Harnessing Data Loss with Forgetful Data Structures

Azza Abouzied, Jay Chen
New York University Abu Dhabi
azza@nyu.edu, jchen@cs.nyu.edu

Abstract

Forgetting, losing, or corrupting data is almost universally considered harmful in computer science and blasphemous in database and file systems. Typically, loss of data is a consequence of unmanageable or unexpected lower layer deficiencies that the user process must be protected from through multiple layers of storage abstractions and redundancies. We argue that forgetfulness can be a resource for system design and that, like durability, security or integrity, can be used to achieve uncommon, but potentially important goals such as privacy, plausible deniability, and the right to be forgotten. We define the key properties of forgetfulness and draw inspiration from human memory. We develop a data structure, the *forgit*, that can be used to store images, audio files, videos or numerical data and eventually forget. Forgits are a natural data store for a multitude of today's cloud-based applications and we discuss their use, effectiveness, and limitations in this paper.

Categories and Subject Descriptors E.1 [*Data*]: Data Structures

Keywords memory loss; forgetful data structures; ephemeral storage

1. Introduction

The notion of loss frequently appears as a problem to be solved for reliable storage and communication. While improvements in hardware, protocols, and algorithms have over the past several decades made datacenter losses rare and generally more tolerable events, we argue in this paper that there are potential benefits to loosening our intransigent refusal to accept loss. The notion of embracing loss is actually less contentious than it first appears; loss is already widely used to positive effect in the form of soft-state, caching, and lossy compression. The key observation in all these cases being that if loss has predictable properties, then it may be harnessed for specific purposes.

We define forgetfulness as having four key properties: passive loss, graceful degradation, no hi-low storage and negligible preprocessing overhead (Section 1.1). We begin by describing two motivating applications for a forgetful data store with these properties.

Ephemeral Media Storage & Sharing. Popular ephemeral content-sharing apps like Snapchat [26], Wickr [31] and Silent Circle [25], are designed to protect privacy by deleting content after a finite time period. They allow users to communicate more freely with the knowledge that messages will eventually be forgotten [24]. Ephemeral apps challenge the assumption that data persistence is the right default and, by doing so, address (i) a market need for private and transient communications, (ii) a societal need to forget and move on [16] as codified in laws such as the EU's 'right to be forgotten' [15], and (iii) a corporate need for *plausible deniability* when subpoenaed for past records or subjected to government surveillance.

Compressed Archival Records. Our thirst for recording the minutiae of everyday life is leading to an enormous data glut that is overwhelming our datacenters and cloud stores. While cheap, storage is not free and the cost of storing data is quickly surpassing the economic benefit of archiving. A stop-gap measure is to compress such archival data. Once storage resources are exhausted, data managers then run an active delete thread with the worry of potentially irreversibly deleting important data. Figure 1(a) shows how we typically order our data for deletion. With forgetful data stores more recent data is preserved at full fidelity and older data at lower fidelity. Data such as time-series sensor readings, stock price trends, etc. allow transform-based compression techniques to re-encode data into most significant to least significant components allowing forgetful data stores to gradually forget the data by dropping least significant components first.

Forgetful data stores can provide a stronger sense of privacy for users who worry about their data being preserved indefinitely without their consent [29]. Redesigning the store so that it passively loses data allows users to be free from depending on an active deletion effort. Moreover, gradually losing data allows users to observe and accept impending

SoCC '15, August 27 - 29, 2015, Kohala Coast, HI, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3651-2/15/08 . . . $15.00.
DOI: http://dx.doi.org/10.1145/2806777.2806936

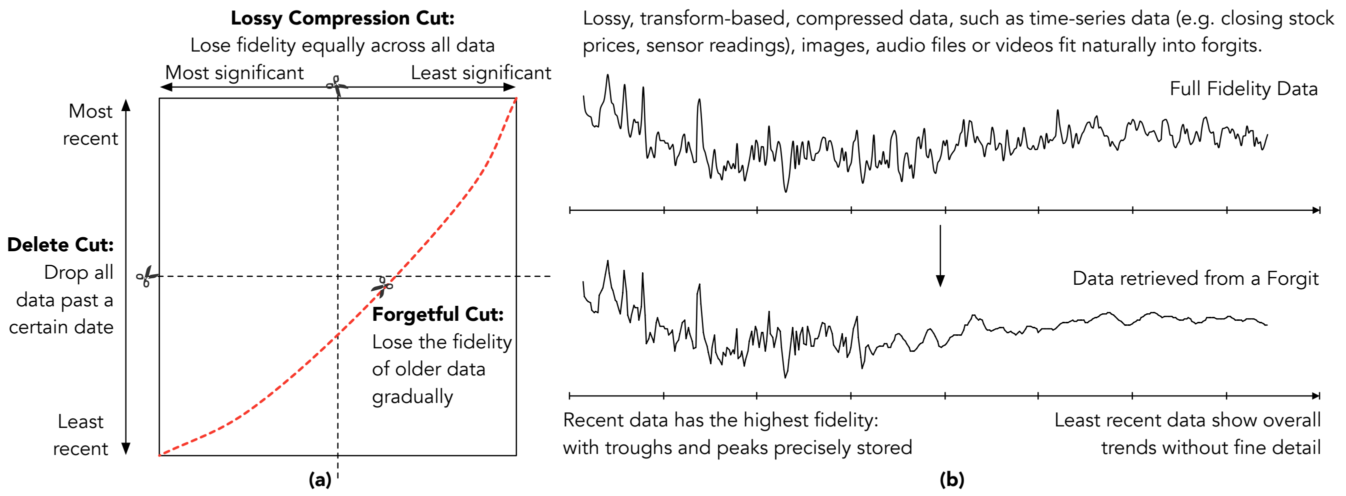
Lossy Compression Cut:
Lose fidelity equally across all data
Most significant
Least significant
Most recent
Delete Cut:
Drop all data past a certain date
Forgetful Cut:
Lose the fidelity of older data gradually
Least recent
(a)
Lossy, transform-based, compressed data, such as time-series data (e.g. closing stock prices, sensor readings), images, audio files or videos fit naturally into forgits.
Full Fidelity Data
Data retrieved from a Forgit
Recent data has the highest fidelity: with troughs and peaks precisely stored
Least recent data show overall trends without fine detail
(b)

Figure 1. (a) Forgetful model of loss compared with lossy compression and deletion/eviction. (b) Illustration showing how forgetting archival data gradually still allows limited computations to be performed even with partially forgotten data. Different features (e.g. precision, trends, etc.) may be preserved or discarded depending on application requirements.

losses compared to abrupt deletions. A system that gradually blurs past data is more compatible with human intuition of forgetting older memories until they are completely gone.

1.1 Defining Forgetfulness

Forgetting is the *passive and gradual* process of losing data fidelity until the data eventually disappears. In conventional storage, data either exists in a memory location or does not exist. This data can be explicitly compressed with a lossy data compression algorithm resulting in some loss of fidelity. Data can also be stored with varying levels of fidelity: for example geo-spatial images are stored in varying resolutions and sizes to support smooth zooming; document thumbnails are low fidelity images of documents that allow users to visually identify documents during searches. In both of these cases, the active processing that results in loss of fidelity is *not*, by our definition, forgetting. Forgetting is when data is initially stored in its highest precision representation and gradually, as new data is pushed into the store, the older data loses its precision until it is completely overwritten.[1]

The defining properties of a forgetful data store are:

1. *Passive fidelity loss*: A forgetful data store does not actively scan its data and selectively drop fragments. Instead, loss occurs naturally as a side-effect of pushing more data into a finite data store.
2. *Graceful degradation*: Data stored in forgetful data stores exists in one of F possible states of varying degrees of fidelity. Thus, as the store starts to forget data, it incrementally loses fragments of the data starting with least significant fragments to most significant fragment until the data is completely lost.
3. *No hi-lo storage*: A forgetful data store does not inflate the stored representation of the data to support its subsequent degradation. An example of hi-lo storage is if a single image of 1 MB is stored as multiple images that collectively occupy 2 MB, one high-fidelity image at 1MB, a half fidelity image at 0.5 MB, a quarter fidelity image at 0.25M and so on; then, the higher fidelity images are dropped as the store runs low on space.[2]
4. *Negligible pre-processing overhead*: The overhead of pushing data into the store should be negligible compared to storing data in a conventional store. Thus, a forgetful data store should store data in its native or near-native representation.

Forgetfulness is defined this way to reflect our natural intuition about human memory. This makes using forgetful storage easier to reason about. In situations where physical storage is large relative to the amount of stored data, the model is identical to the conventional one i.e. all data is stored and may be retrieved perfectly. As the amount of stored data begins to exceed the available storage space, fragments of the oldest data are lost. We introduce a forgetful data structure, the *forgit*, which supports append and retrieve operations while satisfying the above four properties. We implement two proof-of-concept forgits (numerical, image) and discuss limitations with respect to text forgits.

2. The Forgits

The forgit is a simple data structure that supports infinite appends and retrievals with the provision that it will forget older data. It consists of F fidelity levels of circular buffers

[1] Tangential to the concept of forgetting is data corruption. Several mechanisms such as cyclic redundancy checks, parity bits, error correction codes, detect and repair data corruption.

[2] This scheme is reminiscent of hi-lo pricing strategies where stores inflate their prices only to bring them down during discounts giving customers the illusion of a bargain.

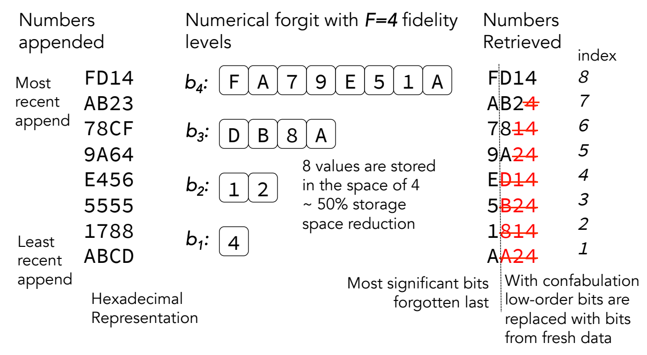
Numbers appended
Numerical forgit with F=4 fidelity levels
Numbers Retrieved
index
Most recent append
FD14
AB23
78CF
9A64
E456
5555
1788
ABCD
Least recent append
Hexadecimal Representation
b4: F A 7 9 E 5 1 A
b3: D B 8 A
b2: 1 2
b1: 4
8 values are stored in the space of 4 ~ 50% storage space reduction
Most significant bits forgotten last
FD14 8
AB2~~4~~ 7
78~~14~~ 6
9A~~24~~ 5
E~~D14~~ 4
5~~B24~~ 3
1~~814~~ 2
A~~A24~~ 1
With confabulation low-order bits are replaced with bits from fresh data

Figure 2. Illustration of the behavior of a numerical forgit after a series of appends and retrieves. Hexadecimal instead of binary representation is used for clarity.

Algorithm 1 Append

```
1: procedure APPEND($O$)
2:     $N$ is the number of objects in the forgit
3:     $N \leftarrow N + 1$
4:     $\{s_1, ..., s_F\} \leftarrow$ Segment($O$ , $F$)
5:     for $i$ from 1 to $F$ do
6:         $b_i(N \mod \text{size}(b_i)) \leftarrow s_i$
7:     return $N$
```

$\{b_1, b_2, ...b_F\}$ such that $1 \leq \text{size}(b_{i-1}) \leq \text{size}(b_i)$ as illustrated in Figure 2.

The forgit supports data types with the following property: objects of the data type should be decomposable into F segments, $\{s_1, s_2, ..., s_F\}$ such that segment s_i is appended to buffer b_i and segment s_{i-1} is less significant than segment s_i. Each object appended to a forgit returns an index c to later retrieve the object. This index is simply a counter. The append procedure is described in algorithm 1.

To retrieve an object with index c, we use the retrieve procedure described in algorithm 2. Forgits can mimic a human memory quirk: *confabulation*. With confabulation enabled, lower-order segments of a partially forgotten object are replaced by lower-order segments of more recent data. When confabulation is disabled, lower-order segments are zeroed out during retrieval. Thus, the forgit has the following tunable parameters: (i) maximum size of elements fully or partially remembered by a forgit by setting the size of buffer of b_F (i) degree of degradation by setting the number of fidelity levels F and the size of each buffer, (iii) and whether the structure supports confabulation.

Confabulation could be useful in at least two ways. First, in the case of aggregate data, zeroing out lower-order segments of data may actually be less 'natural' than sampling lower-order segments from other data, especially if the data is drawn from the same overall distribution. So rather than flattening out the high-frequency data, confabulation gives an impression of 'realistic' data drawn from sampling the low-order segments from the dataset. If, for example, a user wished to perform computation on the data, confabulation could give better aggregates and less overall bias. Second, confabulation could be used take advantage of certain patterns in the data or to provide interesting visual effects. For example, if a user has a set of images that are clustered in terms of similarity of the low-order bits (say, patterns or colors), and these clusters are stored in separate forgits the confabulation would fill in details that appear reasonable.

As with conventional storage, forgits may be allocated with access controls to preserve isolation of user data.

Algorithm 2 Retrieve

```
1: procedure RETRIEVE($c$ : index into forgit)
2:     $N \leftarrow$ number of objects pushed into forgit
3:     for $i$ from 1 to $F$ do
4:         if CanConfabulate $|| N - c \leq \text{size}(b_i)$ then
5:             $s_i \leftarrow b_i(\text{size}(b_i) \mod c)$
6:         else
7:             $s_i \leftarrow \emptyset$
8:     return Concatenate $(s_1, ..., s_F)$
```

Different data types have different segment and concatenate functions. We illustrate these functions with three data types: numerical, image, and text illustrating the strengths and limitations of forgits for each type.

2.1 Numerical Forgit

The natural binary representation of numbers preserves *significance ordering*, i.e. the bits are ordered left to right from most significant to least significant. This natural ordering is advantageous when using forgits as no pre-processing is required to determine which segments of a binary representation are most significant and hence should be forgotten last. The segment operation is straightforward: given F fidelity levels the binary representation of the number is split into F equal sized segments. For example the two byte number, 1111001110100101 can be split into four segments 1111, 0011, 1010, 0101 where the most significant segment 1111 is placed in buffer b_4 and the least significant segment is placed in b_1. Concatenating segments retrieved from the forgit is also straightforward. Figure 2 illustrates the operation of a forgit for two byte numbers. Floating point representations are also amenable to significance ordering by dropping lower significand bits.

2.2 Image Forgit

Extending the forgit to more complex data types such as images is also straightforward as long as the data type has a native representation that preserves significance ordering. Commonly used compressed image (and other media like audio and video) formats such as JPEG naturally preserve significance. To store an image in JPEG form, the image is first quantized through a *discrete cosine transform* (DCT). This converts the image from a pixel-by-pixel representation into a frequency domain matrix representation: the DCT

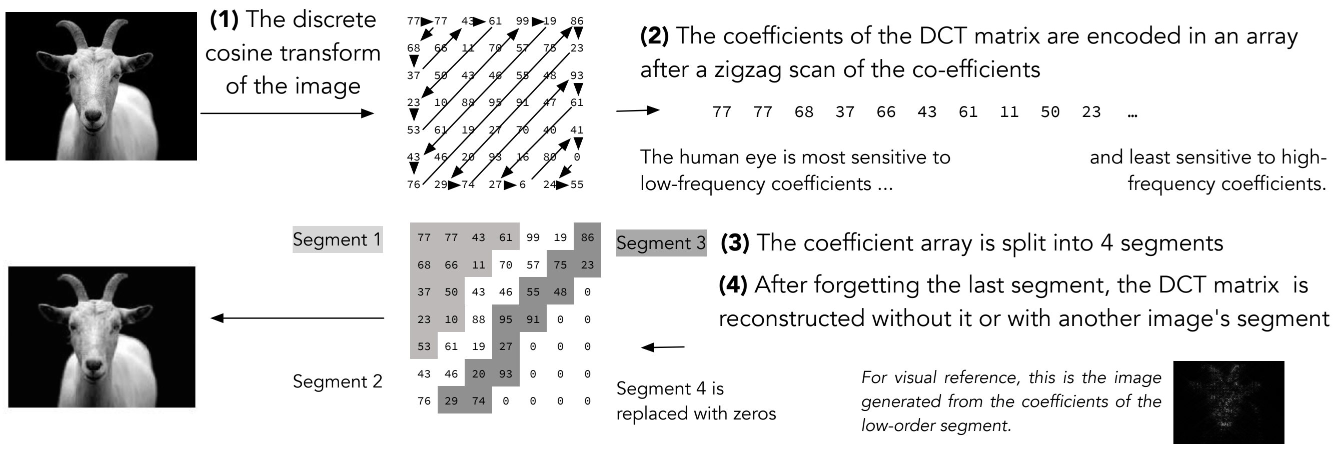
(1) The discrete cosine transform of the image
(2) The coefficients of the DCT matrix are encoded in an array after a zigzag scan of the co-efficients
77 77 68 37 66 43 61 11 50 23 …
The human eye is most sensitive to low-frequency coefficients …
and least sensitive to high-frequency coefficients.
Segment 1
Segment 2
Segment 3
(3) The coefficient array is split into 4 segments
(4) After forgetting the last segment, the DCT matrix is reconstructed without it or with another image's segment
Segment 4 is replaced with zeros
For visual reference, this is the image generated from the coefficients of the low-order segment.

Figure 3. Transforming an image into its JPEG representation with segmentation and forgetfulness illustrated.

coefficient matrix. Since the human eye cannot accurately distinguish the exact strength of high frequency variations, image compression is typical achieved by ignoring the high frequency components of the image (the bottom-right half of a DCT matrix). The natural significance ordering of coefficients in the DCT matrix is a zigzag through the matrix starting from the top-left coefficient and ending at the bottom-right coefficient. The matrix is typically stored as an array with coefficients ordered in the zigzag order.

The segment operation is to divide the array into F equal sized segments. The segments from the bottom-right half of the DCT matrix are least significant and hence are dropped first. At retrieval, the segments are concatenated back into one array before being streamed into a JPEG decoder. Figure 4 illustrates the quality of eight images pushed in a forgit with four fidelity levels.

Both numerical and media forgits[3] satisfy all properties of forgetfulness. The loss is passive: adding data to the structure will overwrite older data. The degradation is graceful: before the data is completely forgotten, its most significant components can still be retrieved. There is no hi-lo storage: the representation is not expanded before storage and no copies of the data are made. Finally the overhead of segmentation and concatenation is $O(1)$ and may be further accelerated with hardware assistance.

2.3 Text Forgit

A compelling application for forgetfulness is text communication whether it is in the form of emails, tweets or chat messages. Unfortunately data that has no natural significance ordering such as English sentences, may require both preprocessing and hi-lo storage to achieve passive and graceful loss.

[3] Note time-series forgits are similar to media forgits as time-series data can be compressed with a frequency-domain transform [20]. A choice of whether low or high frequency variations are more significant allows forgits to select which components of the time-series data to forget first.

We describe two techniques here that can be applied to text with the caveat that they do not satisfy the two forgetfulness properties: no hi-lo storage and negligible preprocessing overhead. Suppose the relative importance of each word or n-gram is stored in a frequency dictionary [14], then a sequence of words can be ordered by importance where uncommon words are the most important and common words are the least important and are dropped first from a message. Such a technique engages in hi-lo storage: for each word we must also store its position in the original text.

Alternatively, we can pre-process the text using auto-summarizers such as LexRank [7] or TextRank [17] that construct a graph of the original text where highly ranked sentences represent the most significant components of the text. The graph representation naturally fits into a forgit: nodes with high in-degrees are placed in the highest fidelity level buffer and hence dropped last. Unfortunately, such a technique requires significant text pre-processing.

2.4 Encrypted Forgit

Forgits do not natively provide any privacy guarantees. On public, untrusted cloud stores, it may be useful to encrypt as well as gradually forget data. Combining forgetfulness with encryption is straightforward, and can be done by encrypting data stored in a forgit at a per-segment level. If the encryption scheme results in variable ciphertext length, wasting space could be avoided by re-implementing forgits using pointers to segments rather than circular buffers. If stronger timeout guarantees are required on when data must be forgotten, encrypted segments could be combined with other privacy preserving schemes such as Vanish [9]. One drawback of using such a system in conjunction with forgits would cause expired data to occupy space though it is unrecoverable until it is overwritten. In these straightforward applications of security mechanisms the overall security properties are dictated by the mechanism applied to the forgits rather than any properties of the forgits themselves. We leave

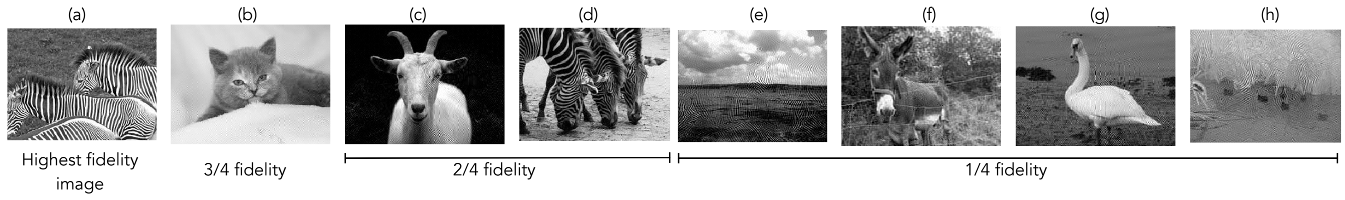

Figure 4. The loss of fidelity as images (h, g, f, e, d, c, b, a) are pushed into a forgit. The oldest four images (e, f, g, h) have lowest fidelity: $1/4^{th}$ the original image. Image confabulation is most prominent in images with low intensity, zebra patterns are clearly visible in images (e) & (f).

the devising of more sophisticated schemes that can produce stronger guarantees as an area of interest for future work.

3. Related Work

Data privacy and security are rising to the forefront of human concerns as the information age saturates our social, cultural, and geo-political domains. Cloud computing and ubiquitous computing are rapidly eroding any expectations of privacy even despite santization efforts through anonymization or obfuscation of data [29]. The growing popularity of ephemeral consumer applications such as Snapchat [26] and regulatory proposals such as the right to be forgotten [15] signal resistance to this trend.

3.1 Data Privacy

There are surprisingly few mechanisms designed to achieve the goal of forgetting data beyond active periodic deletion [13]. Research efforts around ephemeral data typically focus on guaranteeing the deletion or expiration of data [9, 19, 28]. Ephemerizer and similar systems [18, 19] rely on storing the encrypted data on a separate machine from the key used to decrypt the data. Later systems like Vanish [9] further extend this idea using distributed hash tables (DHTs) to store keys and protect against adversaries attempting to access data after timeout. Other recent attempts to preserve data privacy do so by securing mobile data [27] or protecting communication channels by erasing all traces of execution [6].

Unlike these works, we do not propose to offer guarantees on expiration or destruction of data. Instead, the data structure for storing data passively implements data loss through gradual use. We also do not consider adversarial threat models; if stronger guarantees are required, then active schemes using encryption may be implemented around the forgit storage primitive or at higher abstraction levels.

3.2 Data Loss

Within computer science, notions related to data loss frequently appear as obstacles to be overcome when computing limits are encountered (e.g. data loss due to lack of storage space where, as with privacy, the solution is active deletion [13]). However, losses do already occur predictably and ubiquitously in caching (eviction), (lossy) compression, and (soft) state. Here, we only describe closely related work to the techniques we employ, namely, compression and representation.

Compression exploits statistical redundancies of data across space, time, or frequency. Lossy compression schemes for audio, image, and video data achieve one to two orders of magnitude greater compression than lossless schemes by exploiting the fact that reconstructed data does not necessarily need to be identical to the original data. Instead, only humanly perceptible differences must be preserved, and thus, encodings that consider variations in human sensitivity to certain frequencies, luminance, and color may be used (MP3, JPEG, MPEG-4, etc. [3, 4, 21, 30]).

Our forgetful data structures are directly compatible with data compression schemes whose outputs can be decomposed into independent segments ranked by 'significance'. Fourier and related signal-frequency transforms have been widely used since the 1960s [5] and naturally produce this significance segmentation property after being applied to signals (time-series data). In the case of images, the discrete cosine transform (DCT) [2] used in JPEG image compression are ideally suited for segmentation (Section 2). The idea of storing images in limited space with lossy rate controlled compression has also been proposed previously [12].

Though forgits can represent certain data in compressed form, this is different from a general compression strategy where data is stored at only one pre-defined level of precision or fidelity. The benefit being that forgits allow for the passive reduction in precision through the loss of low order fragments.

3.3 Approximate Computation

Since the 1970's, many novel ideas along the lines of fuzzy, lossy, and approximate computation were proposed, but compelling applications have been limited to specific tasks in domains such as media processing, artificial intelligence, and data encoding, where tolerating minor errors can drastically improve performance [21, 30]. Approximate computation models are related to forgetful data structures in that both ideas seek to relax accuracy requirements, but forgetfulness is primarily concerned with storage rather than computation. Approximate computation using forgetful data structures may be potentially made faster than normal computation because there is less data to traverse.

In databases, online aggregation was proposed by Hellerstein et al. to give users real-time feedback and control over their aggregation queries [11] through the incremental computation of approximate aggregates that improve over time. Since then, database query approximation systems allow users to tradeoff query accuracy for response time or energy usage while providing statistical error guarantees [1, 8, 10, 22, 23]. We allow users to tradeoff the precision of stale data for storage space reductions. Unlike these works, we do not explicitly focus on bounding error.

4. Conclusion and Future Work

Forgetful storage directly contravenes the typical reliability and durability requirements that systems builders have come to expect. In this paper, we argue that forgetfulness can actually be a useful and intuitive property if harnessed properly. We define forgetfulness as a passive process that gradually degrades data without inflation or additional overheads. We then demonstrate how to implement such a data structure through the design of a proof of concept, the forgit. We show through proof-of-concept forgits that forgetful data structures can produce loss properties useful for different applications. The design of low-overhead transforms for other data types and the integration of forgetful data structures into large-scale high performance systems are challenging research problems that merit more study.

References

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42. ACM, 2013.

[2] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *Transactions on Computers*, 100(1):90–93, 1974.

[3] V. Bhaskaran and K. Konstantinides. *Image and video compression standards: algorithms and architectures*. Springer Science & Business Media, 1997.

[4] M. Bosi, K. Brandenburg, S. Quackenbush, L. Fielder, K. Akagiri, H. Fuchs, and M. Dietz. Iso/iec mpeg-2 advanced audio coding. *Journal of the Audio engineering society*, 45(10):789–814, 1997.

[5] R. Bracewell. The Fourier Transform and IIS Applications. *New York*, 1965.

[6] A. M. Dunn, M. Z. Lee, S. Jana, S. Kim, M. Silberstein, Y. Xu, V. Shmatikov, and E. Witchel. Eternal sunshine of the spotless machine: Protecting privacy with ephemeral channels. In *OSDI*, pages 61–75, 2012.

[7] G. Erkan and D. R. Radev. Lexrank: graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, pages 457–479, 2004.

[8] D. Fisher. Incremental, approximate database queries and uncertainty for exploratory visualization. In *Symposium on Large Data Analysis and Visualization (LDAV)*, pages 73–80. IEEE, 2011.

[9] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy. Vanish: Increasing data privacy with self-destructing data. In *USENIX Security Symposium*, pages 299–316, 2009.

[10] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, et al. Approximate string joins in a database (almost) for free. In *VLDB*, volume 1, pages 491–500, 2001.

[11] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *SIGMOD*, 26(2):171–182, 1997.

[12] C. Herley. Storage of digital camera images. In *Image Processing*, volume 3, pages 940–942. IEEE, 1999.

[13] M. L. Kersten. Big data space fungus. In *CIDR*, 2015.

[14] Linguistic Data Consortium - Web 1T 5-gram Version 1. **https://catalog.ldc.upenn.edu/LDC2006T13**.

[15] A. Mantelero. The EU Proposal for a General Data Protection Regulation and the roots of the 'right to be forgotten'. *Computer Law & Security Review*, 29(3):229 – 235, 2013.

[16] V. Mayer-Schonberger. *Delete: The Virtue of Forgetting in the Digital Age*. Princeton University Press, 2009.

[17] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. Association for Computational Linguistics, 2004.

[18] S. K. Nair, M. T. Dashti, B. Crispo, and A. S. Tanenbaum. A hybrid pki-ibc based ephemerizer system. In *New Approaches for Security, Privacy and Trust in Complex Environments*, pages 241–252. Springer, 2007.

[19] R. Perlman. The ephemerizer: Making data disappear. 2005.

[20] G. Reeves, J. Liu, S. Nath, and F. Zhao. Managing massive time series streams with multi-scale compressed trickles. *PVLDB*, 2(1):97–108, 2009.

[21] I. E. Richardson. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004.

[22] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *SIGPLAN Notices*, volume 46, pages 164–174. ACM, 2011.

[23] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage in solid-state memories. *ACM TOCS*, 32(3):9, 2014.

[24] E. Shein. Ephemeral data. *CACM*, 56(9):20–22, Sept. 2013.

[25] Silent Circle. **https://silentcircle.com/**.

[26] Snapchat. **https://www.snapchat.com/**.

[27] Y. Tang, P. Ames, S. Bhamidipati, A. Bijlani, R. Geambasu, and N. Sarda. Cleanos: Limiting mobile data exposure with idle eviction. In *OSDI*, volume 12, pages 77–91, 2012.

[28] Y. Tang, P. P. Lee, J. C. Lui, and R. Perlman. Fade: Secure overlay cloud storage with file assured deletion. In *Security and Privacy in Communication Networks*, pages 380–397. Springer, 2010.

[29] V. Toubiana and H. Nissenbaum. An analysis of google logs retention policies. *Journal of Privacy and Confidentiality*, 3(1):2, 2011.

[30] G. K. Wallace. The jpeg still picture compression standard. *CACM*, 34(4):30–44, 1991.

[31] Wickr. **https://www.wickr.com/**.