CMPE226_Team4_Paper_Report

Task to Tackle (Problem to Resolve)

Facebook is one of the biggest social networking website, serving millions of users and still growing. High performance, reliability, efficiency and scalability are strictly required. Billions of messages are sent and received per day by its users, giving rise to a problem called the Inbox Search problem. The Inbox Search is a tool that allows the users to search through all of their messages either by the name of the person they communicated with or by a keyword present in the text of the message.

With billions of writes per day, the system is required to keep up with the increasing number of users, provide high write throughput and replicate data across the numerous data centers distributed across the globe while keeping search latencies down.

Assumptions/Operating Environments/Intended Usage

While developing the system, certain assumptions and operating environments have been defined. Some of them are as follows:

- At any given point of time, there is a small, but significant, number of servers and network components that are failing.
- Millions of users spread throughout the world are using the system, and the number is increasing continuously.
- The data centers are distributed geographically. Their locations have varying geographical and environmental factors like altitude, temperature, humidity, earthquake zone, etc.
- The system provides consistent data.

Given the assumptions, the system is intended to be used in a high scale distributed fashion. For this, the system must be able to provide high availability, scalability, performance and efficiency.

Design Tradeoffs

Since the datacenters are distributed, the system is required to provide data consistency throughout the system. However, the system is also supposed to provide high availability, performance, scalability and efficiency. According to the CAP theorem, it is impossible for a distributed system to provide consistency, availability and partition tolerance at the same time.

The major design tradeoff for the consistency issue is that Cassandra provides eventual consistency by offering Tunable Consistency mechanism. The consistency of the requested data is set by the clients themselves. Response time vs result accuracy depends on the user requirements. Cassandra v2.0 provides linearizable consistency. However this results in significant loss in performance.

Solutions and Implementation

High availability and performance are the explicit requirements of the Inbox Search tool of Facebook. However, to provide them, consistency of data goes down. Cassandra's take on solving this problem includes the following techniques:

- For consistency, Cassandra provides eventual consistency to the system. This allows the system to achieve high availability.
- Cassandra is able to scale incrementally, i.e. new nodes can be added as and when needed and the system will adapt to those nodes automatically. Consistent hashing is used to provide better scalability compared to traditional has tables.
- Cassandra provides high availability through replication by using three policies: Rack Aware, Rack Unaware and Datacenter Aware. Cassandra is also integrated with Zookeeper for electing the leader node. Spreading

storage nodes across multiple datacenters connected through high speed network links also helps with the availability of the system.

Cassandra uses replication technique to provide high availability. Each data item is replicated at N hosts, where N is the replication factor configured “per instance”.

Cassandra processes on a single machine consist of the following abstractions:

- Partitioning Module:
- Cluster membership and Failure Detection Module:
- Storage Engine Module

Each module is implemented from the scratch in the Java programming language. The cluster membership and failure detection module is built on top of the Network Layer in the TCP/IP stack. All system messages are sent using the UDP for replication, while routing relies on TCP. The system can be configured for synchronous or asynchronous writes, but for high throughput, asynchronous replication is used.

Cassandra uses a modified version of the φ Accrual Detector for failure detection, which is also used for avoiding communication attempts with unreachable nodes. For data persistence, Cassandra relies on local file system. To increase disk throughput, a dedicated disk is used for each machine for the commit log. Writes are sequential and generate an index for efficient lookup. For read operations, a bloom filter is used to summarize the keys in a file.

A compaction process similar to that of Bigtable is used to merge sort multiple, pre-sorted files. The system always merges file that are close to each other in size. A major compaction process is run occasionally in order to merge multiple files into a single, large file.

Results

The key characteristics of the system like partitioning, replication, membership, failure, handling and scaling are achieved successfully. Cassandra can truly support a high update throughput while maintaining low latency. Facebook’s Inbox Search has been using Cassandra since 2008 for its Inbox Search tool, keeping up with its promise of high performance and efficiency and scaling up with the increasing number of users throughout these years.

What we have learnt

Cassandra is an open source distributed database management system written in Java. It’s designed to be a highly scalable second-generation distributed database. In Cassandra documents are known as “columns” which are really just a single key and value. It also has Bigtable-like features, columns & columns families. You can query by column, range of keys. There’s a timestamp field which is for internal replication and consistency. The value can be a single value but can also contain another “column”. These columns then exist within column families which order data based on a specific value in the columns, referenced by a key. In Cassandra, nodes represent ranges of data. By default, when a new machine is added, it will receive half of the largest range of data. You can change this behaviour by choosing different configuration options during node start-up. There are certain configuration requirements to ensure safe and easy balancing, and there is a rebalance command that can perform the work throughout all the data ranges. It comes with monitoring tool that allows you to track the progress of the re-balancing. Cassandra is much lighter on the memory requirements, especially if you don’t need to keep a lot of data in cache. Cassandra requires a lot more Meta data for indexes and requires secondary indexes if you want to do range queries.

Another advantage of using Cassandra, it has much more advanced support for replication. The server can be set to use a specific consistency level to ensure that queries are replicated locally, or to remote data locations. This means you can let Cassandra handle redundancy across nodes, where it is aware of which rack and datacentre those nodes are on. Cassandra can also monitor nodes and route queries away from “slow” responding nodes. You can choose between synchronous or asynchronous replication for each update. It has got highly available asynchronous operations.

Major disadvantage is that Cassandra replication settings are done on a node level with configuration files whereas MongoDB allows very granular ad-hoc control down the query level through driver options which can be called in code at run time.

Best used: When you have more writes compared to read, sometime like logging events. Financial institutes are great examples as they care about each activity and logs more data. Some of the strong points of Cassandra can be summarized as:

- Highly scalable and highly available with no single point of failure
- NoSQL column family implementation
- Very high write throughput and good read throughput
- SQL-like query language (since 0.8) and support search through secondary indexes
- Tunable consistency and support for replication
- Flexible schema

Our Insight and Criticism

Cassandra is very different from traditional Relational Database systems. The most striking difference is that unlike the relational model that breaks data into many tables, Cassandra tends to keep as much as data as possible within the same row to avoiding having to join that data for retrieval. Some of the other differences are:

- Cassandra does not support ACID properties. It does not support joins either, instead stores as much data as possible in a single row.
- Cassandra does not support foreign keys, so the application has to handle data consistency. Also users cannot change the keys.
- Each key, for example row keys and column keys, has to be unique in its scope, and if the same key has been used twice it will overwrite the data.
- Cassandra supports idempotent operations which leave the system in the same state regardless of how many times the operations are carried out. All Cassandra operations are idempotent. If an operation fails, you can retry it without any problem. This provides a mechanism to recover from transient failures.
- Searching is not built into the core of the Cassandra architecture, and search mechanisms are layered on top using sort orders. Cassandra supports secondary indexes where the system automatically builds them, with some limited functionality. When secondary indexes do not work, users have to learn the data model and build indexes using sort orders and slices.
- Cassandra super columns can be useful when modeling multi-level data, where it adds one more level to the hierarchy. Anything that can be modeled with super columns, however, can also be supported through columns. Hence, super columns do not provide additional power. Also, they do not support secondary indexes. Therefore, the Cassandra developers discourage the use of super columns.
- If a node in a Cassandra cluster has failed, the cluster will continue to work if you have replicas. Full recovery, which is to redistribute data and compensate for missing replicas, is a manual operation through a command line tool called node tool.
- Cassandra is designed such that it continues to work without a problem even if a node goes down (or gets disconnected) and comes back later. A consequence is this complicates data deletions. For example, assume a node is down. While down, a data item has been deleted in replicas. When the unavailable node comes back on, it will reintroduce the deleted data item at the syncing process unless Cassandra remembers that data item has been deleted.

References

http://www.ibm.com/developerworks/library/os-apache-cassandra/
http://cassandra.apache.org/
http://docs.datastax.com/en/cassandra/2.1/cassandra/gettingStartedCassandraIntro.html
https://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf