Hybrid Transactional and Analytical Processing

Jana Giceva, Mohammad Sadoghi

Synonyms

- *Hybrid Transactional and Analytical Processing*
- *HTAP*
- *Operational Analytics*
- *Transactional Analytics*

Definitions

Hybrid Transactional and Analytical Processing (HTAP) refers to system architectures and techniques that enable modern database management systems (DBMS) to perform real-time analytics on data that is ingested and modified in the transactional database engine. It is a term that was originally coined by Gartner where Pezzini et al (2014) highlight the need of enterprises to close the gap between analytics and action for better business agility and trend awareness.

Overview

The goal of running transactions and analytics on the same data has been around for decades, but has not fully been realized due to technology limitations. Today, businesses can no longer afford to miss the real-time insights from data that is in their transactional system as they may lose competitive edge unless business decisions are made on *latest data*[1] or *fresh data*[2]. As a result, in recent years in both academia and industry there has been an effort to address this problem by designing techniques that combine the transactional and analytical capabilities and integrate

[1] Analytics on *latest data* implies allowing the query to run on any desired level of isolations including dirty read, committed read, snapshot read, repeatable read, or serializable.

[2] Analytics on *fresh data* implies running queries on a recent snapshot of data that may not necessarily be the latest possible snapshot when the query execution began or a consistent snapshot.

them in a single Hybrid Transactional and Analytical (HTAP) system.

Online transaction processing (OLTP) systems are optimized for write-intensive workloads. OLTP systems employ data structures that are designed for high volume of point access queries with a goal to maximize throughput and minimize latency. Transactional DBMSs typically store data in a row format, relying on indexes and efficient mechanisms for concurrency control.

Online analytical processing (OLAP) systems are optimized for heavy read-only queries that touch large amounts of data. The data structures used are optimized for storing and accessing large volumes of data to be transferred between the storage layer (disk or memory) and the processing layer (e.g., CPUs, GPUs, FPGAs). Analytical DBMSs store data in column stores with fast scanning capability that is gradually eliminating the need for maintaining indexes. Furthermore to keep high performance and to avoid the overhead of concurrency, these systems only batch updates at predetermined intervals. This, however, limits the data freshness visible to analytical queries.

Therefore, given the distinct properties of transactional data and (stale) analytical data, most enterprises have opted for a solution that separates the management of transactional and analytical data. In such a setup, analytics is performed as part of a specialized decision support system (DSS) in isolated data warehouses. The DSS executes complex long running queries on data at rest and the updates from the transactional database are propagated via an expensive and slow extract-transform-load (ETL) process. The ETL process transforms data from transactional-friendly to analytics-friendly layout, indexes the data and materializes selected pre-aggregations. Today's industry requirements are in conflict with such a design. Applications want to interface with fewer systems, and try to avoid the burden of moving the transactional data with the expensive ETL process to the analytical warehouse. Furthermore, systems try to reduce the amount of data replication and the cost that it brings. More importantly, enterprises want to improve data freshness, and perform analytics on operational data. Ideally, systems should enable applications to immediately react on facts and trends learned by posing an analytical query within the same transactional request. In short, the choice of separating OLTP and OLAP is becoming obsolete in light of exponential increase in data volume and velocity and the necessity to enrich the enterprise to operate based on real-time insights.

In the rest of this entry, we discuss design decisions, key research findings, and open questions in the database community revolving around exciting and imminent HTAP challenges.

Key Research Findings

Among existing HTAP solutions, we differentiate between two main design paradigms: (1) operating on a single data representation (e.g., row or column store format) vs. multiple representations of data (e.g., both row and column store formats), and (2) operating on a single copy of data (i.e., single data replica) vs. maintaining multiple copies of data (through a variety of data replication techniques).

Examples systems that operate on a single data representation are SAP HANA by Sikka et al (2012), HyPer by Neumann et al (2015), L-Store by Sadoghi et al (2016a,b, 2014, 2013). They provide a unified data representation and enable users to analyze the latest (not just fresh data) or any version of data. Here we must note that operating on a single data representation does not necessarily mean avoiding the need for data replication. For instance, the initial version of HyPer relied on a copy-on-write mechanism from the underlying operating system while SAP HANA keeps the data in a main and a delta, which contains the most recent updates still not merged with the main. L-Store maintains the history of changes to the data (the delta) alongside the lazily maintained latest copy of data in a unified representation form.

Other systems follow the more classical approach of data warehousing, where multiple data representations are exploited, either within a single instance of the data or by dedicating a separate replica for different workloads. Example systems are SQL Server's column-store index proposed by Larson et al (2015), Oracle's in-memory dual-format by Lahiri et al (2015), SAP HANA SOE's architecture described by Goel et al (2015), and the work on fractured mirrors by Ramamurthy et al (2002), to name a few.

Unified Data Representation

The first main challenge for single data representation HTAP systems is choosing (or designing) a suitable data storage format for hybrid workloads. As noted earlier, there are many systems whose storage format was optimized to support efficient execution of a particular class of workloads. Two prominent example formats are row-store (NSM) and column-store (DSM), but researchers have also shown the benefit of alternative stores like *partition attribute across* (PAX) proposed by Ailamaki et al (2001) that strike a balance between the two extremes. OLTP engines typically use a row-store format with highly tuned data structures (e.g., lock-free skip lists, latch-free BW trees), which provide low latencies and high throughput for operational workloads. OLAP engines have been repeatedly shown to perform better if data is stored in column-stores. Column stores provide better support for data compression and are more optimized for in-memory processing of large volumes of data. Thus, the primary focus was on developing new storage layouts that can efficiently support both OLTP and OLAP workloads.

The main challenge is the conflict in the properties of data structures suitable to handle large volume of update-heavy requests on the one hand and fast data scans on the other hand. Researchers have explored where to put the updates: whether to apply them in-place, append them in a log-structured format, or store them in a delta and periodically merge them with the main; how to best arrange the records: row-store, column-store, or a PAX format; how to handle multiple versions and when to do garbage collection?

Sadoghi et al (2016a,b, 2014, 2013) proposed L-Store (Lineage-based Data Store) that combines the real-time processing of transactional and analyt-

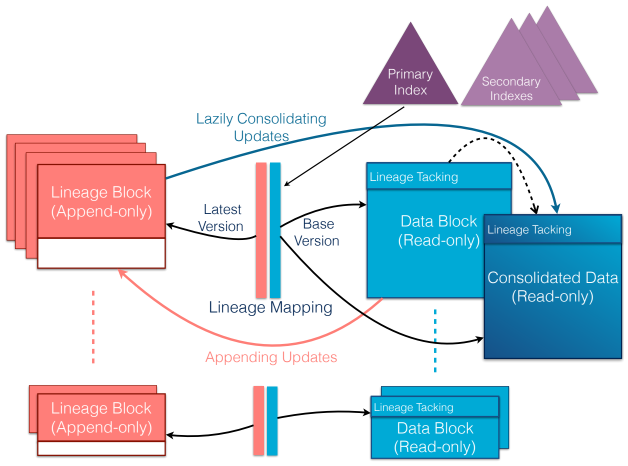

Fig. 1 Lineage-based Storage Architecture (LSA).

ical workloads within a single unified engine. L-Store bridges the gap between managing the data that is being updated at a high velocity and analyzing a large volume of data by introducing a novel update-friendly lineage-based storage architecture (LSA). This is achieved by developing a contention-free and lazy staging of data from write optimized into read optimized form in a transactionally consistent manner without the need to replicate data, to maintain multiple representation of data, or to develop multiple loosely integrated engines that limits real-time capabilities.

The basic design of LSA consists of two core ideas, as captured in Figure 1. First, the base data is kept in read-only blocks, where a block is an ordered set of objects of any type. The modification to the *data blocks* (also referred to as *base pages*) is accumulated in the corresponding *lineage blocks* (also referred to as *tail pages*). Second, a lineage mapping links an object in the data blocks to its recent updates in a lineage block. This essentially decouples the updates from the physical location of objects. Therefore, via the lineage mapping both the base and updated data are retrievable. A lazy, contention-free background process merges the recent updates with their corresponding read-only base data in order to construct new consolidated data blocks while data is continued being updated in the foreground without any interruption. The merge is necessary to ensure optimal performance for the analytical queries. Furthermore, each data block internally maintains the lineage of the updates consolidated thus far. By exploiting the decoupling and lineage tracking, the merge process, which only creates a new set of read-only consolidated data blocks, is carried out completely independently from update queries, which only append changes to lineage blocks and update the lineage mapping. Hence, there is no contention in the write paths of update queries and the merge process. That is a fundamental property necessary to build a highly-scalable distributed storage layer that is updatable.

Neumann et al (2015) proposed a novel MVCC implementation, which does update-in-place and stores prior versions as before-image deltas, which enables both an efficient scan execution and fine-grained serializability validation needed for fast processing of point access transactions. From the NoSQL side, Pilman et al (2017) demonstrated how scans can be efficiently implemented on a KeyValue store (KV-Store) enabling more complex analytics to be done on large and distributed KV-Stores. Wildfire from IBM by Barber et al (2016, 2017) uses Apache Parquet to support both analytical and transactional requests. IBM Wildfire is a variant of IBM DB2 BLU that is integrated into Apache Spark to support fast ingest of updates. The authors adopt the relaxed last-writer-wins semantics and offer an efficient snapshot isolation on recent view of data (but stale) by relying on periodic shipment and writing of the logs onto a distributed file system.

Manifold of Data Representations

An alternative approach comes from systems that propose using hybrid stores to keep the data in two or more layouts, either by partitioning the data based on the workload properties, or by doing partial replication. They exploit a manifold specialization techniques that are aligned with tenet “one size does not fit all” as explained by Stonebraker and Cetintemel (2005). Typically the hybrid mode consists of storing data in row and column formats by collocating attributes that are accessed together within a query. The main differentiators among the proposed solutions come by addressing the following challenges: How does a system determine which data to store in which layout—is it specified manually by the DB administrator, or is it derived by the system itself; Is the data layout format static or can it change over time? If the latter, which data is affected by the transformations—is it the hot or cold data?; Does the system support update propagation or is the data transformation only recommended for read-only data? If the former, when does the data transformation take place—is it incremental or immediate? Does it happen as part of query execution or is it done as part of a background process?

Grund et al (2010) with their HYRISE system propose the use of the *partially decomposed* storage model and automatically partition tables into vertical partitions of varying depth, depending on how the columns are accessed. Unfortunately, the bandwidth savings achieved with this storage model come with an increased CPU cost. A follow up work by Pirk et al (2013) improve the performance by combining the partially decomposed storage model with Just-in-Time (JiT) compilation of queries, which eliminates the CPU-inefficient function calls. Based on the work in HYRISE, Plattner (2009) in SAP developed HANA, where tuples start out in a row-store and are then migrated to a compressed column-store storage manager. Similarly, also MemSQL, Microsoft’s SQL Server, and Oracle support the two storage formats. One of the main differentiators among these systems is who decides on the partitioned layout: i.e., whether the administrator manually specifies which relations and attributes should be stored as a column store or the system derives it automatically. The latter can be achieved by monitoring and identifying hot vs. cold data or which attributes are accessed together in OLAP queries. There are a few research systems that have demonstrated the benefits of using adaptive data stores that can transform the data from one format to another with an evolving HTAP workload. Dittrich and Jindal (2011) show how to maintain multiple copies of the data in different layouts and use a logical log as a primary storage structure before creating a secondary physical representation from the log entries. In H2O, Alagiannis et al (2014) maintain the same data in different storage formats and improve performance of the read-only workload by leveraging multiple processing engines. The work by Arulraj et al (2016) performs data re-organization on cold data. The re-organization is done as part of an incremental background process that does not impact the latency-sensitive transactions, but improves the performance of the analytical queries.

Data Replication

Operating on a single data replica may come at a cost due to an identified trade off between degree of data freshness, system performance, and predictability of throughput and response time for a hybrid workload mix. Thus, the challenges for handling HTAP go beyond the layout in which the data is stored. A recent study by Psaroudakis et al (2014) shows that systems like SAP HANA and HyPer, which rely on partial replication, experience interference problems between the transactional and analytical workloads. The observed performance degradation is attributed to both resource sharing (physical cores and shared CPU caches) and synchronization overhead when querying and/or updating the latest data.

This observation motivated researchers to revisit the benefits of data replication and address the open challenge of efficient update propagation. One of the first approaches for addressing hybrid workloads was introduced by Ramamurthy et al (2002), where the authors proposed replicating the data and storing it in two data formats (row and columnar). The key advantage is that the data layouts and the associated data structures as well as the data processing techniques can be tailored to the requirements of the particular workload. Additionally, the system can make efficient use of the available hardware resources and their specialization. This is often viewed as a loose-form of an HTAP system. The main challenge for this approach is maintaining the OLAP-replica up-to-date and avoiding the expensive ETL process. BatchDB by Makreshanski et al (2017) relies on primary-secondary form of replication with the primary replica handling the OLTP workload and the updates being propagated to a secondary replica that handles the OLAP workload (Figure 2). BatchDB successfully addresses the problem of performance interference by spatially partitioning resources to the two data replicas and their execution engines (e.g., either by allocating a dedicated machine and connecting the replicas with RDMA over InfiniBand, or by allocating separate NUMA nodes within a multi-socket server machine). The system supports high performance

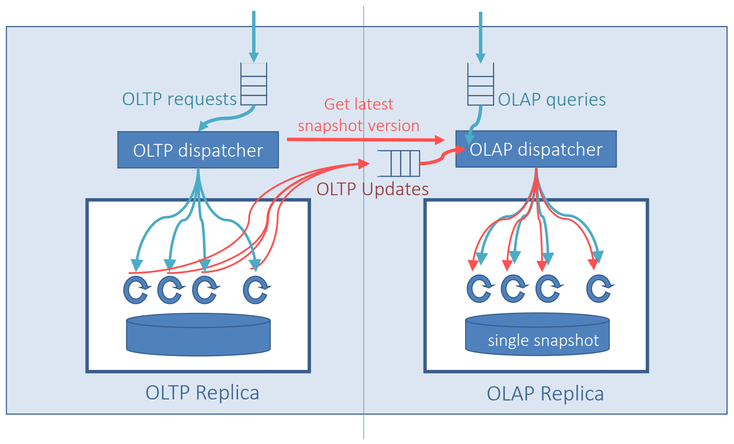

Fig. 2 Replication-based HTAP architecture

analytical query processing on fresh data with snapshot isolation guarantees. It achieves that by queuing queries and updates on the OLAP-side and scheduling them in batches, executing one-batch-at-a-time. BatchDB ensures that the OLAP queries see the latest snapshot of the data by using a lightweight propagation algorithm, which incurs a small overhead on the transactional engine to extract the updates and efficiently applies them on the analytical side at a faster rate than what has been recorded as a transactional throughput in literature so far. Such a system design solves the problem of high performance and isolation for hybrid workloads, and most importantly enables data analytics on fresh data. The main problem is that it limits the functionality of what an HTAP application may do with the data. For instance, it does not support interleaving of analytical queries within a transaction and does not allow analysts to go back in time and explore prior versions of the data to explain certain trends.

Oracle GoldenGate and SAP HANA SOE also leverage data replication and rely on specialized data processing engines for the different replicas. In GoldenGate, Oracle uses a publish/subscribe mechanism and propagates the updates from the operational database to the other replicas if they have subscribed to receive the updates-log.

Examples of Applications

There are many possible applications of HTAP for modern businesses. With today's dynamic data-driven world, real-time advanced analytics (e.g., planning, forecasting and what-if analysis) becomes an integral part of many business processes rather than a separate activity after the events have happened. One example is online retail. The transactional engine keeps track of data including the inventory list and products, the registered customers and manages all the purchases. An analyst can use this online data to understand customer behavior and come up with better strategies for product placement, optimized prising, discounts, personalized recommendations, as well as to identify products which are in high demand and do pro-active inventory refill. Another business use for HTAP comes from the financial industry sector. The transactional engine already supports millions of banking transactional per second and real time fraud detection and action can save billions of dollars. Yet another example of real-time detection and action is inspired by Content Delivery Networks (CDNs), where businesses need to monitor the network of web-servers that deliver and distribute the content at real time. An HTAP system can help identify Distributed Denial-of-Service (DDoS) attacks, find locations with spikes to redistribute the traffic bandwidth, get the top-*k* customers in terms of traffic usage and be able to act on it immediately.

Future Directions of Research

At time of writing this manuscript, we argue there is yet any system or technique that fulfill all HTAP promises to successfully interleave analytical processing within transactions. This is sometimes referred to as *true* or *in-process* HTAP, i.e., real-time analytics that is not limited to latest committed data and posed as a read-only request, but incorporates the analytics as part of the same write-enabled transaction request. Supporting *in-process* HTAP is an on-going effort in the research community and much sought-after feature for many enterprise systems.

Another open question is to investigate whether HTAP systems could and should support different type of analytical processing in addition to traditional OLAP analytics, e.g., the emergence of PolyStore by Duggan et al (2015). Training and inferring from machine learning models that describe fraudulent user behavior could supplement the existing knowledge of OLAP analytics. Furthermore, with the support of graph processing one can finally uncover fraudulent rings and other sophisticated scams that can be best identified using graph analytics queries. Therefore, it would be of great use if the graph analytical engine can process online data and be integrated in a richer and more expressive HTAP system (e.g., Hassan et al (2017)).

Finally, there is the question on how hardware heterogeneity could impact the design of HTAP systems. With the end of Moore's Law, the hardware landscape has been shifting towards specialization of computational units (e.g., GPUs, Xeon Phi, FPGAs, near-memory computing, TPUs) Najafi et al (2017, 2015); Teubner and Woods (2013). Hardware has always been an important game-changer for databases, and as in-memory processing enabled much of the technology for approaching true HTAP, there is discussion that the coming computing heterogeneity is going to significantly influence the design of future systems as highlighted by Appuswamy et al (2017).

References

Ailamaki A, DeWitt DJ, Hill MD, Skounakis M (2001) Weaving relations for cache performance. In: VLDB, pp 169–180

Alagiannis I, Idreos S, Ailamaki A (2014) H2O: a hands-free adaptive store. In: SIGMOD, pp 1103–1114

Appuswamy R, Karpathiotakis M, Porobic D, Ailamaki A (2017) The Case For Heterogeneous HTAP. In: In CIDR

Arulraj J, Pavlo A, Menon P (2016) Bridging the Archipelago Between Row-Stores and Column-Stores for Hybrid Workloads. SIGMOD, pp 583–598

Barber R, Huras M, Lohman G, Mohan C, Mueller R, Özcan F, Pirahesh H, Raman V, Sidle R, Sidorkin O, Storm A, Tian Y, Tözun P (2016) Wildfire: Concurrent Blazing Data Ingest and Analytics. SIGMOD '16, pp 2077–2080

Barber R, Garcia-Arellano C, Grosman R, Müller R, Raman V, Sidle R, Spilchen M, Storm AJ, Tian Y, Tözün P, Zilio DC, Huras M, Lohman GM, Mohan C, Özcan F, Pirahesh H (2017) Evolving Databases for New-Gen Big Data Applications. In: Online Proceedings of CIDR

Dittrich J, Jindal A (2011) Towards a One Size Fits All Database Architecture. In: CIDR

Duggan J, Elmore AJ, Stonebraker M, Balazinska M, Howe B, Kepner J, Madden S, Maier D, Mattson T, Zdonik S (2015) The BigDAWG Polystore System. SIGMOD Rec 44(2):11–16

Goel AK, Pound J, Auch N, Bumbulis P, MacLean S, Färber F, Gropengiesser F, Mathis C, Bodner T, Lehner W (2015) Towards Scalable Real-time Analytics: An Architecture for Scale-out of OLxP Workloads. PVLDB 8(12):1716–1727

Grund M, Krüger J, Plattner H, Zeier A, Cudré-Mauroux P, Madden S (2010) HYRISE - A Main Memory Hybrid Storage Engine. PVLDB pp 105–116

Hassan MS, Kuznetsova T, Jeong HC, Aref WG, Sadoghi M (2017) Empowering in-memory relational database engines with native graph processing. CoRR abs/1709.06715

Lahiri T, Chavan S, Colgan M, Das D, Ganesh A, Gleeson M, Hase S, Holloway A, Kamp J, Lee TH, Loaiza J, Macnaughton N, Marwah V, Mukherjee N, Mullick A, Muthulingam S, Raja V, Roth M, Soylemez E, Zait M (2015) Oracle Database In-Memory: A dual format inmemory database. In: ICDE, pp 1253–1258, DOI 10.1109/ICDE.2015.7113373

Larson PA, Birka A, Hanson EN, Huang W, Nowakiewicz M, Papadimos V (2015) Real-time Analytical Processing with SQL Server. PVLDB 8(12):1740–1751

Makreshanski D, Giceva J, Barthels C, Alonso G (2017) BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. SIGMOD '17, pp 37–50

Najafi M, Sadoghi M, Jacobsen H (2015) The FQP Vision: Flexible Query Processing on a Reconfigurable Computing Fabric. SIGMOD Record 44(2):5–10

Najafi M, Zhang K, Sadoghi M, Jacobsen H (2017) Hardware Acceleration Landscape for Distributed Real-Time Analytics: Virtues and Limitations. In: ICDCS, pp 1938–1948

Neumann T, Mühlbauer T, Kemper A (2015) Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In: SIGMOD, pp 677–689

Pezzini M, Feinberg D, Rayner N, Edjali R (2014) Hybrid Transaction/Analytical Porcessing Will Foster Opportunities for Dramatic Business Innovation. https://www.gartner.com/doc/2657815/hybrid-transactionanalytical-processing-foster-opportunities

Pilman M, Bocksrocker K, Braun L, Marroquín R, Kossmann D (2017) Fast Scans on Key-value Stores. PVLDB 10(11):1526–1537

Pirk H, Funke F, Grund M, Neumann T, Leser U, Manegold S, Kemper A, Kersten ML (2013) CPU and cache efficient management of memory-resident databases. In: ICDE, pp 14–25

Plattner H (2009) A Common Database Approach for OLTP and OLAP Using an In-memory Column Database. In: SIGMOD, pp 1–2

Psaroudakis I, Wolf F, May N, Neumann T, Böhm A, Ailamaki A, Sattler KU (2014) Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. In: TPCTC 2014, pp 97–112

Ramamurthy R, DeWitt DJ, Su Q (2002) A Case for Fractured Mirrors. VLDB '02, pp 430–441

Sadoghi M, Ross KA, Canim M, Bhattacharjee B (2013) Making updates disk-I/O friendly using SSDs. PVLDB 6(11):997–1008

Sadoghi M, Canim M, Bhattacharjee B, Nagel F, Ross KA (2014) Reducing database locking contention through multi-version concurrency. PVLDB 7(13):1331–1342

Sadoghi M, Bhattacherjee S, Bhattacharjee B, Canim M (2016a) L-store: A real-time OLTP and OLAP system. CoRR abs/1601.04084

Sadoghi M, Ross KA, Canim M, Bhattacharjee B (2016b) Exploiting SSDs in operational multiversion databases. VLDB J 25(5):651–672

Sikka V, Färber F, Lehner W, Cha SK, Peh T, Bornhövd C (2012) Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. SIGMOD '12, pp 731–742

Stonebraker M, Cetintemel U (2005) "One Size Fits All": An Idea Whose Time Has Come and Gone. ICDE, pp 2–11

Teubner J, Woods L (2013) Data Processing on FPGAs. Synthesis Lectures on Data Management, Morgan & Claypool Publishers