Who cares about HTAP?

Tianyu Li
MIT CSAIL
litianyu@mit.edu

Hybrid Transactional/Analytical Processing (HTAP) has been something of a holy grail for database practitioners in the past decade – building one system that excels at both workloads enables organizations to run realtime analysis on their transactional data and gain valuable and timely insights. However, building an HTAP system is also notoriously hard. Part of this is due to the engineering challenges of building such a complex system; additionally, HTAP operators may find it difficult to properly provision resources for the two parts of the system and limit interference. Therefore, HTAP still arguably remains the "emerging architecture" it was a decade ago, with limited commercial success.

I believe this is because HTAP is a rather poorly motivated problem. The community is working on devising solutions for running transactions and analytics on the same system, because they (correctly) believe that doing so will enable realtime analytics. Firstly, this does not mean that to enable realtime analytics, one must run transactions and analytics on the same system. Secondly, "realtime" in the requirement is a statement about the latency between realworld events and derived, actionable insight, rather than the staleness of data items. Hence, running a lengthy, 10 min query on a snapshot that is 5 min scale is more "realtime" than taking 20 min to run the same query on the latest snapshot. Consider, then, a decidedly non-HTAP setup, where an OLTP engine continuously streams updates into an analytical engine. As the streaming latency and freshness approaches 0, this setup infinitely approaches an HTAP system in its ability to answer realtime queries, and may even exceed HTAP system due to their more sophisticated and specialized implementations. Recent announcement of various zero-ETL schemes by cloud vendors (e.g., between Amazon Aurora and Redshift) bring this vision closer than ever to reality. I believe it is time to start sketching a post-HTAP research agenda – what should realtime analytical systems look like, using fast and realtime data streaming as the fundamental building block?

From first principles, a realtime data analysis pipeline can be thought of as a stream of data flowing from event sources and users, passing through various processing stages as value is extracted. Each system along such stream can be thought of as a "pond" that has limited capacity to retain timely data, in some prepared format for consumption (e.g., Arrow or Parquet). For example, the first such pond may be an OLTP system that retains transitionally relevant hot data, connecting to a warehouse where dashboard queries are run, and eventually into a data lake (what is a lake if not a large pond?). Multiple streams can lead into one pond, and a pond can flow out in multiple streams. Each data pond can be specialized in different ways and allow for different querying and accessing patterns, in a cloud-native fashion similar to lakehouses. Under this architecture, a user will select a topology for their data flow by specifying a collection of ponds and streams, and attach processing systems to each pond to begin extracting value. Many interesting architectures can be expressed as such connected "pondhouses". For example, Amazon Aurora can be thought of as two pondhouses: one OLTP engine that directly processes the stream of user events into in-memory pond of row data and emits a stream of log records, and a second, "cloud-native persistence" pondhouse that materializes a pond of replicated, materialized pages.

The idea of pondhouses excite me because it enables a number of interesting research directions. Lineage tracking becomes explicit in such an architecture. One might imagine that the state of each pond is uniquely identified by the offsets they have seen across all their input streams, which is then used to tag entries from their output streams and any query results from that state. This vector-clock like tracking allows easy management of consistency and help enforce policies around data protection. Pondhouses are reusable components that can be used to quickly assemble new systems. For example, one might create a new cloud-native Aurora-like DBMS just by writing a new in-memory transactional engine as a pondhouse, define a log format translator, and connect it to the cloud-native persistence pondhouse from Aurora without modifying their codebase or Aurora's.

To sum up, who cares about HTAP systems? Just like how lakehouses can replace data warehouses, the future of realtime analytics and data processing lies with ponds and streams.