Delta Encoding Overhead Analysis of Cloud Storage Systems using Client-side Encryption

Eric Henziger
Linköping University, Sweden

Niklas Carlsson
Linköping University, Sweden

***Abstract*—With client-side encryption (CSE), a user's data is encrypted before being transferred to a cloud provider. This ensures that only the intended user has access to the information, but complicates effective file synchronization (between different devices and the cloud). Motivated by prior findings that empirically show that the largest performance differences between popular CSE services (CSEs) and non-CSEs typically are related to the implementation of delta encoding solutions to reduce bandwidth usage, in this paper, we evaluate and provide insights into the practical CSE-related delta encoding overheads. First, we use targeted experiments to demonstrate the delta encoding problem associated with CSE and to compare the practical overhead differences associated with three example services implementing delta encoding. Second, we develop an analytic cost model and use it to show that a simple threshold-based CSE policy can reduce the bandwidth and storage usage seen by the best CSE considered here, that such a policy has a provable worst-case overhead within a factor two of the best non-CSE, and typically performs much better. The results are highly encouraging, and show that it is possible to provide CSE at limited additional overhead compared to non-CSE services.**

***Index Terms*—Client-side encryption; Cloud storage systems; Delta encoding; File synchronization; Bandwidth overheads**

I. Introduction

Cloud storage applications such as Dropbox, Google Drive, Microsoft OneDrive, and iCloud together have billions of active users [1]. These services provide users with flexible low-cost synchronization that enables easy access to files using multiple devices, regardless of geographical location.

However, despite their popularity and an increasing need for users being able to store information securely and confidentially, most of these services do not provide any guarantees regarding the confidentiality and integrity of the data stored. Instead, most of the popular services have direct access to the data itself and many of them are fairly blunt regarding the rights they retain for this data. For example, Dropbox's end-user agreements give them, including their affiliates and trusted third parties, the right to access, store and scan the data [2]. Similarly, Google's terms of service [3] gives Google "a worldwide license to use, host, store, reproduce, modify, [...], publicly display and distribute such content." where "such content" refers to the user's stored content.

Clearly, such terms are not acceptable for some users and content types. A solution to provide confidential cloud storage is to use client-side encryption (CSE). With CSE, the user's data is encrypted before being transferred to the cloud provider. This ensures that the content is transferred and stored in an encrypted format and that only clients with the appropriate decryption keys have access to the non-encrypted information. The value of CSE is further highlighted by surveillance backdoors [4] or software bugs [5].

While CSE improves content confidentiality for end users, it complicates file synchronization techniques such as deduplication and delta encoding, commonly used to reduce the traffic associated with personal cloud storage systems. Such bandwidth saving features are important since the bandwidth costs associated with synchronization traffic can be substantial.

To assess the overheads with CSE services (CSEs), we therefore recently set out to empirically measure and compare the overhead observed with four popular CSEs (Mega, Sync.com, SpiderOak, and Tresorit) and non-CSEs (Dropbox, iCloud, Google Drive, Microsoft OneDrive). Our initial findings were encouraging and showed that existing CSEs are able to implement CSE together with bandwidth saving features such as compression and deduplication with low additional overhead compared to the non-CSEs [6]. The main downside with CSE instead appears to be associated with the use of delta encoding, a feature that has been shown to help reduce Dropbox's synchronization traffic substantially [7]. For example, among the eight tested services only one CSE (SpiderOak) and two non-CSEs (Dropbox, iCloud) implement some form of delta encoding, and (most importantly) the overhead differences associated with these three encoding implementations were substantial.

In this paper, we use targeted experiments of the above services and a model-based analysis of CSE-based delta encoding overheads to demonstrate the delta encoding problem associated with CSE, characterize the practical overheads associated with the delta encodings used by these services, and (most importantly) to determine the potential room for further improvements. Our experiments show that much of the bandwidth and storage overheads associated with CSEs are due to CSE cloud providers not being able to decode delta encoding messages, and highlight that there are significant differences in the effectiveness in how delta encoding is implemented and that there is much room for improvements. A simple cost model is then developed that captures multi-device scenarios. Our model based results include worst-case bounds of the delta encoding penalty associated with CSEs, as well as a characterization of the CSE overheads observed under both synthetic workloads and example traces. Overall, the results show that bandwidth and storage costs of CSEs

can be worst-case bounded by a factor two of the best non-CSEs, with average differences within a factor 1.5 across both long-tailed (Pareto) and short-tailed (exponential, normal, and deterministic) delta encoding size distributions and a broad range of other workload parameters. These results demonstrate that there are significant cost saving opportunities not yet leveraged by current CSEs.

The remainder of the paper is organized as follows. Section II presents some of our previous empirical findings, motivating this work. Section III then provides targeted empirical example results, highlighting the delta-encoding problem associated with CSEs, before Section IV presents our model and model-driven results. Finally, related work and conclusions are presented in Sections V and VI, respectively.

II. Background and Motivating Findings

As noted in the introduction, we recently empirically measured and compared bandwidth saving features implemented and the client-side overheads observed with four popular CSEs (Mega, Sync.com, SpiderOak, and Tresorit) and non-CSEs (Dropbox, iCloud, Google Drive, Microsoft OneDrive) [6]. At a high level, the results showed larger differences within the two categories of services (CSEs and non-CSEs) than between the two categories themselves, and that the client-side resource usage of CPU, disk, and memory often were correlated to the set of bandwidth saving features (compression, deduplication, and delta encoding) that the services implemented. In particular, the services with the highest overhead were typically those that implemented all three bandwidth saving features (only SpiderOak and Dropbox). However, for the purpose of this paper, our results also highlighted that CSEs are able to implement CSE together with compression and deduplication at low (if any) additional overhead compared to the non-CSEs [6], but that implementing delta encoding effectively is much harder for CSEs. This is an important finding since synchronization bandwidth costs can be substantial for these services and delta encoding has been shown to substantially reduce Dropbox's bandwidth usage [7]. In this section, we summarize the main findings regarding delta encoding when evaluating four popular CSEs (Mega, Sync.com, SpiderOak, Tresorit) against four popular non-CSEs (Dropbox, iCloud, Google Drive, Microsoft OneDrive).

Few services implement delta encoding: Of the tested services, one CSE service (SpiderOak) and two non-CSEs (Dropbox, iCloud) implement delta encoding. To determine whether or not each service used (at least) some form of delta encoding, all eight client applications (four CSEs and four non-CSEs) underwent three basic tests. In all tests, we started with placing a 5 MB file into the application under test's sync folder, and then incrementally increased the size of this file in steps of 5 MB until the size reached 25 MB. In each step, we inserted 5 MB random bytes (i) at the end (append), (ii) at the beginning (prepend), or (iii) into a random position (random insert) of the file. For each test, we measured the number of uploaded bytes by recording the network traffic (using the Python modules `netifaces`, `pcapy`, and others) and analyzing the collected packet trace files. For these scenarios, in the ideal case, the number of uploaded bytes (in each step) by a client using delta encoding would be similar to the size of the change; i.e., 5 MB. On the other hand, for clients that did not take advantage of delta encoding the number of uploaded bytes is expected to be close to the file size after each modification; i.e., 5, 10, 15, 20, 25 MB.

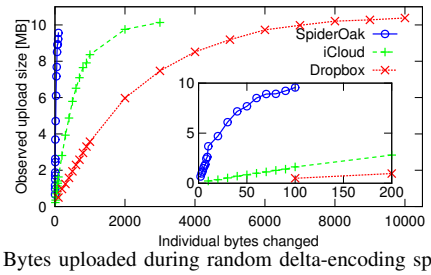


Fig. 1. Bytes uploaded during random delta-encoding sprinkle tests.

Delta encoding overhead differences between services are substantial: Comparing the bandwidth overheads associated with similar upload patterns we observed significant differences between the services. This was perhaps best illustrated by a worst-case scenario in which we randomly picked n bytes to change and then measured the number of bytes uploaded by the application. Figure 1 shows the results of these tests, when applied on a 10 MB file. (The insert zooms in on the low parameter range.)

These results show that (i) delta encodings work poorly on random file changes (and would hence not be useful on encrypted file data) and (ii) both Dropbox and iCloud significantly outperform SpiderOak. For example, consider the number of random bytes that can be changed before each service have uploaded the equivalent of another 10 MB file. For SpiderOak, on the order of 100 bytes are needed. In contrast, with iCloud and Dropbox approximately 2,000 and 6,000 bytes need to change, respectively. While the large differences partially may be due to implementation differences for sparse use cases, the results show that the room for improvements in SpiderOak's solution is significant.

To summarize, we found that the CSEs that we studied had been equally successful as the top-four non-CSEs to achieve bandwidth savings using compression and deduplication, but that it appears much harder for CSEs to implement effective delta encoding. This is perhaps why only SpiderOak of the tested CSEs implement delta encoding, and why both Dropbox and iCloud (the two non-CSEs performing delta encoding) significantly outperform SpiderOak.

The delta encoding problem with CSEs: Delta encoding is made difficult for CSEs mainly by the cloud provider not having access to the non-encrypted data and delta encoding being extremely inefficient on encrypted file versions (as they see close to randomly scattered file changes). Therefore, to allow the provider to seamlessly share the file with other devices of the client there are two main alternatives: (i) the client always upload the full file whenever they make a change, ensuring that the provider always has the latest copy to deliver, or (ii) the client submit encrypted versions of delta

encodings that the provider can store and deliver together with the original encrypted file.[1] The first option comes at significant upload overhead, since even a very small change result in the full file (or block) being uploaded. In contrast, the second option has low upload overhead, but much larger storage and download bandwidth overhead, since the provider must store and deliver the full change-log sequence needed by the downloading device to recreate the most recent file copy.

Three-out-of-four CSEs can perform arbitrarily bad: Remember that three out of the four studied CSEs (Mega, Sync.com, Tresorit) do not use delta encoding, but instead replace the full file when changes are made. Such an approach can be arbitrarily bad. For example, consider a small file change to a file of size N that requires a delta encoding of size Δ. In this case, a CSE replacing the full file would require an upload bandwidth proportional to N, whereas one that uses delta encoding would only need to upload Δ, resulting in a relative penalty of $\frac{cN-\Delta}{\Delta}$, where c captures the compression factor, for example. This penalty is unbounded when $N \to \infty$ and also becomes very large when Δ is small.

III. Use-case Driven Example Analysis

SpiderOak's bandwidth and server-side storage overhead: Among the CSEs, only SpiderOak performs delta encoding. However, their implementation is proprietary, making it non-trivial to analyze all details of their solution. Here, we use targeted experiments to provide initial insights into their delta encoding and the associate overheads.

First, and most importantly, it is easy to see that SpiderOak indeed stores a sequence of delta encodings on the server side, and that a second device downloads both the original file and the change-log of delta encodings. For example, consider a basic experiment in which we start with a file of size 10MB, consisting of random bytes, and then modify bytes 0-0.5 MB, bytes 1-1.5 MB, and so forth over 10 file changes. In this scenario, the original upload was of size 10.049 MB, and the 10 delta encoding updates were (measured in MB): 0.531, 1.058, 1.058, 1.058, 1.058, 0.531, 0.794, 0.794, 0.794, 0.794. In total, this resulted in 18.521 MB uploaded data; 3 MB more than the theoretic bound of 15 MB (if uploading only the size of the original file plus the changed data). When syncing with a second client, we could also confirm that the SpiderOak client indeed downloaded the full (18.5 MB) change log, and then recreated the file as seen on the first client. Again, this download size is expected since a provider should not be able to take advantage of the delta encodings to save server storage or download bandwidth for the second device.

The corresponding tests with Dropbox (non-CSE) looked quite different. While the original upload was somewhat larger (10.578 MB), the following 10 updates were smaller (measured in MB): 0.662, 0.540, 0.535, 0.535, 0.537, 0.535, 0.535, 0.540, 0.537, 0.535. In total, this resulted in 16.070 MB uploaded data. This shows that Dropbox uses more efficient delta encoding, only requiring 1 MB extra overhead. Furthermore, when syncing with a second device, we could confirm that the second device only had to download 10.353 MB, confirming that Dropbox efficiently applies delta encodings on the servers. The above examples clearly demonstrate some of the added overheads when applying delta encoding on CSEs.

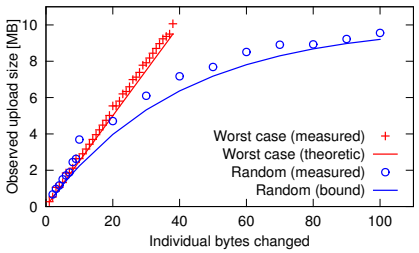


Fig. 2. Bytes uploaded by SpiderOak during worst-case and random tests.

Block-based delta encodings: Second, when investigating the behavior of SpiderOak uploads, we found that SpiderOak uses block-based delta encoding, with a block size of 2^{18} bytes (or 256 kB). To confirm this, we monitor the size of SpiderOak's delta encoding uploads when changing two bytes separated by x bytes. As long as x was less than 2^{18}, the uploads were 256 kB (plus some small overhead). However, as soon as x was 2^{18} or larger, the uploads became twice as big (512 kB). More generally, we have found that the size of the delta encoding uploads typically match well with the number of blocks affected by the file changes (plus some overhead). Furthermore, it does not matter how many bytes change within a block; only whether at least one byte changes within the block. Having said that, we have found that some changes appear to affect an extra block or add/use an extra block, for example, compared to what one otherwise would expect. For example, in the original example in this section, one would expect uploads to be either the size of 2 or 3 blocks. However, in practice, we also observed four 4-block uploads (1.058 MB), in addition to two 2-block uploads (0.531 MB) and four 3-block uploads (0.794 MB). As in the example, these inflated uploads typically occurs towards the beginning, possibly suggesting some form of block re-alignment.

Worst-case overhead: Based on our observations above, it is easy to identify worst-case workload patterns. In particular, if we focus only on the upload bandwidth associated with a single file synchronization, the largest ratio between the bytes uploaded and the bytes that actually are changed is 2^{18}. This is the case whenever exactly one byte is changed per 256 kB block with a byte change. We have validated this worst-case behavior using simple experiments. The "worst-case" lines in Figure 2 shows example results with a 10 MB file. With the file consisting of 40 blocks, only 40 bytes need to be changed for the delta encoding uploads to equal the file size (10 MB). We also see that the experiments nicely match the theoretic bound for the full range (0-40 bytes changed). For SpiderOak, we also include points for "random" and a lower bound $M(1-(1-\frac{1}{M})^n)$ times the 256 kB block size, where M=40. To derive this bound, note that the probability that an arbitrary block is

[1]While focus here is on files, it is possible to apply both the above approaches also on a per-block basis. This case typically increases complexity significantly, may reduce confidentiality, and requires the block structure to be passed along with block changes.

changed is equal to $\prod_{i=0}^{n-1}(1-\frac{K/M}{K-i})$, where K is the size of the file. This expression is upper bounded by $(1-\frac{1}{M})^n$. Taking the complement (resulting in a lower bound), and summing over all M blocks gives the result. The somewhat lose fit can be further explained by SpiderOak uploading more data than only the blocks (e.g., to specify changed blocks).

IV. Bounding CSEs' Delta Encoding Costs

In the previous section, we observed that the investigated CSEs either do not perform delta encoding (Mega, Sync.com, Tresorit) or do it inefficiently (SpiderOak). In this section, we use an analytic model to provide further insights into the potential room for improvements.

A. System models: Multi-device use case

Let us consider the total synchronization cost associated with a single file, including upload bandwidth, download bandwidth, and cloud storage costs. Without loss of generality, we measure all costs relative to the cost of uploading one unit of data to the cloud. With these normalized units, the costs of writing one unit of data is one, the costs of reading one unit of data is c_R, and the costs of storing one unit of data (in the cloud) is c_S per time unit.

For simplicity, we will first consider a *basic cost model* in which the storage cost c_S=0, and then extend our results to the *full cost model*. To illustrate the cost model, consider the initial SpiderOak example from Section III. Here, the first device uploaded 18.52 MB and the second device downloads the full 18.52 MB change-log, resulting in a total cost of $18.52+18.52c_R$, measured in units of MB. The corresponding cost when using Dropbox is $16.07+10.35c_R$. For this example, the cost ratio of the two techniques is between 1.15 (when c_R=0) and 1.79 (when $c_R\to\infty$). Furthermore, since the storage cost at any given time is proportional to the size of the current change-log (equal to the cost of a read event) and the SpiderOak change-log was monotonically increasing in our example, also the ratio of storage costs is upper bounded by 18.52/10.35≈1.79. Of course, the exact cost ratio depends on the exact timing of all upload and download instances. Finally, and most importantly, we note that there is nothing stopping a CSE to implement as efficient delta encoding as Dropbox. In this case, the CSE's (bandwidth) costs would be $16.07 + 16.07c_R$, providing a cost ratio between 1 and 1.55. Motivated by this observation, in the following, we will assume that the delta encoding scheme is given (e.g., Dropbox's) and instead focus on system policies that bound and/or minimize the CSE's cost ratio.

Regardless of cost model, in the following, we make the following system assumptions. First, only one client at a time has the write token for a file. Second, the cloud stores (i) a complete base copy, and (ii) a sequence of delta encodings, that combined can be used to reproduce the most recent file copy. Third, when the client with the write token writes to the cloud, a system policy determines whether the client should upload a new base copy or update the sequence of stored delta encodings. Fourth, clients reading the file from the cloud must download the full change log, including both the base copy and the stored delta encodings. Finally, at each such read instance, the client recreates the latest file copy and the system policy determines whether to upload it as a new base copy.

For our analysis, we consider an arbitrary event sequence $\mathcal{E}$, consisting of $N = |\mathcal{R}| + |\mathcal{W}|$ events, where $\mathcal{R}$ is the set of reads and $\mathcal{W}$ is the set of writes. Let i ($1 \le i \le N$) denote the i^{th} such event, and let t_i be the inter-event time between events $i-1$ and i, where 0 denote the start of the system. Furthermore, let S_i^c denote the size of the file as seen on the client with the write token, let S_i^s denote the size of the change-log seen on the cloud servers, and let us separate file size changes δ_i seen on the client and the delta encoding sizes Δ_i uploaded to the cloud servers, both capturing changes between copies $i-1$ and i. For simplicity, we assume non-decreasing file sizes (i.e., $\delta_i \ge 0$) and note that $\Delta_i \ge \delta_i$.

B. Baseline policies

Non-CSE: When CSE is not used, the server can maintain a copy of the client copy (using information in the delta encodings) and can therefore always serve requesting clients the latest copy. Assuming that $\Delta_i \le S_i^c$ (otherwise it is better that the client uploads the file itself), we therefore always have synchronization costs Δ_i for all writes and $c_R S_i^c$ for all reads. A lower bound can therefore easily be provided by:

$$\sum_{i\in\mathcal{W}} \Delta_i + c_R \sum_{i\in\mathcal{R}} S_i^c. \tag{1}$$

No delta encoding: In the case no delta encoding is used, changes to the file always result in the full file being uploaded. Therefore, all write events are associated with a cost S_i^c and read events with a cost $c_R S_i^c$; resulting in the following cost:

$$\sum_{i\in\mathcal{W}} S_i^c + c_R \sum_{i\in\mathcal{R}} S_i^c. \tag{2}$$

C. Binary system policies

We consider system policies that have two choices: (i) upload a new base copy (containing the client's most up-to-date file) that replaces both the old base copy and the change log, or (ii) append another delta encoding entry to the change log, based on the changes since the most recent entry. (We have also analyzed more complex policies in which the client also has the choice to overwrite part of the change log. A summary of these results can be found in Section IV-D.) Consider first a write event $i \in \mathcal{W}$. In this case, the first choice (i.e., to replace the base file) has cost S_i and the second choice (i.e., to upload another delta change) has cost Δ_i. For read events ($i \in \mathcal{R}$), assuming that a full copy of the file was most recently uploaded at time j, the read cost is equal to: $c_R S_i^s = c_R(S_j^c + \sum_{k=j+1}^{i-1} \Delta_k) = c_R(S_{i-1}^c + \sum_{k=j+1}^{i-1} \delta_k)$. In addition, when a policy decides to synchronize at read events, associate the event with an additional write cost S_i^c.

Optimal offline policy: Given a known event sequence $\mathcal{E}$, the optimal offline policy can be derived by considering all possible choices that the system may make and then picking the one with the lowest total cost. To find the optimal sequence

of such choices we use dynamic programming. First, let us define the sub problem $DP(M,m)$ as the problem of finding the minimum cost for the first M events given that we have not saved a full copy for the last m events ($m \leq M \leq N$). Second, note that the base case $DP(0,0)=S_0^c$. Third, we formulate the following recurrences for the *basic model*:

- $DP(M,0) = S_M^c + \min_{0 \leq k < M} DP(M-1,k)$, if $M \in \mathcal{W}$,
- $DP(M,m) = \Delta_M + DP(M-1,m-1)$, if $m > 0 \cap M \in \mathcal{W}$,
- $DP(M,0) = S_M^c + \min_{0 \leq k < M}(c_R(S_{M-k}^c + \sum_{j=M-k+1}^{M-1} \Delta_j) + DP(M-1,k))$, if $M \in \mathcal{R}$,
- $DP(M,m) = c_R(S_{M-m}^c + \sum_{j=M-m+1}^{M-1} \Delta_j) + DP(M-1,m-1)$, if $m > 0 \cap M \in \mathcal{R}$.

Finally, to solve the optimization problem, a two-level nested for-loop ($0 \leq M \leq N$ and $0 \leq m \leq M$) is used, starting from the base case $DP(0,0)$ until all DPs up to $DP(N,N)$ have been calculated. The optimal value is then given by $\min_{0 \leq m \leq N} DP(N,m)$, and the optimal solution is obtained using parent pointers (at a complexity of $\mathcal{O}(N^2)$).

Threshold-based policy with worst-case guarantees: Consider a simple policy that replaces the base file (i.e., option one) at write event i whenever $S_{i-1}^s + \Delta_i \geq 2S_i^c$. Intuitively, this ensures that (i) the cumulative write cost to the server remains below twice the file size, and (ii) the "read cost" always is less than twice the cost of downloading the actual client-side file. More formally, it can be shown that this policy has a cost ratio relative to the non-CSE policy within a factor 2, and hence has a delivery cost that is guaranteed to be within a factor 2 of optimal, regardless of the read-write event sequence. This is formalized in Theorem 1.

Theorem 1. *The above threshold-based policy has a cost ratio relative to the non-CSE policy within a factor 2.*

Proof. Consider an arbitrary event sequence $\mathcal{E} = \mathcal{R} \cup \mathcal{W}$. We next bound the cost ratio based on worst-case sequences, selected by an adversary. First, without loss of generality, assume that the size of the file (both at the client and the server) is S_0 after the initial copy has been uploaded. At this time, the ratio is one; clearly, satisfying the worst-case bound of 2. We next use induction on the number of sync events considered thus far, and consider the cumulative write and read costs for all event up-to and including the next sync event (if there is one). For this step, let us assume that the bound holds up to a sync event A (including the initial upload), and let event B be the next sync event, or the end of the event sequence, whichever comes first.

Now, let C_A and C_A^* be the cost of the CSE policy and the (optimal) non-CSE cost, respectively, up-to the time of event A. First, consider an arbitrary event b such that $A < b < B$. For this case, we can bound C_b^* by inserting (i) $C_A \leq 2C_A^*$ and (ii) $S_j^s < 2S_j^c$ ($A < j < B$) into the expression for the policy cost at event b and relating this to the corresponding non-CSE cost. More specifically, we have: $C_b = C_A + \sum_{k \in \mathcal{W} | A < k \leq b} \Delta_k + c_R \sum_{k \in \mathcal{R} | A < k \leq b} S_k^s \leq 2C_A^* + 2\sum_{k \in \mathcal{W} | A < k \leq b} \Delta_k + 2c_R \sum_{k \in \mathcal{R} | A < k \leq b} S_k^c = 2C_b^*$. Second, consider the case cost at event B. For this case, we also use that $S_B^c \leq \sum_{k \in \mathcal{W} | A < k \leq B} \Delta_k$. (To see this, note that $2S_B^c \leq S_{B-1}^s + \Delta_B = S_A^c + \sum_{k \in \mathcal{W} | A < k \leq B} \Delta_k \leq S_B^c + \sum_{k \in \mathcal{W} | A < k \leq B} \Delta_k \leq S_B^c$, which after subtraction of S_B^c on both sides gives us the results.) Insertion into the expression for the cumulated policy cost at event b then gives us: $C_B = C_A + S_B^c + \sum_{k \in \mathcal{W} | A < k < B} \Delta_k + c_R \sum_{k \in \mathcal{R} | A < k < B} S_k^s \leq 2C_A^* + 2\sum_{k \in \mathcal{W} | A < k \leq B} \Delta_k + 2c_R \sum_{k \in \mathcal{R} | A < k < B} S_k^c = 2C_B^*$. □

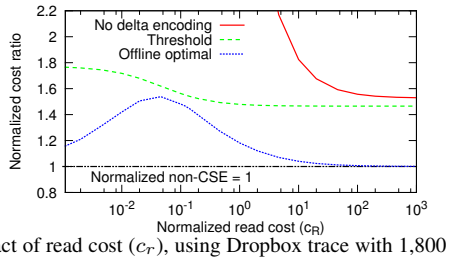


Fig. 3. Impact of read cost (c_r), using Dropbox trace with 1,800 write events.

Furthermore, the bound is tight. For example, consider the case when the adversary gives an event sequence in which we almost entirely change the file content (i.e., $(\Delta_1 - \delta_1) = S_0^c - \epsilon$, for some small ϵ), but do not increase the file size (i.e., $\delta_1 = 0$), followed by a very large number of reads. In this case, the read cost will dominate, and the ratio will approach $\frac{2S_0^c - \epsilon}{S_0^c}$, which as $\epsilon \to 0$ approaches 2. It may be tempting to try to use a different threshold than 2. However, this only leads to looser worst-case bounds, showing that our policy provides an optimized worst-case bound.

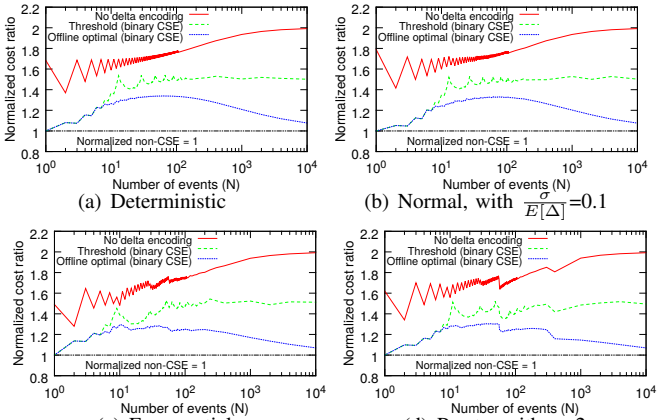
Full model: The above results extends naturally to the full model, where we also must add up the cumulative storage costs. For the non-CSE baseline, we simply add $c_S \sum_i t_i S_{i-1}^c$ to equation (1). For the optimal offline policy, additional storage terms $c_S T_i S_{i-1}^s$ are added to each equation in the DP formulation. Otherwise, the algorithm and solution method remains the same. For the threshold-based policy, no changes are needed. To see that the bound still holds when adding storage costs, note that $S_i^s \leq 2S_i^c$ at all times. Therefore, we can easily add the corresponding terms (i.e., $\sum_{k \in \mathcal{E} | A < k \leq B} S_{k-1}^s t_k$ and $2\sum_{k \in \mathcal{E} | A < k \leq B} S_{k-1}^c t_k$) to the respective sides of the cost inequalities in the proof, without otherwise affecting the proof.

Trace-based example comparison: Figure 3 shows the cost ratio as a function of the relative read cost c_R, in a trace-based scenario in which we recorded the size of the delta encodings made by our Dropbox client, as we made 1,800 file modifications, and emulated another 942 read events. For this scenario, we started with a 1MB file, and then randomly selected one of three types of file modifications: append, insert, or overwrite. Each modification was of a random size, in the range 100 bytes to 1MB, and the write location of insertions and overwrites were selected uniformly at random. As desired, the threshold policy caps the maximum cost penalty and for intermediate read costs performs close to optimal. Naturally, when the read cost is small the relative cost ratio of not using delta encoding (and hence uploading the full file each time) is very large. On the other, when the read cost (or storage cost) is large, the no delta encoding policy can perform similar (as shown) or better than the threshold policy. Depending on the workloads and relative cost terms, different cloud providers may therefore

lean towards different policies. Interesting future work could include the development of adaptive hybrid policies.

Impact of workloads: To provide some intuition for the impact that the workloads have on the relative cost differences, we performed trace-based simulations for four different distributions of the delta encoding sizes, ranging from short-tailed distributions (deterministic, normal, and exponential) to long-tailed distributions (Pareto). We also varied the number of events N, the normalized read cost c_R, the relative client-side file increase $\frac{\delta}{\Delta}$, the average delta encoding size $E[\Delta]$, and the ratio of read-write events $\frac{|\mathcal{R}|}{|\mathcal{W}|}$.

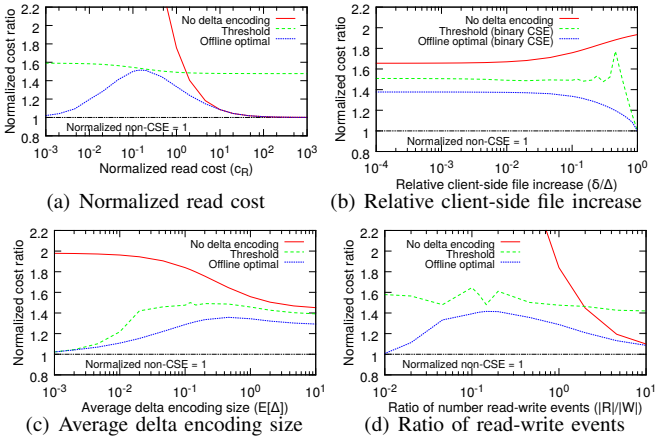
In general, we have found small differences between the different distributions. This is shown in Figure 4, where we show the normalized cost ratio as a function of N, for each of the four distributions. Figure 5 shows the impact of the different workload or cost parameters (using the deterministic distribution). In each figure, we vary one parameter at a time, while keeping the others fixed at their default values: N=100, c_R=1, $\frac{\delta}{\Delta}$=0.1, $E[\Delta]$=0.2, and $\frac{|\mathcal{R}|}{|\mathcal{W}|}$=1. Referring to Figure 4, the initial zig-zag pattern for the *no delta encoding* baseline are due to every second event being a read event (smallest possible cost with this policy) and write event (high cost with this policy). We also see that the penalty of this policy gets worse with time (i.e., increasing N), when the relative read cost is low (i.e., c_R is small), the relative delta overhead decreases (i.e., $\frac{\delta}{\Delta}$ increases), we have small delta changes (i.e., Δ is small), or there are many more writes than reads (i.e., $\frac{|\mathcal{R}|}{|\mathcal{W}|}$ is small). In contrast, the *threshold policy* typically closes most of the gap to what is achievable with a *non-CSE policy* (normalized lower bound of one), and in many case performs close to the *optimal offline policy*. Finally, comparing the optimal offline CSE policy with the lower bound when not using CSE, we note that there is an inherent penalty to using CSE in general, with the penalty peaking around 1.5.



(a) Deterministic (b) Normal, with $\frac{\sigma}{E[\Delta]}$=0.1

(c) Exponential (d) Pareto, with α=2

Fig. 4. Impact of size distributions and number of events.



(a) Normalized read cost (b) Relative client-side file increase

(c) Average delta encoding size (d) Ratio of read-write events

Fig. 5. Policy comparison across parameters. Example results based on deterministically sized delta encodings Δ.

D. Further policy flexibility and optimizations

In this section we briefly provide some insights into the potential value added by also allowing the system policy to overwrite part of the change log. More specifically, assuming that a sequence of k delta encoding changes currently are stored in the log, the client can now make one of the following choices: (i) replace the base file, (ii) append a delta change corresponding to the change k to $k+1$, or (iii) replace delta changes j to $k+1$ with a single new entry that include the delta encoding change between copy $j-1$ and $k+1$, where $1 \leq j \leq k$ and copy 0 corresponds to the base file.

Optimal offline policy: In contrast to with the binary system policy, we have not found any computationally feasible way to find the offline optimal when the number of events N is large. The reason is that the best choice at each such event depend on what choices have been made in the past. The dynamic programming approach is therefore not applicable. Having said that, dynamic programming can be used to effectively find the exact number of candidate solutions that need to be considered by a brute force method. To see this, note that the number of candidate solutions is equal to the number of leafs in a tree with depth N, in which branch points corresponds to candidate choices made by the client. Noting that a log with k delta encoding entries (beyond the basefile) would have $k+2$ branch choices, we can now write out a recurrence on the number of leafs $L(c,C)$ of a sub-tree in which we start with a log with c branch choices (i.e., $c=k+2$) and at the last level have at most C choices (in the case the log has grown by one in each step): $L(c,C)=\sum_{i=2}^{c+1} L(i, C+i-k-1)$. Adding the base case $L(c,c)=c$, it is now easy to calculate the total candidate solutions by iteratively calculating $L(2,N+1)$ using a double-nested for-loop ($2 \leq C \leq N+1; C+1 \geq c \geq 2$). Furthermore, leveraging the structure of the tree, it easy to prove that the leafs in the tree grows faster than $\Omega(3^N)$, when $N \geq 5$, and numeric evaluation suggests that the number of leafs may be upper bounded by $\mathcal{O}(4^N)$.[2]

Greedy policy: To decide when and how much the change logs should be pruned, a simple greedy policy can be used that weight the importance of (i) using small delta encoding updates, and (ii) reducing the log size when given such

[2]Our upper-bound conjecture is based on studying the tree structure and numeric evaluations of the ratio $L(2,C)/L(2,C-1)$ using two different DP formulations. Using double precision, these two versions could solve the problem for N up to 518 (i.e., $C=519$), so based on pessimistic linear extrapolation with $\log(\log(\log(C)))$ transform we can only state that the upper bound appears sound up to C of 1,000. To put the complexity into perspective, we also note that $L(2,519) = 1.4 \cdot 10^{308}$.

opportunities. In particular, at each choice point j, the policy uploads the delta change $\Delta_{i(j)^*,j}$ that minimize the following objective function:

$$\arg\min_{i \in \log} \Delta_{i,j} + fc_R(S_i^s + \Delta_{i,j}), \quad (3)$$

where S_i^s is the log size up-to-and-including entry i and f is a policy parameter, whenever the corresponding objective value is less than $(1+fc_R)S_j^c$, and otherwise updates the base file itself. Here, $i(j)^*$ is the best choice i, given current j. When f=0, all weight is given to minimize the upload bandwidth usage (i.e., $\Delta_{i,j}$), and, when $f \to \infty$, to reduce the log size.

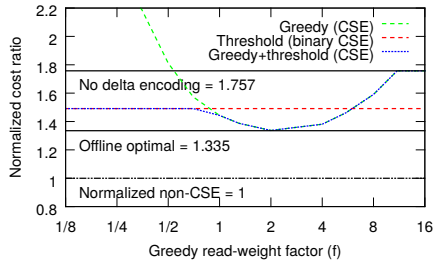
Threshold extension: Similar to the binary case, we can ensure a worst-case ratio of 2 by augmenting the greedy policy by always replacing the base file whenever $i(j)^*$ satisfies: $S^s_{i(j)^*} + \Delta_{i(j)^*,j} \geq 2S_j^c$.

Policy comparison: We have found that the greedy policy is best when combined with the threshold policy, as the pure greedy policy can build very large logs whenever the selected read-weight factor f is too much smaller than the actual read/write ratio ($\frac{|\mathcal{R}|}{|\mathcal{W}|}$). This is illustrated in Figure 6. Here, we plot the normalized cost ratio as a function of f, for our default scenario in which $\frac{|\mathcal{R}|}{|\mathcal{W}|}$=1. To capture the multi-step $\Delta_{i,j}$ changes, we assume a combination of appends and changes to random independent bytes. Under these assumptions, $\delta_{i,j} = \delta_{i,j-1} + \delta_{j-1,j}$ and $\Delta_{i,j} = \Delta_{i,j-1} + \delta_{j-1,j} + S^s_{j-1}(1 - \frac{\Delta_{i,j-1}}{S^s_{j-1}})\frac{\Delta_{j-1,j} - \delta_{j-1,j}}{S^s_{j-1}}$, where $\delta_{j-1,j}$ and $\Delta_{j-1,j}$ are the one-step changes (following the same distributions as in prior experiments), and the server-side log size S^s_{j-1} easily can be calculated given the current log content. Interestingly, the added flexibility does not provide much improvement, and only for a limited region of the parameter space. In fact, when the read-weight factor f is too large (relative to the actual read-write ratio), even the threshold-based greedy variation can be outperformed by the (simpler) binary threshold policy. The above observations are consistent when varying the other workload parameters (results mostly omitted), including when changing the ratio between appends (δ) and random byte changes ($\Delta - \delta$), as illustrated in Figure 6(b). Our results show that a simple binary threshold policy indeed may be a good candidate to use in practice.

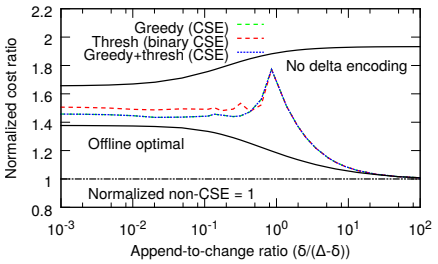
V. Related Work

Common techniques to reduce the cloud storage and bandwidth costs include deduplication [8], [9], [10], [11], [12], [13], [14], delta encoding [15], [16], device-to-device synchronization [17], compression [18], and caching [19]. Motivated by high storage costs and significant redundancy in the data stored (both by individual clients and across clients), most of these works have focused on deduplication [8], [9], [10]. This include effective and secure data deduplication solutions that combine convergent encryption and clever key management [11], [12], [13], [14]. It is therefore perhaps not surprising that we have observed that three of the four considered CSEs implement effective deduplication [6].

However, over the past few decades, storage costs have dropped multiple orders of magnitude (e.g., $ per MB), and



(a) Greedy read-weight factor



(b) Append-to-change ratio

Fig. 6. Multi-step greedy policy tests. (Default scenario.)

are today relatively cheaper compared to bandwidth costs than in the past. Despite this evolution making a strong case for the delta encoding problem becoming relatively more important, less work has been done on delta encoding solutions, and such solutions are relatively less frequently implemented by the most popular services [6]. We note that the importance of effective delta encoding solutions is further augmented by much file data being modified many times and that data often is accessed from multiple devices. These arguments are confirmed by the workloads observed by researchers studying the most beneficial user behaviors for effective file synchronization [20] and the client behavior itself [21], [22], [18].

Similar to us, Drago et al. [7] find that Dropbox is able to reduce the synchronization traffic significantly by using delta encoding. Others have implemented middleware solutions (e.g., that can be used in conjunction with Dropbox) to improve the synchronization process [16], [23] or proposed techniques that try to reduce the synchronization traffic by aggregating multiple changes or in other ways optimize the delta encodings traffic [15], [16]. For example, Lee et al. [15] present an MDP-based solution that tries to optimize the tradeoff between file consistency differences and bandwidth savings. In contrast to these works, we consider the case of CSE, and note that the bounds and threshold policy described here are valid together with any such technique. For the purpose of our empirical evaluation we use Dropbox traces, allowing us to capture the performance of the current state-of-the-art production algorithms.

There is very limited work characterizing public CSEs. As described in Section II, where we summarize our prior characterization work [6], we have previously empirically measured and compared the featured implemented and the overhead observed when using four popular CSEs and four popular non-CSEs. In this work we substantially expand on these initial findings and provide both new targeted experiments (Section III) and a novel model-based analysis (Section IV),

each of which provide new system insights into the important delta encoding problem associated with CSEs.

Mager et al. [24] studied the now discontinued CSE service Wuala. In the context of CSEs, others have uncovered weaknesses when enabling data sharing [25] or proposed solutions for sharing data in dynamic groups over an untrusted cloud storage service [26], [27], [28]. Yet others have considered many other interesting security/privacy related cloud aspects than CSE [29], [30], [31], [32], [33], [34]. In contrast to the above works, we analyze the delta encoding problem of CSEs.

VI. Conclusions

This paper focuses on the delta encoding overhead associated with CSE. Using a combination of targeted empirical experiments and a model-based analysis of these overheads, we characterize the current state-of-the-art and provide insights into further improvements. Our experiments with eight services (four CSEs and four non-CSEs) show that the performance overheads associated with implementing delta encoding typically are substantially higher than other bandwidth saving features such as compression and deduplication. Through targeted experiments with the services implementing some form of delta encoding (SpiderOak among the CSEs and Dropbox + iCloud among non-CSEs), we evaluate these differences and provide insights and model the effects of the fixed-sized blocks used by SpiderOak. We then develop an analytic cost model that allows us to compare and contrast the best possible CSE delta encoding policies, assuming the same delta encoding algorithm is implemented as the best non-CSE (allowing fair comparison). Finally, using the model, we show that a simple threshold-based CSE policy has a worst-case cost within a factor 2 of the corresponding non-CSE, with numeric results showing that such a policy typically would perform much better, has an average difference (compared to optimal) below 1.5 across a wide range of delta size distributions (Pareto, exponential, normal, deterministic) and other parameters, and that the policy improves on the SpiderOak results. The results are encouraging as they show significant cost saving opportunities for CSEs and demonstrate that CSEs can achieve most of the cost savings that delta encoding provides.

Acknowledgements: This work was funded in part by the Swedish Research Council (VR).

References

[1] Cisco Public, "Cisco Global Cloud Index: Forecast and Methodology, 2015-2020," Tech. Rep., 2016.

[2] Dropbox Inc. (2019) Dropbox Terms of Service. [Online]. Available: https://www.dropbox.com/terms

[3] Google LLC. (2018) Google Terms of Service. [Online]. Available: https://www.google.com/intl/en/policies/terms/

[4] G. Greenwald, E. MacAskill, L. Poitras, S. Ackerman, and D. Rushe, "Microsoft handed the NSA access to encrypted messages," *The Guardian*, 2013. [Online]. Available: https://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data

[5] Arash Ferdowsi - Dropbox Inc. (2011) Yesterday's Authentication Bug. [Online]. Available: https://blogs.dropbox.com/dropbox/2011/06/yesterdays-authentication-bug/

[6] E. Henziger and N. Carlsson, "The overhead of confidentiality and client-side encryption in cloud storage systems," in *Proc. IEEE/ACM UCC*, 2019.

[7] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, "Benchmarking Personal Cloud Storage," in *Proc. IMC*, 2013.

[8] D. T. Meyer and W. J. Bolosky, "A Study of Practical Deduplication," *ACM Trans. on Storage*, vol. 7, no. 4, 2012.

[9] R. N. Widodo, H. Lim, and M. Atiquzzaman, "A new content-defined chunking algorithm for data deduplication in cloud storage," *Future Generation Computer Systems*, vol. 71, 2017.

[10] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side Channels in Cloud Services: Deduplication in Cloud Storage," *IEEE Security & Privacy*, vol. 8, no. 6, 2010.

[11] M. Storer, K. Greenan, D. Long, and E. Miller, "Secure data deduplication," in *Proc. ACM Storage Security and Survivability workshop*, 2008.

[12] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1615–1625, 2013.

[13] P. Puzio, R. Molva, M. Önen, and S. Loureiro, "Cloudedup: Secure deduplication with encrypted data for cloud storage," in *Proc. IEEE CloudCom*, 2013.

[14] J. Hur, D. Koo, Y. Shin, and K. Kang, "Secure data deduplication with dynamic ownership management in cloud storage," *IEEE Trans. on Knowledge and Data Engineering*, vol. 28, pp. 3113–3125, 2016.

[15] G. Lee, H. Ko, and S. Pack, "An Efficient Delta Synchronization Algorithm for Mobile Cloud Storage Applications," *IEEE Trans. on Services Computing*, vol. 10, no. 3, 2017.

[16] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai, "Efficient Batched Synchronization in Dropbox-like Cloud Storage Services," in *Proc. ACM/IFIP/USENIX Middleware*, 2013.

[17] G. Gonçalves, A. B. Vieira, I. Drago, A. P. C. Da Silva, and J. M. Almeida, "Cost-benefit tradeoffs of content sharing in personal cloud storage," in *Proc. IEEE MASCOTS*, 2017.

[18] Z. Li, Y. Dai, G. Chen, and Y. Liu, "Towards Network-level Efficiency for Cloud Storage Services," in *Proc. IMC*, 2014.

[19] G. Gonçalves, I. Drago, A. P. C. Da Silva, A. B. Vieira, and J. M. Almeida, "The impact of content sharing on cloud storage bandwidth consumption," *IEEE Internet Computing*, vol. 20, no. 4, pp. 26–35, 2016.

[20] R. Gracia-Tinedo, Y. Tian, J. Sampe, H. Harkous, J. Lenton, P. G. Lopez, M. Sanchez-Artigas, and M. Vukolic, "Dissecting UbuntuOne: Autopsy of a Global-scale Personal Cloud Back-end," in *Proc. IMC*, 2015.

[21] G. Goncalves, I. Drago, A. da Silva, A. B. Vieira, and J. M. Almeida, "Modeling the Dropbox Client Behavior," in *Proc. IEEE ICC*, 2014.

[22] I. Drago, M. Mellia, M. M. Munafò, A. Sperotto, R. Sadre, and A. Pras, "Inside Dropbox: Understanding Personal Cloud Storage Services," in *Proc. IMC*, 2012.

[23] P. G. Lopez, M. Sanchez-Artigas, S. Toda, C. Cotes, and J. Lenton, "StackSync: Bringing Elasticity to Dropbox-like File Synchronization," in *Proc. ACM Middleware*, 2014.

[24] T. Mager, E. Biersack, and P. Michiardi, "A Measurement Study of the Wuala On-line Storage Service," in *Proc. IEEE P2P*, 2012.

[25] D. C. Wilson and G. Ateniese, ""To Share or not to Share" in Client-Side Encrypted Clouds," in *Proc. ISC*, 2014.

[26] I. Lam, S. Szebeni, and L. Buttyan, "Invitation-oriented TGDH: Key management for dynamic groups in an asynchronous communication model," in *Proc. IEEE ICPP Workshops*, 2012.

[27] ——, "Tresorium: Cryptographic file system for dynamic groups over untrusted cloud storage," in *Proc. IEEE ICPP Workshops*, 2012.

[28] I. Lam, S. Szebeni, and T. Koczka, "Client-side encryption with DRM," 2015, US Patent 9,129,095.

[29] A. Bessani, M. Correia, B. Quaresma, F. Andre, and P. Sousa, "DepSky: Dependable and Secure Storage in a Cloud-of-clouds," *ACM Trans. on Storage*, vol. 9, no. 4, p. 12, 2013.

[30] A. N. Bessani, R. Mendes, T. Oliveira, N. F. Neves, M. Correia, M. Pasin, and P. Verissimo, "SCFS: A Shared Cloud-backed File System," in *Proc. USENIX ATC*, 2014.

[31] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of Ownership in Remote Storage Systems," in *Proc. ACM CCS*, 2011.

[32] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish, "Depot: Cloud Storage with Minimal Trust," *ACM Trans. on Computer Systems*, vol. 29, no. 4, 2011.

[33] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. R. Weippl, "Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space," in *Proc. USENIX Security*, 2011.

[34] W. C. Garrison III, A. Shull, S. Myers, and A. J. Lee, "On the practicality of cryptographically enforcing dynamic access control policies in the cloud," in *Proc. IEEE S&P*, 2016.