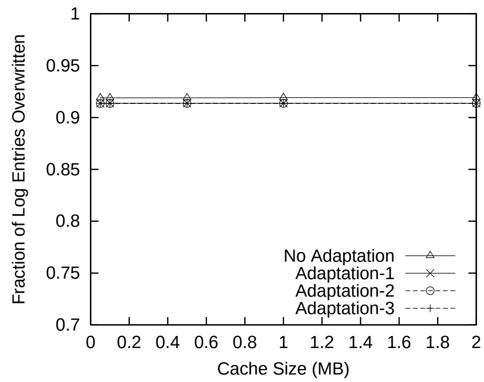
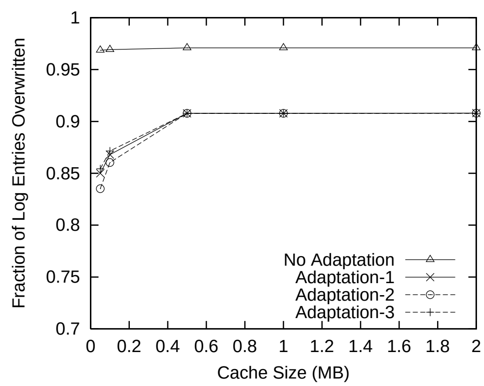


Figure 6.29: Bytes Write Back

Trace 1



Trace 2



Trace 3

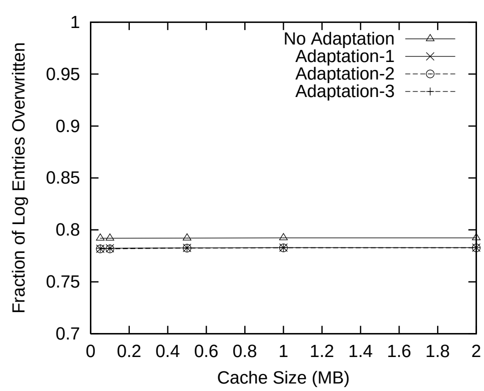


Figure 6.30: Log Entries Overwritten

and non-adaptive approaches.

The negative impact from both adaptive write back and adaptive block caching still exists, however. Adaptive write back, being more aggressive than non-adaptive write back, writes more bytes to the file server as shown in Figure 6.29. The increase in the bytes written back is also trace sensitive. The trace with the highest percentage of write requests (Trace 2) has the largest increase in bytes written back. Adaptive write back also reduces the effect of log optimization, since the updates are propagated to the file server earlier than non-adaptive write back, as shown in Figure 6.30. Also, the impact of adaptive block caching on the number of network accesses is still present. Adaptive write back has no effect on the number of network accesses, as seen by comparing Figure 6.4 to Figure 6.24.

6.4 Conclusions

Adapting the size of the block in a block caching implementation to changes in available bandwidth in the changing mobile environment attempts to provide mobile users with reduced read service time and therefore reduced time expansion. The performance impact of adaptive block caching was examined in three aspects: as a function of cache size, as a function of block size, and its impact on other adaptive approaches (in this case, adaptive write back).

The simulation results show that adapting the block size to changes in available bandwidth can reduce time expansion and read service time if the block size is appropriately set at different available bandwidths. When the bandwidth increases, the amount of data to be brought into the client cache should be increased, and when the bandwidth decreases, it should be reduced. The time savings can be substantial, adaptation-1, for example, a savings of 12% was achieved for Trace 1 at a cache size of 1 MB. With a smaller cache, the reduction in read service time by adaptive block caching is ever larger. The simulation results also show that the negative impact of adaptive block caching (for example, the number of network accesses) is small. The worst case is for Trace 1, but at a cache size of 1 MB, the increase in the number of network accesses is less than 3%. For other performance metrics, including write interference time, time in log, bytes written back, log size, and log entries overwritten, there are fewer differences between adaptive approaches and non-adaptive.

If the block size used in the system is large, it is even more important to implement adaptive block caching. The difference between adaptive approaches and the non-adaptive approach in time expansion and read service time increases as the block size increases.

The performance of adaptive block caching was also examined in combination with adaptive write back. The simulation results show that adaptive block caching and adaptive write back have little conflict. All the performance benefits from adaptive block caching and adaptive write back can be achieved while the negative impacts from these two approaches to adaptation are no worse.

Chapter 7

Summary and Conclusions

This thesis investigates the effects of adapting file cache management to bandwidth changes in the mobile environment. Two application areas, adaptive write back and adaptive block caching, have been studied. This chapter summarizes the research work that was done and presents suggestions for future research in this area.

7.1 Summary of the Thesis

The first three chapters of the thesis provide background information about mobile computing and file cache management for supporting mobile file access. The constraints and limitations in the mobile environment are presented and the need for adaptation in the constantly changing mobile environment is reviewed.

Chapter 4 investigates the impact of the changing environment on cache performance. There are many factors that change from time to time and affect the performance for mobile users, such as bandwidth, the communication expenses, and the mobile client's local disk space. File cache management strategies can adjust their behaviour when the bandwidth changes to improve performance for mobile users. Main issues arising in cache management adapting to changes in a client's connection bandwidth are discussed in this chapter. The performance of adaptive cache management is evaluated using trace-driven simulation. Chapters 5 and 6 present the results from experiments with adaptive write back and adaptive block caching, respectively.

Since read requests should be given a higher priority to access the network in mobile data access, update reintegration may be slow and this may cause problems such as write-write conflicts for shared data and data loss or damage. Adapting write back to bandwidth changes can reduce the update reintegration time. In the experiments, delayed write back techniques were used. Adaptation was realized by adjusting the minimum delay time parameter in the delayed write back policy at different bandwidth levels. The results show that adaptive write back can reduce update reintegration time significantly. As a by-product, the log size is reduced as well (which has positive implications for memory use at the client). At the same time, the negative impact from write back adaptation is modest. There is almost no extra write interference time by adapting write back, and the time expansion is kept essentially the same as in non-adaptive write back. The increase in bytes written back is also acceptable.

Mobile users always get longer read service time and time expansion than stationary users. Adapting block size to bandwidth changes seeks to reduce the read service time and therefore the time expansion by dynamically adjusting the amount of data to be transferred remotely to the client cache. Compared to non-adaptive block caching, the results show that adaptive block caching can reduce the read service time with appropriate settings for block size at different bandwidth levels. Furthermore, adaptive block caching has little conflict with adaptive write back.

7.2 Contributions

The constantly changing mobile environment requires systems and applications to adapt their behaviour in order to provide functionality with acceptable performance. The research described in this thesis is aimed at improving file system performance for mobile users by adapting file cache management to changes in available bandwidth. Issues in adaptive file cache management were investigated.

Although file cache management strategies have been developed to support mobile computing and to provide some level of performance, adapting file cache management to changes in the mobile environment has rarely been addressed[1]. This thesis has proposed strategies for adapting write back to bandwidth changes in order to reduce update reintegration time and strategies for adapting block caching to reduce time expansion and read service time.

This work shows that adapting file cache management to changes in the mobile environment has the potential for improving cache performance. Setting appropriate parameters at different bandwidth levels is critical to improving cache performance and keeping the negative impact of adaptation small. Some potential conflicts of different adaptive elements were examined. The results show that there is no significant conflict among the different adaptive elements studied.

7.3 Future Work

There are several possible directions for future research into adapting file cache management for mobile computing following from this thesis.

First, other elements in file cache management can be involved in adaptation. Prefetching, for example, brings soon-to-be used data to the client's cache and can be adaptive. The amount of data to be prefetched could be made adaptive to changes in the environment, such as bandwidth, cache capacity, the communication cost, etc. Prefetching can be performed at network idle time to avoid its impact on the read traffic. This involves competing with write back traffic which should also use network idle time.

Three traces from an academic environment were used to examine the performance of adaptive file cache management. A wider range of workloads from different environments and more recently collected traces could be used to examine the sensitivity of the results to workloads. The reason why these three traces were used in this study was to enable a clear comparison with results from the previous study.

These experiments were based on the assumption that the mobile client would

[1] In Mummert [29], adaptation of the update reintegration chunk size to bandwidth changes is a different approach.

have different bandwidth available from time to time. Since it is not clear what patterns of bandwidth changes a mobile user might encounter in practice, a randomly generated bandwidth sequence from five bandwidths was used in this investigation. More realistic bandwidth sequences with more bandwidth levels could be used to examine the performance implications more fully.

Finally, other changing factors such as the communication service cost and the client's local disk space may be concerns that file cache management could adapt to. Write back, prefetching, and block caching could all be made to adapt to changes in these factors. Adapting to changes in service cost could reduce the communication expenses for mobile users. Adapting to changes in disk space may avoid data overflow and make efficient use of the scarce local resource.

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] M. Baker, J. Hartman, M. Kupfer, K. Shirriff, and J. Ousterhout. Measurements of a distributed file system. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, pages 198–212, Pacific Grove, CA, October 1991.

[3] 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.

[4] V. Bharghavan. Challenges and solutions to adaptive computing and seamless mobility over heterogeneous wireless networks. *IEEE Personal Communications Magazine*, 4(2), March 1997.

[5] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a context-aware tourist guide: the GUIDE project. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, pages 20–31, Boston, MA, August 2000.

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

[7] M. Ebling, L. Mummert, and D. Steere. Overcoming the network bottleneck in mobile computing. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

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

[9] K. Froese. *File cache management for mobile computing.* M.Sc. thesis, Department of Computer Science, University of Saskatchewan, Saskatoon, SK, 1997.

[10] 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.

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

[12] R. Guy, J. Heidemann, W. Mak, Jr. T. Page, , G. Popek, and D. Rothmeir. Implementation of the Ficus replicated file system. In *Proceedings of the Summer USENIX Conference*, pages 63–72, Anaheim, CA, June 1990.

[13] J. Heidemann, T. Page, R. Guy, and G Popek. Primarily disconnected operation: Experience with Ficus. In *Proceedings of the Second Workshop on the Management of Replicated Data*, pages 2–5, Monterey, CA, November 1992.

[14] P. Honeyman and L. Huston. Communications and consistency in mobile file systems. *IEEE Personal Communications*, 6(2):44–48, December 1995.

[15] P. Honeyman, L. Huston, J. Rees, and D. Bachmann. The LITTLE WORK project. In *Proceedings of the Third Workshop on Workstation Operating Systems*, pages 11–14, Key Biscayne, FL, April 1992.

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

[17] 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.

[18] L. Huston and P. Honeyman. Peephole log optimization. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 25–32, Santa Cruz, CA, December 1994.

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

[20] T. Imielinski and B. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37(10):18–28, October 1994.

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

[22] R. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, January 1994.

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

[24] L. Kleinrock. Nomadic computing. In *Proceedings of International Conference on Information Network and Data Communication*, pages 223–233, Trondheim, Norway, June 1996.

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

[26] T. La Porta, K. Sabnani, and R. Gitlin. Challenges for nomadic computing: Mobility management and wireless communications. *Mobile Networks and Applications*, 1(1):3–16, 1996.

[27] Y. Lee, K. Leung, and M. Satyanarayanan. Operation-based update propagation in a mobile file system. In *Proceedings of the USENIX Annual Technical Conference*, pages 43–56, Monterey, CA, June 1999.

[28] H. Lei and D. Duchamp. An analytical approach to file prefetching. In *Proceedings of USENIX 1997 Annual Technical Conference*, pages 275–288, Anaheim, CA, January 1997.

[29] 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.

[30] A. Muthitacharoen, B. Chen, and D. Maziéres. A low-bandwidth network file system. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles*, pages 174–187, Chateau Lake Louise, October 2001.

[31] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134–154, January 1987.

[32] B. Neuman. The Prospero file system: A global file system based on the virtual system model. *Computing Systems*, 5(4):407–432, September 1992.

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

[34] B. Noble, M. Price, and M. Satyanarayanan. A programming interface for application-aware adaptation in mobile computing. In *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, pages 57–66, Ann Arbor, MI, April 1995.

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

[36] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating System Principles*, pages 276– 287, Saint Malo, France, October 1997.

[37] E. Pitoura and B. Bhargava. Building information systems for mobile environments. *Proceedings of the Third International Conference on Information and Knowledge Managment*, pages 371–378, Gaithesburg, MD, Nov. 1994.

[38] E. Pitoura and B. Bhargava. Dealing with mobility: Issues and research challenges. Technical report, Department of Computer Science, Purdue University, West Lafayette, IN, 1994.

[39] 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.

[40] J. Saldanha. A file system for mobile computing. Technical report, Department of Computer Science and Engineering, University of Notre Dame, South Bend, IA, 1993.

[41] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network filesystem. In *Proceedings of 1985 Summer UNIENIX Conference*, pages 119–130, Portland, OR, June 1985.

[42] M. Satyanarayanan. Mobile computing. *IEEE Computer*, 26(9):447–459, 1993.

[43] 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.

[44] M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1):26–33, 1996.

[45] 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.

[46] C. Tait. *A file system for mobile computing*. PhD thesis, Department of Computer Science, Columbia University, New York, NY, 1993.

[47] 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.

[48] 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 *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–183, Copper Mountain Resort, CO, December 1995.

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

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

[51] M. Weiser. Some computer science problems in ubiquitous computing. *Communications of the ACM*, 36(7):74–84, July 1993.