Toward Building Deletion Compliant Data Systems*

Subhadeep Sarkar, Boston University

The Out-of-Place Paradigm. Modern data store designs are driven by three fundamental trends. (A) Applications need to store increasingly more data, fueled by the decreasing price of storage. (B) The data boom, in turn, has driven data stores to ingest large volumes of data with sustained high throughput. (C) Retain and analyze a growing amount of data to extract as much meaningful information as possible. This has led to the birth and wide-spread adoption of data stores that are based on the out-of-place paradigm. Out-of-place systems (i) offer high ingestion throughput (ii) without interfering with the query path, and they do so (iii) at the cost of a larger memory/storage footprint [1, 4, 5].

What is the Problem? By design, out-of-place systems realize updates and deletes through ingestion of new entries that logically invalidate the target entries (instead of modifying them in place). The logically invalidated entries are retained in the system, and with memory/storage costs decreasing over time, most of these systems are built with the underlying assumption of *perpetual data retention*. Eventually, the logically invalid data may be physically removed from a data store during periodic consolidation of a database; but, there is no theoretical upper-bound on how long such invalid data may live within a data store. This has severe implications on the privacy front, especially, in light of the newly enforced privacy regulations. The GDPR's *right to be forgotten*, *right to delete* in California's CCPA and CPRA, and *deletion right* in Virginia's VCDPA are all aimed toward the common goal of enabling the users to express their preferences about the deletion of their data [7, 10]. To demonstrate compliance, service providers must **persistently delete** all copies of the data under deletion **in a time-bound manner**. Now, state-of-the-art **out-of-place systems are unable to provide any latency guarantees on persistent deletion of data**. In fact, in principle, *the design goals of out-of-place systems are antithetical to those of ensuring timely delete persistence* [9].

So How are so Many Systems Living On? In practice, companies strive to demonstrate regulation compliance to the best of their abilities, and they do so, by periodically (typically, every few weeks) going over the entire data collection, and physically purging all logically invalidated data [3, 8]. This comes at a remarkably high cost pertaining to accessing and processing large volumes of data, which also leads to undesired latency spikes.

Our Vision. Our vision is to enable **privacy-through-deletion** as a design element in modern data systems empowering them with the ability to ensure timely and persistent deletion of data, and we envision to do so without hurting performance or increasing operational cost.

The Steps We Took. We began by identifying the two classes of deletion requests that systems need to support, based on the regulatory requirements [2, 6]. (A) **On-demand deletion**, where the user submits an ad-hoc deletion request, and the service providers must persistently delete the data within a threshold duration. (B) **Retention-driven deletes**, where the user sets a retention duration, and any data older than that, must be persistently deleted from the data stores. Below, we discuss how we can efficiently support the two classes of deletes in out-of-place systems.

To realize *on-demand deletes timely*, every meaningful block of entries is assigned a timestamp, which is periodically monitored for health. We then piggyback the deletion job with the data store's own consolidation routine (i.e., compaction for LSM-trees, node split/merge for B-trees, etc.), and based on the health of a data block's timestamp, prioritize a block for consolidation [8]. This way, we persist logical deletes in a timely manner without adding any significant performance overheads. To facilitate *retention-driven deletes* efficiently, we designed a new weaved data layout between the (primary) sorting attribute and the (secondary) delete attribute. The benefit of this weaved data layout is that in the case of retention-driven deletes, we can discard entire data blocks at a time, signaling the file system to reclaim this page instantly, essentially converting the delete action to a page reclamation action that has very low latency compared to a full database consolidation [8]. Further, to translate the user deletion requirements into system-interpretable form, we extend SQL by augmenting (i) every `INSERT` with a *retention duration*, indicating the duration after which the entry must be persistently removed from the system, and (ii) every `DELETE` with a *delete persistence threshold*, indicating the upper-bound in time by which it must be persisted [6].

The Steps Moving Forward. While our efforts allow us to establish an end-to-end framework to build **deletion compliant data systems**, there are still several research challenges and open questions. For example, how do we enforce persistent deletion policies at the file system level and persistent and timely deletion of data from views and logs? Most cloud-based data systems use virtualized devices and object storage which abstract out the details of how the low-level device and page management strategies – how do we provide timely deletion guarantees in such frameworks? In distributed/federated computing environments, how do we manage different regulatory requirements across

*This work is accepted for presentation in the biennial invitational International Workshop on High Performance Transaction Systems (HPTS).

different geographical domains, to ensure timely deletion of all data replicas? Finally, can we include compliance demonstration as a primitive for systems design (does SeL4 hold the key to this)?

References

[1] 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.

[2] M. Athanassoulis, S. Sarkar, T. I. Papon, Z. Zhu, and D. Staratzis. Building Deletion-Compliant Data Systems. In *IEEE Data Engineering Bulletin*, 2022.

[3] 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.

[4] P. Helland. Immutability Changes Everything. *Communications of the ACM*, 59(1):64–70, 2016.

[5] 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.

[6] S. Sarkar and M. Athanassoulis. Query Language Support for Timely Data Deletion. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 429–434, 2022.

[7] 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.

[8] 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.

[9] 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.

[10] 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.