Journal of Network and Computer Applications 43 (2014) 142–156

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Leveraging client-side storage techniques for enhanced use of multiple consumer cloud storage services on resource-constrained mobile devices

Hui-Shyong Yeo [a], Xiao-Shen Phang [b], Hoon-Jae Lee [c], Hyotaek Lim [c,*]

[a] Department of Ubiquitous IT, Dongseo University, 617-716 Busan, South Korea
[b] Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia
[c] Division of Computer and Information Engineering, Dongseo University, 617-716 Busan, South Korea

ARTICLE INFO

Article history:
Received 14 August 2013
Received in revised form
27 February 2014
Accepted 1 April 2014
Available online 30 April 2014

Keywords:
Multiple cloud storage
Mobile devices
Erasure coding
Fault-tolerance
Storage techniques

ABSTRACT

Despite high adoption rate among consumers, cloud storage services still suffer from many functional limitations and security issues. Recent studies propose utilization of RAID-like techniques in addition to multiple cloud storage services as an effective solution, but to the best of our knowledge, there is no research work done on applying this approach to resource-constrained mobile devices. In this paper, we propose a solution for mobile devices that unifies storage from multiple cloud providers into a centralized storage pool that is better in terms of availability, capacity, performance, reliability and security. First, we explore the feasibility of applying various storage technologies to address the aforementioned issues. Then, we validate our solution in comparisons with single cloud storage by implementation of a working prototype on mobile device. Our results show that it can improve the usage of consumer cloud storage at zero monetary cost, while the minimal overheads incurred are actually compensated by the performance gained.

1. Introduction

Cloud storage services provide online storage where data is stored in virtualized "pools" of storage hosted by third parties, usually spanning large data centers in different geographical locations. The service can be accessed by users from anywhere, during anytime and using any device, through an Internet connection. It has become a trend in ubiquitous data access and has revolutionized the way users access their personal data, by eliminating the need to keep in one's possession external storage devices all the time. This has been mainly attributed to the increasingly desire of users to share content and the expectation to have continuous access to their data using multiple computing devices such as smartphones and tablets. It also has become a viable backup, file sharing and collaboration solution.

Growth in use of smart mobile devices, along with shortages of hard disk inventory during 2011 together have provided an impetus for cloud storage adoption among consumers. As media quality continues to improve, storage poverty on mobile device is becoming an imminent issue. Fortunately, ubiquitous wireless access to cloud storage services on mobile devices allows people to rely on cloud storage as the main repository for their ever growing media collection. Gartner (2012) predicts consumers will store 36% of their digital content in the cloud by 2016, compared to a mere 7% during 2011.

Many consumer cloud storage services are available in the market. They vary in terms of pricing, features and performance, as visualized in Table 1. One may notice that every provider offers very limited free storage capacity while the pricing for paid subscription is relatively expensive (Naldi and Mastroeni, 2013; Han, 2011). In addition to functional limitations such as limited file upload size, lack of media streaming support and slow performance (Armbrust et al., 2010), there are other non-performance related issues that are often overlooked by users such as security breaches (Abu-Libdeh et al., 2010), vendor lock-in issues (Armbrust et al., 2010; Abu-Libdeh et al., 2010), frequent service outages (Armbrust et al., 2010), data corruption (Kumar and Lu, 2010) and privacy concerns (Ion et al., 2011).

In this paper, we study the feasibility of leveraging client-side efforts to address these limitations. In particular, we propose to use simple storage techniques and exploit services from multiple providers to improve the usage model of cloud storage in current ecosystem. We developed the proof-of-concept prototype of a middleware application on real hardware devices and evaluated its performance in real world scenario. Nevertheless, we restrict

* Corresponding author.
E-mail addresses: hsyeo@dongseo.ac.kr (H.-S. Yeo), ms.phang2@gmail.com (X.-S. Phang), hjlee@dongseo.ac.kr (H.-J. Lee), htlim@dongseo.ac.kr (H. Lim).

http://dx.doi.org/10.1016/j.jnca.2014.04.006
1084-8045/

Table 1
Comparison between several popular consumer cloud storage services available in the market.

	Dropbox	Google drive	Box	SkyDrive	Sugar sync	Amazon cloud	Spider oak	iCloud	Wuala
Free quota	2 GB	5 GB	5 GB	7 GB	No	5 GB	2 GB	5 GB	5 GB
Paid quota (GB)	100–500	100–1000	25–50	20–200	60–500	20–1000	100	10–50	20–2000
Monthly cost per GB	$0.1	$0.05	$0.4	$0.0416	$0.1	$0.0416	$0.1	$0.167	$0.14
Media streaming on mobile app	Yes	No	No	No	Yes	Yes	No	Yes	Yes
File size limitation	No limit	10 GB	250 MB/5 GB*	2 GB	No limit	2 GB	No limit	25 MB/250 MB*	100 GB
File versioning	30 days	30 days	No/25 versions*	25 days	5 versions	No limit	No limit	No	10 versions

* Available on paid subscriptions only.

the scope of our study on consumer cloud storage services and mobile devices because several interesting challenges are raised when applying such computational intensive storage techniques on mobile device with constrained resources. We demonstrate through an experimental study that our solution can achieve extra availability, capacity, reliability, security and performance at zero monetary cost.

The remaining of this paper is structured as follows. Section 2 outlines current limitations and issues related to consumer cloud storage services and mobile devices. Section 3 provides the design and architecture of our proposed solution. In Section 4, we present the actual prototype system and some implementation considerations. In Section 5, we evaluate the performance alongside with discussions. Section 6 gives an overview of related work. Finally, in Section 7 we conclude our paper and discuss on how this research can be further explored.

2. Background and motivation

In this section, we discuss the inherent limitations of cloud storage services and how it impacts the user, especially when accessing cloud storage service on mobile device.

2.1. Functional limitations

Limited storage capacity and expensive pricing. Typically, mobile devices have limited internal storage capacity, ranging from 16 GB to 32 GB only. Therefore, cloud storage is promising as a great alternative to extend storage capacity on these devices. However, most cloud storage services only offer limited free storage capacity (2 GB to 7 GB) that is unquestionably insufficient. On the other hand, the cost for paid subscriptions that offer higher storage capacity are relatively expensive and it may be a burden to casual users. This is because users need to pay a monthly or annual fee as long as they are subscribed to the service (pay-per-use model). It reveals that while the entry cost into cloud storage services is relatively cheap, it is actually terribly expensive in the long term (Han, 2011; Abu-Libdeh et al., 2010).

Low performance and bandwidth utilization. It is inevitable that the I/O performance of using cloud storage is slower than using local storage device, mainly due to the latency and throughput in accessing the data stored in the cloud via network connection. In theory, the performance is proportional to the user's network bandwidth but in reality, the performance is usually bottlenecked on the provider side. As cloud storage service is generally offered to a vast number of public users, the available bandwidth of the data center is shared between multiple users at the same time. To ensure steady operation and quality of service (QoS) of their services, some providers perform bandwidth control and throttling[1] to allocate limited bandwidth for each user fairly. As a result, user's network bandwidth is not being fully utilized and may experience slow performance.

Low energy efficiency. The official mobile application of each provider only allows one concurrent network operation at one time. It seriously bottlenecks the entire system, causing slow data throughput and short traffic bursts, which result in long waiting time and idle periods (Google, 2012; Alexandre et al., 2011; Qian et al., 2011), during which a device keeps the radio channel occupied. This is undesired because low efficiency of radio resource usage will drain the battery quickly. The energy issue is not a concern in the desktop environment, but it is particularly crucial in the context of mobile devices due to the limited battery capacities.

Limited features. There are others functional limitations such as limited file upload size and lack of media streaming support. These features, which are these services are mostly lacking, are key factors for achieving seamless data access especially on mobile devices.

2.2. Non performance related limitations

Vendor lock-in issue. Depends solely upon a single cloud provider incurs a great risk of experiencing vendor lock-in (Abu-Libdeh et al., 2010) when users are seemingly “trapped” in their use of one particular service provider and it is often prohibitively expensive for them to switch to another provider, in terms of monetary, time and bandwidth cost. The longer a user is “trapped” with that provider, the higher the switching cost because more data needs to be migrated. This phenomenon is commonly known as “data inertia”. Furthermore, data migration requires a two-way operations (download and re-upload), which results in doubling the initial cost. Hence, users are vulnerable to price hikes or new pricing terms. Users are also subjects to the possibility of data loss if the provider goes out of business suddenly.

Frequent service outages. Despite guaranteed high availability and uptime (99.9%) in Service-Level Agreement (SLA), there are many cases where public cloud services encountered occasional service outages[2]. It may due to human errors, hardware failures or natural disasters. During such events, depending solely on a single provider results in the possibility that users are not able to access their data due to a single point of failure. Outages may result in severe financial losses or other kind of losses on both the provider and client.

Security breaches. Many providers encountered security breaches in the past[3]. Since most of the providers maintain the encryption key by themselves, all users' data will be in risk if the key is compromised. Moreover, the security scheme and its level of security occupied by the service provider are also not well understood, let alone controllable by the users. In fact, recent security

[1] We observed bandwidth throttling on Dropbox and SkyDrive during our evaluation.

[2] Amazon AWS outages (April 2011, June, October 2012), Dropbox service disruptions (August, October 2012).

[3] Dropbox accounts were publicly accessible for several hours (June 2011), Dropbox employee's account password was stolen and spam messages had been send to users (August 2012).

breaches have revealed that even tech giants are using inappropriate or insufficient security schemes (Kamp et al., 2012).

Data loss and data corruption. Despite service providers claiming to have used redundancy protection scheme such as RAID, yet unrecoverable data losses have occurred in the past[4]. Many users believe that their provider is liable in case of data loss (Ion et al., 2011), but in fact most of the providers are not responsible for, nor take liability for, any data loss, as stated in their terms and service. Therefore, users are always responsible for backing up their data into several locations by replication or other similar redundancy scheme.

Privacy concerns. Cloud storage raises many questions concerning compliance with privacy and security laws as the loss of data through industrial espionage and theft of personal and commercial intellectual property is the equivalent of terrorism to the virtual world. A key problem with privacy is regarding the data ownership because the service provider might use the data in a way the user never intended. A great example is the existence of the PRISM surveillance program that allows the government to spy on users' data directly.

3. System architecture

With the aim to tackle all the issues outlined in Section 2, we propose an overall architecture design as illustrated in Fig. 1. It follows the layered design outlined in RAOC (Spillner et al., 2013). There are three main layers to the architecture: (i) Mini cloud gateway controller. (ii) Data processing layer. (iii) Data presentation layer. The controller layer consists of several functions that will be further discussed in this section, but can be summarized as parallelization (Section 3.4), load balancing (Section 3.5), caching, pre-fetching and network awareness (Section 3.8). The data processing layer is further divided into three internal modules, which can be pipelined. The first module consists of four modes (normal, data striping, erasure coding, and replication) and only single mode can be chosen at a time. Each mode has different benefits and trade-offs that will be further discussed in Sections 3.2 and 3.3. The second and third internal modules consist of compression (Section 3.7) and encryption (Section 3.6) module, which are optional. Finally, the presentation layer consists of streaming proxy (Section 3.9), version control (Section 3.10), de-duplication (Section 3.7) and file abstraction (Section 3.1). The system is designed to be modular so that new module can be added easily to extend the features such as supporting more cloud providers and adding new storage techniques. Moreover, our proposed solution is designed as a portable system that can run on any compatible device.

3.1. Unified cloud storage access

An ingenious solution to boost one's available cloud storage capacity is to sign up for multiple free services offered by different providers. However, another problem arises therefrom as users face increasing difficulties in finding their files that are scattered among various locations (clouds). To take advantage of this solution, we need to provide users with a unified cloud storage access across multiple providers where they are dealing with a single storage pool (Fig. 2(a)) and can easily access or organize their entire files collection (Fig. 2(b)). Additionally, users should be able to perform simple file manipulation operations such as Create, Rename, Update and Delete (CRUD) or advanced operations such as searching, filtering and sorting. Hence, single point of contact, centralized management and intuitive user interface are critical factors in designing the architecture of our system.

3.2. Data striping

To bypass the file size limitation imposed by the provider, we utilize a common storage technique known as data striping, which stripes a piece of data into smaller chunks so that each chunk can be uploaded (Fig. 3(a)) without encountering the size limitation. The smaller chunks can be stored in either the same provider or different providers using simple Round-Robin scheme. An advantageous side effect of applying data striping and storing smaller chunks in different providers is the vendor lock-in issue can be mitigated. The cost of data migration is effectively reduced as the amount of data stored in each provider is only $1/n$, where n is the total number of cloud storage services being subscribed to. Other benefits that arise from data striping such as improved overall bandwidth utilization and load balancing will be further discussed in Sections 3.4 and 3.5, respectively.

3.3. Replication and erasure coding

Replication is a technique commonly used in years prior to provide fault-tolerance and extra availability for storage system. It incurs high storage overheads that are proportional to the level of redundancy desired. This technique is also applicable to multiple cloud storage locations but there are additional bandwidth overheads. In seeking an alternative solution, recent studies (Ion et al., 2011; Slamanig and Hanser, 2012; Spillner et al., 2013; Seiger et al., 2011; Bian and Seker, 2009; Plank et al., 2008; Plank et al., 2009; Hu et al., 2012; Resch and Plank, 2011; Bessani et al., 2011; Bowers et al., 2009; Mu et al., 2012; Cachin et al., 2010) propose to stripe and store data across multiple cloud vendors retaining data reliability by applying techniques such as erasure coding (Fig. 3(b)). Erasure coding breaks an object into k equal size fragments and generates extra n fragments resulting in a total of m fragments such that the original object can be recovered from any k fragments. It ensures data reliability at much lower overheads (storage cost is increased by a factor of $1/r$ where $r=k/n$) as compared to replication (Weatherspoon and Kubiatowicz, 2002). Data loss is only occurred when at least k–n fragments are permanently lost. Besides, extra security can be achieved by efficiently dispersing data to several targets using information dispersal algorithm (IDA) (Rabin, 1989) that is based on secret sharing (Shamir, 1979). It can provide data integrity and protect against data corruption. In short, by applying erasure codes to cloud storage, users are more resilient in the face of disasters, outage risks and vendor lock-in because it is unlikely that catastrophic events would happen to all providers at the same time.

3.4. Parallelization and pipelining

Parallel processing allows computations to be executed simultaneously so that performance is improved. It is commonly used in systems with multi-core processor. Data parallelism allows data to be divided into multiple chunks so that each chunk is processed concurrently. Pipelining is a technique that allows a set of operations to be executed in a time-sliced fashion so that the overall throughput is increased. Different storage techniques outlined in this section can be executed in a pipeline fashion. It can significantly improve the overall performance (Dongara and Vijaykumar, 2003; Ashokkumar et al., 2010) and avoid idle times, at the cost of higher CPU utilization. In addition, storage operations can be pipelined with network operations such as upload and download

[4] Microsoft Sidekick and Amazon EC2 service lost unrecoverable customer data in widely publicized incidents.

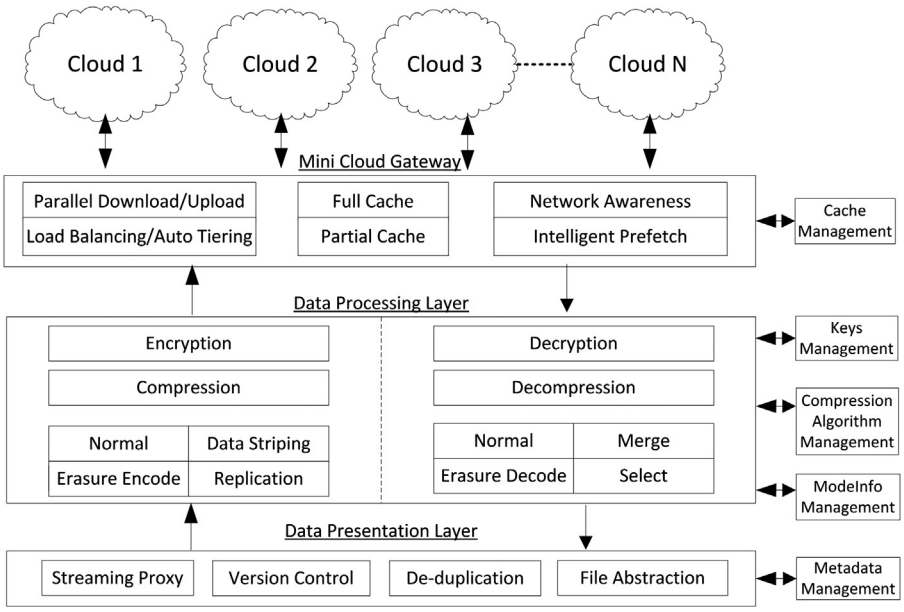

Fig. 1. Proposed overall system architecture design.

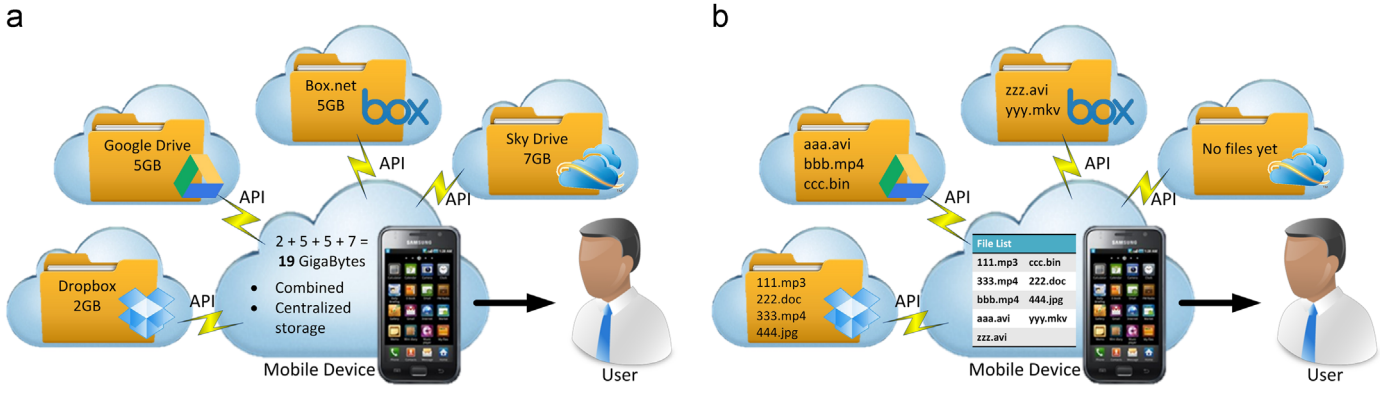

Fig. 2. (a) Aggregating multiple cloud storage services. (b) A single list view of all files in all services.

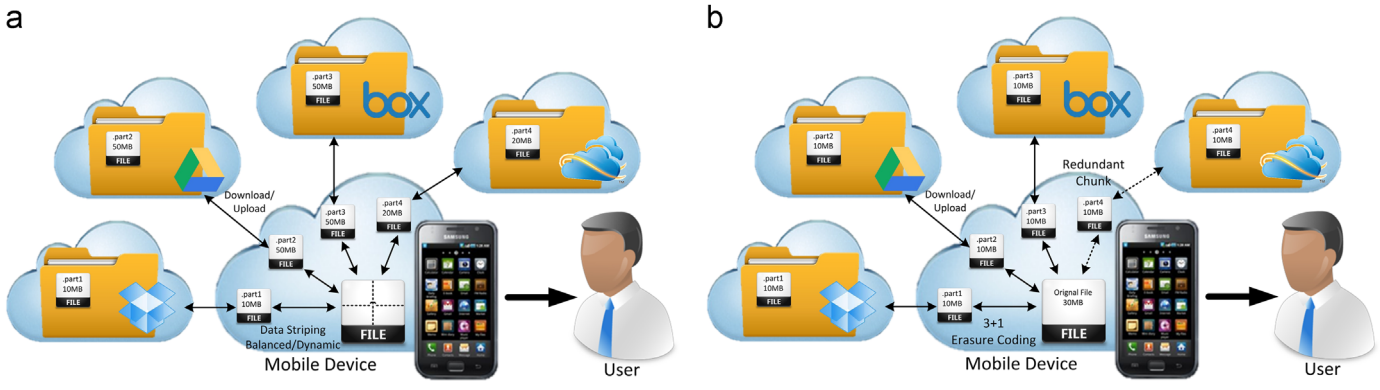

Fig. 3. (a) Data striping + parallel upload/download. (b) Erasure coding + parallel upload/download.

(Mohamed et al., 2013; Rodriguez and Biersack, 2002; Philopoulos and Maheswaran, 2001) to improve the overall bandwidth utilization (Fig. 4), similar to HTTP pipelining technique. Fig. 4 illustrates the concept where the completion time is much faster than sequential processing.

To minimize the effect of bandwidth throttling practiced by some providers, we can leverage parallel upload and download operation by transmitting different parts to different providers concurrently or vice versa. This improves the transmission speed by aggregating bandwidth from different providers and keeps the user's bandwidth saturated all the time (Mohamed et al., 2013; Rodriguez and Biersack, 2002; Philopoulos and Maheswaran, 2001). If the provider practices per-connection based throttling instead of per-account based throttling, we can establish multiple connections to both the same provider and the different providers to transmit different parts of the file concurrently (Philopoulos and Maheswaran, 2001). This improves the effective bandwidth and bandwidth utilization further.

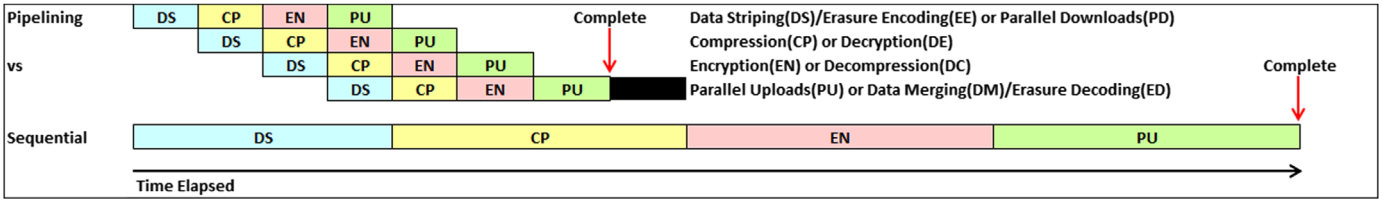

Fig. 4. Illustration of pipelining vs. sequential architecture (not according to scale).

3.5. Load balancing and automated tiering

Depends on many factors such as network condition and geographical location, different cloud services can have different network speed. Therefore, each parallel download/upload operation usually completes at different times. The effective throughput is proportional to the slowest link bandwidth. In worst-case scenario, the "tail time" (Qian et al., 2011) can be significantly long, which may lead to inefficient bandwidth utilization and high energy consumption (Google, 2012). In order to reduce this effect, we perform load balancing by assigning different priorities (automated tiering) to different providers (Mohamed et al., 2013; Rodriguez and Biersack, 2002; Philopoulos and Maheswaran, 2001) instead of using simple Round-Robin selection. When combined with data striping technique, we can load balance based on different parameters. For example, we can transmit more data back and forth between client and the provider that has (i) largest storage capacity (ii) highest link throughput (iii) largest storage capacity left unused (iv) lowest network delay and response time. File size is an important factor that can impact the result as perceived by the user. Normally, high throughput is favorable because the transfer time will dominate the entire operation. But in the case of small files, low response time is more favorable because the operation should complete almost instantly. The current remaining capacity should also be taken into consideration as there is no point to keep storing data to the fastest provider when it is low in capacity.

By automatically tiering different providers, load balancing technique can also be applied on data processed with erasure coding. Basically, the higher tiered providers with better performance in terms of throughput or storage capacity will store more erasure coded parts. For example, in a $4+2$ redundancy scheme (where $k=4$, $n=2$, $m=6$), two parts can be stored in tier-1 provider while the rest are stored in three tier-2 providers and local cache. In this case, it can tolerate one failure of tier-1 location or two failures of any of the tier-2 locations, but cannot tolerate two failures that include the tier-1 location. For the retrieval of data, only the k fastest providers are selected because k coded parts are sufficient to reconstruct the original file.

3.6. Encryption

Privacy concerns grow strongly among consumers especially after the recent disclosure of the PRISM program. Instead of relying on the security scheme provided by the service provider that may be inadequate, we employ a simple yet effective solution, which is to perform client-side encryption before storing the data in the cloud. Symmetric encryption turns a data from plaintext into meaningless cipher text using an encryption key only known to the user. With encryption, users' data are not exposed even if the service provider experiences a security breach or judicial subpoena. Client-side encryption also enables users to flexibly choose between different encryption schemes (full disk encryption, per-file encryption), encryption algorithms (DES, AES, Twofish) and key length (128 bits, 256 bits) to adjust the level of security for different providers.

3.7. Data aware compression and data de-duplication

Data can be reduced in size before transmission through the network to the cloud servers. It can improve network transmission speed and achieves bandwidth and storage quota saving. A common technique is data compression, which involves encoding the data using fewer bits of storage than the original. However, not all files can be compressed equally because different files possess different compressibility characteristics. Some files can be highly compressed (e.g. text, documents, raw media content) while some incompressible files (e.g. pdf, docx, jpeg, mp3) do not compress much (Winzip 2012). It is because the incompressible files are already in highly compressed state or are very unique in their nature, such as encrypted files. Besides, different types of file also yield different compression efficiency when using different compression algorithm (e.g. LZ77, LZW, JPEG). Hence, we can utilize a data aware compression technique in our system where compression is applied only on compressible data (Harnik et al., 2013), and the best algorithm that will yield the highest compression ratio is chosen automatically. Without this mechanism, simply applying compression on any file may introduce extra overhead without significant benefits.

Another data reduction technique is data de-duplication, which aims to reduce storage consumption by identifying distinct chunks of data with identical content and removing them. Only a single copy of the unique chunk is stored along with metadata about how to reconstruct the original files from the chunks. By performing source-based inline data de-duplication, data needing to be sent through the network can be substantially reduced. In addition, data de-duplication and data compression complement each other; they can be combined to further reduce the data size.

3.8. Caching, prefetching and network awareness

One of the inherent limitations of cloud storage is Internet access is required for it to function properly. It is because the data are stored remotely and are retrieved on demand via network connection. Due to the unstable nature of wireless network, there may be occasions when Internet connectivity is disrupted. Therefore, files can be cached locally to allow offline access during such occasions. Caching is a mechanism for temporary storing of data locally, usually to reduce bandwidth usage or perceived lag. A full cache stores the entire file where a partial cache stores only a subset of the total data. Partial cache can be combined with erasure coding mechanism to increase the fault-tolerance at no extra overhead on the cloud storage side. For example, in a $3+2$ erasure coding scheme (where $k=3$, $n=2$, $m=5$), one coded part is kept in local cache while four coded parts are stored in different cloud storage locations. In this case, it can actually tolerate two cloud servers outage at only 33% storage overhead in both the cloud storage and the local storage. It also reduces the transmission time because only 66% data need to be retrieved.

Data access pattern usually possesses spatial locality where it is likely that data nearby will be referenced in the near future. Hence, we can intelligently pre-fetch data earlier before it is needed, and hope that it will be consumed by the user later.

In addition to reducing response time, this can also avoid sending short bursts of data over the network frequently, allowing the network radio to remain in idle state more often.

Network awareness means a device is aware of which type of network it is currently utilizing (Wi-Fi or mobile network), whether the Internet is accessible, and the network behavior. It is extremely beneficial because using mobile Internet such as 3 G or LTE can be terribly expensive. In addition, if the network condition is unstable or lost, network operations may fail frequently and resulting in bandwidth wastage. This network awareness mechanism can substantially reduce the expensive mobile data charges and bandwidth wastage by simply delaying the non-priority tasks. For example, in an erasure coded file, uploading all m encoded parts immediately is not crucial. It is because only k parts must be uploaded first while the extra n parts can be delayed to a later time. Another example is when encountering unstable network conditions; the data that has not yet been written to the cloud is temporary stored in a cache. The upload process will continue automatically when Internet connection is returned to normal or when a cheaper network (Wi-Fi) connection is available.

Yet, heavy caching can consume significant storage space, given that the internal storage capacity on mobile devices is already quite limited to begin with. Therefore, caching must be done intelligently to avoid wasting precious storage space. One method is to keep only the new and frequently accessed data in the cache, while periodically performs clean-up on old and inactive data, similar to the Least Frequently Recently Used (LFRU) policy. An SQLite database is responsible for keeping track of all files, files metadata and usage frequency. It is then encrypted and periodically replicated to each of the cloud storage to allow seamless switching over to a new device while maintaining all files' information. Combining these three techniques can provide users with a smooth and seamless user experience in almost every use cases.

3.9. Progressive media streaming

Media streaming is a useful feature because users do not need to wait for the file download to be completed. Instead, users can start viewing the content immediately. However, only some cloud storage service providers offer media streaming capability, with a limited number of supported formats. To overcome this limitation, we utilize progressive streaming so users can stream from any provider even though the provider does not actually support media streaming feature.

3.10. Exploiting file versioning

Most of the cloud storage services include file versioning, an useful feature where users can restore unintentionally deleted files, or revert the files to the previous version in case of data corruption. For free subscription, it is limited to 30 days or 100 revisions only. This feature is interesting because the deleted data can be easily recovered, but it does not count towards the storage quota. It is somewhat similar to the “recycle bin” concept in desktop environment except the data stored inside can be theoretically unlimited. Hence, users can exploit this feature to temporary boost their effective storage space. This can be easily achieved by temporary deleting some large files to regain some storage capacity, and then recover those deleted files at a later time. This feature can be further exploited to achieve an unlimited storage capacity illusion, by repeating the process indefinitely. To prevent the files stored in “recycle bin” get deleted forever, it must be recovered at least once before the expiration window (usually 30 days). This process can be easily done by restoring the file and immediately delete it again, and then the file will last for another period of expiration. This process can also be done programmatically by the application and does not require any user intervention. However, heavy exploitation of this feature maybe considered as unethical and does not comply with the terms and regulations of certain service providers.

4. Implementation

In this section, we delve more deeply into the specifics of our implementation and present the actualization of our proposed solution, through a prototype implementation on Android mobile devices, although it should be applicable to other platforms as well. As resources on a mobile device are rather limited compared to its desktop counterparts, the performance might be degraded when performing CPU intensive task. Thus, we must carefully consider the impact and overhead incurred by our system to justify the feasibility of deploying it.

4.1. Overall implementation

Our prototype (Fig. 5) implements four popular cloud storage services according to a survey (Alan 2013), which are Dropbox, Google Drive, Box and Skydrive. Our prototype included most of the techniques mentioned in Section 3 except data de-duplication (Section 3.7), automatic selection of best compression algorithm (Section 3.7), intelligent pre-fetching (Section 3.8) and automatic restoration in exploiting file-versioning (Section 3.10), which will be further explored in our future research work. The current prototype is able to connect to multiple cloud storages services, indexing files located in different cloud storage system, and present it to users in a single list view under the same namespace. Simple CRUD operations or advanced storage techniques such as data striping, erasure coding, compression, encryption, parallel network operations, pipeline data processing and caching are also possible. As a prototype, the “exploit file versioning” feature proposed in Section 3.10 is implemented on only one of the cloud storage provider (Dropbox).

In our prototype, we utilized JigDFS (Bian and Seker, 2009) library for erasure coding. The other libraries used in our prototype such as Bouncy Castle cryptography library, Java Zip library, Java NIO and SQLite library are already included among Java SDK and Android development tools. To connect to cloud storage service by different providers, we use Dropbox SDK, Box SDK, Google Drive SDK and Microsoft Live SDK for Android.

4.2. Android application

Our solution is implemented as a software application that can run on any compatible device instead of file system due to several reasons: (i) It is easier to develop a proof-of-concept prototype. (ii) It is easier to distribute the solution to a vast amount of users via .apk file in Google Play market, and it is easier for users to perform the installation. iii) Mounting Filesystem in Userspace (FUSE) requires root level access, which means users need to “root” their device and void the manufacturer warranty. (iv) Most of the official Android ROM does not include FUSE support except some unofficial third party custom ROM. It means users need to flash custom ROM or kernel, and that is a relatively difficult process especially for average non tech-savvy users. Flashing custom ROM or kernel also possesses high chance of bricking the device, rendering the device unusable. On the other side, there are several drawbacks to our approach, which are: (i) It is not mountable like file system and therefore not capable of full storage virtualization, which means data can be accessed only using the application.

Fig. 5. Screenshots of the prototype implementation on Android device.

(ii) Performance may not be as good as file system implementation. Nevertheless, we believe the advantages outweigh the drawbacks in most circumstances. For unified cloud storage access, all files' attributes and metadata are fetched from each of the cloud storage in parallel using the specific provider's API. Then, they are combined under a single namespace and finally presented to user in a single list view (Fig. 5). Additionally, users can perform simple file manipulation operations on any of the files such as Create, Rename, Update and Delete (CRUD). Searching, filtering and sorting can also be performed on all these files even though they are actually stored in different cloud storage. These metadata are also cached locally to avoid frequent refresh if no modification has been made.

4.3. OAuth 2.0 protocol

Our implementation does not require any extra sign up on third party websites, except for the cloud storage services that a user wishes to use. The authentication process is performed directly with each of the cloud storage providers using OAuth 2.0 protocol, which means our system do not keep track of any user's id and password. Our system only stores the OAuth 2.0 tokens that can be revoked at any time. The authorization process is a one-time only process that only needs to be performed once by users during installation, as shown in Fig. 6.

4.4. Load balancing implementation

The optimum ratio should be calculated based on parameters such as effective bandwidth, round-trip time (RTT) and bandwidth delay product (BDP) of each provider. However, our load balancer only uses the effective network throughput because the other parameters (RTT and BDP) cannot be easily obtained by using vendor specific API. We denote $S_i \ldots S_n$ as the effective network speed for i to nth provider. Each provider has its own download speed ($S_{d,i}$) and upload speed ($S_{u,i}$).

$$S_i = \frac{(S_{d,i} \times f_d) + (S_{u,i} \times f_u)}{f_d + f_u} \quad R_i = \frac{S_i}{\sum_i^n S} \quad \text{where} \quad R_i \leq 1$$

where f_d and f_u are the frequency of download and upload, respectively. Then we find the effective ratio for each provider $R_{i \ldots n}$ and use it to load balance across multiple providers. Due to the fact that network condition is highly unstable and hardly predictable, a running average of the ratio is also being dynamically updated every time a network operation is completed, so that it can adapt better to network conditions or geographical location changes. The ratio

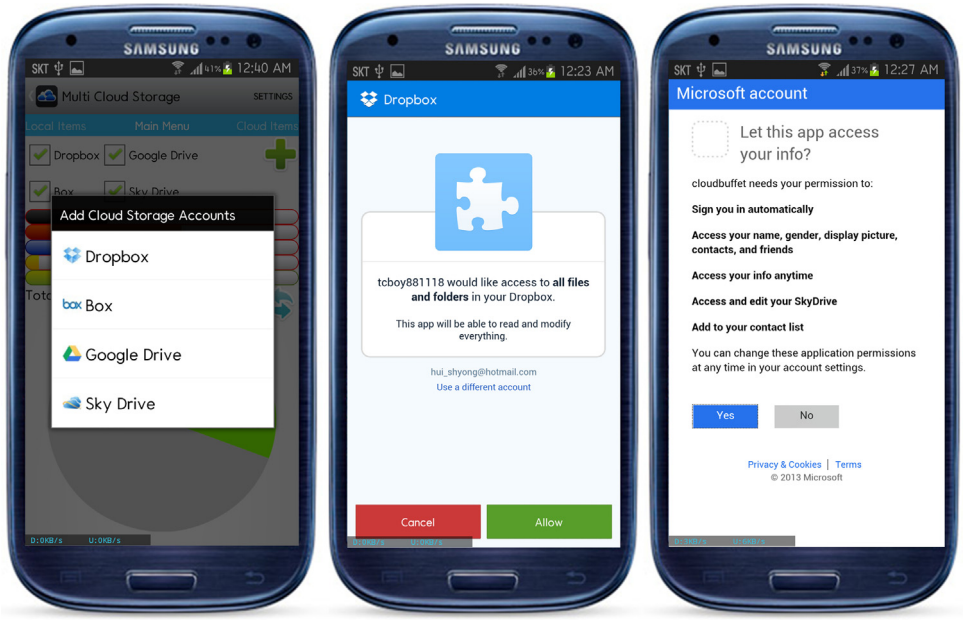

Fig. 6. Screenshots of performing OAuth2 authentication with different providers.

should be more biased toward download bandwidth, as typical users tend to download more often than upload in normal use cases. They are updated regularly and being stored in an SQLite database. Since our method is based on historical performance data, it does not guarantee the theoretical best performance, but it should provide reasonably good performance as perceived by users. Another concern is regarding load balancing data processed with data striping which the upload ratio will become the limiting factor because the uploaded portion in each provider must be retrieved in full next time. Most of the providers are having symmetric upstream/downstream bandwidth so we can safely neglect this factor.

4.5. Parallelization and pipelining implementation

To achieve high performance, we mainly utilize the strength of parallelization and pipelining based on the fact that file can be divided into smaller chunks and each chunk can be processed independently. Besides improving the processing performance and network throughput, parallelization can also reduce the cost of retransmission in case of a network failure or disconnection. It is because only those chunks which have failed to transmit must be retransmitted again instead of the whole file.

Due to the asymmetric processing time for each stage in the pipeline, there may be a lot of data buffer held in the memory at any given time. However, older Android devices usually have low memory capacities (512 MB) and limited heap size (64 MB). Therefore, in our implementation, the output is written into a temp file before passing it to the next stage in the pipeline. This severely limits the speedup gained from pipelining so that better optimization such as increasing the heap size on newer Android devices and better scheduling are to be explored in future research. For this reason, a low chunk size (2 MB) for data striping and erasure coding is chosen for the prototype implementation.

4.6. Erasure coding and checksum

For erasure coding, we utilize the JigDFS library (Bian and Seker, 2009), which is the Java implementation of the Jerasure library (Plank et al., 2008) based on the Cauchy Reed Solomon algorithm. Study (Plank et al., 2009) shows that the Jerasure library is superior to other erasure codes libraries, in terms of speed and efficiency. There are essentially three main operations in the encoding process, which consist of: (i) Hashing the entire file for a fingerprint. (ii) Encoding the file to produce $k+n$ chunks. (iii) Hashing each encoded chunk for a fingerprint. The operations are similar but in a reversed way for decoding process: (i) Hashing each retrieved chunk for a fingerprint and compare it with the fingerprint obtained earlier for integrity checking. (ii) Decoding k encoded chunks to reconstruct the original file. (iii) Hashing the reconstructed file for a fingerprint and comparing it with the fingerprint obtained earlier for integrity checking. The hashing mechanism provides data integrity checking where any data tampering or data corruption can be detected if the newly calculated fingerprint does not match that of the previously stored fingerprint. However, these hashing processes are very costly in terms of time and processing power, especially on mobile devices with sparse resources. Therefore, we utilize Cyclic Redundancy Check (CRC) as default checksum algorithm instead of cryptographic hash function (SHA) for better performance and lower overheads, while sacrificing some level of data integrity. CRC can protect against data corruption and unintentional data tampering, which is sufficient in typical single user cloud storage usage. SHA-1 or SHA-256 is available and optional.

Unlike typical erasure coding, we do not encode a file as a whole. Instead, we divide a file into smaller slices and encode each slice. This approach can avoid reading a large file entirely into memory, given that the memory and heap size on mobile device is limited to begin with. For example, we uses a default of 2 MB chunk size in a $3+1$ erasure coding scheme, where $k=3$, $n=1$ $m=4$. It means the original file is divided into several 6MB slice ($k \times 2$ MB) and the remaining part is equally divided by k, as shown in Fig. 7. Since sliced encoding will yields many small encoded chunks (total slices $\times$ m encoded chunks), it may cause extra overheads (HTTP handshaking) during network transmission when uploading to cloud storage server (Fig. 7). Hence, it is also rational to merge these chunks into a bigger chunk before uploading, which we have denoted as a container. For retrieving, as soon as the first k chunks in the first slice have arrived, they are passed to the decoding module for decoding, while the k chunks in the second slice continue to be downloaded in the background. Each of the decoded blocks is then appended to the previous block

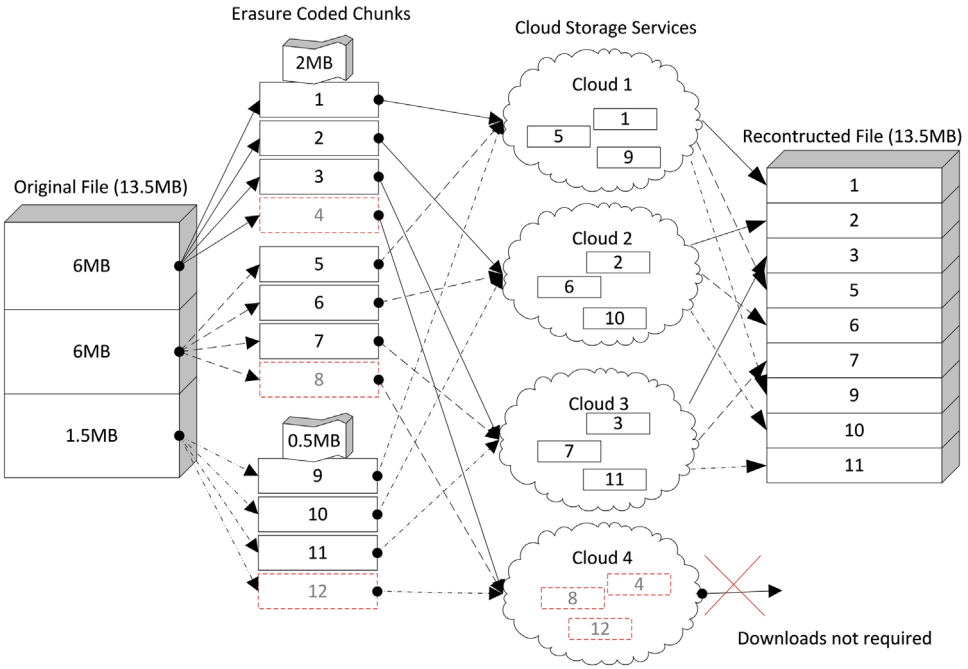

Fig. 7. Coded blocks are transmitted directly to/from cloud storage to allow parallel acceleration.

to reconstruct the original file. Similarly, to retrieve a striped file, all related blocks are downloaded in parallel and merged together.

4.7. CPU and network energy consumption

Even though performing CPU intensive operations such as compression, encoding and encryption on mobile devices may consume a lot of battery energy, the energy consumed by network component are often comparable to or higher than the energy consumed by the CPU (Pathak et al., 2012), especially on mobile network such as 3 G or LTE. In addition, a slow network operation drains more energy by keeping the wireless radio in high power state and incurs long "tail times" (Google, 2012; Alexandre et al., 2011; Qian et al., 2011). In contrast, high data throughput allows network operations to complete faster, allowing the device switches its wireless radio into idle state as quickly as possible. In idle state, power consumption is approaching zero and is negligible compared to keeping the wireless radio constantly in low power or full power state (Google, 2012). In short, it is better to transmit the data over a short period and allow the radio to switch into sleep state instead of transmitting the equivalent in data over a longer period and keeping the radio awake. Consequently, we argue that our solution could result in battery power savings in most circumstances.

4.8. Progressive streaming implementation

Progressive streaming is implemented with a lightweight local proxy server on the device itself. It performs downloading/buffering in the background and then pushes the content to media player as soon as the data arrives, as depicted in Fig. 8. Our prototype is currently able to stream from any cloud provider, but it does not support processed (encrypted or encoded) file.

4.9. Encryption and compression

Symmetric encryption (AES-256) and cryptographic hash function (SHA-1) are implemented in our prototype. User provides a password which is then derived into encryption key using Password-Based Key Derivation Function (PBKDF2) (Kaliski, 2000). For data chunks processed with data striping or erasure coding, each chunk is treated as a single file and encrypted independently. Similarly for data compression using Zip, each file or each chunk is compressed independently. We believe this is a simpler approach but we are keen to explore more effective methods in the future research work, such as interleaved encryption.

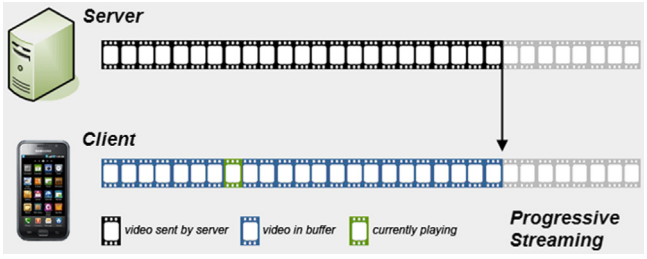

Fig. 8. Illustration of progressive media streaming.

5. Performance evaluation

In this section, we present the performance evaluation of our system implementation which tries to answer three main questions: (i) What are the advantages in terms of performance, cost and availability of using multiple providers? (ii) Does the performance gain outweigh the overheads? (iii) Does it solve or alleviate the limitations outlined in Section 2?

We evaluated the system on four Android smartphones from different generations, launched in year 2010, 2011, 2012 and 2013, respectively. They consist of a low-end Galaxy S (Single-core processor, 512 MB RAM), one middle level Galaxy Nexus (Dual-core processor, 1 GB RAM), one high-end Galaxy S3 (Quad-core processor, 2 GB RAM) and a latest Galaxy Note 3 (Quad-core processor, 3 GB RAM). To simulate a real-world environment, we evaluated our system on two networks: SK-Telecom (SKT) LTE mobile network and Korea Telecom (KT) residential Internet

network via a Wi-Fi connection. Bandwidth benchmark with nearest server using Ookla speed test service[5] during non-peak hour resulted in an average download/upload bandwidth[6] of 37/34 Mbits/s and 29/23 Mbits/s, respectively. We report the results of several different performance tests in this section. Each test was performed five times and the average time consumption is recorded, unless otherwise specified.

5.1. Comparison between different cloud storage providers and a combined approach utilizing multiple cloud storage providers

Fig. 9 illustrates the free storage capacity, pricing and performance difference when utilizing our prototype system with different providers, normalized with respect to the highest factor. We measure the time consumption and calculate the network throughput for uploading and downloading a 25 MB test file with each of the different provider and then with multiple providers in parallel. For parallel operations, we stripe the data into multiple blocks of 2 MB each and then send them equally via multiple connections to/from each provider in parallel. From Fig. 9, we noticed that although SkyDrive offers the largest storage capacity for free subscription and cheapest pricing for paid subscription (monthly cost of $0.0416 per GB); the network performance is comparatively lower than other providers so there is clearly a trade-off between lower cost and better performance. On the other hand, Google Drive's paid subscription offers much better network performance than SkyDrive but is only slightly more expensive. Finally, Box charges the highest monthly subscription fee and offers the highest network bandwidth, albeit lower than the combined bandwidth of our approach.

From the results of our combined approach, it is shown that the effective storage capacity and network performance are improved significantly by aggregating resources from multiple cloud storage services. One major advantage is that the per-connection based bandwidth throttling can be bypassed by using multiple connections. This is especially obvious in the case of SkyDrive. With our approach, we reach a combined bandwidth of 4.2 MB/s for uploads and 4.3 MB/s for downloads. These peaks occur at this value due to different limitations on the device and the network itself (802.11n or LTE). Theoretically, the maximum throughput is expected to be higher (more than 6 MB/s) on better hardware such as on 802.11ac devices or better network such as LTE-Advance with carrier aggregation (CA). With our approach, the effective monthly cost is also down on average from the most expensive provider. In this case, the Box pricing of $0.4 per GB can be reduced to $0.1479 per GB, which is roughly 60% cheaper. Although the combined pricing ($0.1479) is still higher than the other three providers, users can actually benefit from several advantages discussed in Section 3 (avoiding vendor lock-in, lower migration cost, fault-tolerance and improved performance) which are not possible when using a single provider. In addition, these advantages basically come for free if the said user is only using free cloud storage services. User gets a total storage capacity of 19 GB without paying a single cent and it can be further expanded by adding more service providers.

5.2. Performance evaluation of different data processing techniques

To evaluate our prototype implementation, we measure the time taken to process different storage operations on a 100 MB incompressible file retrieved from the Linode[7] website. A 2MB block size is chosen for both data striping and erasure coding with a 3+1 redundancy scheme ($k=3$, $n=1$). For compression and encryption, we use DEFLATE compression algorithm and AES-256, respectively. The experiments are repeated on the four aforementioned mobile devices of different specifications and the results are shown in Fig. 10. The results show that these operations do not cause significant processing time, and the time consumption decreases dramatically on newer device. The processing speed is surprisingly fast such that all operations take less than 10 s to complete on the latest model device, with an exception on the compression technique. We conclude that the data processing is not likely to become a bottleneck until wireless gigabit Internet to become more common.

5.3. Processing performance on mobile devices from different generations

With the results from previous experiments, we calculate the processing performance by dividing the file size over the time consumption. In general, the performance improvement is approaching two times faster vis-a-vis each device from each generation, as shown in Fig. 11. This improvement does not depend solely on a single element; it reflects a combination of mobile technology improvements in terms of processor speed, memory bandwidth and NAND flash speed. Therefore, it is safe to assume the performance may be improved subsequently over the coming years, as the performance of mobile devices continues to improve and as the overheads incurred by data processing become lower and less significant, which makes our proposed solution of applying storage techniques on the client-side more feasible.

5.4. Comparison between sequential processing and parallel processing

To demonstrate the improvement in performance of parallel and pipeline processing using a combination of different data processing methods discussed in Section 3, we measured the time consumption and compared the data obtained therefrom with results of sequential processing. The results of sequential processing are basically a summation of the results collected from previous experiment, where we assume there is no delay between executions of the processes. We use the same parameters as previous experiments, i.e. small block size of 2MB, 3+1 erasure coding scheme, AES-256 and DEFLATE algorithm. The results are shown in Fig. 12 (results for Galaxy S3 only). It is shown that data processing time is reduced significantly thanks to parallelization and pipelining that allows each chunk of data to be processed concurrently, which in turn can optimize processor utilization and improve the execution time. The performance could be further improved with a better implementation (by processing everything in memory instead of writing into a temp file first). Nevertheless, the amount of speed-up is will be limited according to Amdahl's law.

5.5. Evaluation of erasure coding performance by varying different parameters

The erasure coding performance is influenced by many factors. The main contributors are the degree of redundancy, the checksum algorithm and the device's processing power. We evaluate the performance by measuring the time consumption for encoding and decoding a 100 MB incompressible file with different parameters. We uses 2 MB block size and vary the erasure parameters (k and n) and checksum algorithm for different benchmarks. They are repeated on three different devices equipped with multi-core processor only (e.g. Galaxy Nexus, Galaxy S3 and Galaxy Note 3). The results are shown in Fig. 13. It is shown that the time consumption of performing erasure coding is surprisingly low even on mobile devices with limited computation power. By varying redundancy level and the checksum algorithm (CRC-32 or SHA-1) for data integrity checking, the time

[5] ⟨http://www.speedtest.net/⟩.
[6] The effective bandwidth may vary according to location, network condition and device's capability.
[7] ⟨www.linode.com/speedtest⟩.

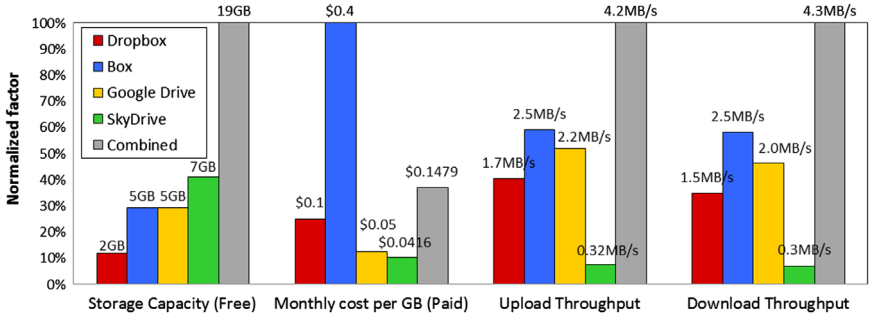

Fig. 9. Comparison between different cloud storage service providers (performed on Galaxy S3).

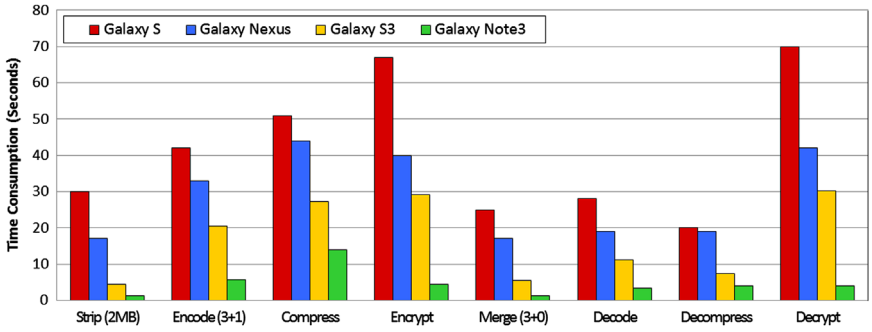

Fig. 10. Time consumption of different data processing techniques on four Android devices (100 MB file).

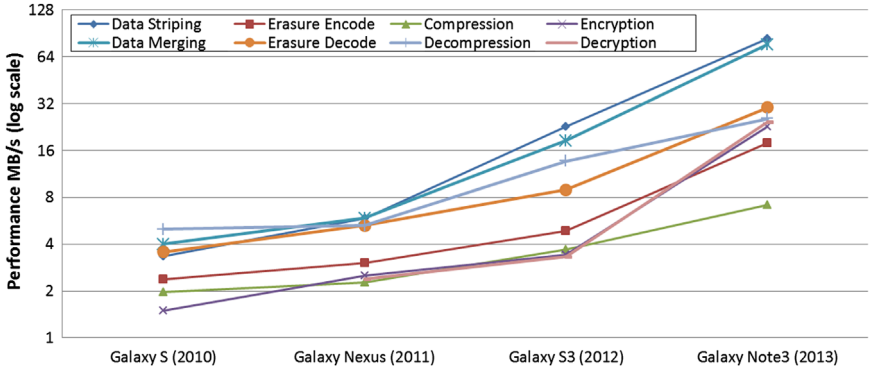

Fig. 11. Performance on different devices.

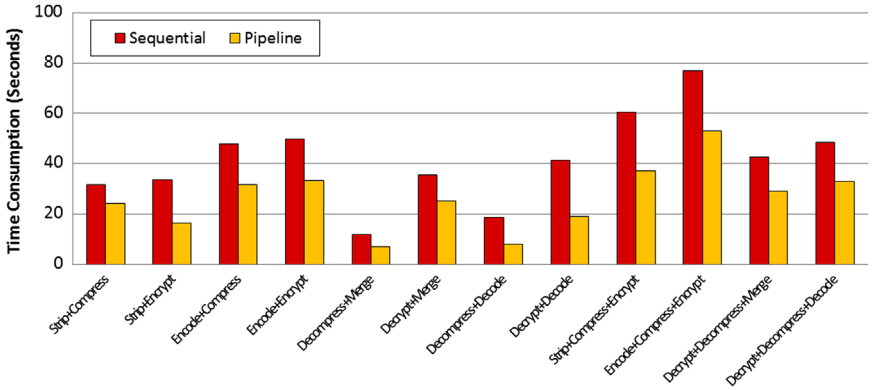

Fig. 12. Time consumption of sequential vs. pipelining processing (performed on Galaxy S3).

consumption increase accordingly. It is revealed that the main cause for the significant amount of processing time is the SHA-1 hashing mechanism, so our proposal of replacing SHA-1 with CRC-32 and trading data integrity for speed is effective.

5.6. Evaluation of network performance

To evaluate the network performance and to observe how our approach can improve the bandwidth utilization, we performed

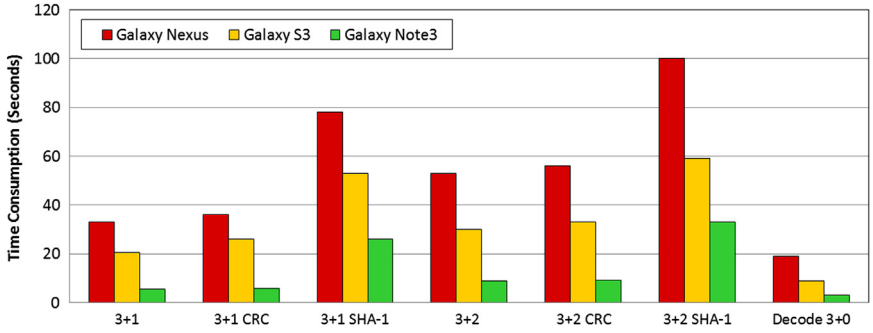

Fig. 13. Erasure coding a 100 MB file with different parameters ($k=3$, $n=1$, $n=2$) and checksum algorithm.

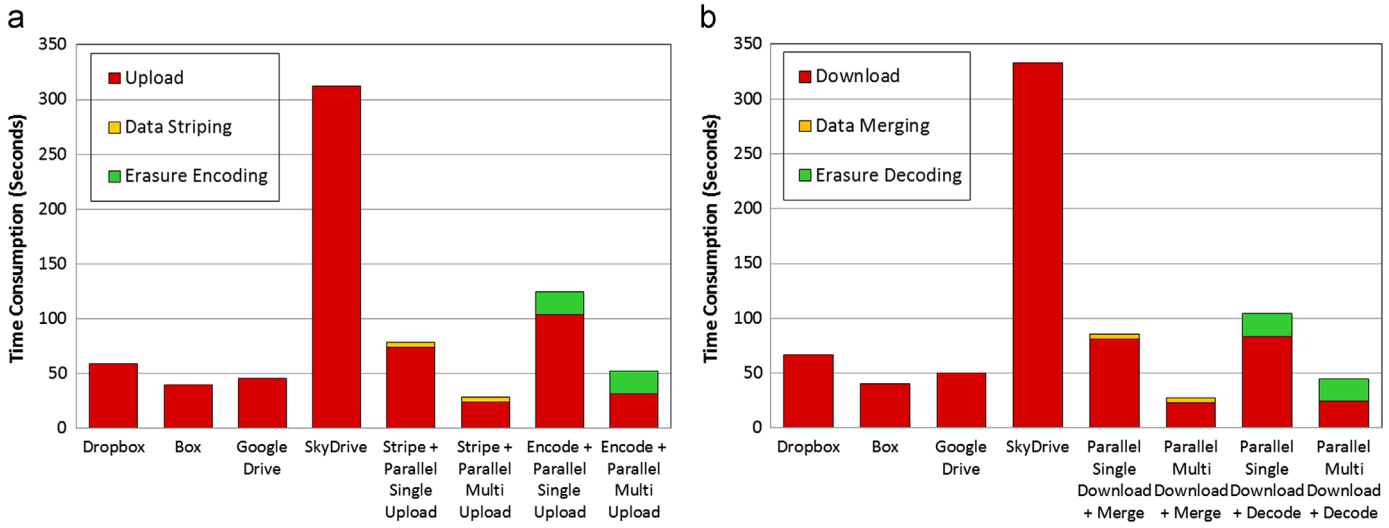

Fig. 14. Time consumption of upload/download operations on Wi-Fi network (100 MB file). (a) Upload (b) download.

a series of benchmark of using a single provider and multiple providers (our approach). The results are shown in Fig. 14. The benchmarks were run on Galaxy S3 by transferring a 100 MB file between the cloud storage services and the device. For the parallel operations, we use a 2 MB block size, a 3+1 redundancy scheme and symmetric transfer for each provider out of the four providers we utilized here. The two parallel operations, denoted as "Parallel Single" and "Parallel Multi" are slightly different. In "Parallel Single", there is only single connection to each provider, so there are only a maximum of four connections at any given time. In "Parallel Multi", there are multiple connections to each provider, so there are typically more than four connections at any given time, as a way to bypass the per-connection based bandwidth throttling imposed by service provider.

Based on the results of these tests as presented in Fig. 14, it is shown that the overheads incurred by data striping and erasure coding process are actually remarkably little compared to the overall time for network operations. We attribute this to the strength of parallelism and pipelining. The network I/O will almost always be the limiting factor rather than the storage I/O (at least not until wireless gigabit Internet is more widespread), hence the time consumption is dominated by the network operations. Using multiple providers (Parallel Single) is slower than using a single fastest provider because it needs to tolerate the slowest provider (in this case SkyDrive) where the time consumption is actually being averaged down. By establishing multi connections (Parallel Multi) to each provider, this bottleneck can be avoided. As a result, the time consumption can be further reduced, resulting in a performance faster than any single fastest provider does. It also results in higher bandwidth utilization and battery energy saving for the tested device as discussed in Section 2.1. Therefore, the incurred overheads are actually compensated by the performance gained.

5.7. Dynamic load balancing

We trace the network pattern using Dalvik Debug Monitor Server (DDMS) tools provided in Android SDK. Fig. 15 represent the general pattern of bandwidth utilization when performing network operations back and forth between multiple cloud storage services simultaneously. Each colored region indicates the bandwidth utilization of network connections to a different cloud storage service, as stated in the figure. We can observe that by using a dynamic ratio load balancing favoring the faster provider, the overall network throughput is higher. This ensures that all of the download and upload operations complete within a nearly similar time frame, and hence the "tail time" is effectively mitigated. As a result, the wireless radio of the tested mobile device can switch to idle state earlier, which could lead to lower battery energy consumption.

5.8. Discussions

A single interface to access all the cloud storages as a centralized storage is essential to users because of its intuitiveness and simplicity. It allows agile management of data across multiple services. This also allows a user to aggregate resources from multiple service providers, increases the storage capacity by said user at zero cost (for free subscriptions) or reduces the migration costs (for paid subscription). Using multiple service providers enables us to apply essential storage

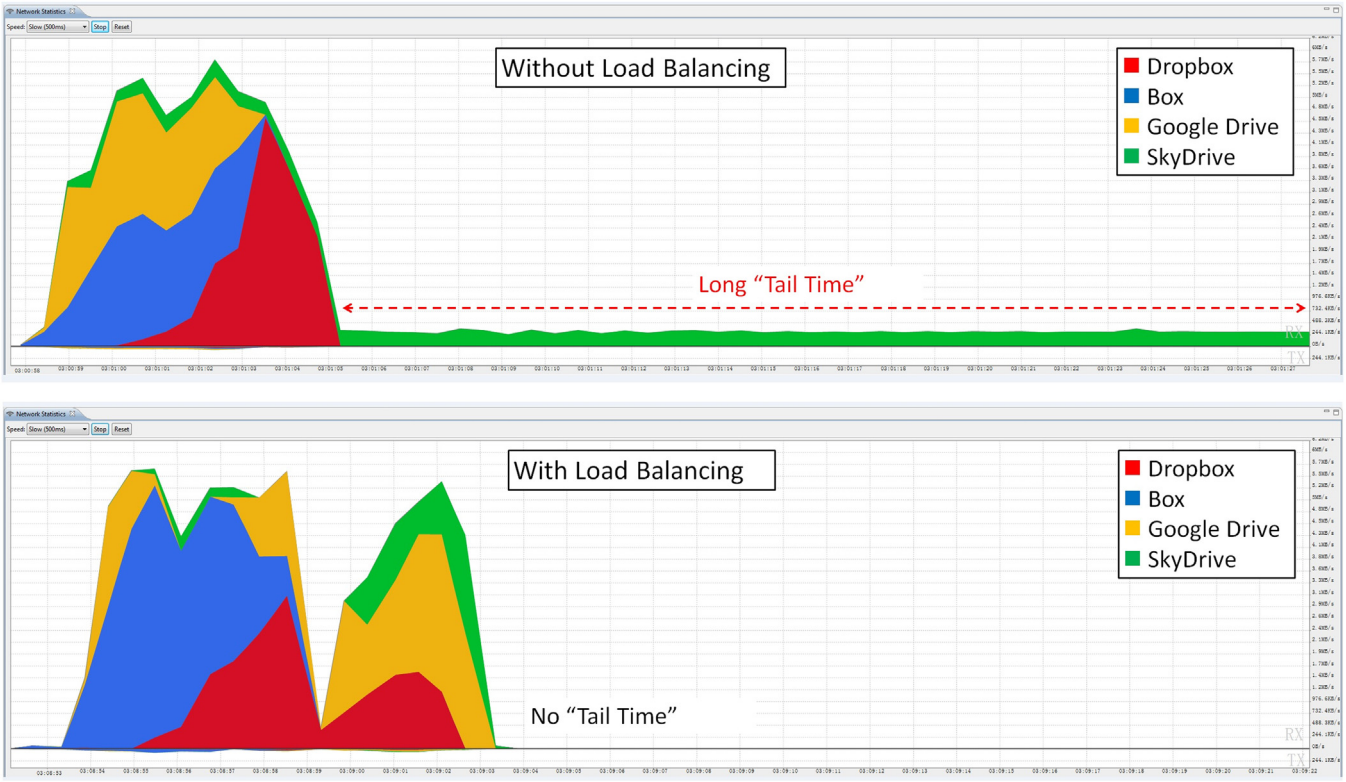

Fig. 15. Parallel downloads without/with load balancing.

techniques on the client-side, but it can cause overheads especially on resource-constrained mobile devices. Nevertheless, today's mobile devices are equipped with multi-core processor and a large amount of RAM. By exploiting parallelization and pipeline processing, overheads are reduced while performance is improved. Our proposed solution is thus economically viable.

There are still a few limitations which exist and are as yet to be mitigated via our current solution. Users are required to sign up for multiple cloud storage services manually. Although this is a simple and one-time only process, it may appear to be troublesome for certain people. Otherwise, if users are already using multiple cloud storage services, only OAuth2 authorization process is required to start using the system. In our experiment, the process of authorizing four cloud storage services took less than one minute. Only new files uploaded via this system can benefit from the techniques applied in data processing layer. Hence, a conversion tool that will convert the previously stored and unprocessed files into processed files is to be explored for by future research work. Users must also aware that the processed files (either striped or erasure coded) cannot be retrieved correctly when using the web interface on the official website. This side effect is actually the intention of an information dispersal algorithm (IDA) which efficiently hides the data from any single service provider. There are clearly trade-offs between using different storage techniques (data striping vs. erasure coding vs. replication) and users must choose depends on their level of own demand. Since our solution is not developed as file system, traditional file system semantics and consistency are not provided. Files can be only access via this system, or can be access locally after they are stored locally, similar to typical cloud storage services.

6. Related work

Aggregating multiple cloud storage services is not entirely new idea as people have been using multiple email accounts for years. However, a problem arises when user needs to explicitly find their files as if it were a needle in a haystack. Third party services such as StorageMadeEasy, Otixo, Primadesk aim to overcome this problem, although their efforts seem incomplete in doing so (Geeks, 2013). These services merely provide a single interface to simplify the management of multiple cloud services, but the unorganized files are still scattered across different cloud locations. Furthermore, users are required to sign up with on a contractual basis to another service provider abiding by its terms and conditions. These services store user information and authorization token, which means they have access to user's data in the cloud, and this may raise privacy concerns. Moreover, network connection is routed to their service (similar to proxy), which will raise other issues like performance bottlenecks, service outages and man-in-the-middle security concerns. Finally yet importantly, those services cost money. In short, these services do not solve any actual problems but rather incur additional costs upon their users.

Therefore, several studies (Ion et al., 2011; Slamanig and Hanser, 2012; Spillner et al., 2013; Seiger et al., 2011; Bian and Seker, 2009; Plank et al., 2008; Plank et al., 2009; Hu et al., 2012; Resch and Plank, 2011; Bessani et al., 2011; Bowers et al., 2009; Mu et al., 2012; Cachin et al., 2010) proposed relevant approaches to improve cloud storage services from different aspects, by applying redundancy techniques on top of multiple cloud services, commonly known as cloud-of-clouds (Slamanig and Hanser, 2012) or the intercloud (Cachin et al., 2010) approach. Redundancy techniques have long been utilized in local and distributed storage system, but it has just recently caught on with the blooming growth of cloud storage. The redundant array of inexpensive disk (RAID) is the most common means typical of this technique. There are several different RAID levels where RAID-0 is strictly equal to replication, while RAID-5 is basically erasure coding with $4+1$ scheme. However, each relevant study focused on different objectives. In terms of cost, RACS (Abu-Libdeh et al., 2010) was the pioneer research that utilizes erasure coding and has a working

prototype implementation. RACS focuses only on increasing availability and avoiding vendor lock-in issues by reducing migration cost. It also relies on a proxy which may become a bottleneck and single point-of-failure. On the other hand, NCCloud (Hu et al., 2012) focuses on maintaining fault tolerance while reducing the repair cost by introducing F-MSR coding scheme, which aims to achieve cost-effective repair for a permanent single-cloud failure. Our approach not only aims to lowering the cost by averaging the subscription prices of multiple providers but also improving the free cloud storage services involving no costs at all.

There are more relevant research studies that focus on the security side of cloud storage services. DEPSKY (Bessani et al., 2011) focuses on confidentiality, integrity and availability (CIA) of information stored in the cloud through encryption, encoding and replication. Similarly, HAIL (Bowers et al., 2009) acts as a distributed cryptographic system, which allows cloud servers to compute the proof of availability and integrity of the stored data. JigDFS (Bian and Seker, 2009) focuses on a file system with strong encryption and a certain level of plausible deniability. AONT-RS (Resch and Plank, 2011) described a new dispersal algorithm that can achieve high security with low computational and storage costs. Taking an overall perspective, RAOC (Spillner et al., 2013) focuses on creating optimal cloud storage systems by considering non-functional properties. SecSCIE (Seiger et al., 2011) proposed a system for use within enterprises where many users connect to a central proxy. μLibCloud (Mu et al., 2012) is most similar to our approach, but their choice of cloud providers are mainly targeted at the enterprise level instead of average household consumers. In our design, we follow RAOC layered and modular design so that new features can be added easily but we have proposed additional techniques that have never been applied in the cloud-of-clouds approach before, such as data de-duplication and exploitation of file versioning.

Most of all, the aforementioned approaches have used erasure coding or information dispersion algorithm (IDA) as outlined by Rabin (1989) as the basis. Unlike existing studies, our works aim to improve the system from almost every aspect with a focus on overall performance and minimizing the incurred overheads. Clearly, our main research target also differs from previous studies that focus on enterprise or desktop environment where CPU and network resources are not a concern. Our study focuses on mobile devices that allow ubiquitous access to cloud storage but are resources constrained in terms of computing power, network resources and battery power. In addition, our solution is implemented as software appliance that can be easily distributed and installed on the device itself instead of requiring dedicated hardware or proxy (Abu-Libdeh et al., 2010; Seiger et al., 2011).

7. Conclusion and future works

Herein we have presented the prototype of a multi cloud storage middleware application for mobile devices. It allows users to enjoy better cloud storage services from multiple providers at zero cost with only minimal efforts necessary on the client-side. The key insight of this paper is that those limitations of individual cloud storage can be easily overcome by applying essential storage techniques on the client-side. Generally, it has addresses current limitations to cloud storage services and delivers improved performance in regard to several aspects of service including speed and energy consumption. It allows users to take advantage of each service provider's strongest features while minimizing any weaknesses. Our results show that it is feasible to be applied even on resource-constrained mobile devices at the cost of minimal overheads. Our system is available on Google Android market (Multi Cloud Storage Prototype ⟨https://play.google.com/store/apps/details?id=com.tcboy.multi.cloud.storage.system&hl=en⟩). It can be distributed to and installed by public users easily.

In future research, we aim to improve the overall performance with a better implementation (native code or Renderscript). Other techniques that have been proposed in Section 3 but that have as yet to be implemented, such as data de-duplication and data-aware compression, are planned for the near future. More choices regarding providers will be added to the cloud gateway layer, and it will be upgraded to the more powerful REST or SOAP protocol. We are also interested to look into multiple users scenario and reducing the repair cost in the case of service outages. Finally, a more thorough performance analysis on the network and battery utilization will be performed with ARO tools (Alexandre et al., 2011; Qian et al., 2011).

Acknowledgments

This work was supported in part by the National Research Foundation of Korea under Grant 2011-0009349. Thanks are also due to all reviewers for their comments and recommendations, which have greatly improved the manuscript.

References

Abu-Libdeh, H, Princehouse, L, Weatherspoon, H. (2010, June). RACS: a case for cloud storage diversity. In: Proceedings of the first ACM symposium on cloud computing. ACM; pp. 229–240.

Alan, H. (2013) Five best cloud storage providers [online], available ⟨http://lifehacker.com/five-best-cloud-storage-providers-614393607⟩ [accessed 23 November 2013].

Alexandre, G, Subhabrata, S, Oliver, S. (2011) A call for more energy-efficient apps [online], available ⟨http://www.research.att.com/articles/featured_stories/2011_03/201102_Energy_efficient?fbid=plCYzUNB0e3⟩ [accessed 9 December 2012].

Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, et al. A view of cloud computing. Commun ACM 2010;53(4):50–8.

Ashokkumar S, Karuppasamy K, Srinivasan B, Balasubramanian V. Parallel key encryption for CBC and interleaved CBC. Int J Comput Appl 2010;2(1):21–5.

Bessani, A, Correia, M, Quaresma, B, André, F, Sousa, P. (2011, April). DepSky: dependable and secure storage in a cloud-of-clouds. In: Proceedings of the sixth conference on computer systems. ACM; pp. 31–46.

Bian, J, Seker, R. (2009, March). Jigdfs: a secure distributed file system. In: Computational intelligence in cyber security, 2009. CICS'09. IEEE symposium on IEEE; pp. 76–82.

Bowers, KD, Juels, A, Oprea, A. (2009, November). HAIL: a high-availability and integrity layer for cloud storage. In: Proceedings of the 16th ACM conference on computer and communications security. ACM; pp. 187–198.

Cachin C, Haas R, Vukolic M. Dependable storage in the intercloud. IBM Res 2010;3783:1–6.

Dongara, P, Vijaykumar, TN. (2003, March). Accelerating private-key cryptography via multithreading on symmetric multiprocessors. In: Performance analysis of systems and software, 2003. ISPASS. 2003 IEEE international symposium on. IEEE; pp. 58–69.

Gartner (2012) Gartner says that consumers will store more than a third of their digital content in the cloud by 2016 [online], available ⟨http://www.gartner.com/newsroom/id/2060215⟩ [accessed 9 December 2012].

Genius Geeks (2013) Manage multiple cloud storage services efficiently with these tools [online], available ⟨http://geniusgeeks.com/manage-multiple-cloud-storage-services-efficiently⟩ [accessed 23 March 2013].

Google (2012) Optimizing downloads for efficient network access [online], available ⟨http://developer.android.com/training/efficient-downloads/efficient-network-access.html⟩ [accessed 9 December 2012].

Han Y. Cloud computing: case studies and total cost of ownership. Inf Technol Libr 2011;30(4):198–206.

Harnik, D, Kat, R, Margalit, O, Sotnikov, D, Traeger, A. (2013, February). To zip or not to zip: effective resource usage for real-time compression. In: Proceedings of the 11th USENIX conference on file and storage technologies. USENIX Association.

Hu, Y, Chen, HC, Lee, PP, Tang, Y. (2012, February). NCCloud: applying network coding for the storage repair in a cloud-of-clouds. In: Proceedings of the 10th USENIX conference on file and storage technologies. USENIX Association; pp. 21–21.

Ion, I, Sachdeva, N, Kumaraguru, P, Čapkun, S. (2011, July). Home is safer than the cloud!: privacy concerns for consumer cloud storage. In: Proceedings of the seventh symposium on usable privacy and security. ACM; p. 13.

Kaliski, B. (2000). RFC 2898: PKCS# 5: password-based cryptography specification version 2.0. IETF, September.

Kamp Poul-Henning, et al. LinkedIn password leak: salt their hide. ACM Queue 2012;10.6:20.

Kumar K, Lu YH. Cloud computing for mobile users: can offloading computation save energy? Computer 2010;43(4):51–6.

Mohamed N, Al-Jaroodi J, Eid A. A dual-direction technique for fast file downloads with dynamic load balancing in the cloud. J Netw Comput Appl 2013.

Mu, S, Chen, K, Gao, P, Ye, F, Wu, Y, Zheng, W. (2012, September). μLibCloud: providing high available and uniform accessing to multiple cloud storages. In: Grid computing (GRID), 2012 ACM/IEEE 13th international conference on IEEE; pp. 201–208..

Multi Cloud Storage Prototype, ⟨https://play.google.com/store/apps/details?id=com.tcboy.multi.cloud.storage.system&hl=en⟩.

Naldi, M, Mastroeni, L. (2013, April). Cloud storage pricing: a comparison of current practices. In: Proceedings of the 2013 international workshop on Hot topics in cloud services. ACM; pp. 27–34.

Pathak, A, Hu, YC, Zhang, M. (2012, April). Where is the energy spent inside my app? Fine grained energy accounting on smartphones with eprof. In: Proceedings of the seventh ACM European conference on computer systems. ACM; pp. 29–42.

Philopoulos, S, Maheswaran, M. (2001, August). Experimental study of parallel downloading schemes for internet mirror sites. In: Thirteenth IASTED international conference on parallel and distributed computing systems (PDCS'01); pp. 44–48.

Plank, JS, Simmerman, S, Schuman, CD. (2008). Jerasure: a library in C/C++ facilitating erasure coding for storage applications—Version 1.2. University of Tennessee, Tech. Rep. CS-08-627, 23.

Plank JS, Luo J, Schuman CD, Xu L, Wilcox-O'Hearn Z. A performance evaluation and examination of open-source erasure coding libraries for storage. In: FAST 2009;9:253–65.

Qian, F, Wang, Z, Gerber, A, Mao, Z, Sen, S, Spatscheck, O. (2011, June). Profiling resource usage for mobile applications: a cross-layer approach. In: Proceedings of the ninth international conference on mobile systems, applications, and services. ACM; pp. 321–334.

Rabin MO. Efficient dispersal of information for security, load balancing, and fault tolerance. J ACM (JACM) 1989;36(2):335–48.

Resch, JK, Plank, JS. (2011, February). AONT-RS: blending security and performance in dispersed storage systems. In: Proceedings of the ninth USENIX conference on file and storage technologies. USENIX Association; pp. 14–14.

Rodriguez P, Biersack EW. Dynamic parallel access to replicated content in the internet. IEEE/ACM Trans Networking (TON) 2002;10(4):455–65.

Seiger, R, Groß, S, Schill, A. (2011, September). SecCSIE: a secure cloud storage integrator for enterprises. In: Commerce and enterprise computing (CEC), 2011 IEEE 13th conference on IEEE; pp. 252–255.

Winzip (2012) Why don't some files compress very much? [online], available ⟨http://kb.winzip.com/kb/?View=entry&EntryID=104⟩ [accessed 9 December 2012].

Shamir A. How to share a secret. Commun ACM 1979;22(11):612–3.

Slamanig, D, Hanser, C. (2012, December). On cloud storage and the cloud of clouds approach. In: Internet technology and secured transactions, 2012 international conference for IEEE; pp. 649–655.

Spillner J, Müller J, Schill A. Creating optimal cloud storage systems. Future Gener Comput Syst 2013;29(4):1062–72.

Weatherspoon H, Kubiatowicz JD. Erasure coding vs. replication: a quantitative comparison. In peer-to-peer systems. Berlin Heidelberg: Springer; 328–37.