Cassandra

Magesh khanna Vadivelu
magevadi@indiana.edu
Indiana University Bloomington

Abstract

Cassandra is an open source distributed storage system for managing very large amount of data across multiple data centers. Cassandra run on cheap commodity hardware and delivers high read/write efficiency over other relational data bases that exist. Cassandra does not provide relational data model.

In this survey paper we will analyze data model, storage methods, efficient retrieval techniques of Cassandra and possible advantages and disadvantages over other data storage and retrieval techniques. Originally developed by Facebook to meet expected performance and scalability for its very high volume of user, later Cassandra was released as open source.

Key Words: Chubby, Distributed, Facebook, Google File System, scalability.

1. Introduction

Cassandra was developed by Facebook to solve the Inbox search problem. Cassandra was designed to handle Facebook's millions of search hits per day. Created by Avinash Lakshman and Prashant Malik. Later in 2008 it was released to Google code as open source project. In 2009 it became apache's incubator project and promoted to top-level project in 2010.

Cassandra has a structured key-value store with tunable consistency. Each key maps to multiple values, which can be grouped into multiple column families. Columns can be added to a family any time after creating a Cassandra database, but column families are fixed while creating. Different columns can have different number of keys in a given family.

2. Background

Data storage was made easy by SQL based products such as Oracle, MySQL [1], and Microsoft SQL, but this relation data base model is not scalable for very high data sets such as millions of records categorized into thousands of categories. SQL based data bases took very long time to read and write for such a large data sets.

Peer to Peer storage systems was one storage models that helped come out of conventional relational data model and served as inspiration for several distributed file models after that. It supports flat namespaces around the ring. Replication made easy through P2P systems up to a level. Update conflicts have been addressed by conflict resolution techniques. High availability and scalability can be achieved through consistent replication. This was found through extended researches in P2P applications.

Several industries which process data in very high volume came up with solutions to handle very large data sets. Some of those solutions are: Bigtable, dynamo, Voldemort, Hypertable, Hbase, and Cassandra.

The need for a flexible data model like BigTable, Dynamo or Cassandra for content oriented data applications is consistency, availability, partition tolerance that it delivers.

Although our main focus in this literature review is Cassandra, a short introduction about birth of Cassandra. Cassandra was inspired from Google's BigTable and Amazon's Dynamo.

BigTable is a proprietary data base system built on Google File System (GFS), Chubby Lock Service and SSTable originally designed for powering Google's products. GFS stores data in a distributed chunk servers and stores the information about the data (Meta data) in a single server that serves as central server for all queries. This GFS master server is made fault tolerant with the aid of Chubby [2].

Dynamo is a key-value structured storage system. It also has characteristics of data base and Distributed Hash Tables, designed by Amazon for Amazon Web Services. Dynamo detects update conflicts using vector clock scheme. Other way to perform this is by client side conflict resolution mechanism. Each write operation in Dynamo needs a read operation to be performed before that, this can be a disadvantage at situations like where there is loads of read overhead and needs write efficiency.

Cassandra has characteristics of both Google's BigTable and Amazon's Dynamo. Cassandra has gossip based membership algorithm to help each node to maintain information about every other nodes, this idea is from Dynamo.

3. Properties

Cassandra is linearly scalable, consistent, high availability, partition tolerance; it has no single point of failure, ease of administration, replica placement, automated provisioning, and column based storage, fault tolerant, and load balancing. Cassandra does not provide revision history. Since Cassandra is written in Java, the variables are byte strings. Usually applications use a dedicated Cassandra cluster and manage them as part of their service. Although the system supports the multiple tables all implementations have only one table in their schema.

4. Data model

Data in Cassandra are organized as keys, values and columns. Value can be a highly structured object. Each row in a Cassandra table can be identified by key. A key can be a string with no size restrictions, but typically it is between 16 and 36 bytes. Every operation under a single row key is atomic irrespective of how many columns that the row key is associated into or how many columns are manipulated in that particular operation.

4.1. Columns

Each column stores its data as name, value and timestamp. The timestamp is used by Cassandra to preserve differences across nodes, not to store revisions, Cassandra does not provide revisions. The timestamp is generated automatically from UNIX time () function; since Cassandra was written in Java, the name and values are stored as byte arrays. Each column is represented by UUID. Columns are found by the name value pairs and not UUID. UUID is only for internal representation and is automatically done by Cassandra. It is not possible to have multiple columns with the same name. Creating a column with a same name will replace the existing column in that particular key space.

4.2. Super Columns

Super columns are columns inside columns. It allows us to create nested columns. As a result, it is possible to store any number of super columns inside a key so that a hierarchy of columns is also possible by this data structure. Super column does not have timestamp as column does. Any column is of type super can be accessed with the following convention:

Column_family : Super_column : column[6]

4.3. Column family

Similar to BigTable, columns are grouped together as column families. Column families can be of two types, simple column family or super column family. Column family has infinite number of rows, similar to RDBMS. Super column family can be seen as a column family inside column family. Any column within a column family can be accessed with by the convention *column_family : column*[6]

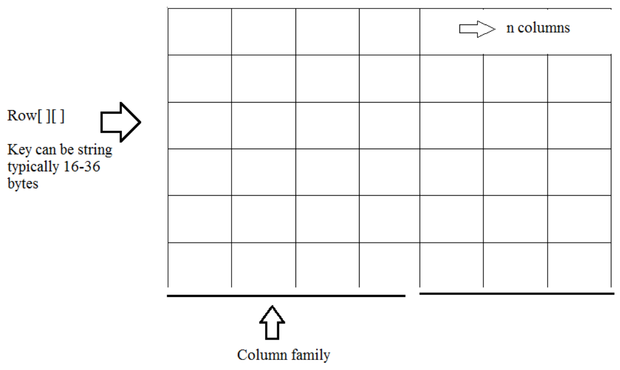
Column family can be two types. Standard column family or super column family

4.4. Super Column Family

A combination of column family and super column, a super column family is the largest container. Instead of columns in the container, we can have super columns so that its possible to create super column family hierarchy. The key of the map which has the super column must contain the same name as the super column. This is also the case of column family.

4.5. Columns vs. SuperColumns

Columns and SuperColumns both are a tuples with name and value. The only difference is that a standard Column's value is a string and SuperColumn's value is a Map of Columns. Their values contain different types of data. Another difference is that SuperColumn's don't have a timestamp component.



A brief graphical representation of columns and keys

4.6. Keyspace

Keyspace has multiple column families and super column families without any relationship between each of them. Unlike tables in Relational Data Bases, it is impossible to join them. Columns in each keyspace has unique name. Keyspace can be seen as outer ring of the whole data structure.

5. Sorting

It is not that easy to query out data in Cassandra as we do in SQL. Further, we cannot have our data sorted during the query like SQL. The data is always sorted when we throw it into the Cassandra cluster. This sorted data always remain sorted in the cluster.

This gives much performance boost during read operation but in exchange for we have to plan ahead of our data model during the time of design in such a way that, any of the sort models will benefit us during the read operation in future.

Columns in Cassandra are always sorted within their row by the Column's name. This is a very advantageous future of Cassandra. Cassandra decides how to compare the names depending on the Column Family's *CompareWith* option. We have the following options for column's name:

- *BytesType*
- *UTF8Type*
- *LexicalUUIDType*
- *TimeUUIDType*
- *AsciiType*
- *LongType*

Each of these options treats the Columns' name as a different data type, which gives us a lot of flexibility in terms of column name.

Ex: Using *LongType* will treat Column names as 64bit Long.

In Cassandra, it is possible to write our own custom sorting function. Sorting function in Cassandra is overridable.

6. Snitch

In Cassandra, Snitch is a way to map node to a physical location in the network. Snitch helps to find the location of a node relative to another node in order to assist with discovery and ensure efficient routing. The following are types of Snitch.

- Simple snitch
  - Default
  - It has N-1 consecutive nodes
- Property File Snitch
- Rack Inferring Snitch

7. Read Operation

Clients make read requests to random node in Cassandra cluster. The node which receives the request acts as proxy and determines which nodes have copies of data. This node then requests the data from the nodes which has the copy of the data.

The requested client can figure out the strength of the read consistency.

- Single read: The request returns once it gets the first response, but there is a chance that the data can be obsolete, so this read must be avoided in possible situations.

- Quorum read: The request returns only after the majority of nodes responded with the same value. This read ensures that the data is fresh and consistent among nodes, but it takes a toll on read performance.

The requested node also performs read repair, if there is any kind of inconsistent response from the resultant read operation. Each node reading data uses either in memory Memtable or SSTables located at disk

To scan the SSTable, Cassandra uses a row-level column index and bloom filter to find the necessary blocks on disk, desterilizes them and determines the actual data to return. During this operation there is a lot of disk IO latency makes the read latency higher than a DBMS.

The underground reason behind uncached reads that are slower in Cassandra not just because the SSTable is very IO intensive. SSTable is actually a b-tree based storage that offers very data retrieval at very low seek time. It is because in most of the disk access cases in SSTables, there is a need to merge more than one SSTables to complete the request. This happens just because the SSTables are not up to date in place to serve the requests as and when it has been placed.

To overcome the problem of high latency due to uncached disk access of SSTables. Cassandra employs a row caching technique that solves the read latency problem.

8. Write Operation

Client submits its write request to a single random Cassandra node, similar to a read operation. The node behavior is similar to proxy writing data to the cluster.

Writes are replicated to N nodes according to the replication placement strategy. Each of the N nodes performs two actions when receiving a write in form of RowMutation.

Append the mutation to the commit log for transaction purposes. Update an in-memory Memtable structure with the change.

During which the following asynchronous operations are performed:

- Memtable is written to disk in a structure called SSTable
- SSTables corresponding to a *column family* are merged into a raw ColumnFamily datafile.

9. Partitioning

Cassandra's partitioning strategies fall into two broad categories. Random Partitioning (RP) and Order Preserving Partitioning (ORP). These control how our data is preserved in the distributed cluster of data base. Once we partition the data and stored in the cluster, this can never be rolled back or re partitioned again. So, one must be careful before partitioning the data.

10. Accessing the data

Data in Cassandra can be accessed through apache Incubator **Thrift API**. Thrift is available for wide variety of languages

Thrift can be accessed by any programming language or application. Thrift is a compact binary RPC framework. Even though there are lot of documentation that support this Thrift API. The collective response for this API is negative.

11. Downside of Cassandra

Even though Cassandra delivers an incredible scaling and offers very high write through. It has several notable disadvantages worth mentioning. Cassandra has no joins, it has only index and sort keys. Cassandra is not suitable for large blobs. Cassandra rows must fit into the memory. Its built on Thrift API, which is the only way to access the data.

12. Applications

Cassandra is extensively used by its founders Facebook [6] for Inbox searching and other search options. Digg [8], Rackspace [10], RocketFuel are the other implementations of Cassandra at Industry level. IBM Research also uses Cassandra for its CloudDB [3] experimentations, its releases are: BlueRunner [4] and scalable text index [5].

13. References

[1] **MySQL** Query Analyzer: Improve database application performance
*www.**mysql**.com*.

[2] Mike Burrows. The chubby lock service for Loosely-coupled distributed systems. In OSDI '06: Proceedings of the 7th symposium on Operating Systems design and implementation, pages 335{350, Berkeley, CA, USA, 2006. USENIX Association.

[3] IBM Cloud DB: IBM Research Lab
http://www.almaden.ibm.com/cs/projects/clouddb/

[4] Jun Rao: BlueRunner: Building an Email, IBM Almaden Research Center

[5] Ning Li, Jun Rao, Eugene Shekita, Sandeep Tata: Leveraging a Scalable Row Store to Build a Distributed Text Index:

[6] A. Lakshman, P. Malik: Cassandra - A Decentralized Structured Storage System, Facebook.

[7] Cassandra Read Explained:
http://nosql.mypopescu.com/post/474623402/cassandra-reads-performance-explained

[8] Cassandra in Production @ Digg
https://nosqleast.com/2009/slides/sarkissian-cassandra.pdf

[9] Dietrich Featherston: Cassandra: Principles and Application, University of Illinois

[10] Cassandra at Rackspace:
http://www.rackspace.com/cloud/blog/2009/09/23/the-cassandra-project/