A Technical Comparison of Apache HBase and Cassandra

I. 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), being activley maintained and supported by Apache. This review highlights the main similarities and differences between HBase and Cassandra, identifying suitable use cases for each where appropriate.

II. General Architecture

A. 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 being 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). As with a RDBMS data is stored in tables, however each table is divided into column families, with each column family comprising of multiple columns. Each row required 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'. In order to ensure storage limits of individual nodes are not exceeded, to maintain data redundancy, and to ensure optimum read/write times, tables are automatically sharded between the available HDFS nodes.

In order to account for high write-deman applications, HBase makes use of Log-Structures Merge Trees. 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. To improve read access, bloom filters may be enabled in order to reduce the number of disk accesses.