**Causal Clustering Minimum Core Size At Formation**

causal\_clustering.minimum\_core\_cluster\_size\_at\_formation is defined as the minimum number of Core machines initially required to form a cluster. The cluster will form when at least this many Core members have discovered each other.

The following example shows a 5 core cluster with causal\_clustering.minimum\_core\_cluster\_size\_at\_formation=3. Core 1 and 2 are initialised and the first election occurs as below:

2019-03-31 14:14:22.028+0000 INFO [o.n.c.n.Server] raft-server: bound to 127.0.0.1:7000

2019-03-31 14:14:22.038+0000 INFO [o.n.c.d.CoreMonitor] Waiting for a total of 3 core members...

2019-03-31 14:14:30.910+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Cluster discovery service started

2019-03-31 14:14:30.955+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Core topology changed {added=[{memberId=

MemberId{fb0bc6e1}, info=CoreServerInfo{raftServer=localhost:7001, catchupServer=localhost:6001,

clientConnectorAddresses=bolt://localhost:7688,http://localhost:7460,https://localhost:7461, groups=[], database=default,

refuseToBeLeader=false}}, {memberId=MemberId{3811e9ed}, info=CoreServerInfo{raftServer=localhost:7000,

catchupServer=localhost:6000, clientConnectorAddresses=bolt://localhost:7687,http://localhost:7474,https://localhost:7470,

groups=[], database=default, refuseToBeLeader=false}}], removed=[]}

2019-03-31 14:14:30.956+0000 INFO [o.n.c.c.c.m.RaftMembershipManager] Target membership: [MemberId{fb0bc6e1},

MemberId{3811e9ed}]

2019-03-31 14:14:31.022+0000 INFO [o.n.c.d.CoreMonitor] Discovered core member at localhost:5001

2019-03-31 14:14:32.099+0000 INFO [o.n.c.d.CoreMonitor] Waiting for a total of 3 core members...

2019-03-31 14:14:42.141+0000 INFO [o.n.c.d.CoreMonitor] Waiting for a total of 3 core members...

2019-03-31 14:14:52.188+0000 INFO [o.n.c.d.CoreMonitor] Waiting for a total of 3 core members...

2019-03-31 14:15:02.225+0000 INFO [o.n.c.d.CoreMonitor] Waiting for a total of 3 core members..

Core 3 now joins and the cluster forms successfully:

2019-03-31 14:15:02.225+0000 INFO [o.n.c.d.CoreMonitor] Waiting for a total of 3 core members...

2019-03-31 14:15:12.281+0000 INFO [o.n.c.d.CoreMonitor] Waiting for a total of 3 core members...

2019-03-31 14:15:22.287+0000 INFO [o.n.c.d.CoreMonitor] Discovered core member at localhost:5002

2019-03-31 14:15:22.298+0000 INFO [c.n.c.d.SslHazelcastCoreTopologyService] Core topology changed

{added=[{memberId=MemberId{9274358d}, info=CoreServerInfo{raftServer=localhost:7002, catchupServer=localhost:6002,

clientConnectorAddresses=bolt://localhost:7689,http://localhost:7476,https://localhost:7472, groups=[], database=default,

refuseToBeLeader=false}}], removed=[]}

2019-03-31 14:15:22.568+0000 INFO [o.n.c.d.CoreMonitor] This instance bootstrapped the cluster.

**Causal Clustering Minimum Core Size At Runtime**

causal\_clustering.minimum\_core\_cluster\_size\_at\_runtime is defined as the minimum size of the dynamically adjusted voting set (which only core members may be a part of). Adjustments to the voting set happen automatically as the availability of core members changes, due to explicit operations such as starting or stopping a member, or unintended issues such as network partitions. Note that this dynamic scaling of the voting set is generally desirable as under some circumstances it can increase the number of instance failures which may be tolerated. A majority of the voting set must be available before voting in or out members.

Let’s try out causal\_clustering.minimum\_core\_cluster\_size\_at\_runtime=2 on a 3-node cluster. If we lose one core, the cluster still has consensus and can scale down to 2. But if we lose 1 more we’re at a single node left and lack consensus so can’t scale down, and we’re waiting for that just-failed node to become available again. At this point, if the first node of 3 that failed comes back online, it can’t be added back to the cluster since we lack consensus to add it back in. We’re stuck waiting on only the last failed node.

In the example below, core 1 (leader) sees core 3 leaving the cluster as below:

2019-03-31 15:10:31.234+0000 WARN [o.n.c.d.CoreMonitor] Lost core member at localhost:5002

At this point, the cluster still has quorum for writes done at leader, core 1. But after a subsequent period of causal\_clustering.leader\_election\_timeout (default 7s), core 3 is removed from the cluster \_because of the cluster size at runtime set to 2.

If we then take core 2 offline also, the cluster becomes read only:

2019-03-31 15:11:08.005+0000 INFO [o.n.c.c.c.s.RaftState] Leader changed from MemberId{e7f79e48} to null

2019-03-31 15:11:08.006+0000 INFO [o.n.c.c.c.s.RaftLogShipper] Stopping log shipper MemberId{a2ef543b}[matchIndex: 5,

lastSentIndex: 5, localAppendIndex: 5, mode: PIPELINE]

However, if we now add back core 3 *before* adding back core 2, we still end up with two cores in *follower* state, leaving the cluster as read-only and we see the below in core 1’s log:

2019-03-31 15:12:06.251+0000 DEBUG [o.n.c.c.c.RaftMachine] Should vote for raft candidate false: requester log up to date:

false (request last log term: 1, context last log term: 1, request last log index: 1, context last append: 5) voted for other

in same term: false (request term: 1, context term: 1, voted for another: false)

And a write transaction results in the below exception:

Neo.ClientError.Cluster.NotALeader

Neo.ClientError.Cluster.NotALeader: No write operations are allowed directly on this database. Writes must pass through the

leader. The role of this server is: FOLLOWER

It is not until core 2 is added back, that the three cores form a writeable cluster once again:

2019-03-31 15:13:04.377+0000 INFO [o.n.c.c.c.RaftMachine] Election started with vote request: Vote.Request from

MemberId{e7f79e48} {term=2, candidate=MemberId{e7f79e48}, lastAppended=5, lastLogTerm=1} and members: [MemberId{a2ef543b},

MemberId{e7f79e48}]

2019-03-31 15:13:04.377+0000 INFO [o.n.c.c.c.RaftMachine] Moving to CANDIDATE state after successful pre-election stage

2019-03-31 15:13:04.384+0000 INFO [o.n.c.c.c.RaftMachine] Moving to LEADER state at term 2 (I am MemberId{e7f79e48}), voted

for by [MemberId{a2ef543b}]

2019-03-31 15:13:04.384+0000 INFO [o.n.c.c.c.s.RaftState] First leader elected: MemberId{e7f79e48}

If we however, stick with the default cluster size at runtime of 3, then the cluster could not have scaled down to 2 (the first node that failed wouldn’t have been voted out), but we would have kept consensus and write ability. But then when the second node fails and we’re down to 1, we lose consensus and write capability, just like the previous scenario, but we’re able to get back consensus and write capability if either of the two failed nodes comes back online, not just the latest failed node.

In conclusion, the only effective difference between minimum\_core\_cluster\_size\_at\_runtime at 2 instead of the default of 3 is that when we’re down to 1 operational node (after having scaled down to cluster size of 2), we have to wait until the just-failed node comes back online, the one that failed before that can’t rejoin because adding a “new” node to the cluster requires consensus.

Having a smaller minimum\_core\_cluster\_size\_at\_runtime is therefore a more relevant optimisation when the base/resting cluster size is larger (e.g. 5). In that situation, having a minimum\_core\_cluster\_size\_at\_runtime of 3, rather than 5, allows the cluster to tolerate 3 failures, rather than 2, before losing write capability, i.e. *provided* that those 3 failures don’t happen faster than the cluster is able to vote out failing members (causal\_clustering.leader\_election\_timeout). Using 2 instead of the default of 3 doesn’t affect the ability to tolerate 1 failure in 3. There typically however, isn’t a a good reason to have it set to 2. One may however set minimum\_core\_cluster\_size\_at\_runtime to a smaller than the total number of cores in a cluster of 5 or more. ”’