CDStore: Toward Reliable, Secure, and Cost-Efficient Cloud Storage via Convergent Dispersal

Mingqiang Li, Chuan Qin, Jingwei Li, and Patrick P. C. Lee

***Abstract*—We present CDStore, a unified multi-cloud storage solution for users to outsource backup data with reliability, security, and cost efficiency guarantees. CDStore builds on an augmented secret sharing scheme called *convergent dispersal*, which supports deduplication by using deterministic content-derived hashes as inputs to secret sharing. We present the design of CDStore, and in particular, describe how it combines convergent dispersal with two-stage deduplication to achieve both bandwidth and storage savings and also be robust against side-channel attacks that can be launched by a malicious user on the client side. We demonstrate via cost analysis that CDStore achieves significant monetary cost savings over baseline cloud storage solutions.**

***Index Terms*—Secret sharing, deduplication, cloud storage.**

I. Introduction

Multi-cloud storage coalesces multiple public cloud storage services, operated by independent vendors, into a single storage pool. By dispersing data with some form of redundancy across multiple clouds, multi-cloud storage provides a plausible way to realize *reliable* and *secure* outsourced storage. By reliable, we mean that the stored data remains available even in the presence of cloud failures; by secure, we mean that the stored data is protected with the guarantees of confidentiality (i.e., the data is kept secret from unauthorized parties) and integrity (i.e., the data remains uncorrupted).

Secret sharing is one form of redundancy that provides both reliability and security guarantees, and it has been realized in multi-cloud storage (e.g., [2], [3], [5], [14]). Given the configuration parameters r, k, and n (where $r < k < n$), it transforms a data input (called *secret*) into n coded outputs (called *shares*), such that the secret can be recovered with any k out of n shares and the secret cannot be inferred if only r shares are available. Secret sharing often comes with high redundancy. For example, Shamir's scheme [16] has the same storage overhead as replication. However, it is plausible to reduce the redundancy of secret sharing to be slightly higher than that of erasure coding, while preserving security in the computational sense [14].

However, existing secret sharing algorithms prohibit storage savings achieved by *deduplication*, which works by keeping only one physical data copy and having it shared by other copies with identical content. Field measurements show that deduplication is especially effective for some workloads with high content similarity, such as backups [17]. On the other hand, secret sharing uses random input seeds to generate shares. If users embed different random input seeds, their shares will differ and cannot be deduplicated, even though their original data is identical.

Motivated from users' perspectives, we present *CDStore*, which provides a unified multi-cloud storage solution with reliability, security, and cost efficiency guarantees. CDStore builds on an enhanced secret sharing scheme called *convergent dispersal*, whose core idea is to replace the random input seeds of traditional secret sharing with deterministic cryptographic hashes derived from the original data, while the hashes cannot be inferred by attackers without knowing the whole original data. Thus, identical secrets are always transformed into identical shares, which can be deduplicated. We believe that unifying secret sharing and deduplication is beneficial for cloud storage: secret sharing provides reliability and security guarantees, while deduplication provides cost-efficiency guarantees by offsetting the dispersal-level redundancy of secret sharing with the removal of content-level redundancy.

This article is an extension to our conference paper [10], in which we have discussed the design and implementation details of CDStore and presented its performance results based on testbed experiments. Here, we provide detailed cost breakdown analysis, and show how CDStore significantly saves monetary costs over baseline cloud storage solutions (e.g., by more than 70% in some cases). The source code of our CDStore prototype is available online[1].

II. CDStore Design

CDStore is designed for an organization to outsource the storage of data of a large group of users to multiple clouds. It builds on the client-server architecture shown in Figure 1. Each user of the same organization runs a *CDStore client* to store and access his data in multiple clouds over the Internet. In each cloud, a co-located virtual machine (VM) instance owned by the organization runs a *CDStore server* between multiple CDStore clients and the storage backend.

CDStore targets backup workloads, which are known to have high content similarity [17]. We consider a type of backups obtained by snapshotting applications, file systems, or virtual disk images. In CDStore deployment, each user machine submits a series of backup files (e.g., in UNIX `tar` format) to the co-located CDStore client, which processes the backups and uploads them to the CDStore servers in all clouds.

A. Goals and Assumptions

We state the design goals and assumptions of CDStore in three aspects: reliability, security, and cost efficiency.

Mingqiang Li is now with Hong Kong Advanced Technology Center, Ecosystem & Cloud Service Group, Lenovo Group Ltd. This work was done when he was with the Chinese University of Hong Kong.

Chuan Qin, Jingwei Li, and Patrick P. C. Lee are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong.

[1]http://ansrlab.cse.cuhk.edu.hk/software/cdstore

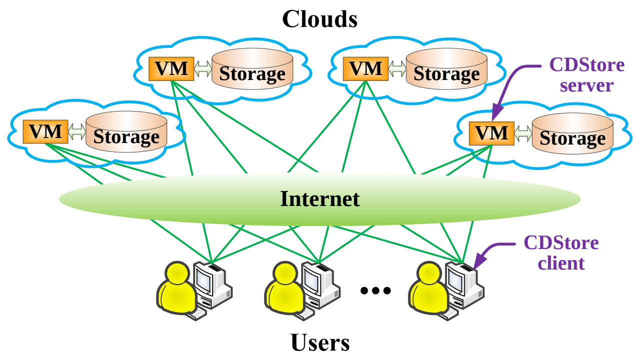


Fig. 1. CDStore architecture.

Reliability: CDStore tolerates failures of cloud storage providers and even CDStore servers. Outsourced data is accessible if a tolerable number of clouds (and their co-located CDStore servers) are operational. CDStore also tolerates client-side failures by offloading metadata management to the server side (see Section II-D). In the presence of cloud failures, CDStore decodes original secrets and then rebuilds any lost share as in Reed-Solomon codes [13], and we do not consider cost-efficient recovery here [9].

Security: CDStore ensures *confidentiality* and *integrity* of outsourced data, as long as a tolerable number of clouds are uncompromised. Note that the confidentiality guarantee requires that the secrets be drawn from a very large message space, so that brute-force attacks are infeasible [1]. CDStore also avoids side-channel attacks [7], [8] via two-stage deduplication (see Section II-C). Here, we do not consider strong attack models, such as Byzantine faults in cloud services [2]. We also assume that the client-server communication over the network is protected, so that an attacker cannot infer any secret by eavesdropping the transmitted shares.

Cost efficiency: CDStore uses deduplication to reduce both bandwidth and storage costs. It also incurs limited overhead in computation (e.g., VM usage) and storage (e.g., metadata).

B. Convergent Dispersal

Convergent dispersal enables secret sharing with deduplication by replacing the embedded random input seed with a deterministic cryptographic hash derived from the secret. Thus, two secrets with identical content must generate identical shares, making deduplication possible. Also, it is computationally infeasible to infer the hash without knowing the whole secret. Our idea is inspired by convergent encryption [6] used in traditional key-based encryption, in which the random key is replaced by the cryptographic hash of the data to be encrypted. Figure 2 shows the main idea of how we augment a secret sharing algorithm with convergent dispersal.

Here, we present a convergent dispersal instantiation based on an existing secret sharing scheme called *AONT-RS* [14], which combines Rivest's all-or-nothing transform (AONT) [15] for confidentiality and Reed-Solomon coding [13] for fault tolerance. We call our convergent dispersal instantiation *CAONT-RS*, which aims to inherit the reliability and security properties of the original AONT-RS, and makes two key modifications. First, to improve performance, CAONT-RS replaces Rivest's AONT [15] with another AONT based on *optimal asymmetric encryption padding (OAEP)* [4]. The rationale is that Rivest's AONT performs multiple encryptions on small-size words, while OAEP-based AONT performs a single encryption on a large-size, constant-value block. Also, OAEP-based AONT provably provides no worse security than any AONT scheme [4]. Second, CAONT-RS replaces the random input seed in AONT with a deterministic cryptographic hash derived from the secret. Thus, it preserves content similarity in dispersed shares and allows deduplication.

We elaborate the encoding and decoding of CAONT-RS, both of which are performed by a CDStore client. Figure 3 shows an example of CAONT-RS with $n = 4$, $k = 3$. Also, CAONT-RS has $r = k - 1 = 2$, as in AONT-RS [14].

Encoding: We first transform a given secret X into a CAONT package. Specifically, we first generate a hash key h, instead of a random key, derived from X using a (optionally salted) hash function $\mathbf{H}$ (e.g., SHA-256):

$$h = \mathbf{H}(X). \tag{1}$$

To achieve confidentiality, we transform (X, h) into a CAONT package (Y, t) using OAEP-based AONT, where Y and t are the head and tail parts of the CAONT package and have the same size as X and h, respectively. To elaborate, Y is generated by:

$$Y = X \oplus \mathbf{G}(h), \tag{2}$$

where '$\oplus$' is the XOR operator and $\mathbf{G}$ is a generator function that takes h as input and constructs a mask block with the same size as X. Here, we implement the generator $\mathbf{G}$ as:

$$\mathbf{G}(h) = \mathbf{E}(h, C), \tag{3}$$

where C is a constant-value block with the same size as X, and $\mathbf{E}$ is an encryption function (e.g., AES-256) that encrypts C using h as the encryption key.

The tail part t is generated by:

$$t = h \oplus \mathbf{H}(Y). \tag{4}$$

Finally, we divide the CAONT package into k equal-size shares (we pad zeroes to the secret if necessary to ensure that the CAONT package can be evenly divided). We encode them into n shares using the systematic Reed-Solomon codes [13].

To enable deduplication, we ensure that the same share is located in the same cloud. Since the number of clouds for multi-cloud storage is usually small, we simply disperse shares to all clouds. Suppose that CDStore spans n clouds, which we label $0, 1, \cdots, n-1$. After encoding each secret using convergent dispersal, we label the n generated shares $0, 1, \cdots, n-1$ in the order of their positions in the Reed-Solomon encoding result, such that share i is to be stored on cloud i, where $0 \le i \le n-1$. This ensures that the same cloud always receives the same share from the secrets with identical content, either generated by the same user or different users. This also enables us to easily locate the shares during restore.

Decoding: To recover the secret, we retrieve any k out of n shares and use them to reconstruct the original CAONT package (Y, t) (note that the retrieval size is slightly more than

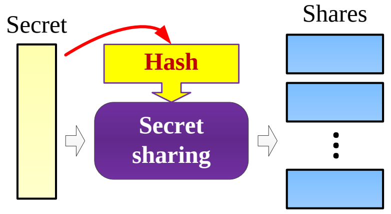


Fig. 2. Idea of convergent dispersal.

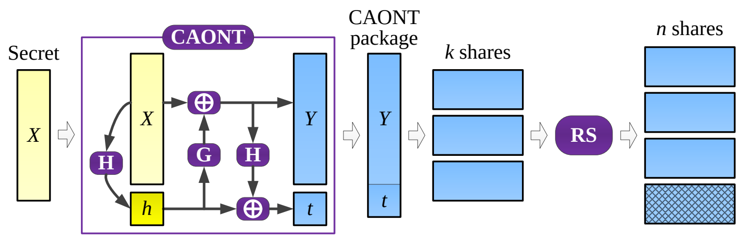


Fig. 3. Example of CAONT-RS with $n = 4$ and $k = 3$. Note that $r = k - 1 = 2$.

the original secret size by the size of h). Then we deduce hash h by XOR'ing t with $\mathbf{H}(Y)$ (see Equation (4)). Finally, we deduce secret X by XOR'ing Y with $\mathbf{G}(h)$ (see Equation (2)), and remove any padded zeroes introduced in encoding.

We can also verify the integrity of the deduced secret X. We simply generate a hash value from the deduced X as in Equation (1) and compare if it matches h. If the match fails, then the decoded secret is considered to be corrupted.

Remarks: We briefly discuss the security properties of CAONT-RS. CAONT-RS ensures confidentiality, provided that an attacker cannot gain unauthorized accesses to k out of n clouds, and ensures integrity through the embedded hash in each secret. It leverages AONT to ensure that no information of the original secret can be inferred from fewer than k shares. We note that an attacker can identify the shares of different users and perform brute-force dictionary attacks [1] inside the clouds, and thus we require that the secrets be drawn from a large message space (see Section II-A). To mitigate brute-force attacks, we may replace the hash key in CAONT-RS with a more sophisticated key generated by a key server [1], with the trade-off of introducing the key management overhead.

C. Two-Stage Deduplication

We first overview how deduplication works. Deduplication divides data into fixed-size or variable-size *chunks*. Each chunk is uniquely identified by a *fingerprint* computed by a cryptographic hash of the chunk content. Two chunks are said to be identical if their fingerprints are the same, and fingerprint collisions of two different chunks are assumed to be unlikely. Deduplication stores only one copy of a chunk, and refers any duplicate chunks to the copy via small-size references.

To realize deduplication in cloud storage, a naïve approach is to perform global deduplication on the client side. Specifically, before a user uploads data to a cloud, it first generates fingerprints of the data. It then checks with the cloud by fingerprint for the existence of any duplicate data that has been uploaded by any user. Finally, it uploads only the unique data to the cloud. Although client-side global deduplication saves upload bandwidth and storage overhead, it is susceptible to *side-channel attacks* that can be launched by a malicious user on the client side [7], [8]. One attack is to infer the existence of data of other users [8]. Specifically, an attacker generates the fingerprints of some possible data of other users and queries the cloud by fingerprint if such data is unique and needs to be uploaded. If no upload is needed, then the attacker infers that other users own the data. Another attack is to gain unauthorized access to data of other users [7]. Specifically, an attacker uses the fingerprints of some sensitive data of other users to convince the cloud of the data ownership.

CDStore prevents side-channel attacks through *two-stage deduplication*, which eliminates duplicates first on the client side and then on the server side. Suppose that each CDStore server maintains a deduplication index that keeps track of which shares have been stored by each user and how shares are deduplicated. Then we implement the two deduplication stages as follows.

Intra-user deduplication: A CDStore client first runs deduplication only on the data owned by the same user, and uploads the unique data of the user to the cloud. Before uploading shares to a cloud, the CDStore client first checks with the CDStore server by fingerprint if it has already uploaded the same shares. Specifically, the CDStore client first sends the fingerprints generated from the shares to the CDStore server. The CDStore server then looks up its deduplication index, and replies to the CDStore client a list of share identifiers that indicate which shares have been uploaded by the CDStore client. Finally, the CDStore client uploads only unique shares to the cloud based on the list.

Inter-user deduplication: A CDStore server runs deduplication on the data of all users and stores the globally unique data in the storage backend. After the CDStore server receives shares from the CDStore client, it generates a fingerprint from each share (instead of using the one generated by the CDStore client for intra-user deduplication), and checks if the share has already been stored by other users by looking up the deduplication index. It stores only the unique shares that are not yet stored at the storage backend. It also updates the deduplication index to keep track of which user owns the shares. Here, we cannot directly use the fingerprint generated by the CDStore client for intra-user deduplication. Otherwise, an attacker can launch a side-channel attack, by using the fingerprint of a share of other users to gain unauthorized access to the share [7].

Remarks: Two-stage deduplication forces a user to upload shares that may have been uploaded by another user and hence avoiding client-side inter-user deduplication [8]. It makes deduplication patterns independent across users' uploads, so an attacker cannot infer the data content of other users through deduplication occurrences.

Both intra-user and inter-user deduplications effectively remove duplicates. Intra-user deduplication eliminates duplicates of the same user's data, for example, when a user makes repeated backups of the same application or file system. Inter-user deduplication further removes duplicates of multiple users, for example, when the machines of multiple users of the same organization share the same operating system configurations [11]. The removal of duplicates translates to cost savings (see Section III).

D. Implementation Details

We have implemented a CDStore prototype based on the client-server model in Figure 1. Here, we highlight the important implementation details of CDStore, while the full details are in our conference paper [10].

Chunking: CDStore can perform deduplication on fixed-size or variable-size chunks. We implement variable-size chunking based on Rabin fingerprinting [12]. Here, we set the size of each secret (chunk) on the order of kilobytes to effectively remove duplicates. For example, our default average chunk size is 8KB.

Server-side metadata management: We make CDStore servers keep and manage all metadata on behalf of CDStore clients, which are generally less reliable. There are two types of metadata: (i) *file metadata*, which describe file information, and (ii) *share metadata*, which describe each unique share being stored. We distribute the metadata across all CDStore servers for reliability.

In particular, we encode and disperse file pathnames, which are considered to be sensitive metadata, via CAONT-RS. Each CDStore server stores both file and share metadata at the storage backend, and keeps the respective file and share indices in local index structures to reference metadata.

Container management: CDStore mitigates I/O overheads by arranging storage in units of *containers*. We have two types of containers: *share containers*, which hold unique shares, and *recipe containers*, which hold file recipes (i.e., the complete file descriptions). All shares and file recipes are packed into respective containers, with a default size 4MB each.

Multi-threading: We exploit multi-threading to parallelize intensive operations. For example, we parallelize the encoding/decoding operations of CAONT-RS: in file uploads, we pass each secret output to one of the threads for encoding; in file downloads, we pass the received shares of a secret to a thread for decoding. Also, we use multiple threads for communications to fully utilize the network transfer bandwidth.

III. Cost Analysis

We present detailed cost breakdown analysis on how CDStore saves monetary costs over baseline cloud storage solutions. Our conference paper [10] also presents performance results of CDStore based on testbed experiments.

A. Cost Components

We first characterize the cost components of CDStore.

Our analysis is based on the tiered price plans of four cloud providers in November 2015, including Amazon Singapore[2], Google Asia[3], Azure Southeast Asia[4], and Rackspace Hong Kong[5]. Free charges apply to inbound transfers to VM instances as well as data transfers between co-located VM instances and cloud storage. We now break down the total backup cost incurred by CDStore in two aspects.

- *VM cost:* The VM cost is the fee charged for running VM instances for hosting CDStore servers. The VM cost has two parts: (i) *running cost*, which is the total fee paid for running VM instances based on actual usage, and (ii) *transfer cost*, which charges the outbound transferred data from the VMs to CDStore clients (e.g., when a CDStore server informs a CDStore client of what unique shares need to be uploaded).
- *Backend cost:* The backend cost is the fee charged for storage. We consider backend cost in two parts: (i) *storage cost*, which charges the data storage for unique shares and file recipes, and (ii) *request cost*, which charges the storage requests (e.g., PUT, GET).

As an example, Table I shows the price plans of Amazon EC2 on-demand instances and S3 storage for our calculations of the VM and backend costs, respectively. Note that Amazon provides cheaper pricing options. For example, EC2 offers reserved instances that allow significant discounts for long-term usage, while charging upfront payments for resource reservations; Glacier[6] only charges one-third of S3's storage cost, but with a higher retrieval cost. In addition, the price plans vary across cloud providers. In particular, Rackspace charges the highest storage cost among the four providers, yet its request cost is free of charge. How to harness different pricing options for further cost savings is an important future work.

In practice, we reclaim the space of expired data to avoid unlimited data growth. Our analysis assumes that CDStore arranges containers by the backup time, so each container keeps either all active data, or all expired data. CDStore simply issues DELETE requests to expired containers. Since DELETE is free in all storage services, the reclaim cost is zero.

B. Case Study

We analyze the monthly backup cost of CDStore, and elaborate our calculations via a case study.

We derive our parameters from the backup datasets in EMC's field study [17]. The datasets cover up to 200TB of pre-deduplicated data. The deduplication ratio ranges from $2\times$ to $14\times$. The retention time ranges from 3 days to 3 years. The study also indicates that backup files are of large size, possibly over 100GB.

In our case study, suppose that an organization schedules weekly backups for its user data. Let the weekly backup size be 16TB per week, the deduplication ratio be $10\times$, the retention time be 26 weeks (about six months), and the average file size and average chunk (secret) size be 100GB and 8KB, respectively. We fix $(n, k) = (4, 3)$ for the four different cloud providers we consider. We assume that each CDStore server is always online, and each CDStore client has an average upload

[2]https://aws.amazon.com/pricing/services/
[3]https://cloud.google.com/pricing/
[4]https://azure.microsoft.com/en-us/pricing/
[5]http://www.rackspace.com/cloud/public-pricing/
[6]https://aws.amazon.com/glacier/pricing

TABLE I
AMAZON EC2 AND S3 PRICE PLANS (IN US$) IN NOVEMBER 2015. WE ONLY LIST THE ESSENTIAL ITEMS FOR OUR ANALYSIS.

(a) EC2 On-demand Instances

CPU	Storage (GB)	Price ($/hour)
4	80	0.265
8	160	0.529
4	800	1.018
8	1,600	2.035
16	3,200	4.07
32	6,400	8.14

(b) EC2 Monthly Transfer Prices

Data Range	Price ($/GB)
First 1GB	Free
Up to 10TB	0.120
Next 40TB	0.085
Next 100TB	0.082
Next 350TB	0.080

(c) S3 Monthly Storage Prices

Data Range	Price ($/GB)
First 1TB	0.0300
Next 49TB	0.0295
Next 450TB	0.0290
Next 500TB	0.0285
Next 4000TB	0.0280

(d) S3 Request Price

Request	Price ($/10,000 req)
GET	0.004
PUT	0.05
Delete	Free

bandwidth of 15MB/s. We now explain how we derive each cost component.

VM cost: We first examine the running cost. Our goal is to choose the cheapest VM based on two principles: (i) *CPU core principle*, in which there are enough CPU cores to serve CDStore clients, and (ii) *VM storage principle*, in which a VM can keep all file and share indices locally (see Section II-D) according to the estimated storage size and deduplication efficiency.

For the CPU core principle, we measure the average number of CDStore clients that can be concurrently served by dividing the amount of backup data (i.e., 16TB/week) by the per-client upload bandwidth (i.e., 15MB/s), so we have 1.85. If we use one CPU core to serve each CDStore client and reserve one additional CPU core for the main thread of a CDStore server, then we need at least three CPU cores here.

For the VM storage principle, we compute the average number of files managed in a VM by dividing the total amount of backups (i.e., 416TB over six months) by the average file size (i.e., 100GB), so we have around 4,260 files. We also compute the average number of unique shares managed in a VM by dividing the amount of unique data being stored after deduplication (i.e., 41.6TB for 10$\times$ deduplication ratio) by the average chunk size (i.e., 8KB), so we have around 5.58×10^9 shares. In our CDStore implementation, the file and share index entry sizes are 64 bytes and 72 bytes, respectively. Thus, we need around 370GB of VM local storage.

Based on the above derivations, we choose the VM type with four CPUs and 800GB storage for Amazon EC2 (see Table I(a)), whose monthly running cost is around US$740 (i.e., US$1.018/hour$\times$24$\times$365/12). Similarly, we can derive the monthly running costs for Google (US$180), Azure (US$700), and Rackspace (US$1,620).

The transfer cost of CDStore is actually very small. It is charged when a CDStore server informs a CDStore client of what shares need to be uploaded, such that each share identifier can be embedded in a single byte. For 16TB weekly backups, the monthly transfer cost of each VM is only around US$1.

Backend cost: We first examine the backend storage cost, which depends on the amounts of unique shares and file recipes being stored in each cloud. For unique shares, we divide the total amount of unique data being stored after deduplication (i.e., 41.6TB for 10$\times$ deduplication ratio) divided by k (i.e., 3), so we have 13.87TB per cloud[7]. For file recipes, our implementation contains 16-byte headers for all files and 40-byte fingerprint entries for all secrets in the original backup data. The total amount of file recipes stored is 16$\times$4,260 + 40$\times$5.58 $\times$ 10^{10} $\approx$ 2.03TB (the numbers of files and secrets are obtained based on the VM cost calculations). Thus, each cloud stores around 15.87TB of data, and the monthly storage cost is calculated from Amazon prices as $1 \times 1024 \times 0.03$ + $14.87 \times 1024 \times 0.0295$ $\approx$ US$480. Similarly, we can calculate the storage cost for other cloud services as Google (US$420), Azure US$380 and Rackspace US$1,480.

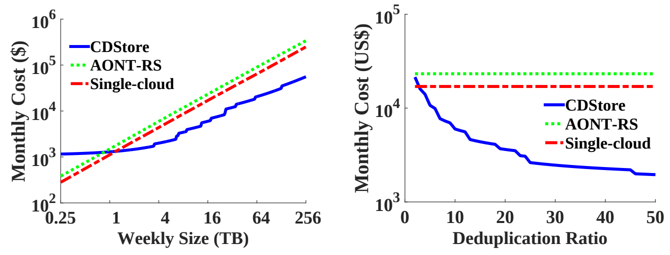
We next examine the backend request cost. From above, we upload to each cloud around 15.87TB/6 $\approx$ 2.65TB of data per month, including unique shares and file recipes. Recall that CDStore arranges storage in units of containers (see Section II-D). If the container size is 4MB, there will be 2.65TB/4MB $\approx$ 695,000 PUT requests to each cloud per month. For Amazon, the monthly request cost is only 695,000$\times$0.05/10,000 $\approx$ US$3. Both Google and Azure charge very low request costs, and Rackspace is free of charge for requests. Our study shows that the storage cost dominates over the request cost.

C. Cost Results

We evaluate the monthly costs of CDStore. We consider the same setting in Section III-B, except that we now vary the weekly backup size and deduplication ratio. We compare CDStore with two baseline cloud storage systems that do not support deduplication: (i) an AONT-RS-based multi-cloud system that achieves the same fault tolerance and security as CDStore, and (ii) a single-cloud system that incurs zero redundancy for fault tolerance, but encrypts user data with random keys. Both baseline systems incur no VM cost, and we assume that they have zero storage costs due to metadata and zero request costs.

For CDStore and AONT-RS-based multi-cloud system, we use the tiered price plans of Amazon, Google, Azure, and Rackspace; for the single-cloud system, we compute the cost of each of the four cloud providers and present the average. We show that even though CDStore introduces redundancy for fault tolerance and VMs for deduplication management, it still achieves cost savings through deduplication.

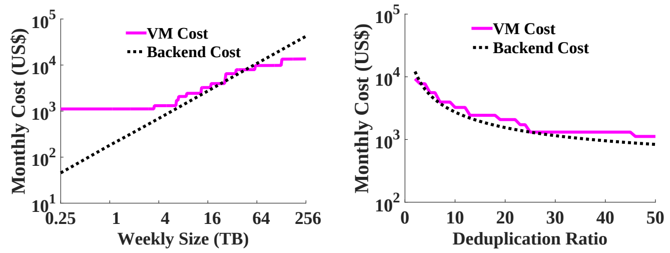
[7]For simplicity, we assume that the redundancy for fault tolerance is $\frac{n}{k}$. Note that the storage redundancy of AONT-RS [14] (on which CDStore builds) is slightly larger than $\frac{n}{k}$ (see Section II-B), but the difference is small and has limited impact on our cost results.



(a) Varying weekly backup size (step size is 1GB)

(b) Varying deduplication ratio (step size is one)

Fig. 4. Monthly costs of different cloud storage systems.



(a) Varying weekly backup size (step size is 1GB)

(b) Varying deduplication ratio (step size is one)

Fig. 5. Breakdown of the monthly costs of CDStore.

Figure 4 shows the monthly costs of the cloud storage systems. Figure 4(a) shows the monthly costs versus different weekly backup sizes, while we fix the deduplication ratio as $10\times$. CDStore incurs a higher cost than the baseline systems for the weekly backup size less than 1TB due to the VM cost, but achieves cost savings when the weekly backup size increases. For example, for the weekly backup size 64TB, CDStore has a monthly cost of US$20,600, while those of the AONT-RS and single-cloud systems are US$90,000 and US$64,000 (i.e., 77% and 68% of savings from CDStore), respectively. Note that the jagged curve of CDStore is caused by switching to the cheapest VM instance.

Figure 4(b) shows the monthly costs of the cloud storage systems versus different deduplication ratios, while we fix the weekly backup size as 16TB. The cost of CDStore decreases with the increase of deduplication ratio, yet those of both baseline systems remain unchanged. For the deduplication ratio $50\times$, CDStore incurs a monthly cost of US$1,950 only, and achieves 92% and 89% savings when compared to the AONT-RS-based (i.e., US$23,200) and single-cloud systems (i.e., US$17,000), respectively. To summarize, CDStore achieves high cost savings over the baseline systems for large weekly backup sizes and large deduplication ratios.

Figure 5 presents the breakdown of the monthly costs of CDStore, in terms of the VM and backend costs. Figure 5(a) shows the results versus different weekly backup sizes, while we fix the deduplication ratio as $10\times$. The VM cost is higher than the backend cost when the weekly backup size is small, yet the backend cost will become dominant as the weekly backup size increases. Figure 5(b) shows the results versus different deduplication ratios, while we fix the weekly backup size as 16TB. Both the VM and backend costs decrease as the deduplication ratio increases.

Our analysis focuses on the backup cost, and we briefly discuss the restore cost of CDStore. To restore a backup, a CDStore client downloads slightly more data than the original backup size to decode the original secrets (see Section II-B). The restore cost of CDStore is the sum of outbound transfer costs of k clouds; if all clouds have the same outbound transfer cost, it is roughly the same as that of the single cloud system.

IV. Conclusions

CDStore is a multi-cloud storage system for organizations to outsource backup and archival storage to public cloud vendors, with three goals in mind: reliability, security, and cost efficiency. The core design of CDStore is convergent dispersal, which augments secret sharing with the deduplication capability. CDStore also adopts two-stage deduplication to achieve bandwidth and storage savings and prevent side-channel attacks. Our cost analysis demonstrates that CDStore achieves significant cost savings via deduplication.

Acknowledgments

This work was supported in part by Research Grants Council of Hong Kong (ECS CUHK419212 and GRF CUHK413813) and Cisco University Research Program Fund (CG#593822) from Silicon Valley Community Foundation.

References

[1] M. Bellare, S. Keelveedhi, and T. Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In *Proc. USENIX Security*, 2013.
[2] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and secure storage in a cloud-of-clouds. *ACM Trans. on Storage*, 2013.
[3] A. Bessani, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, and P. Verissimo. SCFS: A shared cloud-backed file system. In *Proc. USENIX ATC*, 2014.
[4] V. Boyko. On the security properties of OAEP as an all-or-nothing transform. In *Proc. CRYPTO*, 1999.
[5] C. Cachin, R. Haas, and M. Vukolić. Dependable storage in the intercloud. IBM Research Report RZ 3783, 2010.
[6] 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 *Proc. IEEE ICDCS*, 2002.
[7] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proc. ACM CCS*, 2011.
[8] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy*, 2010.
[9] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang. NCCloud: Applying network coding for the storage repair in a cloud-of-clouds. In *Proc. USENIX FAST*, 2012.
[10] M. Li, C. Qin, and P. P. C. Lee. CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. In *Proc. USENIX ATC*, 2015.
[11] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *Proc. USENIX FAST*, 2011.
[12] M. Rabin. Fingerprint by random polynomials. Technical report, Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.
[13] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 1960.
[14] J. K. Resch and J. S. Plank. AONT-RS: Blending security and performance in dispersed storage systems. In *Proc. USENIX FAST*, 2011.
[15] R. L. Rivest. All-or-nothing encryption and the package transform. In *Proc. FSE*, 1997.
[16] A. Shamir. How to share a secret. *Communications of the ACM*, 1979.
[17] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu. Characteristics of backup workloads in production systems. In *Proc. USENIX FAST*, 2012.

Mingqiang Li is now with Hong Kong Advanced Technology Center, Ecosystem & Cloud Service Group, Lenovo Group Ltd. His current research interests include cloud computing and storage systems. Email: `mingqiangli.cn@gmail.com`.

Chuan Qin is now a PhD student at The Chinese University of Hong Kong. His research interests include cloud security and deduplication. Email: `cqin@cse.cuhk.edu.hk`.

Jingwei Li is now a postdoctoral researcher at The Chinese University of Hong Kong. His research interests include applied cryptography and cloud security. Email: `lijw1987@gmail.com`.

Patrick P. C. Lee is now an Associate Professor of the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests are in various applied/systems topics including storage systems, distributed systems and networks, operating systems, dependability, and security. Email: `pclee@cse.cuhk.edu.hk`.