J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Wukong: A cloud-oriented file service for mobile Internet devices☆

Huajian Mao [a], Nong Xiao [a,*], Weisong Shi [b,c], Yutong Lu [a]

[a] Department of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan 410073, China
[b] Department of Computer Science, Wayne State University, Suite. 14102, 5057 Woodward Ave, Detroit, MI 48202, USA
[c] Tongji University, Shanghai, China

ARTICLE INFO

Article history:
Received 11 February 2011
Received in revised form
23 July 2011
Accepted 31 October 2011
Available online xxxx

Keywords:
Cloud computing
File services
Mobile devices
Plugins

ABSTRACT

Along with the rapid growth of heterogeneous cloud services and network technologies, an increasing number of mobile devices use cloud storage services to enlarge their capacity and share data in our daily lives. We commonly use cloud service client-side software in a straightforward fashion. However, when more devices and users participate in heterogeneous services, the difficulty of managing these services efficiently and conveniently increases. In this paper, we report 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. Wukong supports mobile applications, which may access local files only, transparently accessing cloud services with a relatively high performance. To the best of our knowledge, Wukong is the first file service that supports heterogeneous cloud services for mobile devices by using the innovative storage abstraction layer. We have implemented a prototype with several plugins and evaluated it in a systematic way. 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 support plugin. Wukong achieves an acceptable throughput of 179.11 kB/s in an ADSL environment and 80.68 kB/s under a countryside EVDO 3G network with negligible overhead.

1. Introduction

Both the use and the demands of mobile devices such as smart phones, Netbooks, iPads, and mobile Internet devices (MIDs) [36] are growing steadily in our daily life. People commonly work on multiple devices, such as capturing pictures with a mobile phone, modifying these pictures with a laptop, showing them in a digital picture frames, and so on. However, these devices are often constrained by limited storage capacity and complex synchronization steps. One commonly used method to overcome this problem is storing the data in a backup device or service (e.g., Amazon S3 [4], Google Docs [14], Dropbox [12]). When changes are made to the data, users synchronize 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, such as data copying and sharing across multiple users, are increasing steadily.

Also, we commonly share information and collaborate [8,30,21]. Computer users can upload audio, video, and other media into storage services and share it with their friends, and they cans acquire information published by others. For example, a mobile device user may capture a beautiful view, and upload the picture onto the Internet to share it with friends.

With the development of cloud computing [10,32,3] and network technologies [43,40,31], we have found potential feasible methods for large storage capacity and convenience of sharing. Mobile devices carry services such as Google Docs, Amazon S3, and MobileMe [28], as the storage backend in order to enlarge the capacity of storage, and achieve resource sharing via mobile network ubiquitously. However, the difficulty increases when more devices and users participate, and more applications need to use the heterogeneous resources in cloud storage to keep their world in sync, conveniently, and in a secure manner. Three 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 interfaces [26,9], finishing an action with complex operations creates a terrible experience.

☆ Wukong, known as the Monkey King, is the main character in the classical Chinese epic novel Journey to the West. He can travel on a cloud using a technique called a Jindouyun (cloud-somersault), and he has the ability of transforming himself or other objects into other shapes. We use it as the name of our file service.

* Corresponding author.
E-mail addresses: huajianmao@nudt.edu.cn (H. Mao), nongxiao@nudt.edu.cn (N. Xiao), weisong@wayne.edu (W. Shi), ytlu@nudt.edu.cn (Y. Lu).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.10.017

2. Users always use multiple types of service in order to satisfy their common requirements. They may use Google Docs to backup documents and slides, use Picasa to save their pictures, and use Amazon S3 to store some other files. A system that supports only a special service would not enable 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 network latency, bandwidth, and security vary with network type. Also such networks are sometimes not very stable, which may lead to poor usability. In order to overcome these challenges, the system should consider additional optimization (e.g., cache, compression) in the design.

So, not only do we need to make data sharing available, but we also need to do it in a user-friendly and effective way. We envision that a file system interface would be better for most of the applications already working on such devices.

1.1. Our approach

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

- A storage abstraction layer (SAL) and its plugin mechanism abstract the remote services APIs into uniform interface, while the plugins play the interaction with these special services. Using the SAL and its plugin mechanism, Wukong is able to support multiple heterogeneous storage services. It can be easily extended to support other new storage services, since we only need to implement the interface-well-defined plugin, which only needs about 50–150 lines of code.
- Wukong introduces the capability of service mashup for users. Wukong based applications can be developed to support different services, e.g., Google Docs, Google Storage, Amazon S3, at the same time. Applications also can be designed to support network sharing, integrated with Facebook, Twitter, and other social network services. Besides, with this property, Wukong can also be extended to implement other applications.
- Wukong contains a file service, designed to be POSIX compliant, so that most of the existing applications, which access local files only, can use the data in a cloud environment with network connections (e.g., 3G/WiFi) transparently. It solves the capacity deficiency problem of the devices.
- Wukong includes an optimization stack, and optimization functions are implemented as elements of it. This stackable framework gives Wukong flexibility and extensibility. Wukong uses a set of strategies to make 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 in regard to aspects of interface support, system performance, and resource cost. The result shows that the performance and usability of this file service are acceptable, and that 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 the prototype system. Section 5 proposes several potential Wukong based applications. Section 6 talks about prior related work. We conclude our paper and present the future directions in Section 7.

Table 1
Component interfaces of Wukong.

Interface	Parameters	Description
init	child, Priv	Initialize this module
getattr	path	Get file attribution by path
readdir	path	Read the list of a directory
create	path, mode, dev	Create a file
trash	path, type	Delete a file
open	path, flags, mode	Open file
read	path, length, offset	Read content from the file
write	path, buf, offset	Write content to the file
close	path	Close file

2. Wukong design

2.1. Assumptions and overview

In designing a file service for our needs, we have been guided by some assumptions.

- Most of the files are operated in the one-write–many-read schema. The case in which many write operations happen concurrently is rare.
- One device is used by one person at a time, which means that the case in which more than one user logs in to use a device is not possible.
- The size of a single file operated on the device is always smaller than the local storage capacity.

Based on these assumptions, Wukong supports heterogeneous backend services, allowing ubiquitous and safe data access. Through its design and implementation, 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. A fake service based offline mode makes the system have a high availability. Wukong introduces a cache management strategy to implement the operations with a high performance and low latency. By implementing a relaxed-lock cache coherence protocol, it assures the correctness of the service. According to the properties of the 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.

Two major parts in a system using Wukong are 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. Fig. 1 is an overview of Wukong. It shows the relationship of the clients, the backend services, and the network connection.

2.2. File service components

Wukong uses a layered file service component design to do the data processing. These layers include the interface layer, the cache manager, and the data processing layer. They share the same interfaces, similar to the POSIX compliant file system calls. Table 1 shows part of these interfaces. The designs of these components are given in the following sections.

POSIX compliant file service interface layer. The interface layer, the top layer of Wukongs components, implements most of the

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

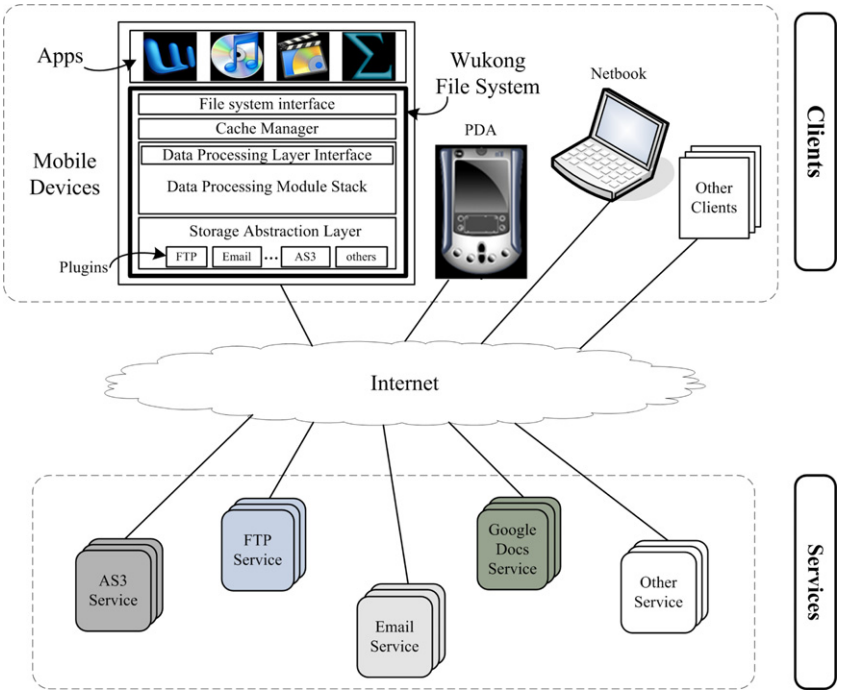

Fig. 1. An overview of Wukong.

POSIX interfaces, standard UNIX group, owner, and permission semantics with the Filesystem in userspace, FUSE [39]. When an application accesses a file, a virtual file system (VFS) determines which file system to access with the path information. If the file exists in a Wukong file service, Wukong will send the request to this interface layer by the VFS. Wukong interacts with the file system requests from applications, and manages, locates, stores, fetches, and queries data which may scatter anywhere on the services. 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. Wukong can treat the remote resources as if they are local.

Cache manager. We use the 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. First, we apply it locally in the local storage medium, such as flash memory or hard disk. Then Wukong encodes the changes into proper objects, lazily distributed over remote storage services on the Internet. Besides, we apply a data prefetching algorithm and a lease lock file based consistency protocol in Wukong to make sure that Wukong works properly with low network traffic and high performance. With this consistency protocol, Wukong assures correctness even if the service does not support the relative operations (such as lock) directly. In order to simplify the implementation, we borrow the whole file caching strategy from Coda [33], and assume that a single file is always smaller than the local storage medium capacity.

Data processing layer. According to the properties of the mobile network, we design a modular optimization stack framework, the data processing layer (DPL), in order to improve the system performance and data security. The DPL mainly handles the additional data processing for Wukong, such as data compression, data encryption, and so on. We implement the functions of the data processing layer as modules in a function stack framework. In order to make sure that the introduction of a function module or deletion will not affect the correctness of the system, every function module has the same interfaces as those in Table 1. This design of the DPL gives Wukong high configuration, and improves its functional flexibility. For example, users can configure the system to compress data first and then encrypt; or users can also configure the system to encrypt data first then compress the encrypted data. Details of the optimization will be introduced in Section 3.

2.3. Storage abstraction layer

The storage abstraction layer (SAL), a key component of Wukong, makes Wukong support heterogeneous services cooperating with its plugin mechanism. There are two main functionality of the SAL: heterogeneous service operation abstraction and different service mashup.

Cloud storage service clients and platforms. As a proxy implemented at the bottom of the Wukong framework, the SAL abstracts all the operations (e.g., 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 APIs. Actually, Wukong implements these necessary file system interfaces on the client side. In these, *put* stores data objects, including data and metadata, *get* grabs data objects with a unique identifier, *delete* trashes the objects either not available anymore or deleted, and *query* searches and gets information from remote services with a certain keyword. This specification makes Wukong much easier to be extended to support an unforeseen backend service.

The SAL implements a plugin mechanism to support the heterogeneous services. Every plugin implements uniform interfaces specified by the SAL so that the details of the services can be hidden. In our prototype system, we require plugin developers to implement 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. The 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. In order to make the upper layers loose coupling, the

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

```
def getattr(self, path):
    query = plugin.query(path, 'meta')
    if query successful:
        set sstat with query
    else: sstat = os.lstat(lpath)
    return sstat

def readdir(self, path, offset):
    dlist =plugin.query(path, 'list')
    return list(dlist + ['.', '..'])

def create(self, path, mode, typ):
    create object in local cache
    plugin.put(path, typ, '')

def trash(self, path, typ='f'):
    plugin.delete(path, typ)

def open(self, path, flags):
    if plugin.query(path, 'chk'):
        on_remote = True
    if exists in local: in_local = True
    if open for read:
        if on_remote or in_local: return
        else: raise OSError()
    elif open for write:
        if on_remote: plugin.delete(path, 'f')
        plugin.put(path, 'f', '')

def read(self, length, offset):
    content = plugin.get()
    return content[offset:offset+length]

def write(self, buf, offset):
    if on_remote: plugin.delete(path)
    content = compose with buf & offset
    plugin.put(path, 'f', content)

def close(self):
    release resources

def others...
    ...
```

Fig. 2. The interface conversion between SAL interfaces and plugin APIs. Parts of the interface conversions are shown in this figure.

SAL shares the same interfaces as the DPL. The SAL uses the plugin APIs to implement these interfaces. Fig. 2 shows the transform between SAL interfaces and the plugin APIs. We will give a sample plugin implementation sketch in Section 3.5.

Service mashup. This is the other of the two most important properties of the SAL. It is motivated by the observation that users always use multiple types of service in order to satisfy their common requirements. They may use Google Docs to backup the documents and slides, use Picasa to save their pictures, and use Amazon S3 to store some other files. A system that supports only a special service would not enable 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.

With this service mashup property, Wukong based applications can be developed to support different services, e.g., Google Docs, Google Storage, Amazon S3, at the same time. Applications also can be designed to support sharing integrated with Facebook, Twitter, and other services. With this property, Wukong even can be used in multi-mode annotation applications. We will give several potential applications based on Wukong in Section 5.

2.4. Backend services

For public remote services, having a programmable interface to them is important, so that users can write client software by using this interface to manage the service. However, different service providers have published their proprietary APIs, which always have differences. Fortunately, the REST interface is supported by most of the services. So, Wukong runs with a thin-server model, which means that a minimal interface set of the backend service (*put*, *get*, *delete*, and *query*) trivially portable to virtually any online storage service is required. However, it is easy for Wukong to change to use the other interfaces such as SOAP, filesystem interfaces, and so on. The modification work is just to modify the SAL. Also, it is convenient for the plugins to convert these types of interface into REST interfaces.

Besides, as the SAL abstracts the backend service as a local resource, most of existing applications can be used 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. The SAL implements the plugin mechanism to support heterogeneous storage. Every plugin gives uniform interfaces defined by the SAL, so we may hide the details of the services. The plugin interacts with its service directly. They implement these interfaces and operate with the special service. The SAL with the use of plugin mechanism makes it possible to easily integrate new storage services in WAN, LAN, or local environments.

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 a read-only FTP service. In this schema, Wukong cannot execute put and delete operations. But we may still use this interface-insufficient service, which publishes content that can be used as the storage backend by Wukong, rendering Wukong as a read-only file service. These properties of Wukong make it much easier for extension.

With the broad deployment of wireless networks, devices and storage services can always be connected. Users may possibly access data in the services ubiquitously. Besides, we can deploy our system in a 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 a network connection, and use online services, such as cloud service, Email service, and FTP, as its backends.

2.5. Security

Several assessments and surveys [22,18,37] have reported that it is important to improve the security of cloud computing services. [6,13] also researched the security of file systems. In a threatening environment, we should keep data safe from corruption and under suitable control and access. That means that the system must use data security to ensure privacy and protect personal data in an insecure environment.

The major security concern of a distributed file system is client request authentication and authorization. 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 configurable 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 reduce the system performance, so we design it to be optional and configurable; users may configure it according to their usage schema. In our future work, we may improve this layer to support self-adaptive or hint-direct selection, with or without encryption.

3. Implementation and optimization

Wukong includes a set of optimization strategies, such as using a cache and compression. Wukong uses these methods to make the operations with a high performance and low latency, and to ensure the correctness of the system. In this section, we discuss the details of the implementation and optimization of these technologies.

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

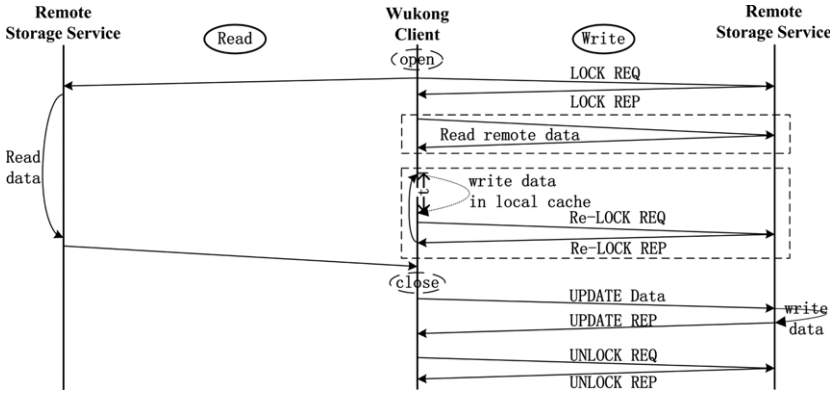

Fig. 3. The interaction of the lease lock file based consistency protocol. The left part shows the process of read operation while the right is for write. Cache validation is implemented in the open phase.

3.1. Lease lock file based consistency protocol

Consistency protocol is an important part of a file system with a cache. File systems such as Coda [24], Ceph [42], GFS [15], and PNUTS [11] use their designed consistency model to ensure the correctness of the system. However, Wukong is different from these file systems for the reason that the servers of these systems have the direct ability to support their consistency protocol, while the service used by Wukong can only do the *put*, *get*, *query*, and *delete* operations. So we have designed a relaxed consistency protocol with the name lease lock file based consistency protocol. Relaxed consistency has been researched for a long period in DSM systems; see, for example, [2,23,34].

With the principle of improving the performance of common operational functions, and ensuring all operations with correctness, we simplify the design. We use a relaxed lock, lease lock file based consistency protocol, to make sure that the cache system works correctly. Fig. 3 shows the details of the interactions of our protocol. The basic idea of the proposed approach 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 the lease time on the backend service.

Each time we request the *open* operation for *write*, it will 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, will the write operation continue. This will first 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. However, the *write* operation needs to periodically re-lock the file as the lease file may expire. The lease time t is configurable. On the other hand, if the system has failed to get the lock, it will stop its following actions. Algorithms 1 and 2 show the lock and unlock processes of our lease lock file based consistency protocol. When *write* is done, Wukong will close the file and update the modified data onto the service. The modified time of this file will be recorded in the remote service by the *close* operation.

In the *read* operation, Wukong tracks two timestamps for each entry in the cache: the last validated time T_c and the last modified time T_m. T_c records the last checking time in this client, while T_m represents the time when the file is modified by the clients. So T_m is gotten from the remote service every checking time, and it changes when there is any committed modification to the file by another client. With these two timestamps, a cache entry is valid at time T if the interval between T and T_c is less than the configured freshness interval t_i, or that the T_m recorded at the client is the same as the T_{m1} gotten from the service at the checking time. The cache entry is validated at the *open* phase. If a cache entry is invalid, we should trash it in order to keep the system in approximate consistency. As the client and server side timestamps do not need to be compared with each other, although the clients and the remote service share different clocks, the lease lock file based protocol can work correctly. The reason is that the comparison between timestamps only happens between the client timestamps or between the server timestamps. For example, when checking if $T - T_c < t_i$, both T and T_c are the timestamps from the client, and they share the same clock, while when checking if $T_m = T_{m1}$, both T_m and T_{m1} are the timestamps generated by the remote service, and they also share the same clock. So although the client and the server do not share the same clock, our protocol can work.

For Wukong, we need to consider the selection of the interval time t_i. Actually, the selection of the interval time t_i is a compromise between consistency and efficiency. Wukong achieves a close approximation to one copy with a very short interval t_i at the sacrifice of the performance. In contrast, the performance of Wukong is higher when the interval t_i is longer, but the consistency becomes weaker. It is therefore a tradeoff to select the interval t_i between the consistency and the performance. If the user does not often access data in conflict ways, a relative longer interval t_i is preferred. For example, if the user is sure that no conflict will occur, such as the situation in which only one client is active, the user can disable the lock to achieve a relatively high performance. Otherwise, if the user wants to achieve a better consistency, the interval should be set to small.

Algorithm 1 Lock algorithm

```
Require: Convert path to Tag first by Tag=md5sum(path)
Require: LOCK_PREFIX is set, /.lock/ by default
Ensure: Tag is unique
lockpath ← LOCK_PREFIX + Tag
lockexists ← child.getattr(lockpath)
if lockexists then
  raise OSError
else
  // create a lock file in the remote service
  child.create(lockfile, 0644, 'f')
end if
```

Algorithm 2 Unlock algorithm

```
Require: Get the lockpath used by lock first
child.trash(lockpath, 'f')
```

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

3.2. File type based adaptive compression

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

On the 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 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 ratios, which relate to the content and type. In the common case, text files always have a higher compression ratio, so we will experience more benefits for the file service if we compress data before transfer. For binary files, the compression ratio always remains extremely low, and compressing always costs a lot of computational resource; we should not consider this deficient option for our system when we compress the data.

In this system, we use the file type specific adaptive compression method, which decides to use compression based on the file type adaptively. Algorithm 3 shows the adaptive compression algorithm. The file types which will be compressed are specified by the configuration. These files always have extensions such as*.log*,*.txt*,*.tex*,*.c*,*.py*, which are always text files. Wukong checks the file types when the user requests to open them, and then we test the file type to see if this file type exists in the compression candidate set. If so, the compression module will record the path information. And in the following steps, it will be necessary to compress for write and to decompress for read.

Algorithm 3 Adaptive compression algorithm

```
comptype ← candidate types in configuration
if typ == 'f' then
  ext ← extension of path
  if ext in comptype then
    add path into candidate set
  end if
end if
```

When Wukong compresses the file because of the match of its type, let us suppose that the size of the original file is S. Meanwhile, suppose that the extra cost size of delivering data to the storage services is 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 $T_{csum} = r * S + D + D_\delta$, where D_δ represents the changed quantity of D which caused by compression which is always small, and r represents the compression ratio, which is always less than 1. We can use compression to reduce the data transfer amount especially for data that has a high compression ratio. Accordingly, the performance of Wukong is improved. In the prototype of Wukong, we can configure the compression method; by default, it uses *gzip*.

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

3.3. Data prefetching

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

Data prefetching happens when Wukong receives an open request. The cache component is checked first to see if we have cached the file. If so, the open operation returns successful immediately; otherwise, Wukong will do a prefetching in the backend. The configuration during the initial mount determines the prefetching size. For example, if the configuration sets the size to 0, this means the system will not prefetch anything, while if the configuration sets it to −1, this means that the system will prefetch the whole file when opened. Algorithm 4 gives the pseudocode of the prefetch algorithm. When the prefetched size in memory comes to a predefined value *MAXLEN*, which is also specified by the configuration, Wukong will store the data in memory to a persistent medium, and then go on prefetching. With the assumption given before, data prefetching can work correctly.

Algorithm 4 Prefetch algorithm

```
rfile ← child.open(path, 'r')
left ← size
while left > 0 do
  this ← min(left, MAXLEN)
  content ← rfile.read(size, offset)
  write content into local cache medium
  left ← left - this
  offset ← offset + this
end while
rfile.close()
change cache status
```

Data prefetching always improves the sequential read operation performance, whereas, in the random schema, this method may increase the performance not as much as that of sequential read. So in our future work, we will pay attention to the data prefetch and also the cache replacement strategy for Wukong.

3.4. Fake service based offline mode

Because nodes may be disconnected for a long period, the system is designed to be runable in an offline mode. As data on the servers may be inaccessible for users in offline mode, only a subset of operations is usable. It is permitted to create new files or delete an accessible file (e.g., a file created in offline mode). But in an offline mode it may be impossible to access data which stays only on servers, so modification to those data is forbidden in this mode. In this paper, we introduce a fake-service based offline mode.

Fig. 4 gives the examples of this offline mode. In (1), the system is online, and the requests reached at SAL objects will be answered regularly by the proper plugin, which interacts with the designated service. The system loses the connection with the service and it is in an offline mode in (2). In this situation, the requests reached at SAL objects cannot be processed immediately by the plugins, so the plugin will return an error code to the SAL object. In order to keep the system working, Wukong creates a fake service for each plugin, which records all the logs, and returns a successful code to the SAL object. In this way, the requests seem to be processed normally, but actually they are not. (3) shows that the system successfully reconnects to the service. When the connection is created, the fake service object will redo the operations recorded when the system is in offline mode.

Wukong silently records all modifications made during the offline period, and propagates them to servers as soon as the system becomes online again. If the file system is modified on multiple clients in no-conflict ways (e.g., creating files with different names or modifying different files) when they are disconnected, Wukong will replay the requests in sequence. But conflicts may occur when operations are made at different nodes

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

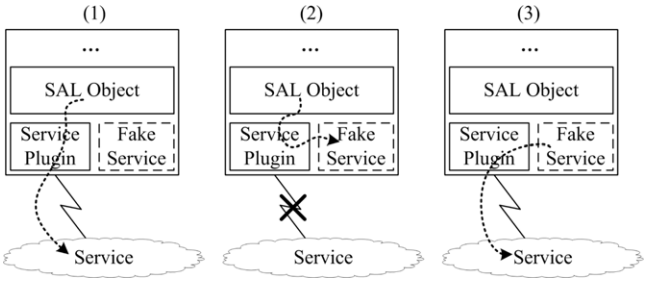

Fig. 4. The offline mode of Wukong.

in offline mode, resulting in inconsistency in namespace. For example, suppose that two clients create two files with the same file name in offline mode, then when they connect to servers, a conflict occurs. A common policy to manage these conflicts is to append each conflicting mapping name with.#*X*, where *X* is the identity of the node that generated the conflicting mapping. In Wukong, conflicts can be solved manually or automatically similarly. For example, suppose that there are two clients *C1* and *C2*, and that both of them created a file named *A* in offline mode. Suppose that *C1* commits to the server first with the file name *A*; when *C2* wants to commit the record, then conflict occurs. Wukong will store the data with file name *A.#C2* by default, and all requests to *A* from *C2* will be redirected to *A.#C2* until the file system is remounted or manually synchronized. The usage of offline mode gives the system high availability.

Besides, extra benefits can be achieved within the offline mode. In the normal mode, Wukong synchronizes the modification as soon as possible when the file is modified and closed. In this way, there is a potential waste of bandwidth as Wukong synchronizes all of the temporal data on the remote service, which may be deleted soon. If the offline mode is activated, Wukong can hide those modifications for the remote service, and minimize the transfer data size. So, offline mode not only ensures the availability, but also reduces the transferred data size and the time of interactions.

3.5. Plugin implementation example

With the use of the SAL and its plugin mechanism, Wukong supports multiple heterogeneous storage services. Wukong can be easily extended 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. In the prototype system, we have implemented several plugins for the services including Amazon S3, Google Docs, Google Picasa, FTP, Email Service, and local disk.

We give a sketch view of the plugin implementation and show a plugin example with Amazon S3 as its backend service to explain the plugin implementation process. Briefly, Amazon Web Services provides Amazon S3 as an online storage web service. It stores arbitrary objects up to 5 gigabytes in size, each accompanied by up to 2 kilobytes of metadata. Those objects owned by Amazon Web Services (AWS) accounts are organized into buckets. Each bucket is identified by a unique, user-assigned key. Buckets and objects can be *created*, *deleted*, *listed*, and *retrieved* using either a REST-style HTTP interface or a SOAP interface. In the REST API of AS3 [5], there are different operations on the service, buckets, and the objects. In these operations, the *PUT Operation* adds an object to a bucket, the *DELETE Object* removes the null version (if there is one) of an object and inserts a delete marker which becomes the latest version of the object, the *GET Bucket (List Objects)* returns some or all (up to 1000) of the objects in a bucket and can be used with the request parameters as selection criteria to return a subset of the objects in a bucket, and the *GET Object* retrieves objects from Amazon S3.

The *put*, *delete*, *query*, and *get* operations of the Wukong plugin are implemented with the REST APIs of Amazon S3. The details of the conversion from the service APIs to the plugin interfaces are shown in Fig. 5. It is quite similar to the plugins for other services. For example, in a plugin for a disk, ***put*** can be implemented with the file operations *open*, *write*, *close*, or *mkdir*, and ***get*** can be implemented with *open*, *read*, and *close*, while ***query*** can be implemented with *Get_Object_List* which uses *readdir* and *stat*, and ***delete*** can be implemented with *rmdir* and *unlink*. Whatever the case, the plugins can be implemented with such a code sketch.

In summary, as the sketch of the plugin shows, what the plugin developer needs to do is just design and fill the detail logic of the service-specified part with the usable interfaces published by the services. So it is easy for the developer to make a new service available for Wukong, and this gives Wukong high extensibility.

3.6. A walk through

This section gives a walk through all these components of Wukong with a scenario in which the user wants to use Amazon S3 as a local file system. Wukong is always used in the steps as follows: *config*, *mount*, *use*, and then *umount*. When a user wants to make Amazon S3 Wukong's backend service, he/she will first specify the configuration of Wukong in a specification file, which tells our service about which plugin should be used. The specification also gives how the service is composed by the components. Fig. 6 gives this scenario, including the configuration, system layout, and all of the other important elements in Wukong.

With the specification set, the user will then mount the remote service to local disk, for example, suppose that the file service is mounted on the directory of `/tmp/wukong`. All of the access to the backend service can be done by accessing this directory. When the *mount* command is called, Wukong parses the parameters of the *mount* command and starts to read the specification file, which we set to be.*wukong.conf*, and then constructs the service according to the user configuration. The plugin is initialized in this phase. The plugin will get the parameters from the user specification file, and do the authentication with the remote service. The authentication of the plugin may also happen when the connection to the remote service is lost and the plugin wants to reconnect to the remote service. After all of these steps are finished correctly, the Wukong service will start serving. With configuring the service properly, users will start to use the remote Amazon S3 service as a local file system, which means that all of the applications can use the files stored in Amazon S3 as they are in local. When Wukong finishes playing this role, it can be stopped by the *umount* operation, which will release all of the resources used.

In the serving time, Wukong serves the applications with the POSIX compliant interfaces through the VFS [7]. When a system call is requested, the VFS layer will first check according to the parameter of the operation. If the accessed file is located in a normal file system, for example the `/home/me/foo` in a local ext3 file system in the directory `/home/me/`, then the VFS will process the file operation request locally by the ext3 file system module in the kernel. Otherwise, if the accessed file is in Wukong file service, the VFS will send the request to FUSE, and then FUSE will forward the request to Wukong, which will call the proper operation of the interface layer. For example, suppose an application needs to read the content of the file `/tmp/wukong/bar`. In this situation, the application will first open this file by calling `f = open('/tmp/wukng/bar', 'r')`, and then read the content. In the open phase, Wukong may prefetch some content from the remote service. After reading the content, the application will close this file by calling `close(f)`. In this process, all the operations will be forwarded to Wukong in the step (1) as shown in Fig. 6, and will be handled layer by

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

```
def init(self, path):
   * do init *

def put(self, path, content):
   key = path2key(path)
   put content to service with key

def delete(self, path, typ='f'):
   key = path2key(path, typ)
   delete object with key in remote service

def Get_Object_List(path):
   if now - last_updtime > interval:
      entries = as3.list_bucket
      foreach entry in entries:
         record it in cache
   last_updtime = now
```

```
def get(self, path):
   key = path2key(path)
   get content from service with key
   return content

def query(self, path, typ):
   Get_Object_List(path)
   if typ == 'meta':
      if Cache has the entry for path:
         return the stat tuple of the entry
      else: return None
   elif typ == 'list':
      dlist = ['.', '..']
      if self._dirmd5.has_key(mymd5):
         foreach file in directory path: dlist.append(file)
      return dlist
   elif typ == 'chk':
      if Cache has the entry for path: return True
      else: return False
```

Fig. 5. Plugin implementation example with AS3 as Wukong's backend service. Get_Object_List is implemented in this version with the *GET Bucket (List Objects)* of AS3 REST API. The authentication may happen in the plugin initialization phase and the time when the plugin detects the connection failure and tries to reconnect.

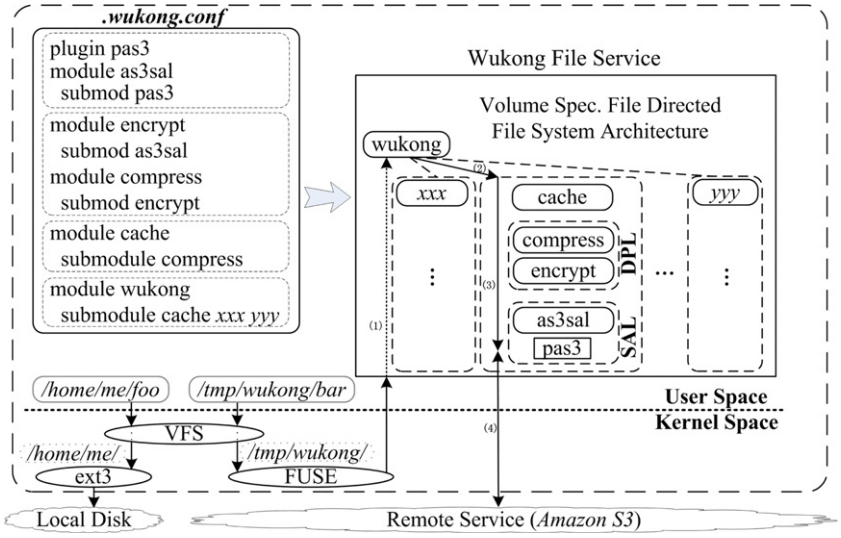

Fig. 6. The steps of Wukong walking through all these components with a scenario in which Amazon S3 is used as the remote service.

layer in the function stack as steps (2) and (3). When this process reaches the bottom of the function stack, the plugin will be in a position to communicate with Amazon S3 in step (4). When the file is opened and the read operation is executed, Wukong starts reading the content from cache or from the remote service with the *pas3* plugin. The *write* operation of Wukong acts in a similar way as *read* except as follows. First, in the *open* operation for the write purpose, Wukong needs to do the consistency checking by the lock file based consistency protocol. Second, the *write* operation in Wukong caches all the changes in the local medium, so in the *close* operation, Wukong should update the modifications into the remote service; otherwise, all the modifications will be lost.

4. Performance evaluation

This section evaluates Wukong with several experiments. Fig. 7 shows the experimental platform, where two kinds of client host, laptop and MID, use two types of network connection, ADSL and 3G (EVDO), provided by China TeleCom, to connect to services, including Google Docs, Amazon S3, Google Picasa amongst others. Fig. 7 also shows the hardware and software configurations of the clients. In the testing, we intentionally choose a rural site, which means that the 3G network is not very stable and may downgrade occasionally. Although the ideal bandwidth is 3.1 Mbps, the real bandwidth is always much smaller than this value.

Table 2
Files of workload for evaluating Wukong service.

File type	File number	Total size	Source
Binary	9	3.3 MB	/usr/bin
Doc	6	108 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

We synthesize the workload with files of different types: binary files (binary), word documents (doc), images (image), system log (log), printable document files (pdf), music (music). The input to the operations consists of 43 files, about 43 Mbytes in total, and the details of the workload files are shown in Table 2.

With this workload, we evaluate Wukong as follows. First, we examine the function of Wukong with applications, and state the convenience and flexibility of the Wukong plugin with the

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

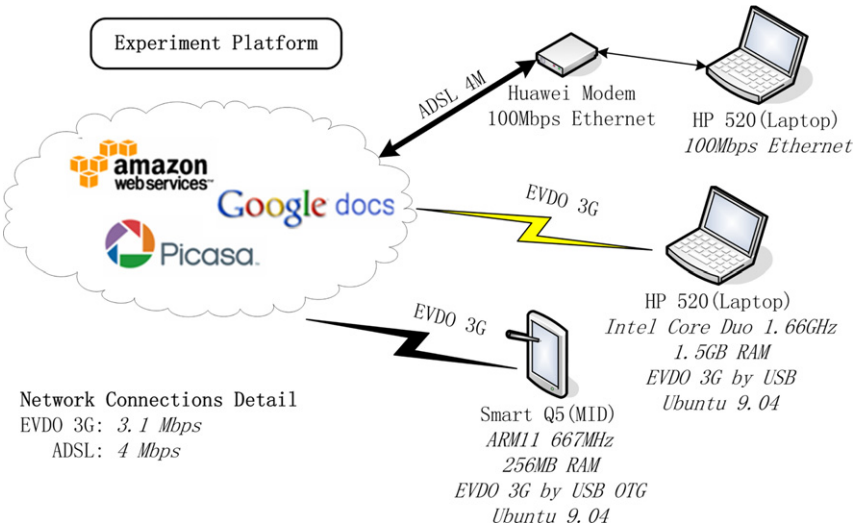

Fig. 7. Configuration of the experimental platform. All the client hosts are deployed with Wukong.

statistics of the numbers of lines of code (LoCs) 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 previously under the environment shown in Fig. 7.

4.1. Functions

In this section, we present two experiments focusing on checking the service interfaces. First, we use several benchmarks, which call common file system interfaces, including *open*, *creat*, *close*, *read*, *write*, *lseek*, *mkdir*, *unlink*, and so on, in 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 such as *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 chose several typical upper layer applications to check if they can get along well with Wukong. We use 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. LoCs of Wukong plugins

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

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

Table 3
Number of LoCs of the Wukong plugins.

Plugin	pgdocs	pgpic	pas3	pftp	pimap4	pdisk
LoCs	77	95	108	63	137	44

As the result shows, the plugin mechanism brings Wukong several advantages. First, the plugin is separated from the overall architecture, and what the plugin developer cares about is the logic of the plugin while avoiding the complex management logic of the file system. Second, only a small amount of work, such as the numbers of LoCs of the plugins shown in Table 3, needs to be done for a new service support. Besides, the plugin mechanism gives Wukong high function extensibility.

4.3. Evaluation on cache

We present two experiments based on the Amazon S3 service to see how the cache layer affects Wukong. We run several file system related commands, such as cp, ls, and rm, in the deployed Wukong without any other optimization and record the latencies of those commands. The *cp* commands copy files shown in Table 2 to/from the remote Amazon S3 service. We execute all of 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 commands in the condition of no cache. Fig. 8 shows the comparison of latencies of different commands.

From the result shown in this figure, we can find that the latencies of no cache are always smaller than those of a cool cache. The reason is that the cache manager introduces an overhead not only to store the data in the cache but also to maintain the cache consistency. But the 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 costs only a little to finish most commands which are far smaller than those of the cool cache. However, for *'cp all to remote'* and *'rm'*, the warm cache costs a little more. This is because the cache manager cannot hide the network interaction for these commands as they have to write or delete data on the remote service.

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

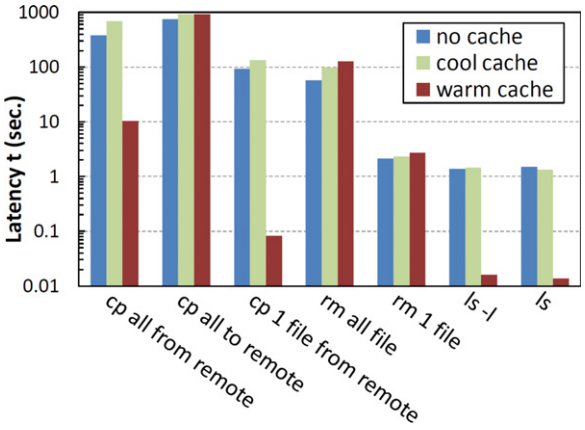

Fig. 8. Comparison of latencies of different commands.

Table 4
Comparison 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

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 it takes to read the whole file. We present here the second experiment evaluating the prefetch of Wukong.

In this experiment, we first open several files in the deployed Wukong, then wait for an interval whose values are chosen to be 0, 1, 2, 4, 8, 16, 32 and 64 s, 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 shows, the latencies to read the whole files are 55.7 s for the music file, 11.9 s for the binary file, and 11.4 s for the log file. Besides, we find that the sum of the interval time and the latency of the read operation almost remain the same for all these three files. This means that 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 costs 1.43 s for the music file, 1.26 s for the binary file, and 1.25 s for the log file to read the content from the cache. With this characteristic, Wukong will be good in those situations where there is a large time gap between the open and read operations.

4.4. Compression

First, we evaluate the efficiency of our file type based adaptive compress with the files in Table 2. In this experiment, we choose those with extensions such as 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, Table 4 contains the values of none and full compression strategies as well. The unit of the values is *MBytes*.

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 Wukong spends on compressing. Fig. 9 shows the overhead of file type based adaptive compression compared with the none and full ones.

As shown in Table 4, we find that adaptive compression costs time, but it keeps the consumption as small as possible. We find

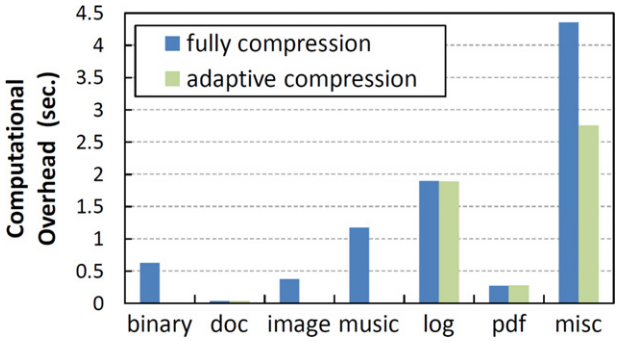

Fig. 9. Computation time of the compression overhead.

Table 5
Latencies of copying 1.5 GB files locally with and without Wukong.

	With Wukong	Without Wukong
Laptop	145.4 s	102.9 s
MID	2259.6 s	993.0 s

that the above two experiment results confirm the benefits of the file type based compression method, not only that the time consumption costs of adaptive compression are similar to those for the method without any compression, but also that the size reduction by adaptive compression remains only a little smaller than that of full compression. However, as shown in Table 4, 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 certainly choose a good candidate set for file type based adaptive compression layer, so that the adaptive compression method can reduce the transfer size, while keeping the overhead minimum.

4.5. Wukong overhead analysis

In this section, we evaluate the overhead of using Wukong in a local copy. We compare the latencies of the situations copying files both with and without Wukong locally. In both these two situations, the ext3 file system is used as the backend file system.

As the memory size of the laptop is 1.5 GB, we use the file set by multiplying the files 36 times as Table 2. So the data set in this experiment is about $43\ \text{MB} * 36 = 1548\ \text{MB}$, which is a little bigger than the laptop's memory size.

Table 5 shows the result of this experiment. As shown in this table, the latency and the resource cost is a little higher in the environment with Wukong than without it. In the laptop, the latency with Wukong is about 1.4 times that without Wukong, while in MID this value comes to 2.3 times. This is mainly for the following reasons. First, the disk plugin of Wukong uses ext3 as its storage backend. So the time cost of the plugin would be almost the same as that of the copy time in the no-Wukong situation. Second, Wukong introduces several layers, which also increases the latencies. Third, some strategies, such as compression and some other ones, are designed for low bandwidth situations. In a local copy, these may slow down the copying operations. Adding these overheads up, the additional latency of Wukong would come to 0–2 times that without Wukong. However, as the bandwidth to the Internet is always far smaller than that of the local disk, the additional latency will become negligible when the backend service is changed from a local disk to some service on the Internet. For example, in our experiments, the total overhead caused by Wukong in the laptop is about 42 s. It costs about $42\ \text{s}/36 \approx 1.2$ s for each 43 MB dataset. This is a negligible overhead.

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

Table 6
Throughput and average resource consumption of Wukong.

Environment	Copy from remote service			Copy to remote service		
	Laptop ADSL	*Laptop 3G*	*MID 3G*	*Laptop ADSL*	*Laptop 3G*	*MID 3G*
Throughput (kB/s)	179.11	48.93	80.68	66.82	16.92	17.99
Average CPU (%)	0.38	0.01	3.5	0.24	0.036	2.06
Average memory (kB)	6761	5412	6712	12700	6711	11066

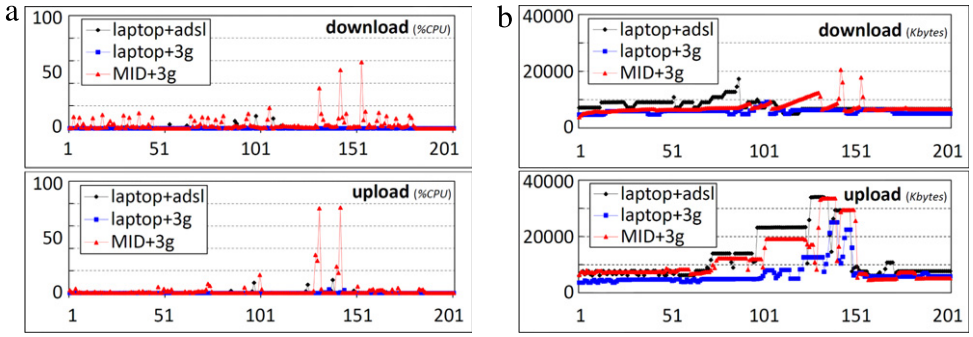

Fig. 10. Computation time of the compression overhead.

4.6. Throughput and resource cost

In this section, we will evaluate the throughput and the system resource (cpu and memory) consumption in the environment shown in Fig. 7.

Wukong easily transfers files from one remote service to another; it copies files one by one from one remote service to local disk and then copies them to the other service. In order to see this process clearly, we perform the experiments by evaluating the processes of copying files from a remote service to local disk (download) and copying files to a remote service from local disk (upload) separately. The system resource consumption is captured by the Linux command *top*. We averagely sampled 200 points for each process of the experiments in three environments *laptop + adsl*, *laptop + 3g*, and *MID + 3g*. Both the CPU and memory consumption information are recorded.

We evaluate the throughput and resource consumption with the files shown in Table 2. Table 6 shows the throughput and the average consumption in our experiment platform. From the result, Wukong achieves an acceptable throughput of 179.11 kB/s (35% of the ideal bandwidth) in an ADSL environment and 80.68 kB/s (20% of the ideal bandwidth) under a countryside EVDO 3G network. As the upload link bandwidth is much smaller than the download link bandwidth, the performance of copying to a remote service is smaller compared to the performance of download. The reason why the throughput of Laptop + 3G and MID + 3G is only about 20% of the ideal bandwidth is that our 3G network is a countryside EVDO, and the signal is always weak and unstable. Meanwhile, as the total transfer size stays the same, as the throughput increases, the system takes shorter amounts of time to transfer the data.

In addition, we monitored the system resource consumption at the same time as recording the throughput. Fig. 10(a) shows the CPU ratio that Wukong uses at the sampled timestamps. The memory consumption is shown in Fig. 10(b).

As Fig. 10(a) shows, the CPU consumption of MID+3G is always higher than those of the other two. This is mainly because the CPU frequency of MID is 667 MHz, while that of the laptop is 2.33 GHz. Although the tasks are the same and they do the same computation for compression and other actions, the MID needs a relative higher CPU consumption ratio than that of the laptop. However, as the average value of CPU ratio in Table 6 shows, the average consumption of the MID is only about 3.5% for download and 2.06% for upload. This resource consumption is negligible.

In Fig. 10(a) and (b), there are several peak points whose values are much larger than the other points. In order to see what happens with these timestamps, we look into the details of the data transfer process, and we find that the peak points of the CPU ratio reside in the times when the system needs a lot of compression, and those of the memory consumption reside in the times when the system is processing the big files.

From the results of our experiments, we can find that the prototype system has acceptable performance and resource consumption. It is worth noting that Wukong improves the usability of cloud services significantly, although a (negligible) overhead is introduced.

4.7. Performance comparison

In this section, we evaluate Wukong by comparing with S3FS, which is also a user space file system. S3FS is used to treat the remote Amazon S3 service as a local resource and operate on the files in AS3 as local files. We also compare Wukong and S3FS with a method without Wukong, namely a method which downloads/uploads files from/to the Amazon S3 service with the http connection using the python library offered by Amazon. First, we compare them by putting files from a client to the Amazon S3 service, copying files from local disk to remote service both with Wukong and S3FS, and also directly uploading the files with the AS3 uploading APIs. Second, we evaluate the performance of getting files from the Amazon S3 service to local disk by copying files both with Wukong and S3FS, and also downloading the files with the API that Amazon S3 provides. To fairly compare the performance, we do not use the strategies such as compression and encryption in these experiments.

The experiments were done with the same dataset shown in Table 2. To demonstrate the adaptivity under different network environments, we conducted our experiments in two different network environments: ADSL with 2 Mbps and the EVDO 3G connection. We found that the performance under MID and Laptop share a similar performance trend. Thus we mainly focus our experiments on the environment of a laptop under 3G and ADSL connections.

Table 7 shows the results of the comparison between Wukong, S3FS, and without Wukong. As the results show, the direct download and upload throughput are about 105.54 and 59.35 kB/s

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

Table 7
Comparison between Wukong, S3FS, and without Wukong (w/o for short).

	ADSL 2M			EVDO 3G		
	Wukong	S3FS	w/o	Wukong	S3FS	w/o
Up (kB/s)	51.81	39.88	59.35	27.53	37.63	42.90
Down (kB/s)	102.04	85.29	105.54	67.57	51.28	75.09

under the ADSL, and about 75.09 and 42.90 kB/s under the EVDO 3G environment. They are smaller than the ideal download and upload bandwidth due to the disturbing of the environment. Also, as shown in the table, the download throughput of Wukong under the 2 Mbps ADSL is about 102.04 kB/s, and it comes to about 67.57 and 27.53 kB/s under the EVDO 3G environment. The download throughput of S3FS is about 85.29 kB/s and the upload throughput is about 39.88 kB/s under the ADSL environment, and comes to about 51.28 kB/s under the EVDO 3G.

From the results, we can see that Wukong achieves a performance of near the direct download and upload speeds. And compared to the S3FS, Wukong outperforms it by about 20% for downloading, and by about 10% for uploading under the ADSL environment. By comparison, we found that Wukong has several advantages over S3FS. First it has a better performance than S3FS in terms of throughput. Second, it provides more interesting functions, such as service integration, participatory sharing, and heterogeneous service support. Compared to the without-Wukong method, we can find that the overhead of Wukong is proved to be negligible.

5. Potential Wukong based applications

With the SAL's mashup, Wukong can be used as a platform to develop different applications for multiple purposes. In this section, we present two potential Wukong based applications, *service integration*, and *participatory sharing*, to show the potentials of Wukong.

Service integration. Users always use multiple types of services in order to satisfy their common requirements. They may use Google Docs to backup the documents and slides, use Picasa to save the pictures, and use Amazon S3 to store some files. A system that supports only one special service would not enable our users to access resources conveniently. Since different service providers always offer different public APIs, users have to pay attention to the access method of the services.

Wukong can be used to integrate different services, mash them up, and present a uniform interface with high usability to the users who operate on these services. To implement such a service, first, users should assign the plugins for the services on the deployed device, so that, the application can interact with the special service fluently. We have implemented several plugins in our prototype system, e.g., Google Docs, Picasa, and Amazon S3. Then, Wukong use its configuration file to set the map between special service and its plugin. After the mapping is assigned, Wukong can present a uniform, POSIX compliant file system interface for the integrated services to the users.

Participatory sharing. Social networking has gradually outpaced emails as the major communication tool for human beings, and its use is still growing [44,25]. Services such as Facebook, Twitter, and Google Plus, are increasingly popular. There is more than one place for Internet users to post their current status, to express their feeling, and so on. They always have to manage several clients to make all the friends both on Facebook and Twitter be notified, but not only their friends on Facebook or only those following on Twitter. Besides, in some situations, it is not very easy for the social network user to describe what happened: words cannot express the surrounding feelings. But if we can record the voice or sound at that moment, and picture the situation, more information is shared than if only words are used. For example, when we participate in a party, we usually do not have much time to type the words to express our happiness, but if we can record the sounds and picture the scenario, it will not cost much time, and much more information is included. Nowadays, cameras and voice recorders are de facto part of the configuration of many smartphones, so we can use the camera application and the voice recorder on the device to snapshot the exciting moment, and save them in Wukong; then Wukong uses a plugin to publish these multimedia files onto social networks (e.g., share a short URL on Twitter, a thumb on Facebook, with a voice control icon to play the sound recorded).

For such application, the social network service API, e.g., Facebook API, Twitter API, is needed for implementing the plugin. As described in Section 3.5, the *put*, *delete*, *query*, and *get* operations are recommended to be implemented. For example, when a user wants to post a new sharing, Wukong uses *put* to interact with the social network service, and posts content on the service. After the plugins are implemented, Wukong knows how to communicate with the social network service. Then the following work is to tell Wukong which directory should be managed. As soon as the user creates a file in the managed directory, Wukong automatically posts the content to the remote service. This can be done by mounting the Wukong service at a local directory. When the local directory is assigned, it becomes the Wukong managed directory. After all of these are done, what the user needs to do is just taking a snapshot of the moment, and stores them in the Wukong managed directory. In this way, Wukong calls the proper plugin, and the plugin will automatically post a new thread on the social network service according to the content of the file in the managed directory.

6. Related work

Wukong shares its goals with several recent efforts aimed at simplifying the data management for cloud service [1]. 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 straightforward and common way for data management on the services, since most of the clients can be easily obtained from the service providers. Dropbox [12] 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 [38] is a similar service provided by Syncplicity Inc. Besides, Google gives Internet users the ability to store any type of data on their Google Docs service, and Memeo publishes Memeo Connect, which is a client to manage the Google Docs. Also, there are several cloud service platforms which are very usable and well known. MobileMe [28] 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 other aspects ubiquitously. Similar to MobileMe, Live Mesh [27], 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 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 users, especially for those whose devices are poor in terms of user interface.

Network and distributed file systems. These have been extensively studied in the past. The Andrew file system (AFS) [19] is

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

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 [33] descends from AFS. It has many great features. It keeps working even when the network disconnects or has a weak connection [20]. It has high performance through client-side persistent caching, and so on. LBFS [29] 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 [35] proposes building 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 networks to low bandwidth dial-up connections. It provides lots great ideas on the key techniques. Several other works in this area, such as Zebra [17], GmailFS [16], Cumulus [41], and S3FS, cover similar regions. Although these works on diverse aspects, including network challenges, cache management, offline operations, and so on, have proposed and studied the system performance, usability and availability, most of the file systems are based on a special server. For example, Coda needs the support of Coda Server, and S3FS or Cumulus need the Amazon S3 service, but if 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 support the new service.

7. Conclusions

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

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

With these features, we envision that Wukong will definitely facilitate the wide deployment of services on mobile devices. It is worth noting that Wukong is just an initial step; several interesting directions might be pursued for cloud-oriented file services on mobile devices: first, enhancing the storage abstraction layer to support more rich services; second, reducing the extra overhead, e.g., latency and throughput; finally, detailed workload characterization and analysis of mobile cloud services.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments and kindly suggestions. This work is supported by the National Natural Science Foundation of China under Grant No. 60736013, Grant No. 61025009, and Grant No. 60903040.

References

[1] D. Abadi, Data management in the cloud: limitations and opportunities, Data Engineering (2009) 3.
[2] S. Adve, K. Gharachorloo, Shared memory consistency models: a tutorial, Computer 29 (12) (1996) 66–76.
[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., Above the clouds: a Berkeley view of cloud computing, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28.
[4] Amazon S3, http://status.aws.amazon.com/s3-20080720.html.
[5] Using the REST API, http://docs.amazonwebservices.com/AmazonS3/latest/index.html?RESTAPI.html.
[6] M. Blaze, A cryptographic file system for Unix.
[7] D. Bovet, M. Cesati, A. Oram, Understanding the Linux Kernel, O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2002.
[8] D. Boyd, N. Ellison, Social network sites: definition, history, and scholarship, Journal of Computer Mediated Communication—Electronic Edition 13 (1) (2007) 210.
[9] Q. Brown, F. Lee, D. Salvucci, V. Aleven, Interface challenges for mobile tutoring systems, in: Intelligent Tutoring Systems, Springer, pp. 693–695.
[10] D. Chappell, A short introduction to cloud platforms, Microsoft.
[11] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, R. Yerneni, PNUTS: Yahoo!'s hosted data serving platform, Proceedings of the VLDB Endowment 1 (2) (2008) 1277–1288.
[12] Dropbox, http://www.dropbox.com.
[13] P. Eaton, S. Weis, Examining the security of a file system interface to oceanstore.
[14] Google Docs, http://docs.google.com.
[15] S. Ghemawat, H. Gobioff, S. Leung, The Google file system, ACM SIGOPS Operating Systems Review 37 (5) (2003) 43.
[16] GmailFS: GMail virtual file system, http://richard.jones.name/googlehacks/gmail-filesystem/gmail-filesystem.html.
[17] J. Hartman, J. Ousterhout, The Zebra striped network file system, ACM Transactions on Computer Systems (TOCS) 13 (3) (1995) 274–310.
[18] J. Heiser, M. Nicolett, Assessing the security risks of cloud computing, Gartner Report.
[19] J. Howard, et al., An overview of the andrew file system, in: Proceedings of the USENIX Winter Technical Conference, Citeseer, 1988, pp. 23–26.
[20] L. Huston, P. Honeyman, Disconnected operation for AFS, in: Mobile & Location-Independent Computing Symposium on Mobile & Location-Independent Computing Symposium, USENIX Association, 1993, p. 1.
[21] T. Hu, B. Thai, A. Seneviratne, Supporting mobile devices in Gnutella file sharing network with mobile agents, in: ISCC03: Proceedings of the Eighth IEEE International Symposium on Computers and Communications, Citeseer, p. 1035.
[22] M. Jensen, J. Schwenk, N. Gruschka, L. Iacono, On technical security issues in cloud computing, in: Proceedings of the 2009 IEEE International Conference on Cloud Computing, IEEE Computer Society, 2009, pp. 109–116.
[23] P. Keleher, A. Cox, W. Zwaenepoel, Lazy release consistency for software distributed shared memory, in: Proceedings of the 19th Annual International Symposium on Computer Architecture, ACM, 1992, pp. 13–21.
[24] J. Kistler, M. Satyanarayanan, Disconnected operation in the Coda file system, ACM Transactions on Computer Systems (TOCS) 10 (1) (1992) 3–25.
[25] H. Kwak, C. Lee, H. Park, S. Moon, What is twitter, a social network or a news media? in: Proceedings of the 19th International Conference on World Wide Web, WWW'10, ACM, 2010, pp. 591–600.
[26] J. Landay, T. Kaufmann, User interface issues in mobile computing, in: Workstation Operating Systems, 1993. Proceedings, Fourth Workshop on, 1993, pp. 40–47.
[27] Live Mesh, http://en.wikipedia.org/wiki/Live_Mesh.
[28] MobileMe, http://en.wikipedia.org/wiki/MobileMe.
[29] A. Muthitacharoen, B. Chen, D. Mazieres, A low-bandwidth network file system, in: Proceedings of the eighteenth ACM symposium on Operating systems principles, ACM, 2001, pp. 174–187.
[30] A. Pasick, File-sharing network thrives beneath the radar, London Reuters.
[31] G. Perrucci, F. Fitzek, G. Sasso, W. Kellerer, J. Widmer, On the impact of 2G and 3G network usage for mobile phones battery life, European Wireless, 2009.
[32] T. Raman, Cloud computing and equal access for all, in: Proceedings of the 2008 International Cross-disciplinary Conference on Web Accessibility (W4A), ACM, 2008, pp. 1–4.
[33] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, D. Steere, Coda: a highly available file system for a distributed workstation environment, IEEE Transactions on Computers 39 (4) (1990) 447–459.
[34] W. Shi, Performance optimization of software distributed shared memory systems.
[35] W. Shi, H. Lufei, S. Santhosh, Cegor: an adaptive, distributed file system for heterogeneous network environments, in: Proceedings of the tenth International Conferences on Parallel and Distributed Systems.
[36] Smart Q5, http://www.smartdevices.com.cn/index.html.
[37] L. Sotto, B. Treacy, M. McLellan, Privacy and data security risks in cloud computing, Electronic Commerce & Law Report.
[38] Syncplicity, http://syncplicity.com.
[39] M. Szeredi, et al. FUSE: Filesystem in userspace, Accessed on.
[40] W. Tan, F. Lam, W. Lau, An empirical study on 3G network capacity and performance, in: IEEE INFOCOM 2007. 26th IEEE International Conference on Computer Communications, 2007, pp. 1514–1522.
[41] M. Vrable, S. Savage, G. Voelker, Cumulus: filesystem backup to the cloud, ACM Transactions on Storage (TOS) 5 (4) (2009) 1–28.
[42] S. Weil, S. Brandt, E. Miller, D. Long, C. Maltzahn, Ceph: a scalable, high-performance distributed file system, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), 2006.
[43] K. Wu, H. Tan, Y. Liu, J. Zhang, Q. Zhang, L. Ni, Side channel: bits over interference, in: Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking, ACM, 2010, pp. 13–24.
[44] M. Zuckerberg, 500 Million stories, http://blog.facebook.com/blog.php?post=409753352130.

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017

Huajian Mao is currently a Ph.D. student at the School of Computer Science and Technology of the National University of Defense Technology. His current research focuses on distributed file systems, and mobile and cloud storage. He received his B.S. degree in 2007 from Zhejiang University and his M.S. degree in 2010 from the National University of Defense Technology in China, both in Computer Science.

Nong Xiao is a Professor of Computer Science at the National University of Defense Technology. He received his B.S. and Ph.D. degrees from the National University of Defense Technology. His recent research focuses on grid and cloud computing, storage systems, and architecture. He has chaired several conferences and workshops, and served on technical program committees of numerous international conferences.

Weisong Shi is an Associate Professor of Computer Science at Wayne State University. He received his B.E. from Xidian University in 1995, and his Ph.D. degree from the Chinese Academy of Sciences in 2000, both in Computer Engineering. He has authored two books, and published over 100 publications, cited more than 1000 times. He is currently serving on the editorial board of the Journal of Computer Science and Technology (JCST) and International Journal of Sensor Networks. He has served as a guest co-editor of several top journals, including IEEE Internet Computing Magazine and Journal of Parallel and Distributed Computing.

Yutong Lu is a Professor of Computer Science at the National University of Defense Technology. She received her B.S. and Ph.D. degrees from the National University of Defense Technology. Her research focuses on high-performance computing, parallel file systems, and parallel system software.

Please cite this article in press as: H. Mao, et al., Wukong: A cloud-oriented file service for mobile Internet devices, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.10.017