Hybrid Transactional/Analytical Processing: A Survey

Fatma Özcan
IBM Resarch - Almaden
fozcan@us.ibm.com

Yuanyuan Tian
IBM Research - Almaden
ytian@us.ibm.com

Pınar Tözün
IBM Research - Almaden
ptozun@us.ibm.com

ABSTRACT

The popularity of large-scale real-time analytics applications (real-time inventory/pricing, recommendations from mobile apps, fraud detection, risk analysis, IoT, etc.) keeps rising. These applications require distributed data management systems that can handle fast concurrent transactions (OLTP) and analytics on the recent data. Some of them even need running analytical queries (OLAP) as part of transactions. Efficient processing of individual transactional and analytical requests, however, leads to different optimizations and architectural decisions while building a data management system.

For the kind of data processing that requires both analytics and transactions, Gartner recently coined the term *Hybrid Transactional/Analytical Processing (HTAP)*. Many HTAP solutions are emerging both from the industry as well as academia that target these new applications. While some of these are single system solutions, others are a looser coupling of OLTP databases or NoSQL systems with analytical big data platforms, like Spark. The goal of this tutorial is to 1-) quickly review the historical progression of OLTP and OLAP systems, 2-) discuss the driving factors for HTAP, and finally 3-) provide a deep technical analysis of existing and emerging HTAP solutions, detailing their key architectural differences and trade-offs.

1. INTRODUCTION

In this tutorial, we plan to survey existing and emerging HTAP (Hybrid Transactions and Analytics Processing) solutions. HTAP is a term created by Gartner to describe systems that can support both OLTP (On-line transaction processing) as well as OLAP (on-line analytics processing) *within a single transaction.* However, the term HTAP is currently used more broadly, even for solutions that support insertions (not necessarily ACID transactions) as well as OLAP queries. Some of these systems have the ability to run analytical queries over the very recent data, while others need some delay before the queries see the latest data.

To understand HTAP, we first need to look into OLTP and OLAP systems and how they progressed over the years. Relational databases have been used for both transaction processing as well as analytics. However, OLTP and OLAP systems have very different characteristics. OLTP systems are identified by their individual record insert/delete/update statements, as well as point queries that benefit from indexes. One cannot think about OLTP systems without indexing support. OLAP systems, on the other hand, are updated in batches and usually require scans of the tables. Batch insertion into OLAP systems are an artifact of ETL (extract transform load) systems that consolidate and transform transactional data from OLTP systems into an OLAP environment for analysis.

After the seminal paper of Stonebraker [34] arguing for multiple specialized systems, the database field has seen an influx of specialized column-oriented OLAP systems, such as BLU [30], Vertica [23], ParAccel, GreenPlumDB, Vectorvise, as well as many in-memory OLTP systems, including VoltDB [35], Hekaton [13], MemSQL [24] among others. The main driver for this re-surgency in database engines is the advances in modern hardware. This second generation of OLAP and OLTP systems take better advantage of multi-core, various levels of memory caches, and large memories.

At the same time, the last decade seen an explosion of many big data technologies, driven by new generation applications. NoSQL or key-value stores, such as Voldemort [32], Cassandra [8], RocksDB [31], offer fast inserts and lookups, and very high scale out, but lack in their query capabilities, and offer only loose transactional guarantees (see Mohan's tutorial [25]). There have been also many SQL-on-Hadoop [10] offerings, including Hive [36], Big SQL [15], Impala [20], and Spark SQL [3], that provide analytics capabilities over large data sets, focusing on OLAP queries only, and lacking transaction support. Although all these systems support queries over text and CSV files, their focus have been on columnar storage formats, like ORCFile [27], and Parquet [1].

Recent years have seen the need for more real-time analytics. In addition, mobile and Internet of Things have given rise to a new generation of applications that are characterized by heavy ingest rates, i.e. they produce large amounts of data in a short time, as well as their need for more real-time analysis. Enterprises are pushing for more real-time analysis of their data to drive competitive advantage, and as such they need the ability to run analytics on their operational data as soon as possible.

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA
© 2017 ACM. ISBN 978-1-4503-4197-4/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3035918.3054784

With these developments, there is now a lot of interest, and research focus on providing HTAP solutions over big data sets. In this tutorial, we plan to provide a quick historical perspective into the progression of different technologies and discuss the current set of HTAP solutions. We will examine the different architectural aspects of the current solutions, identifying their strengths and weaknesses. We will categorize existing systems along many technological dimensions, and provide deep dives into a few representative systems. Finally, we will discuss existing research challenges to realize *true* HTAP, where a single transaction can contain both insert/update/delete statements, as well as complex OLAP queries.

2. HTAP SOLUTIONS: DESIGN OPTIONS

HTAP solutions today follow a variety of design practices. This part of the tutorial goes over them to highlight their main trade-offs while giving examples from industrial offerings and academic solutions.

One of the major design decisions HTAP systems have to make is whether or not to use the same engine for both OLTP and OLAP requests.

2.1 Single System for OLTP and OLAP

The traditional relational databases (e.g., DB2, Oracle, Microsoft SQL Server) have the ability to support OLTP and OLAP in one engine using single type of data organization (mainly row-stores). However, they are not very efficient for either of these workloads.

Therefore, following the *one size doesn't fit all* rule [34], the past decade has seen the rise of specialized engines for OLTP and OLAP exploiting the advances in modern hardware (larger main-memory, multicores, etc.). Various vendors and academic groups have built in-memory optimized row-stores (e.g., VoltDB [35], Hekaton [13], MemSQL [24], Silo [37], ...) and column-stores (e.g., MonetDB [7], Vertica [23], BLU [30], SAP HANA [14], ...) specialized for transactional and analytical processing, respectively. These systems departed from traditional code-bases of relational databases, and built leaner engines from scratch to also avoid the large instruction footprint of the traditional engines.

However, many of the systems optimized for one type of processing, later started adding support for the other type in order to support HTAP. These systems mainly differ based on the data organization they use for their transactional and analytical requests.

2.1.1 Using Separate Data Organization for OLTP and OLAP

SAP HANA [14] or Oracle's TimesTen [22] have engines that are mainly optimized for in-memory columnar processing, which is more beneficial for OLAP workloads. These systems also support ACID transactions. However, they use a different data organization for data ingestion (row-wise) and analytics (columnar).

Conversely, MemSQL [24] has an engine that was primarily designed for scalable in-memory OLTP, but today it supports fast analytical queries as well. It ingests the data in row format as well as keeping the in-memory portion of the data in row format. When data is written to disk, it is converted to columnar format for faster analytics. Similarly, IBM dashDB [11] is the evolution of a traditional row store into an HTAP system with hybrid row-wise and columnar data organizations for OLTP and OLAP workloads, respectively.

On the other hand, from the beginning, HyPer [19] was designed to support both fast transactions and analytics using one engine. Even though, initially it used row-wise processing of data for both OLTP and OLAP, today it also provides the option for choosing a columnar format to be able to run the analytical requests more efficiently.

Finally, the recent academic project Pelaton [28] aims to build an autonomous in-memory HTAP system. It provides adaptive data organization [4], which changes the data format at run-time based on the type of requests.

All these systems require converting the data between row and columnar formats for transactions and analytics. Due to these conversions, the latest committed data might not be available to the analytical queries right away for these types of systems.

2.1.2 Same Data Organization for both OLTP and OLAP

H^{2}TAP [2] is an academic project that aims to build an HTAP system focusing mainly on the hardware utilization of a single node when running on heterogeneous hardware. It falls under this category since the system is designed as a row-store.

Among the SQL-on-Hadoop systems, there has also been HTAP solutions that extend existing OLAP systems with the ability to update data. Hive, since version 0.13, has introduced the transaction support (insert, update, and delete) at the row level [18] for ORCFile, their columnar data format. However, the primary use cases are for updating dimension tables and streaming data ingest. The integration of Impala [20] with the storage manager Kudu [21], also allows the SQL-on-Hadoop engine to handle updates and deletes. The same Kudu storage is also used for running analytical queries.

Since these systems do not require conversion from one data organization to another in order to perform transactional and analytical requests, the OLAP queries can read the latest committed data. However, they might face the same shortcomings the traditional relational engines faced. They do not have a data organization that is optimal for both types of processing. Therefore, they may rely on batching of requests for fast transactions due to the overheads of processing data over a non-row-wise format, or perform sub-optimally for analytics due to non-columnar format.

2.2 Separate OLTP and OLAP Systems

The systems under this category can be further distinguished in the way they handle the underlying storage, i.e., whether they use the same storage for OLTP and OLAP.

2.2.1 Decoupling the Storage for OLTP and OLAP

Many applications loosely couple an OLTP and an OLAP systems together for HTAP. It is up to the applications to maintain the hybrid architecture. The operational data in the OLTP system are aged to the OLAP system using standard ETL process. In fact, this is very common in the big data world, where applications use a fast key-value store like Cassandra for transactional workloads, and the operational data are groomed into Parquet or ORC files on HDFS for a SQL-on-Hadoop system for queries. As a result, there is a lag between what data the OLAP system can query and what data the OLTP system sees.

2.2.2 *Using the Same Storage for OLTP and OLAP*

There are alternative HTAP offerings from some of the database vendors that combine their traditional products with the Spark ecosystem to enable large-scale HTAP. SAP HANA Vora [16] is such an example. In HANA Vora, the transactional processing is carried out through HANA, whereas analytical requests are handled by Spark SQL with subqueries pushed down to the database.

Several recently developed data management engines take a similar path as well. For example, SnappyData [26] uses the transactional engine GemFire for OLTP and takes advantage of the Spark ecosystem for OLAP.

Key-value stores, such as HBase [17] and Cassandra [8], are chosen by many modern applications as the online operational data store for fast updates. In order to bring in the missing OLAP capabilities, key-value stores are often used together with SQL-on-Hadoop engines. In one approach, both systems see exactly the same data that are stored in the key-value store. This requires SQL-on-Hadoop systems to directly query against data in key-value stores. Many extensions to existing SQL-on-Hadoop systems have been developed to enable such integration. Utilizing the Data Source API in Spark SQL similar to the systems above, the Spark HBase connector [38] and the Spark Cassandra connector [12] allow Spark SQL to directly query HBase and Cassandra data, respectively. The main problem with this approach is their slow performance. Running queries that scan large amounts of data, which is typical in an analytical query, is very slow via these connectors.

In another approach, many SQL-on-Hadoop systems, such as HIVE [36], Impala [20], IBM Big SQL [15], and Actian VectorH [9], use HBase as an updatable storage engine to store tables that need frequent updates. Users can send both operational and analytical requests to the same interface in SQL-on-Hadoop systems. Underneath, requests to the HBase tables are executed through HBase's processing engine.

Systems like Splice Machine [33] and Phoenix [29] provide a SQL layer on top of HBase. They also allow updates and transactions for data stored in HBase tables. As a result, they rely on HBase for the updates. Splice Machine even supports ACID transactions.

Both the SQL-on-Hadoop systems that access HBase tables directly, as well as Splice Machine and Phoenix run analytical queries slowly, as the scans over HBase tables are inefficient. HBase has been designed for fast insertion, and single-record lookups. Hence, none of these systems provide fast OLAP capabilities.

Wildfire [5] is a recent project from IBM Research that builds an HTAP engine where both analytical and transactional requests go over the same columnar data organization, which is Parquet [1]. By using a single data organization for both data ingestion as well as analytics, Wildfire enables the analysis on latest committed data right away. Wildfire also utilizes the Spark ecosystem to enable large-scale distributed analytics. The requests to Wildfire enters through Spark SQL and are pushed down to the Wildfire engine as much as possible.

Since the OLTP and OLAP components share the same underlying storage for the systems in this category, the latest committed transactions are immediately query-able for analytics.

3. ROAD TO TRUE HTAP

Before concluding our tutorial, we plan to highlight the challenges that HTAP system builders, as well as HTAP users still face today.

Even though there are many systems out there labeled as HTAP solutions (as Section 2 describes), none of them support *true HTAP*. Existing solutions indeed provide a decent platform for supporting both transactional and analytical requests when they are sent to the system separately. However, none of the existing solutions really support efficient processing of transactional and analytical request within the same transaction. To fully support HTAP, systems should allow analytics on recent data not only after the transaction that is ingesting or updating that data has committed, but also as part of the same request.

In addition, most HTAP solutions today use multiple components to provide all the desired capabilities. These different components are usually maintained by different groups of people. Therefore, keeping these components compatible and providing the end-users with the illusion of a single system is a challenging task.

Finally, indexing the data that is distributed, and accessed at a large-scale to enable efficient point lookups is not straight-forward. Moreover, most of these systems are deployed on public or private clouds, which use object stores as well as shared file systems like HDFS. Fine-grain indexing is needed to enable efficient point-lookups, and richer OLTP. Fast OLTP engines keep the index in memory, but in case of large scale data, and HTAP, those indexes cannot stay only in memory. One can cache the portion of the index for the most frequently accessed data and reduce the access cost to index entries. However, large-scale distributed OLAP systems use shared file systems and data organizations that are mainly optimized for scans, which does not provide fast access to individual records or columns. Faster point access to these shared files systems, and object stores is still an open problem.

4. TUTORIAL

Length: 1.5 hours

Target Audience:

1-) Researchers and developers who would like to learn the challenges HTAP poses while building systems, as well as the recent industry trends and offerings in the HTAP market,

2-) and PhD students who are seeking a high-impact research topic in this area.

Related Previous Tutorials:

This tutorial has not been presented in any other venue. Although, there has been recent tutorials on similar topics (e.g., [6]), this tutorial aims to focus on unique aspects of HTAP, and cover a broader set of offerings from the industry and academia while providing a list of design trade-offs.

Outline

- Introduction
  - Tutorial goal, audience, and schedule
  - Overview of traditional & specialized OLTP & OLAP systems

  - Driving factors for the rise of HTAP
- Deep-dive into HTAP solutions examining several dimensions including the following:
  - Query processing and ingestion engines
  - Storage options
  - Data organization
  - Transactional semantics
  - Recency of data being read by OLAP
  - Indexing support
- Research challenges on the road to true HTAP
- Summary & Conclusions

5. BIOGRAPHY

Fatma Özcan is a Research Staff Member and a manager at IBM Almaden Research Center. Her current research focuses on platforms and infra-structure for large-scale data analysis, storage and querying of knowledge graphs, SQL-on-Hadoop, and query processing and optimization of semi-structured data. Dr Özcan got her PhD degree in computer science from University of Maryland, College Park. She has over 15 years of experience in semi-structured and structured data management, query processing and optimization, and has delivered core technologies into IBM DB2 and BigInsights products. She is the co-author of the book "Heterogeneous Agent Systems", and co-author of several conference papers and patents. She has chaired program committees for various conferences, and served on NSF (National Science Foundation) panels. She is a member of the ACM and ACM SIGMOD, and serves on the board of trustees of the VLDB endowment.

Yuanyuan Tian is currently a Research Staff Member at IBM Almaden Research Center. She obtained her Ph.D. in computer science from the University of Michigan. Her research interests include HTAP, SQL-on-Hadoop, big data federation, graph analytics platforms, and large-scale systems for machine learning. She has published over 30 articles in top database venues. Dr. Tian has served in the editorial board for the new encyclopedia for Big Data, as an Associate Editor for PVLDB, and as a PC Chair for ICDE 2017 demo track and CIKM 2013 poster track. She has also served in several NSF panels. She is the recipient of the Distinguished Academic Achievement Award from the University of Michigan in 2008, and the Outstanding Technical Achievement Award from IBM Research in 2016.

Pınar Tözün is a research staff member at IBM Almaden Research Center. Before joining IBM, she received her PhD from École polytechnique fédérale de Lausanne (EPFL). Her research focuses on HTAP engines, performance characterization of database workloads, and scalability and efficiency of data management systems on modern hardware. She received a Jim Gray Doctoral Dissertation Award Honorable Mention in 2016. During her PhD, she also spent a summer in Oracle Labs (Redwood Shores, CA) as an intern. Before starting her PhD, she received her BSc degree in Computer Engineering department of Koç University in 2009.

6. REFERENCES

[1] Apache Parquet. https://parquet.apache.org/.

[2] R. Appuswarmy, M. Karpathiotakis, D. Porobic, and A. Ailamaki. The Case For Heterogeneous HTAP. In *CIDR*, 2017.

[3] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark SQL: Relational Data Processing in Spark. In *SIGMOD*, pages 1383–1394, 2015.

[4] J. Arulraj, A. Pavlo, and P. Menon. Bridging the Archipelago Between Row-Stores and Column-Stores for Hybrid Workloads. In *SIGMOD*, pages 583–598, 2016.

[5] R. Barber, C. Garcia-Arellano, R. Grosman, R. Mueller, V. Raman, R. Sidle, M. Spilchen, A. Storm, Y. Tian, P. Tözün, D. Zilio, M. Huras, G. Lohman, C. Mohan, F. Özcan, and H. Pirahesh. Evolving Databases for New-Gen Big Data Applications. In *CIDR*, 2017.

[6] A. Boehm, J. Dittrich, N. Mukherjee, I. Pandis, and R. Sen. Operational analytics data management systems. *PVLDB*, 9:1601–1604, 2016.

[7] P. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR*, 2005.

[8] Apache Cassandra. http://cassandra.apache.org.

[9] A. Costea, A. Ionescu, B. Răducanu, M. Switakowski, C. Bârca, J. Sompolski, A. Luszczak, M. Szafrański, G. de Nijs, and P. Boncz. Vectorh: Taking sql-on-hadoop to the next level. In *SIGMOD '16*, pages 1105–1117, 2016.

[10] Danial Abadi and Shivnath Babu and Fatma Özcan and Ippokratis Pandis. Tutorial: SQL-on-Hadoop Systems. *PVLDB*, 8, 2015.

[11] IBM dashDB. http://www.ibm.com/analytics/us/en/technology/cloud-data-services/dashdb.

[12] DataStax Spark Cassandra Connector. https://github.com/datastax/spark-cassandra-connector.

[13] C. Diaconu, C. Freedman, E. Ismert, P.-Å. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL Server's memory-optimized OLTP engine. In *SIGMOD*, pages 1243–1254, 2013.

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

[15] S. Gray, F. Özcan, H. Pereyra, B. van der Linden, and A. Zubiri. IBM Big SQL 3.0: SQL-on-Hadoop without compromise. http://public.dhe.ibm.com/common/ssi/ecm/en/sww14019usen/SWW14019USEN.PDF, 2014.

[16] SAP HANA Vora. http://go.sap.com/product/data-mgmt/hana-vora-hadoop.html.

[17] Apache HBase. https://hbase.apache.org/.

[18] Hive Transactions. http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.3.0/bk_dataintegration/content/hive-013-feature-transactions.html.

[19] A. Kemper and T. Neumann. HyPer – A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *ICDE*, pages 195–206, 2011.

[20] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A modern, open-source SQL engine for Hadoop. In *CIDR*, 2015.

[21] Apache Kudu. https://kudu.apache.org/.

[22] T. Lahiri, M.-A. Neimat, and S. Folkman. Oracle TimesTen: An In-Memory Database for Enterprise Applications. *IEEE DEBull*, 36(3):6–13, 2013.

[23] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The Vertica Analytic Database: C-store 7 Years Later. *PVLDB*, 5(12):1790–1801, 2012.

[24] MemSQL. http://www.memsql.com/.

[25] C. Mohan. History Repeats Itself: Sensible and NonsenSQL Aspects of the NoSQL Hoopla. In *EDBT*, 2013.

[26] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav. SnappyData: A Unified Cluster for Streaming, Transactions and Interactice Analytics. In *CIDR*, 2017.

[27] Apache ORC. https://orc.apache.org/.

[28] A. Pavlo, J. Arulraj, L. Ma, P. Menon, T. C. Mowry, M. Perron, A. Tomasic, D. V. Aken, Z. Wang, and T. Zhang. Self-Driving Database Management Systems. In *CIDR*, 2017.

[29] Apache Phoenix. http://phoenix.apache.org.

[30] V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Mueller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. Storm, and L. Zhang. DB2 with BLU Acceleration: So Much More than Just a Column Store. *PVLDB*, 6:1080–1091, 2013.

[31] RocksDB. http://rocksdb.org/.

[32] Roshan Sumbaly and others. Serving large-scale batch computed data with project Voldemort. In *Proc. of the 10th USENIX conference on File and Storage Technologies*, 2012.

[33] Splice Machine. http://www.splicemachine.com/.

[34] M. Stonebraker and U. Cetintemel. ”One Size Fits All”: An Idea Whose Time Has Come and Gone. In *ICDE*, pages 2–11, 2005.

[35] M. Stonebraker and A. Weisberg. The VoltDB Main Memory DBMS. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.

[36] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - A Petabyte Scale Data Warehouse Using Hadoop. In *ICDE*, 2010.

[37] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy Transactions in Multicore In-memory Databases. In *SOSP*, pages 18–32, 2013.

[38] Z. Zhang. Spark-on-HBase: Dataframe Based HBase Connector. http://hortonworks.com/blog/spark-hbase-dataframe-based-hbase-connector.