Several major causal cluster operations require a majority of cluster members to be online, a majority quorum. When a causal cluster loses majority quorum, it loses write capability as well as the ability to add or remove members of the cluster.

This article explains the reasoning for this behavior, and how it impacts recovery from certain cluster issues.

**Quorum commits and data durability**

When we lose quorum, the cluster goes into a read-only state and no longer has a leader to perform writes.

This behavior is part of the Raft protocol that backs causal clustering, and is in place to ensure data durability and maintain database ACIDity.

Since we have quorum commits, when quorum is lost, in the worst case scenario the quorum of nodes with the latest data are all offline, and none of the online nodes have the latest commits.

It would be a mistake to allow writes until we restore quorum and ensure the cluster reflects the latest committed data.

**Quorum vote in/out and cluster recovery**

Quorum is also needed to vote in and out cluster members, and this affects both cluster size scaling and options available for cluster recovery.

Note that cluster membership is separate from a cluster node’s online or offline state. It is possible for a node to be a member of the cluster yet be unreachable or offline.

While an unreachable or offline node may trigger a vote-out attempt to remove the node from cluster membership, the vote-out operation has two requirements:

1. That the membership count of cluster members is above causal\_clustering.minimum\_core\_cluster\_size\_at\_runtime.
2. That there is a majority quorum present to pass the vote.

As discussed in the article on [cluster size scaling](https://support.neo4j.com/hc/en-us/articles/360012314414-Understanding-causal-cluster-size-scaling), a cluster can gracefully scale down in size down to the configured causal\_clustering.minimum\_core\_cluster\_size\_at\_runtime (default 3), provided that quorum is maintained.

As clusters scale down (or up!) in size, the number of online nodes required for majority quorum will change according to the cluster membership size.

When we lose quorum, nodes that went offline/unresponsive that were responsible for the loss of quorum must be restored in order to regain quorum. Remember, even though these members may be offline, they are still counted as members of the cluster, as they were never voted out due to the loss of quorum.

We cannot add in new cluster members in order to recover quorum, since quorum is required to vote in new cluster members.

While this may seem restrictive in that we cannot simply add new nodes to a cluster to restore normal operation, this behavior still supports data durability.

In the worst case scenario as discussed in the previous section, when quorum is lost, there is the possibility that only the offline nodes contain the latest committed data (due to quorum commits).

If we incorrectly allowed new nodes to be added to restore quorum, we could restore write operations, but data would be lost with the offline members. Even if we were to later restore these members, in the meantime the new commits may be in conflict with the lost data, resulting in either duplicated or inconsistent data, with no clear way to resolve the conflict.

Requiring quorum for both commits and cluster vote in/out operations is required to maintain data durability in the cluster.

**Example log messages for vote in/out operations**

Raft membership operations appear in the debug log, here are a few examples of what you might see.

We’re using a causal cluster with up to 6 core members. For this configuration, we’re using discovery ports 5001 through 5006 for each member accordingly, and raft listen addresses of 7001 through 7006 accordingly.

At this point in time, we have 4 core nodes online, corresponding with nodes 1, 2, 3, and 5. At this point we attempt to start node 6.

In the debug logs, we’ll see multiple debug messages, but these in particular show discovery of new member and a (successful!) attempt to add the new member to the cluster:

2019-03-29 21:57:59.670+0000 INFO [o.n.c.d.CoreMonitor] Discovered core member at localhost:5006

2019-03-29 21:57:59.676+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Core topology changed {added=[{memberId=MemberId{416eca31}, info=CoreServerInfo{raftServer=localhost:7006, catchupServer=localhost:6006, clientConnectorAddresses=bolt://localhost:7667,http://localhost:7464,https://localhost:7463, groups=[], database=default, refuseToBeLeader=false}}], removed=[]}

2019-03-29 21:57:59.676+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Target membership: [MemberId{1b45d851}, MemberId{cf17e1cf}, MemberId{2011b3fb}, MemberId{3e6dbf35}, MemberId{416eca31}]

...

2019-03-29 21:58:14.806+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Getting consensus on new voting member set [MemberId{2011b3fb}, MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}, MemberId{416eca31}]

2019-03-29 21:58:14.806+0000 INFO [o.n.c.c.c.m.RaftMembershipChanger] ConsensusInProgress{}

2019-03-29 21:58:14.813+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Appending new member set RaftMembershipState{committed=MembershipEntry{logIndex=5, members=[MemberId{2011b3fb}, MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}]}, appended=MembershipEntry{logIndex=6, members=[MemberId{2011b3fb}, MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}, MemberId{416eca31}]}, ordinal=7}

If we stop node 6, we’ll see the member set change accordingly as the rest of the cluster votes node 6 out of the cluster:

2019-03-29 22:09:40.191+0000 WARN [o.n.c.d.CoreMonitor] Lost core member at localhost:5006

2019-03-29 22:09:40.203+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Core topology changed {added=[], removed=[{memberId=MemberId{416eca31}, info=CoreServerInfo{raftServer=localhost:7006, catchupServer=localhost:6006, clientConnectorAddresses=bolt://localhost:7667,http://localhost:7464,https://localhost:7463, groups=[], database=default, refuseToBeLeader=false}}]}

2019-03-29 22:09:40.203+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Target membership: [MemberId{1b45d851}, MemberId{cf17e1cf}, MemberId{2011b3fb}, MemberId{3e6dbf35}]

2019-03-29 22:09:40.204+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Getting consensus on new voting member set [MemberId{2011b3fb}, MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}]

2019-03-29 22:09:40.204+0000 INFO [o.n.c.c.c.m.RaftMembershipChanger] ConsensusInProgress{}

2019-03-29 22:09:40.209+0000 INFO [o.n.c.m.RaftOutbound] No address found for MemberId{416eca31}, probably because the member has been shut down.

2019-03-29 22:09:40.209+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Appending new member set RaftMembershipState{committed=MembershipEntry{logIndex=6, members=[MemberId{2011b3fb}, MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}, MemberId{416eca31}]}, appended=MembershipEntry{logIndex=7, members=[MemberId{2011b3fb}, MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}]}, ordinal=9}

In both cases, you can see that membership changes are implemented as consensus commits.

Example log messages when dropping below minimum core size at runtime

In this scenario, nodes 1, 3, and 5 are up for a cluster of 3. We will stop node 3 and observe the log messages:

2019-03-29 22:12:20.280+0000 WARN [o.n.c.d.CoreMonitor] Lost core member at localhost:5003

2019-03-29 22:12:20.281+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Core topology changed {added=[], removed=[{memberId=MemberId{cf17e1cf}, info=CoreServerInfo{raftServer=localhost:7003, catchupServer=localhost:6003, clientConnectorAddresses=bolt://localhost:7637,http://localhost:7434,https://localhost:7433, groups=[], database=default, refuseToBeLeader=false}}]}

2019-03-29 22:12:20.281+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Target membership: [MemberId{1b45d851}, MemberId{3e6dbf35}]

2019-03-29 22:12:20.281+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Not safe to remove member [MemberId{cf17e1cf}] because it would reduce the number of voting members below the expected cluster size of 3. Voting members: [MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}]

The target membership of 2 nodes is not achievable, since this would reduce the number of voting members below the runtime cluster size of 3. So even though node 3 is offline, it cannot be removed from the membership set. Quorum size remains at 2/3 nodes for majority, which we maintain since 2 cluster nodes remain online. In the logs we would likely see connection attempts to the offline node, since it’s still considered a cluster member even if offline.

If we remove one more node we will lose quorum:

2019-03-29 22:23:05.055+0000 WARN [o.n.c.d.CoreMonitor] Lost core member at localhost:5005

2019-03-29 22:23:05.056+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Core topology changed {added=[], removed=[{memberId=MemberId{1b45d851}, info=CoreServerInfo{raftServer=localhost:7005, catchupServer=localhost:6005, clientConnectorAddresses=bolt://localhost:7657,http://localhost:7454,https://localhost:7453, groups=[], database=default, refuseToBeLeader=false}}]}

2019-03-29 22:23:05.056+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Target membership: [MemberId{3e6dbf35}]

2019-03-29 22:23:05.056+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Not safe to remove members [MemberId{1b45d851}, MemberId{cf17e1cf}] because it would reduce the number of voting members below the expected cluster size of 3. Voting members: [MemberId{3e6dbf35}, MemberId{1b45d851}, MemberId{cf17e1cf}]

We would likely see this accompanied by a stepdown event, as we cannot have a leader or write capability without quorum:

2019-03-29 22:23:20.100+0000 INFO [o.n.c.c.c.s.RaftState] Leader changed from MemberId{3e6dbf35} to null

...

2019-03-29 22:23:20.101+0000 INFO [o.n.c.c.r.RaftReplicator] Lost previous leader 'MemberId{3e6dbf35}'. Currently no available leader

2019-03-29 22:23:20.101+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Step down event detected. This topology member, with MemberId MemberId{3e6dbf35}, was leader in term 2, now moving to follower.

We would likely see this accompanied by connection attempts being made to the two offline cluster members, along with failed elections as quorum isn’t present.

Let’s see what happens if we attempt to start node 6 at this point. Remember, node 6 has already been voted out of the cluster, so it isn’t considered a cluster member, and quorum of cluster members isn’t present, so we should see the join attempt fail:

2019-03-29 22:29:41.003+0000 INFO [o.n.c.d.CoreMonitor] Discovered core member at localhost:5006

2019-03-29 22:29:41.004+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Core topology changed {added=[{memberId=MemberId{416eca31}, info=CoreServerInfo{raftServer=localhost:7006, catchupServer=localhost:6006, clientConnectorAddresses=bolt://localhost:7667,http://localhost:7464,https://localhost:7463, groups=[], database=default, refuseToBeLeader=false}}], removed=[]}

2019-03-29 22:29:41.004+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Target membership: [MemberId{3e6dbf35}, MemberId{416eca31}]

The topology and target membership messages show that the new node has been detected and wants to join in the cluster, but we do not see the consensus messages on the new voting member set or the appending of the new member set with a commited MembershipEntry.

Since we don’t see the consensus messages or the commit membership entry messages, we know that node 6 was not able to successfully join the cluster.

If we restore one of the offline member nodes and reestablish quorum by having the majority of cluster members online, we will see the consensus and committed membership entry messages resume as quorum would allow the cluster to start voting in and out cluster members once again, and the quorum size would change accordingly as the membership set changes.

**How neo4j-admin unbind impacts causal cluster recovery**

Several recovery procedures for causal cluster issues require usage of neo4j-admin unbind, which deletes cluster state, and is usually executed in concert with deleting, overwriting (such as from a restore) or moving the graph.db on the instance. Since the graph data is being changed, the cluster state no longer reflects the state of the store, thus the unbind operation is necessary.

It is important to understand how this affects cluster membership, and how in certain situations this may impact cluster recovery:

**A node which has had its cluster state unbound cannot be used to restore quorum (and thus write operation) to a cluster.**

When you execute neo4j-admin unbind, because that node’s cluster state is being destroyed, its identity in the cluster is also destroyed. When the node is brought back online it will have a new member id and appear as a brand new node to the cluster.

If the cluster currently has a majority quorum of cluster members online, then there should be no negative consequences, as quorum should allow the newly recovered node to be voted into the cluster.

But if the cluster does not have quorum, then the newly recovered node cannot be voted in and cannot contribute to stabilizing the cluster. Even if the recovered node’s previous member id is still counted as a cluster member (has not been voted out of the cluster), the node cannot assume its previous identity, and cannot pass itself off as a current member. And it would not be correct to allow such to happen.

This behavior is intended, and is part of the Raft implementation which contributes to data durability. Without this behavior, then there could be scenarios that could result in data loss or data conflict in the cluster.

When a cluster loses quorum, the only way to restore it is to get one of the current offline members (which hasn’t been voted out) to rejoin the cluster, and neo4j-admin unbind must not have been used on the member.

If quorum has been lost, and there is no way to bring any current offline members back online without usage of neo4j-admin unbind, then you may be forced to fall back to full cluster recovery procedures, which will require bringing the cluster offline. ”’

* Last Modified: 2020-10-22 22:13:22 UTC by Andrew Bowman.
* Relevant for Neo4j Versions: 3.1, 3.2, 3.3, 3.4, 3.5.
* Relevant keywords cluster.