Cassandra

Prakhar Srivastav (ps2894)

Summary

The **motivation** for Cassandra came from the need to store massive amounts of data in a scalable manner. In particular, the Inbox Search application provided the impetus to build a storage solution that could handle very high write throughput, billions of writes per day and also scale with the number of users.

To achieve this goal, the designers of Cassandra used a sysnthesis of well-known techniques in distributed systems to achieve scalability and availability. The designs and tradeoffs made in applying each of these techniques are outlined below as a part of this summary.

Data Model

The data model of Cassandra is extremely similar to the Bigtable system developed by Google. Like BigTable, tables in Cassandra are organized as a distributed multi-dimensional map indexed by a key. Each key points to a column family which are of two kinds - Simple and Super. The column families support sorting both by name and timestamp - this configuration is typically dictated by the application.

Architecture

In a distributed system at Cassandra's scale a lot of issues need to be managed - including but not limited to partitioning, replication, membership and failure handling. The paper primarily discusses about the those aspects which are relevant from a distributed systems point of view.

At a high-level, the read/write requests work first by determining the replicas for a particular key. This is required since any node in a Cassandra cluster can get a read/write request.

- **READ**: Based on the consistency guarantees required by the client, the system either routes to the closest replica or to all the replicas and waits for a quorum of responses. A typical read operation always looks up data first in the in-memory data structure. If found the data is returned to the application since the in-memory data struc- ture contains the latest data for any key. If not found then we perform disk I/O against all the data files on disk in re- verse time order.
- **WRITE**: The system routes the requests to the replicas and then waits for a quorum of replicas to acknowledge the completion of writes.

Partitioning

Data partitioning is the **de rigueur** of any data system that is designed to storage large amounts of data. For partitioning, Cassandra uses the well-known consistent hashing technique in which each node is assigned a range of keys for which it acts as a coordinator. In case of a failure or incremental addition of nodes, this scheme ensures that a node's immediate neighbors are affected - thereby making sure that only a fixed set of keys need to be re-assigned. One slight modification that Cassandra makes on the naive consistent hashing implementation involves factoring in the load information so as to load-balance deterministically.

Replication

The key premise of high availability in a distributed system is redundancy which is obtained through replication. In these systems, nodes are continously being added and removed from the cluster. In Cassandra, each node is responsible for a range of keys. Each of these keys are also replicated across various nodes. Since they are multiple nodes. Since there are multiple replicas for each key, there is requirement for having a leader which will dictate and orchestrate the keys each node is responsible for. For this, Cassandra relies on another system called Zookeeper. All nodes on joining the cluster contact the leader who tells them for what ranges they are replicas for and leader makes a concerted effort to maintain the invariant that no node is responsible for more than N-1 ranges in the ring.

Membership

For membership, Cassandra uses a gossip-based protocal called Scuttlebutt and like many other systems, the protocol is not only used for membership but also for piggy-backing system state information. The interesting part of membership protocol is failure detection and in Cassandra, this is done using a Accrual Failure Detector that works primarily by assign suspicion on a node rather than a boolean value. This value defined ϕ is a dyniamically adjusted value that is indicative of the network and load conditions in the system.

Bootstrapping

Bootstrapping is the process of adding/starting a new node in the cluster. Since the new node needs to know what keys it will handle, its configuration file contains a list of contact points (called seeds of the cluster) that it starts communicating with to learn about the cluster.

Scaling

Since Cassandra was designed to scale up incrementally, adding a new node is a very common process carried out in the system. When a new node is added into the system, it gets assigned a token such that it can alleviate a heavily loaded node. This results in the new node splitting a range that some other node was previously responsible for.

Persistence

Cassandra uses a write-ahead commit log for durability and recoverability. Any write that is recieved by a node first gets added to the log and only then mutates the in-memory data structure. When the in-memory data structure crosses a certain threshold, calculated based on data size and number of objects, it dumps itself to disk. When these writes are dumped to disk, an index is created in order to support fast lookup. Every time the in-memory data structure for a particular column family is dumped to disk we set its bit in the commit log stating that this column family has been successfully persisted to disk. Lastly, in order to prevent lookups into files that do not contain the key, a bloom filter, summarizing the keys in the file, is also stored in each data file and also kept in memory. This bloom filter is first consulted to check if the key being looked up does indeed exist in the given file.

Conclusion

In conclusion, the Cassandra paper demonstrates that how a system built using well known techniques and iterating by testing the ideas emperically can lead to robust, performant systems.