User-oriented Mobile Filesystem Middleware for Mobile Cloud Systems

Dongbo Liu, Peng Xiao
Department of Computer and Communication, Hunan Institute of Engineering, Xiangtan, 411104, China
Email: liudongbo74@126.com

***Abstract*—More and more mobile equipment use independent filesystem to expand their capacity, however the mobile storage systems often perform undesirable when plenty of users participate in these systems. In this paper, we present a novel mobile storage framework, which is aiming to provide convenient mechanism for mobile equipments. In the proposed storage framework, several optimization mechanisms are incorporated for improving the performance of massive data synchronization and backup. The results show that it is more flexible and reliable than other existing systems. In addition, its exhibits highly robustness when the system is in presence of a large number of concurrent users.**

***Index Terms*—cloud, distributed storage, mobile computing, reliability, data cache**

I. INTRODUCTION

Mobile devices (i.e., SmartPhone, iPad, Laptop) have become more and more important in our daily lives. Many people have spent most of their time on these mobile devices for working or entertainment, such as sharing working or personal data, playing games, collecting favorite information and watching movies. However, these devices always has only limited storage space and need multiple steps for simple data-synchronization [1,2]. The typical approach to dealing with this problem is backup the user's data in independent systems [3,4,5]. However, many of these systems can only provide some basic data-backup functionality, and users have to manually manage their data. When users have multiple mobile devices, they have deal with the data-synchronization work on each device one by one.

Recently, Cloud platforms are emerged as promising paradigms to providing the capability of parallel data processing by making large number of servers available to various kinds of users who lack such facilities at hands [6,7]. With the popular of "Software as a Service" (SaaS) computing architecture, large-scale storage pools have been built and deployed on more and more Cloud-based data centers by using cheaper disk components. Traditionally, these storage systems are designed for scientific research or high-performance computing. Recent years, many people have realized that Cloud-based storage system is a very cost-effective for commercial applications. Unfortunately, the performance of cloud-based storage systems is facing a great challenge, that is, when more and more devices and users participate, stability and reliability of these systems will be degraded significantly [5,8,9,10,11,12]. Many existing studies have shown that the performance bottlenecks of mobile storage systems are:

- Unpredictable user's requests often result in performance fluctuation especially for those non-dedicated systems [2,5].
- Different classes of data often require different performance optimization, which is difficult to be satisfied by the general storage systems [9,10,11].
- In mobile computing environment, storage cache is very expensive because of the limitation of wireless bandwidth [8,10,12].

In this work, we design and implement a light-weighted storage middleware, namely user-oriented Mobile File System (uMFS), which is aiming to provide massive-data storage service for mobile end-users with high performance, friendly user-interface and extended scalability. uMFS is a standalone middleware that can be easily deployed in most existing cloud systems. It fully takes advantage the underlying cloud infrastructure as well the legacy storage systems. More specifically, it uses virtualization technology to implementation an uniform storage interface for upper-lever users; on the other hand, it enable off-line bulk-data synchronization directly with existing storage system (i.e., GFS, NFS and gFTP) so as to provide enhanced-performance and save expensive wireless bandwidth. Unlike existing storage middleware, uMFS is incorporated with file navigation mechanism for attribute-based file retrieve or semantic-oriented data query; it also enable users to verify their data integrity.

The rest of this paper is organized as follows. Section II presents the related work. In section III, we briefly describe the framework of the proposed system; In Section IV, implementation details are presented. In section V, experiments are conducted to investigate the effectiveness of the proposed system. Finally, Section VI concludes the paper with a brief discussion of future work.

II. RELATED WORK

Many big cloud providers have provided their Cloud-based storage solutions. For instance, Amazon's *Simple Storage Service* (S3) is a popular commercial Cloud storage system, which is designed to be simple and open standard for constructing certain applications [12]; Microsoft's *Live Mesh* is aiming at the efficient data backup across different network domains [13]; Mozy [14]

doi:10.4304/jcp.8.9.2209-2216

and Symantec's Protection Network (SPN) [15] is mainy for the data backups. Other commercial distributed storage systems include Cumulus [16], Brackup [17] and Jungle Disk [18]. In general, these systems are aiming to providing reliable and extensible data backup service for common users.

Currently, many end-user storage systems (i.e. Twister [19], Spark [20], HaLoop [21] and Piccolo [22]) have be developed for data storage in mobile computing environments. Most of them applies MapReduce programming paradigm for iterative data-processing, which can be categorized as distributed-memory storage systems and the communication between sub-tasks are often based message-passing mechanism. This technique is effective to speedup the total data-processing efficiency, however it also tends to produce plenty of redundant communication messages when synchronizing a great number of sub-task nodes [7,9,11].

To overcome the multi-point synchronization problem, several light-weight storage middleware are developed to easy the work of end-users. For example, Dropbox [23] is a cloud-based storage middleware, which enables users to store and backup their data online. The most characteristic of Dropbox is its across-platform file-folder synchronization service, which enable those users who have several accounts on different sites to synchronize their data between these accounts. Syncplicity [24] is GFS-based storage middleware, by which users can backup their data on the Google site onto any other compatible distributed storage systems. MobileMe [25] is another online service services offered by Apple Incorporation. Currently, it allows users to manage their data, mail, calendar, address book, and other personal data in an integrated toolkit. The above middleware are developed by famous cloud-providers for providing enhanced-service for their own customers.

To efficiently caching the data in mobile computing systems, *Distributed Data Diffusion* technology has been widely applied in many storage systems [26]. It allows data owners store critical data items in the local disks and querying other items through wireless network. So, it mainly used in P2P environments, where massive common user's data files can be widely shared as much as possible, while critical files can also be protected by local security mechanisms provided by mobile devices and software. In [27], Kim et al. proposed a P2P-based mobile storage framework called ELIAS, which enables user applications to transparently store/retrieve data from a location without concerning with low-level detailed implementation. The most significantly advantages of ELIAS is that it can be used as a common middleware for most of existing storage systems. In [28], Wang et al. proposed a flexible distributed storage integrity auditing mechanism, which uses homomorphic token to encode data. The proposed design allows users to audit the cloud storage with very lightweight communication and computation cost. In addition, it also improved the data location efficiency by a paralleling query mechanism.

III. THE FRAMEWORK AND DESCRIPTIONS

The overview of the uMFS framework is shown in Fig. 1, which depicts the key components of uMFS and interactions between uMFS and the underlying cloud infrastructure. As shown in the framework, there are six key components in the uMFS, including user's programming API and toolkits, data mapping service, data navigator, IaaS adaptor, key and authentication manager, and performance manager. In this section, we mainly describe the functionalities of these components, and the implementation of them is presented in Section 4.

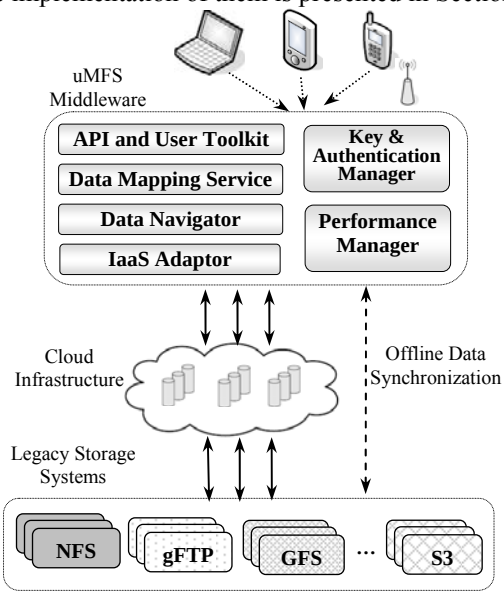


Fig. 1. Framework Overview of uMFS

- *API & User Toolkit*: This component includes a set of well-defined user's API, by which developers can extend uMFS with more components. Currently, we simply expose the underlying filesystem operation interface through this part. In this way, we can separate the underlying API with the user's API, which is of significant importance for the future refining of our system. User toolkit main includes some UI component of the other components as well as their combinations, such as semantic-based data navigator, offline data synchronization, and personal folder management.
- *Data Mapping Service*: Due to the limitation of mobile devices, many of filesystem operations should be migrated from mobile devices to the middleware. To do this, we designed this component which is responsible for all the operations between abstract data views and concrete data achieve. For example, when user issues a backup operation, it will translate this command into a set of calling-chain including finding available storage nodes, evaluating the feasibility of the current operation, checking the redundant data items and etc. By the help of this component, many power-consuming works will be conducted in uMFS instead of the mobile devices. In the user toolkits, almost all users' utilities use this component to finish their work. More details of it will be presented in the implementation section.

- *Data Navigator*: As mentioned above, customized data navigation service is designed for satisfying the requirements of certain mobile communities. However, there are still many users used to the conventional hierarchical directory tree. So, we developed a hierarchical attribute-based navigation component to meet various requirements of different users. The raw data maintained in the metadata server including the mappings scheme, and the mappings from chunks to chunk servers and the attribute based namespaces indicating the structure of a set of files.
- *Performance Manager*: In order to evaluation the performance of uMFS, we implement this component, which can logs all the requests from mobile users and analyzes the workload of other components, also it keeps track of the underlying storage nodes. For instance, when a new storage node or device is added to our system, the performance manager logs this event it with a time-stamp. If a storage node is deleted from the system, it also logs the event. For each storage node or device, it maintains a real-time performance table for it. Currently, we mainly use this component to evaluate the performance of the prototype implementation. Lately, we are planning to implement it for optimizing the performance of some advanced storage strategies.
- *Key & Authentication Manager*: As many users have several accounts on different storage vendors, it becomes a heavy burden for user to management so many account. More important, uMFS is designed for providing a uniform storage framework to end-users, so, it is critical to automatically manage multiple user's account so as to realize this goal. In the prototype of uMFS, it allows users to define the validity durations of the each account. When data accessing operation is conducted, other components may calling this component for the first time until the validity of the current account is expired.
- *IaaS Adaptor*: It is implemented as a interface adaptor, which allows the uMFS can easily support different cloud platform implementations given there are some any cloud-based infrastructure with more and more are emerging. In fact, it is consists of a set of forward interfaces which can be implemented in different ways. So, the implementation of the upper-level components need not be modified when we add supporting for other cloud infrastructures.

As shown in the uMFS framework, it is clear that uMFS is an extensible framework, which can be compatible with different underlying resources as well as the different mobile devices.

IV. Implementation of the System

A. Data Mapping Services

Based on the characteristic of wireless network and the mobile devices, we design the Data Mapping Service (DMS) in order to improve the system performance and user's requirements. As mentioned above, DMS is mainly responsible for data-accessing operations between abstract data views and concrete data achieve. In DMS, three optimization mechanisms are implemented to achieve the designing goal, which are shown in the following.

(a) Raw Data Layout Optimization

For mobile devices, I/O operation is time and power consuming. Many existing studies have shown that optimized data layout in embedded system is the prerequisite for better performance. Normally, data layout optimization can be automated managed by OS system or manually managed by user's application. In some system, it also can be implemented in compiler-level.

In mobile computing environment, data-intensive applications often spent a long time duration in a tight loop nests. It is well known that data layouts of multidimensional arrays are fixed at a specific form. However, in some applications fixing the data layout to the same form for all arrays can hurt the performance severely. It is clear that instead of fixing layouts of these arrays, it would be suitable to define one of the layouts to row-major and the other to column-major. By this approach, we only need to spend consecutive time for data item location.

(b) Aggregation of I/O Operations

Aggregating I/O operation is to collect as much as possible user's I/O requests and send them to underlying physical storage devices in batch manner. This optimization is very effective when the logical layout of data is different from their corresponding physical layout. In typical mobile storage platforms, we found that many users tend to issue multiple I/O operations on a single data item. For example, a movie clip might be searched by its attributes (i.e., name, class, produced time and etc.), and then be previewed, and then be stored to favorite folder by its attributes. If all this I/O operations are executed in sequence, the response time of the storage system will be intolerable. By aggregating multiple I/O operations, uMFS can automatically optimize the low-level data cache. In DMS, we implement I/O aggregation in the raw data achieve level, instead of application-level which can be seen in other systems. It is because that we wish uMFS can be directly interact with low-lever filesystem so as to improve the performance of offline data synchronization as shown in Fig. 1.

When performing I/O concurrently for several user's requests, we optimize the I/O performance at the expense of a few extra costs. Although this approach slightly increases the communication time of the program, it generally minimizes the number of I/O calls which in turn reduces the execution time significantly as I/O. As we can see that uMFS can combine multiple I/O operations into a group-based request. Additionally, the total I/O workload can be partitioned among the underlying storage nodes dynamically.

(c) Virtual Data Processing

In cloud environments, almost all the applications are executed on virtual machines. Currently, it is popular to

deploy traditionally network filesystem through virtualization technology, i.e. Amazon S3. Therefore, how to interact with these virtualization platforms is critical important for the uMFS. We define a set of virtual data-accessing interfaces (vDAI) in the DMS component. When an application accesses a data item, vDAI determines which file system to access with the path information. If the file exists in one of the underlying storage achieve, uMFS will send the request to this interface layer by the vDAI. Therefore, vDAI interacts with underlying I/O requests from applications, and manages these data coming from anywhere.

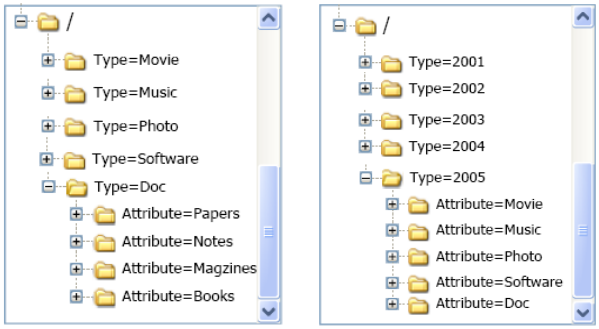
As we often use standard interfaces to access the file system, we may run many interesting and useful applications on this system and transparently access the remote backend services without any modification. To make sure that the introduction of the virtual data-accessing interfaces will not affect the correctness of the system, most of these interfaces have the same definition as traditional filesystem interface, i.e. read, write, seek, open and close.

B. Attribute-base Data Navigator

Unlike the hierarchical directory tree, our filesystem namespaces are generated based on the semantic queries of the users. Such a mechanism can be described in terms of a set of <attribute, value> pairs. To allow this kind of functionality, extra mappings from attributes to files are necessary. In this way, user's data will be associated to multiple set of semantic attributes which is for runtime generation of namespaces. For instance, when a file is associated to the attributes <type=music, author=*>, then applications can obtain the namespace of the files by submitting a query to the metadata server, and get all the music that categorized into different directories according to the name of authors.

In order to make the namespaces compatible with directory tree, we design the hierarchical namespaces based on attributes mappings. In this way, attribute-based namespace can be dynamically constructed according to the query sentences from users by using the mapping tables. For example, a user's query like <type=movie, attri=action, attri=U.S, attri=2011> will be translated into a set of paths like "/movie/U.S/action/2011", "/movie/action/2011/U.S", "/movie/2011/U.S/action". So, a predefined mapping fashion should be decided. In UMFS implementation, we provide multiple predefined schemes according to the common behavior of users. At the same time, users are enabled to define their own style when navigating their files. Two examples of predefined attribute namespace mappings are shown in Fig. 2.

So, the file types are defined as the top-key when users navigate their files in uMFS. Users are allowed to customize their file types at anytime and any where. The attributes of all files are used to construct the left part of the attributed-based namespace. In this way, users can easily define their favors, and the underlying storages can dynamically mapping user's files to physical storage regardless of their hierarchical locations.



(a) types={data_content} (b) types={data_time}

Fig. 2. Examples of Attribute-Namespace Mappings

C. Performance Manager

In uMFS, the data mapping service has to take on heavy responsibilities in generating namespaces. In addition, the maintenance of various mappings, the monitoring of storage node states all belongs to performance manager. In order to avoid the performance bottleneck when the number of users becomes large, data cache mechanism is necessary in the implementation of uMFS. Firstly, uMFS provides a support for clients to cache their personal metadata such as attribute mappings, file mappings and chunk mappings. Therefore, namespaces can be generated locally and interactions with servers can be reduced. In uMFS a two-layer cache framework is implemented to reduce the number of requests transmitted to the underlying storage devices. First, we apply it locally in the local storage medium, such as flash memory or hard disk; Secondly, uMFS apply a data pre-fetching algorithm and a lease lock file based consistency protocol to make sure that uMFS works properly with wireless network traffic and high performance. With this consistency protocol, users can assure correctness even if the service does not support the relative operations (such as lock) directly. The framework of the two-level cache architecture is shown in Fig. 3.

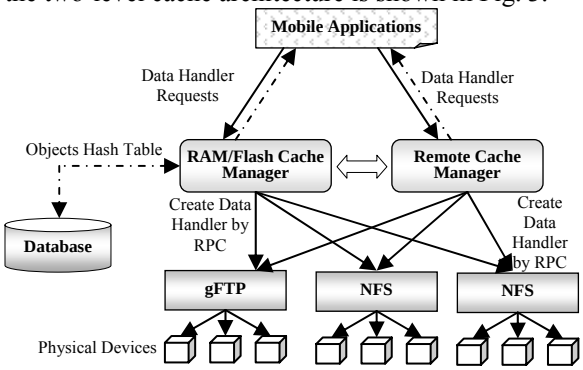


Fig. 3. Two-layered Cache Framework

In the uMFS, it maintains all the recent users' requests in performance manager component, which is used to cache the short-term data or messages. In common sense, data cache will raise the issue on data consistency in some cases. Currently, we provide a simple consistency model to address the issues. The model can be summarized as following two rules: (1) there is no write operation allowed after committed; (2) modification

operation of metadata must be committed to the metadata server instead of local cache. In this cache system, all the user's requests for data handler are first forward to either local cache manager or remote cache manager. Since the calling API of data objects are uniform, the underlying cache framework is transparent to up-level applications. Both two cache managers will maintains all the recently accessed data object handler, which consists of its closest item location, attributes and other data information. If the requested handler does not exist or its validity time is expired, further calling be automatically issued to underlying storage system (i.e. NFS, gFTP or NFS) for creating the handler. Here, it is noteworthy that local cache and remote cache can change information between other, because of the limitation of memory system in mobile devices. Typically, the expiring duration of local cache is shorter than those in remote cache manager.

D. Integrity of User's Data Accessing

In cloud storage system, the most concern of users is the verification of data integrity at unreliable and undependable storage servers. For example, the storage provider, which experiences Byzantine failures occasionally, may decide to hide the data errors from the clients for their own profits. More seriously, storage providers might deliberately delete rarely accessed data belonging to clients so as to saving storage space. So, users should be provided an efficient way to check their data integrity. As mentioned before, mobile devices are of limited storage space and computational capability, therefore, such an integrity check or verification should be conducted remotely without the local copy of target files.

In the existing studies, many security solutions and models are proposed for solving the data integrity verification in distributed systems. However, most of them are not suitable for mobile computing environment, since they do not take the limitations of mobile devices into account. Among these studies, we find that PDP security protocol provides a lightweight security model and can be adopted into our system if some modifications are made. Therefore, we implement our data security service based on PDP security protocol. Here, we only briefly represent our implementation and modifications.

In PDP security model, the efficiency of large-size dataset operations will put heavy burden on clients. For example, the file signature is constructed by the file index information. When a file block is deleted or added, the computation overhead for re-construct the signature becomes unacceptable since the signatures of all the following file blocks should be recomputed with the new indexes. To deal with this problem, we do not use the index information when constructing file signatures. More specifically, considering that a data file D consists of a finite ordered set of blocks $\{m_1, m_2, \ldots, m_n\}$, our signature function is noted as $Sig(m_i)$ instead of $Sig(D, i)$. Furthermore, let M: $G \times G \rightarrow G_T$ represent a bilinear map, where G is Gap Diffie-Hellman (GDH) group and G_T is another multiplicative cyclic group, and let H:$\{0,1\}^* \rightarrow G$ represent a random hash function. Then, the procedure of our security protocol is shown in following steps:

(1) The user send a random generated $\{i, j\}$ to UMFS with a target filename, where i is the file block index and j is the initial hash value in (0, 1].

(2) Both the public and the private key are generated by *Key Manager Service*, and the target file's signature is produced by $Sig(m_i)$. Then, proof message *ProofMsg*(*key*, *Sig*)is produced according the PDP model and sent to authentication server.

(3) Authentication server is responsible to verify the proof message by using *VerifyProof*(*pfm*, *key*, *H*(*j*)). If verifying operation fails, it return false to users; otherwise, an *Integrity Assurance* procedure will be invoked by uMFS.

E. IaaS Adaptor

The IaaS Adaptor is the key component of uMFS, which makes the whole system support heterogeneous architectures of the underlying resources. Also, it is implemented by using plug-in mechanism, which makes it very portable for different platforms. Every plug-in implements an uniform interface specified by the adaptor so that the details of the services can be hidden. In our prototype system, we require plug-in developers to implement the four interfaces (put, get, delete and query) for each plug-in. And the plug-in interacts with its backend service directly to implement the function logic with the service APIs.

Currently, IaaS Adaptor is implemented as a proxy at the bottom of the uMFS framework, and for generalizing all the operations supported by the backend services to the uniform interface of this layer. In order to support most of the online storage service, we set a specification on the interfaces the services should apply, which limit to put, get, delete and query operations; thus the system does not demand the other APIs. The system that supports only a special service would not enable our users to access resources conveniently. Since different service providers always offer different APIs, users need to pay attention to the service that they are using. By the adaptor, uMFS enables mobile applications can uniform multiple storage sites in an identical storage system.

V. EXPERIMENTS EVALUATION AND ANALYSIS

In this section, we present the performance evaluation of uMFS when it is in real world experiments. In addition, we also conduct simulative experiments to investigate the performance of uMFS when it is in presence of heavy workloads, burst workloads and traffic congests. So, the experimental results are categorized into two classes: real world performance, simulative performance.

A. Performance Evaluation by Practical Conditions

As shown in Fig. 1, the performance Manager component is designed to evaluate the other components' performance. As the system is in the phrase of prototype, we incorporate many functions in this component so as to conveniently test the system. During the experiments, there are 531 registered users and about 12% of them use uMFS more than 3 times per day. So we define them as

active users. Firstly, we mainly care about the performance on storage capacity and efficiency. The performance results are collected from 2012-8-1 to 2012-10-1, and shown in Table 1.

TABLE 1

PERFORMANCE STATISTICS ON DATA OPERATIONS

Performance Metrics	Max Value	Min Value	Mean Value
Data Upload Speed (KB/sec)	1553	791	963
Data Download Speed (KB/sec)	2273	1159	1337
Data Throughput (GB/day)	11	0.3	3.85
Response Time of File avigation (sec)	5.9	0.5	1.89
Time of Data Integrity Check (sec/GB)	0.6	0.07	0.274
Total Storage Capability (TB)	33.31	17.42	22.37
Online Active Users (/min)	71	0	5.63
Data Size of Users (GB)	29	0	1.72

The performance evaluation in Table 1 indicates some valuable results for our system. At first, we notice that data download speed is faster than upload speed by 35%. The reason is that the read speed in mobile device is faster than the write speed. As to the metric of data throughput, we notice that it is far from the designing limitation of uMFS. It can be explained by the small number of active users. For example, the online active users are only 5.63/min, which is based on the statistics between 8:00 am to 12:00 pm. If we take the night into account, this value will be smaller. To test the I/O performance that uMFS middleware can provided for up-level applications, we logged all the file-related operations performance, which are shown in Table 2.

TABLE 2.

MEAN I/O PERFORMANCE IN UMFS

Operation	Time (s)	I/O Vol.	Percentage in I/O time	Percentage in Exec Time
Open	1.33	N/A	1.5%	0.08%
Read	14.4	20MB	32.5%	36.2%
Seek	1.25	N/A	1.06%	0.06%
Write	25.6	50MB	61.15%	66.3%
Close	0.21	N/A	0.13%	0.01%

First, we find reads and writes are dominant in I/O time, and volume of data, because the reads are repeated over several iterations in our benchmark. Mainly I/O operations consist of read operations (about 36.2% of the total time). It is followed by writes, which are done to write the integrals to files in the write phase and they contribute to 66.3% of the total I/O time. All other operations such as open, close, seek, and flush take less than 2 percent of the total I/O time. We clearly see that, for this input, I/O plays a major role.

Summarizing the performance results, we can see that our prototype system is capable of deal with more users. In order to investigate its performance under the potential large-size of user groups, we have to conduct simulative experiments, in which we can create virtual users as many as possible, as well as the storage requirements from these virtual users.

B. Performance Evaluation by Synthetic Workload

In the simulative experiment, we used 55 PCs to create virtual mobile users. These PCs are distributed in different institutes, even in different cities. All of them are running a simulation agent, which can create virtual users and send requests to the uMFS server. At first, we test the data upload/download speed with different number of active users, and the results are shown in Fig. 4.

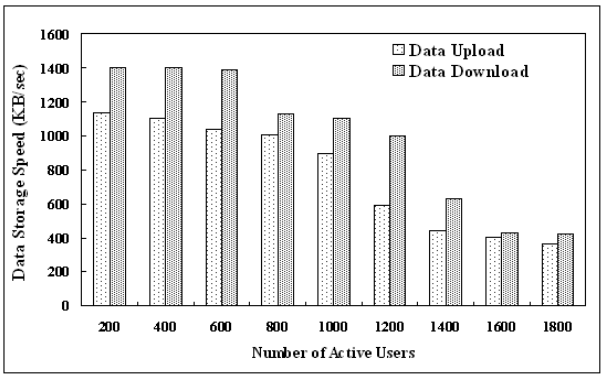


Fig. 4. Data Storage Speed with various Number of Active Users

In simulation, we set the number of active users increased from 200 to 1800. From the results in Fig. 4, we can see that both upload and download speed are decreased when the number of active users increases. However, such performance decreasing is not significant when the active users are less than 800. As soon as the number is over 1000, both upload speed and download speed decrease quickly. Until the active users are over 1800, the data storage speed decreased about 75% relative to the maximal speed. It is noteworthy that such an upload/download seems to be acceptable for mobile users. So, we are confirmed that the uMFS can accommodate at lest 1800 active user concurrently. By our real world experience, the percentage of active users is often less 10%. So, uMFS might be able to provide service for virtual mobile communities with 20000 registered users.

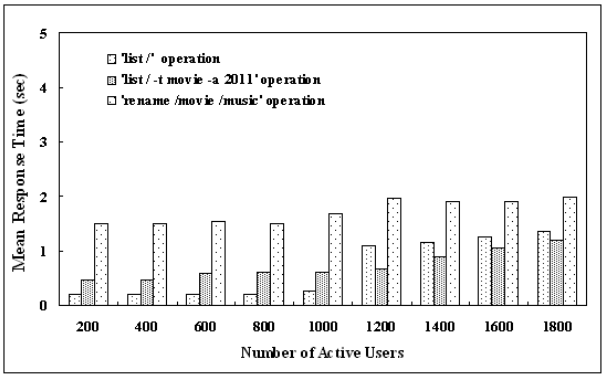


Fig. 5. Response Time of Navigation Performance

We also examine the mean response time when users conduct file navigation operations, and the results are shown in Fig. 5. We mainly test three navigation operation: (1) "list /" is to query the list of user's root directory; (2) "list / -t move –a 2011" is to navigate all the movies that produced in 2011; (3) "rename /movie /music" is to rename a directory. The results show some interesting findings. For example, we notice that rename operation requires more time than other two navigation operations. However, its performance is not affected by the number of active users. It is because that rename operation will lead to the namespace re-mapping, which spends the extra response time. As the list operation, we notice that the response time of "list /" operation increases significantly when the number of active users is over 1000. By carefully examine the logs, we find that the frequency of "list /" operation is more than that of "list / -t move –a 2011" operation about 5 times. So, its performance is more likely to be affected by the number of active users.

Finally, in order to test the performance of throughput metric when system is in presence of heavy workload, we log the real-time throughput of uMFS when active users are 1800, and the results are shown in Fig. 6. The sampling period is an hour.

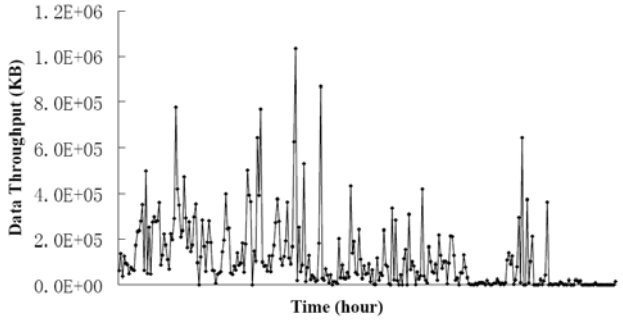


Fig. 6. Real-time Throughput in Simulation with 1800 Active Users

As shown in Fig. 6, the data throughput fluctuates dramatically. The results are decided by the behavior of users. If too many user concurrently sent data upload/down quests, the system throughput will increases in bursts. According to our designing objective, we hope that the top throughput of uMFS is over 0.7 TB per day. This experimental results show that the total data throughput is about 22 GB, which is only 30% of our designing limitation.

V. Conclusion

In this paper, we present a Cloud-based storage framework for providing an easy-to-use file navigation service and lightweight data security mechanism in mobile computing environments. The implantations of the proposed framework are presented in details. Both real world performance and simulative experimental results are evaluated and analyzed. The experimental evaluations show that the proposed framework is effective to provide flexible and reliable data sharing in mobile computing environments. Based on the experimental results, we are confirmed that when the number of registered users is over 20000, the proposed systems can performance well. Currently, our system is just in prototype phase, we plan to conduct more evaluations so as to improve its performance. Also, we are going to incorporate a storage pricing service to supporting the utility-based resource providers.

Acknowledgment

This work is supported by the Provincial Science & Technology plan project of Hunan Grant No: 2012GK3075).

References

[1] T.M. Maarouk, D. E. Saidouni, M. Khergag, "Towards a Calculus for Distributed, Real-Time and Mobile Systems," Journal of Software, Vol 7, No 3 (2012), pp. 564-574, 2012.

[2] C. Boonthum, I. B. Levinstein, S. Olariu et al., "Mobile computing: Opportunities for optimization research," *Computer Communications*, vol. 30, no. 4, pp. 670-684, 2007.

[3] I.-C Hsu, "An Architecture of Mobile Web 2.0 Context-aware Applications in Ubiquitous Web," Journal of Software, Vol 6, No 4 (2011), pp. 705-715, 2011

[4] K.-H. Huang, Y.-F. Chung, C.-H. Liu et al., "Efficient migration for mobile computing in distributed networks," *Computer Standards & Interfaces*, vol. 31, no. 1, pp. 40-47, 2009.

[5] S. Ilarri, E. Mena, and A. Illarramendi, "A system based on mobile agents to test mobile computing applications," *Journal of Network and Computer Applications,* vol. 32, no. 4, pp. 846-865, Jul, 2009.

[6] C. Wang, Q. Wang, K. Ren et al., "Toward Secure and Dependable Storage Services in Cloud Computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220-232, 2012.

[7] P. Mahajan, S. Setty, S. Lee et al., "Depot: Cloud Storage with Minimal Trust," *ACM Transactions on Computer Systems*, vol. 29, no. 4, pp.1-34, 2011.

[8] C. X. Mavromoustakis, and H. D. Karatza, "Under storage constraints of epidemic backup node selection using HyMIS architecture for data replication in mobile peer-to-peer networks," *Journal of Systems and Software*, vol. 81, no. 1, pp. 100-112, 2008.

[9] T. Xie, and H. Wang, "MICRO: A multilevel caching-based reconstruction optimization for mobile storage systems," *IEEE Transactions on Computers*, vol. 57, no. 10, pp. 1386-1398, 2008.

[10] K. Kim, T. Xu, and Y. Cai, "ELIAS: An Efficient Storage Underlay for Mobile Peer-to-Peer Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 11, pp. 1851-1861, 2011.

[11] C.-T. Yang, F.-Y. Leu, and M.-F. Yang, "A Peer-to-Peer Video File Resource Sharing System for Mobile Devices," *Journal of Internet Technology*, vol. 11, no. 1, pp. 69-78, 2010.

[12] Amazon Simple Storage Service (S3): http://www.amazon.com/s3/

[13] Windows Live Mesh: http://www.mesh.com/

[14] Mozy homepage: http://mozy.com

[15] Symantec's Protection Network: http://www.spn.com/

[16] V. Michael, S. Stefan, M.V. Geoffrey. "Cumulus: Filesystem Backup to the Cloud". *Proc. of USENIX Conference on File and Storage Technologies*, pp.1-8,

2009.
[17] Fitzpatrick, B., Brackup. http://code.google.com/brackup/
[18] Jungle disk: http://www.jungledisk.com/
[19] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.H. Bae, J. Qiu, G. Fox. "Twister: a runtime for iterative mapreduce," *Proc. High Performance and Distributed Computing*, pp. 220-227, 2010.
[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica. "Spark: Cluster Computing with Working Sets," *Proc. Of IEEE International Conference on Hot Cloud*, pp.1-8, 2010.
[21] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. "HaLoop: Efficient iterative data processing on large clusters", *Proc. of International Conference on Very Large Database Systems*, pp.102-111, 2010.
[22] R. Power and J. Li. "Piccolo: Building fast, distributed programs with partitioned tables", *Proc of ACM Conference on Open System and Distributed Intelligence*, pp. 331-338, 2010.
[23] Dropbox, http://www.dropbox.com.
[24] Syncplicity, http://syncplicity.com.
[25] MobileMe, http://en.wikipedia.org/wiki/MobileMe.
[26] O. Wolfson, B. Xu, H. Yin, and H. Cao. "Search-and-Discover in Mobile P2P Network Database", Proc. IEEE Int'l Conference Distributed Computing Systems, 2006.
[27] K. Kim, T. Xu, and Y. Cai. "ELIAS: An Efficient Storage Underlay for Mobile Peer-to-Peer Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 22, no.11, pp.1851-1861, 2011.
[28] C. Wang, Q. Wang, K. Ren, et al., "Toward Secure and Dependable Storage Services in Cloud Computing", IEEE Transactions on Services Computing, vol. 5, no. 2, pp. 220-232, 2012.

Dongbo Liu received his master degree in Jiangsu University of in 2004. He obtained the Ph.D degree in Hunan University in 2011, and currently works in Hunan Institute of Engineering as associate professor. His research interests include cloud computing, mobile computing, multi-agent systems and distributed storage management. He is now an IEEE member and ACM member.

Peng Xiao was born in 1979. He received his master degree in Xiamen University in 2004. Now, he works in Hunan Institute of Engineering and is a Ph.D candidate in Central South University. His research interests include grid computing, parallel and distributed systems, network computing, distributed intelligence. He is now an IEEE member and ACM member.