Gurjus Singh

MSDS 432 Foundations of Data Engineering

July 26th, 2020

Module 5– Reading Comprehension

1. **What is Leader-Based Replication? Give some examples and discuss some problems. (4 pts)**

   In order to understand leader-based replication, one needs to understand what replication is. Replication is the process of copying data on different storage machines. The reason you will want to replicate is based on geographical constraints to keep users in close proximity, to keep workers productive in case work is lost on one machine and to help make queries faster for reading.

   With this definition Leader-Based Replication is when one node is used to write and read on which is called the "leader". The leader after collecting the data from the user, uses the data and passes on to its followers which are the other nodes or in this case is known as "replicas". The way the leader passes its data is through what is known as a replication log or change stream. The followers take the data from this replication log. The user can also use the replicas to read from. The replicas are not used to write the data though, only the leader has the written data first. One example of this is when a person wants to add a data into a database such as a comments or numbers. The data will first be stored in locally on the leader machine who will then send its data to the log for other replica nodes to take. Then the leader will wait for the response from the first follower, before saying request successful to the user.

   An important thing to note while thinking about leader-based replication is asynchronous and synchronous. It is synchronous in the context when the leader node, passes the data it waits for follower 1 to say OK I have been updated, and it is asynchronous in the sense that it does not wait for the other followers to say that.

   There is also what is known as known as Multi-Leader. This involves more than one node that is indicated as the leader. One leader can basically act as a follower to the other if one leader only receives the write from the user. Multi-Leader Replication is best used in a situation where there are multiple data centers. The leader can be in each data center. Some examples of Multileader replication include such as when a user uses a calendar app on a device. When the user makes a write request such as a new event, the device could be offline, in this case the write is save on the device it was created on and is called the leader, and when the person has access to the internet, then the person could sync this with their replicas otherwise known as other devices. Another example of multi-leader is when when using an application such as Google Docs. The user making the edit is on the leader node, while the users that are invited are on their own computers which can be called the replica nodes.

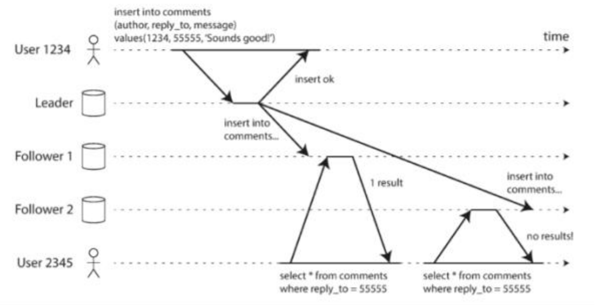
Some problems with leader replication are is when the leader fails what do you do then? This is called failover, and the solution to this is to promote a new leader from the set of followers. The user also needs to be able to update the new leader, so this has to be reconfigured in advance. There are also what are known as replication lags. One way this can happens is when a user wants to read data as soon as possible, and the data is not there. Another way this can happen is when data is not update entirely when looking at different nodes. Thirdly replication lags can happen when a node is ahead in data writes than another node. Problems that can occur on multi-leader replication are conflicts. These conflicts can be when users change data to different values. Another problem when a booking system accidentally book two people for the same room.

2. **Define, compare, and contrast the following types of Multi-Leader Replication Topologies (3 pts):**
   - **Circular Topology**
   - **Star Topology**
   - **All-to-All Topology**

   Before dining these types of topologies, replication topology describes the pattern in which send data writes to each node. There are several ways these can be accomplished such as Circular Topology, Star Topology and All-to-All topology. First Circular Topology is defined as each node receiving writes from one node and forwards it to the other and writes of its own. Star Topology is described as one root node receiving the data and forwards it to all the other nodes. It is similar to the tree. Thirdly All-to-All is described as every leader sending its write to every other leader.

   Some similarities occur between circular and star topologies since the data needs to pass through all the nodes, there should be a unique identifier, so the data is not passed through the same node twice. One problem between the two topologies is since nodes can fail, this can disrupt the flow of the how the topologies work. The topologies have to fixed manually in this case, and until they are fixed there is little communication that can be done. Since all-to-all is different, this problem is not prevalent because there can be many different paths that can be selected in topology architecture. All-to-all topology has some issues where some networks are faster than others. This can cause issues where user writes can be accepted earlier than others.

3. **Explain what is going on in the following architecture and what we can do to fix it. (3 pts)**


insert into comments
(author, reply_to, message)
values(1234, 55555, 'Sounds good!')
time
User 1234
insert ok
Leader
insert into comments...
Follower 1
1 result
insert into comments...
Follower 2
no results!
User 2345
select * from comments where reply_to = 55555
select * from comments where reply_to = 55555

The diagram shows what happens when once wants to read after inserting data. The user 1234 is basically inserting a comment in the database. The comment is then written to the Leader which is then passed to the followers, and the leader basically processes OK as soon as Follower 1 is updated. This involves synchronous updates where the leader waits for the follower to update as well as asynchronous where the leader does not have to rely on follower 2 to continue notifying the user.

Later on, User 2345 tries to read what User 1234 wrote and inserted into the database. He does several reads on two replicas only to find one replica being updated while the other does not have any sign of the data. This is part of a problem called replication lag, because the replication has not received the write. The solution to this problem is called monotonic reads.

This is a guarantee that this kind of anomaly does not happen. This means that the user will not see that one replica/node was updated before the other, which could cause confusion. One way to guarantee monotonic reads is to consistently make the user read from the same replica always. This replica can be chosen based on hashing where the USER ID is hashed to a particular node where the queries statements can go to. This has to be updated if there is a sudden node failure though.