Distributed Systems Architecture

brought to you by Alexey Grishchenko

Cloudera Kudu: Catching a Unicorn

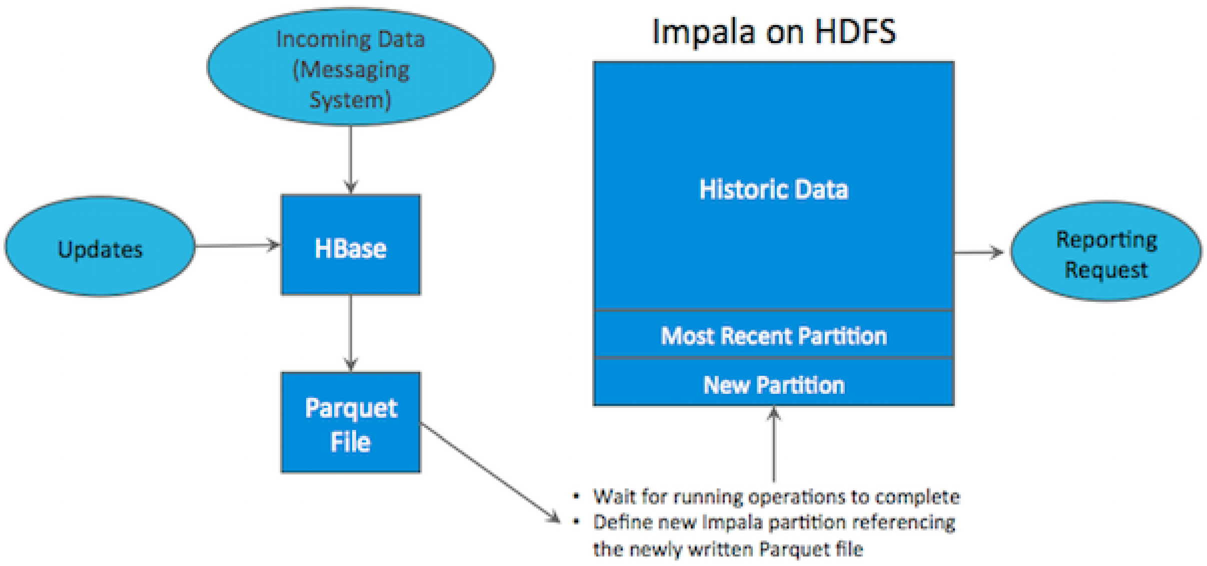
Recently Cloudera announces new storage engine for fast analytics and fast data called Kudu. This is a very interesting piece of code and I couldn't withstand an attraction of analyzing this technology deeper and going beyond the marketing.

Small disclaimer: all of the facts written beyond is my personal opinion based on my experience on DWH and Hadoop markets.

Official Information

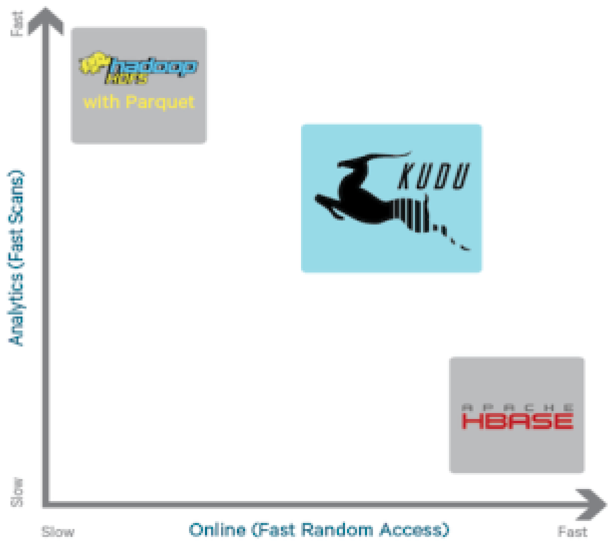
First let's start with official links for the ones who likes to analyze complex systems themselves and doesn't like to rely on the biased opinions (just like me). Here's the official statement, technical whitepaper with design details and the source code itself.

We should take a look at the official statement from Cloudera. This system aims to cover the gap between the capabilities of HBase as fast transactional store and Impala on HDFS as fast analytical engine. The typical workload of many customers, which comes along with my experience, looks similar to this:


Incoming Data (Messaging System)
Updates
HBase
Parquet File
Impala on HDFS
Historic Data
Most Recent Partition
New Partition
Reporting Request
• Wait for running operations to complete
• Define new Impala partition referencing the newly written Parquet file

The data first arrives to the transactional store used as a backend for business-critical applications, and then with batch ETL it is moved to the analytical engine where the business users build their reports on this data. This is a proven design having its roots in 30+ years old traditional data warehousing approach. It tells you to have a number of separate transactional systems replicating their data (daily ETL or ELT or CDC) to a single analytical store called DWH, acting as a single point of truth for the business users. It is the right architecture that proved its validity.

Here is the official positioning of Cloudera Kudu:



Design Analysis

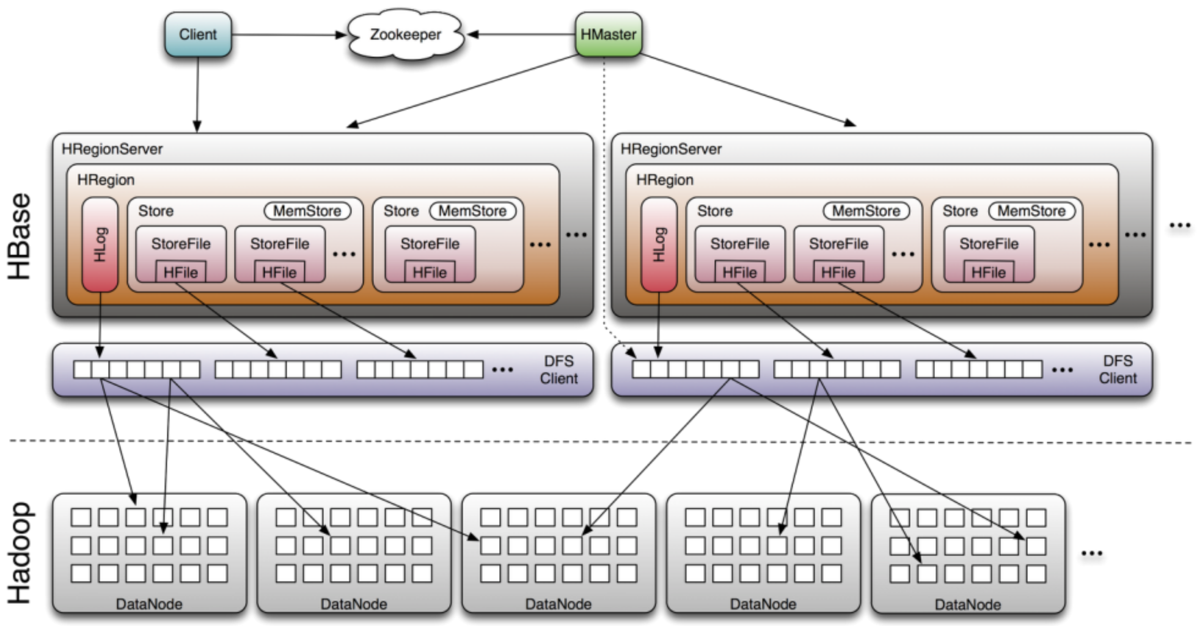
As you can see, it is in between analytical and online workloads. To be more special, it is between OLTP and OLAP systems. Interestingly, just a couple of weeks ago I published an article discussing exactly the same design approach and the problems it introduces.

Official design goals of Cloudera Kudu according to their announcement are the following:

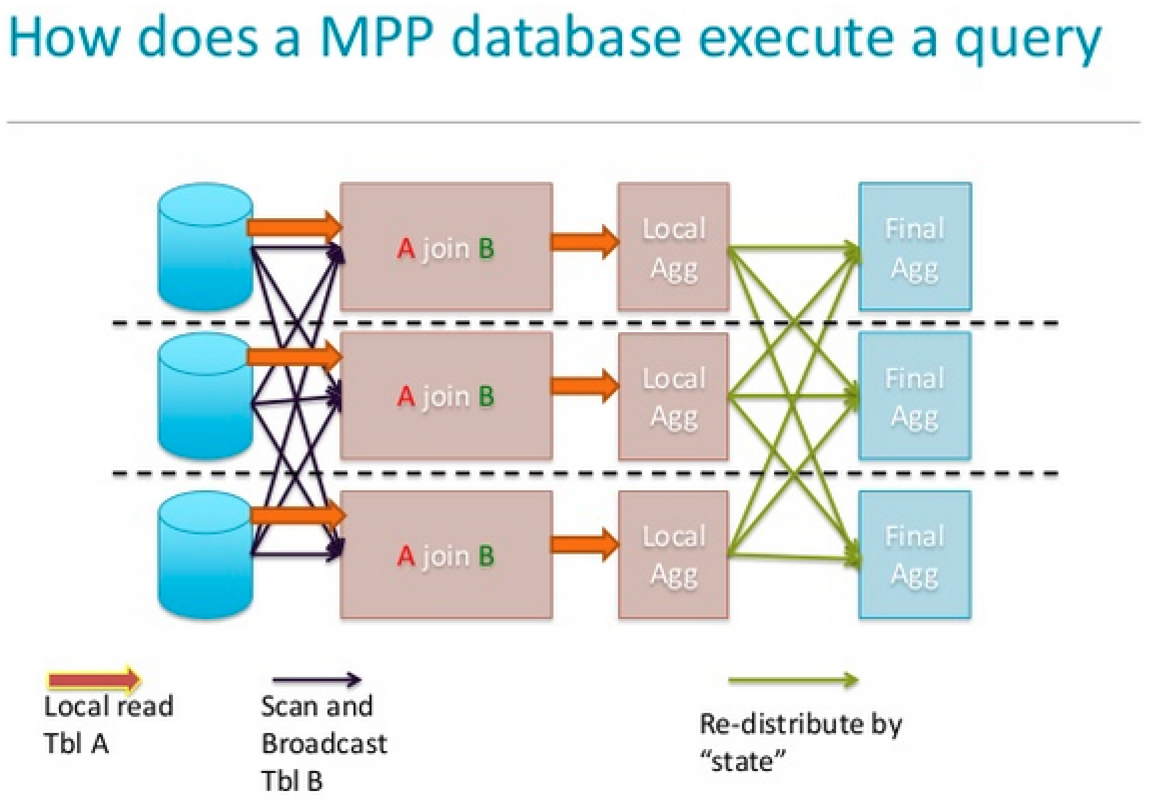
- Strong performance for both scan and random access to help customers simplify complex hybrid architectures
- High CPU efficiency in order to maximize the return on investment that our customers are making in modern processors
- High IO efficiency in order to leverage modern persistent storage
- The ability to update data in place, to avoid extraneous processing and data movement
- The ability to support active-active replicated clusters that span multiple data centers in geographically distant locations

The first point here is the most important. Combining random access with sequential “scan” access to simplify hybrid architectures.

First let’s think about how the customers use their “random access” systems, or namely HBase. The idea under its design is to have a fast key-value store capable of scaling to petabytes of data on thousands of machines. But why do you need the key-value store first of all? You need it when you want to achieve consistent, low-latency lookup capabilities over the huge dataset. Based on the practice, average performance is not the main concern of these systems, but rather p999 or p9999 response time is the crucial part. If in 99% cases your system is responding in 10ms, in 0.99% its response time goes to 300ms and in 0.01% its response would be more than 1s, and it might become a problem. If your system is designed to handle millions of transactions a day, then 0.01% becomes thousands of operations daily, which most likely would cause the problem on customer side visible to the business.

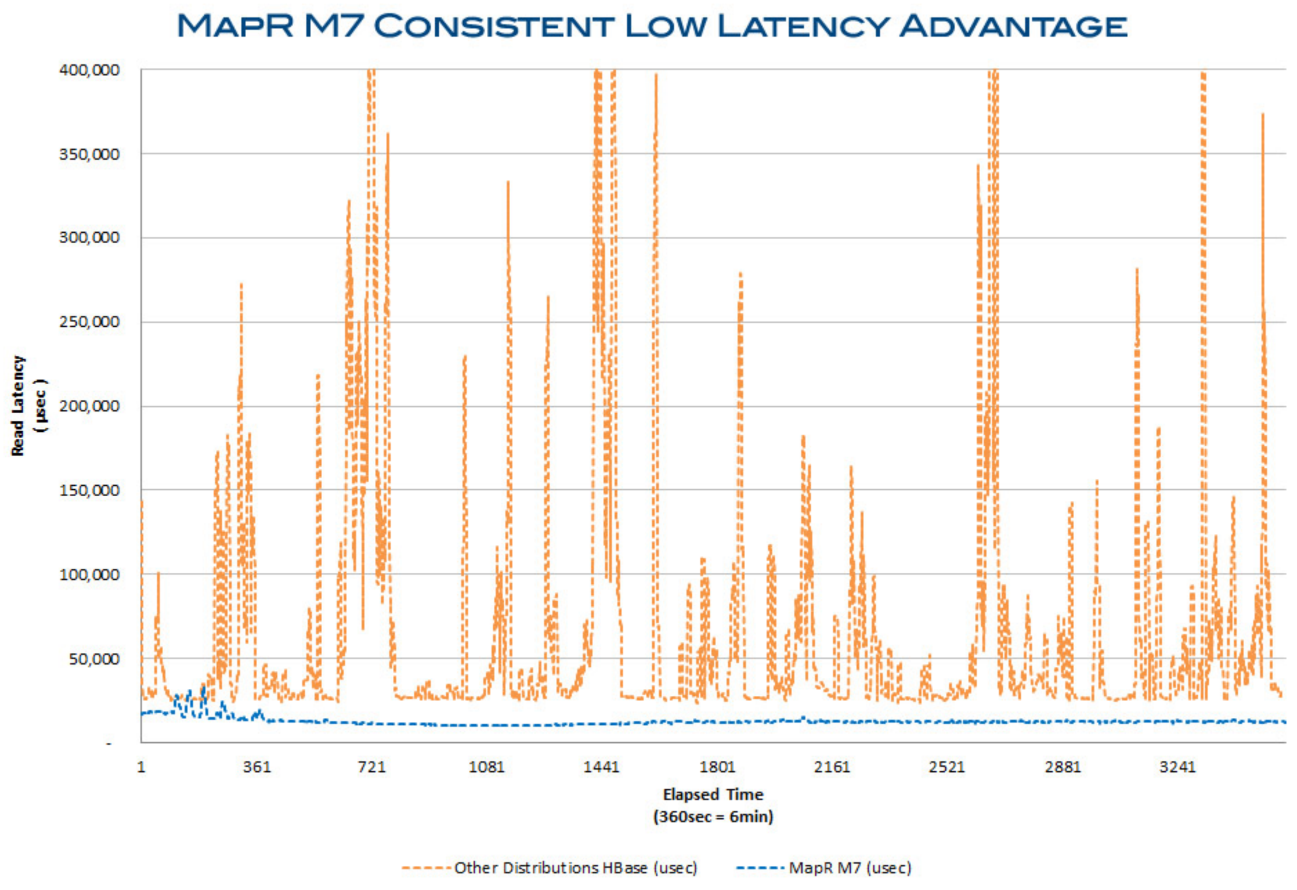


Now let’s switch to analytical systems. What we usually see on the DWH and Data Lake part of the picture is that allowing ad-hoc queries to the end users in analytical system is usually the main requirement, and these particular end users is the main source of problems of analytical system. They tend to generate non-optimal queries with Cartesian products in result set, stress the underlying filesystem by omitting the partitioning column condition which causes full table scan instead of partition pruning, generate huge joins causing big table re-hashing across the nodes stressing both network and CPU. Usually analytical systems support very low concurrency, like 10-15 parallel queries at max. Cloudera Impala is built on MPP principle, which allows any single query to optimally utilize almost all system resources. If 10 queries of this kind were running in parallel, this would cause complete cluster saturation of all IO, CPU and network resources.



Now combine these two workloads together. Business users generate ad-hoc queries causing cluster saturation, at the same time business-critical real-time workload experience big spikes in latency, which is similar to the service outage from the user standpoint. This is the typical mistake for such combination, even Cloudera itself is warning you: "*Be careful when running mixed workloads on an HBase cluster. When you have SLAs on HBase access independent of any MapReduce jobs (for example, a transformation in Pig and serving data from HBase) run them on separate clusters*".

Here's an example of how it might look like, with a glance of MapR marketing that can be omitted:



I don't say that Cloudera Kudu is a bad thing or has a wrong design. In fact, it is a very interesting piece of code, implementing a big amount of optimizations and stability features. I think that it has a very big potential, but its positioning is completely wrong. Maybe its just an issue with Cloudera marketing that they focus the OLTP/OLAP combined case, which is well-known to be a unicorn of data processing systems and catching it is mainly pointless.

Cloudera Kudu from architecture standpoint is

- Kudu is a new distributed system written in plain C++ by Cloudera engineers
- Kudu has no dependencies on other Hadoop stack components
- Kudu offers you updatable storage, its SQL operations support update and delete
- Kudu does not have its own SQL interface, you can either use Cloudera Impala for optimal SQL performance, or Spark with MapReduce input formats provided by Cloudera
- Kudu is a store for structured data with fixed table schema. It uses column-oriented disk store and row-oriented in-memory store, similar to Vertica
- Each table must have a primary key, it is used for data sharding across the cluster. Unlike HBase, it allows you to combine hash and range sharding (first hash, then range), data is stored in sorted order
- Fast OLTP access is available only by primary key
- MVCC versioning of data with snapshot isolation read consistency
- Many performance optimizations are built-in. Bloom filters on chunks of data stored, linked concurrent B-tree stored in-memory, lazy materialization of columnar data, RAFT algorithm for consensus and so on

It is a very interesting technology, and I think it might be promising if the community would accept it in a right way and if it would be positioned appropriately.

Cloudera Kudu is good for

- Analytical user workload
- DWH-like use case when you replicate data from many source systems. In my opinion it would be a great solution as a target for CDC replication from a number of OLTP databases
- Updatable storage makes it even closer to DWH systems
- Combination of Cloudera Kudu and Cloudera Impala might be positioned as a comprehensive DWH platform with MPP-like design

In general, it is resembles Oracle Exadata for me. Its engine also has an "intelligent storage" that is capable of predicate pushdown, partition elimination and efficient columnar scans for the Oracle DB instances running above it. Here the logic is very similar with Cloudera Impala running on top of Cloudera Kudu.

Real drivers of Cloudera Kudu introduction

- Better presence at DWH market for Cloudera. Kudu offers updatable storage which DWH customers are not yet ready to give up on
- Introducing greater "distinction" from other Hadoop distributions. More Cloudera-only components make customer retention easier and allows to better position the "distinctions" to the prospects
- Overcoming the limitations of "community" for existing Hadoop components in evolving their design. Hadoop community is not only Cloudera, and they are not always welcome to do radical changes. With in-house built system like Kudu, Cloudera offers an alternative to HDFS as a given project, i.e. it is a revolution instead of evolution

If the Cloudera would position it right, not lose the momentum for the community building and attract great names as early adopters of this technology, I think it has a promising future.

Share this:

27

This entry was posted in DBMS, Hadoop, SQL-on-Hadoop and tagged cloudera, dwh, hbase, impala, kudu, mpp, sql-on-hadoop on September 29, 2015 [https://0x0fff.com/cloudera-kudu-catching-a-unicorn/] .