Fast and Secure Laptop Backups with Encrypted De-duplication

Paul Anderson
University of Edinburgh
dcspaul@ed.ac.uk

Le Zhang
University of Edinburgh
zhang.le@ed.ac.uk

Keywords: backup, de-duplication, encryption, cloud computing.

Abstract

Many people now store large quantities of personal and corporate data on laptops or home computers. These often have poor or intermittent connectivity, and are vulnerable to theft or hardware failure. Conventional backup solutions are not well suited to this environment, and backup regimes are frequently inadequate. This paper describes an algorithm which takes advantage of the data which is common between users to increase the speed of backups, and reduce the storage requirements. This algorithm supports client-end per-user encryption which is necessary for confidential personal data. It also supports a unique feature which allows immediate detection of common subtrees, avoiding the need to query the backup system for every file. We describe a prototype implementation of this algorithm for Apple OS X, and present an analysis of the potential effectiveness, using real data obtained from a set of typical users. Finally, we discuss the use of this prototype in conjunction with remote cloud storage, and present an analysis of the typical cost savings.

1 Introduction

Data backup has been an important issue ever since computers have been used to store valuable information. There has been a considerable amount of research on this topic, and a plethora of solutions are available which largely satisfy traditional requirements. However, new modes of working, such as the extensive use of personal laptops, present new challenges. Existing techniques do not meet these challenges well, and many individuals and organisations have partial, ad-hoc backup schemes which present real risks. For example:

- Backups are often made to a local disk and copies are not stored offsite.
- Backups are not encrypted and vulnerable to theft.
- Personal (rather than corporate) information is accidentally stored in plaintext on a corporate service where it can be read by other employees.
- Backups often just include "user files" in the assumption that "system files" can be easily recovered from elsewhere[1].
- The inconvenience of making backups leads to infrequent and irregular scheduling.

Even recent attempts to make backups largely transparent, such as Apple's Time Machine [10] suffer from the first two of the above problems, and may even lead users into a false sense of data security.

There has recently been a proliferation of "Cloud" backup solutions [3, 7, 12, 4, 1, 8, 6, 2]. In theory, these are capable of addressing some of the above problems. But, in practice, complete backups are unreasonably slow[2]:

> *"I have a home Internet backup service and about 1TB of data at home. It took me about three months to get all of the data copied off site via my cable connection, which was the bottleneck. If I had a crash before the off-site copy was created, I would have lost data"* [3]

And many organisations may prefer to hold copies of the backup data themselves.

[1] In practice, we found a small but significant number of unique files outside of the "user space" (figure 3) which means that this may not be such a reasonable assumption.

[2] Home broadband connections usually have upload speeds which are very significantly less than the download speed

[3] Henry Newman, October 9th 2009 - http://www.enterprisestorageforum.com/technology/features/article.php/3843151

1.1 De-duplication & Encryption

We observed that there is a good deal of sharing between the data on typical laptops (figure 3). For example, most (*but not all*) of the "system files" are likely to be shared with at least one other user. And it is common for users in the same environment to share copies of the same papers, or software packages, or even music files. Exploiting this duplication would clearly enable us to save space on the backup system. But equally importantly, it would significantly reduce the time required for backups in most cases – upgrading an operating system, or downloading a new music file should not require any additional backup time at all if someone else has already backed-up those same files.

There has been a lot of interest recently in *de-duplication* techniques, using *content-addressable storage* (CAS). This is designed to address exactly the above problem. However, most of these solutions are intended for use in a local filesystem [18, 9, 11] or SAN [20]. This has two major drawbacks: (i) clients must send the data to the remote filesystem before the duplication is detected – this forfeits the potential saving in network traffic and time. And (ii) any encryption occurs on the server, hence exposing sensitive information to the owner of the service – this is usually not appropriate for many of the files on a typical laptop which are essentially "personal", rather than "corporate"[4].

1.2 A Solution

This paper presents an algorithm and prototype software which overcome these two limitations. The algorithm allows data to be encrypted independently without invalidating the de-duplication. In addition, it is capable of identifying shared sub-trees of a directory hierarchy, so that a single access to the backup store can detect when an entire subtree is already present and need not be recopied. Clearly, this algorithm works for any type of system, but it is particularly appropriate for laptops where the connection speed and network availability are bigger issues than the processing time.

Initial versions of the prototype were intended to make direct use of cloud services such as Amazon S3 for remote storage. However, this has proven to be unworkable, and we discuss the reasons for this, presenting a practical extension to enable the use of such services.

Section 2 describes the algorithm. Section 3 describes the prototype implementation and gives some preliminary performance results. Section 4 presents the data collected from a typical user community to determine the practical extent and nature of shared files. This data is used to predict the performance in a typical environment, and to suggest further optimisations. Section 5 discusses the problems with direct backup to the cloud and presents a practical solution, including an analysis of the cost savings. Section 6 presents some conclusions.

[4]Of course, performing local encryption with personal keys would produce different cipher-text copies of the same file and invalidate any benefits of the de-duplication.

2 The Backup Algorithm

The backup algorithm builds on existing de-duplication and convergent encryption technology:

2.1 De-duplication

A *hashing function* (e.g. [14]) can be used to return a unique key for a block of data, based only on the contents of the data; if two people have the same data, the hashing function will return the same key[5]. If this key is used as the index for storing the data block, then any attempt to store multiple copies of the same block will be detected immediately. In some circumstances, it may be necessary store additional metadata, or a reference count to keep track of the multiple "owners", but it is not necessary to store multiple copies of the data itself.

2.2 Convergent Encryption

Encrypting data invalidates the de-duplication; two identical data blocks, encrypted with different keys, will yield different encrypted data blocks which can no longer be shared. A technique known as *convergent encryption* [16, 21, 19, 23] is designed to overcome this – the encryption key for the data block is derived from the contents of the data using a function which is similar to (but *independent of*) the hash function. Two identical data blocks will thus yield identical encrypted blocks which can be de-duplicated in the normal way. Of course each block now has a separate encryption key, and some mechanism is needed for each owner to record and retrieve the keys associated with "their" data blocks.

Typical implementations (such as [25]) involve complex schemes for storing and managing these keys as part of the block meta-data. This can be a reasonable approach when the de-duplication is part of a local filesystem. But there is considerable overhead in interrogating and maintaining this meta-data, which can be significant when the de-duplication and encryption is being performed remotely – and this is necessary in our case, to preserve the privacy of the data.

[5]Technically, it is possible for two different data blocks to return the same key. However, with a good hash function, the chances of this are sufficiently small to be insignificant - *"if you have something less than 95 EB of data, then your odds don't appear in 50 decimal places"*[5] - i.e. many orders of magnitude less than the chances of failure in any other part of the system.

2.3 The Algorithm

We have developed an algorithm which takes advantage of the hierarchical structure of the filesystem:

- Files are copied into the backup store as *file objects* using the convergent encryption process described above.
- Directories are stored as *directory objects* – these are simply files which contain the normal directory meta-data for the children, *and* the encryption/hash keys for each child.

To recover a complete directory hierarchy, we need only know the keys for the root node – locating this directory object and decrypting it yields the encryption and hash keys for all of the children and we can recursively recover the entire tree. This has some significant advantages:

- Each user only needs to record the keys for the root node. Typically, these would be stored independently on the backup system (one set for each stored hierarchy), and encrypted with the user's personal key.
- The hash value of a directory object acts as a unique identifier for the whole subtree; if the object representing a directory is present in the backup store, then we know that the entire subtree below it is also present. This means that we do not need to query the store for any of the descendants. It not uncommon to see fairly large shared subtrees[6], so this is a significant saving for remote access where the cost of queries is likely to be high. Section 4.3 presents some concrete experimental results.
- No querying or updating of additional metadata is required[7]. This means that updates to the backup store are atomic.

This algorithm does have some disadvantages. In particular, a change to any node implies a change to all of the ancestor nodes up to the root. It is extremely difficult to estimate the impact of this in a production environment, but preliminary testing seems to indicate that this is not a significant problem. There is also some disclosure of information; if a user has a copy of a file, it it possible to tell whether or not some other user also has a copy of the same file. This is an inevitable consequence of any system which supports storage sharing – if a user stores a file, and the size of the stored data does not increase, then there must have been a copy of this file already present.

[6]For example, between successive backups of the same system, or as the same application downloaded to different systems.

[7]If it is necessary to support deletion of shared blocks, then some kind of reference counting or garbage-collection mechanism is necessary, and this may be require additional metadata.

2.4 Implementation

An efficient implementation of this algorithm requires some care. The hash key for a directory object depends on its contents. This in turn depends on the keys for all of the children. Hence the computation of keys must proceed bottom-up. However, we want to prevent the backup from descending into any subtree whose root is already present. And this requires the backup itself to proceed top-down. For example:

```
BackupNode(N) {
    If N is a directory, then let O = DirectoryObjectFor(N)
    Otherwise, let O = contents of N
    Let H = Hash(O)
    if there is no item with index H in the backup store, then {
        Store O in the backup store with index H
        If N is a directory {
            For each entry E in the directory, BackupNode(E)
        }
    }
}

DirectoryObjectFor(D) {
    Create an empty directory object N
    For each entry E in the directory D {
        If E is a directory, then let O = DirectoryObjectFor(E)
        Otherwise, let O = contents of E
        Let H = Hash(O)
        Add the metadata for E, and the hash H to N
    }
    Return N
}
```

Of course, this is still a rather naive implementation – the hash for a particular object will be recomputed once for every directory in its path. This would be unacceptably expensive in practice and the hash function would probably be memoized. A production implementation presents many other opportunities for optimisation; caching of directory objects, parallelisation of compute-intensive tasks (encryption,hashing), and careful detection of files which have been (un-)modified since a previous run.

3 Prototype System

We developed a prototype backup system for Apple OS X as a proof of concept of the proposed algorithm[8]. The purpose is to be able to backup all files on a user's laptop to a central remote storage. The prototype was implemented as a set of command line utilities each performs a single task in the backup process such as scan-

[8]An earlier, simpler proof-of-concept was implemented under Windows as a student project [22].

ning file system changes, uploading files, restoring file system from backup etc. When wrapped in a GUI front-end, our application can be used as a drop-in replacement for the built-in Time Machine backup application [10].

3.1 Architecture

The underlying storage model we adopt is a write once, read many model. This model assumes that once a file is stored on the storage, it will never be deleted or rewritten. This is the common practice employed in enterprise environment where key business data such as electronic business records and financial data are required by law to be kept for at least five and seven years respectively [13]. This kind of storage model, commonly found in DVD-R or tape backup storage, is usually optimised for throughput rather than random access speed. The alternative storage model is write many, read many model that permits deleting older backups to reclaim some space. To achieve that some kind of reference counting mechanism is needed to safely delete un-referred files on the storage. To keep the prototype simple we opt to use the write once, read many storage model.

Our system is architected as a client/server application, where a backup client running on a user's laptop uploads encrypted data blocks to a central server which includes dedicated server side processing. This is in contrast to the thin-cloud approach where only minimal cloud-like storage interface is required on the server end as proposed in the Cumulus system [24]. Cumulus demonstrated that with careful client side data aggregation, backing up to the thin-cloud can be as efficient as integrated client/server approaches. This thin-cloud approach is appealing in a single user environment, however it raises three problems in a multi-user setting with data de-duplication technology:

1. The cloud storage model used by the thin-cloud backup solution does not have a straightforward way of sharing data between different user accounts without posing serious security threat.

2. There is no way to validate the content address of an uploaded object on the server. A malicious user can start an attack by uploading random data with a different content address to corrupt the whole backup.

3. The client side data aggregation used in the thin-cloud approach for performance reason will make data de-duplication very difficult, if not impossible.

In our prototype system we argue for a thin-server approach that addresses all these issues. The majority of computation will happen on the client side (hashing, encryption, data aggregation). A dedicated backup server will handle per-user security required in a multi-user environment. Instead of going for a full-blown client/server backup architecture with custom data transfer protocol, user authentication mechanism and hefty server end software we want to re-use the existing services provided by operating system itself as much as possible. To this end, we used standard services that come with many POSIX systems (Access control list (ACL) mechanism, user account management, Common Internet File System/Server Message Block (CIFS/SMB) server) and deployed a small server application written in Python on the server side to handle data validation. In addition, the whole server, once properly configured with ACL permissions, server application scripts, storage devices, can be packed as a virtual machine image and deployed into an enterprise's existing visualised storage architecture. This means the number of supported clients can be easily scaled to meet the backup demands.

Figure 1 depicts the architecture employed in our prototype. The system consists of several modules described below.

3.1.1 FSEvents Module

FSEvents was introduced as a system service in the OS X Leopard release (version 10.5). It logs file changes at the file system level and can report file system changes since a given time stamp in the past. This service is part of the foundational technologies used by Apple's built-in backup solution Time Machine. Our prototype system utilises the FSEvents service to get a list of changed files for incremental backup.

For efficiency reason, the event reported by the FSEvents API is at directory level only, i.e. it only tells which directory has changes in it but not exactly what was changed. To identify which files are changed we use a local meta database to maintain and compare current files' meta information with their historical values.

3.1.2 Local Meta DB

The local meta DB is used to implement incremental backup. The content of the DB includes: pathname, file size, last modification time, block-level checksums, and a flag indicates if the file has been backed up or not. For each backup session, the local meta DB will produce a list of files to backup which is sorted by directory-depth from bottom up. This is to ensure that a directory will be backed up only after all its children have been backed-up.

3.1.3 Backup Module

The backup module encapsulates the backup logic and interfaces with other modules. For each backup session, it first retrieves a list of files to backup from the local

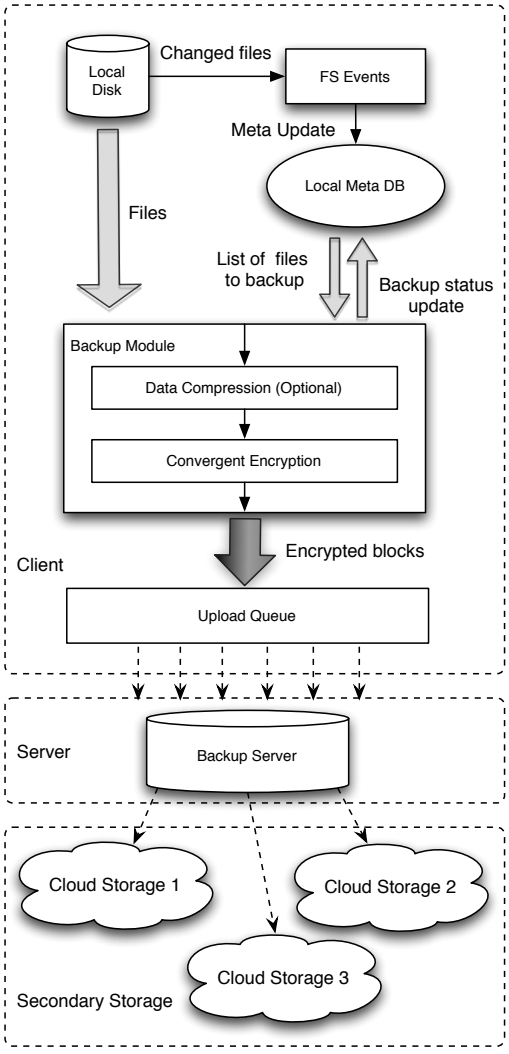

Figure 1: Architecture of the proposed backup system. Optionally data on the backup server can be replicated to multiple cloud storages in background.

meta DB, then it fetches those files from the local disk, calculates file hashs at block level, and encrypts each data block with an encryption key generated from the block's hash value. The final sequence of encrypted data blocks, together with their unique content addresses, are put in an upload queue to be dispatched to the remote storage. Optionally, data compression can be applied prior to the data encryption step to further reduce the size of the data to upload. We use a 256-bit cryptographically strong hash function (SHA2 [14]) for the content address and a 256-bit symmetric-key encryption (Salsa20 [17]) for data encryption.

3.1.4 Upload Queue

The system maintains an upload queue in memory for those data blocks to be uploaded to the backup server via CIFS protocol. First the content address of each data block is checked to see if the same data block is already on the server. If not the block will be scheduled to one of the uploading threads. A typical user machine contains many small files, which are less efficient to transfer over a standard network file system like CIFS compared to a custom upload protocol. We therefore perform data aggregation in the upload queue to pack small blocks into bigger packets of 2 MB before sending them over the network.

3.1.5 Server Application

A Python script on the server periodically checks any new files in a user's upload directory. ACL permission was set up so that the server application can read/write files in both public data pool and users' own upload directories. For each incoming packet, the server will disassemble the packet into original encrypted data blocks. If the block checksum matches its content address, the block will be moved to the public data pool. If a block has incorrect checksum due to network transmission error, it will be discarded and a new copy will be requested from the client.

Optionally, the server can choose to replicate its data to multiple public cloud storage vaults in the background. The use of cloud storage will be discussed in section 5.

4 Laptop Filesystem Statistics

The characteristics of the data, and the way in which it is typically organised and evolved, can have a significant effect on the performance of the algorithm and the implementation. We are aware of several relevant studies of filesystem statistics (e.g. [15]), however these have significant differences from our environment, such as the

operating system, or the type of user. We therefore decided to conduct our own small study of laptop users within our typical target environment. We are reasonably confident that this represents "typical usage".

A data collection utility was distributed to voluntary participants in the university to compute a 160-bit cryptographically strong hash for each file on their Mac laptop, along with other meta information such as file size, directory depth etc. Each file was scanned multiple times to get hash values of the following block sizes:128KB, 256KB, 512KB and 1024KB as well as single file hash value. Filenames were not collected to maintain privacy. We grouped the stats gathered from each machine into three categories: `USR`, `APP` and `SYS`. All data within a user's home folder is labelled as `USR`, data in `/Applications` is classified as `APP`, and all the rest are labelled as `SYS`. This is to help us identify where the data duplication occurs. We would expect a high degree of inter-machine data redundancy among Application and System files, but not so much between users' own data. In a real backup system, the amount of data transfer for subsequent incremental backups is typically much smaller than that of the initial uploads. To help estimate the storage request of incremental backup we collected the statistics twice over a two-month period.

4.1 Key Statistics

We gathered filesystem statistics from 24 Mac machines within the university, all of them are running either OS X 10.5 or 10.6. Although this is a small sample, we believe that this is a good representation of a typical target environment. Key statistics of the data are given in table 1. The histogram of file sizes is given in figure 2. The file size distribution follows normal distribution. A further breakdown reveals that the majority (up to 95%) of the files are relatively small (less than 100 KB).

The presence of huge number of small files will likely impose a speed issue when backing up due to I/O overhead and network latency. In addition, a cloud service provider is likely to charge for each upload/download operation. Therefore direct uploading to a cloud storage may not be an economically viable option, as will be seen in section 5.1.

4.2 Backup Simulation

We simulated a backup process by backing up one machine a time to the backup server. After each simulated backup, the projected and actual storage was recorded and the data duplication rate calculated. This was repeated until all machines were added to the backup storage, and this clearly demonstrates the increasing savings (in space per machine) as more machines partici-

Machines	24
Files	20,332,615
Directories	4,607,966
Entries (File + Dir)	24,940,581
File Sizes	
Median	2.4 K
Average	77.9 K
Maximum	32.2 GB
Total	1.94 TB
File Category	
`USR`	1.22 TB (62.94%)
`APP`	149.32 GB (7.68%)
`SYS`	570.8 GB (29.38%)
Harddisk Size	
Average HD Size	290 GB
Average Used HD	115 GB

Table 1: Filesystem statistics from Mac laptops.

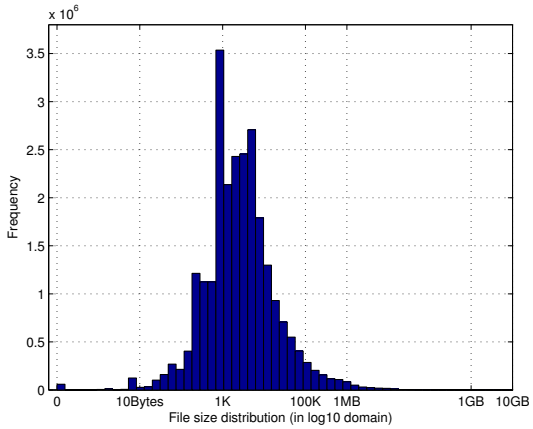

Figure 2: Histogram of file size distribution, the X-Axis is log_{10}(file size).

pate. Figure 3 shows the projected and actual storage for different numbers of machines. As expected, there is greater data sharing among System and Application files, but less so between users' own files.

4.3 Data Duplication

To measure data duplication (or redundancy) we define the data duplication rate as:

$$\frac{\text{Projected Storage Size-Actual Storage Size}}{\text{Projected Storage Size}} \times 100\%$$

where the projected storage size is the sum of all data to backup, actual storage size is the actual amount of storage required due to data de-duplication. For instance, if we need to store a total amount of 100 GB data from two machines A and B, and only 70 GB is needed for actual storage, then the data duplication rate for A and B is 30%.

Backup Block Size

File backup can be performed at whole file level or sub-file level where a file is split into a series of fixed-size blocks. There are two main benefits of performing backup at sub-file block level. First is the increased level of data de-duplication as a result of inter-file block sharing, which will use less storage space. Second, it is more efficient to backup certain type of files where only a small portion of the file content is constantly changing. For instance, database files and virtual machine disk images are usually a few GB in size and are known to subject to frequent updating. Modifications to those files are usually made in-place at various small portions of the file for performance reason. Block-level backup enables us to only backup those changed blocks. Finally, there is also a practical benefit: it is more reliable to upload a small block of data over remote network than a big file. Even the transfer fails due to network glitch, only the last block needs to be resent.

Despite the listed advantages, backup at block-level will incur some overhead that can be significant. Depending on the size of the block, the number of total stored objects could be much higher than whole file backup, which would be an concern when backing up to a cloud storage where the network I/O requests are charged (see section 5.1). Also extra storage is needed to record the block relationship which could offset some of the benefit of data de-duplication.

We tested different block sizes as well as whole file, single block backup in the simulation experiment. The result is plotted in figure 4.

As expected, all the sub-file blocks achieve a higher data duplication rate than single block, shown in figure 4a. The bigger the block size is, the lower data duplication rate due to less inter-file block sharing. For block size of 128KB, we the data duplication rate of 32.08%, which is 9.5% higher than the 29.31% of single block. This transfers into less storage used for all sub-file blocks (Figure 4b). However, the increased number of block objects could be quite substantial: for block size of 128KB, the total number of objects is 38m (million), 64.4% more than that of single block objects which is 24.94m (Figure 4c).

Overall	USR	SYS	APP
29.31%	8.91%	63.34%	63.06%

Table 2: Overall data duplication rate by category.

Directory Tree Duplication

As mentioned in section 2, the directory meta data is stored in directory objects. The size of all directory objects is 16.24 GB. With data de-duplication, the actual required storage is 6.4 GB, or 0.47% of total used storage. The collected stats also reveals that, among all 4,607,966 directory objects, only 1,052,338 unique ones are stored on the server. This suggests that up to 77% directory objects are shared. This strongly supports the value of sub-tree de-duplication.

Finally, we report overall data duplication rate in table 2.

Changes Over Time

Once an initial, full backup of a user machine is made, subsequent backups can be much faster as only changed data needs to be uploaded. To get an estimate of the file change rate, we collected and analysed the data for a second time towards the end of the two-month pilot experiment. We are mainly interested in the sizes of newly added files and changed files that will be included in the incremental backup. Files deleted since last scan were not included. The average daily and monthly per-machine data change rates calculated using a block size of 128KB are presented in table 3. In addition, we observed that the estimated monthly per-machine data change rate would raise from 17.17 GB to 20.61 GB if the backup is performed at file level. This confirms our earlier assumption that backing up at sub-file level can be more efficient in dealing with partial changes in large files. In our scenario it would reduce the amount of data to backup by 3.44 GB for each machine on a monthly basis.

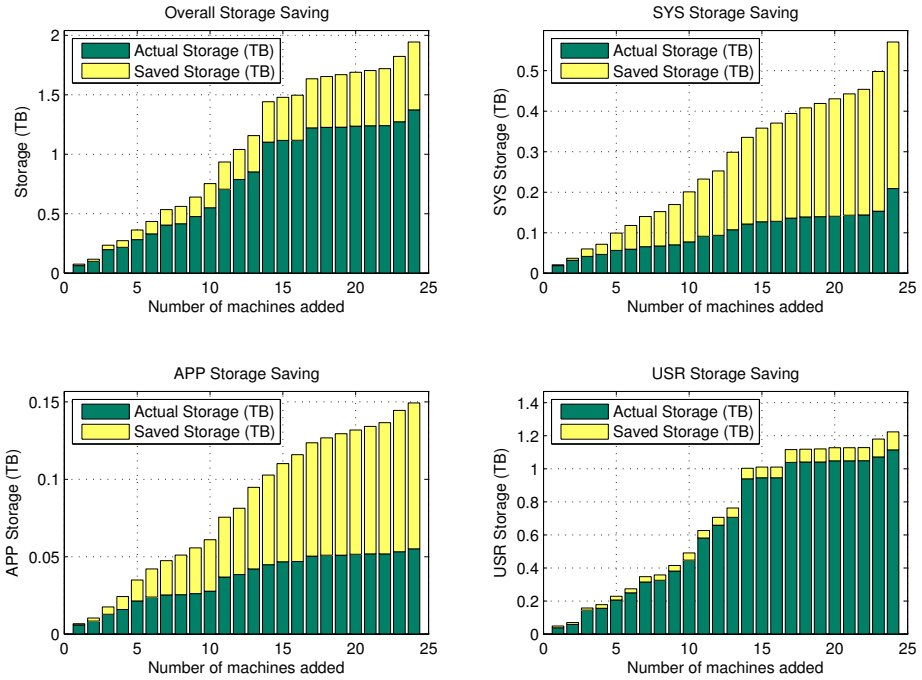

Figure 3: Projected and actual storage by number of machines added during simulated backup.

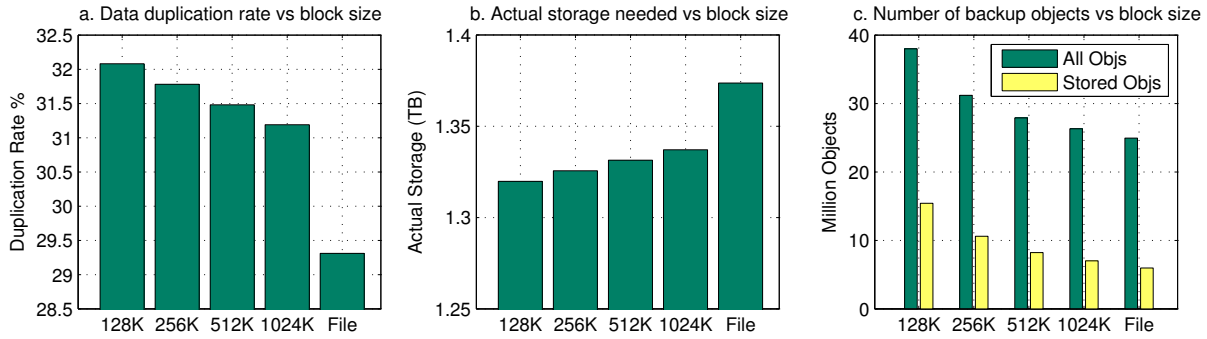

Figure 4: Data duplication rate and actual storage under different block sizes (File means the whole file is treated as a single block).

	Overall	USR	SYS	APP
Daily	0.57 GB	0.31 GB	0.10 GB	0.03 GB
Monthly	17.17 GB	9.20 GB	2.90 GB	0.86 GB

Table 3: Estimated daily and monthly (30 days) per-machine data change rates by category.

5 Using Cloud Storage

Backing up to cloud-based storage becomes increasing popular in recent years. The main benefits of using a cloud storage are lower server maintenance cost, cheaper long term operational cost, and sometimes enhanced data safety via a vendor's own geographically diverse data replication. However, there are still some obstacles to integrating cloud storage into a full-system backup solution:

1. Network bandwidth can be a bottle-neck: uploading data directly to a cloud storage can be very slow while requiring a reliable network connection.

2. The cloud interface has yet be standardised with each vendor offering its own security and data transfer models. In addition, many organisations prefer to have a backup policy that can utilise multiple cloud services to avoid vendor lock-in.

3. A cloud service provider is likely to impose a charge on individual data upload/download operations, which means backing up directly to a cloud can be very costly.

Despite these disadvantages, in the next section we will show that the proposed de-duplicated backup algorithm can effectively reduce the cost of backing up to a cloud storage by a large margin. Furthermore, in our backup architecture it is possible to adopt cloud storage as the secondary storage of the backup server (Figure 1), thereby largely ameliorating the above issues. In particular, the benefits of employing a cloud-based secondary storage are:

1. Backing up to local backup server can still be very fast with all the security features enabled, while the data replication to cloud storage can be performed in the background.

2. New cloud services can be added easily on the backup server to provide enhanced data safety and to reduce the risk of vendor lock-in.

3. Upload cost to cloud storage can be reduced via data aggregation techniques such as employed in [24].

5.1 Cost Saving: Data De-Duplication

In this section we measure the estimated cost of backup to a cloud storage in terms of the bill charged by a typical storage provider: Amazon S3[9]. Its data transfer model is based on standard HTTP protocol requests like `GET` and `PUT`.

For de-duplication backup, the client first needs to check if an object exists on the cloud via an `HEAD` operation. If not, the object is then uploaded via a `PUT` operation. Fortunately, if a file already exists on the server due to data de-duplication, we do not need to use the more expensive `PUT` operation, and no data will be uploaded.

Using the Amazon S3 price model[10] we plot the estimated cost of backing up 1.94 TB data from 24 machines to S3 in terms of US dollars in figure 5. The data de-duplication technology, coupled with data aggregation (see next section), is able to achieve 60% cost reduction for the initial data upload.

5.2 Cost Saving: Data Aggregation

Backing up directly to Amazon S3 can be very slow and costly due to large number of I/O operations and the way Amazon S3 charges for the uploads. Client side data aggregation packs small files into bigger packets before sending them to the cloud. We packed individual files into packets with a size of 2 MB each and compared the estimated cost against that of direct upload (see figure 6). The result shows that an overall of 25% of cost saving can be achieved via data aggregation alone. Moreover, even when cost is not an issue, as is the case when backing up to a departmental server, data aggregation can still speed up the process significantly. In our experiment we observed that the underlying file system (CIFS in this case) did not handle large amount of small I/O requests efficiently between the client and the server, even on a high-speed network. This resulted in a much slower backup speed. Uploading aggregated data to the server and unpacking the data there overcomes this problem.

5.3 Case Study: Six-Month Bill Using S3

Using the gathered information so far, we are able to estimate the cost of backing up all the machines we have to Amazon S3 over a six-month period. The initial upload of 1.94 TB data (25m objects) will cost $434 without data de-duplication, together with storage cost ($291) that is $725. With data de-duplication, only a total of

[9] `https://s3.amazonaws.com/`

[10] As of writing this paper, data upload to Amazon S3 is charged at $0.1 per GB. Data storage rate is $0.15 per GB for the first 50 TB/Month of storage. Operating cost is $0.01 per 1,000 requests for PUT operation and $0.01 per 10,000 requests for HEAD operation, respectively. All prices quoted are from US Standard tier.

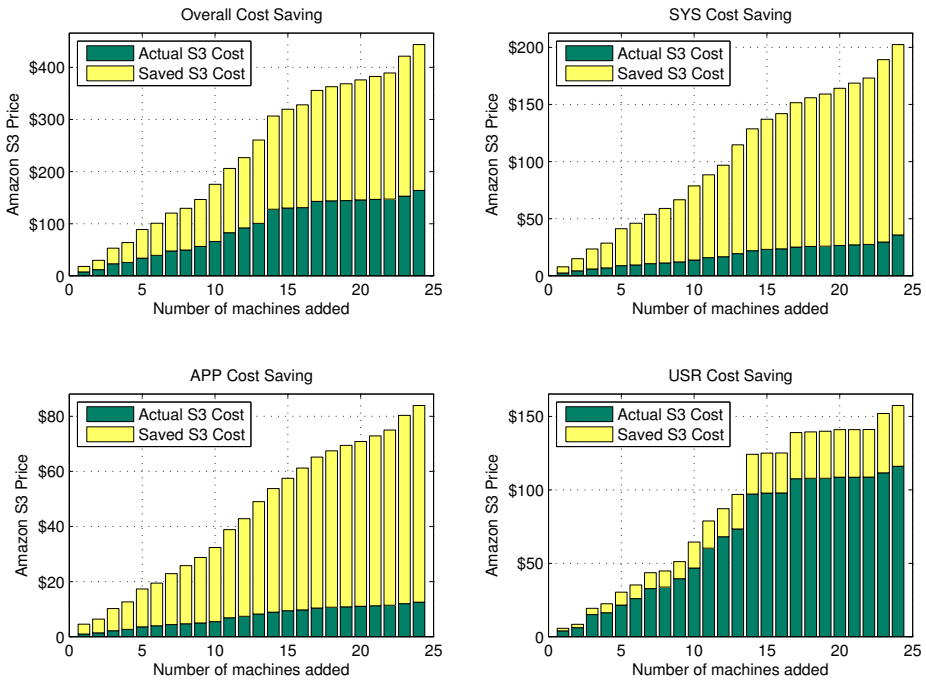

Figure 5: Estimated cost when backing up 24 machines to Amazon S3 (total cost for the initial backup of all machines)

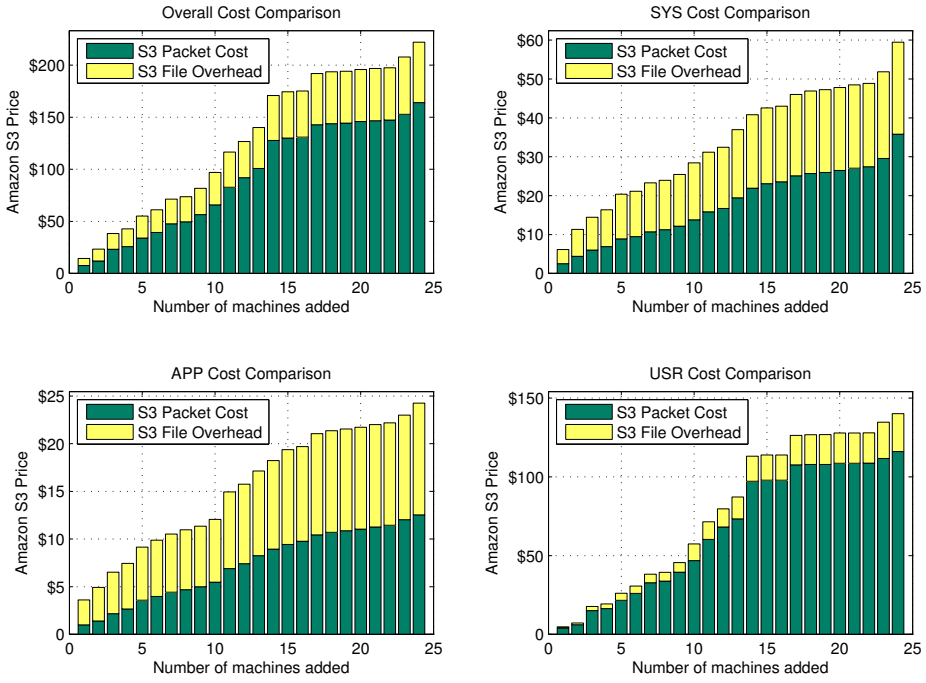

Figure 6: Estimated cost comparison of client side data aggregation (packet) and normal upload (file).

Months	Conventional Storage	Conventional Cost	De-Duplicated Storage	De-Duplicated Cost
1	1.94 TB	$725	1.37 TB	$370
2	2.35 TB	$445	1.66 TB	$284
3	2.76 TB	$507	1.95 TB	$328
4	3.18 TB	$569	2.25 TB	$372
5	3.59 TB	$630	2.54 TB	$415
6	4.00 TB	$692	2.83 TB	$459
Total		$3,568		$2,228

Table 4: Estimated monthly and accumulated bills for backing up to Amazon S3 over a six-month period without and with the De-Duplicated backup. The monthly data change rate is estimated to be 0.49 TB for 24 machines using figures in table 3 with an estimated duplication rate of 29.31%. The monthly cost is the sum of S3 storage bill (currently $150 per TB) plus the estimated data upload cost from the pilot experiment.

$370 ($164 for upload and $206 for storage) is needed, saving $355 or 49.0% of the initial upload cost. Moreover, as only 1.37 TB of data needs to be uploaded, it is estimated to save 29.4% uploading time. The saving can be even greater over time as less cloud storage is used and billed. The accumulated costs of using Amazon S3 over the course of six months are $3,568 using conventional backup method, and $2,228 using de-deuplicated backup. Monthly estimated bills are presented in table 4.

6 Conclusions

We have shown that a typical community of laptop users share a considerable amount of data in common. This provides the potential to significantly decrease backup times, and storage requirements. However, we have shown that manual selection of the relevant data - for example, backing up only home directories - is a poor strategy; this fails to backup some important files, at the same time as unnecessarily duplicating other files.

We have presented a prototype backup program which achieves an optimal degree of sharing at the same time as maintaining confidentiality. This exploits a novel algorithm to reduce the number of files which need to be scanned and hence decreases backup times.

We have shown that typical cloud interfaces, such as Amazon S3 are not well suited to this type of application, due to the time and cost of typical transfers, and the lack of multi-user authentication to shared data. We have described a implementation using a local server which can avoid these problems by caching and pre-processing data before transmitting to the cloud. This is shown to achieve significant cost savings.

7 Acknowledgements

This project has been funded by the Edinburgh University *Initiating Knowledge Transfer Fund* (iKTF)[11] and the IDEA Lab [12]. Thanks to Toby Blake from the School of Informatics for support with the infrastructure, and to all those who contributed data.

[11] http://www.eri.ed.ac.uk/commercial/developmentfunding/ikta.htm

[12] http://www.idea.ed.ac.uk/IDEA/Welcome.html

References

[1] Backblaze: online backup [cited 2nd April 2010]. http://www.backblaze.com/.

[2] Backjack: online backup [cited 2nd April 2010]. http://www.backjack.com.

[3] Carbonite: online backup [cited 2nd April 2010]. http://www.carbonite.com/.

[4] Crashplan: online backup [cited 2nd April 2010]. http://www.crashplan.com.

[5] Hash collisions: The real odds [cited 31st March 2010]. http://www.backupcentral.com/content/view/145/47/.

[6] Humyo: online backup [cited 2nd April 2010]. http://humyo.com/.

[7] IDrive: remote data backup [cited 2nd April 2010]. http://www.idrive.com.

[8] JungleDisk: online backup [cited 2nd April 2010]. http://www.jungledisk.com/.

[9] Lessfs – a high performance inline data deduplicating filesystem for Linux [cited 2nd April 2010]. http://www.lessfs.com.

[10] Mac OS X Time Machine [cited 2nd April 2010]. http://www.apple.com/macosx/what-is-macosx/time-machine.html.

[11] SDFS: A deduplication file-system for Linux [cited 2nd April 2010]. http://www.opendedup.org/.

[12] Soonr: active backup [cited 2nd April 2010]. http://www.soonr.com/.

[13] Sarbanes-Oxley act of 2002, February 2002. http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/content-detail.html.

[14] Federal information processing standards publications 180-2: Secure hash standard (SHS). Technical report, National Institute of Standards and Technology Gaithersburg, MD 20899-8900, October 2008. http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf.

[15] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, pages 31–45, 2007. http://www.usenix.org/events/fast07/tech/agrawal.html.

[16] K. Bennett, C. Grothoff, T. Horozov, and I. Patrascu. Efficient sharing of encrypted data. In *In Proceedings of ASCIP 2002*, pages 107–120. Springer-Verlag, 2002. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.9837.

[17] D. Bernstein. The Salsa20 family of stream ciphers. *New Stream Cipher Designs*, pages 84–97, 2008. http://dx.doi.org/10.1007/978-3-540-68351-3_8.

[18] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in Windows 2000. In *Proceedings of 4th USENIX Windows Systems Symposium*. Usenix, 2000. http://research.microsoft.com/apps/pubs/default.aspx?id=74261.

[19] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of the international conference on measurement and modeling of computer systems (SIGMETRICS)*, 2007. http://research.microsoft.com/apps/pubs/default.aspx?id=74262.

[20] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in SAN cluster file systems. In *Proceedings of the 2009 Usenix Annual Technical Conference*. Usenix, June 2009. http://www.usenix.org/events/usenix09/tech/full_papers/clements/clements.pdf.

[21] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *In Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS*, 2002. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=75E78117EB6C02C4493CA49F28775D65?doi=10.1.1.8.7586&rep=rep1&type=pdf.

[22] G. Gonsalves. Content addressable storage for encrypted shared backup. Master's thesis, School of Informatics, University of Edinburgh, 2009. http://homepages.inf.ed.ac.uk/dcspaul/publications/CASFESB.pdf.

[23] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller. Secure data deduplication. In *StorageSS '08: Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 1–10, New York, NY, USA, 2008. ACM. http://portal.acm.org/citation.cfm?id=1456469.1456471#.

[24] M. Vrable, S. Savage, and G. M. Voelker. Cumulus: Filesystem backup to the cloud. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, February 2009. http://cseweb.ucsd.edu/~voelker/pubs/cumulus-fast09.pdf.

[25] D. Wang, L. Wang, and J. Song. SEDBRS: A secure and efficient desktop backup and recovery system. *Data, Privacy, and E-Commerce, International Symposium on*, 0:304–309, 2007. http://www.computer.org/portal/web/csdl/doi/10.1109/ISDPE.2007.27.