Building Deletion-Compliant Data Systems

Manos Athanassoulis, Subhadeep Sarkar, Tarikul Islam Papon, Zichen Zhu, Dimitris Staratzis
Boston University

Abstract

Most modern data systems have been designed with two goals in mind – fast ingestion and low-latency query processing. The first goal has led to the development of a plethora of write-optimized data stores that employ the out-of-place paradigm. Due to their write-optimized design, out-of-place data systems perform deletes logically via invalidation, and retain the invalid data for arbitrarily long. However, due to the recent enactment of new data privacy regulations, the requirement of timely deletion of user data has become central. The right to be forgotten (in EU's GDPR), right to delete (in California's CCPA and CPRA), or deletion right (in Virginia's VCDPA) mandates that service providers persistently delete a user's data within a pre-set time duration. Logical deletion in out-of-place data systems, however, does not offer guarantees for timely and persistent deletion, and attempting to enforce it using existing tools leads to poor performance and increased operational costs.

In this paper, we present a new framework for building deletion-compliant data systems from a holistic perspective. We analyze the new regulations and the requirements derived from the new policies, and we propose changes in the application and the system layer of data management. We outline the new types of deletion requests that need to be supported, the query language modifications needed to be able to request for timely persistent data deletion, and the system-level changes needed to realize timely and persistent deletes. The proposed framework for deletion compliance lays the groundwork for a new class of data systems that can offer system-level guarantees for user data privacy. We present recent results spanning all layers of the framework: the requirements and the application layer target any database system, while the system layer discussion is geared towards out-of-place systems. Finally, we conclude with a discussion on next steps and open challenges on building deletion-compliant data systems.

1 Introduction

Data-intensive social and commercial applications and new advancements in domains like Internet-of-things, edge computing, 5G communications, and autonomous vehicles, generate a vast amount of *personal data* processed by several data companies [26, 40]. The increasing demand for efficient collection, storage, and processing of user data over the past two decades, has driven the development of data systems that can *sustain high ingestion rates* without compromising the ability to *access and analyze the data quickly*.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Out-of-Place Systems. The need for optimizing data ingestion while maintaining efficient data access has led to the prominence of the *out-of-place* paradigm, which fulfills these goals by minimizing the interference between reads and writes. Today, several commercial relational and array-based data stores [13, 34, 39, 46, 50, 53, 59, 71, 85] and NoSQL data stores [10, 12, 11, 32, 38, 42, 48, 79] have adopted the out-of-place paradigm.

Relational and Array-based Systems. Relational systems that buffer updates before applying them lazily on the base data, essentially, follow the out-of-place paradigm. The columnar and array data stores implemented by Vertica [59, 85], SciDB [68, 86], and TileDB [67, 88] use an in-memory storage component that stores incoming inserts, updates, and deletes out of place, and applies the changes lazily on the disk-resident data. Similarly, the state-of-the-art column-store system MonetDB [50] uses an in-memory positional index for incoming data [46], and SAP HANA uses a delta store per table to facilitate fast ingestion without affecting its read-optimized data layout [39]. Finally, several research-prototype systems use a separate delta store on faster storage (e.g., SSDs/NVM) to offer efficient access to incoming data [13, 14, 34, 53, 71].

NoSQL Systems. More than relational systems, production-grade NoSQL key-value stores predominantly employ the out-of-place paradigm, frequently based on the log-structured merge (LSM) paradigm. An LSM-tree is a heavily write-optimized out-of-place data structure that maintains several on-disk components, which can be viewed as several out-of-place delta stores [66, 31, 49, 62, 73, 92]. Key-value stores such as RocksDB [36, 38] and LevelDB [43] at Facebook, BigTable [24] at Google, X-Engine [48, 91] at Alibaba, Voldemort [61] at LinkedIn, DynamoDB [32] at Amazon, Cassandra [12], HBase [11], and Accumulo [10] at Apache, and bLSM [79] and cLSM [42] at Yahoo are based on the log-structured merge (LSM) paradigm. Other out-of-place architectures employed by NoSQL systems are B^+-tree, B^ϵ-tree, and fractal tree-based storage engines with *buffering support*, such as COLA [16], TokuDB [58], and B*ε*rtFS [17, 51].

Cloud-based Systems. Cloud-based systems naturally employ the out-of-place paradigm as they rely on the immutability of cloud storage. Hence, systems like Amazon Redshift [8, 44], Cloud Data Platform [28] at Snowflake, and Delta Lake [29, 30] at Databricks employ variations of the out-of-place paradigm in the interest of performance. Deletes and updates are initially performed logically and are gradually propagated to persistent media through periodic merging with base data.

Deletes in Out-of-Place Systems. A key property of out-of-place systems is that they treat deletes (and updates) similarly to inserts, i.e., instead of deleting (updating) entries in-place, they insert a new version of the entry to be deleted that *logically invalidates* the target entries. These special entries that are responsible for logical deletes are termed *delete markers* [59] or *tombstones* [36, 76].

Logical data deletion is a quintessential out-of-place operation, but it does not guarantee *purging* of the data under deletion within a definite timeframe. Rather, the data is marked as invalid; essentially, *not accessible* to external users. In practice, logically deleted entries are kept for arbitrarily long in the system, since the time to definitively delete the data (termed *persistent deletion*) depends on the state of the system, and not on when the user request expects the data to be deleted [76]. In fact, most out-of-place data stores are built with the underlying assumption of *perpetual data retention* in order to gain more insights from the user and organizational data [90], hence *timely persistent deletion* has not been part of their design goals. In addition to deletes, logical updates in out-of-place systems are applied lazily too, however, the implications of out-of-place deletes are critical in terms of the privacy regulations, and thus, are our focus.

1.1 Problem: The Privacy Concern

Cost of Logical Deletes. Logical deletes and updates in out-of-place systems boost ingestion performance, however, they come at a significant cost. In fact, when tasked with deleting user data persistently in a timely manner, out-of-place systems suffer both in terms of (a) data privacy protection and (b) the overall system performance. Such systems are designed to retain the logically invalidated data indefinitely, and the time required for persistent removal of the physical data entries depends on (i) the data layout, (ii) the data re-organization policy (e.g., node splitting/merging in B-trees, compaction in LSM-trees, consolidation in TileDB), (iii) the

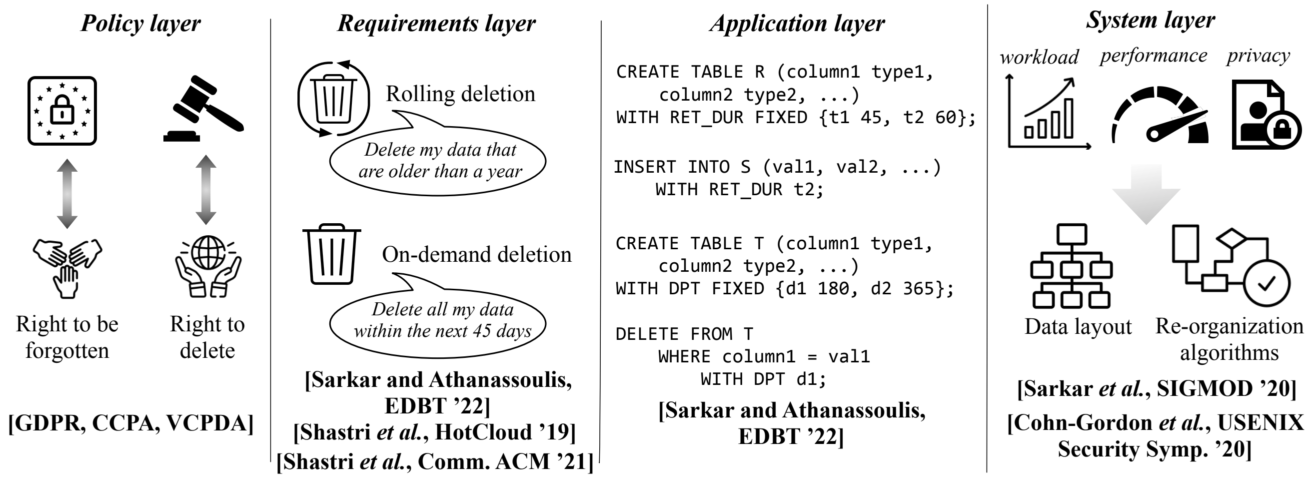

Figure 1: The four layers of deletion-compliant data systems.

design of the storage engine (such as the fanout of a tree and the size of a database), and (iv) the composition and distribution of the workload – factors that are *beyond the control of the application and system developers or administrators*. Thus, most out-of-place systems are unable to provide any latency guarantees for persistent deletion of user data [76].

The Legal Frontier. In recent years, a number of government-driven efforts across the globe unfolded, aiming to protect the privacy of user data and give back to the users the control of their personal data. On the legal side, regulations such as the EU's GDPR [26], California's CCPA [3] and CPRA [6], and Virginia's VCDPA [7] have been introduced, which mandate that data companies ensure *privacy through deletion* [83, 84]. GDPR's *right to be forgotten*, CCPA and CPRA's *right to delete*, and the *deletion right* in VCDPA particularly focus on *persistent deletion of user data on-demand and in a timely manner* [9, 35, 41, 52, 75, 83, 24, 89].

The Technological Roadblock. Treating *deletes* as *first-class citizens* is new for the data systems community, and it would require a significant amount of work to transform classical systems to be efficient deletion-wise. Even today, it continues to be a critical technological challenge for the biggest of data companies using state-of-the-art storage engines to demonstrate compliance with the deletion regulations and to efficiently delete user data on-demand [81, 29, 84]. To translate this into numbers, between January 2020 and January 2022, the penalties under GDPR paid by data companies amounted to more than $1B, which includes large contributions from companies such as Amazon ($877M), WhatsApp ($255M), Google Ireland ($102M), and Facebook ($68M), H&M ($41M), British Airways ($26M), and Marriot ($23M) [64, 70, 87]. Thus, to demonstrate compliance, many companies end up performing expensive database-wide consolidations periodically (e.g., every few weeks), to ensure timely persistent deletion of user data [76, 77]. Such operations are remarkably expensive in terms of time and money, cause undesirable latency spikes, and hence, should be avoided.

1.2 Deletion-Compliant Data Systems

In this paper, we present our vision and first results on designing data systems that ensure data privacy through timely and persistent deletion of user data. Existing efforts that attempt to delete user/application data on-demand suffer in terms of performance as the underlying data layout and data management mechanisms are ill-suited for the purpose. We identify the missing links, in terms of technological tools, both at the application level and the system level, and we propose a hierarchical framework that enables our vision of privacy through deletion in out-of-place data systems (Figure 1).

Roadmap. The privacy through deletion framework is a roadmap toward building deletion-compliant data systems. We begin by outlining the challenges associated with each layer of our vision, i.e., in the context of (i) translating the legal mandates to user requirements, (ii) expressing the user requirements through a declarative API, and (iii) realizing the application-level requirements at the system level. Next, we identify and categorize the different classes of user requests for deletes in light of the legal regulations. Based on this, we present the challenges associated with transforming the classes of deletion requests into application-level specifications, and we propose an SQL extension as an example that can be extended to other query languages to support deletion of user data *periodically* and *on-demand*. Further, we outline the design and tools necessary at the system level to support the application-level requirements. Finally, we conclude with a discussion on how the proposed framework drives us toward building deletion-compliant data systems, and what further research challenges remain open to fully realize this vision.

2 From Regulation to Practice

The legal landscape for data privacy has changed drastically over the past few years, and governments across countries, as well as across different states in the US, have enforced acts and regulations to control the consumption of user data by service providers and give back to the users the control of their personal data. Translating the new regulations to new *user-data privacy-compliant system behavior* still faces significant challenges. In this section, we present in more detail the requirements from the regulation point of view, and we showcase through three realistic scenarios the limitations of the state-of-the-art data systems when tasked to implement these requirements.

2.1 Regulations on Timely Data Deletion

While the new regulations propose an array of new requirements, we particularly focus on the legal policies concerning data retention and data deletion, with the objective of ensuring *privacy through deletion*.

Right to be Forgotten, *EU GDPR*. The General Data Protection Regulation (GDPR) has revolutionized the data privacy and security landscape across the European Union countries [26]. One of the fundamental changes introduced through the GDPR (over the older Data Protection Act (DPA) that it replaced), is the *right to be forgotten*, which empowers the users with the *right* to request a service provider to delete all their personal data persistently from its domain. Such deletion requests may be presented either up-front or on-demand. The service provider must comply with those requests, unless it has compelling reasons for acting otherwise (Art. 17(3)).

Right to Delete, *CCPA*, *CPRA*. The California Customer Protection Act (CCPA), which will eventually be replaced by the California Privacy Rights Act (CPRA) in 2023, allows the users/consumers in California to request from service providers to permanently delete all data personal to the user [3, 6]. Under CCPA and CPRA, the service providers must acknowledge such a user request within 10 days, and respond to the request within 45 business days [19]. Persistent deletion must be performed by removing the target data across all domains, barring archive and backup systems, along with data anonymization as required.

Right to Delete, *VCDPA*. Similarly to CCPA, the Virginia Consumer Data Protection Act (VCDPA) empowers users in Virginia to exercise their right to delete their personal data from a provider's domain [7]. VCDPA requires the service providers to serve a delete-request from a user within 45 business days [19].

Right to be Forgotten, *UK GDPR, DPA*. The UK GDPR, along with the Data Protection Act (DPA) 2018 provides the country's citizens with similar rights about personal data deletion as the EU GDPR. The users are allowed to express their deletion preference verbally or in writing, to which the service providers must respond within 30 days [1, 4].

Other Efforts. Among other countries, Argentina [23, 69], Singapore [25], India [55], Canada [5], and South Korea [18] have some implementation of the right to deletion as a part of their respective privacy protection acts.

2.2 Limitations of the State of the Art

In light of the deletion regulations, we now present three real-life scenarios to highlight why state-of-the-art data systems are ill-equipped to support deletes efficiently without hurting performance. We do so by identifying the missing links in different hierarchical levels of the proposed privacy-through-deletion framework. Below, we illustrate (i) that the users are unable to express their preferences about deleting their personal data, (ii) why it is difficult for application developers to support the deletion requests from the users, and (iii) why it is difficult to realize persistent deletes in a timely manner in live production servers.

Scenario 1: *Alice* is a user of a smart-home ecosystem, *HomeComp*, which provides real-time services including video surveillance, remote temperature, and illumination control. *Alice* enjoys the services of *HomeComp*, but concerned about her personal data privacy, she wants *HomeComp* to permanently delete all her data older than 30 days on a rolling basis.

The problem? Like most service providers, *HomeComp*'s data model is built around the assumption of perpetual data retention; deletion of user data needs a human-in-the-loop that performs the necessary actions. Thus, *HomeComp* does not allow its user to request for rolling timestamp-based data deletion.

Scenario 2: *StreamEra* is a company that provides real-time insights for data streams, and allows its users to request on-demand deletion of their personal data, as it is bound by the *right to be forgotten*. *StreamEra* uses an SQL-based wrapper on top of its storage layer.

The problem? While *StreamEra* wants to serve its users by ensuring timely persistent deletion of their personal data, SQL does not provide support for such an operation. Instead, the backend engineers implement the user-requested deletion functionality at the application level in an ad-hoc manner as it is not native to SQL.

Scenario 3: A cloud-based online data analysis company *ClouData*, stores user data using immutable files within its HTAP data store. *ClouData* is bound by the *right to be forgotten*, and thus, has to delete all user data that are older than D days.

The problem? As the data organization on disk is not based on the ingestion timestamp and aims to accelerate read queries, it uses the most frequently queried attribute to partition. Hence, the objects qualifying for a timestamp-based deletion may be dispersed within the data store. As in-place deletion is not supported due to immutability, state-of-the-art data stores periodically consolidate the entire data set to delete all invalid entries. Ensuring privacy via this approach is costly in terms of disk writes and overall accesses, and causes latency spikes leading to performance unpredictability.

Other Challenges of Logical Deletes. In addition to not complying with regulatory requirements, logical deletes may cause more hurdles. Specifically, by retaining invalidated data (that should not be used anymore), a data company:

1. Wastes storage space and energy on data that cannot exploit in any way. Further, data maintenance results in additional write amplification that wears off the underlying storage devices [15].

2. Risks that a security leak will reveal user data that users expect to be deleted [64].

3. Hurts read performance, as its data management layer uses metadata and indexes for all data regardless of whether they are invalidated [76].

3 Privacy Through Timely Deletion

We now outline our vision toward developing deletion-aware data systems, which by design, are capable of *deleting user data persistently, and in an efficient and timely manner*. Toward this, we introduce a new set of application level and system level tools that capture, transform, and realize the user-requirements for deletes.

Figure 1 shows our four-layered approach. The first step is the *policy layer*, implemented by the governments, that enact specific clauses to protect data privacy through deletion. The second layer is the *requirements layer* that translates the regulations into application requirements. Next, we have the *application layer* that proposes the necessary changes in query languages to allow applications to easily express their constraints. Finally, the *system layer* implements efficient means for data deletion and demonstrates regulation compliance.

3.1 Requirements Layer

Challenge. This layer analyzes the regulations from the policy layer and categorizes the various requests the user *should be* able to make on the application layer. The impact of the newly enacted policies is on various aspects including accountability (audit), security (protect data access), and right of access (efficient accessing) [81, 84]. In this work, we focus on *storage limitation* ("data should not be stored beyond its purpose"), the *right to be forgotten* ("find and delete groups of data") [81], and how to transform them into concrete requirements.

Types of Deletion Requests. We codify the two types of data deletion requests as requirements for (a) retention-driven *rolling deletion* and (b) *on-demand deletion* both with a timely constraint [76], as illustrated in the second part of Figure 1.

Retention-driven deletes. In cases that the purpose of storing the data has expired, a rolling deletion should take place, which will ensure that the underlying data management solutions persistently delete this data, based on a pre-set *retention duration*. This duration can be governed by legislation, the specific application, or even user preference, hence it has to be tunable. To abide by the policies, the data management layer has to permanently delete expired items within a specific timeframe, provided by the service-level agreement (SLA) between the user and the service providers.

Deletion on-demand. The regulations for deletes also allow users to submit on-demand deletion requests for any personal data, upon which the service provider has to delete user data persistently. On-demand deletion requests can be submitted through an API provided by the service provider, and upon submission, all data for a user are purged persistently within a threshold period. This threshold for persistent deletion is also set by the provider following the regulations and is agreed upon in the form of an SLA-clause.

3.2 Application Layer

Challenge. With the deletion-related regulations translated to deletion requirements, the next step is to transform them into a format that is interpretable by the application layer. The interface of data stores is typically declarative query languages (e.g., SQL, GraphQL, DMX, LINQ, and N1QL) that support expressing complex queries as well as inserting new data, updates, and deletes. The missing link here is that state-of-the-art query languages do not have support for data deletion based on retention and does not have a way to express the timely deletion requirements.

Extending SQL. Hence, to implement the deletion requirements, we propose an extension to SQL [74] that includes support for timely deletion both in the data definition (DDL) and the data manipulation (DML) parts of the language, as summarized in the third part of Figure 1. The objective of the SQL extension is three-fold.

1. To support *retention-driven deletion*, we augment both the `CREATE TABLE` and `INSERT INTO` statements so that a relational table can be associated with a number of options for specific time-to-live (TTL). Every data object is bound to a specific TTL according to the application SLA or to user preference.

2. To ensure *timely persistence of on-demand deletion requests*, we augment the `CREATE TABLE` and `DELETE FROM` statements to allow a relational table to support a predetermined set of timely deletion guarantees, and each deletion to select the level of service to which it adheres.

3. Finally, we extend the `CREATE TABLE`, `INSERT INTO` and `DELETE FROM` statements to support *arbitrary delete thresholds* for retention duration and deletion persistence.

Enabling retention-driven deletes. To support retention-driven deletes, we extend `CREATE TABLE` to allow an application developer to specify several levels of retention duration as a table property.

```
CREATE TABLE R (column1 type1, column2 type2, ...)
WITH RET_DUR FIXED (t1 <ret1>, t2 <ret2>, ...);
```

The above `CREATE TABLE` statement creates a table `R` that supports retention-based deletes with specific retention duration of `ret1`, `ret2`, etc, which are mapped to symbolic representations `t1`, `t2`, etc. In general, the `WITH RET_DUR` clause is an optional clause when creating a new table, and will be necessary only for tables that need to support deletes with predefined retention duration values. In such cases, each `INSERT` statement can use (up to) one of the predefined retention duration values, say `t1` to classify the specific object as one to be deleted after `ret1` time. For example, a table that is configured to support retention duration of 30 days and 60 days (`CREATE TABLE R (...) WITH RET_DUR FIXED (t1 '30 days', t2 '60 days');`), can only receive inserts with retention duration `t1` or `t2`. An ingestion without a retention period explicitly mentioned, is kept perpetually following the logic of a classical insert. The syntax of an insert, now, has the optional `WITH RET_DUR` clause as follows.

```
INSERT INTO R (val1, val2, ...) WITH RET_DUR t<i>;
```

Support for arbitrary retention duration. To support arbitrary retention duration, we further add the `ARBITRARY` keyword to both the `CREATE TABLE` and `INSERT` statements. The support for arbitrary retention duration is necessary particularly for systems in a distributed setting that replicate data across physical data stores in different geolocations, each bound by different regulatory requirements. The full syntax of the proposed SQL extension for retention-based deletion is below.

```
CREATE TABLE R (column1 type1, column2 type2, ...)
WITH RET_DUR {ARBITRARY | FIXED (t1 <ret1>, t2 <ret2>, ...)};

INSERT INTO R (val1, val2, ...) WITH RET_DUR { <t> | t<i> } ;
```

Note that having a *pre-defined set of retention duration values* provides more information to the system compared to allowing arbitrary duration. As a result, it *allows the system to better prepare* to offer efficient retention-driven deletes. Conversely, we expect that data stores that aim to support arbitrary retention duration will face increased system-level challenges.

Enabling timely on-demand deletion. We further propose to augment SQL to express timely on-demand deletion. To do so, we introduce the concept of *delete persistence threshold* (DPT) [76], which denotes the maximum delay between a logical delete and its persistence. Every relational table can be associated with several such thresholds that are defined from the legal constraints or based on user preference. Similarly to retention-driven deletes, we also extend SQL to support arbitrary DPTs when the DPTs are not specified *a priori*. Below, we outline the modifications to the DDL and DML parts of SQL to support on-demand timely deletion requests.

```
CREATE TABLE S (column1 type1, column2 type2, ...)
WITH DPT {ARBITRARY | FIXED (d1 <dpt1>, d2 <dpt2>, ...)};

DELETE FROM S WHERE (...) WITH DPT { <d> | d<i> };
```

Table S can support several DPTs (e.g., dpt1, dpt2) as long as the DPTs are pre-determined. Applications can trigger on-demand deletion with any such DPT through the DELETE command. Similarly to retention-driven deletes, timely persistent deletion of data on-demand is easier to handle from a storage engine if the DPTs supported are known *a priori* during the table creation.

Putting everything together. With the proposed SQL extensions, a relational table can now support multiple (pre-defined or arbitrary) thresholds for both retention-based and on-demand deletes, with the following CREATE TABLE statement.

```
CREATE TABLE T (column1 type1, column2 type2, ...)
WITH RET_DUR {ARBITRARY | FIXED (t1 <ret1>, t2 <ret2>, ...)};
WITH DPT {ARBITRARY | FIXED (d1 <dpt1>, d2 <dpt2>, ...)};
```

Note that typically retention-based deletes come from the *application requirements*, and on-demand deletion requests are issued *by the user*. However, in both cases, the deletes have to happen *timely* as per the regulatory requirements. The proposed changes in SQL are not enough to guarantee that the system will deliver on the need for timely and persistent data deletion. Rather, they create the interface for data systems so that need for timely deletion. Rather, they create the interface for data systems so that the users and applications can express the deletion requests which is enforced by the regulations. The data deletion *per se* is realized at the *system level*, and we will next discuss advances and challenges on that front.

3.3 System Layer

With the requirement analysis and the declarative interface in place, the users and the applications can express all the mandated deletion requests and the underlying system is now tasked with implementing them. Before discussing the challenges of implementing timely retention-based and on-demand deletes, we discuss the taxonomy of system-level deletes the application layer may initiate. In other words, we want to understand what delete patterns may be generated at the system layer.

3.3.1 Taxonomy of Deletes at the System Layer

The behavior of a low-level delete operation depends on (i) the logical organization and physical layout of the data, and (ii) the attribute based on which the deletion requests are issued. To better understand this *delete design space*, we classify different delete operations across two dimensions: (a) deletes on *primary* vs. *secondary* attributes, and (b) deletes based on a single value of the delete attribute, termed *point deletes*, vs. deletes on a range of the delete attribute, termed *range deletes* [76]. Table 1 summarizes the state of the art in out-of-place systems for different delete operations, their performance impact, and their at-large implications.

Delete Workloads	**Primary Deletes** Point	Range	**Secondary Deletes** Point	Range
State-of-the-art implementation	insert point tombstones (*logical*)	insert range tombstones (*logical*)	not supported	full-tree compaction
Point query path	search for key; stop if a tombstone is found	search for key; compare fetched key with the histogram (discard if invalidated)	N/A	N/A
Range query path	merge qualifying sorted runs; discard on the fly if TS exist	merge qualifying sorted runs; check each value against histogram	N/A	N/A
Implications	unbounded persistence latency high space amplification high write amplification	unbounded persistence latency high space amplification high write amplification severely affects read performance	N/A	huge latency spikes high write amplification superfluous reads from disk

Table 1: Implications of logical deletes on performance in state-of-the-art out-of-place data stores.

Primary Deletes. In out-of-place systems, data is ultimately organized based on a so-called *sort key* (for example, the key of the key-value pairs in an LSM-tree). The sort key often is the primary key of the database, hence, a majority of delete operations can be expressed as deletes based on the sort key, or *primary deletes*. Note that even deletes on other attributes may be preferable to be converted to primary deletes if there is a secondary index. Both point and range primary deletes use the notion of a *delete marker* or a *tombstone* that is inserted in the data collection (on the deleted sort key) and invalidates prior version of the key(s), and they are very common in real workloads [22]. Primary deletes can be triggered either by user activity (i.e., on-demand) or by automated processes (e.g., data migration).

Implications and Challenges. When an out-of-place system has files that contain tombstones, then both point and range query paths are affected. Specifically, since the system accesses files with decreasing age (i.e., the most recent ones first) when looking for a key (*point query*), it will also be looking for tombstones. If a tombstone is found, the search will terminate because all other (older) instances of that key are invalid. In the presence of range deletes, the implementation is more complex as it is hard to use per-file range delete tombstones [63]. Instead, a database-wide histogram of deleted ranges is maintained and every query compares against this histogram before proceeding [76]. When considering *range queries*, all the sorted files have to be merged and the deleted entries are discarded on the fly by comparing against the visited point and range tombstones [21].

The implications of primary deletes are multi-fold. First, while out-of-place systems support deletes, any deletion is logical, and there is no *a priori* bound on the delete persistence latency. Second, by maintaining both the tombstones and the invalid entries for arbitrarily long time, the systems pay in terms of increased space amplification. Thirdly, by reorganizing data including invalid entries and tombstones, we further pay in terms of increased write amplification. Note that space and write amplification [15, 36] are two fundamental sources of cost when deploying data system. Finally, while range tombstones are used to offer the range delete functionality, they are rather cumbersome and impact the read performance severely [21, 63].

Secondary Deletes. In some other cases, we may need to organize data based on a sort key, but we have a majority of deletes on a different attribute. Note that if we have individual deletes on a different attribute, the most prudent approach is to guarantee that we have a secondary index and transform a secondary delete in one (or more) primary point delete, hence in Table 1, we see the lack of support for point secondary deletes. However, in some cases, we may have long range deletes on a secondary attribute. For example, when working on a window of the most recent data we can repetitively delete data based on a timestamp. A similar case is the retention-based deletes introduced earlier.

Implications and Challenges. Secondary range deletes are not native in out-of-place systems, since the underlying data is organized based on the sort key. While converting them to a collection of point primary deletes might work in several cases, it will overload the system with tombstones. Instead, several systems opt to perform a full-database merging and re-writing periodically to fulfill any secondary range deletion constraints they might have to follow. This approach leads to significant write amplification, superfluous data accesses, and a large penalty in terms of latency spikes on the workload during this merging.

From Delete Requirements to Delete Types. The delete taxonomy at the system level helps us map the delete requirements to low-level data operations. A retention-driven deletion is typically modeled as a secondary range delete, and if the delete range has few objects (i.e., low selectivity), it can be implemented as a collection of primary point deletes. On the other hand, on-demand deletion is typically implemented as a primary point or range delete and there are several performance challenges to be addressed.

3.3.2 Realizing Timely Deletes

Timely data deletion while respecting the retention SLAs without hurting the system performance is a key challenge. The efficiency of deletion depends on the schema and the physical data layout, the data re-organization strategy, the workload, and the design of the storage engine. The right-most part of Figure 1 outlines the

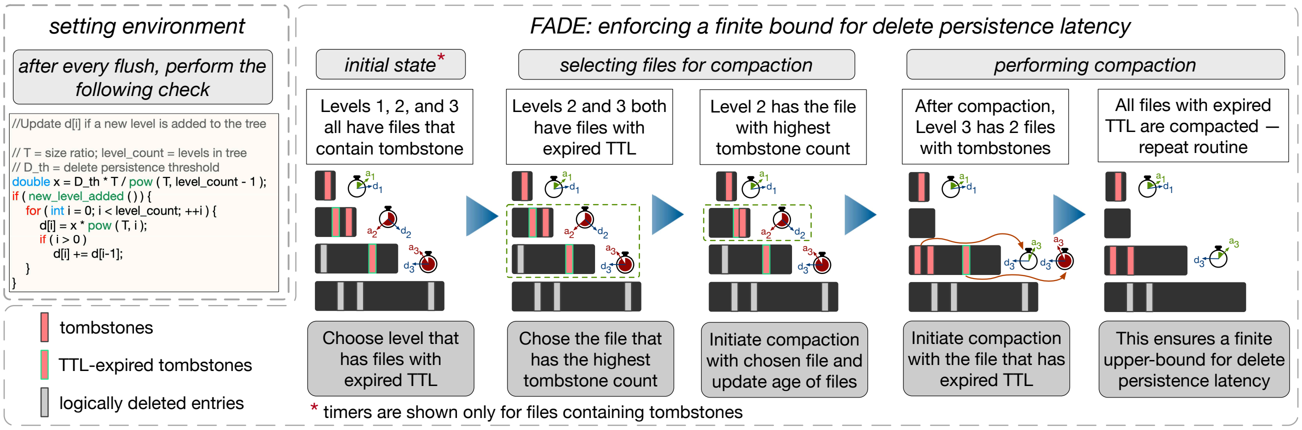

Figure 2: FADE persists tombstones within DPT, thus, improving overall performance.

design changes needed and the input parameters used to co-optimize the performance of a storage engine while ensuring timely delete persistence. We now discuss how both classes of deletes can be realized efficiently through modifications in the design of out-of-place storage engines, focusing on LSM-bases storage engines.

Realizing Primary Deletes. An LSM-based storage engine implements a primary delete by inserting a tombstone on the desired key (or key-range) with a DPT associated. The application-provided DPT is an indicator for how long a logical delete may live before its persistence, that is, before purging any invalidated versions of the key(s) under deletion. The tombstone representation is augmented with additional metadata, e.g., an extra byte to account for 128 possible different DPTs in the same table. During its natural course of data re-organization the storage engine checks for any data blocks (i.e., pages, files, or sorted runs) with an expired TTL and consolidates them to ensure timely persistence. This data consolidation in LSM-based data stores is called *compaction* [72, 77] and is the process of selecting some components of the database (files, sorted runs) to merge and discard invalid entries. At any point of time an LSM-based system has several files that may be compacted (it can be in the order of several thousands) so the decision which files to compact is a crucial decision. In general, the decision is based on read query metrics, however, in Lethe [76] we propose a new approach that prioritizes compactions of files depending on the age of the tombstones they contain. Specifically, we assign different TTLs based on the level of the underlying LSM-tree and when there is a tombstone with an expired TTL we select the file that contains it for compaction, as shown in Figure 2. A key decision is how to ensure that the multi-step merging of tombstones will always respect the application-defined DPT. This is ensured by assigning a different TTL to each tombstone after every compaction in a way that the sum of all its TTL amounts to the desired DPT.

Realizing Secondary Deletes. As we discussed above, several instances of secondary deletes can be realized as a collection of primary point deletes. However, when we are frequently tasked to deleted a range of values based on a secondary attribute, we can achieve something significantly better. In particular, a new weaved data layout between the original sort key and the (secondary) delete key can offer much more efficient and timely secondary range deletion while maintaining competitive read performance. The key idea is to create a nested data organization that alternates between organizing data based on the sort key (to facilitate good search performance) and based on the delete key (to allow for consecutive chunks of data to be deleted at a time).

This approach is implemented in the KiWi data layout [76] as shown in Figure 3. The core idea is that while the major components of the database (files) are organized based on the sort key, every file is composed of delete tiles that are internally organized based on the delete key, partitioning the data accordingly. Lastly, each data page is again organized on the search key to facilitate efficient in-memory search. The benefit for this weaved data layout is that in the case of secondary range deletes, we can discard entire groups of pages at a time, signaling the file system to reclaim this page instantly, essentially converting the secondary delete to a page reclamation action

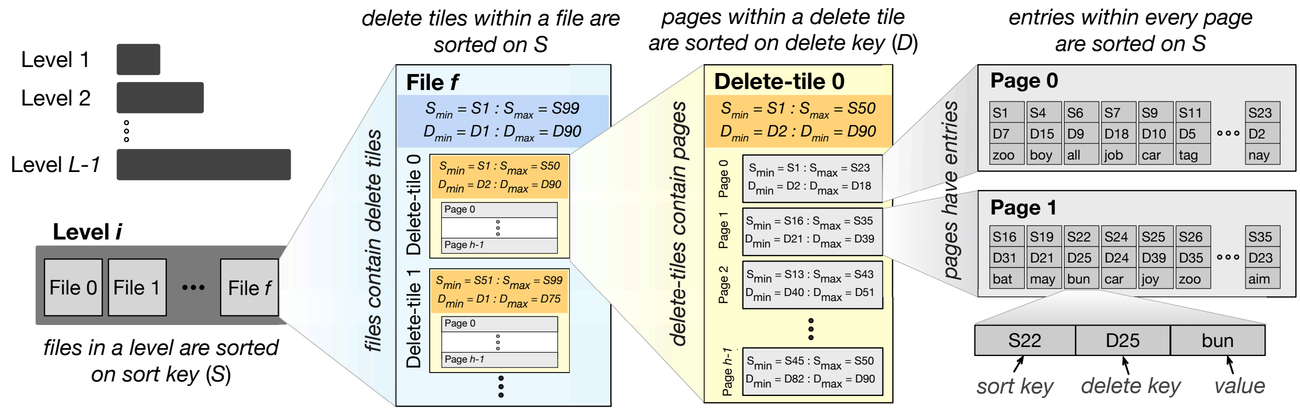

Figure 3: KiWi stores data in an interweaved fashion on the sort and delete key to facilitate efficient secondary range deletes while offering competitive read performance.

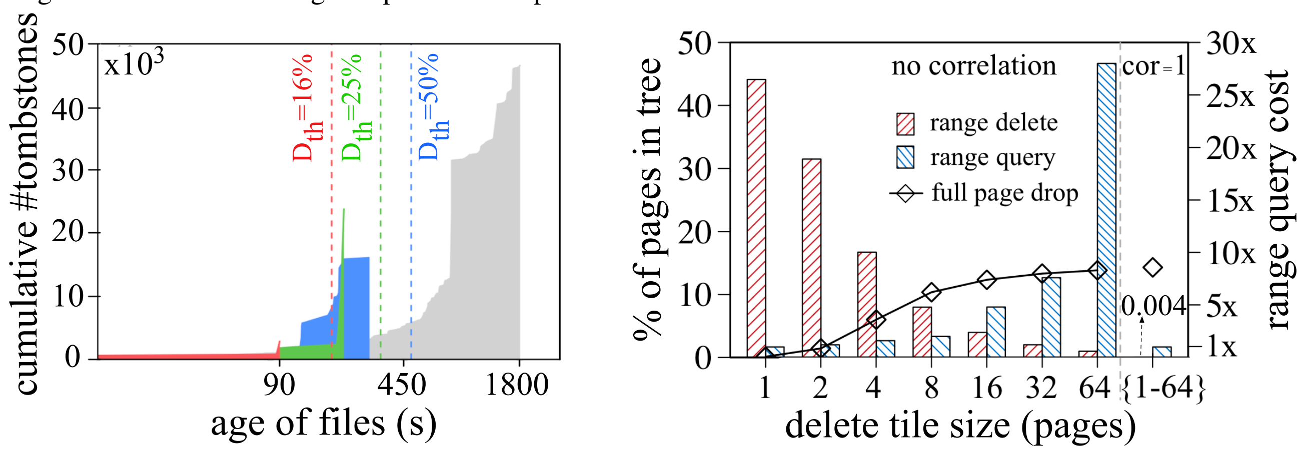

Figure 4: Lethe ensures timely persistence of logically invalidated data within LSM-based out-of-place data systems for both primary and secondary classes of deletes.

that has very low latency compared to a full database reorganization. In the worst case, we will have to in-place edit a few pages at the edge of the range, which is a tunable parameter that controls the maximum secondary deletion persistence latency as a tradeoff vs. read performance.

Evaluation. The approaches presented above for timely deletion were implemented as part of the LSM-based system Lethe [76], and achieved efficient timely deletion respecting predetermined guarantees. The left hand-side of Figure 4 shows the CDF of the tombstone age while varying the desired DPT to 16%, 25%, and 50% of the duration of the experiment. The colored areas correspond to the number of cumulative tombstones for the corresponding age on the x-axis, while the horizontal dotted-line is the desired DPT. We observe that Lethe was able to always deliver the requested DPT. The gray area corresponds to the age of tombstones of the state of the art, where no DPT is imposed and deletes are not persisted timely. Notably, we also measured that enforcing the desired DPT shows benefits in terms of access time because the amount of invalid data was reduced. Similarly, we saw benefits in space amplification, and only marginal cost increase in amortized write amplification.

The right hand-side of Figure 4 shows the fraction of fully dropped pages during a range delete as we vary the size of the delete tiles. We observe that the fraction of pages fully dropped increases with the delete tile size, allowing for efficient reclamation of the invalid data. Conversely, the read queries become more expensive as we allow for more page drops, so the ideal delete tile size should be tuned based on the workload.

Deletes in a Complex Data Model. The previous discussion focuses on handling deletes in a per-instance manner without considering multiple copies of the data in a more complex setting. Cohn-Gordon *et al.* [27] proposed the deletion framework *DELF* that ensures reliable data deletion from an online social network (OSN). *DELF* enables detection of inconsistent data deletion in OSNs and also facilitates data recovery in cases where user data was incorrectly deleted. Minaei *et al.* [65] proposed a framework for persistently deleting all instances of user data in presence of observers, thereby, ensuring privacy through timely content concealment and removal.

4 Challenges and Opportunities

In Section 3, we outlined the steps taken to realize the four-layered vision of delete-compliant data systems presented in Figure 1, however, there are still open questions and challenges for such systems which pose opportunities for further innovative systems research.

Device-level Deletion. Data management solutions rely on storage devices and treat them as black boxes. However, deleting data persistently at the device level and from data archives is an open technological challenge. Current endeavors in this direction are mainly focused on encryption-based solutions [54, 57, 60]. Nevertheless, retention-based deletes entail persistent deletion of a "quantum" of data (e.g., the data ingested in a day) posing the following challenges for encryption-based solutions. First, it is hard to *determine the encryption granularity* while minimizing the encrypt/decrypt overhead. Second, with several data streams for different users/applications (and thus, bound by different SLAs), it is hard to *manage the encryption keys efficiently and in a scalable manner*. Third, efficient and scalable *deletion from archives and backup stores on-demand* is hard to be supported by encryption-based deletion as the encrypt/decrypt cost and the fine encryption granularity adds prohibitive overheads. Finally, from a legislation point-of-view it is not yet clear whether encrypting and discarding the key is an accepted form of deletion.

When considering a system-level deletion similar approach to the one presented in Section 3.3 storage devices are essentially one more level of managing data at the physical layer and similar approaches have to be implemented in the file system or the file and data systems have to be developed in tandem.

Cloud-Level Deletion. Further, operating on the cloud, data systems use virtualized devices and object storage which is even more abstract hiding the details of how the low-level device and page management is taking place. Offering guarantees for timely data deletion in virtualized storage will require a similar multi-layered approach where the file system and the device firmware will expose knobs to allow the application on top to request specific page reclamation properties.

Deletion in Distributed/Federated Computing Environment. With more and more data stores being transformed to cloud-based stores, user data may be collected, processed, and stored across multiple domains, spread across different geographic locations [75]. With different geographic locations being bound by different privacy regulations, we need to design *solutions to ensure consistency for persistent deletion* of user data. Our intuition is that existing solutions for data stream-tainting [37], cross-domain data tracing [33, 47], and related data provenance solutions [20, 45] can be useful to address this problem.

Compliance Demonstration. Last but certainly not least, data systems have to be able to prove compliance when audited. The natural way to do so now is via log auditing, however, a more light-weight algorithmic way for providing this will benefit both systems and users. Inspecting logs and the underlying data is a time-consuming process and the long-term goal of the community should be to design system-level tools that can verifiably prove compliance with the privacy regulations. One interesting development in this direction is the evolution of security-driven operating systems, such as seL4 [56, 80]. Another approach that can be taken is to show that the codebase of a data system has the necessary code-paths for timely deletion via static and dynamic analysis. An open challenge is to develop static and dynamic analysis tools that can prove that a system deletes data respecting the timely deletion requirements set.

5 Conclusion

In this paper, we highlight that the recently enacted regulations mandate new data deletion requirements, requiring a new breed of data systems to support them. We show that existing state-of-the-art out-of-place systems are ill-equipped for this task, and we present a four-layered approach towards building the necessary infrastructure. We present recent work on that front, and we conclude by discussing several open research challenges.

Acknowledgments. This work was partially funded by National Science Foundation under Grant No. IIS-1850202 and a Facebook Faculty Research Award.

References

[1] Right to erasure. *https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/individual-rights/right-to-erasure/*.

[2] Regulation (EU) 2016/679 of the European Parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. *Official Journal of the European Union (Legislative Acts)*, pages L119/1 – L119/88, 2016.

[3] California Consumer Privacy Act. *Assembly Bill No. 375, Chapter 55*, 2018.

[4] Data Protection Act 2018. *https://www.legislation.gov.uk/ukpga/2018/12/pdfs/ukpga_20180012_en.pdf*, 2018.

[5] PIPEDA in brief. *https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/pipeda_brief/*, 2019.

[6] The California Privacy Rights Act of 2020. *https://thecpra.org/*, 2020.

[7] Virginia Consumer Data Protection Act. *https://www.sullcrom.com/files/upload/SC-Publication-Virginia-Second-State-Enact-Privacy-Legislation.pdf*, 2021.

[8] Amazon. Redshift. *https://aws.amazon.com/redshift/*.

[9] M. L. Ambrose and J. Ausloos. The Right to Be Forgotten Across the Pond. *Journal of Information Policy*, 3:1–23, 2013.

[10] Apache. Accumulo. *https://accumulo.apache.org/*.

[11] Apache. HBase. *http://hbase.apache.org/*.

[12] Apache. Cassandra. *http://cassandra.apache.org*, 2021.

[13] M. Athanassoulis, S. Chen, A. Ailamaki, P. B. Gibbons, and R. Stoica. MaSM: Efficient Online Updates in Data Warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 865–876, 2011.

[14] M. Athanassoulis, S. Chen, A. Ailamaki, P. B. Gibbons, and R. Stoica. Online Updates on Data Warehouses via Judicious Use of Solid-State Storage. *ACM Transactions on Database Systems (TODS)*, 40(1), 2015.

[15] M. Athanassoulis, M. S. Kester, L. M. Maas, R. Stoica, S. Idreos, A. Ailamaki, and M. Callaghan. Designing Access Methods: The RUM Conjecture. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 461–466, 2016.

[16] M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-Oblivious B-Trees. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 399–409, 2000.

[17] M. A. Bender, M. Farach-Colton, W. Jannen, R. Johnson, B. C. Kuszmaul, D. E. Porter, J. Yuan, and Y. Zhan. An Introduction to Bϵ-trees and Write-Optimization. *White Paper*, 2015.

[18] C. T. Brown and T. D. Manoranjan. South Korea Releases Guidance on Right to Be Forgotten. *https://www.lexology.com/library/detail.aspx?g=21be3837-0c43-4047-b8b5-9e863960b0b9*, 2016.

[19] G. A. Brown. Consumers' "Right to Delete" under US State Privacy Laws. *https://www.securityprivacybytes.com/2021/03/consumers-right-to-delete-under-us-state-privacy-laws/*, 2021.

[20] P. Buneman and W.-C. Tan. Data Provenance: What next? *SIGMOD Rec.*, 47(3):5–16, 2018.

[21] M. Callaghan. Deletes are fast and slow in an LSM. *http://smalldatum.blogspot.com/2020/01/deletes-are-fast-and-slow-in-lsm.html*, 2020.

[22] Z. Cao, S. Dong, S. Vemuri, and D. H. C. Du. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 209–223, 2020.

[23] E. L. Carter. Argentina's Right to be Forgotten. *Emory International Law Review*, 27(1), 2013.

[24] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 205–218, 2006.

[25] W. B. Chik. The Singapore Personal Data Protection Act and an assessment of future trends in data privacy reform. *Comput. Law Secur. Rev.*, 29(5):554–575, 2013.

[26] Cisco. Cisco Global Cloud Index: Forecast and Methodology, 2016–2021. *White Paper*, 2018.

[27] K. Cohn-Gordon, G. Damaskinos, D. Neto, J. Cordova, B. Reitz, B. Strahs, D. Obenshain, P. Pearce, I. Papagiannis, and A. Media. DELF: Safeguarding deletion correctness in Online Social Networks. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, 2020.

[28] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, A. W. Lee, A. Motivala, A. Q. Munir, S. Pelley, P. Povinec, G. Rahn, S. Triantafyllis, and P. Unterbrunner. The Snowflake Elastic Data Warehouse. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 215–226, 2016.

[29] Databricks. Online reference. *https://databricks.com/*.

[30] Databricks. Table deletes, updates, and merges. *https://docs.databricks.com/delta/delta-update.html#delete-from-a-table*, 2021.

[31] N. Dayan, M. Athanassoulis, and S. Idreos. Monkey: Optimal Navigable Key-Value Store. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79–94, 2017.

[32] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-value Store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[33] B. Demsky. Cross-application data provenance and policy enforcement. *ACM Transactions on Information Systems (TOIS)*, 14(1):6:1—-6:22, 2011.

[34] F. Deng, Q. Cao, S. Wang, S. Liu, J. Yao, Y. Dong, and P. Yang. SeRW: Adaptively Separating Read and Write upon SSDs of Hybrid Storage Server in Clouds. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 76:1—-76:11, 2020.

[35] A. Deshpande and A. Machanavajjhala. ACM SIGMOD Blog: Privacy Challenges in the Post-GDPR World: A Data Management Perspective. *http://wp.sigmod.org/?p=2554*, 2018.

[36] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum. Optimizing Space Amplification in RocksDB. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017.

[37] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. D. McDaniel, and A. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 393–407, 2010.

[38] Facebook. RocksDB. *https://github.com/facebook/rocksdb*, 2021.

[39] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Engineering Bulletin*, 35(1):28–33, 2012.

[40] Gartner. Gartner Says 8.4 Billion Connected “Things" Will Be in Use in 2017, Up 31 Percent From 2016. https://tinyurl.com/Gartner2020, 2017.

[41] M. Goddard. The EU General Data Protection Regulation (GDPR): European Regulation that has a Global Impact. *International Journal of Market Research*, 59(6):703–705, 2017.

[42] G. Golan-Gueta, E. Bortnikov, E. Hillel, and I. Keidar. Scaling Concurrent Log-Structured Data Stores. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, pages 32:1–32:14, 2015.

[43] Google. LevelDB. *https://github.com/google/leveldb/*, 2021.

[44] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan. Amazon Redshift and the Case for Simpler Data Warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1917–1923, 2015.

[45] S. S. Hasan, N. H. Sultan, and F. A. Barbhuiya. Cloud Data Provenance using IPFS and Blockchain Technology. In *Proceedings of the International Workshop on Security in Cloud Computing (SCC)*, pages 5–12, 2019.

[46] S. Héman, M. Zukowski, and N. J. Nes. Positional Update Handling in Column Stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 543–554, 2010.

[47] R. Herbster, S. DellaTorre, P. Druschel, and B. Bhattacharjee. Privacy Capsules: Preventing Information Leaks by Mobile Apps. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys 2016, Singapore, June 26-30, 2016*, pages 399–411, 2016.

[48] G. Huang, X. Cheng, J. Wang, Y. Wang, D. He, T. Zhang, F. Li, S. Wang, W. Cao, and Q. Li. X-Engine: An Optimized Storage Engine for Large-scale E-commerce Transaction Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 651–665, 2019.

[49] S. Idreos, N. Dayan, W. Qin, M. Akmanalp, S. Hilgard, A. Ross, J. Lennon, V. Jain, H. Gupta, D. Li, and Z. Zhu. Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2019.

[50] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Engineering Bulletin*, 35(1):40–45, 2012.

[51] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, M. A. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter. BetrFS: A Right-optimized Write-optimized File System. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 301–315, 2015.

[52] M. L. Jones. It's About Time: Privacy, Information Lifecycles, and the Right to Be Forgotten. *Stanford Technology Law Review*, 16(2):54, 2012.

[53] W.-H. Kang, S.-W. Lee, and B. Moon. Flash as cache extension for online transactional workloads. *The VLDB Journal*, 25(5):673–694, 2016.

[54] R. Kissel, A. Regenscheid, M. Scholl, and K. Stine. Guidelines for Media Sanitization. *NIST Special Publication 800-88*, 2014.

[55] P. Kittane, I. S. Charles, A. Kamath, and G. Gokhale. Privacy and Data Protection – India Wrap 2020. *The National Law Review*, XI(162), 2021.

[56] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an operating-system kernel. *Communications of the ACM*, 53(6):107–115, 2010.

[57] B. Köppel and S. Neuhaus. Analysis of a hardware security module's high-availability setting. *IEEE Security Privacy*, 11(3):77–80, 2013.

[58] B. C. Kuszmaul. A Comparison of Fractal Trees to Log-Structured Merge (LSM) Trees. *Tokutek White Paper*, 2014.

[59] A. Lamb, M. Fuller, and R. Varadarajan. The Vertica Analytic Database: C-Store 7 Years Later. *Proceedings of the VLDB Endowment*, 5(12):1790–1801, 2012.

[60] B. Li and D. H. C. Du. TASecure: Temperature-Aware Secure Deletion Scheme for Solid State Drives. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 275–278, 2019.

[61] LinkedIn. Voldemort. *http://www.project-voldemort.com*.

[62] C. Luo and M. J. Carey. LSM-based Storage Techniques: A Survey. *The VLDB Journal*, 29(1):393–418, 2020.

[63] A. Madan and A. Kryczka. DeleteRange: A New Native RocksDB Operation. *https://rocksdb.org/blog/2018/11/21/delete-range.html*, 2018.

[64] R. McKean, E. Kurowska-Tober, and H. Waem. DLA Piper GDPR fines and data breach survey: January 2022. *https://www.dlapiper.com/en/us/insights/publications/2022/1/dla-piper-gdpr-fines-and-data-breach-survey-2022/*, 2022.

[65] M. Minaei, M. Mondal, P. Loiseau, K. P. Gummadi, and A. Kate. Lethe: Conceal Content Deletion from Persistent Observers. *Proceedings on Privacy Enhancing Technologies (PoPET)*, 2019(1):206–226, 2019.

[66] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.

[67] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson. The TileDB Array Data Storage Manager. *Proceedings of the VLDB Endowment*, 10(4):349–360, 2016.

[68] Paradigm4. Online reference. *https://www.paradigm4.com/*.

[69] D. Pardo. First Decision on the "Right to be Forgotten" in Argentina. *https://scholarlycommons.law.emory.edu/cgi/viewcontent.cgi?article=1097&context=eilr*, 2020.

[70] D. Piper. GDPR Data Breach Survey 2020. *https://www.dlapiper.com/en/us/insights/publications/2020/01/gdpr-data-breach-survey-2020/*, 2020.

[71] M. Sadoghi, K. A. Ross, M. Canim, and B. Bhattacharjee. Exploiting SSDs in operational multiversion databases. *The VLDB Journal*, 25(5):651–672, 2016.

[72] S. Sarkar, K. Chen, Z. Zhu, and M. Athanassoulis. Compactionary: A Dictionary for LSM Compactions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2022.

[73] S. Sarkar and M. Athanassoulis. Dissecting, Designing, and Optimizing LSM-based Data Stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2022.

[74] S. Sarkar and M. Athanassoulis. Query Language Support for Timely Data Deletion. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2022.

[75] S. Sarkar, J.-P. Banâtre, L. Rilling, and C. Morin. Towards Enforcement of the EU GDPR: Enabling Data Erasure. In *Proceedings of the IEEE International Conference of Internet of Things (iThings)*, pages 1–8, 2018.

[76] S. Sarkar, T. I. Papon, D. Staratzis, and M. Athanassoulis. Lethe: A Tunable Delete-Aware LSM Engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 893–908, 2020.

[77] S. Sarkar, D. Staratzis, Z. Zhu, and M. Athanassoulis. Constructing and Analyzing the LSM Compaction Design Space. *Proceedings of the VLDB Endowment*, 14(11):2216–2229, 2021.

[78] M. Schwarzkopf, E. Kohler, M. F. Kaashoek, and R. T. Morris. Position: GDPR Compliance by Construction. In *Selected Papers from VLDB Workshop on Polystore Systems for Heterogeneous Data in Multiple Databases with Privacy and Security Assurances (POLY)*, volume 11721 of *Lecture Notes in Computer Science*, pages 39–53, 2019.

[79] R. Sears and R. Ramakrishnan. bLSM: A General Purpose Log Structured Merge Tree. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 217–228, 2012.

[80] SeL4. Online reference. *https://sel4.systems*.

[81] A. Shah, V. Banakar, S. Shastri, M. Wasserman, and V. Chidambaram. Analyzing the Impact of GDPR on Storage Systems. In *Proceedings of the USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*, 2019.

[82] S. Shastri, V. Banakar, M. Wasserman, A. Kumar, and V. Chidambaram. Understanding and Benchmarking the Impact of GDPR on Database Systems. *Proceedings of the VLDB Endowment*, 13(7):1064–1077, 2020.

[83] S. Shastri, M. Wasserman, and V. Chidambaram. The Seven Sins of Personal-Data Processing Systems under GDPR. In *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[84] S. Shastri, M. Wasserman, and V. Chidambaram. GDPR anti-patterns. *Communications of the ACM*, 64(2):59–65, 2021.

[85] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. R. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-Store: A Column-oriented DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 553–564, 2005.

[86] M. Stonebraker, J. Duggan, L. Battle, and O. Papaemmanouil. SciDB DBMS Research at M.I.T. *IEEE Data Eng. Bull.*, 36(4):21–30, 2013.

[87] Tessian. 25 Biggest GDPR Fines So Far (2019, 2020, 2021, 2022). *https://www.tessian.com/blog/biggest-gdpr-fines-2020/*, 2022.

[88] TileDB. Online reference. *https://tiledb.io*.

[89] A. Tsesis. The Right to be Forgotten and Erasure: Privacy, Data Brokers, and the Indefinite Retention of Data. *Wake Forest Law Review*, 48:51, 2014.

[90] Z. Whittaker and N. Lomas. Even years later, Twitter doesn't delete your direct messages. *https://techcrunch.com/2019/02/15/twitter-direct-messages/*, 2019.

[91] L. Yang, H. Wu, T. Zhang, X. Cheng, F. Li, L. Zou, Y. Wang, R. Chen, J. Wang, and G. Huang. Leaper: A Learned Prefetcher for Cache Invalidation in LSM-tree based Storage Engines. *Proceedings of the VLDB Endowment*, 13(11):1976–1989, 2020.

[92] Q. Zheng, C. D. Cranor, D. Guo, G. R. Ganger, G. Amvrosiadis, G. A. Gibson, B. W. Settlemyer, G. Grider, and F. Guo. Scaling Embedded In-Situ Indexing with DeltaFS. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 3:1–3:15, 2018.