vUPS: Virtually Unifying Personal Storage for Fast and Pervasive Data Accesses

Mohammed A. Hassan, Kshitiz Bhattarai, and Songqing Chen

Department of Computer Science,
George Mason University
{mhassanb,kbhattar,sqchen}@gmu.edu

Abstract. More and more overlapping functions on all kinds of mobile devices with these on traditional computers have significantly expanded the usage of mobile devices in our daily life. This also causes the demand surge of pervasively and quickly accessing files across different personal devices owned by a user. Most existing solutions, such as DropBox and SkyDrive, rely on some centralized infrastructure (e.g., cloud storage) to synchronize files across different devices. Therefore, these solutions come with potential risks of user privacy and data secrecy. In addition, continuously maintaining strong consistency among multiple replicas of a file is very costly.

On the other hand, today a common user often owns sufficiently large storage space across her personal home desktop, office computer, and mobile devices. Therefore, in this paper, we aim to design and implement a system to *v*irtually *U*nify *P*ersonal *S*torage (vUPS) for fast and pervasive accesses of personal data across different devices. vUPS provides similar services as offered by existing cloud-based storage services, but (1) vUPS consists of only personal computers without involving any third party, thus it minimizes the risks of user privacy and data secrecy; (2) vUPS organizes all storage in a distributed fashion so that it is not prone to the single point of failure; (3) vUPS differentiates files and maintains different consistency policies in order to reduce the consistency maintenance cost. Having implemented vUPS with HTML5, we conduct extensive experiments to evaluate its performance. The results show that vUPS offers similar user performance when compared to DropBox.

1 Introduction

With the ever-increasing processing power and ever-decreasing prices, mobile devices are getting more and more popular. According to International Data Corporation, the total number of smartphones sold in 2010 was 305 millions [11], which is a 76% increase from 2009, and there are already over 4.6 billion mobile subscribers in the world and the number is still growing [12]. The prediction is that there will be around 982 million smartphones in 2015 [11].

To some extent, today mobile devices are replacing their counterparts for Internet accesses, such as emailing, web surfing, and Internet entertainment. For

example, mobile video traffic is now over 50% of the mobile data traffic on the Internet, and it is predicted to occupy 2/3 by 2015 according to Cisco [3].

Such a trend has also caused the demand surge of pervasive data accesses, such as for photos, across a user's different storage space, such as her iPhone and desktop computer. To respond to such a fast increasing demand, the current research and practice have provided many solutions, such as `Dropbox` [5], `Google Docs` [8], `Amazon s3` [2], `Windows SkyDrive` [14] and `SME Storage` [15]. They mainly rely on cloud-based services or a server-based approach.

These centralized cloud or server-based approaches [2] [5] [14] typically require a user to store all files on the storage owned by the service providers, which risks data security, privacy, and availability. For example, it has been reported on the news about Mark Zuckerberg's pictures leak incident in Facebook [13], DropBox account breach with wrong passwords [6], Amazon's data center failure in 2011 [1], etc. Some modern file systems [15] [32] [41] have taken into account user's storage to avoid third party storage compromise. But they maintain a strong consistency model for different types of files, resulting in unnecessary and heavy performance overhead. For example, smartphones are often associated with consuming and producing data which are mostly non-editable (photo, music, and video) and more than 27% pictures are taken by smartphones [4]. For these files, a strong consistency model is an overkill.

On the other hand, today an average user typically possesses multiple computers, such as personal home computers, office computers, and mobile devices. While the storage of one device at a time may not be sufficient for storing all data files of the user, the total storage space is often large enough to store all files owned by the user. Even when the total storage space of a user is not large enough, it is relatively cheap to buy additional storage nowadays as one-time cost.

Therefore, in this study, we aim to design and implement a system to *v*irtually *U*nify *P*ersonal *S*torage (vUPS) for fast and pervasive file accesses. With vUPS, all participating devices of the user are transparently and seamlessly unified. A file can be accessed through a web browser, considering that today the web browser is the vehicle for end-user's accesses. In vUPS, there is no central node that maintains the file system states. Instead, any device can serve as the server when it is actively used. To minimize data transferring, only meta-data is proactively accessed while data files are accessed in an on-demand fashion. In vUPS, different types of files are treated with different consistency policies in order to balance the consistency and the maintenance overhead.

To evaluate vUPS, we have implemented a prototype based on HTML5 and Javascript. The prototype provides a standard API for other web and desktop applications to access and manipulate vUPS data. We have conducted experiments with micro-benchmarks and also compared to DropBox. The results show that vUPS can offer a similar user experience to DropBox.

In summary, we have made the following contributions in this paper.

– vUPS provides an alternative solution to existing cloud-based or other centralized approaches for responding to the demand surge of quick and pervasive file accesses across multiple devices owned by a user.
– By differentiating and treating different types of files, vUPS strives to achieve a balance between the file consistency and the maintenance overhead.
– With a web browser interface and a standard file access interface, vUPS can be adopted by other applications to transparently and seamlessly access personal files.

The rest of the paper is organized as follows. We discuss the design of vUPS in section 2 and the consistency models in section 3. vUPS implementation is presented in section 4, and we evaluate its performance in section 5. Some related work is discussed in section 6 and we make concluding remarks in section 7.

2 vUPS Design

In this section, we first briefly overview the architecture of vUPS. Then we discuss its components.

2.1 vUPS Overview

Instead of a centralized architecture, vUPS adapts a flexible P2P architecture. Figure 1 illustrates the architecture of vUPS that runs across multiple devices, such as a user's home computer, an office computer, a laptop, and a smartphone.

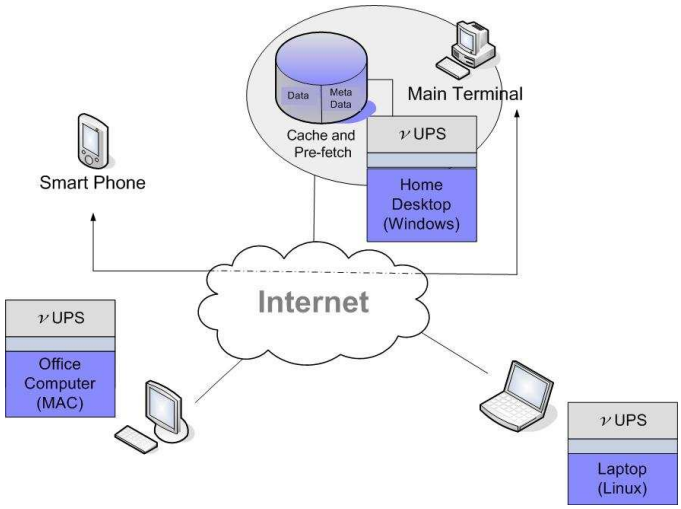

Fig. 1: The Architecture of vUPS

In vUPS, the participating devices peer with each other to virtually form **a** single and global storage space. Communications (e.g., to fetch data or to execute a user's commands) between devices are based on RESTful web services [25]. With such a P2P architecture, the device that the user is actively using as the main device (i.e., on which the user initializes her request) is responsible for supporting the main functions of vUPS. For this reason, we refer to this device as the `main terminal` of vUPS. Currently, at each time, there is only one main terminal. As the principal component of the vUPS, the main terminal is responsible for maintaining the vUPS namespace. In our design, the vUPS namespace is stored in a logically separate site. Once the main terminal is activated upon user accesses, the namespace will be loaded. In practice, this namespace can always be stored on the user's home desktop computer. All other participating devices are referred to as passive terminals, as they are mainly responsible for responding to the requests from the main terminal. In addition, users and other applications can interact with vUPS via vUPS APIs. Currently, we have designed a web-based user interface based on vUPS APIs.

Note that when a user actively uses her mobile device as the main terminal, it can deplete the limited battery power soon, because we expect that the main terminal could be relatively stable and stay on-line for a long time. Thus, in vUPS, the main terminal functions can be delegated by the mobile device to a more powerful computer, such as the home desktop (as shown in Figure 1) or the office computer.

When an application or a user needs to access a file, vUPS first finds the real device hosting the file based on the proper mapping in the vUPS namespace. vUPS resolves this mapping via the user name, device ID, resource path and operation type as: http://<usr>.vUPS.com/DeviceID/Path&Operation. Note that in the case of a delegated main terminal, the actual file transferring happens directly between the two involved devices without involving the main terminal.

2.2 vUPS Components

To support the desired functionalities, we design vUPS with the vUPS API, the vUPS Controller, the vUPS Synchronizer, the vUPS Data Manager. Figure 2 depicts the design with these major components. Next we discuss each of these components.

- *vUPS API:*
  To provide an interface of vUPS for users and applications, we develop vUPS API. These vUPS APIs support typical file operations, such as create, delete, read, write, as well as typical directory operations. To provide an easy access to users, we further develop a Web browser based user interface based on vUPS APIs. It uses HTML5 and Javascript to make it accessible from heterogeneous platforms such as smartphones, desktop and office computers that may run different operating systems.
- *vUPS Controller:*
  The vUPS controller consists of two modules. The first one is the bootstrap-

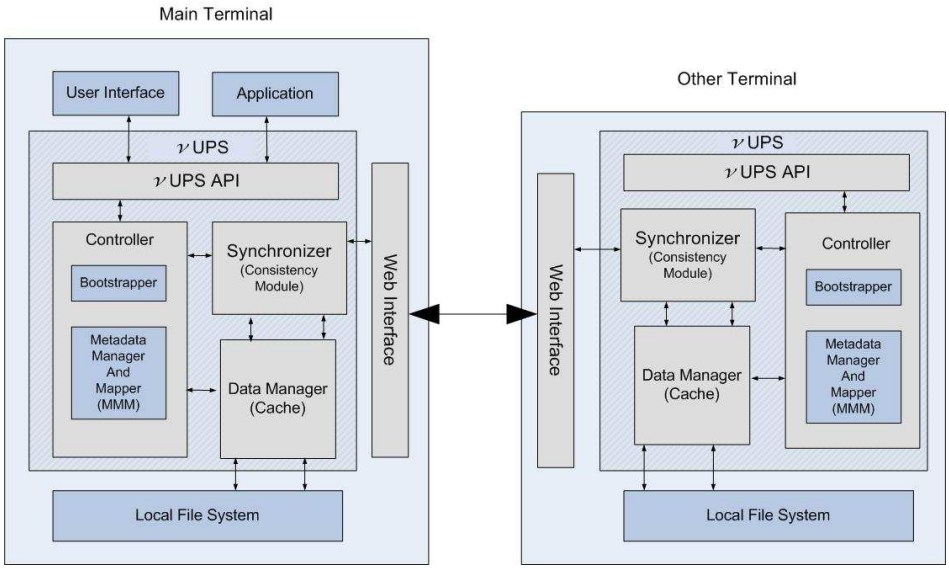

Fig. 2: vUPS Components

ping module or bootstrapper. Basically, when the main terminal is accessed by the user, it first needs to load the vUPS namespace, which contains the metadata and the directory hierarchy of the vUPS file system. When a new device wishes to join the system, it also provides the bootstrapping information so that any device can be added or removed from the system.
The second module is the metadata manager and mapper (MMM). Metadata is crucial to vUPS and strong consistency is maintained for metadata. MMM also maps files in the vUPS system to their physical locations in different devices. When a file is created by the user, a new entry is created to the appropriated location in the namespace. Accordingly, a file deletion removes the entry from the namespace.

– *vUPS Synchronizer:*
Maintaining consistency is essential to any distributed file system. vUPS Synchronizer is responsible for consistency maintenance among multiple copies of a file on different devices. As we shall discuss later, vUPS uses a non-traditional consistency model based on the file types in order to balance the overhead and the performance. Note that because the namespace is loaded on each participating device, the vUPS is also responsible for maintaining the consistency of namespace.
– *vUPS Data Manager:*
The data manager deals with the physical storage of the data in the local disk via traditional filesystem APIs. A particular notable function of the vUPS Data Manager is to manage cache for better user response time. As the core

of the cache management, a LRU (Least Recently Used) replacement policy is used for cache management.

These components work together in a collaborative manner. Basically, the namespace of the vUPS is often stored in a reliable device or a web site if a bootstrapping site is used. A participating device needs to contact this bootstrapping storage to load the initial namespace as well as the main terminal address.

When a file operation takes place (either by the user or by some application) through vUPS API, vUPS API passes the request to the vUPS Controller, which finds the appropriate file through the MMM.

The MMM then checks with the Synchronizer for cache validation. For read operation, the Synchronizer checks the cache and validates the cache if it is found in the cache. If it is not valid, then the Synchronizer finds the physical location of the data from the MMM. Then the Synchronizer either contacts the local or remote Data Manager to fetch the data. A remote Data Manager is invoked via the web interface. Whenever a request is received from the web interface, the Synchronizer enforces the operation via the local Data Manager. If the operation is write, then the Synchronizer finds the devices that contain the replicas and/or the cached copy of the data. Then the Synchronizer updates its local cache and data (if any) through the Data Manager. It also propagates the update to all the relevant devices via appropriate web service interfaces.

Note that vUPS relies on a storage place to store the initial namespace. This single point of failure can be eliminated by replicating the namespace among all the devices. This may however increase the maintenance overhead.

3 Consistency and Availability

Achieving availability and strong consistency at the same time is a major challenge in a distributed file system [30]. Traditional file systems replicate the data across multiple devices to increase availability, which also increases the overhead for consistency. Popular file systems have different policies to make the data consistent among replicas. For example, Coda [39], Ficus [27], and Ivy [33] apply optimistic concurrency control between the replicas. Coda resolves consistency conflict using version controlling. Bayou [43] provides high availability with a weak consistency model. Bayou was mainly designed for intermittently connected devices, which is also suitable for mobile devices. With pervasive network connections today, it is ideal to use continuous synchronization and have quick consistency. For example, cloud storage like Google Drive [9] uses strong consistency, while DropBox [5] applies continuous consistency between the replicas.

Both the optimistic and the continuous consistency models suffer from the scalability issue. In addition, continuous consistency also suffers from the overhead for continuous updates. That is, even a byte change in the data may result

in a lot of network traffic (several thousand more than the update itself). Such network overhead is amplified by the number of devices in the system.

Observing the fact that strong consistency is not required for every type of files, vUPS has different replication and consistency policies for different types of files. Today a user often owns much more media data (e.g., audio, video, and image) files than before. For example, it has been shown that a typical Windows user has 11.6% image files of the total storage [21]. This number is increasing due to the vast use of smartphones and tablets, because they are generally used to take pictures or videos. On the other hand, the documents and other files only consist of a small portion of the storage (i.e. 2.5% of the total file system [21]). Note that media files are not frequently changed. Only the corresponding metadata (favorite, ratings, etc.) may be modified. Therefore, we may relax the consistency model for media files as they are often non-editable, while the replicas of other documents have to be consistent as they are frequently edited. In addition to that, fetching the document files on demand is not expensive compared to that for media files as the media files size [16] is often bigger than that of documents files [21] on average.

Thus, in vUPS, files are divided into two categories: editable and non-editable. The audio, video, image, and program files are considered as non-editable. On the other hand, all other files, including doc, xml, and text files, etc., are categorized as editable files. Note that these non-editable files can also be updated very occasionally, but vUPS only maintains weak consistency among their replicas. In the current implementation, vUPS differentiates different types of files based on their filename extensions.

With two categories of files, vUPS has the following two different policies for different types of files: (1) limited availability with strong consistency; (2) high availability with weak consistency.

3.1 High availability with weak consistency

Considering the different types of popular files [22], we categorize the popular video, audio, and image files as non-editable files. Although in our current design these files are recognized solely based on file extensions, any self-learning clustering algorithm can classify the files over time based on the modification history. These non-editable files are seldom modified. Thus, the access to update rate is often high for these files. Moreover, the size of these media files is often larger than the editable files such as doc or xml [16] [21]. Caching and replicating these files on every possible device improves the access time to those files and results in less network traffic, but it also increases the overhead of maintaining consistency between copies.

As these files are seldom modified, vUPS follows a weak consistency model between the copies. A user can request the file from any device. The device may contain the data locally or fetch it from other devices (through the main terminal) and cache it. Whenever a modification takes place in a device, the change is reflected on the local/cached copy. The user does not need to wait for the update being propagated to other devices. Similar to Bayou, vUPS propagates

updates during pairwise contact. This epidemic approach ensures that as long as the participating devices are not permanently disconnected, the writes will eventually reach every copy [20].

For non-editable files, vUPS follows an invalidation protocol to notify the changes to other copies. As an invalidation protocol only notifies the changes, it saves network traffic and latency as the real update is not sent. The application in the devices may or may not pull the update for that file, depending on the policy of that application. vUPS aims to support application-specific conflict resolution, and each application may provide specific dependency check for updates. If the dependency check is valid, then the update takes place. Otherwise, conflict arises and the updates are either merged or marked invalid. Unlike scheduling or database transactions, media files may not have conflicting updates. Thus the merging procedure for media files ensures the latest update to a particular file is applied when any conflict arises. To detect the latest update, vUPS has vClock (vUPS vector logical clock) for each file and each folder. Each file has a vector of the Lamport logical clock [31], where each entry in the vector represents the timestamp for each device associated with that file. Whenever a file is created, a vector timestamp is created with entries for each device where the file is replicated. In addition to that, whenever that file is cached, an additional timestamp entry is added to that vector. If a cached copy is deleted or a replica is removed, the corresponding entry is removed from the vector. For every update from any device, the logical clock for that device in the vector is incremented. Whenever a device joins vUPS, all the data are updated to the latest vector. Thus, the copies are always up-to-date and synchronized if at least one copy of the data is always online. If only one copy of a file is kept connected to the vUPS and all the other copies are offline (that is, only one copy is mutual-exclusively online), then the file is updated if the timestamps of all the vectors are greater than the previous values. If all the entries in the vector are smaller than the new values, then it is not updated. Otherwise, the file is marked as conflicted and both copies are kept for the user to resolve manually. Thus, vUPS ensures the total ordering for the updates between the devices provided at least one copy is always online.

Note that, for non-editable files, the metadata may be changed frequently (a user may change the rating of movies, tag of pictures, etc.), but vUPS considers metadata as editable data, for which strong consistency is maintained among replicas.

Whenever a non-editable file is invoked by a device, the file is cached on the device and in the main terminal according to the caching policy. As the access-to-update ratio is higher, leaving a copy behind will improve the access performance [42]. So, when that file is closed, it is synchronized (if necessary), and a copy is left in the cache. The namespace contains a callback reference to that address for providing response from that copy to other devices.

3.2 Limited availability with strong consistency

For copies of the editable files, vUPS maintains strong consistency. As strong consistency is not scalable, vUPS maintains a minimal number of copies to maintain the availability and consistency to get the best of both worlds.

Similar to non-editable files, whenever an editable files is accessed by a device, the file is cached on the device and in the main terminal according to the caching policy. All the modifications take place on the cached copy and are propagated to remote copies (local read/remote write). When that file is closed, it is synchronized (if necessary), and the local cached copy is deleted. The callback reference is also deleted from the namespace once the file is closed. That is, vUPS enforces a strong consistency policy between the copies. It sends the update operation where data should change (active replication).

As the access-to-update ratio is lower, keeping a local copy close to the device may not be better [42] as it may send frequent updates to all the replicas while these updates may not be accessed by the user. In this case, keeping sending these updates can waste bandwidth and may not be scalable. So it deletes all the cached copies once the read/write operation in the cached copy is completed. This approach makes vUPS more scalable. As the average size of editable files is smaller than that of non-editable files, bringing these files to the cache when needed does not affect the performance too much. In addition, for these types of files, a typical replication policy results in less network traffic.

Assume the failure probability of one machine is p and the number of replication is r. Then, the failure probability of all replicas is:

$$M(n,p,r) = (p^r \times \sum_{i=0}^{n-r} p^i \times (1-p)^{(n-r)-i}) \tag{1}$$

Let the unavilability probability of one machine due to network is q and the number of replication is r. Then, the unavailability probability of all replicas due to network failure is:

$$N(n,q,r) = (q^r \times \sum_{i=0}^{n-r} q^i \times (1-q)^{(n-r)-i}) \tag{2}$$

$M(n,p,r)$ and $N(n,q,r)$ are independent from each other. So, the availability of the over all system is:

$$A(n,p,q,r) = 1 - (M(n,p,r) + N(n,q,r) - M(n,p,r) \times N(n,q,r)) \tag{3}$$

The typical disk failure rate is 4% [7]. If we take this into account, the availability of vUPS can be deduced to be more than 96% for a typical system with four machines with no replication. We can get more than 99% availability by keeping a backup (a replica) of each file, which is comparable to amazon services (with a failing rate 0.1-0.5% [7]).

In case of a terminal failure, the active requests for that terminal is re-directed to other terminals with replicated copies. When the device joins vUPS again, the replica in that device is checked again (with SHA-1) with other replicas and synchronized if necessary.

vUPS is also highly robust in the sense that any terminal can be initiated, allocated, deleted or backed up without interrupting the system. vUPS also allows adding or deleting a disk, allocating storage, moving data, or interrupting system availability. This design allows all data to be remained online and available to clients and other terminals.

4 vUPS Implementation

We have implemented a prototype of vUPS with Java and HTML. The user may add devices with any operating system and can browse, access, and store data in the vUPS storage consisting of those devices. In addition to the user interface, we have implemented the vUPS APIs in Java for applications. Figure 3 shows a screen-shot of vUPS user interface.

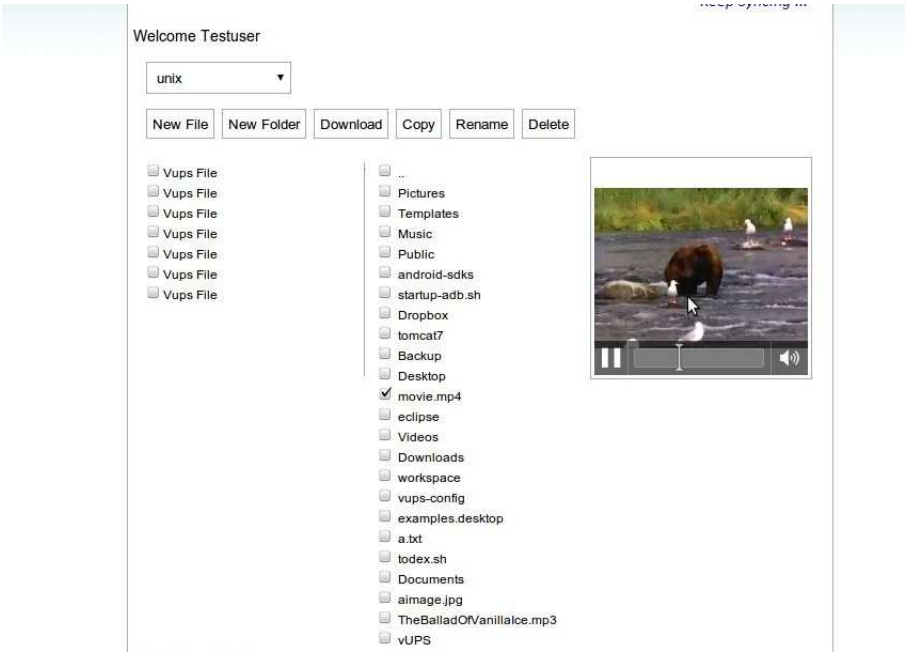

Fig. 3: vUPS User Interface

The *vUPS APIs* are built with Javascript and HTML5. A local web server is used for bootstrapper right now where a user registers her devices. After login,

the user is provided with the main HTML page where they may select any devices, access data, copy between devices, etc. These operations are implemented by Javascript to call the web services in the devices. The *Metadata Manager and Mapper (MMM)* manages the mapping between the vUPS files to their physical locations that are saved in a SQLite database.

The *vUPS Synchronizer* communicates with the devices through the RESTful architecture as mentioned earlier. The *Data Manager* gives an abstraction over the local file system with the underlying physical storage in the local disk via Java filesystem APIs.

For mobile devices, we also build a vUPS app to access the data. There is also vUPS API for Android to access vUPS from other apps. The current prototype maintains minimal security over the RESTful architecture in HTTP. We use the IP address of the device as the device ID in the current prototype implementation.

In the current implementations, files are simply differentiated based on their name extensions. Extensions like .mpeg, .wmv, .dat, .mov are considered as the popular movie filename extensions. vUPS includes the .mp3 and .mid as the popular audio files, where the .jpeg, .jpg, .gif, .bmp, and .png are considered as filename extensions of image files. Files with these name extensions are all considered as non-editable files. By default, all other files are considered to go under frequent modifications, and vUPS have different policies for them.

5 Performance Evaluation

In this section, we evaluate the performance of vUPS based on the prototype we have implemented. In the experiments, we first use the micro-benchmarks to evaluate vUPS. Then we compare the performance of vUPS with the popular application Dropbox.

In these experiments, we configure vUPS with three commodity desktops, one laptop, and one smartphone. To compare with the service provided by DropBox, we have also set up a DropBox account.

All the desktop machines have 8 GB memory and 2.7 GHz CPU. The laptop has 4 GB memory and 2.7 GHz CPU. The desktop machines run Windows 7, Ubuntu 11 and Mac operating systems, respectively. We use a Google Nexus One phone with 512 MB memory and 1 GHz CPU running Android 2.3 operating system. The DropBox account has 2 GB of storage.

5.1 File I/O Performance

First, we study the performance of file reading. In vUPS, a desktop is designated as the main terminal. Files of 1 KB, 10 KB, 50 KB, 100 KB, 200 KB, 500 KB, 1 MB, and 3 MB are read by the smartphone via the main terminal. To emulate the realistic scenario, files are randomly distributed on these devices. When a read request for a file is sent from the smartphone to the vUPS, the main

terminal receives the request from the smartphone. The main terminal searches the namespace for the ID of the device that stores the file, and forwards the web address of the resource to the smartphone. The smartphone then completes the read operation by bypassing the main terminal. The results are compared against when a direct/local read from the sdcard of the smartphone and when the file of the same size is located on DropBox. We take the average of 100 runs for each file size and the results are shown in Figure 4. The x-axis represents the file size. The y-axis shows the response time. We set the network speed as 500 Kbps for local accesses. The maximum and the minimum response time are within 2% of the average with 95% confidence level. As expected, the read time increases with the increase of the file size and neither vUPS nor DropBox is comparable to the local read performance. However, vUPS constantly outperforms DropBox, for which the network speed is unconstrained with an average of 3 Mbps, although the advantages tend to diminish along the increase of the file size.

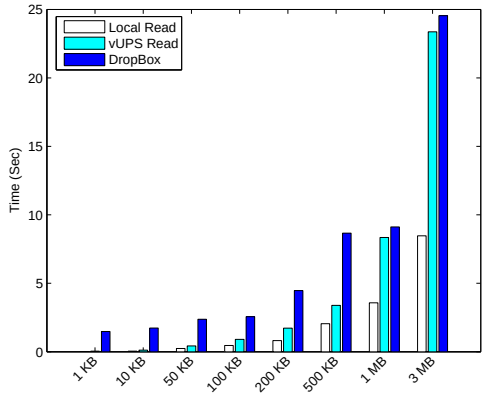

Fig. 4: File Size vs. Response Time

To bridge the performance gap for the sequential read in vUPS read, in the next experiment we use parallel threads for multiple file fetching. Figure 5 shows the result when the number of parallel file fetching grows from one to four. The x-axis represents the number of files fetched in parallel and the y-axis represents the throughput of the system. For the counterpart of the local read, four files are read by four parallel threads from the sdcard of the smartphone.

To read files in parallel from vUPS, the smartphone contacts the main terminal and requests the files. Each file is stored on a random machine selected arbitrarily. The main terminal returns to the smartphone with the web address of the file resources and the smartphone then fetches those files in parallel. Figure 5 shows that the throughput increases with the degree of parallelism. The throughput also increases with the network bandwidth for vUPS when the network bandwidth increases from 500 Kbps to 1 Mbps. When the available network

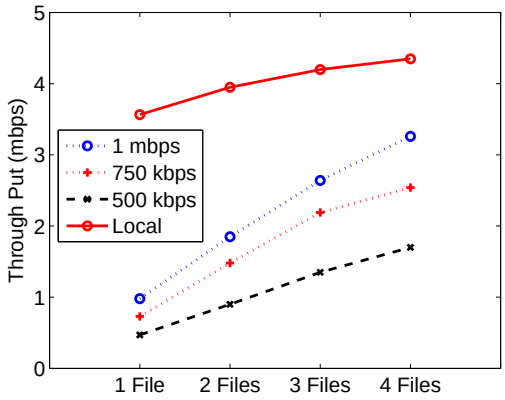

Fig. 5: Parallel Threads for File Read

bandwidth is sufficiently large, the gap between the local read and the vUPS read can be significantly reduced.

We have mentioned before that the file operations are handled directly between the requesting machine and the target machine without involving the main terminal in order to relieve the communication bottleneck. To study the performance gain, we compare it to the case when the main terminal fetches the file and then serves it to the smartphone. Figure 6 shows the results averaged over 100 tests. The results confirm that it is beneficial for the user and the system to bypass the main terminal whenever necessary.

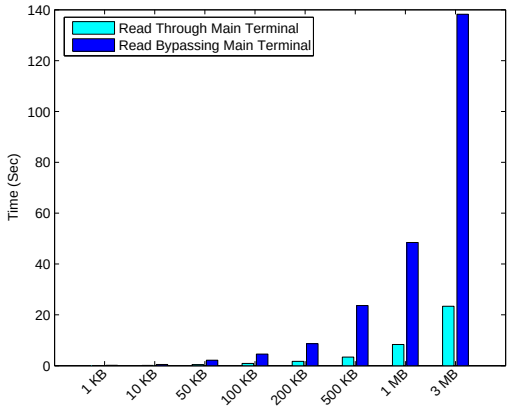

Fig. 6: Performance Gain by Bypassing the Main Terminal

5.2 Performance of Metadata accesses

Metadata is crucial to vUPS to function properly. To measure metadata access performance, we conduct experiments with the *mdtest* benchmark on vUPS. *mdtest* is developed for large scale file systems with script and C, which is not suitable for our RESTful applications. To fit our system, we modify the *mdtest* and implement the simplified version in Java with threading. We replace the file reading/write/edit with Java system calls. In the experiments, we measure the throughput (the number of operations executed per second) for creating 1000 files per terminal from the smartphone.

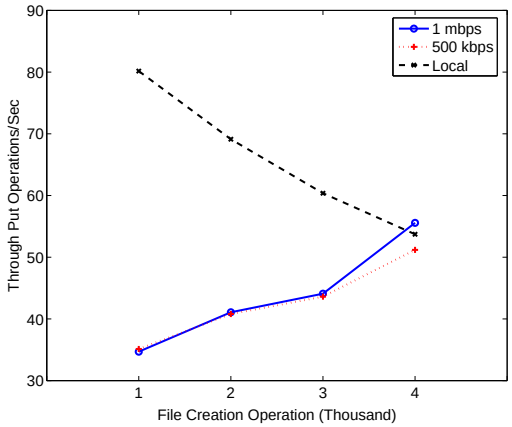

Fig. 7: Throughput of File Creation

Figure 7 shows the throughput for file creation. In the experiments, we compare the time for creating files in our smartphone locally and in vUPS. For local file creation in the smartphone, we create 1000, 2000, 3000, and 4000 files using 1, 2, 3, and 4 threads (a thousand files per thread) in the sdcard of the smartphone. To measure the time of file creation in vUPS, we again create 1000, 2000, 3000, and 4000 files using 1, 2, 3, and 4 threads (a thousand files per thread). Each thread selects one of the devices in the vUPS system. Each thread first contacts the main terminal and requests to create 1000 files in the device.

The result shows that with an increasing number of local file creation requests, the throughput decreases. This is expected because the smartphone thread overhead and the limited file I/O slow down the local file operation, and thus reduce the throughput. On the other hand, for remote devices the performance improves, which is due to the parallel file creation feature in vUPS.

5.3 Network Impact

To study the impact of network bandwidth, we have also run the *mdtest* file creation of 1000 files with varying network speed. Figure 8 shows the response time for 1000 file creation via the smartphone using one thread. The x-axis represents the network speed, while the y-axis represents the system throughput. The figure shows that the network bandwidth does impact the file creation throughput, but the improvement diminishes when the network speed is fast enough. For example, when the network bandwidth is doubled from 5 Mbps to 10 Mbps, the throughput for file creation operation only increases slightly.

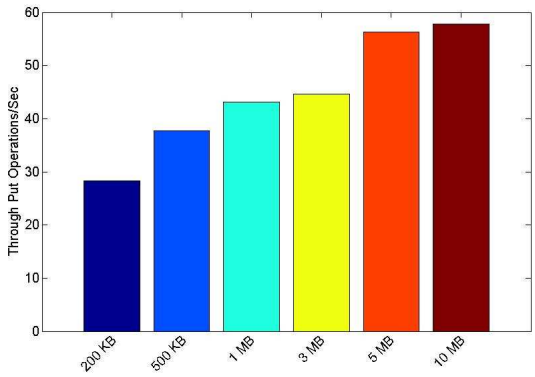

Fig. 8: Network Bandwidth vs File Creation

To study the impact of network speed on file read operations, we vary the network speed and observe the response time to read a 3 MB file in the smartphone fetched from the main terminal. As expected, Figure 9 shows that the read throughput increases with the increase of bandwidth for vUPS while it remains stable for local read and DropBox. Note that because Dropbox is accessed through the public network, we did not constrain the bandwidth of the connection between the smartphone and the Dropbox.

Table 1 shows the response time for 5 trials with different network speed and file read or creation operations. From ANOVA [29] tests, we may conclude to reject the hypothesis that the operation type and network speed have no significance on the model. We can also conclude that the interaction between the network speed and the file operation type does exist.

5.4 Comparison to Dropbox

To compare with the DropBox in a more realistic scenario, we have generated 50 files with lambda distribution [37] as discussed in [24] with the size ranging from 1 KB to 512 MB according to that distribution. For DropBox, we synchronize all

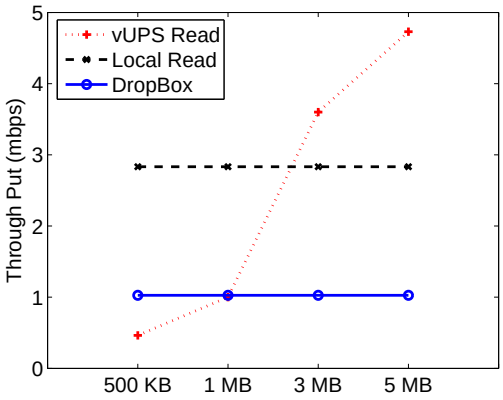

Fig. 9: Network Bandwidth vs File Read

Operation	500 Kbps	1 Mbps
Read	186	120
Read	204	92
Read	222	164
Read	221	101
Read	228	180
Create	55	58
Create	17	40
Create	37	27
Create	43	32
Create	26	29

Table 1: ANOVA Test for Network and Type of Operation

the files first, then access locally. In vUPS, in case of a cache miss, it downloads the file, otherwise checks the cache and retrieves it unless it is stale. We access these files in random order for 10 to 1000 times.

Figure 10 shows the result. The x-axis represents the number of files fetched randomly from these 50 files. The y-axis represents the response time. In the experiment, the DropBox first downloads all the files, and then accesses them randomly. On the other hand, vUPS does not download all the files at first. It downloads the files when being accessed and then caches them for future references. In this way, when the number of accessed file is small, the access time for DropBox is much longer than vUPS as the DropBox first needs to synchronize and download all the files. But when the number of accessed files is larger, the performance of both vUPS and DropBox is similar to each other.

In addition to that, the strong consistency between the replicas introduces synchronization overhead. For example, even a single byte update in dropbox

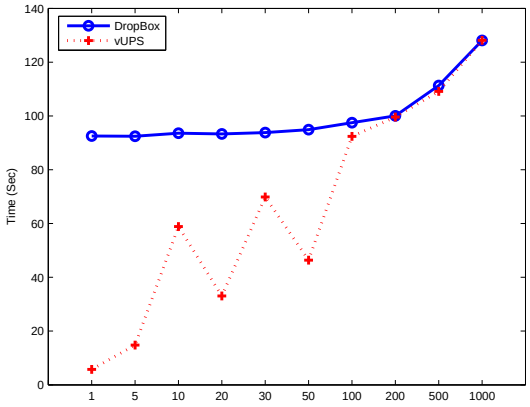

Fig. 10: DropBox vs vUPS

generates 37 KB of network traffic measured by wireshark, where vUPS generates only 1KB of data for the same synchronization.

6 Related Work

Distributed filesystems [38], [28] have been a focus in the field of distributed systems and data replication and consistency have been a major concern for these systems [19].

NFS [38] has been widely used. Coda [39] also proposes offline accesses and replicated access control. Large scale distributed file systems, such as Goolge File System (GFS) [26] and the Hadoop Distributed File System (HDFS) [18, 44], have also been well studied for distributed computing. Panache [23] also proposes improvement for parallel accesses. These traditional and research-oriented file systems are designed for local area network and desktop terminals, while in the modern age various mobile devices (e.g., smartphones) play a crucial role for users' data generation and access over public network.

More recently, modern distributed file systems ZZFS [32],Eyo [41], and BlueFS [34] consider the modern portable storage devices such as tablet and smartphones. BlueFS has optimized the data placement policy whose data update was improved by EnsemBlue [35] from a centralized server to peer to peer. ZZFS [32] has specific data placement polices and the consistency policy to avoid conflicts, while Eyo [41] does not guarantee any consistency. PersonalRAID [40] provides device transparency for partial replication which is not suitable for public network as it requires user to move a storage physically between the devices. Cimbiosys [36] provides efficient synchronization with minimized overhead which is not device transparent. But these systems usually have the same repli-

cation and consistency policy for all types of data, without distinguishing the very difference between the natures of different types of files.

On the other hand, recent commercial products, such as iCloud [10], Google Drive [9], and Dropbox [5] offer to synchronize data among multiple devices of the same user. But storing users' data on the third party storage potentially risks with security and privacy concerns, while the study [17] shows that the desktop and office computers possessed by an average user is enough to store the data owned by the user.

vUPS has been developed to store the data on the user's own devices without storing data in the public cloud. The API's of vUPS are developed with an aim of the end users who access files from samrtphones and/or other terminals.

7 Conclusion

The pervasive adoption of mobile devices calls for an effective approach to access user's personal files across different devices from anywhere and anytime. While commercial products have offered compelling solutions, users are risking their data security and privacy. In addition, potentially, lots of unnecessary traffic has been wasted during continuous file synchronization. In this work, we have designed and implemented vUPS, a system to transparently and seamlessly integrate a user's personal storage space. A web interface is provided to the user with a global view of the files without involving any third party. Experiments based on the implemented prototype system show that vUPS can achieve similar user performance when compared to commodity commercial solutions such as DropBox.

References

1. Amazon EC2 outage. http://www.informationweek.com/news/cloud-computing/infrastructure/229402054.
2. Amazon S3. http://aws.amazon.com/s3/.
3. Cisco Press Release. http://www.rcrwireless.com/article/20110201/.
4. Daily Main, April 1, 2012. http://www.dailymail.co.uk/sciencetech/article-2078020/Death-point-shoot-Smartphone-cameras-27-cent-photos.html.
5. Drop Box. http://www.dropbox.com/.
6. DropBox security breach. http://www.newsy.com/videos/dropbox-security-glitch-leaves-users-accounts-unprotected/.
7. EC2 Failure Rate. http://aws.amazon.com/ebs/.
8. Google Docs. www.docs.google.com.
9. Google Drive. https://drive.google.com/.
10. iCloud. http://www.apple.com/icloud/.
11. International Data Corporation : Press Release 28 Jan and 4 Feb, 2010. http://www.idc.com/.
12. International Telecommunication Union : Press Release 10 June, 2009. www.itu.int.

13. Mark Zuckerbergs pictures leaked. http://www.nydailynews.com/news/national/oops-mark-zuckerberg-pictures-leaked-facebook-security-flaw-article-1.988026?localLinksEnabled=false.
14. Sky Drive. http://explore.live.com/skydrive.
15. SME Storage. http://smestorage.com/.
16. Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A five-year study of file-system metadata. In *FAST*, 2007.
17. William J. Bolosky, John R. Douceur, David Ely, , and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Sigmetrics*, 2000.
18. Dhruba Borthakur. Hdfs architecture : Technical report. In *Apache Software Foundation*, 2008.
19. Eric A. Brewer. Towards robust distributed systems. In *Principles of Distributed Systems*, 2000.
20. Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *PODC*, 1987.
21. John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. In *Sigmetrics*, May 1999.
22. John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. In *SIGMETRICS*, 2002.
23. Marc Eshel, Roger Haskin, Dean Hildebrand Manoj, Naik Frank, Schmuck, and Renu Tewari. Panache: A parallel file system cache for global file access. In *8th USENIX Conference on File and Storage Technologies*, 2010.
24. Kylie M. Evans and Geoffrey H. Kuenning. A study of irregularities in file-size distributions. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, San Diego, CA, 2002.
25. Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. In *PhD Thesis, University of California, Irvine.*, 2000.
26. Sanjay Ghemawat, Howard Gobioff, , and Shun-Tak Leung. The google file system. In *Proceedings of the 19th Symposium on Operating Systems Principles (SOSP)*, NY, USA, 2003.
27. Richard G. Guy, John S. Heidemann, Wai Mak, Jr. Thomas W. Page, Gerald J. Popek, and Dieter Rothmeier. Implementation of the ficus replicated file system. In *USENIX*, March 1990.
28. JOHN H. HOWARD, MICHAEL L. KAZAR, SHERRI G. MENEES, DAVID A. NICHOLS, M. SATYANARAYANAN, ROBERT N. SIDEBOTHAM, and MICHAEL J. WEST. Scale and performance in a distributed file system. In *ACM Transactions on Computer Systems*, 1988.
29. Raj Jain. The art of computer systems performance analysis. 1991.
30. H.T. Kung and John T. Robinson. On optimistic methods for concurrency control. In *ACM Transaction on Database Systems*, March 1981.
31. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Communicatio of ACM*, 1978.
32. Michelle L. Mazurek, Eno Thereska, Dinan Gunawardena, Richard Harper, , and James Scott. Zzfs: A hybrid device and cloud file system for spontaneous users. In *FAST*, 2012.
33. Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *OSDI*, 2002.

34. Edmund B. Nightingale and Jason Flinn. Energy-efficiency and storage flexibility in the blue file system. In *6th conference on Symposium on Opearting Systems Design and Implementation*, 2004.
35. Daniel Peek and Jason Flinn. Ensemblue integrating distributed storage and consumer electronics. In *7th conference on Symposium on Opearting Systems Design and Implementation*, 2006.
36. Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, Meg Walraed-Sullivan, Ted Wobber, Catherine C. Marshall1, and Amin Vahdat. Cimbiosys: A platform for content-based partial replication. In *Network Systems Design and Implementation*, 2009.
37. John S. Ramberg and Bruce W. Schmeiser. An approximate method for generating symmetric random variables. In *Communications of ACM*, 1974.
38. Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the sun network filesystem. In *Summer 1985 USENIX*, June 1985.
39. Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. In *IEEE TRANSACTIONS ON COMPUTERS*, April 1990.
40. Sumeet Sobti, Nitin Garg, Chi Zhangv, Xiang Yu, Arvind Krishnamurthy, and Randolph Y. Wang. Personalraid: Mobile storage for distributed and disconnected computers. In *FAST*, 2012.
41. Jacob Strauss, Justin Mazzola Paluska, Bryan Ford, Chris Lesniewski-Laas, Robert Morris, and Frans Kaashoek. Eyo: Device-transparent personal storage. In *USENIX Technical Conference*, 2011.
42. Andrew S. Tanenbaum. Distributed systems : Principles and paradigms. volume Second Edition.
43. D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demer, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *fifteenth ACM symposium on Operating systems principles*, March 1981.
44. Tom White. Hadoop: The definitive guide (second edition).