

UbiData: Ubiquitous Mobile File Service

Jinsuo Zhang, Abdelsalam (Sumi) Helal and Joachim Hammer

Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611-6120, USA
{jizhang, helal, jhammer}@cise.ufl.edu

ABSTRACT

One of the most challenging objectives of mobile data management is the ubiquitous, any time, anywhere access. This objective is very difficult to meet due to several network and mobile device limitations. Optimistic data replication is a generally agreed upon approach to alleviating the difficulty of data access in the adverse mobile environment. However, the two currently most popular models, both Client/Server and Peer-to-Peer models, do not adequately meet the ubiquity objectives. In our views, mobile data management should adequately support access to any data source, from any mobile device. It should also eliminate user involvement by automating data selection, hoarding, and synchronization, regardless of the mobile device chosen by the user. In this paper, we present UbiData: an application-transparent, double-middleware architecture that addresses these challenges. UbiData supports access and update to data from heterogeneous sources (e.g. files belonging to different file systems). It provides for the automatic and device-independent selection, hoarding, and synchronization of data. We present the UbiData architecture and system component, and evaluate the effectiveness of UbiData's automatic data selection and hoarding mechanisms.

Keywords

Disconnected operation, mobile data access, automated data hoarding, synchronization, and mobile data management.

1. INTRODUCTION

The demand for mobile computing gives rise to the need for data management assistance to mobile users at the system level. However, the mobile computing paradigm presents a dramatic discrepancy relative to currently mature, wired network computing. The various emerged mobile devices have the same kind of characteristics such as less computing power, energy, poor resources (storage, display, etc), and most importantly, network-related constraints [4]. In many situations, weak connection (characterized by lower bandwidth, higher error rate)

This research is funded by an NSF Grant number 0100770

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2003, Melbourne, Florida, USA.

© 2003 ACM 1-58113-624-2/03/03...\$5.00.

and even complete disconnection (either voluntarily or involuntarily) are often inevitably encountered. The weak and disconnected communication conditions create major challenges for mobile computing.

Data hoarding or replication is such a kind of adaptation [8] for mobile data management. Disconnected operation [16] further improves this by hoarding data in mobile device storage and providing a uniform data access method for both the connection and disconnection mode. Disconnected operation supports transparent data access switching between local and remote locations upon the availability of remote data.

However, there are some great challenges for employing this strategy to serve the mobile data management.

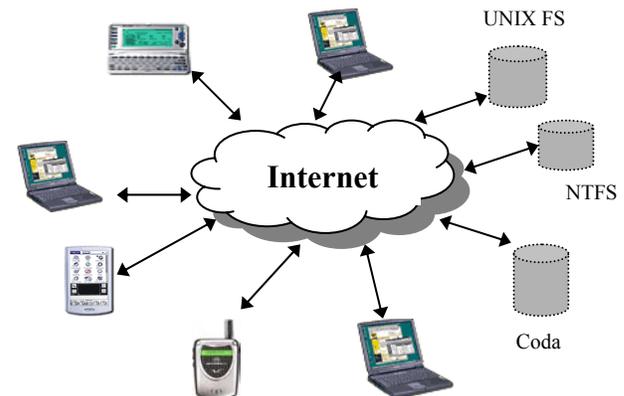


Figure 1: Internet Based Mobile Computing Environment

Figure 1 illustrates today's most commonly used computing paradigm for mobile devices. The heterogeneous mobile devices are usually connected through the Internet. Logically, this is a pure Peer-to-Peer model. All devices are equal in every respect. Any mobile device should have the ability to communicate directly with any other device if necessary. The Client/Server model is obviously not fit for this communication model.

Furthermore, the popular directory-based pure Peer-to-Peer file replication model cannot directly meet the need for a mobile user due to the special conditions in mobile environment. Several obstacles prevent its use.

- Data spectrum. The data that are of interest to the mobile user can be hosted in any place in any host file system of a mobile device, instead of one or more special volumes or directories.

- **Unavailability.** Data residing on mobile devices are often not accessible due to one or more reasons: frequent power off, frequent disconnection, weak connection and/or asymmetric communication.
- **Automatic Data Selection.** The manual selection of frequent data is not only tedious, but also error-prone. Sometimes it is even impossible [9].
- **Heterogeneity.** Heterogeneous mobile devices, in both software and hardware senses, are widely used in existing mobile devices.
- **Variable network connection.** Mobile devices often have a variable network connection in a different period. Most mobile devices usually employ wireless connection to the network in mobile state. Its low bandwidth is a bottleneck for communication.

To address the issues above, we proposed a double-middleware based architecture to tackle the problems. In this paper, we present the principle, design and evaluation of the suggested architecture. The paper is organized as follows. In section 2, we present the design principle for the UbiData architecture. The double middleware based UbiData architecture is introduced in section 3. In section 4, the system implementation is described. In section 5, we introduce the automatic data selection and hoarding system. The trace-based evaluation results are presented in section 6. In section 7, we survey related work. Finally, we summarize and suggest future work in section 8.

2 DESIGN PRINCIPLES AND GOAL

Considering the obstacles encountered in the current mobile environment, we proposed a double-middleware based architecture to alleviate these difficulties. We have these principles in mind for the design.

- **Highly Available Data Service.** Due to the various situations discussed in section 1, data are often not available in a mobile environment. To achieve 24/7 data availability, we introduce a new architecture, and rely on a specifically designed mobile data server to provide highly available data service for mobile devices.
- **Automatic Data Selection and Hoarding.** Automatic data selection and hoarding is a key to helping the mobile user conveniently and easily predict the future working set by observing the user’s working history. From the file access trace we have, we found other factors, except *recency* (used in LRU), also affect prediction accuracy. We use multiple parameters instead of relying solely on recency to predict the future working set.
- **Extensive Data Spectrum.** The mobile data spectrum should include any file in any supported operating system. This can be realized by extending the existing file system.
- **Publish/Import Model.** The selected data will be automatically published to the highly available data server where data can be imported easily by other mobile devices.
- **Application Transparency.** Since file service is such a basic service for the mobile user, it is helpful to provide a system level service for file replication, consistency maintenance, and network adaptation. Double middleware based architecture supports application transparent adaptation by extending the existing file system.

- **Heterogeneity of Mobile Devices.** Support of interoperation between heterogeneous mobile devices is important due to the varieties of mobile devices that co-exist now. The generic characteristics of XML [1] make it an ideal language for communication in such an environment.
- **Mobile Network Adaptation.** Minimizing network traffic is a common task for all mobile oriented systems. Reduction of traffic can be considered from two aspects. First, the potential interaction between mobile devices should be minimized. Second, for inevitable interaction, the minimum data content shipping should be targeted. In our system, we address both aspects of this issue.

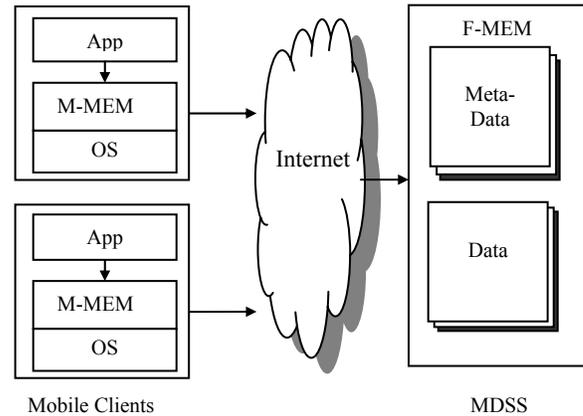


Figure 2: Double-Middleware Based UbiData Architecture

3. DOUBLE MIDDLEWARE-BASED ARCHITECTURE

The overall architecture is shown in Figure 2. One specifically designed server, mobile data service server (MDSS), is introduced to provide highly available data service for the any mobile client. Two middleware, M-MEM (Mobile-Mobile Environment Manager) and F-MEM (Fixed-Mobile Environment Manager) are used in our system to adapt to the mobile environment. M-MEM runs in every mobile host and F-MEM runs in the highly available mobile server. M-MEM and F-MEM cooperate to serve the mobile data selection, replication, consistency control and network adaptation. M-MEM and F-MEM are connected through the Internet. The architecture can be summarized in Table 1.

Table 1: Summary of UbiData Architecture

Dimension	Our approach
Mobile Data Spectrum	Any File in Supported OS
Data Selection	Automatic
Application Transparency	Transparent
Conservative/Optimistic	Optimistic
Client-Server/Peer-to-peer	Hybrid
Immediate/delayed propagation	Delayed
Push/Pull Model	Hybrid Push/Pull
Replication Granularity	Per-File
Replacement Policy	Hybrid Priority
Updated Data Shipping	Incremental Update

3.1 Data Naming

To share data one must first identify data. A unique identifier is necessary for every datum. We adopt the generic, self-explaining naming mechanism: Uniform Resource Identifiers (URI) [10] as the datum identifier.

3.2 Mobile Client Middleware Architecture

M-MEM is introduced in our system to extend the ability of the existing file system to support mobile environment. The extension includes automatic data selection, data hoarding and disconnected operation, consistency maintenance, heterogeneous communication support and network adaptation.

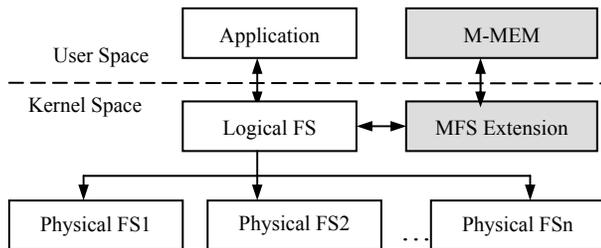


Figure 3: File System Extension Model

Figure 3 illustrates the extension to the mobile host. By extending the logical file system, the M-MEM can participate in file access and management. The major functions of M-MEM include:

1. Automatic data hoarding. Detailed description is illustrated in section 5.
2. Manage local replicas and meta-information such as the active file list, local replica information, etc. The cache manager does this task.
3. Data synchronization. M-MEM detects the file update in the mobile host by tracking the write(...) operation by the MFS extension. If the file is published or an imported replica, M-MEM will propagate the local update by notifying the mobile server. Another major function of M-MEM is to make sure the local replica is up-to-date by invalidating local stale files and retrieving fresh files when necessary.
4. Mobile network adaptation. To adapt the mobile network environment, we employ message queue optimization and incremental update propagation. Propagation may not occur immediately after an event occurs. Some messages may be cancelled or merged. Furthermore, we adopt an incremental update in our system. Thus, after every modification, only the minor differences are shipped. This requires some infrastructure support, such as file version, stale file management, etc.

3.3 Mobile Data Service Server Middleware Architecture

F-MEM is a middleware between M-MEMs in mobile devices. It is introduced in our system mainly for data availability. Besides, F-MEM also plays several other roles such as update conflict arbitrator, network adaptor, and data coordinator.

Figure 4 illustrates the mobile data service server architecture. The mobile data service server provides highly available data and meta-information service to mobile devices. It is stateless.

The mobile server uses two databases: a data server and a meta-data server to store data and meta-data respectively.

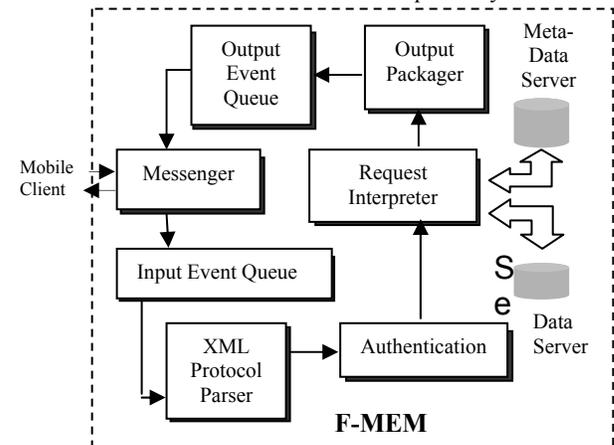


Figure 4: The Organization of Mobile Data Service Server

The meta-data server is a native XML database: Xindice [3] and stores all information about the mobile data and the user. XML document types currently used in Meta-Data Server include both mobile profiles and replica descriptors.

3.4 Communication

The communication between the mobile client and the server is through a set of XML based protocol. The information exchanged is encoded into an XML document and shipped using an asynchronous, reliable message system.

The communication is mobile adapted. First, all the exchanged message is optimized. Redundancy is dramatically reduced. Second, incremental update/hoarding [6, 7] instead of whole data shipping is used. We use Xdelta [13] as difference computing tool in our implementation.

3.5 Dataflow in UbiData Architecture

M-MEM monitors the behavior of mobile user and publishes the active data to mobile data service server, which always resides in a highly available network. The published data can be hoarded by any other mobile device for use in disconnection or weak connection mode. M-MEMs in every mobile device cooperate with F-MEM to maintain the consistency of multiple replicas.

Data are retired by mobile file service when it is aged out from the active file list. Retired file is still regular file in its local file system but not controlled by system for replication and consistence maintenance.

4. SYSTEM IMPLEMENTATION

The UbiData project is implemented in both Linux and Windows platforms using C++/C. A Windows CE port is underway. Except for operating system extensions, M-MEM is executed as an application in user space. F-MEM also executes completely in user space.

The core algorithm is implemented using C++/C and Standard Template Library (STL) and can be compiled in both platforms.

The interface for Windows is implemented using Microsoft Foundation Class (MFC) library. The interface for Linux is developed under GNOME [5] environment of X-Window. The core algorithm is combined with different interface modules to produce executable code for given platform.

The XML DOM parser currently used is libxml [12]. The meta-data database used by the mobile data service server is a native XML database managed by Xindice [3]. The data managed by the mobile data service server is directly mapped and stored into the local file system as indicated by the URI meta-data of data file.

The data exchange between M-MEM and F-MEM is through a suit of XML based protocols. The encoded XML document is shipped to its communication counterpart using either HTTP or SMTP/POP3 protocol. The data content shipping is delta-based. The diff engine we are using is Xdelta [13].

5. AUTOMATIC HOARDING IN UBIDATA

Figure 5 illustrates the workflow of the automation used in our system. The automatic file selection and hoarding mechanism works as follows:

- By hooking into the operating system kernel, we can collect user file access events such as file open, close, write, rename, etc. The events are saved first in the buffer of a pseudo device driver.
- These file access events then are read by a user space application called Collector.
- Collector pipes the events to Filter, where the file access events are filtered. Non-interested actions are discarded immediately. Interesting events are piped into an analyzer.
- Analyzer then computes hybrid priority for every interesting file. The list of files with highest priority during a period of time constitutes the candidates for future hoarding.

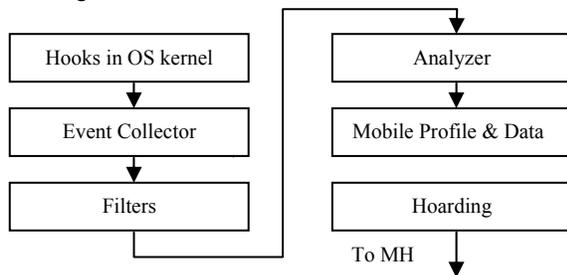


Figure 5: Automatic File Hoarding Workflow

- The list and interested files are published into the mobile server. The list is merged with the existing hoarding list in the user's mobile profile and becomes the final hoarding list for the given user. The published data are stored in the data store of the mobile server.
- Periodically, or before an anticipated disconnection, the hoarding system uses the hoarding list to hoard data – incrementally- to the mobile device. The hoarding procedure will be adapted based on the type of the mobile device and the network connection it has with the server.

5.1 Filtering Mechanism in UbiData

It is expected that a user's true daily working set is small since it is only possible to do a limited things in a limited amount of time. However, spying directly upon on operating system file access will produce an exponential amount of file access events. Initial experiments reveal that lots of file operation is not of an interest to us for mobile data management purpose. These uninteresting file accesses not only create a burden for the analyzer, but also interfere with the accurate prediction of the hoarding list. To eliminate these negative effects, we introduced various *exclusive filters*. Using filters, we can minimize the computing of the analyzer, improve the file access prediction accuracy, and reduce hoarding size to an acceptable size. We also introduced *inclusive filters* for the user's convenience.

5.1.1 Location Based Filter

One important characteristic of some files that needn't to be hoarded is that they are usually located under and below a certain well-known directory, like /etc, /var, etc.

Figure 6 illustrates a trace-driven (more information about trace will be provided in sections 6) analysis we have conducted to study the location distribution of system and software file in a computer. From the figure we can see that a large percentage of file access is to system related files. On average, about 56% of accessed files are system related, or 69% dynamic file accesses are system related. These files can be effectively excluded using location-based filter.

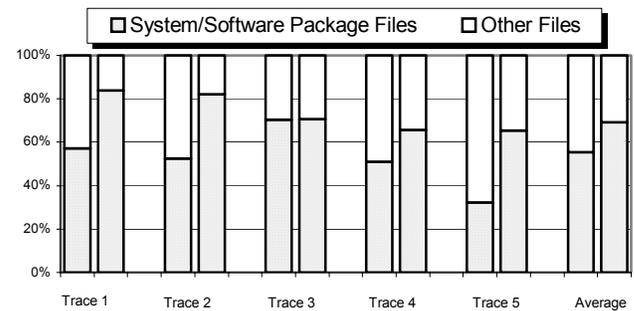


Figure 6: Location Distributions For File Access. For every trace, the left bar stands for the per-file based percentage, the right bar stands for the per-access based percentage

5.1.2 Program based Filter

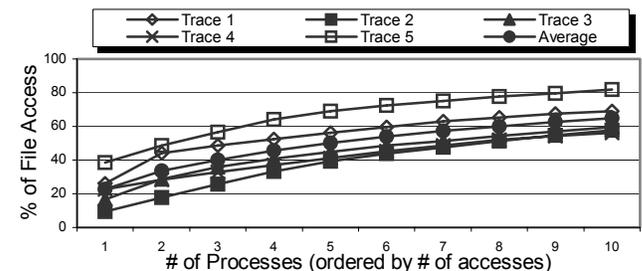


Figure 7: The Accumulative Percentage of File Accesses By Most Active Program

Figure 7 illustrates another piece of analysis we did using the same trace. It shows the relationship between file accesses and the most active programs. For every trace, ten most active programs are picked out and ordered after simulation. The accumulative percentage of accesses relative to number of programs is plotted. From the figure, we can know that most file accesses are only produced by a small number of programs. For example, on average, about 60% of file accesses are generated by only eight programs. Program based filter is specifically designed to filter out file accesses generated by this kind of program.

Besides the predefined filtering for those “famous” programs, the M-MEM also tracks the program and provides the utility to help the user to filter out misbehaving programs in his/her own computer.

5.1.3 Additional Filters

We have defined and implemented additional filters that can further eliminate noisy file access and help improve predicting accurate working set. These include:

- Extension name based filter
- Time based filter
- Derived based filter
- File type based filter
- Meta-info based filter

5.2 Determining the Hoarding Set

File reference modeling (or determining the hoarding set) has been extensively studied in file system research and design. However, our research, presented here, has completely different focus. Existing research emphasizes on the short-term behavior and attempt to improve the system performance. Our automatic selection and hoarding subsystem, focuses instead on relative long-term behavior and tries to predict future file access for hoarding purposes. This is achieved through continuous analysis of the past file reference pattern.

File accesses events, after being filtered by the various filters, are fed into the analyzer. Here, the analyzer calculates *hybrid priority* for every interesting file. A list of files with highest hybrid priority constitutes the hoarding list. The hoarding list is used to hoard potential data periodically, or prior to anticipated disconnection.

5.2.1 Hybrid Priority based Algorithm

The future probability of a file reference may heavily depend on the access history due to the locality property. History can be represented by many parameters, such as recency (basis of LRU), frequency (basis of LFU), among other parameters. The effects of these parameters to locality have been extensively studied in hardware cache and software buffer management research. However, only LRU is studied in automatic data selection mechanisms [9, 11]. In our research, we studied the effect of various parameters on the file access repetition and proposed a new metric for potential reference of files in the future.

The advantage of LRU is in its simplicity and speed. LRU and its variants as a file replacement policy are carefully studied in

[9]. The study shows that LRU basically works well but can not catch the references that had occurred far back in the past.

We introduced a new metric, called *hybrid priority*, to represent the possibility that a file will be accessed in the near future. It is defined as:

$$F(\langle \text{Recency} \rangle, \langle \text{Access Frequency} \rangle, \langle \text{Active Period} \rangle)$$

For every file access event that is not filtered out, the analyzer will compute its hybrid priority on the fly and insert it into an ordered list. When the space exceeds the upper limit, the file with the least hybrid priority will be removed from the list. In hoarding stage, the bigger the hybrid priority, the higher privilege the file should be given during hoarding.

5.2.2 Factors Affecting Caching Policy

The following three factors affect our hybrid priority caching policy.

- **Recency:** the time between now and the most recent access. This is the basis of LRU. Because LRU has been extensively studied and proved effective, it is not presented here.

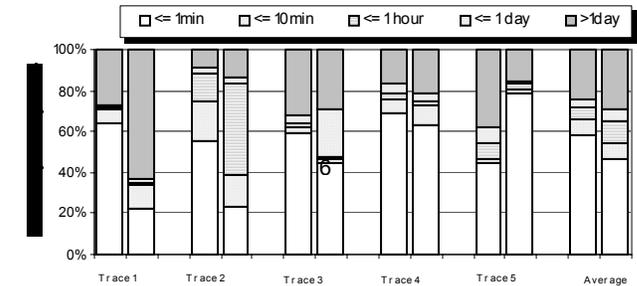


Figure 8: The File Active Period Distribution. For every trace, the left bar stands for information about all files; the right bar shows only the information about interesting files, i.e., excluding files eliminated by filters

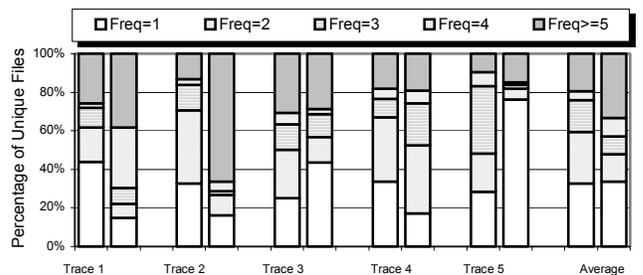


Figure 9: File Access Frequency Distribution. For every trace, the left bar stands for the information about all files, the right bar shows only the information about interesting files, i.e., excluding files eliminated by filters

- **Active Period:** which is the time period between recorded first time access and last time access. Some files are accessed only for a short period and never re-referenced again. Some others, however, will be active for a long time until the owner finishes the work that file belongs to. In Figure 8, we studied the active period of files from the five

traces. From the figure, we can see that many files are active only for a very short period of time. On average, about 58% overall files, or 47% interesting files are active only for a minute or less, only about 25% overall files, or 29% interesting files are found to stay active for more than one day. Obviously files with very short active periods should be given less priority for the purpose of hoarding.

- **Access Frequency:** Access frequency is defined as the number of accesses to a particular file. Our trace-based simulation analysis shows that most files are accessed only once and never accessed again. Figure 9 below shows the distribution of file access frequency. We can see that on average about 33% of overall files, or 34% of interesting files are only accessed once in the whole traces.

5.2.3 Definition of Functions in Hybrid Priority based Algorithm

Let t , f , and a be the recency, frequency and active period of a particular file, its hybrid priority, $\mathcal{F}(t, f, a)$, is defined as:

$$\mathcal{F}(t, f, a) = \alpha_1 \mathcal{F}_1(t) + \alpha_2 \mathcal{F}_2(f) + \alpha_3 \mathcal{F}_3(a)$$

Where α_1, α_2 , and α_3 are the weight of recency, frequency, and active period, and $\mathcal{F}_1, \mathcal{F}_2$, and \mathcal{F}_3 are scaling function.

$\mathcal{F}_1(t)$ is defined as: $\mathcal{F}_1(t) = H_0 - \langle \text{aging parameter} \rangle * \langle \text{current time} - \text{last access time} \rangle$

$\mathcal{F}_2(f)$ is defined as: $\mathcal{F}_2(f) = FA * \text{frequency}$

$\mathcal{F}_3(a)$ is defined as: $\mathcal{F}_3(a) = AA * \langle \text{active period} \rangle > H_0 ?$
 $H_0: AA * \langle \text{active period} \rangle$

H_0 is the starting height, FA is frequency accelerator, AA is active period accelerator. In all definitions above, all time units represent number of files opened, not clock time.

5.2.4 Properties of Hybrid Priority based Algorithm

Property 1: If $\mathcal{F}(t, f_1, a_1) > \mathcal{F}(t, f_2, a_2)$, then

$$\text{for } \forall \tau > t, \mathcal{F}(\tau, f_1, a_1) > \mathcal{F}(\tau, f_2, a_2)$$

Property 1 means that if two files are ordered by hybrid priority and no file has been touched since then, then the order is always kept.

Property 2: The file list ordered by hybrid priority need not be updated between any two file access intervals.

Property 3: If one file in file list is accessed, the order after this file in file list still holds.

Property 4: If one file in file list is accessed, the file list subset before this file can be re-ordered in $O(\log(n))$.

Proofs of aforementioned properties are omitted here for space limitations.

5.2.5 On-the-Fly Analysis

One characteristic of our analyzer is its on-the-fly analysis. The file access events are collected from the operating system kernel and analyzed on line by analyzer. This computing is possible due to the employment of various effective filters. The on-the-fly characteristic of the analyzer makes the system practical for daily use.

6. EVALUATION OF AUTOMATIC HOARDING

6.1 Simulation Methodology

To validate the automatic hoarding scheme, trace-driven based simulation is used. The traces are collected by UCLA SEER project [9]. In the simulation, instead of using file access events from OS extension, the M-MEM collects events from the trace interpreter. The MFS extension is temporarily disabled.

The trace collectively describes five different traces, each describing access behavior by different users. Summary information of these traces that we used in our experiments is summarized below. More details can be obtained from [9].

Table 2. the information about trace used in simulation

	Trace Collecting Period	Footprint
Trace 1	63 days	8k
Trace 2	62 days	8k
Trace 3	232 days	37k
Trace 4	132 days	100k
Trace 5	61 days	4k

6.2 Effectiveness of Filters

In Figure 10, the effectiveness of filters is plotted. For every trace, two bars are drawn. The left bar is for distinct files (static counter), and the right bar is for file accesses (dynamic counter). From the figure, we can see that on average about 87% of files, or 84% of file accesses are filtered out. This effective filtering mechanism provides good working conditions for analyzer.

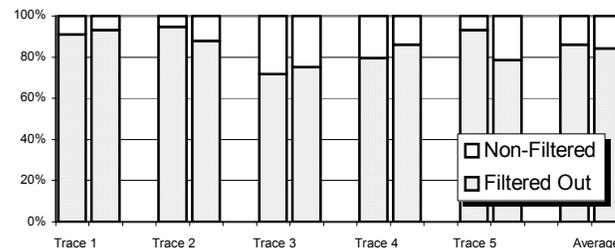


Figure 10. Effectiveness of Filters

6.3 Mobile Data Working Set Study

The purpose of the filter is to eliminate the noisy file access events and get the true user working set. In this section, the working set for every trace is examined.

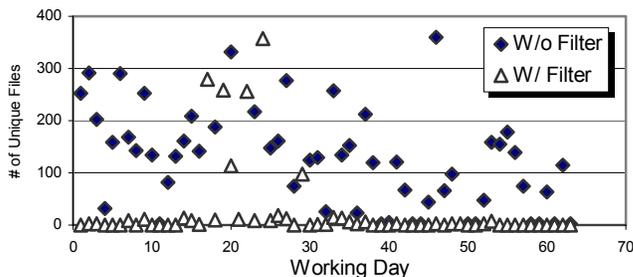


Figure 11(a). Daily Working Set After Filtering for Trace 1

Figure 11 (a) is the daily working set for trace 1. The X-Axis is the calendar day when the trace is collected. Y-Axis is the number of distinct files. The empty triangle stands for the working set with filtering, whereas solid box stands for working set without filtering. Some daily working set sizes without filtering are so big, in order to make the figure clear more readable, the scale is reduced to contain most data, so a few dramatically large numbers exceeding maximum Y cannot be shown in the figure.

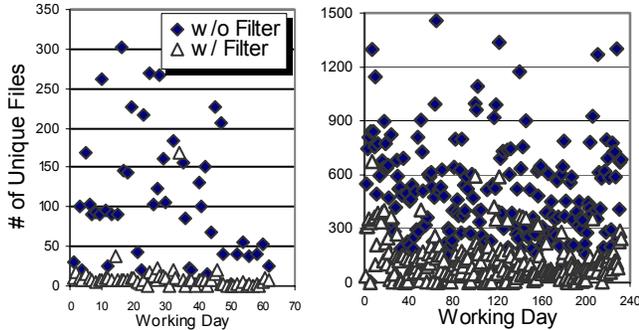


Figure 11(b). Trace 2

Figure 11(c). Trace 3

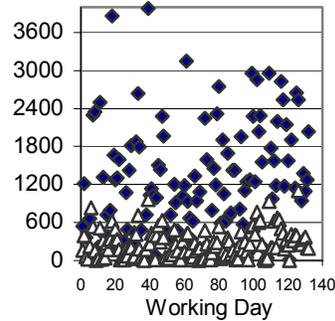


Figure 11(d). Trace 4

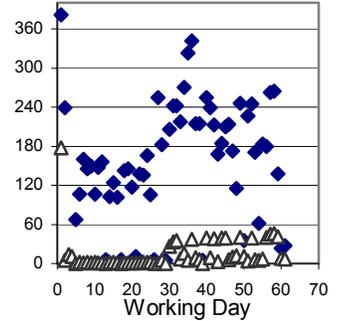


Figure 11(e). Trace 5

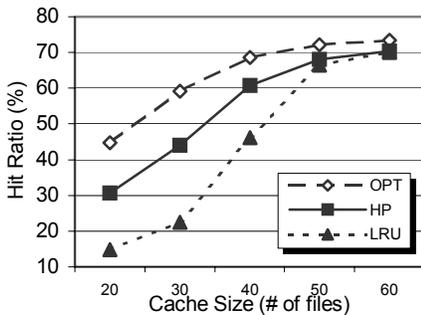


Figure 12(a). Hit Ratio of Trace 1

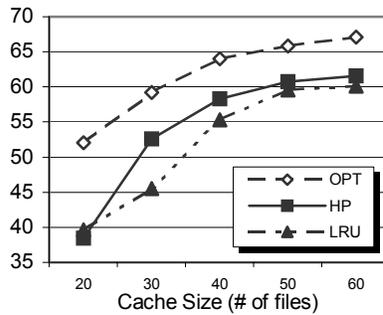


Figure 12(b). Hit Ratio of Trace 3

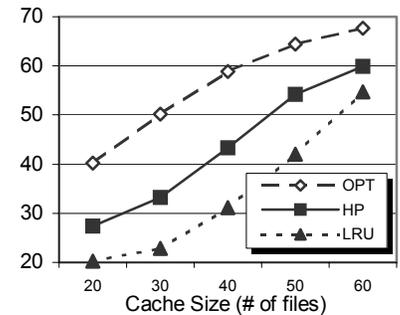


Figure 12(c). Hit Ratio of Trace 4

6.4 Evaluation for File Reference Modeling

The hit ratio is studied and compared with other policies. In this experiment, three cache replacement schemes are tested and compared: Optimal scheme (OPT), Hybrid Priority Algorithm (HP), and LRU (Least Recently Used).

Table 3. Parameter Used In Hybrid Priority Algorithm

Parameter	
α_1	0.5
α_2	0.3
α_3	0.2
H_0	1000
Aging parameter	Cache size/128
FA	cache size/4
AA	1

When the cache size exceeds upper limit, the three policies use different metric to evict cached item. LRU always evicts the item with biggest age. HP policy always evicts the item with least hybrid priority. OPT policy can “see” future file access pattern and always evicts the item with longest next access

Figures 11(b-d) are the working set simulation result for trace 2 to trace 5 respectively.

From the simulation, we can see that filters can effectively exclude the noisy file access events and reduce the working set to several times smaller than it would be without filtering. Despite its effectiveness, the filtering mechanism is *conservative*, which means it leaves the files it is not sure about their usefulness to analyzer to decide if replicating upon the file’s hybrid priority parameters.

interval. OPT is usually used as an upper bound compared with practical solution.

The parameters used in the simulation for hybrid priority algorithm is summarized in table 3. Varying the parameters also shows the same trend in the experiment.

For every trace, seven calendar days are simulated. The results of trace 1,3 and 4 are plotted in Figure 12. The results for trace 2 and 5 are not plotted here. Trace 5 has so small working set that any scheme can capture it very well. Trace 2 is a typical sequential flooding access: every time about 110 files are accessed sequentially, then re-accessed from the beginning. No small cache can capture this pattern effectively whereas bigger cache can easily capture it.

From the figure, we can see that the HP scheme consistently performs better than LRU. This is consistent with the results shown in Figures 8 and 9.

7. RELATED WORK

Coda [16] supports mobile environment by providing infrastructures such as disconnected operation, file hoarding,

weak connection adaptation, among others. Coda architecture is based on the Client/Server model. Coda tracks the local file modification and ships the CML (Client Modification Log) to save bandwidth [14]. Intermezzo file system [2] borrows the similar idea in its implementation.

The Roma [17] personal meta-data service provides a Peer-to-Peer based file information service. Roma uses a centralized, highly available server to store meta-information of interesting data so they can be easily located. Physical data remain distributed in the various origin machines. Roam [15] is a Peer-to-Peer model based directory replication system.

The SEER system [9] correlates the user-accessed files into cluster/project by calculating the semantic distance between referenced files, rather than considering them as discrete files. Cluster/Project is hoarded as a whole unit to mobile devices by the hoarding system. Lei and Duchamp in [11] proposed a tree-based algorithm to track and analyze program executions.

SyncML [18] is a specification designed for mobile data replication and synchronization between heterogeneous mobile devices.

Our previous work [6, 7] tested the incremental hoarding and re-integration based on the Coda system. Incremental update is also used in Rsync [19] and XDFS [13].

8. SUMMARY AND FUTURE WORK

In this paper we described the UbiData architecture, which provides for ubiquitous file services in mobile environment. UbiData supports device-independent access to data belonging to heterogeneous sources (e.g., two different file systems). We have presented the Ubidata system and its automated file selection and hoarding policies and mechanisms. We have presented an experimental study that evaluated the effectiveness of our filtering mechanisms and hoarding schemes (hybrid priority). We have also presented “true” working set studies to evaluate the overall mobile access performance. Trace based simulation results show that more hit ratio can be achieved from hybrid priority based algorithm.

The architecture introduced here has already been implemented in both Linux and Windows platform. A port of the mobile client part (M-MEM) to the Windows CE platform is also nearing completion. More functions, such as external data support, mobile transaction, and additional synchronization methods are under investigation.

9. ACKNOWLEDGEMENT

The UbiData project is funded by NSF grant. The authors also thank the anonymous reviewers for their valuable comments.

REFERENCES

- [1] Bray, T., Paoli, J., and Sperberg-McQueen, C. M., Extensible Markup Language (XML), W3C Proposed Recommendation. Dec 1997. <http://www.w3.org/TR/PR-xml-971208>.
- [2] Braam, P. J., Callahan, M., and Schwan, P., The InterMezzo File System, at <http://www.intermezzo.org/docs/perintermezzo.pdf>, 1999.
- [3] dbXML group, <http://www.dbxml.org/>.
- [4] Forman, G.H., and Zahorjan, J., The Challenges of Mobile Computing. IEEE Computer, 27(6), Apr 1994.
- [5] Gnome website: <http://www.gnome.org/>
- [6] Helal, A., Hammer, J., Zhang, J., Khushraj, A., A Three-tier Architecture for Ubiquitous Data Access. Proceedings of the First ACS/IEEE International Conference on Computer Systems and Applications, Beirut, Lebanon, June 2001.
- [7] Khushraj, A., Helal, A., and Zhang, J., Incremental Hoarding and Reintegration in Mobile Environments. Proceedings of the IEEE/IPSJ International Symposium on Applications and the Internet (SAINT), Feb 2002 Nara, Japan.
- [8] Katz, R., Adaptation and mobility in wireless information systems. IEEE Personal Communications, 1994
- [9] Kuenning, G. H., Seer: Predictive File Hoarding for Disconnected Mobile Operation. Technical Report UCLA-CSD-970015, 1997.
- [10] Lee, T. B., Fielding, R., Masinter, L., Uniform Resource Identifiers (URI): Generic Syntax. Network Working Group, Request for Comments: 2396, <http://www.ietf.org/>, Aug 1998.
- [11] Lei, H., and Duchamp, D., An Analytical Approach to File Prefetching, USENIX Conference Proceedings, Anaheim, California, Jan. 1997.
- [12] Libxml website: <http://www.libxml.org/>
- [13] MacDonald, J. P., File System Support for Delta Compression. Master thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 2000.
- [14] Mummert, L., Exploiting Weak Connectivity in a Distributed File System. PhD thesis, School of Computer Science, Carnegie Mellon University.
- [15] Ratner, D. H., Roam: A Scalable Replication System for Mobile and Distributed Computing. PhD thesis, University of California, Los Angeles, Los Angeles, CA, Jan 1998.
- [16] Satyanarayanan, M., Coda: A Highly Available File System for a Distributed Workstation Environment. Proceedings of the Second IEEE Workshop on Workstation Operating Systems, Sep. 1989, Pacific Grove, CA
- [17] Swierk, E., Kiciman, E., Laviano, V., and Baker, M., The Roma Personal Metadata Service. Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications, Monterey, California, December 2000.
- [18] SyncML Consortium, SyncML Specification, version 1.0.1, <http://www.syncml.org/download.html>.
- [19] Tridgell, A., and Mackerras, P., The rsync Algorithm. Technical Report TR-CS-96-05, Australian National University.