Ownership Transfer Towards Local Access in Geo-Replicated Database Systems

Paper Number: 344

A proof of correctness

We first set up a concept of "range ownership", rather than record ownership we mentioned before: if key_1 belongs to region A, and on A, the smallest key that larger than key_1 is key_2, then we say, $[key_1, key_2)$ belongs to A.

A range can be changed from one group to another group, can be splitted into two new range, or can be combined with its preceeding range by a delete operation.

We ensure that region A can only operate on the range $[key_1, key_2)$ if it's the leader of the range.

What we are trying to provide is that, at each time point, there is only one leader taking control of the range. Note that due to stale replay, there might be no region to be the leader of a specific range at a given time point.

The proof sketch is like this:

- switch maintains the bound of the range.
- after ownership switch, any further operation on the range on the new leader is sorted behind the switch
- for each ever existed range, all its involved transactions has a total order.
- at any timepoint, there is only up to one leader any given subrange.

A.1 Changing range ownership

The change of a range $[key_1, key_2)$'s ownership happens when:

- Changing the group ID of key_1;
- Deleting key_1 causes the range to be combined.

The first one is easy to understand. The deletion case causes the range to be combined with its preceeding range, thus the range is taken by the preceeding range's leader, if that leader replayed the deletion. Note that due to staleness of replay, there might be no region to be the leader of the deleted range.

Our range ownership switch requires the leader to lock the range $[key_1, key_2)$ by locking key_1, then do the update of group ID or deletion. Thus the switch made sure that if there is only one leader of key_1 before the switch, the leader is still unique after the switch.

What we are missing is that, the upper bound the new leader sees may be different with the origin leader. If staleness causes the upper bound to be bigger than before, i.e. the new leader failed to see the insertion of key_2 after it becomes the leader of a range that key_1 is in it, then there may be two regions both be the leader of the range's subset $[key_2, key_3)$.

A.2 CRUD Transaction

Here we describe how we perform normal transactions.

Read/Update/Delete To read, update, or delete key_1, the region A must be the leader of the range key_1 belongs to and lock it. In other words, key_1 must belong to A.

Absence Ensurance To ensure a key key_1 does not exist, the region A must be the leader of the range key_1 belongs to. This is exactly what the prev key protocol does.

Insert To insert a key key_1 on region A, the region must first ensure the absence, lock the range that key_1 belongs to, and do the insertion. The insertion can be considered as splitting the range $[key_0, key_2)$ into $[key_0, key_1)$ and $[key_1, key_2)$. The two new ranges trivially still belong to region A.

A.3 Dependency Build-up

We use dependency to describe the order between transactions. All transactions inside a Paxos group is trivially ordered, thus the heading transactions is depended by tailing transactions. Depdendency is also built across groups. If transaction T_1 depends on T_2, during each region's transaction replay, T_1 cannot be replayed until T_2 is commited.

Dependency in Ownership switch Our method ensures that, for any transaction started on region A that envolves $key_1 \in [key_0, key_2)$ with A is the leader of $[key_0, key_2)$, or changes the ownership of $[key_0, key_2)$, must be depends on the transaction that changed or created the range to A.

The proof is quite straight forward. Consider what is the operation. For update, read, delete and ownership switch, the IsDepended Flag ensured it. For insert operation, if key_1 is changed here by changeGid of key_0, the dependency is built by IsDepended flag. Otherwise, the key_1 is changed to A by deleting key_i with $key_0 < key_i < key_1$, then the occurance of mark deleted key key_i when seeking prev key would make sure that the dependency is built.

Dependency for range upper bound If on region A, a range $[key_1, key_2)$ is observed belong to A, then any further operation(ownership switch, split) on that range would be depended on key_2's insertion.

The proof uses induction. Let's consider how the range is built here. If $[key_1, key_2)$ is built by splitting $[key_1, key_3)$ to $[key_1, key_2)$ and $[key_2, key_3)$, then for any further operation on $[key_1, key_2)$, its location in Paxos log is behind the insertion of key_2, thus have the dependency.

If $[key_1, key_2)$ is built by splitting $[key_0, key_2)$ to $[key_0, key_1)$ and $[key_1, key_2)$, by induction, the splitting is depended on the insertion of $[key_0, key_2)$'s upper bound key_2. Since further operation on $[key_1, key_2)$ is sorted behind the splitting(insertion of key_1) in Paxos log, and is also depended on the insertion of key_2.

Consequently, if range $[key_1, key_2)$'s ownership is switched to a new region, even incase of staleness, the new leader still must sees the insertion of key_2, thus the new leader won't mistakenly sees an extended range that shouldn't belong to it.

A.4 Conclude

At the initial status, there is a common prev key in the database, aka there is only one range in the database, initially belongs to the first region. As the system runs on, for each range, that ever existed, all its involved transactions has a total order, since the ownership switch ensured the dependency, and the bound is maintained.

Thus, at any given time point, there is only up to one leader of any subrange, since for each ownership switch, the leader first give up the ownership, thus there won't be more than one leader at the same time.

Now lets go back to focus on each single record.

Theorem 1. *Consider two transaction T_1 and T_2 submitted to two sites A and B for inserting k, respectively. Let k_1 and k_2 be the previous keys T_1 and T_2 attempt to seek and lock, respectively. Assume T_2 successfully locks k_2 and verifies the non-existence of k. Then, T_1 will eventually see the missing changes of keys in the range of $[k_2, k]$.*

Proof. Since T_2 locked k_2 and verified k's absence, the region B that T_2 is on must be the leader of range $[k_2, k_3)$ with $k < k_3$. As we proved before, the region A that T_1 is on can not be the leader of any range that intersect with $[k_2, k_3)$.

If $k_1 > k_2$, then when T_1 tries to change k_1, it will eventually sees the deletion of k_1, thus the prev key becomes another key. If $k_1 < k_2$, then when T_1 tries to change k_1, it will eventually sees the upper bound of the preceeding range of $[k_2, k_3)$, that is, k_2. So both case leads to T_1 sees k_2 as k's prev key, i.e. T_1 will eventually see the missing changes of keys in the range of $[k_2, k]$. □

Theorem 2. *At any point in time, for each record, there exists at most one Paxos group managing it.*

Proof. This stands trivially since for each range, there exists at most one Paxos group managing it. Consider the range that starts with the given record, and there is at mose one Paxos group managing it if we set the range length small enough, i.e. there is at mose one Paxos group managing the record. □