H-Store Paper for I524

KARTHICK VENKATESAN[1]

[1]School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*]Corresponding authors: laszewski@gmail.com

paper1, February 5, 2017

This paper provides information on the H-Store database technology.The paper is an attempt to provide details on the problems that it solves and its application in large scale data processing. It also provides details on the architecture and refferences details on research done to compare the database to current market OLTP databases

Keywords: H-Store,OLTP, I524

https://github.com/cloudmesh/classes/blob/master/docs/source/format/report/report.pdf

1. INTRODUCTION

H-Store is an experimental database management system (DBMS) designed for online transaction processing applications that was developed by a team of researchers at Brown University, Carnegie Mellon University, the Massachusetts Institute of Technology, and Yale University. The H-Store belongs to a new class of parallel database management systems, called NewSQL,that provide the high-throughput and high-availability of NoSQL systems, but without giving up the transactional guarantees of a traditional DBMS. The H-Store DBMS can seamlessly scale out horizontally across multiple machines to improve throughput.

As noted in [1] the current RDBMS systems were architected more than 25 years ago, when hardware characteristics were much different than today. Processors are thousands of times faster and memories are thousands of times larger. Disk volumes have increased enormously, making it possible to keep essentially everything, if one chooses to. This relentless pace of technology has still not resulted in a architecture of the current day DBMS systems.The current day DBMS include the following architectural features which are not fully utilising the technological advancements in the last 25 years

- □ Disk oriented storage and indexing structures
- □ Multithreading to hide latency
- □ Locking-based concurrency control mechanisms
- □ Log-based recovery

According to [1] the H-Store is a complete redesign of the traditional RDMS database and experimetal shows that the OLTP processing on a H-Store database is faster by a factor of 82

2. ARCHITECTURE

H-Store is a parallel, row-storage relational DBMS that runs on a cluster of shared-nothing, main memory executor nodes.

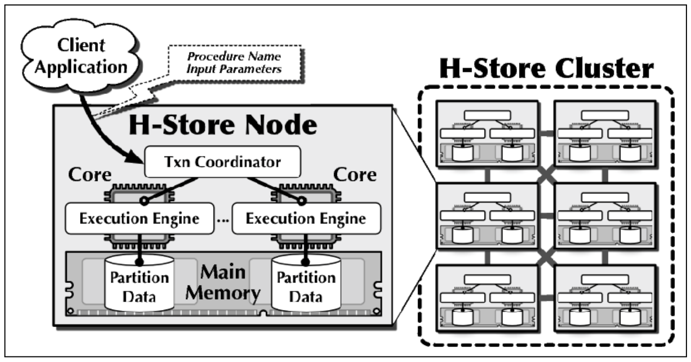

Fig. 1. H-Store Architecture

A single H-Store instance is a cluster of two or more computational nodes deployed within the same administrative domain. A node is a single physical computer system that hosts one or more execution sites and one transaction coordinator. A site is the logical operational entity in the system; it is a single-threaded execution engine that manages some portion of the database and is responsible for executing transactions on behalf of its transaction coordinator. The transaction coordinator is responsible for ensuring the serializability of transactions along with the coordinators located at other nodes. We assume that the typical H-Store node contains one or more multi-core CPUs, each with multiple hardware threads per core. As such, multiple sites are assigned to nodes independently and do not share any data structures or memory.

Every table in H-Store is horizontally divided into multiple fragments (sometimes called shards) that each consist of a disjoint sub-section of the entire table. The boundaries of a table's fragments are based on the selection of a partitioning attribute (i.e., column) for that table; each tuple is assigned to a fragment based on the hash values of this attribute. H-Store also supports partitioning tables on multiple columns. Related fragments from multiple tables are combined together into a partition that is distinct from all other partitions. Each partition is assigned to exactly one site.

All tuples in H-Store are stored in main memory on each node; the system never needs to access a disk in order to execute a query. But it must replicate partitions to ensure both data durability and availability in the event of a node failure. Data replication in H-Store occurs in two ways: (1) replicating partitions on multiple nodes and (2) replicating an entire table in all partitions (i.e., the number of fragments for a table is one, and each partition contains a copy of that fragment). For the former, we adopt the k-safety concept, where *k* is defined by the administrator as the number of node failures a database can tolerate before it is deemed unavailable.

Client applications make calls to the H-Store system to execute pre-defined stored procedures. Each procedure is identified by a unique name and consists of user-written Java control code intermixed with parameterized SQL commands. The input parameters to the stored procedure can be scalar or array values, and queries can be executed multiple times.

Each instance of a stored procedure executing in response to a client request is called a transaction. Similarly as with tables, stored procedures in H-Store are assigned a partitioning attribute of one or more input parameters. When a new request arrives at a node, the transaction coordinator hashes the value of the procedure's partitioning parameter and routes the transaction request to the site with the same id as the hash. Once the request arrives at the proper site, the system invokes the procedure's Java control code, which will use the H-Store API to queue one or more queries in a batch. The control code invokes another command in H-Store to block the transaction while the execution engine executes the batched queries.

Yes. All of operations within a transaction are atomic acros all the partitions involved in the transaction. When a transaction executes, it has exclusive access to the data at the partitions that it locks, therefore all of its operations execute on a consistent view of the database and are isolated from all other transactions (since no other transactions execute concurrently at those partitions). Finally, with snapshots and command logging, the state of the database is completely recoverable and all changes made by transactions are durable [2] [3].

3. LICENSING

A commercial implementation of H-Store is VoltDB.VoltDB is being developed for production environments, and thus it is focused on high-performance throughput for single-partition transactions and provides robust handling of failures that are obviously needed for a main memory system. H-Store does not have VoltDB's tools for managing and maintaining clusters. But because H-Store is a research database (thus does not provide many of the safety guarantees that VoltDB does), this allows H-Store to support additional optimizations, such as speculative execution and arbitrary multi-partition transactions.

For example, in VoltDB every transaction is either single-partition or all-partition. That is, any transaction that needs to touch multiple partitions will cause the VoltDB's transaction coordinator to lock all partitions in the cluster, even if the transaction only needs to touch data at two partitions. However H-Store has a different internal transaction coordination subsystem . For each query batch submitted by a transaction, the heavily optimized H-Store Batch Planner determines what partitions each query needs to access and only dispatches requests to those partitions.

H-Store uses VoltDB's VoltProcedure API, so any stored procedures written for VoltDB should still work in H-Store[4].

4. USE CASE

Big Data is data at rest. Big Data describes data's volume – petabytes to exabytes - and variety: structured, semi-structured and unstructured data that has the potential to be analyzed for information. Big Data systems facilitate the exploration and analysis of stored, large data sets. Big data is often created by data that is generated at incredible speeds, such as click-stream data, financial ticker data, log aggregation, or sensor data. Often these events occur thousands to tens of thousands of times per second. The benefits of big data are lost if fresh, fast-moving data from is dumped into HDFS, an analytic RDBMS, or even flat files, because the ability to act or alert right now, as things are happening, is lost.

So this data stream which is the source for big data is called Fast Data.For Fast Data we're not measuring volume in the typical gigabytes, terabytes, and petabytes . For Fast Data we are measuring volume in terms of time: the number of megabytes per second, gigabytes per hour, or terabytes per day. So the key difference between Big and Fast Data is Volumne vs Velocity.So to build systems and applications to take advantage of Fast Data to make real-time, per-event decisions that have direct, immediate impact on business interactions and observations we need a DBMS which offers high speed transactional ACID performance and the ability to process thousands to millions of discrete incoming events per second is ideal for processing such type of data. Fast data systems operationalize the learning and insights that companies derive from big data.

Traditional SQL databases offer rich transaction capabilities, ad-hoc query and reporting capability, and vast amounts of standards-based tooling. Where they fall short is in the ability to scale out to meet the needs of modern, high-performance applications.NoSQL is a solution to issues of scale that emerged as organizations began dealing with immense volumes, velocity, and variety of big data from a multitude of sources. NoSQL solutions offered availability, flexibility and scale. However, they came with difficult tradeoffs - sacrifice of data consistency and transactional (ACID) guarantees, loss of ad-hoc query capability and increased application complexity. Trading off data consistency and ACID in many cases means application developers, who are not distributed data infrastructure experts, have had to try and fill the gap.NewSQL attempted to do what NoSQL did without those tradeoffs. By removing just the parts of SQL that hampered scalability and performance, it was able to increase both while maintaining key advantages of SQL systems.NewSQL technologies offer the best of both worlds: the scale of NoSQL with the ACID guarantees, strong consistency, minimized application complexity and interactivity of SQL.NewSQL systems are built for Fast Data.[5]

Below is a list of uses cases for Fast Data

- Realtime personalisation of content based on customer usage for instance based on the customers location give recommendations for Restaurants , Places to vist near by .
- Fraud Detection in financial services were when each payment in authorised we need to be able to detect any Fraud based on usage pattern

- Mobile advertising providers to accurately track ad placements and monitor clicks for these adds so that they can enforce budgets and actual advertisement expenditures
- Internet of things applications based on sensor data to predict or alert issues faster enabling quicker resolution .A classic example for a Airline getting streaming data on telemetrics from sensors.

REFERENCES

[1] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The end of an architectural era: (it's time for a complete rewrite)," in *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 1150–1160. [Online]. Available: http://hstore.cs.brown.edu/papers/hstore-endofera.pdf

[2] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi, "H-Store: a high-performance, distributed main memory transaction processing system," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1496–1499, 2008. [Online]. Available: http://hstore.cs.brown.edu/papers/hstore-demo.pdf

[3] "H-StoreArch," Web Page, accessed:2/5/2017. [Online]. Available: http://hstore.cs.brown.edu/documentation/architecture-overview/

[4] "H-StoreFaq," Web Page, accessed:2/5/2017. [Online]. Available: http://hstore.cs.brown.edu/documentation/faq

[5] "FastData," Web Page, accessed:2/5/2017. [Online]. Available: https://www.voltdb.com/fast-data