Wukong*: A Cloud-Oriented File Service for Mobile Internet Devices

Huajian Mao[1], Nong Xiao[1], Weisong Shi[2], Yutong Lu[1]
[1]*National University of Defense Technology, Changsha, Hunan 410073, China*
[2]*Wayne State University, 420 State Hall, 5143 Cass Ave, Detroit, MI 48202, USA*
email: {huajianmao, nongxiao, ytlu}@nudt.edu.cn, weisong@wayne.edu

***Abstract*—Along with the rapid growth of heterogeneous cloud services and network technologies, more mobile devices use cloud storage services to enlarge the capacity and share data in our daily lives. We commonly use cloud storage service clients in a straight forward fashion, since we may easily obtain most client-side software from many services providers. However, when more devices and users participate in heterogeneous services, the difficulty increases to manage these services efficiently and conveniently. In this paper we design and implement a novel cloud-oriented file service, Wukong, which provides a user-friendly and highly-available facilitative data access method for mobile devices in cloud settings. By using the innovative storage abstraction layer and a set of optimization strategies, Wukong supports heterogeneous services with a relatively high performance. By evaluating a prototype in a systematic way on the aspects of the supporting interface, system performance, and the system resource cost, we find that this easily operable file service has a high usability and extensibility. It costs about 50 to 150 lines of code to implement a new backend service supporting plugin. Wukong achieves an acceptable throughput of 179.11 KB/s in an ADSL environment and 80.68 KB/s under a country EVDO 3G network with negligible overhead.**

Keywords-cloud computing; file service; mobile devices; heterogeneous; plugin

I. INTRODUCTION

The rising and demanding use of mobile devices like smart phones, Netbooks, iPads and Mobile Internet Devices (MID) [1] grows steadily in daily life. Many people commonly operate several devices at the same time, including laptops, phones, digital picture frames, music players, and so on, often constrained by limited storage capacity. One commonly used method overcomes this problem by copying the data not used soon by a backup device or service (i.e., Amazon S3 [2], Google Docs [3], Dropbox [11]). When we subject changes to the data, then the user synchronizes their data on each device, one by one. Thus, manual migration and synchronization of data between different places happens frequently. Meanwhile, mobile device users also commonly exchange information to collaborate and share data. The demands of our daily actions, like data copying and sharing across multiple users, use resources ubiquitously. With the development of the cloud computing [4] and network technologies, we have met potential feasible methods for large storage capacity and convenience of sharing. Mobile devices carry services, like Google Docs, Amazon S3, and MobileMe [5] as the storage backend in order to enlarge the capacity of storage, and achieve resource sharing via mobile network ubiquitously. However, difficulty increases when more devices and users participate in and more applications need to use the heterogeneous resources in cloud storage to keep their world in sync and convenient. Four challenging issues arise in using such systems for mobile devices.

1) Most of the mobile applications prefer or only support local file system interfaces, so a mobile user has to download files from the service before using them locally. But for mobile devices, especially for those with poor user interface [6, 19], finishing an action with complex operations creates a terrible experience.
2) The users always use multiple types of services in order to satisfy their common need. A system that supports only a special service would not suit our users to access resources conveniently. Since different service providers always offer different public APIs, users should pay attention to the service that they are using but not transparently use the services.
3) With many types of network, the latency and the network bandwidth vary with network types. In order to overcome these challenges, the system should consider additional optimization in the design.

1.1. Our Approach

As we live in a world full of networks, we commonly share information and collaborate. Computer users can upload audio, video and other media into storage services and share it with their friends, as well as acquire information published by the others. For example a mobile device user may capture a beautiful view, and upload the picture onto the Internet to share his amazing moment with his friends. So, not only do we need to make the data sharing available, but we also need to make it a user-friendly and effective way. We envision that the file system interface would be better for most of the applications already working on the devices.

In this paper, we present Wukong, a file service providing a user-friendly, highly-available and facilitative data access method for multiple mobile devices in cloud settings. Deploying Wukong on these devices will ease access for the users to share data with others on the Internet, transparently treating the remote storage services as locale resources. Wukong is originally designed for Mobile Internet Devices; however, it works well on PCs, Laptops, etc. The contribution of this work includes many of the elements listed below.

* Wukong, known as the Monkey King, is the main character in the classical Chinese epic novel *Journey to the West*. He can travel on cloud with the technique called the Jīndǒuyún (cloud-somersault), and he knows 72 transformations, which allows him to transform into various objects. We use it as the name of our file service..

- A Storage Abstraction Layer (SAL) and its plugin mechanism abstract the remote services APIs into uniform interfaces, while the plugins play the interaction with these special services. With the using of SAL and its plugin mechanism, Wukong supports multiple heterogeneous storage services. Wukong can easily extends to support other new storage services, since we only need to implement the interface-well-defined plugin, which only needs about 50 to 150 lines of code.
- Wukong contains a file service transparently accessing heterogeneous services designed to be POSIX compliant so that most of the existing applications, which access local files only, can use the data in cloud environment with network connections (i.e., 3G/WIFI/ADSL) ubiquitously, and it solves the problem of capacity deficiency of the devices.
- Wukong includes a stackable optimization framework with modularizing the functions of file service optimization as elements of the stack. This stackable framework gives Wukong the flexibility and extensibility. Wukong uses a set of strategies to make the service operations with a high performance and low latency,
- We have implemented an extensible prototype system along with plugins for several services including Amazon S3, Google Docs, Google Picasa, FTP, and Email Service, and then evaluated the prototype in a systematic way on the aspects of interface supporting, system performance and the resource cost. The result shows that the performance and usability of this file service is acceptable, and the overhead is negligible.

The rest of this paper is organized as follows. Section 2 describes the overview of the service architecture design, followed by the implementation details in Section 3. Section 4 gives the evaluation of our work based on a prototype system. Section 5 talks about prior related work. We conclude our paper and present the future work in Section 6.

II. WUKONG DESIGN

In a nutshell, Wukong is a file service supporting heterogeneous backend services, allows ubiquitous and safe data access. By designing and implementing Wukong, a file service using heterogeneous storage services as its backends, Wukong surmounts the storage capacity defects of mobile devices and the difficulty of data management. With a storage abstraction layer and its plugin mechanism, the POSIX compliant interfaces make the local traditional applications transparently access the services without any modification. Wukong introduces a cache management strategy to implement the operations with a high performance and low latency. By implementing a relax-lock cache coherence protocol [7], it assures the correctness of the service. According to the properties of mobile network, we design a modular optimization framework in order to improve the performance and data security. Furthermore, we use an encryption module to deal with the data security and a compression module to limit the network traffic. Wukong provides the mobile user a method to deal with the challenges mentioned in Section I.

Two major parts in a system using Wukong include the client device deployed with Wukong and the services used as the storage backends. By deploying Wukong on the devices connected to the Internet, users conveniently share and manage the data resource on heterogeneous services. Figure 1 is an overview of Wukong. It shows the relationship of the clients, the backend services and the network connection.

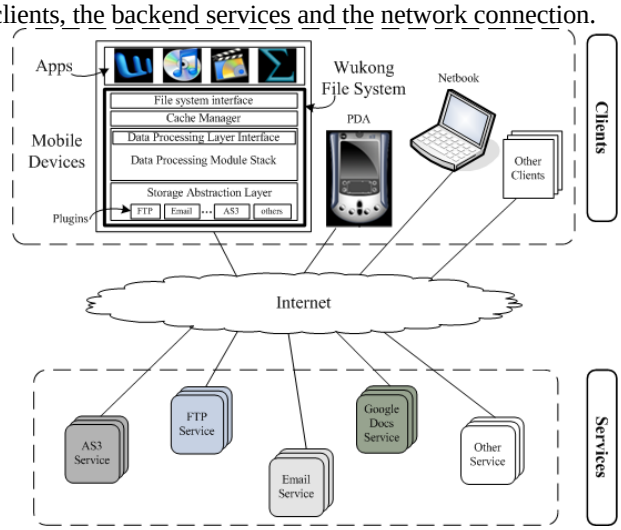

Figure 1. An overview of Wukong.

2.1 File service Components

File service components use a layered design to do the data processing, including the interface layer, the cache manager and the data processing layer. The designs of these components are given in the following sections.

2.1.1 POSIX compliant File Service Interface Layer

The interface layer, the top layer of Wukong's components, implements most of POSIX interfaces, standard UNIX group, owner, and permission semantics. When an application accesses a file, Virtual File System (VFS) determines which file system to access. If the file exists in a Wukong file service, Wukong will send the request to this interface layer by VFS. Wukong interacts with the file system requests from applications. As data may scatter at anywhere on the services, Wukong manages, locates, stores, fetches and queries data with keywords. As we often use standard interfaces to access file system, we may run many interesting and useful applications on this system and transparently access the backend services without any modification. Wukong can treat the remote resources as they are in local.

2.1.2 Cache Manager

We use cache in Wukong to reduce the number of requests transmitted to the remote service. Wukong caches data and metadata, and does not commit the changes to remote services immediately. We apply it first locally and encode it as proper objects, lazily distributed over remote storage services on the Internet. The Cache Manager

manages the cache. We also apply a data prefetching algorithm and a consistency protocol in Wukong to make sure Wukong works properly with low network traffic and a high performance especially for the sequential operations.

2.1.3 Data Processing Layer

According to the properties of mobile network, we design a modular optimization stack framework, *Data Processing Layer (DPL)*, in order to improve the system performance and data security. DPL mainly handles the additional data processing for Wukong, such as data compression, data encryption, and so on. We modularly implement the functions of data processing layer in a function stack framework. In order to achieve these modules pluggable, which means the introduction of a function module or delete will not affect the correctness of the system, every function module has the same interfaces. This design of DPL gives it with more configurability, and improves its functionality flexibility. For example, users can configure the system to compress data first and then encrypt them; or users can also configure the system to encrypt data first then compress.

2.2 Storage Abstraction Layer

The *Storage Abstraction Layer (SAL)*, a key component of Wukong, makes Wukong support heterogeneous services cooperating with its plugin mechanism. As a proxy implemented at the bottom of the Wukong framework, we abstract all the operations (i.e., store, retrieve, trash, identify objects) 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 complicate APIs. Actually, Wukong implements these ones, necessary file system interfaces, on the client side. This specification makes Wukong much easier to be extended to support an unforeseen backend service.

SAL implements a plugin mechanism to support the heterogeneous services. Every plugin implements uniform interfaces specified by SAL so that the details of the services can be hidden for the upper layers. In our prototype system, we require plugin developers the four interfaces (*put*, *get*, *delete* and *query*) for each plugin. And the plugin interacts with its backend service directly to implement the function logic with the service APIs. SAL uses a plugin pool to manage the plugins, which are loaded into the stack at mount time with one plugin for one type of backend service.

Wukong implements the POSIX compliant file system calls with SAL interfaces, for example, *read* by the interface *get*, *write* by the interface *put*, and *lock* by the interface *put* with putting a file representing a lock on the backend service. And these SAL interfaces will call the proper plugin's APIs according to the caller and the user configuration, which is specified at the initialization phase.

2.3 Backend Services

Wukong runs with a thin-server model, which means a minimal interface of the backend service (*put*, *get*, *delete* and *query*) trivially portable to virtually any online storage service, are required. As SAL abstracts the backend service as a local resource, most of the existing applications can use without any modification. With this design, Wukong supports more than one backend service while keeping most applications working, and developers may easily extend its function. SAL implements the plugin mechanism to support the heterogeneous storages. Every plugin gives uniform interfaces defined by storage abstraction layer so that we may hide the details of services. The plugin interacts with its service directly. They implement these interfaces and operate with the special service. SAL with the use of plugin mechanism makes it possible to easily integrate new storage services in WAN or LAN or local.

So, we can use most of the popular storage services for Wukong. Even if the services do not provide these four operations, a possibility for those services to work with Wukong exists. For example, suppose the situation that a user has access to only a read-only FTP service. In this schema, Wukong cannot execute put and delete operations. But we may still use this interface insufficient service as the storage backend by Wukong, rendering Wukong as a read-only file service. These properties of Wukong make it much easier for extension.

2.4 Security

With the broad deployment of wireless network, devices and storage services always stay online. Users may possibly access data in those services ubiquitously. Besides, we can deploy our system in LAN environment too. By connecting the client and the backend service in a LAN environment, the deployed devices can use the resource in a local network conveniently with a high performance. However, we commonly deploy our system on mobile devices, and connect the devices to the Internet with the network connection, and use online services, like cloud service, Email service, FTP, as its backends.

While in a threatening environment, we should keep data safe from corruption and under suitable control and access. That means the system shall use data security to ensure privacy and protect personal data in an insecure environment. Wukong supports standard UNIX file access control mechanisms for users and groups in data security and privacy by plugin operations which can use the ACL APIs offered by the backend services. The security connection is created at the initialization certification phase. Meanwhile, we may deploy the system in an open environment and it may fall prey to attacks in the transmission or on servers, thus, we should encrypt the data. We implement a pluggable module in Wukong to do the encryption. Before storing data to servers, we encrypt it first and then deliver it to the servers. But as encryption costs time, it may lowdown the system performance, so we design it to be optional and configurable, which the user may configure according to their usage schema. In our future work, we may improve this layer to support self-adaptive or hint-direct, with encryption or not.

III. IMPLEMENTATION AND OPTIMIZATION

Wukong includes a set of optimization strategies, like using cache and compression. Wukong uses these methods to make the operations with a high performance and low

latency, and assure the correctness of the cache system. In this section we discuss the details of the implementation and optimization of these technologies.

3.1 Cache Manager

Wukong uses cache to reduce the interactions to the remote service and improve the performance. Files are cached in local storage medium, like flash memory or hard disk. In order to simplify the implementation, we borrow the whole file caching strategy from Coda [8] and assume that a single file is always smaller than the medium capacity. In this section we discuss two technologies of Cache Manager.

3.1.1 Lease-Lock-File Based Consistency Protocol

We use a relax lock, *lease-lock-file based consistency protocol*, to make sure that the cache system works correctly. This is motivated by the observation that the conflict operations do not happen frequently.

In this protocol, lock is implemented by creating a file, which acts as a lock, tagged with lease time on the backend service. Each time we request the write operation, it will first check if any other has locked the object, which means if there is an according lock file. Only if the object returns free which means not locked by other clients, the write operation may continue. This will firstly lock the file by creating a lock file on the remote service in order to avoid writing competition. If the write operation applies the lock successfully, the system then writes data to the remote service, otherwise, the system will stop the following actions.

Wukong tracks with two timestamps for each item in cache: the last validated time T_c and the last modified time T_m. T_c will be updated according to the checking time, and T_m is gotten from the metadata server. T_m will change on the server when any modification any client commits to the file. A cache entry is valid at time T if the interval between T and T_c weighs less than the configurable freshness interval t , or the T_m recorded at the client, similar to the T_m gotten from the server at this time. With an invalid cache entry, we should trash it in order to keep system in approximate consistency. For example, when *A* and *B* open the same file *f*, we Wukong will create locks on these two clients with timestamps included in the lock. The clients will check the locks every interval *t* to find if the validity of the cache. If not, we update the cache. So if *A* edits the file, then we would find invalid cache on *B*, so *B* updates its cache pool.

The selection of a value for *t* is a compromise between consistency and efficiency. We will consistently achieve an approximation to one copy with a very short interval, but it weighs down the load. Furthermore, users do not commonly access data with conflict in our schema, if we stand sure that no conflict will occur, we can disable lock, so that we will achieve a relatively high performance and low system load.

3.1.2 Data Prefetching

With the purpose of using the gap of *open* and *read* operation to improve the sequential operation's performance, we use data prefetching in Wukong.

Data prefetching happens when Wukong receives the open request. The cache component firstly checks to see if we have cached the file. If so, the open operation returns successful immediately, otherwise, it will do a prefetching in backend, and return with the proper code at the meantime. The configuration during the initial mount determines the prefetching size. For example, if the configuration set the size to 0, it means the system shall not prefetch anything, while if the configuration sets it to -1, it means the system will prefetch the whole file when opened. When the prefetched size in memory comes to a pre-defined value, which is also specified by the configuration, Wukong will stores the data to persistent medium, and then go on prefetching. With the assumption given before, data prefetching can work correctly.

Data prefetching always improves the sequential read operation performance. But in random schema, this method costs system resource but may not increase the performance. So, in our future work, we will pay more attention to the data prefetch and also the cache replacement for Wukong.

3.2 File Type Based Adaptive Compression

The network and storage service providers always charge according to the storage size and network traffic, so the smaller data size and traffic would better suit Wukong.

On one hand, we can rely on compression for reducing the object size, which means smaller data will transfer on the Internet and we will use less storage space in the storage service. As we require less data for transfers, the performance of the system will improve, especially for the low bandwidth networks. On the other hand, the processing of compression increases the system resource cost, and not all files have a high compression ratio. Different files have different compression ratio, which relates to its content and type. In the common case, text files always have the higher compression ratio, so we will experience more benefits for the file service to compress data before transfer. While for the binary files, the compression ratio always remains extremely low, and it always costs a lot of computational resource, however, we should not consider this deficient option for our system to compress the data.

In this system, we use the file type specific adaptive compression method, which decides to use compression or not on the file type adaptively. The file types which use compression require configuration by a specification file. And these files have the extensions like *.log, .txt, .tex, .c, .py* which are always text files. Wukong checks the file types when the user requests to open, and then we test the file type to see if this file type exists in the compression type set. If so, the compression module will record the path information. After this step, the operations we decide if we necessitate to compress for write and to decompress for read.

When Wukong compresses the file because of the match of its type, let's suppose the size of original file as *S*. Meanwhile, suppose the extra cost of delivering data to storage services as *D*. So the total quantity that needs to be transferred without compression is $T_{sum} = S + D$. But with the compression module, the data transferred will come to $T_{csum} = r * S + D$, where *r* represents the compression ratio which is always less than 1. We can use compression to reduce the data transfer amount especially for those who have a high compression ratio. And accordingly, the

performance of Wukong improves. In the prototype of Wukong, we can configure the compression method, and by default, it uses *gzip*.

Compared with the strategy without compression, our method significantly reduces the transfer size, especially for those with a high compression ratio, while compared with the compress-everything method, our strategy keeps the overhead minimum, especially for those with low compression ratio.

IV. PERFORMANCE EVALUATION

This section evaluates Wukong with several experiments. Figure 2 shows the experimental platform, where two kinds of client hosts, laptop and Mobile Internet Device (MID), use two types of network connections, *ADSL* and *3G (EVDO)* provided by China TeleCom, to connect to services, including Google Docs, Amazon S3, Google Picasa and others. Figure 2 also shows the hardware and software configurations of the clients. In the testing, we intentionally choose a rural site, which means the 3G network is not very stable and may downgrade automatically. Although the ideal bandwidth is 3.1Mbps, the real one is always much smaller.

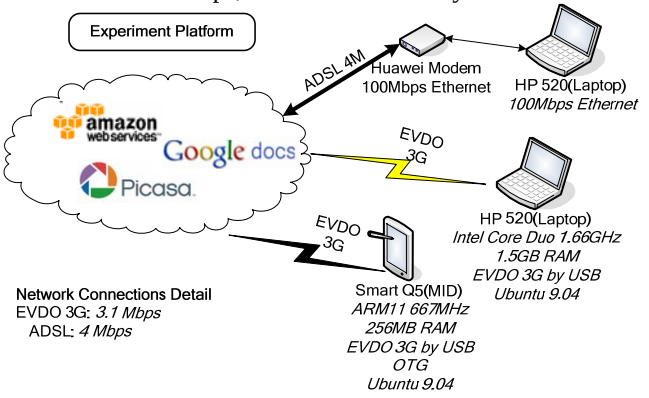

Figure 2. Configuration of the experiental platform. All the client hosts are deployed with Wukong.

We synthesize the workload with files of different types: executable binary file (binary), Microsoft Word Document (doc), images (image), system log (log), Printable Document File (PDF), music file (music). The input to the operations consists of 43 files, about 43 Mbytes in total, and the details of the workload are shown in TABLE I.

TABLE I. WORKLOADS FOR EVALUATING WUKONG SERVICE.

File Type	File Number	Total Size	Source
binary	9	3.3 MB	/usr/bin
doc	6	408 KB	Google Docs
image	14	3.6 MB	Flickr
log	3	23 MB	/var/log
PDF	5	2.7 MB	5 papers of SCC09
music	6	9.9 MB	Google.cn music
misc	43	43 MB	Merge of them

With this workload, we evaluate Wukong as follows: First, we examine the function of Wukong with applications, and state the convenience and flexibility of Wukong plugin with the statistics of the line of code of the plugins. Next, we evaluate the effects of the cache strategies and compression. Finally, we measure the throughput and the system resource consumption with the workloads presented previous under the environment showed in Figure 2.

4.1 Functions

In this section, we present two experiments focusing on checking the service interface. First, we write several benchmarks, which call common file system interfaces including *open*, *creat*, *close*, *read*, *write*, *lseek*, *mkdir*, *unlink*, and so on, then we run them on our deployed client hosts. We find that they pass as expected, which means that Wukong supports most of the standard POSIX compliant file system interfaces. However, Wukong still does not support interfaces like *statfs*/*symlink* in this prototype version.

Besides, Wukong supports applications that involve a set of file system calls, including metadata operations and data operations. We choose several typical upper layer applications to check if they could get along well with Wukong. We used several media players, including *mplayer*, to play the audio and video files stored in Wukong, and we also edit the files in the service by editors such as *emacs* and *vi*. We have proved that Wukong supports applications which involve that set of file system calls.

4.2 LoC of Wukong Plugins

As discussed, Wukong uses SAL and plugin mechanism to support transparently accessing the heterogeneous services. In the prototype system, we implement the following plugins in Python, including *pgdocs*, which interacts with Google Docs Service, *pgpic*, that manages the photos stored in Google Picasa Service, *pas3*, which Amazon S3 oriented manages, *pftp*, which makes Wukong use the FTP Service as the storage backend, *pimap4*, which uses IMAP4 enabled Email Service, and *pdisk*, which uses local storage medium.

What a plugin developer needs to do is to implement the *get*, *put*, *query*, *delete* interfaces as we discussed before. So the work needed to do for implementing a new plugin would be easy. We do a statistics on the Line of Code (LoC) of these plugins, and the result is showed in TABLE II. As shown, the LoCs are always small, inhabit about 50 to 150. This is mainly benefit from the specification of the interfaces. So the main advantages of Wukong plugin include its simplicity and extensibility. Wukong may easily add a new backend service support by implementing a new plugin. For example, suppose the work efficiency of a graduate student is 50 lines per day, then he can write a new plugin in 1-3 days. And if he has a little higher efficiency, he can implement a new backend service supporting even in 1 day.

TABLE II. LOC OF THE WUKONG PLUGINS

Plugin	pgdocs	pgpic	pas3	pftp	pimpa4	pdisk
LoC	77	95	108	63	137	44

As the LoCs of Wukong plugins tell, plugin mechanism takes Wukong lots of advantages. First, plugin is separated from the overall architecture, and what the plugin developer cares about is the plugin logic while avoiding the complex management of the file service. Second, only a small amount

of work, as the LoC of the plugins shown in TABLE II, need to be done for a new service supporting. Besides, the plugin mechanism gives Wukong high function extensibility.

4.3 Evaluation on Cache

We present two experiments based on Amazon S3 service to see how the cache affects Wukong. We run several file system related commands, like '*cp*', '*ls*', '*rm*', in the deployed Wukong without any other optimization, and record the latencies of those commands. We execute these commands twice with cache, one in a warm cache and the other one in a cool cache. With the purpose of comparison, we also record the latencies of the conditions of no cache. Figure 3 shows the comparison of different latencies.

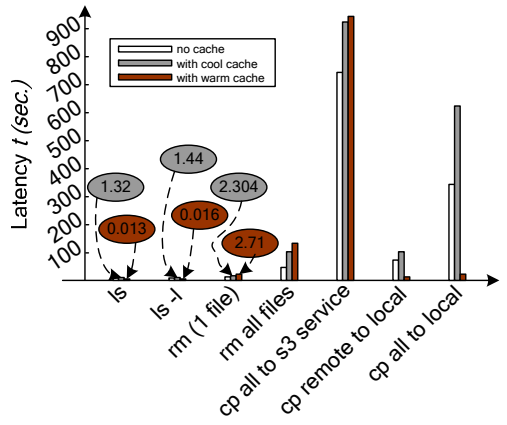

Figure 3. Comparison of latencies of different commands.

From the result shown in the previous figure, we can find that the latencies of no cache are always smaller than those of cool cache. The reason is that the cache manager introduces overhead not only to store the data in cache but also to maintain the cache consistency. But cache manager does take in advantages, which we observed from the comparison between the latencies of the cool cache and the warm one. The warm cache cost only a little to finish these commands which are far smaller than that of the cool cache.

Also, we take an experiment focusing on data prefetching, by opening a file and then reading its content after several intervals, to see how long does it cost for reading. We present here the second experiment evaluating the prefetch of Wukong. In this experiment, we firstly open several files in the deployed Wukong, then wait for an interval whose values is are chosen to be *0*, *1*, *2*, *4*, *8*, *16*, *32* and *64* seconds, and after that, time the latencies of reading the whole opened file. The files are one music file (*5.5 MB*), one binary file (*1.12 MB*) and one log file (*1.16 MB*). As the result says, the latencies to read the files wholly are 55.7 seconds for the music file, 11.9 seconds for the binary file and 11.4 seconds for the log file. Besides, we find that the sum of interval time and the latency of read operation almost keep the same for all these three files. It means Wukong can use the interval between open and read operations to prefetch data, and keep the read latency low. When the whole file is prefetched by the open operation and stored in cache, it cost 1.43s for the music file, 1.26s for the binary file, 1.25s for the log file to read the content from cache. With this characteristic, Wukong will be good at those situations where there is a large time gap between the open and read operations.

4.4 Compression

First, we evaluated the efficiency of our file type based adaptive compress with the workloads in TABLE I. In this experiment, we choose those with the extensions like *doc*, *log*, and *pdf* as our compression candidate files. With this purpose, we transfer all the data to the remote service by Wukong with adaptive compression, and then calculate the data size in the server. For comparison purposes, TABLE III contains the values of none and full compression strategies as well. The unit of the values is *MBytes.*

TABLE III. COMPARASION OF DIFFERENT COMPRESSION STRATEGIES

Type	bin	doc	image	music	log	pdf	misc
none	3.3	0.4	3.6	9.9	23	2.7	44
full	1.9	0.15	3.5	9.8	4.2	2.2	22
adapt.	3.3	0.15	3.6	9.9	4.2	2.2	24

In addition to the efficiency evaluation, we also monitor the overhead of the file type based adaptive compression method. We log the timestamps when the compression start and end, then calculate the delta of the values, and sum up all the delta values to see how long does Wukong spend on compressing. Figure 4 shows the overhead of file type based adaptive compression compared with the none and full ones.

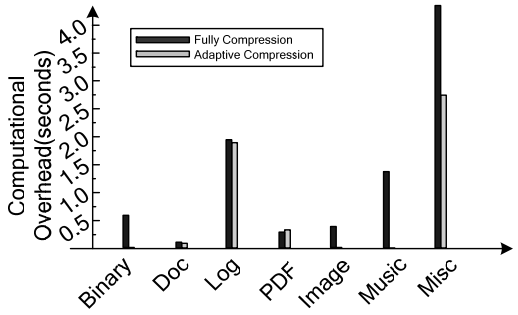

Figure 4. Computation time of the compression overhead.

We find that the above two experiment results confirm the benefits of file type based compression method, not only that the time consumption costs by adaptive compression similar to the method without any compression, but also that the size reduced by adaptive compression remains only a little smaller than the full compression. However, as shown in TABLE III, *pdf* files may not have a high compression ratio, and if we choose the wrong type to be in the compression candidate set, it costs time to compress but reduces the data size only a little. We must vitally choose a good candidate set for file type based adaptive compression layer, so that the adaptive compression method can reduce the transfer size, while keep the overhead minimum.

4.5 Throughput and Resource Cost

In this section, we will evaluate the throughput and the system resources (*cpu* and *memory*) consumption in the environment which are shown in Figure 2.

Wukong easily transfers files from one remote service to another, which copies files one by one from one remote service to local and then copy them to the other service. In order to see this process clearly, we separate it into two parts, evaluating the process of copying files from remote service to local (*download*) and copying file to remote service from local (*upload*) apart. The system resource consumption is captured by the Linux command *top*. We averagely sampled 200 points for each process of the experiments both the CPU and memory consumption information.

TABLE IV. CONSUMPTIONS UNDER DIFFERENT ENVIRONMENTS

Tasks	**Copy From Remote Service**			**Copy To Remote Service**		
Environment	***Laptop ADSL***	***Laptop 3G***	***MID 3G***	***Laptop ADSL***	***Laptop 3G***	***MID 3G***
Throughput (Kbyes/s)	179.11	48.93	80.68	66.82	16.92	17.99
Avg. CPU Ratio (%)	0.38	0.01	3.5	0.24	0.036	2.06
Avg. Mem. (Kbytes)	6761	5412	6712	12700	6711	11066

TABLE IV shows the throughput and the average consumption in our experiment platform. The reason why the throughput of *Laptop + 3G* and *MID + 3G* is far smaller than the ideal bandwidth is that our 3G network is a country-side EVDO, and the signal is always weak and unstable. Meanwhile, as the total transfer size stays the same, as the throughput increases, the system spends shorter amounts of time to transfer. Besides, we monitored the system resource consumption at the same time when we are recording the throughput. Figure 5 shows the CPU ratio that Wukong uses at the sampled timestamps. And the memory consumption is showed in Figure 6.

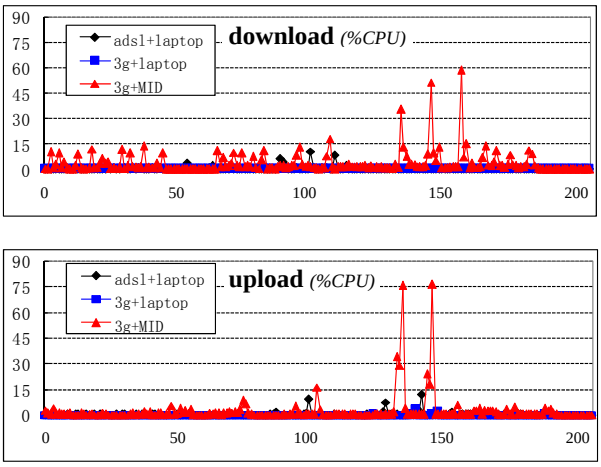

Figure 5. CPU ratio consumption of transferring data. 200 timestamps are sampled for each experiment, and each point in the figure represents the sampled consumption value of *CPU ratio*.

As Figure 5 shown, the CPU consumption of *MID+3G* is always higher than those of the other two. This is mainly because of that the CPU frequency of MID is 667 *MHz*, while that of laptop is 2.33 *GHz*. Although the tasks are the same and they do the same computation for compression and other actions, MID needs a relative higher CPU consumption ratio than that of the Laptop. However, as the average value of CPU ratio shown in TABLE IV, the average consumption of MID is only about 3.5% for download and 2.06% for upload. And this resource consumption is negligible.

Besides, both in Figure 5 and Figure 6, there are several peak points whose values are much larger than the other points. In order to see what happen with these timestamps, we look into the details of the data transfer process, and we find that the peak points of CPU ratio reside in the times when the system need a lot of compression, and that of the memory consumption reside in the times when the system is processing the big files.

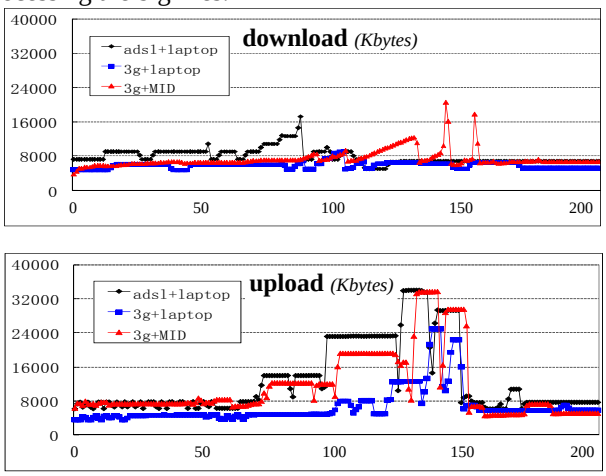

Figure 6. Memory consumption of transferring data. 200 timestamps are sampled for each experiment, and each point in this figure represents the sampled consumption value of *Memory size*.

From the results of our experiments, we can find that the prototype system contains acceptable performance and resources consumption.

V. RELATED WORK

Wukong shares its goals with several recent efforts aimed at simplifying the data management for cloud service [9]. We categorize related research into two groups as *cloud storage service clients and platforms*, and *Network and distributed file systems*.

Cloud Storage Service Clients and Platforms. Using a cloud storage service client is quite a straight and common way for data management on the services, since most of the clients can be easily obtained from the service providers. Dropbox [11] is a cross-platform cloud-based storage application and service. The service enables users to store and sync files online and between computers and share files and folders with others. Syncplicity [12] is a similar service provided by Syncplicity Inc. Besides, as Google gives the Internet users the ability to store any type of data on their Google Docs service, Memeo publishes Memeo Connect [13], which is a client to manage the Google Docs. Also, there are several cloud service platforms which are very usable and famous. MobileMe [5] is a collection of online services and software offered by Apple Inc. Actually it is a service platform which allows users to manage their data, mail, calendar, address book and the others ubiquitously. Similar to MobileMe, Live Mesh [14], a data

synchronization system from Microsoft, allows data sharing and synchronization across multiple devices. The service clients and platforms provide the users a great way to manage the services and make their world in sync. However, they are not very suitable for the situations which involve complex operations. Wukong implements POSIX compliant file system interfaces, so that the applications on the client deployed with Wukong can transparently use the resources on the service. This would be preferred for most of the users, especially for those whose devices are poor in user interface.

Network and distributed file systems. They have been extensively studied in the past. The Andrew File System (AFS) [15] is a distributed networked file system which uses a set of trusted servers to present a homogeneous, location-transparent file name space to all the client workstations. Coda [8] descends from AFS. It has many great features. It keeps working even when the network disconnects or in weak connection [10]. It has high performance through client side persistent caching, and so on. LBFS [16] is a network file system designed for low bandwidth networks. It exploits similarities between files or versions of the same file to save bandwidth. Cegor [17] proposes to build an adaptive distributed file system which provides the "ClosE and Go, Open and Resume" (Cegor) semantics across heterogeneous network connections, ranging from high-bandwidth local area network to low-bandwidth dial-up connection. It provides lots great ideas on the key techniques. Several other works in this area, like Zebra [18], GmailFS [20], Cumulus [21], S3FS cover these similar regions. Although these work on diverse aspects, including network challenges, cache management, offline operations and so on, have proposed and studied for the system performance, usability and availability, but most of the file systems are based on the special server. For example, Coda need the supporting of Coda Server, and S3FS or Cumulus need the Amazon S3 service, but if when only the ftp service is available, Coda and Cumulus will not be usable. Wukong is a file service supporting heterogeneous backend services by using a storage abstraction layer. Even if the backend service is new, it is easy for Wukong to implement a new plugin for this service, and make it deployable with the new service.

VI. Conclusions and Future Work

We presented the Wukong, a cloud-oriented file service for mobile devices in this paper. Wukong characterizes itself with several unique features:

- Provides a standard POSIX compliant interface so that the existing applications can be deployed on this service directly or with few modifications.
- Supports multiple heterogeneous storage services and has a capability to support new or unforeseen services.
- Introduces negligible overhead while providing an easy way to access cloud services in mobile devices.

However, we will address several limitations in our work in the future. First, in the storage abstraction layer, we assume that only a few interfaces are offered, but actually services may offer rich interfaces, which may lead to a high performance. We would do work more in storage abstraction layer. Second, we require systematic optimizations for Wukong to improve its performance, including reducing the latency and improving the throughput. Besides, we base our current evaluation on a random selected data set and a fixed set of programs. While for file system users, the access pattern to the file system has its special characteristics. We will do workload characteristics analysis in the future work.

Acknologyment

We would like to thank the anonymous reviewers for their valuable feedback and suggestions. This work was supported by the National High Technology Research and Development Program of China (863 Program) under Grant No.2006AA01A106 and the National Natural Science Foundation of China under Grant No.60736013.

References

[1] SmartQ5. http://www.smartdevices.com.cn/index.html.

[2] Amazon S3. http://status.aws.amazon.com/s3-20080720.html.

[3] Google Docs. http://docs.google.com.

[4] D. Chappell. A Short Introduction To Cloud. Technical report, Microsoft Corproration, 2008.

[5] MobileMe. http://en.wikipedia.org/wiki/MobileMe.

[6] Landay, J.A. and Kaufmann, T.R.. User interface issues in mobile computing. Proceedings of Fourth Workshop on Workstation Operating Systems, 1993.

[7] Q. Lu and M. Satyanarayanan. Improving data consistency in mobile computing using isolation-only transactions. Proceedings of the Fifth IEEE HotOS Topics Workshop, 1995.

[8] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., and Steere. Coda: A highly available file system for a distributed workstation environment. IEEE Transactions on computers, 1990, 39(4): 447~459.

[9] Abadi, D.J.. Data management in the cloud: Limitations and opportunities. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2009, 32(1): 3~12.

[10] J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. ACM Transactions on computer Systems, 1992, 10(1).

[11] Dropbox. http://www.dropbox.com.

[12] Syncplicity. http://syncplicity.com

[13] Memeo Connect. http://www.memeoconnect.com.

[14] Live Mesh. http://en.wikipedia.org/wiki/Live_Mesh

[15] J. Howard. An overview of the andrew file system. Proceedings of the USENIX Winter Technical Conference, 1988: 23~26.

[16] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. Proceedings of the Eighteenth ACM symposium on Operating systems principles, 2001, 5: 174~187.

[17] W. Shi, H. Lufei, and S. Santhosh. Cegor: An adaptive, distributed file system for heterogeneous network environments. Proceedings of the Tenth International Conferences on Parallel and Distributed Systems, 2004, 10: 145-142.

[18] J. Hartman and J. Ousterhout. The Zebra striped network file system. ACM Transactions on Computer Systems, 1995, 13(3): 274~310.

[19] Brown, Q. and Lee, F.J. and Salvucci, D.D. and Aleven, V.. Interface Challenges For Mobile Tutoring Systems. Lecture Notes in Computer Science, 2008.

[20] GmailFS: GMail virtual file system. http://richard.jones.name/google-hacks/gmail-filesystem/gmail-filesystem.html.

[21] M Vrable, S Savage, GM Voelker. Cumulus: Filesystem Backup to the Cloud. Proceedings of the Seventh USENIX Conference on File and Storage Technologies, 2009.