

ECE 473/573 Fall 2023 - Project 3

Distributed Database Systems

Samman Chouhan
A20561414

II. Cassandra Cluster Management

Question 1: What happens if one misconfigured the CASSANDRA_CLUSTER_NAME variable in docker-compose.yml?

Misconfiguring variables in docker-compose.yml can cause complications in Cassandra cluster setup. The field names the Cassandra cluster to which each node belongs. Misconfiguring this variable can cause nodes to join several clusters, leading to data inconsistency and communication issues within the cluster. Ensure that all Cassandra nodes share the same cluster name.

Question 2: What is the purpose of the CASSANDRA_SEEDS variable in docker-compose.yml? Is it necessary to list all the servers?

The seed variable in docker-compose.yml provides the seed nodes in the Cassandra cluster. Seed nodes control cluster membership and facilitate node discovery during bootstrapping. It's important to include a few reliable servers in the variable, rather than listing all of them. Seed nodes initiate clusters and serve as interaction points for new nodes. Having a subset of nodes as seed nodes improves both stability and performance. It's advised to have two or three seed nodes, however this can vary based on the cluster's size and requirements.

Question 3: What does the UN mean for the first column of the output from nodetool status?

The "UN" in the nodetool status report indicates "Up and Normal". This status confirms that the Cassandra node is operational inside the cluster. Nodes designated as "UN" are actively participating in the cluster and can handle read and write requests. A healthy Cassandra cluster typically has nodes with the "UN" status. This indicates that they are operational and contribute to data storage and processing duties within the cluster.

Question 4: In our cluster consisting of containers, where does Cassandra store data?

Cassandra stores data in node-specific directories within a container cluster. Cassandra nodes run in independent containers (e.g., db1, db2, db3), each with its own data directory for managing keyspaces and tables.

Cassandra's data directory is commonly set in its configuration files (cassandra.yaml). In a Dockerized Cassandra cluster, data directories are often located on the container's filesystem. When using containers, make sure the data directories are properly mapped to the host machine or external storage to preserve data even if the container is terminated or deleted. This enables data to survive container restarts or re-creations.

III. Tunable Consistency

Keep both reader and writer running. In a few minutes you should be able to see that the reader reports missing data. Provide screenshots and briefly explain what is happening.

Using the same consistency level (ONE) for both reading and writing can result in missing data. Cassandra uses the consistency level to determine the number of replicas required for successful write or read operations. Cassandra uses a consistency level of ONE, requiring only one replica to acknowledge the operation.

If a writer inserts data and acknowledges it, and a reader reads from a different replica (e.g., db2), data may not be propagated to that replica immediately. As a result, the reader may not find the expected data while querying the database.

Screen shot of reader db2 and writer db1 and missing data

```
root@instance-20240822-161614:/home/samman/ece573-prj03# docker compose exec client reader test ONE db2
2024/10/21 03:36:52 Connecting cluster at db2 with consistency ONE for topic test
2024/10/21 03:36:52 Found invalid peer 'HostInfo hostname="" connectAddress="172.19.0.4" peer="172.19.0.4" rpc_address="172.19.0.4" broadcast_address="<n/a>" preferred_ip=<n/a> connect_addr="172.19.0.4" connect_addr_source="connect_address" port=9042 data_center="datacenter1" rack="rack1" host_id="97f3a90b-90aa-4178-9e21-2673570327da" version="v4.1.3" state=UP num_tokens=0' Likely due to a gossip or snitch issue, this host will be ignored
2024/10/21 03:36:52 Connected to cluster ece573-prj03
2024/10/21 03:36:53 test: seq 33257 with 33257 rows
2024/10/21 03:36:53 test: seq 33257 with 33257 rows
2024/10/21 03:36:53 test: no more data, wait 10s
```

```
root@instance-20240822-161614:/home/samman/ece573-prj03# docker compose exec client writer test ONE db1
2024/10/21 03:36:19 Connecting cluster at db1 with consistency ONE for topic test
2024/10/21 03:36:19 Connected to cluster ece573-prj03
2024/10/21 03:36:22 test: inserted 1000 rows
2024/10/21 03:36:22 test: inserted 1000 rows
2024/10/21 03:36:22 test: inserted 1000 rows
2024/10/21 03:36:22 test: inserted 3000 rows
2024/10/21 03:36:31 test: inserted 4000 rows
2024/10/21 03:36:31 test: inserted 6000 rows
2024/10/21 03:36:31 test: inserted 6000 rows
2024/10/21 03:36:39 test: inserted 7000 rows
2024/10/21 03:36:42 test: inserted 8000 rows
2024/10/21 03:36:42 test: inserted 8000 rows
2024/10/21 03:36:45 test: inserted 10000 rows
2024/10/21 03:36:45 test: inserted 10000 rows
2024/10/21 03:36:49 test: inserted 11000 rows
2024/10/21 03:36:51 test: inserted 12000 rows
2024/10/21 03:36:51 test: inserted 12000 rows
2024/10/21 03:36:56 test: inserted 14000 rows
2024/10/21 03:36:59 test: inserted 15000 rows
2024/10/21 03:37:01 test: inserted 16000 rows
```

You can terminate the reader by Ctrl+C now and then restart it with the same arguments. Does it report any missing data? Provide screenshots and briefly explain what is happening.

Restarting the reader with the same parameters may result in same lost data as in the prior scenario. Cassandra replication may not have fully caught up with write operations during the short interruption while stopping and restarting the reader.

Missing data in Cassandra may be caused by asynchronous data propagation among nodes, leading to eventual consistency. When the reader restarts, it may read from a replica (db2) that has not yet received the most recent data from the writer, resulting in missing data.

Screenshot missing data again

The screenshot shows a macOS desktop with an iTerm2 window open. The window has two tabs: 'docker-compose.yml' and 'terminal'. The 'terminal' tab contains a log of database insert operations and a Docker command. The log shows numerous 'test: inserted' entries for various row counts (e.g., 26000, 27000, 28000, 29000, 30000, etc.) and several 'test: seq' entries. A Docker command is also present at the top of the terminal log.

```
root@instance-20240822-161614:/home/samman/ece573-prj03# docker compose exec client reader test ONE db2
2024/10/21 03:38:25 test: seq 54267 with 54260 rows, missing 7
2024/10/21 03:38:25 test: seq 54269 with 54262 rows, missing 7
2024/10/21 03:38:25 test: seq 54272 with 54265 rows, missing 7
2024/10/21 03:38:25 test: seq 54274 with 54267 rows, missing 7
2024/10/21 03:38:25 test: seq 54274 with 54267 rows, missing 7
2024/10/21 03:38:25 test: no more data, wait 10s
^Croot@instance-20240822-161614:/home/samman/ece573-prj03# docker compose exec client reader test ONE db2
2024/10/21 03:38:41 Connecting cluster at db2 with consistency ONE for topic test
2024/10/21 03:38:41 Found invalid peer 'HostInfo hostname="" connectAddress="172.19.0.4" peer="172.19.0.4" rpc_address="172.19.0.4" broadcast_address="en1" preferred_ip="en1" connect_addr="172.19.0.4" connect_addr_source="connect_address" port=9042 data_center="datacenter1" rack="rack1" host_id="9f7fa90b-90aa-4178-9e21-267537d2da" version="v4.1.3" state=UP num_tokens=0' Likely due to a gossip or snitch issue, this host will be ignored
2024/10/21 03:38:41 Connected to cluster ece573-prj03
2024/10/21 03:38:43 test: seq 58116 with 58116 rows
2024/10/21 03:38:43 test: seq 58116 with 58116 rows
2024/10/21 03:38:43 test: no more data, wait 10s
2024/10/21 03:38:53 test: seq 60306 with 60306 rows
2024/10/21 03:38:53 test: seq 60336 with 60336 rows
2024/10/21 03:38:53 test: seq 60336 with 60336 rows
2024/10/21 03:38:53 test: no more data, wait 10s
2024/10/21 03:39:03 test: seq 62524 with 62524 rows
2024/10/21 03:39:03 test: seq 62527 with 62527 rows
2024/10/21 03:39:03 test: seq 62530 with 62529 rows, missing 1
2024/10/21 03:39:03 test: seq 62532 with 62531 rows, missing 1
2024/10/21 03:39:03 test: seq 62534 with 62533 rows, missing 1
2024/10/21 03:39:03 test: seq 62538 with 62537 rows, missing 1
2024/10/21 03:39:03 test: seq 62539 with 62538 rows, missing 1
2024/10/21 03:39:03 test: seq 62541 with 62540 rows, missing 1
2024/10/21 03:39:03 test: seq 62543 with 62542 rows, missing 1
2024/10/21 03:39:03 test: seq 62545 with 62544 rows, missing 1
2024/10/21 03:39:03 test: seq 62546 with 62545 rows, missing 1
2024/10/21 03:39:03 test: seq 62548 with 62547 rows, missing 1
2024/10/21 03:39:03 test: seq 62549 with 62548 rows, missing 1
2024/10/21 03:39:03 test: seq 62551 with 62550 rows, missing 1
2024/10/21 03:39:03 test: seq 62554 with 62553 rows, missing 1
2024/10/21 03:39:03 test: seq 62557 with 62556 rows, missing 1
2024/10/21 03:39:03 test: seq 62559 with 62558 rows, missing 1
2024/10/21 03:39:03 test: seq 62562 with 62559 rows, missing 3
2024/10/21 03:39:03 test: seq 62563 with 62560 rows, missing 3
2024/10/21 03:39:03 test: seq 62564 with 62561 rows, missing 3
2024/10/21 03:39:04 test: seq 62565 with 62562 rows, missing 3
2024/10/21 03:39:04 test: seq 62568 with 62565 rows, missing 3
```

At the bottom of the screen is the macOS Dock, which includes icons for various applications like Safari, Mail, and Finder.

Stop both the reader and writer and restart them with a different topic and the QUORUM consistency. Let them run for a few minutes. Is the reader reporting any missing data and does that match your expectation? Provide screenshots and briefly explain why.

Using a QUORUM consistency level that requires more than half of replicas to acknowledge the operation reduces the likelihood of missing data.

QUORUM improves consistency and minimizes data discrepancies among replication.

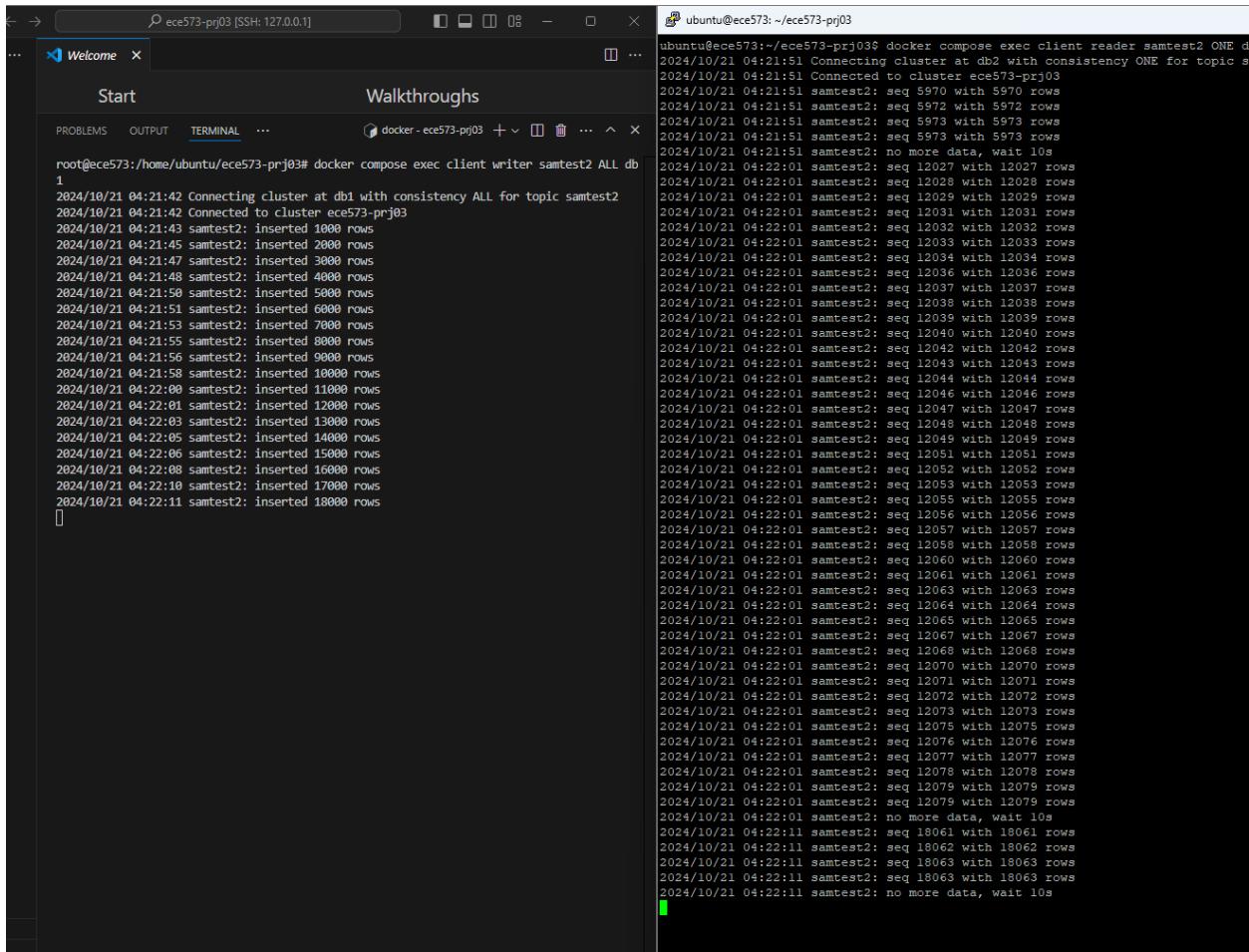
The reader is less likely to report missing data in this case, so the findings should align with this assumption. QUORUM consistency improves data integrity and consistency compared to ONE.

Screenshots of Quorum no loss with topic samtest1

```
root@instance-20240822-161614:/home/saman/ece573-prj03$ docker compose exec client writer samtest1 QUORUM DB
2024/10/21 03:41:07 Connecting to cluster at DB1 with consistency QUORUM for topic samtest1
2024/10/21 03:41:07 Connected to cluster ece573-prj03
2024/10/21 03:41:11 samtest1: inserted 3000 rows
2024/10/21 03:41:11 samtest1: inserted 3000 rows
2024/10/21 03:41:17 samtest1: inserted 3000 rows
2024/10/21 03:41:19 samtest1: inserted 3000 rows
2024/10/21 03:41:22 samtest1: inserted 3000 rows
2024/10/21 03:41:24 samtest1: inserted 3000 rows
2024/10/21 03:41:27 samtest1: inserted 3000 rows
2024/10/21 03:41:32 samtest1: inserted 3000 rows
2024/10/21 03:41:32 samtest1: inserted 3000 rows
2024/10/21 03:41:37 samtest1: inserted 3000 rows
2024/10/21 03:41:37 samtest1: inserted 3000 rows
2024/10/21 03:41:43 samtest1: inserted 3000 rows
2024/10/21 03:41:44 samtest1: inserted 3000 rows
2024/10/21 03:41:47 samtest1: inserted 3000 rows
2024/10/21 03:41:49 samtest1: inserted 3000 rows
2024/10/21 03:41:51 samtest1: inserted 3000 rows
2024/10/21 03:41:55 samtest1: inserted 3000 rows
2024/10/21 03:41:57 samtest1: inserted 3000 rows
2024/10/21 03:41:57 samtest1: inserted 3000 rows
2024/10/21 03:42:02 samtest1: inserted 3000 rows
2024/10/21 03:42:02 samtest1: inserted 3000 rows
2024/10/21 03:42:06 samtest1: inserted 22000 rows
2024/10/21 03:42:06 samtest1: inserted 23000 rows
2024/10/21 03:42:14 samtest1: inserted 25000 rows
2024/10/21 03:42:14 samtest1: inserted 26000 rows
2024/10/21 03:42:19 samtest1: inserted 27000 rows
2024/10/21 03:42:21 samtest1: inserted 28000 rows
2024/10/21 03:42:24 samtest1: inserted 29000 rows
2024/10/21 03:42:27 samtest1: inserted 30000 rows
2024/10/21 03:42:30 samtest1: inserted 30000 rows
2024/10/21 03:42:32 samtest1: inserted 30000 rows
2024/10/21 03:42:35 samtest1: inserted 30000 rows
2024/10/21 03:42:20 samtest1: seq 27194 with 27194 rows
2024/10/21 03:42:20 samtest1: seq 27195 with 27195 rows
2024/10/21 03:42:20 samtest1: seq 27199 with 27199 rows
2024/10/21 03:42:20 samtest1: seq 27200 with 27200 rows
2024/10/21 03:42:20 samtest1: seq 27201 with 27201 rows
2024/10/21 03:42:20 samtest1: seq 27203 with 27203 rows
2024/10/21 03:42:20 samtest1: seq 27206 with 27206 rows
2024/10/21 03:42:20 samtest1: seq 27207 with 27207 rows
2024/10/21 03:42:20 samtest1: seq 27211 with 27211 rows
2024/10/21 03:42:20 samtest1: seq 27213 with 27213 rows
2024/10/21 03:42:20 samtest1: seq 27215 with 27215 rows
2024/10/21 03:42:20 samtest1: seq 27217 with 27217 rows
2024/10/21 03:42:20 samtest1: seq 27218 with 27218 rows
2024/10/21 03:42:20 samtest1: seq 27220 with 27220 rows
2024/10/21 03:42:20 samtest1: seq 27221 with 27221 rows
2024/10/21 03:42:20 samtest1: seq 27223 with 27223 rows
2024/10/21 03:42:20 samtest1: seq 27224 with 27224 rows
2024/10/21 03:42:20 samtest1: seq 27228 with 27228 rows
2024/10/21 03:42:20 samtest1: seq 27230 with 27230 rows
2024/10/21 03:42:20 samtest1: seq 27230 with 27230 rows
2024/10/21 03:42:20 samtest1: no more data, wait 10s
2024/10/21 03:42:30 samtest1: seq 31208 with 31208 rows
2024/10/21 03:42:30 samtest1: seq 31401 with 31401 rows
2024/10/21 03:42:30 samtest1: seq 31409 with 31409 rows
2024/10/21 03:42:30 samtest1: seq 31410 with 31410 rows
2024/10/21 03:42:30 samtest1: seq 31413 with 31413 rows
2024/10/21 03:42:30 samtest1: seq 31417 with 31417 rows
2024/10/21 03:42:30 samtest1: seq 31419 with 31419 rows
2024/10/21 03:42:30 samtest1: seq 31421 with 31421 rows
2024/10/21 03:42:30 samtest1: seq 31424 with 31424 rows
2024/10/21 03:42:30 samtest1: seq 31427 with 31427 rows
2024/10/21 03:42:31 samtest1: seq 31433 with 31433 rows
2024/10/21 03:42:31 samtest1: seq 31440 with 31440 rows
2024/10/21 03:42:31 samtest1: seq 31447 with 31447 rows
2024/10/21 03:42:31 samtest1: seq 31453 with 31453 rows
2024/10/21 03:42:31 samtest1: seq 31457 with 31457 rows
2024/10/21 03:42:31 samtest1: seq 31461 with 31461 rows
2024/10/21 03:42:31 samtest1: seq 31463 with 31463 rows
2024/10/21 03:42:31 samtest1: seq 31466 with 31466 rows
2024/10/21 03:42:31 samtest1: seq 31470 with 31470 rows
2024/10/21 03:42:31 samtest1: seq 31476 with 31476 rows
2024/10/21 03:42:31 samtest1: seq 31478 with 31478 rows
2024/10/21 03:42:31 samtest1: no more data, wait 10s
```

Repeat the above experiment using writer with ALL consistency and reader with the ONE consistency. Don't forget to change to a different topic. Provide a screenshot and briefly explain your observations.

Screenshots



```
ubuntu@ece573:~/ece573-prj03$ docker compose exec client reader samtest2 ONE d
2024/10/21 04:21:51 Connected to cluster ece573-prj03
2024/10/21 04:21:51 samtest2: seq 5970 with 5970 rows
2024/10/21 04:21:51 samtest2: seq 5972 with 5972 rows
2024/10/21 04:21:51 samtest2: seq 5973 with 5973 rows
2024/10/21 04:21:51 samtest2: seq 5973 with 5973 rows
2024/10/21 04:21:51 samtest2: no more data, wait 10s
2024/10/21 04:22:01 samtest2: seq 12027 with 12027 rows
2024/10/21 04:22:01 samtest2: seq 12028 with 12028 rows
2024/10/21 04:22:01 samtest2: seq 12029 with 12029 rows
2024/10/21 04:22:01 samtest2: seq 12031 with 12031 rows
2024/10/21 04:22:01 samtest2: seq 12032 with 12032 rows
2024/10/21 04:22:01 samtest2: seq 12033 with 12033 rows
2024/10/21 04:22:01 samtest2: seq 12034 with 12034 rows
2024/10/21 04:22:01 samtest2: seq 12036 with 12036 rows
2024/10/21 04:22:01 samtest2: seq 12037 with 12037 rows
2024/10/21 04:22:01 samtest2: seq 12038 with 12038 rows
2024/10/21 04:22:01 samtest2: seq 12039 with 12039 rows
2024/10/21 04:22:01 samtest2: seq 12040 with 12040 rows
2024/10/21 04:22:01 samtest2: seq 12042 with 12042 rows
2024/10/21 04:22:01 samtest2: seq 12043 with 12043 rows
2024/10/21 04:22:01 samtest2: seq 12044 with 12044 rows
2024/10/21 04:22:01 samtest2: seq 12046 with 12046 rows
2024/10/21 04:22:01 samtest2: seq 12047 with 12047 rows
2024/10/21 04:22:01 samtest2: seq 12048 with 12048 rows
2024/10/21 04:22:01 samtest2: seq 12049 with 12049 rows
2024/10/21 04:22:01 samtest2: seq 12051 with 12051 rows
2024/10/21 04:22:01 samtest2: seq 12052 with 12052 rows
2024/10/21 04:22:01 samtest2: seq 12053 with 12053 rows
2024/10/21 04:22:01 samtest2: seq 12055 with 12055 rows
2024/10/21 04:22:01 samtest2: seq 12056 with 12056 rows
2024/10/21 04:22:01 samtest2: seq 12057 with 12057 rows
2024/10/21 04:22:01 samtest2: seq 12058 with 12058 rows
2024/10/21 04:22:01 samtest2: seq 12060 with 12060 rows
2024/10/21 04:22:01 samtest2: seq 12061 with 12061 rows
2024/10/21 04:22:01 samtest2: seq 12063 with 12063 rows
2024/10/21 04:22:01 samtest2: seq 12064 with 12064 rows
2024/10/21 04:22:01 samtest2: seq 12065 with 12065 rows
2024/10/21 04:22:01 samtest2: seq 12067 with 12067 rows
2024/10/21 04:22:01 samtest2: seq 12068 with 12068 rows
2024/10/21 04:22:01 samtest2: seq 12070 with 12070 rows
2024/10/21 04:22:01 samtest2: seq 12071 with 12071 rows
2024/10/21 04:22:01 samtest2: seq 12072 with 12072 rows
2024/10/21 04:22:01 samtest2: seq 12073 with 12073 rows
2024/10/21 04:22:01 samtest2: seq 12075 with 12075 rows
2024/10/21 04:22:01 samtest2: seq 12076 with 12076 rows
2024/10/21 04:22:01 samtest2: seq 12077 with 12077 rows
2024/10/21 04:22:01 samtest2: seq 12078 with 12078 rows
2024/10/21 04:22:01 samtest2: seq 12079 with 12079 rows
2024/10/21 04:22:01 samtest2: seq 12079 with 12079 rows
2024/10/21 04:22:01 samtest2: no more data, wait 10s
2024/10/21 04:22:11 samtest2: seq 18061 with 18061 rows
2024/10/21 04:22:11 samtest2: seq 18062 with 18062 rows
2024/10/21 04:22:11 samtest2: seq 18063 with 18063 rows
2024/10/21 04:22:11 samtest2: seq 18063 with 18063 rows
2024/10/21 04:22:11 samtest2: no more data, wait 10s
```

```

[columns] [control] [zoomin] [zoomout] [refresh] [exit] [?]
[1] docker@ec573-prj03: ~
2024/10/21 04:22:15 samtest2: inserted 20000 rows
2024/10/21 04:22:16 samtest2: inserted 21000 rows
2024/10/21 04:22:18 samtest2: inserted 22000 rows
2024/10/21 04:22:20 samtest2: inserted 23000 rows
2024/10/21 04:22:22 samtest2: inserted 24000 rows
2024/10/21 04:22:23 samtest2: inserted 25000 rows
2024/10/21 04:22:25 samtest2: inserted 26000 rows
2024/10/21 04:22:26 samtest2: inserted 27000 rows
2024/10/21 04:22:28 samtest2: inserted 28000 rows
2024/10/21 04:22:30 samtest2: inserted 29000 rows
2024/10/21 04:22:31 samtest2: inserted 30000 rows
2024/10/21 04:22:33 samtest2: inserted 31000 rows
2024/10/21 04:22:35 samtest2: inserted 32000 rows
2024/10/21 04:22:36 samtest2: inserted 33000 rows
2024/10/21 04:22:38 samtest2: inserted 34000 rows
2024/10/21 04:22:40 samtest2: inserted 35000 rows
2024/10/21 04:22:41 samtest2: inserted 36000 rows
2024/10/21 04:22:43 samtest2: inserted 37000 rows
2024/10/21 04:22:44 samtest2: inserted 38000 rows
2024/10/21 04:22:47 samtest2: inserted 39000 rows
2024/10/21 04:22:48 samtest2: inserted 40000 rows
2024/10/21 04:22:50 samtest2: inserted 41000 rows
2024/10/21 04:22:52 samtest2: inserted 42000 rows
2024/10/21 04:22:53 samtest2: inserted 43000 rows
2024/10/21 04:22:55 samtest2: inserted 44000 rows
2024/10/21 04:22:57 samtest2: inserted 45000 rows
2024/10/21 04:22:58 samtest2: inserted 46000 rows
2024/10/21 04:23:00 samtest2: inserted 47000 rows
2024/10/21 04:23:01 samtest2: inserted 48000 rows
2024/10/21 04:23:03 samtest2: inserted 49000 rows
2024/10/21 04:23:05 samtest2: inserted 50000 rows
2024/10/21 04:23:06 samtest2: inserted 51000 rows
2024/10/21 04:23:08 samtest2: inserted 52000 rows
2024/10/21 04:23:09 samtest2: inserted 53000 rows
2024/10/21 04:23:11 samtest2: inserted 54000 rows
2024/10/21 04:23:13 samtest2: inserted 55000 rows
2024/10/21 04:23:14 samtest2: inserted 56000 rows
2024/10/21 04:23:16 samtest2: inserted 57000 rows
2024/10/21 04:23:17 samtest2: inserted 58000 rows
2024/10/21 04:23:19 samtest2: inserted 59000 rows
2024/10/21 04:23:21 samtest2: inserted 60000 rows
2024/10/21 04:23:22 samtest2: inserted 61000 rows
2024/10/21 04:23:24 samtest2: inserted 62000 rows
2024/10/21 04:23:25 samtest2: inserted 63000 rows
2024/10/21 04:23:27 samtest2: inserted 64000 rows
2024/10/21 04:23:29 samtest2: inserted 65000 rows
2024/10/21 04:23:30 samtest2: inserted 66000 rows
2024/10/21 04:23:32 samtest2: inserted 67000 rows
2024/10/21 04:23:34 samtest2: inserted 68000 rows
2024/10/21 04:22:52 samtest2: seq 42148 with 42148 rows
2024/10/21 04:22:52 samtest2: seq 42149 with 42149 rows
2024/10/21 04:22:52 samtest2: seq 42150 with 42150 rows
2024/10/21 04:22:52 samtest2: seq 42152 with 42152 rows
2024/10/21 04:22:52 samtest2: seq 42153 with 42153 rows
2024/10/21 04:22:52 samtest2: seq 42154 with 42154 rows
2024/10/21 04:22:52 samtest2: seq 42155 with 42155 rows
2024/10/21 04:22:52 samtest2: seq 42156 with 42156 rows
2024/10/21 04:22:52 samtest2: seq 42158 with 42158 rows
2024/10/21 04:22:52 samtest2: seq 42158 with 42158 rows
2024/10/21 04:22:52 samtest2: no more data, wait 10s
2024/10/21 04:22:52 samtest2: seq 42159 with 42159 rows
2024/10/21 04:22:52 samtest2: seq 43390 with 43390 rows
2024/10/21 04:22:52 samtest2: seq 43391 with 43391 rows
2024/10/21 04:22:52 samtest2: seq 43392 with 43392 rows
2024/10/21 04:22:52 samtest2: seq 43394 with 43394 rows
2024/10/21 04:22:52 samtest2: seq 43394 with 43394 rows
2024/10/21 04:22:52 samtest2: no more data, wait 10s
2024/10/21 04:23:02 samtest2: seq 54609 with 54609 rows
2024/10/21 04:23:02 samtest2: seq 54610 with 54610 rows
2024/10/21 04:23:02 samtest2: seq 54610 with 54610 rows
2024/10/21 04:23:02 samtest2: no more data, wait 10s
2024/10/21 04:23:02 samtest2: seq 60879 with 60879 rows
2024/10/21 04:23:02 samtest2: seq 60880 with 60880 rows
2024/10/21 04:23:02 samtest2: seq 60882 with 60882 rows
2024/10/21 04:23:02 samtest2: seq 60884 with 60884 rows
2024/10/21 04:23:02 samtest2: seq 60885 with 60885 rows
2024/10/21 04:23:02 samtest2: seq 60885 with 60885 rows
2024/10/21 04:23:02 samtest2: no more data, wait 10s
2024/10/21 04:23:02 samtest2: seq 60886 with 60886 rows
2024/10/21 04:23:02 samtest2: seq 60887 with 60887 rows
2024/10/21 04:23:02 samtest2: seq 64580 with 64580 rows
2024/10/21 04:23:02 samtest2: seq 64583 with 64583 rows
2024/10/21 04:23:02 samtest2: seq 64584 with 64584 rows
2024/10/21 04:23:02 samtest2: seq 64585 with 64585 rows
2024/10/21 04:23:02 samtest2: seq 64586 with 64586 rows
2024/10/21 04:23:02 samtest2: seq 64587 with 64587 rows
2024/10/21 04:23:02 samtest2: seq 64588 with 64588 rows
2024/10/21 04:23:02 samtest2: seq 64589 with 64589 rows
2024/10/21 04:23:02 samtest2: seq 64590 with 64590 rows
2024/10/21 04:23:02 samtest2: seq 64591 with 64591 rows
2024/10/21 04:23:02 samtest2: seq 64592 with 64592 rows
2024/10/21 04:23:02 samtest2: seq 64593 with 64593 rows
2024/10/21 04:23:02 samtest2: seq 64594 with 64594 rows
2024/10/21 04:23:02 samtest2: seq 64596 with 64596 rows
2024/10/21 04:23:02 samtest2: seq 64597 with 64597 rows
2024/10/21 04:23:02 samtest2: seq 64598 with 64598 rows
2024/10/21 04:23:02 samtest2: seq 64599 with 64599 rows
2024/10/21 04:23:02 samtest2: seq 64600 with 64600 rows
2024/10/21 04:23:02 samtest2: seq 64601 with 64601 rows
2024/10/21 04:23:02 samtest2: seq 64602 with 64602 rows
2024/10/21 04:23:02 samtest2: seq 64604 with 64604 rows
2024/10/21 04:23:02 samtest2: seq 64606 with 64606 rows
2024/10/21 04:23:02 samtest2: no more data, wait 10s
2024/10/21 04:23:02 samtest2: no more data, wait 10s

```

Explanation

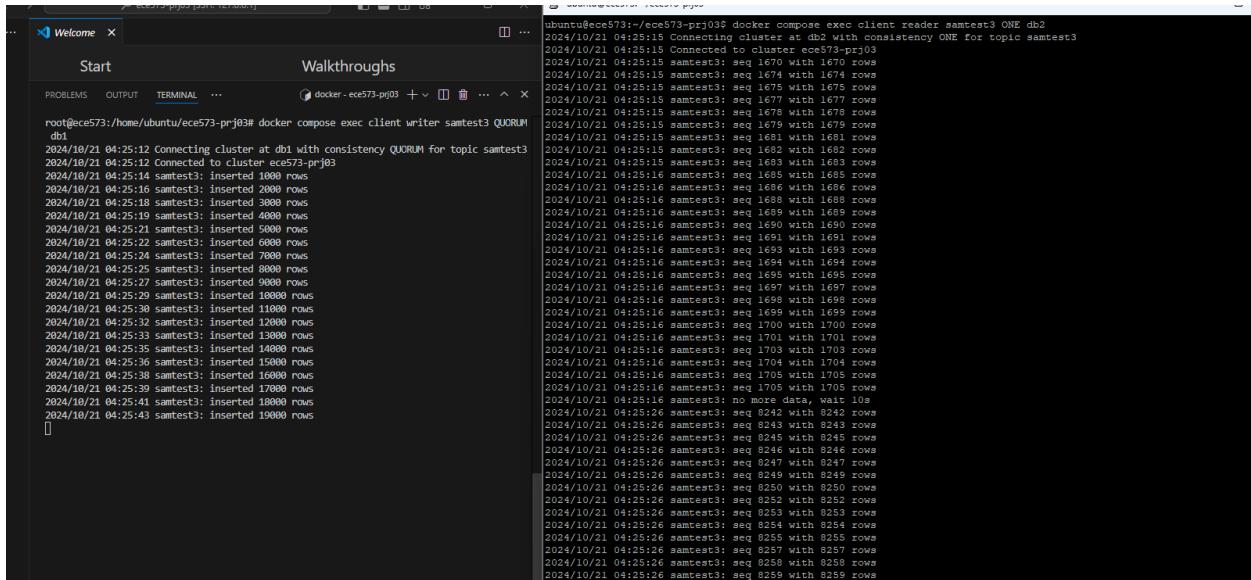
When a writer utilizes ALL consistency and a reader uses ONE consistency, the latter is more likely to report missing data. Cassandra's ALL consistency requires all replicas to recognize an operation. If a writer acknowledges a write with ALL consistency and a reader attempts to read with ONE consistency level, it may read from a separate replica that has not yet received the data. As a result, readers may not find expected data, resulting in missing data.

What may happen if the writer uses the QUORUM consistency and the reader uses the ONE consistency?

If the writer employs QUORUM consistency and the reader utilizes ONE consistency, there may still be some situations of missing data, but not as frequently as with ALL consistency.

QUORUM consistency needs a majority of replicas to recognize an operation and is more resistant to discrepancies than ONE consistency. However, it lacks the strength of ALL consistency. Even if the writer acknowledges a write using QUORUM and the reader reads with ONE, data may not fully propagate to the replica being read from, leading to missing data.

Screenshots



```
root@ece573:/home/ubuntu/ece573-prj03# docker compose exec client writer samtest3 QUORUM db1
2024/10/21 04:25:12 Connecting cluster at db1 with consistency QUORUM for topic samtest3
2024/10/21 04:25:12 Connected to cluster ece573-prj03
2024/10/21 04:25:14 samtest3: inserted 1000 rows
2024/10/21 04:25:16 samtest3: inserted 2000 rows
2024/10/21 04:25:18 samtest3: inserted 3000 rows
2024/10/21 04:25:19 samtest3: inserted 4000 rows
2024/10/21 04:25:21 samtest3: inserted 5000 rows
2024/10/21 04:25:22 samtest3: inserted 6000 rows
2024/10/21 04:25:25 samtest3: inserted 7000 rows
2024/10/21 04:25:27 samtest3: inserted 8000 rows
2024/10/21 04:25:29 samtest3: inserted 9000 rows
2024/10/21 04:25:30 samtest3: inserted 10000 rows
2024/10/21 04:25:32 samtest3: inserted 12000 rows
2024/10/21 04:25:33 samtest3: inserted 13000 rows
2024/10/21 04:25:35 samtest3: inserted 14000 rows
2024/10/21 04:25:36 samtest3: inserted 15000 rows
2024/10/21 04:25:38 samtest3: inserted 16000 rows
2024/10/21 04:25:39 samtest3: inserted 17000 rows
2024/10/21 04:25:41 samtest3: inserted 18000 rows
2024/10/21 04:25:43 samtest3: inserted 19000 rows
[...]
ubuntu@ece573:~/ece573-prj03$ docker compose exec client reader samtest3 ONE db2
2024/10/21 04:25:15 Connecting to cluster db2 with consistency ONE for topic samtest3
2024/10/21 04:25:15 Connected to cluster ece573-prj03
2024/10/21 04:25:15 samtest3: seq 1670 with 1670 rows
2024/10/21 04:25:15 samtest3: seq 1675 with 1675 rows
2024/10/21 04:25:15 samtest3: seq 1680 with 1680 rows
2024/10/21 04:25:15 samtest3: seq 1678 with 1678 rows
2024/10/21 04:25:15 samtest3: seq 1679 with 1679 rows
2024/10/21 04:25:15 samtest3: seq 1679 with 1679 rows
2024/10/21 04:25:15 samtest3: seq 1681 with 1681 rows
2024/10/21 04:25:15 samtest3: seq 1682 with 1682 rows
2024/10/21 04:25:15 samtest3: seq 1683 with 1683 rows
2024/10/21 04:25:15 samtest3: seq 1685 with 1685 rows
2024/10/21 04:25:16 samtest3: seq 1686 with 1686 rows
2024/10/21 04:25:16 samtest3: seq 1688 with 1688 rows
2024/10/21 04:25:16 samtest3: seq 1689 with 1689 rows
2024/10/21 04:25:16 samtest3: seq 1690 with 1690 rows
2024/10/21 04:25:16 samtest3: seq 1691 with 1691 rows
2024/10/21 04:25:16 samtest3: seq 1693 with 1693 rows
2024/10/21 04:25:16 samtest3: seq 1694 with 1694 rows
2024/10/21 04:25:16 samtest3: seq 1695 with 1695 rows
2024/10/21 04:25:16 samtest3: seq 1697 with 1697 rows
2024/10/21 04:25:16 samtest3: seq 1698 with 1698 rows
2024/10/21 04:25:16 samtest3: seq 1699 with 1699 rows
2024/10/21 04:25:16 samtest3: seq 1701 with 1701 rows
2024/10/21 04:25:16 samtest3: seq 1703 with 1703 rows
2024/10/21 04:25:16 samtest3: seq 1704 with 1704 rows
2024/10/21 04:25:16 samtest3: seq 1705 with 1705 rows
2024/10/21 04:25:16 samtest3: seq 1705 with 1705 rows
2024/10/21 04:25:16 samtest3: no more data, wait 10s
2024/10/21 04:25:20 samtest3: seq 8242 with 8242 rows
2024/10/21 04:25:20 samtest3: seq 8243 with 8243 rows
2024/10/21 04:25:20 samtest3: seq 8244 with 8244 rows
2024/10/21 04:25:20 samtest3: seq 8245 with 8245 rows
2024/10/21 04:25:20 samtest3: seq 8246 with 8246 rows
2024/10/21 04:25:20 samtest3: seq 8247 with 8247 rows
2024/10/21 04:25:20 samtest3: seq 8249 with 8249 rows
2024/10/21 04:25:26 samtest3: seq 8250 with 8250 rows
2024/10/21 04:25:26 samtest3: seq 8252 with 8252 rows
2024/10/21 04:25:26 samtest3: seq 8253 with 8253 rows
2024/10/21 04:25:26 samtest3: seq 8254 with 8254 rows
2024/10/21 04:25:26 samtest3: seq 8255 with 8255 rows
2024/10/21 04:25:26 samtest3: seq 8256 with 8256 rows
2024/10/21 04:25:26 samtest3: seq 8257 with 8257 rows
2024/10/21 04:25:26 samtest3: seq 8258 with 8258 rows
2024/10/21 04:25:26 samtest3: seq 8259 with 8259 rows
2024/10/21 04:25:26 samtest3: seq 8259 with 8259 rows
```

Since the source code of both writer and reader are available in our VM, can we run them from the VM directly without using the client container? Briefly explain why or why not.

You can run the writer and reader applications directly from the VM, without utilizing the client container. The source code for both the writer and reader is accessible, allowing for direct execution from the virtual machine. To ensure proper application execution, make sure the VM has all essential dependencies and configurations. Running them in containers offers a more controlled environment, but they can also be run directly on a virtual machine with the necessary software and libraries installed.

IV . Availability

- Modify writer.go to retry for a failed write operation. Use a retry policy to wait 10 seconds before retrying but allow an infinite number of retries.

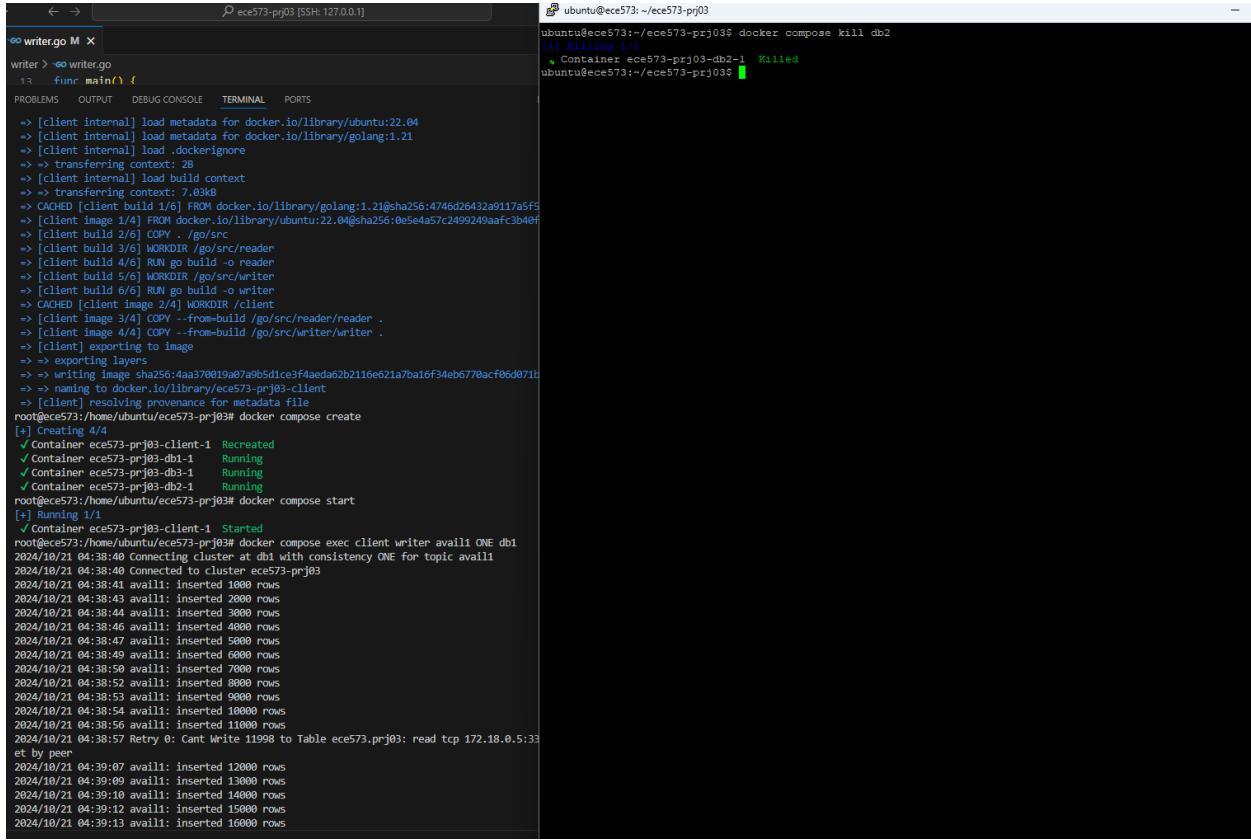
Screenshot with the modified code

```
for seq := 1; ; seq++ {
    value := rand.Float64()
    retries := 0
    for {
        err := session.Query(
            INSERT INTO ece573.prj03 (topic, seq, value) VALUES (?, ?, ?),
            topic, seq, value).
            Exec()
        if err == nil {
            break //log.Fatalf("Cannot write %d to table ece473.prj03: %v", seq, err)
        }
        log.Printf("Retry %d: Cant Write %d to Table ece573.prj03: %v",retries, seq, err)
        time.Sleep(10 * time.Second)
        retries++
    }
    if seq%1000 == 0 {
        log.Printf("%s: inserted %d rows", topic, seq)
    }
}
```

Build, create, and start the client again and demonstrate that the writer can keep adding rows when db2 is killed. Provide screenshots as needed.

When using the modified writer with the infinite retry policy, it should continue to add rows even if db2 is terminated. The retry method attempts to write data until it is successful. No errors will be displayed, and the writer will be able to withstand temporary node failures.

Screenshot



The screenshot shows a terminal window with two tabs. The left tab is titled 'writer.go M' and contains the source code for a Go application named 'writer.go'. The right tab is titled 'ubuntu@ece573: ~/ece573-prj03 [SSH: 127.0.0.1]' and shows the terminal session.

```
ubuntu@ece573:~/ece573-prj03$ docker compose kill db2
[+] Killing 1/1
  Container ece573-prj03-db2-1 Killed
ubuntu@ece573:~/ece573-prj03$
```

The terminal session shows the following sequence of commands:

- Execution of 'writer.go'
- Building the Docker image: 'docker compose build'
- Creating the Docker network: 'docker compose create'
- Starting the Docker containers: 'docker compose start'
- Monitoring the writer's progress: 'writer.go' logs show rows being inserted into the 'avail1' table.
- Killing the 'db2' container: 'docker compose kill db2'
- Continuing to insert rows: The writer continues to insert rows into the 'avail1' table, even after the database has been killed.

What if db3 is also killed? Provide screenshots and briefly explain your findings.

If both db2 and db3 are destroyed, the updated writer with infinite retry policy will continue to attempt writes notwithstanding node failures. The retry technique is particularly handy in this case. You should not observe any errors, and the writer should continue to add rows. The writer's robustness to node failures is due to the retry algorithm.

Please find the photos.

Screenshot

The screenshot displays two terminal windows side-by-side. The left terminal window shows a Go code editor with a file named `writer.go`. The code contains a `func main()` function with a loop that inserts rows into a database table. The log output shows numerous insertions of 24000 rows each, starting from 2024/10/21 04:39:26 and continuing until 2024/10/21 04:40:29. The right terminal window shows a command-line interface where two Docker containers, `ece573-prj03-db2-1` and `ece573-prj03-db3-1`, are being killed using the `docker compose kill` command. The logs indicate that both containers have been killed successfully.

```
.. ← → ece573-prj03 [SSH: 127.0.0.1]
--> writer.go M ×
writer > --> writer.go
  13  func main() {
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
2024/10/21 04:39:26 avail1: inserted 24000 rows
2024/10/21 04:39:27 avail1: inserted 25000 rows
2024/10/21 04:39:29 avail1: inserted 26000 rows
2024/10/21 04:39:30 avail1: inserted 27000 rows
2024/10/21 04:39:32 avail1: inserted 28000 rows
2024/10/21 04:39:34 avail1: inserted 29000 rows
2024/10/21 04:39:35 avail1: inserted 30000 rows
2024/10/21 04:39:37 avail1: inserted 31000 rows
2024/10/21 04:39:38 avail1: inserted 32000 rows
2024/10/21 04:39:40 avail1: inserted 33000 rows
2024/10/21 04:39:41 avail1: inserted 34000 rows
2024/10/21 04:39:43 avail1: inserted 35000 rows
2024/10/21 04:39:44 avail1: inserted 36000 rows
2024/10/21 04:39:46 avail1: inserted 37000 rows
2024/10/21 04:39:47 avail1: inserted 38000 rows
2024/10/21 04:39:49 avail1: inserted 39000 rows
2024/10/21 04:39:50 avail1: inserted 40000 rows
2024/10/21 04:39:52 avail1: inserted 41000 rows
2024/10/21 04:39:53 avail1: inserted 42000 rows
2024/10/21 04:39:55 avail1: inserted 43000 rows
2024/10/21 04:39:56 avail1: inserted 44000 rows
2024/10/21 04:39:58 avail1: inserted 45000 rows
2024/10/21 04:39:59 avail1: inserted 46000 rows
2024/10/21 04:40:01 avail1: inserted 47000 rows
2024/10/21 04:40:02 avail1: inserted 48000 rows
2024/10/21 04:40:04 avail1: inserted 49000 rows
2024/10/21 04:40:06 avail1: inserted 50000 rows
2024/10/21 04:40:07 avail1: inserted 51000 rows
2024/10/21 04:40:09 avail1: inserted 52000 rows
2024/10/21 04:40:10 avail1: inserted 53000 rows
2024/10/21 04:40:12 avail1: inserted 54000 rows
2024/10/21 04:40:13 avail1: inserted 55000 rows
2024/10/21 04:40:15 avail1: inserted 56000 rows
2024/10/21 04:40:16 avail1: inserted 57000 rows
2024/10/21 04:40:18 avail1: inserted 58000 rows
2024/10/21 04:40:19 avail1: inserted 59000 rows
2024/10/21 04:40:21 avail1: inserted 60000 rows
2024/10/21 04:40:22 avail1: inserted 61000 rows
2024/10/21 04:40:24 avail1: inserted 62000 rows
2024/10/21 04:40:25 avail1: inserted 63000 rows
2024/10/21 04:40:27 avail1: inserted 64000 rows
2024/10/21 04:40:28 avail1: inserted 65000 rows
2024/10/21 04:40:29 Retry 0: Cant Write 65680 to Table ece573.prj03: read tcp 172.18.0.5:57
et by peer
2024/10/21 04:40:40 avail1: inserted 66000 rows
2024/10/21 04:40:42 avail1: inserted 67000 rows
2024/10/21 04:40:44 avail1: inserted 68000 rows
2024/10/21 04:40:47 avail1: inserted 69000 rows
2024/10/21 04:40:49 avail1: inserted 70000 rows
2024/10/21 04:40:52 avail1: inserted 71000 rows
[ ]
```

```
ubuntu@ece573:~/ece573-prj03$ docker compose kill db2
[+] Killing 1/1
  ✓ Container ece573-prj03-db2-1 Killed
ubuntu@ece573:~/ece573-prj03$ docker compose kill db3
[+] Killing 1/1
  ✓ Container ece573-prj03-db3-1 Killed
ubuntu@ece573:~/ece573-prj03$
```

Restart writer with consistency QUORUM. Does the writer perform differently when one server is killed and when two servers are killed? Provide screenshots and briefly explain your findings

The QUORUM consistency can be investigated, for instance, from a db2 standpoint, when one server is killed. The writer will continue to work because the QUORUM consistency just requires the majority of the replicas to acknowledge the write operation. At this point, since we have a total of three nodes—db1, db2, and db3—it will work if at least two nodes are up.

- The writer can add rows, and you might have a little higher latency because of the loss of a single node. Zero errors should happen.

Two Servers Killed (db2, db3):

- If two servers are killed, and the consistency used by the writer is set to QUORUM, then that will never reach a quorum, which may lead to an error or sometimes failure to write. This is, in essence, because for any write, QUORUM needs at least two nodes to acknowledge the writer when three nodes are used. Now, with two of the servers killed, only one node is up and running, which cannot satisfy the achieved QUORUM. In this case, the writer is likely to encounter errors or failures to write.

Screenshots

```

ubuntu@ece573:~/ece573-prj03$ docker compose kill db2
[*] Killing db2
Container ece573-prj03-db2-1 Killed
ubuntu@ece573:~/ece573-prj03$ docker compose kill db3
[*] Killing db3
Container ece573-prj03-db3-1 Killed
ubuntu@ece573:~/ece573-prj03$ docker ps
CONTAINER ID IMAGE NAMES COMMAND CREATED STATUS PORTS
ca7cf3393e7f ece573-prj03-client "bin/sh -c 'tail -f'" 4 minutes ago Up 3 minutes
ef50f61d7693 cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 24 minutes 7000-7001/tcp, 7199/tcp
p_9042/tcp, 9140/tcp ece573-prj03-db1-1
ubuntu@ece573:~/ece573-prj03$ docker ps create
"docker ps" accepts no arguments.
See 'docker ps --help'.
Usage: docker ps [OPTIONS]

List containers
ubuntu@ece573:~/ece573-prj03$ docker compose create
[*] Creating 4/0
Container ece573-prj03-client-1 Running
Container ece573-prj03-db1-1 Running
Container ece573-prj03-db2-1 Created
Container ece573-prj03-db3-1 Created
ubuntu@ece573:~/ece573-prj03$ docker compose start
[*] Starting 4/4
Container ece573-prj03-db2-1 Started
Container ece573-prj03-db3-1 Started
ubuntu@ece573:~/ece573-prj03$ docker ps
CONTAINER ID IMAGE NAMES COMMAND CREATED STATUS PORTS
ca7cf3393e7f ece573-prj03-client "bin/sh -c 'tail -f'" 4 minutes ago Up 4 minutes
ef50f61d7693 cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 25 minutes 7000-7001/tcp, 7199/tcp
p_9042/tcp, 9140/tcp ece573-prj03-db1-1
fa043bb399b cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 2 seconds 7000-7001/tcp, 7199/tcp
p_9042/tcp, 9140/tcp ece573-prj03-db3-1
dabe36fe4512 cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 2 seconds 7000-7001/tcp, 7199/tcp
p_9042/tcp, 9140/tcp ece573-prj03-db2-1
ubuntu@ece573:~/ece573-prj03$ 

```

Only db2 is killed

```

ubuntu@ece573:~/ece573-prj03$ docker compose up -d
[*] Creating 4/0
  * Container ece573-prj03-client-1  Running
  * Container ece573-prj03-db1-1    Running
  * Container ece573-prj03-db2-1    Created
  * Container ece573-prj03-db3-1    Created
ubuntu@ece573:~/ece573-prj03$ docker compose start
(*) Starting 2/2
  * Container ece573-prj03-db2-1  Started
  * Container ece573-prj03-db3-1  Started
ubuntu@ece573:~/ece573-prj03$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
ca7cf3393e7f ece573-prj03-client "/bin/sh -c 'tail -f'" 4 minutes ago Up 4 minutes
  * Container ece573-prj03-client-1  Running
  * Container ece573-prj03-db1-1    Running
  * Container ece573-prj03-db2-1    Created
  * Container ece573-prj03-db3-1    Created
ef90f61d7693 cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 25 minutes 7000-7001/tcp, 7199/t
p, 9042/tcp, 9160/tcp ece573-prj03-db1-1
faf043bb399b cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 2 seconds 7000-7001/tcp, 7199/t
p, 9042/tcp, 9160/tcp ece573-prj03-db3-1
dab83ce4512 cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 2 seconds 7000-7001/tcp, 7199/t
p, 9042/tcp, 9160/tcp ece573-prj03-db2-1
ubuntu@ece573:~/ece573-prj03$ docker compose kill db2
(*) Killing 1/1
  * Container ece573-prj03-db2-1  Killed
ubuntu@ece573:~/ece573-prj03$ 

```

Both db killed

```

ubuntu@ece573:~/ece573-prj03$ docker compose up -d
[*] Creating 4/0
  * Container ece573-prj03-client-1  Running
  * Container ece573-prj03-db1-1    Running
  * Container ece573-prj03-db2-1    Created
  * Container ece573-prj03-db3-1    Created
ubuntu@ece573:~/ece573-prj03$ docker compose start
(*) Starting 2/2
  * Container ece573-prj03-db2-1  Started
  * Container ece573-prj03-db3-1  Started
ubuntu@ece573:~/ece573-prj03$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
ca7cf3393e7f ece573-prj03-client "/bin/sh -c 'tail -f'" 4 minutes ago Up 4 minutes
  * Container ece573-prj03-client-1  Running
  * Container ece573-prj03-db1-1    Running
  * Container ece573-prj03-db2-1    Created
  * Container ece573-prj03-db3-1    Created
ef90f61d7693 cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 25 minutes 7000-7001/tcp, 7199/t
p, 9042/tcp, 9160/tcp ece573-prj03-db1-1
faf043bb399b cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 2 seconds 7000-7001/tcp, 7199/t
p, 9042/tcp, 9160/tcp ece573-prj03-db3-1
dab83ce4512 cassandra:4.1.3 "docker-entrypoint.s..." 25 minutes ago Up 2 seconds 7000-7001/tcp, 7199/t
p, 9042/tcp, 9160/tcp ece573-prj03-db2-1
ubuntu@ece573:~/ece573-prj03$ docker compose kill db2
(*) Killing 1/1
  * Container ece573-prj03-db2-1  Killed
ubuntu@ece573:~/ece573-prj03$ docker compose kill db1
(*) Killing 1/1
  * Container ece573-prj03-db1-1  Killed
ubuntu@ece573:~/ece573-prj03$ 

```

Conclusion

In this project, we implement Cassandra Cluster Management by using Docker containers and analyze how the consistency levels of Cassandra would impact the resilience and performance of a distributed database. This project requires the simulation of server failures, configuration of consistency levels, and subsequent study of the behavior of the Cassandra cluster.

Key findings and takeaways of the project:

- **Cassandra Cluster Configuration:** Cassandra Cluster is set up using Docker containers for multiple nodes: cassandra cluster, db1, db2, db3, to simulate a distributed environment.
- **Consistency Levels:** We have tried playing with different consistency levels of ONE and QUORUM to see how it works for read and write operations in a distributed database.
- **Retrying Failed Operations:** We made several changes to the writer program in order for it to retry a failed write operation. It has been quite an important illustration of how applications of distributed systems should handle transient failures.
- **Resilience to Node Failure:** Our modified version of the writer program was able to sustain its writing of data despite the killing of individual nodes—for example, db2—thanks to the retry mechanism.
- **Quorum and Multiple Node Failures:** QUORUM consistency allows the writer to be killed and continue on when one node is killed but might not work when two nodes are killed since a quorum is required.
- **Variability Based on Configuration:** One should remember that Cassandra may have unforeseen behavior based on the replication factor, number of nodes, and special configurations in your cluster.