Paper Review: *Cassandra - A Decentralized Structured Storage System*

Paper Summary

In this work, the authors proposed a new decentralized structured storage system, called Cassandra. Cassandra was developed to solve inbox search problem that Facebook was facing. Cassandra is an open source, peer2peer distributed data store system that can scale out over thousands of nodes and store Terabytes of data. Joining and leaving of nodes in the system does not affect Cassandra functionality and behavior. Cassandra replicates data at different nodes to provide high availability, performance, reliability, and failure tolerance. Furthermore, Cassandra provides different consistency model to solve the consistency issues that might be caused by network partition, failing nodes, or concurrent updating of the same data. The provided consistency models provide different levels of consistency that can be specified by the user according to his application domain. Cassandra could have these properties by borrowing the design model from Dynamo and the data model from Bigtable. Cassandra can be seen as a combination of Bigtable and Dynamo storage systems.

Cassandra used a powerful design of dynamo to construct decentralized p2p storage system as in Dynamo. In this design the nodes are organized in a ring that has range from 0 to 2^{32} -1. The joined nodes in the system are assigned random numbers taken from the ring's range to be used as nodes identifiers. Each node stores data whose identifiers are less than or equal the node identifier and greater than the identifier of the predecessor node. The consistent hashing function MD5 is used to hash the data key to generate numeric data identifiers that also fall in the ring's range. The hashed keys enable evenly dynamic distribution of the data across clusters in different data centers. Also, these identifiers are used to route the read and write operations to the right nodes in O(1) in decentralized manner. This design is applicable for key-value data model that support easy management of data. While this data model brings powerful advantages with, it does not provide the ability to use the relationships between the data. The data value in this model is not structured, which does not support multi-level data access. Cassandra could address this issue by using the column-oriented data model of Bigtable.

Cassandra extended the data model of Bigtable to improve the control over different portions of data. Bigtable uses column family data model where each table contains a set of rows, each of which has unique key and includes a set of columns. The columns can be grouped into different column called column families which can be used to access different groups of data. Cassandra added supper column family to the column family data model. The super column family can contain one or more column families which could be simple or super column family. This improvement gave the power to Cassandra to aggregate over different levels of data. While the data model of Bigtable provides three level data accesses, Cassandra can provide deeper data access levels by make use of super column family that can be seen as recursive map of itself.

Critical Review

The replication of data on N next successor nodes is an efficient way to make the effect of failed node locally. To recover from failing nodes, the number of replicas must be brought to N again. Since each node maintains the data the node is responsible for and the replicas of all data from N-1 predecessors, failing node has significant effect on neighboring nodes. For example, if N =3 and the size of the data stored at each node is 400 GB, then 1.2 TB of data must be transferred over the network to create replicas for the lost data. The huge network data traffic can have significant impact especially if two or more neighboring nodes fail. One expected result could be that huge amount of data would be unavailable for not short time. The neighboring nodes of the failed node could be heavily overloaded at the time of balancing the number of replicas and as result could not serve requests for the whole recovery time. I thing, keeping the effect locally is not a good idea.

One possible solution is avoiding storing replicas on neighboring nodes by using suitable mapping function to disperse replicas across different locations on the ring. For example, if the replica factor r=3, then the coordinator could store the first replica as usual on the first successor node. The second replica could be stored on the node that is responsible for the identifier computed by (coordinatorID * 2) mod 2^{32}. The last replica could be stored on the node that is responsible for the identifier computed by (coordinatorID * 3) mod 2^{32}. By this way, failing of one or more neighboring nodes might cause transferring data in different regions instead of transferring of data between a few numbers of neighboring.

Using of Gossip protocol could be another issue in Cassandra. Cassandra uses Gossip protocol to propagate data in the system to exchange information between nodes in decentralized manner to give the ability that the nodes can know each other and observe the state of the system. Based on this knowledge they can know where the different data are stored and route requests to the appropriate nodes. Although this solution gives decentralized automatic way to let the nodes aware of each other, it has two disadvantages. The system needs O (log n) time steps to deliver a message to all nodes in the system. This could be acceptable in some systems. However, in system like Cassandra which deals with millions of concurrent users could cause a consistency issue. The second disadvantage is growing the size of the exchanged message. Each node that receives a gossip message includes its state and forward the message to other nodes which will do the same producer. Therefore, the size of the message will grow gradually especially if we have system with large number of nodes. Flooding the system with messages whose sizes are always in increase could slow the system down.

One possible solution could be achieved by using the seed nodes that are already in the system to exchange states about the nodes. Any node that detects a failing node should inform one of the seed nodes that should inform only n/4 nodes about the change. The seed node sends the update message to each fourth node on the ring. Each node receives the update message from the seed nodes forward the message to its predecessor and successor nodes. By this way, the number of messages needed to deliver the new state to all nodes will be reduced and the time needed to notify all nodes about new states will be also reduced.

Another noticeable issue is how Cassandra balance load in the system. In the paper, it was mentioned that light loaded node can be moved to the middle of the region managed by heavily loaded node to take part of the data and reduce the load on the heavily loaded node. This solution is inefficient for the following reasons.

First, moving the light loaded node to the region of heavily loaded node could reduce the load of the overloaded node, but it will increase the region of its pervious successor that might be overloaded as result of removing its predecessor. To avoid this problem Cassandra has to find node that does not cause overloading of its successor if it is removed.

Second, moving one node from one location to new location on the ring causes transferring of data in the original location and also in the new location. Since each node stores data of its own region

and replicas from other nodes, removing that node from its location will cause replicating of all the data this node holds. Furthermore, joining this node to new region will leads to receiving data from r – 1 predecessor nodes.

Third, Cassandra has to find the optimal target location to which the lightly loaded node should move to. This procedure can be done based on the capacities of the both nodes. If the capacity of lightly loaded node is double as the capacity of overloaded node, Cassandra has to find location that enables the joined node to hold ¾ of the total load of the overloaded node. It means some measurements must be done to find the best location the lightly loaded node should move to.

Finally, balancing load could also require moving of two or more lightly loaded nodes to the region of overloaded node when the available lightly loaded nodes have poor capacity. Also, the lightly loaded nodes must be not neighboring nodes so that their movement will not cause overloading at another node. In such situation, Cassandra has to do much jobs under some constrains to balance load in the system.

As possible solution for this issue the concept of virtual nodes can be used with node movement solution to balance load in more efficient way. This concept is already used by Dynamo to load balance to exploit the heterogeneities of the nodes. Cassandra uses the same design model of Dynamo. The nodes are organized in ring and consistent hashing is used distributes the data across the nodes on the ring. Therefore, the concept of virtual machines can also be used in Cassandra. The solution of virtual nodes reduces the need of node movement from one location to another one on the ring and makes use of the heterogeneities of the nodes.

The powerful physical machine can be split into a number of virtual machines that could take different locations on the ring. The number of virtual machines at each physical node depends on its capacity. By this way heterogeneity is exploited and physical nodes holds data according to their capacities.

Also, the system could have some idle virtual nodes that can be used to balance load in case of failing nodes that cause overloading at other nodes. These virtual nodes can be assigned to the region of overloaded node with minimum cost. Since these virtual nodes are idle, assigning them to another region will cause transferring of data only in the target region. Also, several virtual nodes can be assigned the region of the overloaded node to distribute the data on more than one node. Moreover, Failing of physical node holding several virtual node will not have significant impact on the system because the virtual nodes will be placed on different locations on the ring and therefore balancing the number of their replicas will be performed on nodes assigned to regions of different locations of the ring.