Efficient User Opt-out from Block Stores

Erwan Le Merrer
Technicolor
France
erwan.lemerrer@technicolor

Nicolas Le Scouarnec
Technicolor
France
nicolas.lescouarnec@technicolor

Abstract

Massive collection of user or device data is a growing trend for personalizing services. Cheap and scalable storage is key to allow advanced analysis on the long run. In this context, block (HDFS) and columnar (Dremel, Parquet) stores are increasingly leveraged. Data protection acts, or simply the will for transparency to the user, impose to opt-out data on demand. Unfortunately, those data stores have departed from traditional databases, and do not provide efficient access and deletion to specific bits of data. In this paper, we study how to cost-efficiently opt-out user data from these stores. We apply two intuitive strategies (systematic erasure and encryption) to the context of big data systems. We model their respective costs and show that in the context of a service running atop Amazon Web Services, there is no general winner strategy (except in the special case where data cannot be compressed). Application constraints, such as data arrival and user opt-out rates, should then be considered to select the most cost-efficient opt-out strategy, while practical means of actions are the sharding policy and the careful setting of block sizes.

***Index Terms*—Block/column stores, opt-out, data deletion, cost modeling, AWS**

I. Introduction

With the promises of the Internet of Things, and the current trend for user-data analytics, big data processing systems are becoming common in the industry. A frequent design pattern (*e.g.*, the lambda architecture [1]) is to reliably store the raw data on a *cheap* storage layer. This scalable data store does not have advanced features, but offers a good cost versus performance trade-off, allowing to efficiently read/write large batches of data. These storage systems (*e.g.*, HDFS [2], GFS [3]) have been optimized for write-once and read-many access patterns that are common to most analytic tasks. To limit the cost associated with each data record that is collected by the service, these objects are concatenated into large files (called *blocks*). Blocks are then written to a distributed file system. One notable characteristic is that those blocks are large (HDFS [2], S3 [4], Azure [5], or GFS have block sizes ranging from 64MB to several GB), in order to optimize the read/write throughput. As an illustration, an Internet of Things application see data objects from many devices concatenated into a single block, implying that this block in practice contains objects from many users.

Paper accepted for publication in the IEEE International Conference on Cloud Engineering (IC2E), April 2016.

In this paper, we study the problem of efficient user opt-out: we review two strategies for deleting data associated with a user that wants to leave the service provider's system. This problem is becoming salient in the industry, due to the current or forthcoming legal obligations of removing user personal data on demand (see *e.g.*, [6] for the European Union), while minimizing the operational cost of implementing such an operation. To the best of our knowledge, the question of efficient data opt-out from those block stores has not been addressed yet. We make the following contributions:

- We review two intuitive strategies for removing user-data from block stores. The first one simply fetches all possible blocks that may contain data from the requesting user, before removing user's objects and writing blocks back into the block store. The second approach is based on data encryption, by overwriting the encryption key associated with a user to make her data unreadable in the block store. The first approach stresses the computing and throughput performances of the system, while the second one puts the load on the overall amount of data stored. We discuss their possible integration into big data systems such as logs on HDFS/S3 and columnar storage (*e.g.*, Dremel [7], Parquet [8]).
- We model the cost associated with each approach depending on system parameters like the data arrival and user opt-out rates. This model highlights the cost tradeoff between computing and storing data in such an opt-out enabled system. Furthermore, this general model will permit the design of other more complex approaches by future works, in the same framework.

The remaining of this paper is organized as follows. Section II reviews the storage layers employed in big data systems, and discusses deletion strategies in related domains. Section III models the cost tradeoff in storing and erasing data. Section IV studies the actual cost of operation on AWS (EC2 and S3) and evaluates four application scenarios in this context. Section V reviews related work and Section VI finally concludes.

II. Background

A popular practice in big data systems is to store the objects received in raw format onto a cheap storage layer, so as to allow re-processing data for new analytics. This approach is advocated in the Lambda architecture [1]. At the other

extreme, user data are also stored in raw format for archival purposes [9]. For throughput efficiency reasons, this leads to the storage of large blocks (*e.g.*, HDFS [2] is not optimized for small files) created from the objects concatenation from many devices/users, as described in Twitter's architecture [10] and LinkedIn's Camus [11].

To achieve a low operational cost, and because no immediate (*e.g.*, indexed) accesses are needed, the objects are stored directly as blocks on file systems optimized for data analytics, such as GFS [3] or HDFS [2]. This differs to the use of databases (as e.g., Cassandra, that allows more efficient access to stored objects, but at the price of a higher storage overhead as compared to raw storage [12]). When using a cloud block storage as S3, it is interesting to store large blocks to amortize the access overhead and the cost per request. As described in [5], [10], [11], the blocks are concatenated into large files. To deal with large number of devices and distribute the object collection on several servers, devices are assigned to *shards*. As the mapping between devices and shards is deterministically fixed, it is possible to access only the relevant shard whenever the object of a given user needs to be accessed or removed.

In order to limit the cost impact of big data storage, compression is often used (*e.g.*, snappy, gzip) with compression rate around 0.5 [13] on a binary serialization format such as Avro [14]. A recent trend [7], [8] for speeding up the processing is to organize these files into columns instead of a simple concatenation of records so as to speed up analysis that access only a few features within each record. This structure also leads to better compressibility with compression rate around 0.25 [7], [8] when compared to a binary serialization format. Of course, block-level compression does not apply when storing binary objects naturally compressed prior to their insertion, such as photos for instance [15].

Traditional filesystems make modifications to a file by overwriting its previous versions. This leads to the most straightforward deletion strategy for cloud block stores, that is to fetch the block containing objects from a user willing to opt-out, to remove objects it contains, and to send back the resulting block into the storage layer.

Deletion using encryption is widely adopted on disk storage: all data are encrypted, and the encryption key is securely overwritten when these data need to be removed. This lead to fast deletion as only a small key is overwritten instead of possibly large files. Please refer to survey [16] for detailed information. One key distinction when porting this concept into cloud block stores is that a hard drive size is fixed, as opposed to dynamically allocated in the cloud context. In this paper, we examine the applicability of this technique to data warehousing, studying the integration of encryption-based deletion into block and column stores.

III. Modeling costs of user opt-out

In this section, we model the efficient opt-out problem, before providing a formula for computing the associated operational costs. We sum up the core parameters of the problem in Table I.

N	Number of users in the system
O	Object size (bytes)
F	Block size (bytes)
S	Number of shards
D	Data retention period (2 years)
λ_a	Object arrival rate (objects/second/user)
λ_o	User opt-out rate (users/second)
C_s	Effective storage cost ($/GB/month)
C_e	Effective erasure cost ($/GB)
c	Compression ratio: 1.0 when no compression is possible 0.5 for log storage 0.25 for column storage
$\mathbf{1}_{\text{err}}$	1 if using the systematic erasure strategy
$\mathbf{1}_{\overline{\text{err}}}$	1 if using the encryption strategy

Table I
Summary of main system parameters.

A. Opt-out strategies

The two strategies we consider for opting-out users' data are:

Systematic:
On each opt-out request, all blocks corresponding to the shard attributed to the requesting user are fetched from the storage facility to the computing instance. Blocks are scanned for objects of that user; objects are removed from those blocks. Resulting blocks are sent back to the storage facility.

Encryption:
Each user in the system is associated with a symmetric cryptographic key, generated at user's subscription. Each incoming object is encrypted with the key of its proprietary user. On an opt-out request, the encryption key from the user is deleted by secure overwriting, making users' objects unreadable. Note that objects thus remain in the block store until the end of the data retention period has been reached.

For the systematic strategy, we also consider a variant, where we jointly use an *indexing structure* (that maps users to blocks where they have data into). This variant avoids fetching all blocks, only those concerned by objects from a given user. More elaborated strategies, for instance using a combination of those two strategies, or garbage collecting the unreadable blocks in case of encryption are left for future works.

B. Cost of storing data

We consider that users interactions with the system (here objects sent by a user device) arrive according to a Poisson process with rate λ_a, which constitutes a classic assumption for providing general and initial results in such a study [17]. For clarity we consider the opt-out of a user having a single device; results can be easily extended to multiple devices.

Objects remain in the system for a duration of D years (service retention period), if there is no opt-out action by the

user. We consider the stationary state where a user has been present for at least the retention period. In such a state, objects are erased at the same rate as they arrive, so the general amount of stored data remains constant. Ignoring opt-out, a single user stores on average what it generates over D years, that is after compression:

$$d_u = \lambda_a c D O, \tag{1}$$

with c being the compression ratio enforced by the service before actually storing data (see Table 1), and O being the size of objects sent by users. We assume that the system operates with a constant number of users N, meaning that new users join the system at the same rate as users opt-out (at rate λ_o). On an opt-out request by a user, an amount d_u of data is thus erased from the system, and gets replaced by data from a new user. Total amount stored d_t remains unchanged: storage cost does not change due to opt-outs, but only deletion costs are to be taken into account.

$$d_t = N d_u \tag{2}$$

We assume that opt-outs follow an exponential distribution with parameter λ_o. The amount of data to be deleted due to opt-out queries for all users is thus (per second):

$$N \lambda_o d_u. \tag{3}$$

When using the encryption strategy, as data is not deleted immediately, the data of the opted-out user must be stored during the time between the opt-out request and the end of retention period D. Given that opt-outs follow an exponential distribution, opt-out times are distributed uniformly over time. In such a setup, and with C_s the effective platform storage cost (in \$/GB/month), the cost associated with the data stored after an opt-out request Y is:

$$Y = \int_0^D C_s(d_u - cO\lambda_a t)\ dt = cC_s(\frac{O\lambda_a D^2}{2}). \tag{4}$$

Hence, the total additional storage cost associated with opt-outs in case of non-immediate deletion is the product of the opt-out rate for all users and the cost for an opt-out request:

$$N\lambda_o Y = N\lambda_o c C_s(\frac{O\lambda_a D^2}{2}). \tag{5}$$

The total storage cost for the system is thus the sum of all data stored and data not immediately erased (if any) times the storage cost per GB:

$$C_s(d_t + \mathbf{1}_{\overline{\text{err}}} N\lambda_o Y) \tag{6}$$

which simplifies to

$$C_s c N \lambda_a D O (1 + \mathbf{1}_{\overline{\text{err}}} \lambda_o \frac{D}{2}). \tag{7}$$

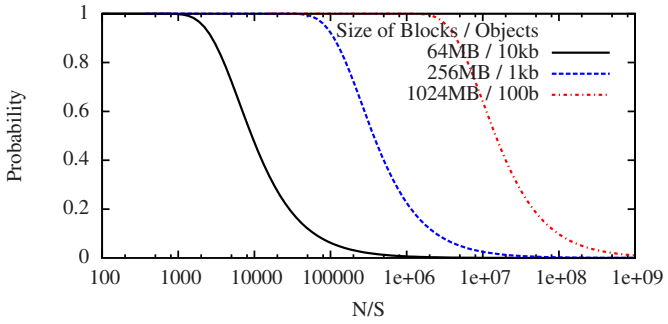


Figure 1. Probability that a block must be processed for erasing at least one user's object.

C. Cost of opting-out a user

We now study the cost related computation for data deletion. First, we stress that erasing data after the expiration of the retention period is free, as objects from different users within a block have the same expiration date because they have arrived on a very short time span (*i.e.*, in the time period needed to form a block for that shard), which allows to discard the complete block without having to read it.

From our model, arrivals of objects follow a Poisson process and are thus independent. Each object concatenated in a block thus belongs to a particular user with probability $p_u = 1/n$, where n is the number of users in the shard associated with the one of the given user.

Consequently, the probability that a block contains at least one object from a particular user (meaning that it must be processed in case of opt-out for that user) is

$$p_r = 1 - (1 - p_u)^o, \tag{8}$$

where o is the total number of objects concatenated in the block (*i.e.*, $o = F/O$).

Figure 1 plots this function when varying the number of users per shard (S shards in total), and object/block sizes. It clearly shows that for most settings, a large number (if not all) of blocks have to be processed upon an opt-out request, when using the systematic erasure strategy. However, whenever there are many users per shard (and the number of concatenated objects per file remains low in proportion), the probability that a block is to be processed tends to zero. In such a case, an indexing strategy that keeps track of what block is composed by at least one object of a given user can be leveraged so as to selectively process only the necessary blocks.

The amount of data processed, given that we do not use an indexing strategy is $1/S$ of the total amount of data stored d_t. Note that this is unrelated (and considerably higher than the actual amount of data to be erased d_u). This is due to the fact that in practice, objects from different users are concatenated into big blocks to limit the number of log files in the system, so that blocks have to be scanned for retrieval of particular objects.

Instance type	AWS cost	Cores	S3 DL	S3 UP	Compression	Encryption	Eff. speed	Eff. cost (C_e^{-1})
m4.large	0.15$/hour	2	60 MB/s	60 MB/s	28 MB/s/core 56 Mb/s total	435 MB/s/core 870 MB/s total	56 MB/s	1344 GB/$
m4.xlarge	0.3$/hour	4	60 MB/s	100/201/303	28 MB/s/core 112 Mb/s total	435 MB/s/core 3480 MB/s total	60 MB/s	720 GB/$
m4.10xlarge	3$/hour	40	550 MB/s	550 MB/s	28 MB/s/core 1120 Mb/s total	435 MB/s/core 22 GB/s total	560 MB/s	660 GB/$

Table II
PROCESSING SPEED PER INSTANCE TYPE. COMPRESSION AND ENCRYPTION SPEEDS ARE GIVEN PER CORE. ACCOUNTING FOR POTENTIAL PARALLELISM IN THE PROCESSING, THE BOTTLENECK IS THE TRANSFER OF DATA FROM S3 TO THE EC2 INSTANCE, WITH INSTANCE TYPE M4.LARGE HAVING THE BEST EFFECTIVE COST.

Each object erasure (assuming they are processed independently) costs C_e per GB. Total cost, including the fact that data is compressed[1] is thus:

$$\mathbf{1}_{\text{err}} p_r N \lambda_o C_e \frac{cN\lambda_a DO}{S}. \tag{9}$$

To model different strategies, in the following, we will use either $p_r = 1 - (1 - p_u)^o$ for strategies that keep an index describing which users have data in each block, or $p_r = 1$ when no such index is kept. Overall, the index is useful when there are few objects per block and many users per shard.

D. Total cost of an opt-out enabled system

From the two previous subsections, we finally express the total expected operational cost for a block storage system allowing user opt-outs, including storage-related costs (left-hand side) and processing-related costs (right-hand side) as:

$$C_s d_c (1 + \mathbf{1}_{\overline{\text{err}}} \lambda_o \frac{D}{2}) + \mathbf{1}_{\text{err}} p_r \lambda_o C_e \frac{N d_c}{S}, \tag{10}$$

with $d_c = cN\lambda_a DO$ the total amount of data (compressed) that is stored.

Central to our analysis is thus formula (10). On a service operator perspective, the only two parameters of that formula that could be adjusted easily for reducing global costs are the sharding policy S and the probability to have to process a block p_r (modified by setting O and F). Next section precisely highlights the impact of the formula's parameters.

IV. EVALUATION

We now apply this cost model, formula (10), to different settings, with related parameters being based on benchmarks and service costs of Amazon Web Services. We stress that we use these settings as examples to derive recommendations in a well-know environment. It is simple to apply this model to other settings, in order to compare the costs of alternative strategies for user opt-out.

A. Preliminary Amazon benchmark for setup

In Amazon Web Services, resources can be elastically adapted to the current service load (*e.g.*, amount of computation to perform at a given point in time, or amount of data to store). Whenever a block processing operation needs to be performed, an EC2 server must be used (and paid for on a hourly rate), while storage is billed per GB/month. Network traffic between EC2 and S3 is not billed as long as the two are in the same zone, thus we do not account for the traffic in this analysis (but can easily be added in the model).

We benchmarked the Amazon EC2/S3 platforms with regards to key metrics for this study, namely encryption, compression, upload and download speed, as seen on Table II. Benchmark was conduced in the Amazon us-east-1 facility with an Amazon Linux AMI, using AWS tools for uploading to and downloading from S3, gzip for compression, openssl with AES-NI for encryption. We considered three standard instance types from m4.large (advertised as having moderate network performance) to the powerful m4.10xlarge one (which has a 10Gb dedicated network card), which capabilities are also presented on Table II.

In order to leverage full capacity for uploads and downloads, we ran 4 AWS commands in parallel for m4.large and m4.xlarge and 8 commands in parallel for m4.10xlarge. Average performance is reported. The block sizes (64MB to 1GB) has a limited impact on performance. Regarding computational tasks, we benchmarked gzip (compression level 0) on random data and openssl (AES-256-CBC with AES-NI enabled). As all instances have similar CPUs, they exhibit similar performance per core. The overall capacity can be obtained by multiplying the per-core performance by the number of cores. We assume that the processing is done in a pipeline so that the overall speed of the pipeline is the speed of the slowest instance component. As shown on Table II, the smallest instance offers the best tradeoff between price and performance for this I/O intensive task, and is able to process more than 1TB for 1$. Costs computed in the remaining of this paper thus leverage this instance's cost/performance ratio, setting $C_e = \frac{1}{1344}$\$/GB. Storage cost is set to current AWS price: $C_s = 0.03$\$/GB/month.

B. Application scenarios

We investigate the opt-out problem over four scenarios, reflecting various applications and platform configurations, as presented in Table III.

Scenario $\mathcal{A}$ mimics the use case of connected devices sending small amounts of data, but very frequently to the cloud (*e.g.*, fitness or gps watches). Scenario $\mathcal{C}$ see packets be send at a regular and less frequent pace, as for *e.g.*, home gateways monitoring [18], where 40KB packets containing

[1]We include compression in the expression of cost as the limiting element in the opt-out process is the I/O (network) performance as we shall see in Section IV.

many status counters are sent by each gateway to the collect point every minute. Scenario $\mathcal{B}$ sits in the middle of $\mathcal{A}$ and $\mathcal{C}$. Finally, scenario $\mathcal{D}$ models a surveillance application, where user's home pictures are send on a hourly basis, for automated analysis and anomaly detection; data received (*e.g.*, JPEG images) have the specificity of not being compressible (we apply a compression ratio of 1 for all setups). Retention duration ranges from 1 month to 2 years, following an inversely proportional attribution to the alleged data sensitivity. We set the opt-out rate to $1/5$ opt-out request/year/user for scenarios $\mathcal{A}$ and $\mathcal{B}$, corresponding to the observed 20% churn at a telco operator in 2004 [19], and more aggressively to $1/2$ request/year/user for $\mathcal{C}$ and $\mathcal{D}$.

We apply these scenarios in following three storage setups:

Raw:
: The data is stored in a raw format, concatenating objects into blocks stored on S3. Given the structure of the data, a compression ratio of 0.5 is applied [13].

Column:
: The data is stored in column form, structuring objects received as in Dremel or Parquet. The impact on our model is that structured data tends to be more compressible; a compression ratio of 0.25 is applied [7], [8].

Encrypted:
: The data is stored uncompressed as applying ciphering prevents to leverage correlation between users for compression. The compression ratio applied is thus 1.0. The data is deleted only after the expiration of the data retention period.

Note that due to the compression gap, Raw storage is thus expected to be more expensive than Column storage in any situation.

C. Cost evaluation results

We plot the total operational cost (the sum of storage costs and processing costs) in dollars ($) per month, for the four scenarios listed in Table III. We vary core parameters to highlight their relative influence (the others being fixed).

Plots present cost curves for *Raw* and *Column* environments, where data compression applies and is leveraged by the systematic erasure strategy. In all scenarios, we plot results when using an indexing structure, and without using it (*i.e.*, $p_r = 1$). Curves from the Raw environment without indexing are not plotted for the sake of readability, as always more costly than in the Column environment. The *Encrypted* environment (plain black curve) is also presented, where the encryption strategy applies.

Impact of sharding. The main observation from Figure 2, where the number of users per shard varies, is that the cost of the systematic strategy depends highly on this sharding. If N/S tends to be small, and unless the opt-out rate is very high (*i.e.*, users leave on average after 3 months), then the cost of systematic deletion remains low when compared to the cost of using encryption. Encryption strategy is unaffected by N/S, as opted-out data remains in the system. Another interesting observation is that the influence of the number of users/shards on the opt-out cost. In scenarios $\mathcal{C}$ and $\mathcal{D}$, in which files contain a (comparatively) small number of objects, the use of an index for tracking blocks is preventing a large portion of the blocks to be processed. This stabilizes the operational cost, whereas the absence of an index makes those costs explode. Scenario $\mathcal{A}$ and $\mathcal{B}$ do not benefit from this effect since objects are small and files are large meaning that many user objects are interleaved in a given block. Key learning is that for large enough objects collected, a differentiator for the best strategy to employ depends on whether or not objects can be compressed: in scenario $\mathcal{C}$, the column-indexed strategy is a win, whereas in scenario $\mathcal{D}$ the encryption strategy does not suffer a cost penalty from its impossibility to compress objects and thus slightly prevails.

Impact of data arrival rate. The data arrival rate significantly impacts the cost of both strategies. It only favors encryption for quick arrivals and small objects in scenarios $\mathcal{A}$ and $\mathcal{B}$, whereas $\mathcal{C}$ favors the column-indexed strategy, as seen on Figure 3.

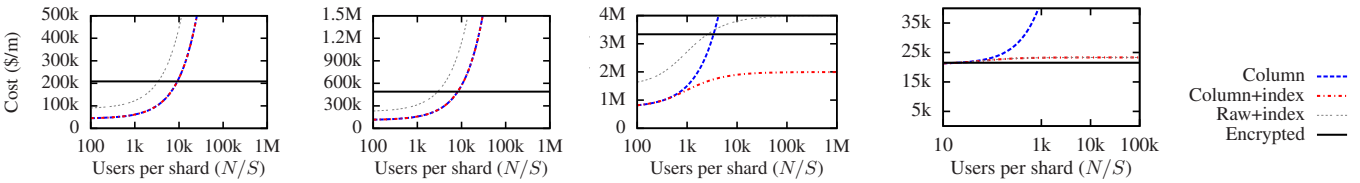
Impact of opt-out rate. The opt-out rate has a clear influence on systematic erasure, while it has a negligible one on the encryption strategy, as seen on Figure 4. Quick opt-outs make systematic erasure inapplicable to scenarios $\mathcal{A}$ and $\mathcal{B}$, while the index variant should be preferred over encryption for the scenario $\mathcal{C}$ and reasonable opt-out rates.

Impact of retention period. As expected, increasing the data retention duration (Figure 5) causes the encryption strategy to become more inefficient as object size increases (scenario $\mathcal{D}$): opted-out objects remain in the system in an unreadable form, which has a clear storage cost on the long run. Scenarios with smaller objects still favor encryption.

Impact of object size. As seen on Figure 6, encryption is to be favored in scenarios $\mathcal{A}$ and $\mathcal{B}$, while the opposite is true for scenarios $\mathcal{C}$: larger objects are less costly to handle with systematic strategies. In $\mathcal{D}$, there is a tie for large objects that cannot be compressed.

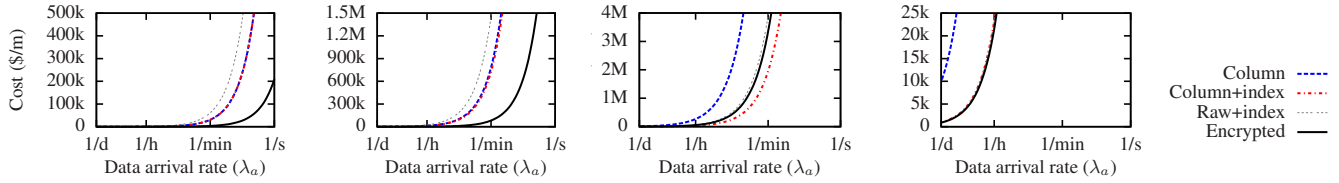
Impact of block size. Finally, as shown on Figure 7, varying the block size alone has a small effect on both strategies, except for the setting $\mathcal{C}$ and systematic erasure strategy. The encryption strategy is always unaffected, as blocks are never fetched for processing.

Evaluation summary. In conclusion of this evaluation, the encryption strategy is a win for scenarios $\mathcal{A}$ and $\mathcal{B}$, with small objects, quick data arrival and opt-out rates, except when the sharding ratio N/S is small. Scenario $\mathcal{C}$ cannot clearly discriminate one best strategy: systematic erasure is prevailing in the column-indexed variant, when objects are larger, and the number of devices per shard is high. In most cases, the bare systematic erasure without block indexing should not be considered, as always superseded by the indexed variant. Finally, dealing with objects that cannot be further compressed at the block storage layer (scenario $\mathcal{D}$) removes the penalty for the encryption scenario, and then makes it a win in nearly all applications.



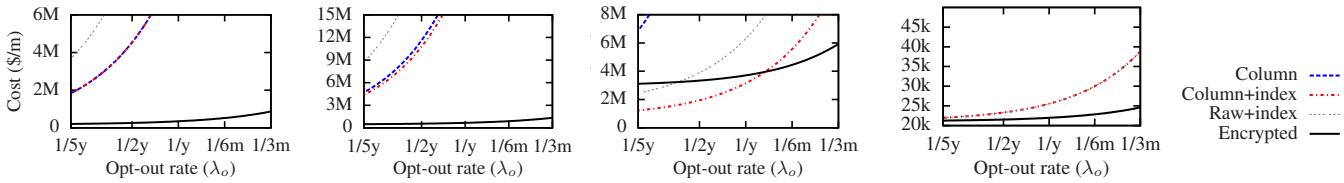
(a) Scenario $\mathcal{A}$ (b) Scenario $\mathcal{B}$ (c) Scenario $\mathcal{C}$ (d) Scenario $\mathcal{D}$

Figure 2. Influence of number of users/shard (N/S). All other parameters are fixed according to Table III.



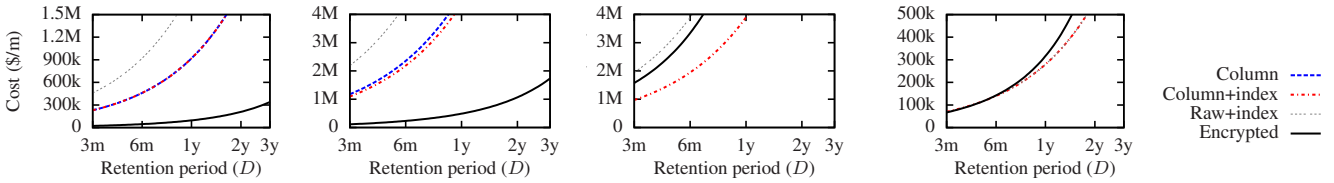
(a) Scenario $\mathcal{A}$ (b) Scenario $\mathcal{B}$ (c) Scenario $\mathcal{C}$ (d) Scenario $\mathcal{D}$

Figure 3. Influence of data arrival rate (λ_a). All other parameters are fixed according to Table III.



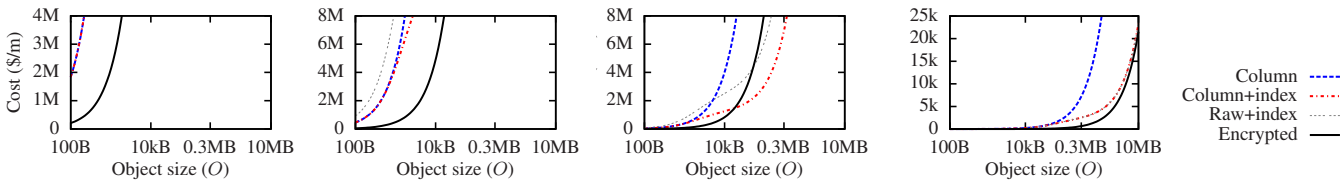
(a) Scenario $\mathcal{A}$ (b) Scenario $\mathcal{B}$ (c) Scenario $\mathcal{C}$ (d) Scenario $\mathcal{D}$

Figure 4. Influence of user opt-out rate (λ_o). All other parameters are fixed according to Table III.



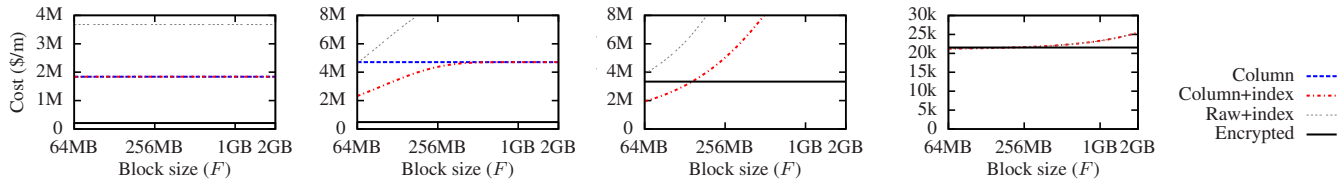
(a) Scenario $\mathcal{A}$ (b) Scenario $\mathcal{B}$ (c) Scenario $\mathcal{C}$ (d) Scenario $\mathcal{D}$

Figure 5. Influence of retention period (D). All other parameters are fixed according to Table III.



(a) Scenario $\mathcal{A}$ (b) Scenario $\mathcal{B}$ (c) Scenario $\mathcal{C}$ (d) Scenario $\mathcal{D}$

Figure 6. Influence of object size (O). All other parameters are fixed according to Table III.



(a) Scenario $\mathcal{A}$ (b) Scenario $\mathcal{B}$ (c) Scenario $\mathcal{C}$ (d) Scenario $\mathcal{D}$

Figure 7. Influence of block size (F). All other parameters are fixed according to Table III.

Scenario	# of users N	# of shards S	Data arrival rate λ_a	Opt-out rate λ_o	Retention D	Object size O	Block size F
$\mathcal{A}$	1M	10	1/1 obj/sec/user	1/5 req/user/year	2y	100B	1024MB
$\mathcal{B}$	5M	50	1/10 o/s/u	1/5 r/y/u	1y	1KB	256MB
$\mathcal{C}$	10M	500	1/60 o/s/u	1/2 r/y/u	6m	40KB	64MB
$\mathcal{D}$	100K	10	1/3600 o/s/u	1/2 r/y/u	1m	10MB	1024MB

Table III
FOUR APPLICATION SCENARIOS FOR COST EVALUATION OF USER OPT-OUT STRATEGIES.

Lessons from this modeling thus indicates that encryption is not always the best option, and that the most naive strategy of systematic deletion on request is competitive in certain scenarios, due to its capacity to operate along data compression where it is not the case using prior encryption. Furthermore, and regardless of the nature of objects to store, the two practical means of action for a service operator are the sharding policy and probability to have to access a block for deletion; if one can have few users per shard, then systematic strategies are prevailing (or are equivalent in scenario $\mathcal{D}$). Regarding the second actionable, having large blocks cause p_r to be close to 1 and thus makes systematic strategies too costly to be applied (*i.e.*, encryption must be considered).

V. RELATED WORK

The possibility for a user to opt-out from cloud services has been identified as crucial since the popularization of cloud-based applications [20], [21]. While the protection of data stored in the cloud has been widely addressed (see e.g., [22]), the consequences of actual data deletion upon user decision has not been addressed in the context of block stores.

In storage systems for big data computing, block stores remain a popular alternative, as they allow to rationalize the cost of metadata and optimize disk access (see *e.g.*, [2]–[5], [12], [23] for platforms, or [15], [24] for massive services using them). Storage in NoSQL databases like e.g., Cassandra, would allow direct access to objects as opposed to the need to fetch full blocks. Yet, the storage cost incurred by such databases as compared to the storage of raw data is significant (e.g., it is of on average 2 times in Cassandra [12]); this leads some storage-intensive applications to still favor block stores.

In block stores, data deletion is generally implemented using a tombstone flag, that is a single indicator that a block is logically deleted [15], [23], [25]. Data thus remains in the system until garbage collection mechanisms are triggered, which can take several days at best (e.g., default is 10 days in Cassandra [25]). This duration may not be acceptable regarding SLAs; we precisely question the cost tradeoff associated with immediate deletion in this paper.

VI. CONCLUSION

We introduced and addressed the problem of efficiently opting-out user data from block stores. By using two simple but conflicting opt-out strategies, we have shown that a specific application scenario should lead the choice of one over the other: systematic erasure trades storage cost for computing costs, and is efficient for applications with large objects and fine grain sharding. On the contrary, encryption trades processing for storage; it performs better when small objects arrive fast to the datacenter.

In general, the two practical actionables to lower total operational cost are to use a finer sharding or play on the ratio size of blocks over size of objects, in order to lower the probability to have a block to process in case of an opt-out request. One important leaning is also that the nature of the objects to store is also core for general performance: objects that cannot be compressed will favor the encryption strategy.

Beside those general guidelines, we believe it remains significant room for improvement using advanced strategies, as for instance *(i)* encryption and periodical garbage collection of unreadable objects, or *(ii)* voluntarily delaying the opt-out requests up to an acceptable limit, in order to operate object deletion in a batch mode. We hope that this study will motivate the design of those more complex and tailored opt-out strategies.

REFERENCES

[1] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2015.

[2] "Apache hadoop distributed file system," http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *SOSP*, 2003.

[4] "Amazon s3," https://aws.amazon.com/s3/.

[5] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, "Windows azure storage: A highly available cloud storage service with strong consistency," in *SOSP*, 2011.

[6] "Data protection day 2015: Concluding the eu data protection reform essential for the digital single market," http://europa.eu/rapid/press-release_MEMO-15-3802_fr.htm.

[7] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," *VLDB*, vol. 3, no. 1-2, pp. 330–339, Sep. 2010.

[8] "Apache parquet," https://parquet.apache.org/.

[9] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *NSDI*, 2005.

[10] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy, "The unified logging infrastructure for data analytics at twitter," *VLDB*, vol. 5, no. 12, pp. 1771–1780, 2012.

[11] "Linkedin camus," http://docs.confluent.io/1.0/camus/docs/intro.html.

[12] "Apache cassandra 2.0," http://docs.datastax.com/en/cassandra/2.0/cassandra/architecture/architecturePlanningUserData_t.html.

[13] "Avro performance, by grisha trubetskoy," http://grisha.org/blog/2013/06/11/avro-performance/.

[14] "Apache avro," http://avro.apache.org/.

[15] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in haystack: Facebook's photo storage," in *OSDI*, 2010.

[16] S. M. Diesburg and A.-I. A. Wang, "A survey of confidential data storage and deletion methods," *ACM Comput. Surv.*, vol. 43, no. 1, pp. 2:1–2:37, Dec. 2010.

[17] L. Gu, D. Zeng, P. Li, and S. Guo, "Cost minimization for big data processing in geo-distributed data centers," *Emerging Topics in Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 314–323, Sept 2014.

[18] I. Pefkianakis, H. Lundgren, A. Soule, J. Chandrashekar, P. Le Guyadec, C. Diot, M. May, K. Van Doorselaer, and K. Van Oost, "Characterizing home wireless performance: The gateway view," in *INFOCOM*, 2015.

[19] J.-H. Ahn, S.-P. Han, and Y.-S. Lee, "Customer churn analysis: Churn determinants and mediation effects of partial defection in the korean mobile telecommunications service industry," *Telecommunications Policy*, vol. 30, no. 10-11, pp. 552 – 568, 2006.

[20] I. Pollach, "What's wrong with online privacy policies?" *Commun. ACM*, vol. 50, no. 9, pp. 103–108, Sep. 2007.

[21] S. Pearson, "Taking account of privacy when designing cloud computing services," in *Workshop on Software Engineering Challenges of Cloud Computing*, 2009.

[22] M. Mowbray and S. Pearson, "A client-based privacy manager for cloud computing," in *COMSWARE*, 2009.

[23] "Openstack swift | enterprise storage from swiftstack," https://swiftstack.com/openstack-swift.

[24] G. Mishne, J. Dalton, Z. Li, A. Sharma, and J. Lin, "Fast data in the era of big data: Twitter's real-time related query suggestion architecture," in *SIGMOD*, 2013.

[25] "Apache cassandra 2.0, distributeddeletes," https://wiki.apache.org/cassandra/DistributedDeletes.