Auto-sharding: Leader election v1.0

Patches:

- https://review.opendev.org/#/c/667030
- https://review.opendev.org/#/c/667579

History

When auto-sharding is turned on currently, we have a very simplistic leader election. Am I node 0. But there are many edgecases where this doesn't work. Working in an eventually consistent system means there can be the possibility of 2 node 0's.

What now

Turns out when we created the idea of composite rings we added a ring version to the ring builder, which is just an incrementing int. When I first pushed up the patches above it included pushing this ring version into the ring itself, which has now been cut out and merged in.

What good is that? Well one of the major problems with our current approach is a node with an older ring, who should really now be a handoff node, thinks their node 0, so finds shards and inserts them into the shard table.
By introducing the ring version into the mix, we suddenly have a new dimension, time, so we can filter out these handoff nodes if their ring is too old.

This means we can solve the major issues we have with the current approach. That's what version 1 is doing. There are still some edge cases we need to work through, some known, some unknown.

Solution

ELECT verb

To help with leader election the container-server has grown a new verb called ELECT.
It is a way to basically ask a container-server, who do you think node index 0 is for a given partition. It returns a json response with the node index 0 and some extra metadata that is useful for the decision making process.

Because one of the major pieces of metadata required is the ring version, this introduces a problem. We've tried to keep loading the ring out of the storage daemons, as the ring can be quite large, so don't want it loaded for each request. But we need the ring version.
So instead, the ring is only loaded on an ELECT verb, so only loaded when we need it.

An ELECT call returns a json response in the form of:

```
{"node": <index 0 node dict from ring>,
 "part": <ring partition>,
 "status": <db status>,
 "version": <ring version>}
```

Status in the returning dict, is either the db_status from the brocker, or if it is actually a handoff node it will be NOTFOUND. This will allow us to potentially target handoff nodes if required. The rest of the json is pretty self explanatory.

Quorum

We're going to start off _very_ conservative. We have 3 types of quorums, listed below, we're going to initially start with the most conservative. I.e. be slow to shard but, hopefully, always do it correctly when we do.

Quorums:

1. ALL Quorum - All primary nodes must agree (q = n)
    a. this is what we'll use by default and until we think we've worked out all the kinks.
2. Majority Quorum - (q = n/2 +1)
3. Quorum - (q = n/2)

Election Algorithm

It's basically a mix of version 1 and version 2 we've had in the past. Please ask if you want me to rehash it all.. You've been warned :P

The current implementation goes something like this:

1. Send an ELECT call to all other primaries and gather responses.
2. Decide if I am the scanner based on the responses, including my own. These entail:
    a. False, if number of responses is < quorum
    b. If there is more than one (>1) index 0 node (comparing only IP and PORT) then filter the responses to only those with the latest ring version [0].
    c. Only if we're not sharding in batches (scanner_batch_size < 1):
        i. False, if any of the responses are in SHARDING or SHARDED state.
        ii. False, Make sure at least a quorum of responses are UNSHARDED[1]
    d. True, if there is still a quorum of responses that think it's me[2]

e. otherwise return false

[0] - If there is a new ring version it might not affect our primaries, so if they all agree it's me regardless of ring version then great.

[1] - At the moment we on ALL quorum, if this ever changes we want a quorum of primaries to agree noting that we'd have bombed out if we found any in the SHARDING or SHARDED state. Why this then, if we hit a node that doesn't have the DB (either because it's new or it's replicated it away already) then the db state will be NOTFOUND. But it still has a ring so should know who it thinks index 0 should be, so maybe it's vote still means something.

[2] - This guy is convinced he is index 0, so just confirming, if it isn't then _elect_leader will return false (ie well it isn't me then).

Things you may need to be aware of

Currently the auto-sharding has been implemented to search for x shards, add them, and then next round find the next x. So scanning takes a while.

With the OP driven sharding, using the tool, how we currently recommend running sharding, we scan for _all_ the shard ranges and then insert them.

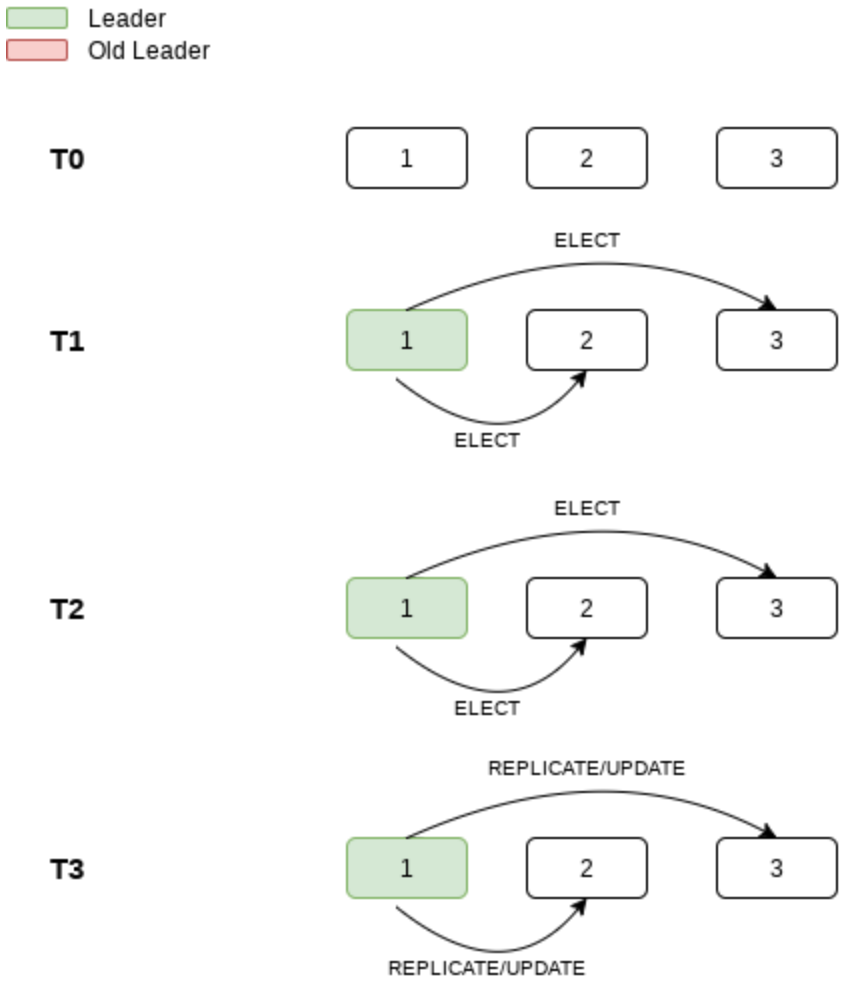
This seems to be working even on large containers. I wonder if we should do the same in auto sharding? Scan for them all, check to see if we are still the leader (the authority on shard ranges) and if so insert them.

I could imagine at some point we can turn on auto sharding for new containers. All old containers should either need to be turned on or even needs the OP tool to find and intially set the shards as a way to deal with large containers (so the scan can happen even more async).

By inserting them all, and not in batches, using the db state we can make a decison. I.E if one of the other primaries are already in the the SHARDING or SHARDED state then the election could decide to fail as someone else has inserted something (an OP or a split brain leader election thing).

Election in pictures

The good, normal path



In this example, there is no rebalance or old node coming online, so it's straight forward:

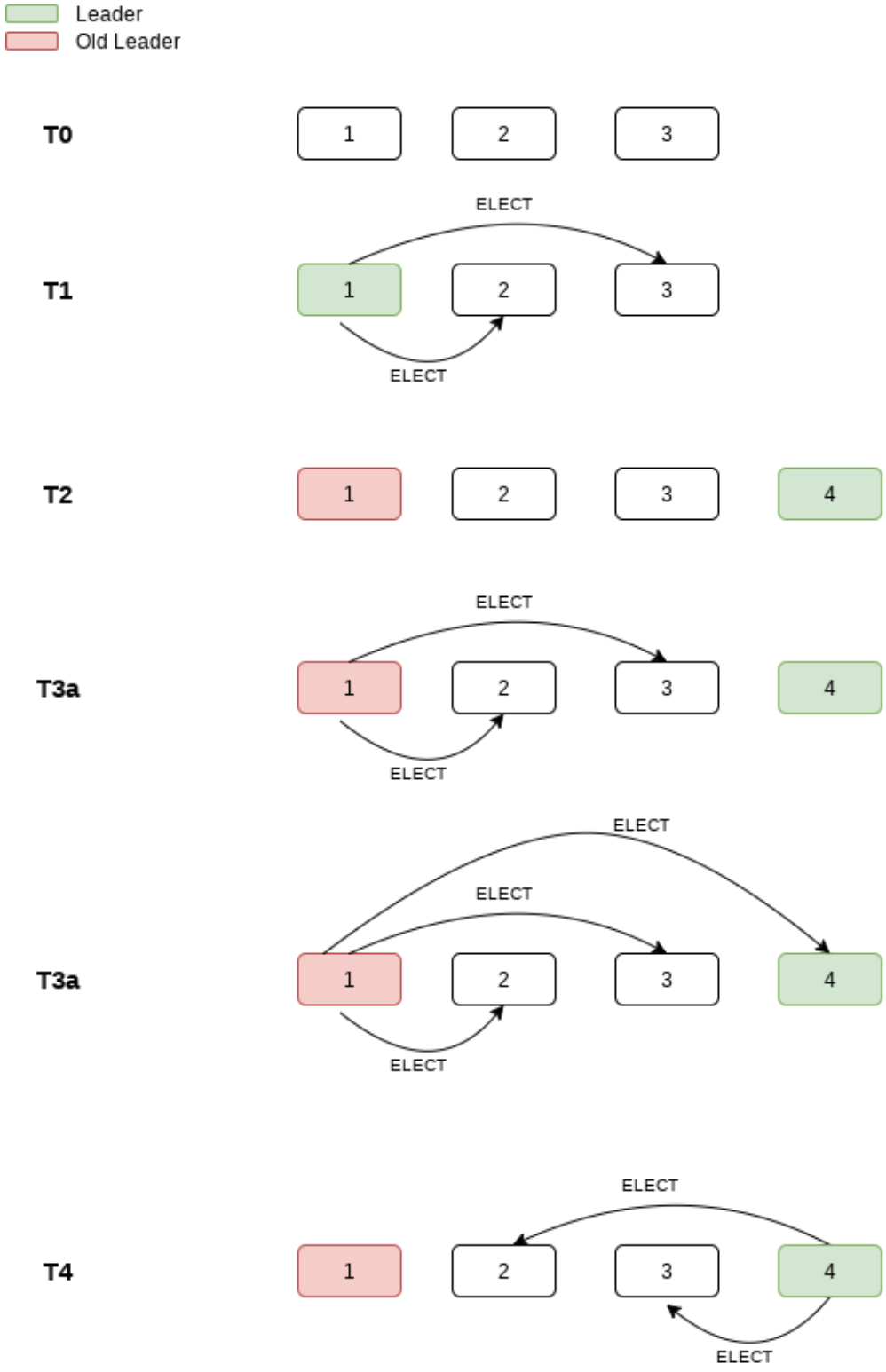
- T0 - This is a 3x replication container ring. We have 3 nodes, node 1, 2 and 3.
- T1 - Node 1 happens to be index 0 in the ring for a certain partition, it makes ELECT calls to the other primaries, in this case everyone agrees node 1 is the leader, so it goes off to scan for ranges.
- T2 - Once it's scanned and found all (or a batch) of ranges, it does another leader election, to make sure it's still the leader.
- T3 - We are still the leader, so we REPLICATE/UPDATE the shard ranges. This could either be insert locally and then call replicate (What we do now) or via UPDATE verb to push them directly into other primaries DB first, and then locally. In the latter, if the

UPDATE fails on _all_ nodes we can roll back (i.e not race the replicator and actually roll back) and try again from the beginning next time.

Other Notes:

- RE: T2
    - Maybe we can have a timeout, as most scans could happen very quickly. But in a large container this can take a long time. So we scan again to double check we are still the leader before dumping.
    - This means we do the election twice, and in any fail we dump and have to start again from scratch (well ATM).
    - So this could cause a bunch of extra requests on the cluster.. But it does make it safer. I think.

Rebalance happens and moves node 0 out of index 0 for the part



In this example, we start the election, but while we have a leader scanning a rebalance happens that just so happens moves index 0 of this part to a new node. Meaning suddenly there is a new index 0:

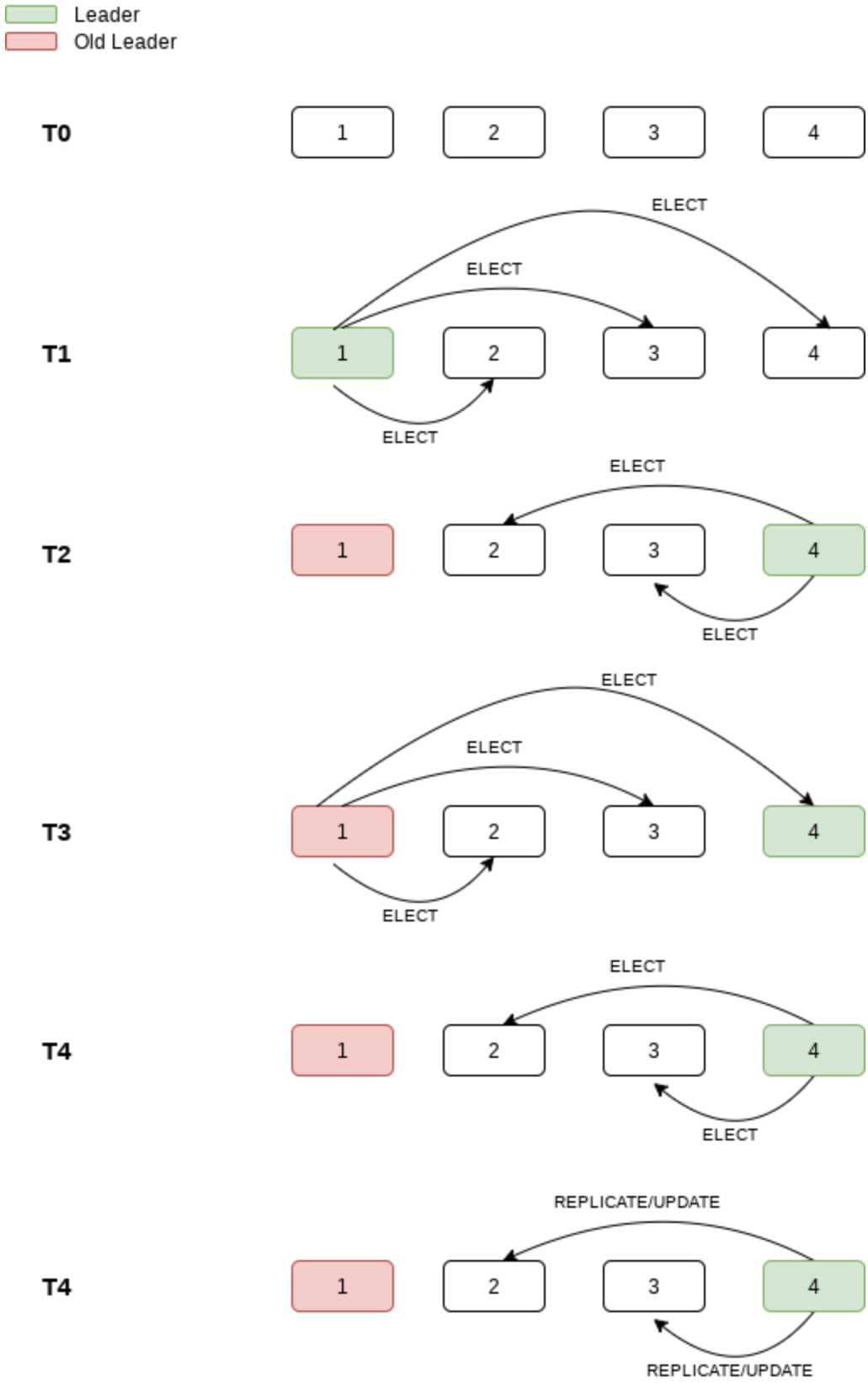
- T0, T1 - Are the same as the last example.
- T2 - While node 1 is scanning, the rebalance happens, node 0 is now an old leader and there is a new sheriff in town, node 4.
- Now there are 2 (but probably more) possibilities that could happen:
    - T3a - Node 1 finishes scanning, but is still running the old ring. Goes back to his view of the partition and reruns the election on the other 2 primaries. We hope here that one of them have an updated ring, and report back node 4, so the elect fails; or
    - T3b - Node 1 happens to have a new version of the ring, so makes a call to the now 3 primaries which will obviously fail.
- T4 - The sharder daemon on node 4 runs, and as node 4 is the new index 0, starts the process.

Notes

- At T3 here, this is one place we need to be careful. What if in T3a all old primaries have the old ring, so the shards are inserted. In this case, if were searching all the (SHARDING/SHARDING) check will stop any new shards from node 4 being added.
    - Though maybe batching is ok too.. It is batching after all. Probably need a set of probe tests.

Change in replica count during scanning

This is another way we could have 2 leaders at once. I guess for the following diagram to work the ring at part x would be: (node1, node4, node2, node3)
But you get the idea.


Leader
Old Leader
T0
1
2
3
4
T1
ELECT
ELECT
ELECT
T2
ELECT
ELECT
T3
ELECT
ELECT
ELECT
T4
ELECT
ELECT
T4
REPLICATE/UPDATE
REPLICATE/UPDATE

This time:

- T0 - We now have 4x replication, well to start with. Nodes 1, 2, 3 and 4 are the primary nodes.
- T1 - Node 1 is index 0 so it does an election, gets a good response so starts scanning for ranges.
- T2 - There is a change in replication number and a rebalance, now node 4 is index 0, and it starts scanning. It does a leader election. I guess in this case nodes 2 and 3 have an updating ring, otherwise election will fail and have to wait a round.
- T3 - node 1 comes back, found the ranges, so does a new election, and finds out it isn't the leader, so just throws away the ranges (maybe we should do something smart here, so there's less wasted work, but initial keep it simple).
- T4 and T5 and now like example 1.

Other gotchas

Obviously what happens in the case where a rebalance happens after the second ELECT. The old leader will write the shards, is there a change the new one could do something as well?
In this case, this is where full scans and the SHARDING/SHARDED check could come into play and save us. As new leader ELECT will fail, and shards will replicate into the new leader.

Other suggestions

From the Swift team meeting:

```
<rledisez> it's probably something to keep in mind for the
auto-sharding. a limit on the number of containers being sharded at a
time
<timburke> could probably even have an optimization where it goes to
check recon dumps *first* to see what's currently sharding, then go
straight to the DBs... skip the treewalk. hmm...
<rledisez> the best would be to estimate the size of each shard and
check there enough space on the devices holding these shards
<rledisez> timburke: totally, that's what we do know: for db in $(cat
| jq … Í …); do
```