A Technical Comparison of Apache HBase and Cassandra

1 Introduction

Apache HBase and Cassandra are superficially similar databases; both following the wide-column model and being based on preceding BigTable databases. Both were initially released in 2008 and have seen widespread use from commercial and academic users in the subsequent decade and they remain a popular alternative to relational database management systems (RDBMS).

2 General Architecture

2.1 HBase Architecture

HBase may be considered one of the components of the Hadoop ecosystem. It runs on top of the HDFS filesystem, and so may be used to store and retrieve data with all the advantages of HDFS, but also allowing for fast data access as HBase is optimised to allow for multiple random reading and writing of data, rather than the 'write once read many times' use case that HDFS is typically used for. Hbase stores data in HDFS files, which are split between 'region servers', these act on top of HDFS data nodes - with the number of region servers being equal to the number of HDFS nodes available. The region servers are controlled by master servers which update individual tables in the region servers as required **?**. Zookeeper, whch is part of the standard Hadoop stack, acts to control the entire cluster, monitoring the region and master servers; therefore allowing for load balancing and node failure detection. A weakness with this setup is that a namenode is required which can form a single point of failure if a hot standby type configuration is not utilised.

The structure of HBase tables themselves is derived from Google's BigTable database. Hbase is a column orientated database and, as such, is designed to store structured data (or at least, semi structured data). Each table is divided into column families, with each column family comprising of multiple columns. Each row requires a key - allowing for unique rows to be identified. Crucially, columns are not saved in a fixed schema; only column families are defined. This allows for dynamic scaling of tables in the 'width' direction, allowing HBase to achieve its design goal of tables on the order of 'billions of rows and millions of columns'. Tables are automatically sharded between the available HDFS nodes **?**.

In order to account for high write-demand applications, HBase makes use of Log-Structures Merge Trees (LSMTs) **?**. Each update is first written to the Write Ahead Log (present for every region server). For each table partition affected by the update the Memstore, which is an in-memory tree, is updated. The memstore is periodically written to HFiles, which are immutable and reside on the disk, as the memory limit is exceeded. Bloom filters may be enabled in order to reduce the number of disk accesses.

2.2 Cassandra Architecture

Cassandra operates independantly of HDFS, using it's own file system - CFS. This means the filesystem is not constrained by the existing architecture of HDFS. Like HBase, Cassandra splits data between multiple nodes in a cluster. However, unlike HBase, each node is identical in structure and purpose. Every second all nodes exchange information with every other node to ensure data is consistent across the entire cluster. This ensures that, providing nodes are located in seperate physical locations, Cassandra under no circumstances has a single failure point. Every node contains a commit log, which is modified every time the node performs a write or update. The high level of distribution and consistent, homogeneous nature of individual nodes makes Cassandra ideal for deployment in multiple locations where it can operate in an 'always on' manner **?**.

Tables in Cassandra differ from HBase in that they are modelled after DynamoDB. Fundementally, the struture of a table is similar to HBase, with each table consisting of a number of column families, each of which contain a number of columns **?**. It is helpful not to think of a Cassandra database as a collection of tables split into columns. Rather, each column family is more analogous to a table in a RDB; within each column family, the columns are arranged in a nested sorted map which allows for a huge number of columns to be

stored and retrieved. Each cell is identified by a unique row and column key. Since the number of column keys is unbounded, and columns can be valueless, the format of columns is variable; making Cassandra a wide column database, like HBase. **?**

At the node level, data writes in Cassandra are similar to HBase. Each write is written to the commit log and then to the memtable where it is indexed in memory. The subsequent use of LSMTs, along with a sequentially updated commit log and periodic consolidation of immutable on-disk structures, make Casandra and HBase structurally similar at the node level.

3 Features Comparison

In order to compare the advantages of each database, the constituents of CAP theorem are considered **?**. In addition, read write performance of each database is also compared. One of the desirable features of NoSQL databases is the ability to run fast write/read operations for applicable use cases, so the performance is crucial when comparing databases.

3.1 CAP Overview

It is impossible to guarantee all three facets of CAP in a distributed database, so systems are typically designed on a 'pick two' philosophy. Cassandra and HBase differ in this respect, in that Cassandra is optimised for Partition Tolerance and Availability (AP) while HBase is optimised for Consistency and Partition-Tolerance (CP). For reference, RDBMS systems typically guarantee Consistency and Availability (CA) at the cost of partition tolerance**?**.

3.2 Consistency

HBase guarantees consistency by first guaranteeing Atomicity. This means any change (mutation) to a row either occurs in entirity or not at all, so it is not possible to partially update a given row. As a result, any row returned by a query is guaranteed to be a complete row that existed at some point in the table history. HBase strictly enforces strong consistency, and as such this requirment is not tuneable. Cassandra does not guarantee consistency, as it is constrained to eventual consistency only, but the level of consistency is tuneable at the expense of availability. Previous tests have shown the inconsistency window following a request to be relativley short, even under high workload; although if the system is places under sufficient computational stress, consistency becomes unpredictable. **?**

3.3 Availability

HBase sacrifices availability for Consistency and is therefore typically not used for the realtime streaming applications that Cassandra is used for. The reduced availability of HBase stems from the use of region servers. The loss of a region server causes availability to be degraded until other servers can be reassigned. This loss of availability actually guarantees consistency, since all users can only access a single version of the data **?**. Cassandra, on the other hand, is designed to ensure high availability. This is achieved through the consistent structure of each node so if a node goes offline the commit log from other nodes may be used to provide access to the data immediately. The drawback of this high availability is that there is no guarantee all nodes will be in sync at the time of the request, so consistency is not guaranteed. However, unlike HBase, Casssandra allows the user to manually set the level of availability, and corresponding loss of consistency. Due to this tuning ability, Cassandra doesnt sit firmly in one corner of the CAP triangle, although it cannot guarantee the same level of consistency as HBase.

3.4 Partition Tolerance

Running atop HDFS, HBase inherently has high partition tolerance. The use of Zookeeper to manage individual nodes ensures coordination between nodes and maintains a high level of redundancy across the

entire cluster **?**. Cassandra exhibits a similar level of partition tolerance to HBase; while it does not depend on HDFS, it runs its own independant protocol to ensure sufficient redundency between nodes **?**.

3.5 Performance

While both databases are designed to support fast random access to data, broadly speaking Cassandra is optimised for writing where HBase is optimised for reading. Multiple studies have observed significantly faster performance from Cassandra for read-intensive queries. Benchmark tests running locally on a virtual machine have showed Cassandra to be consistently faster at both reading and writing **?**, with both query types requiring approximaely half the time for 1000 read operations over on the order of half a million rows. Subsequent tests on a local cluster gave similar results, with Cassandra exhibiting superior write performance as the database size was scaled up **?**. Recent tests on a 4 node cluster indicated marginally better read performance for HBase for 100,000 records **?**. While these tests again showed better write performance for Cassandra, the difference was lesser so than previous tests, with HBase capable of writing 100,000 records in 4000ms, to Cassandra's time of 5500ms. Cassandra was also shown to scale better as the record count increased, with HBase peaking in throughput at 100000 writes, while Cassandra's throughput in operations/second continued to increase with write count.

In reality, benchmark testing can only reveal so much about performance since the two databases are enterprise products used by commercial organisations. Although Cassandra generally outperforms HBase for both write and read operations **?**, this is not necesaily true for all real-world use cases. When databases are scaled up, with nodes and end users seperated geographically, HBase has been shown to exhibit improved read-performance for multiple use-cases **?**.

4 Conclusions

In a hypothetical scenario where all things are equal, Cassandra would be chosen for applications requiring fast read and writes, but where consistency is not as much of a concern. HBase guarantees consistency to a greater extent than Cassandra, so would be chosen for use cases where this is important. Both are highly partition tolerant, a common feature for most NoSQL databases. In reality it is unreasonable to draw any firm conclusions since there is never a 'best' database - only one which is best suited for the use-case at hand.

5 References