

Question:

What distributed database system works best for parallel writes?

1. RQLite
2. MySQL

Experimental setup:

I used three Ubuntu instances on Jetstream2 for the Hadoop setup. These instances served as the master node and two data nodes. For the master node I increased the specs to m3.medium as I needed more storage. This overall gave me 8 CPU cores, 30 GB RAM and 60 GB root disk. This resize was needed as it accidentally crashed after the installation of files and it ran out of memory. The other two instances were m3.quad with 4 CPUs, 15 GB of RAM, and a 20 GB disk size.

Overall Specs:

Master Node:

- 8 CPU cores
- 30 GB Ram
- 60 GB root disk

Data Nodes:

- 4 CPU cores
- 15 GB RAM
- 20 GB disk

The setup for RQLite [2] was really simple and quick as it just took a couple of commands to download the binary files and unzip it. I used rqlite-v8.24.1. We first setup the master node and then the replicating data nodes. Finally connect them through the config file and start the processes. Finally I also install [pyrqlite](#) as this package was needed to make a connection to RQLite using python

The setup for MySQL [4] was simple as well. I just faced some issues with configuring the latest version. Trying to install a slightly older version 8.0.32 worked really well and was less time consuming. Here as well we start with setting up the master nodes, then we setup and configure the data nodes to talk with the master node and finally we set up the client node.

Testing Procedure:

I created a python script that runs an insert query in a loop. The insert query is really simple as it only inserts a name field in the table. The loop runs in different sizes of 500, 1000, 1500, 2000,

and 2500. These sizes were chosen as a reference from the RQLite paper [6]. For each size I ran the script 3 times and then calculated the margin of error to decide if I needed to perform additional runs. The python script is pretty simple as it first creates a connection with the database. It then creates a table called “foo” with an id that is auto incremental and a text field “name”. It then runs the insert command in a loop for the different test sizes. Only the writing to the table is timed. After this we get a count of all the entries in the table and log that as well. Something that may affect the overall time is the time to create name strings to insert in the table. But as it is common for both the tests it can be ignored.

Test Results / Accuracy:

The number of tests are validated by using the below performance formula and we find the margin error.

$$n = \left(\frac{z_c \sigma}{E} \right)^2$$

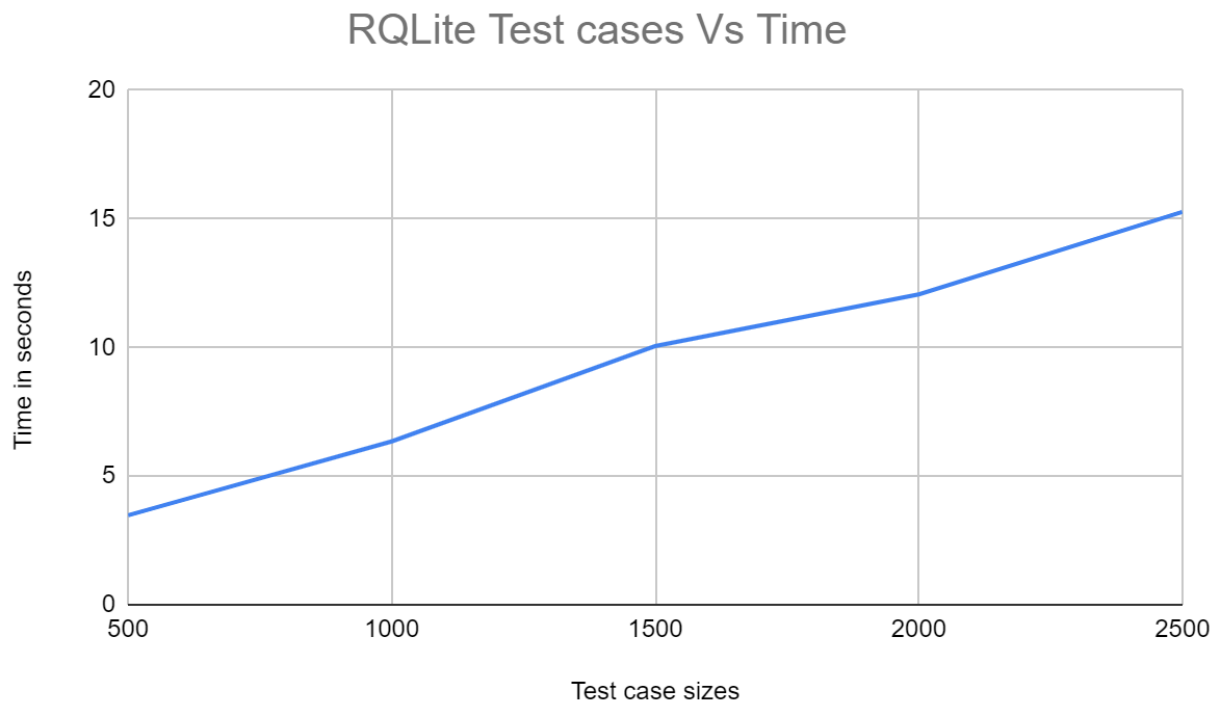
As mentioned in the testing procedure each test case is executed three times and then the margin of error is calculated using the above formula. If the margin of error is greater than 5% of the average of the timings means we need to test for some more runs.

RQLite:

Write	Run 1	Run 1	Run 3	Standard Deviation	Avg	Margin of Error	5% of Avg
500	3.57	3.37	3.48	0.100166528	3.473333333	0.113352422	0.1736666667
1000	6.12	6.49	6.43	0.1985782801	6.346666667	0.2247190699	0.3173333333
1500	10.3	9.85	10.02	0.2272296929	10.05666667	0.2571421467	0.5028333333
2000	12.35	11.74	12.1	0.3066485502	12.06333333	0.3470156804	0.6031666667
2500	15.43	15.63	14.76	0.4556680078	15.27333333	0.5156520181	0.7636666667

sqlite_write_test.log

```
1 2024-05-14 13:08:29,548 - INFO - Inserting 500 records...
2 2024-05-14 13:08:33,122 - INFO - Time taken to insert 500 records: 3.57 seconds
3 2024-05-14 13:08:33,127 - INFO - Number of records fetched: 500
4 2024-05-14 13:10:02,533 - INFO - Inserting 500 records...
5 2024-05-14 13:10:05,904 - INFO - Time taken to insert 500 records: 3.37 seconds
6 2024-05-14 13:10:05,908 - INFO - Number of records fetched: 500
7 2024-05-14 13:10:13,606 - INFO - Inserting 500 records...
8 2024-05-14 13:10:17,088 - INFO - Time taken to insert 500 records: 3.48 seconds
9 2024-05-14 13:10:17,092 - INFO - Number of records fetched: 500
10 2024-05-14 13:16:09,376 - INFO - Inserting 1000 records...
11 2024-05-14 13:16:15,493 - INFO - Time taken to insert 1000 records: 6.12 seconds
12 2024-05-14 13:16:15,501 - INFO - Number of records fetched: 1000
13 2024-05-14 13:16:17,196 - INFO - Inserting 1000 records...
14 2024-05-14 13:16:23,684 - INFO - Time taken to insert 1000 records: 6.49 seconds
15 2024-05-14 13:16:23,690 - INFO - Number of records fetched: 1000
16 2024-05-14 13:16:25,130 - INFO - Inserting 1000 records...
17 2024-05-14 13:16:31,560 - INFO - Time taken to insert 1000 records: 6.43 seconds
18 2024-05-14 13:16:31,567 - INFO - Number of records fetched: 1000
19 2024-05-14 13:18:01,857 - INFO - Inserting 1500 records...
20 2024-05-14 13:18:12,153 - INFO - Time taken to insert 1500 records: 10.30 seconds
21 2024-05-14 13:18:12,162 - INFO - Number of records fetched: 1500
22 2024-05-14 13:18:13,450 - INFO - Inserting 1500 records...
23 2024-05-14 13:18:23,303 - INFO - Time taken to insert 1500 records: 9.85 seconds
24 2024-05-14 13:18:23,314 - INFO - Number of records fetched: 1500
25 2024-05-14 13:18:24,798 - INFO - Inserting 1500 records...
26 2024-05-14 13:18:34,818 - INFO - Time taken to insert 1500 records: 10.02 seconds
27 2024-05-14 13:18:34,827 - INFO - Number of records fetched: 1500
28 2024-05-14 13:19:46,776 - INFO - Inserting 2000 records...
29 2024-05-14 13:19:59,129 - INFO - Time taken to insert 2000 records: 12.35 seconds
30 2024-05-14 13:19:59,146 - INFO - Number of records fetched: 2000
31 2024-05-14 13:20:18,206 - INFO - Inserting 2000 records...
32 2024-05-14 13:20:29,947 - INFO - Time taken to insert 2000 records: 11.74 seconds
33 2024-05-14 13:20:29,960 - INFO - Number of records fetched: 2000
34 2024-05-14 13:21:59,739 - INFO - Inserting 2000 records...
35 2024-05-14 13:22:11,838 - INFO - Time taken to insert 2000 records: 12.10 seconds
36 2024-05-14 13:22:11,849 - INFO - Number of records fetched: 2000
37 2024-05-14 13:25:15,497 - INFO - Inserting 2500 records...
38 2024-05-14 13:25:30,928 - INFO - Time taken to insert 2500 records: 15.43 seconds
39 2024-05-14 13:25:30,944 - INFO - Number of records fetched: 2500
40 2024-05-14 13:25:32,642 - INFO - Inserting 2500 records...
41 2024-05-14 13:25:48,273 - INFO - Time taken to insert 2500 records: 15.63 seconds
42 2024-05-14 13:25:48,288 - INFO - Number of records fetched: 2500
43 2024-05-14 13:25:51,438 - INFO - Inserting 2500 records...
44 2024-05-14 13:26:06,203 - INFO - Time taken to insert 2500 records: 14.76 seconds
45 2024-05-14 13:26:06,215 - INFO - Number of records fetched: 2500
```



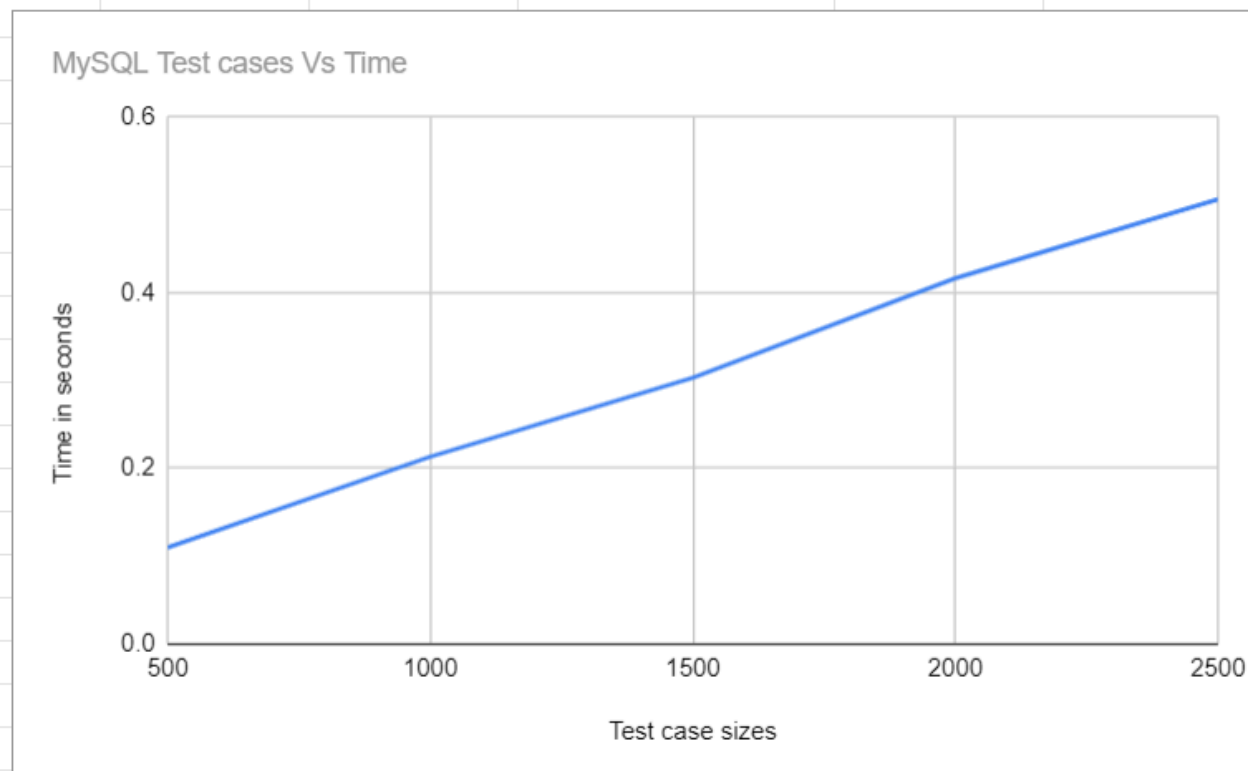
I used the testing procedure described above and calculated the margin of error and in every case running the test 3 times seems sufficient as the margin of error is always less than five percent of the total average.

MySQL:

Write	Run 1	Run 1	Run 3	Standard Deviation	Avg	Margin of Error	5% of Avg
500	0.11	0.11	0.11	0	0.11	0	0.0055
1000	0.22	0.21	0.21	0.005773502692	0.21333333333	0.006533524986	0.01066666667
1500	0.31	0.3	0.3	0.005773502692	0.30333333333	0.006533524986	0.01516666667
2000	0.42	0.41	0.42	0.005773502692	0.41666666667	0.006533524986	0.02083333333
2500	0.5	0.51	0.51	0.005773502692	0.50666666667	0.006533524986	0.02533333333

mysql_write_test.log

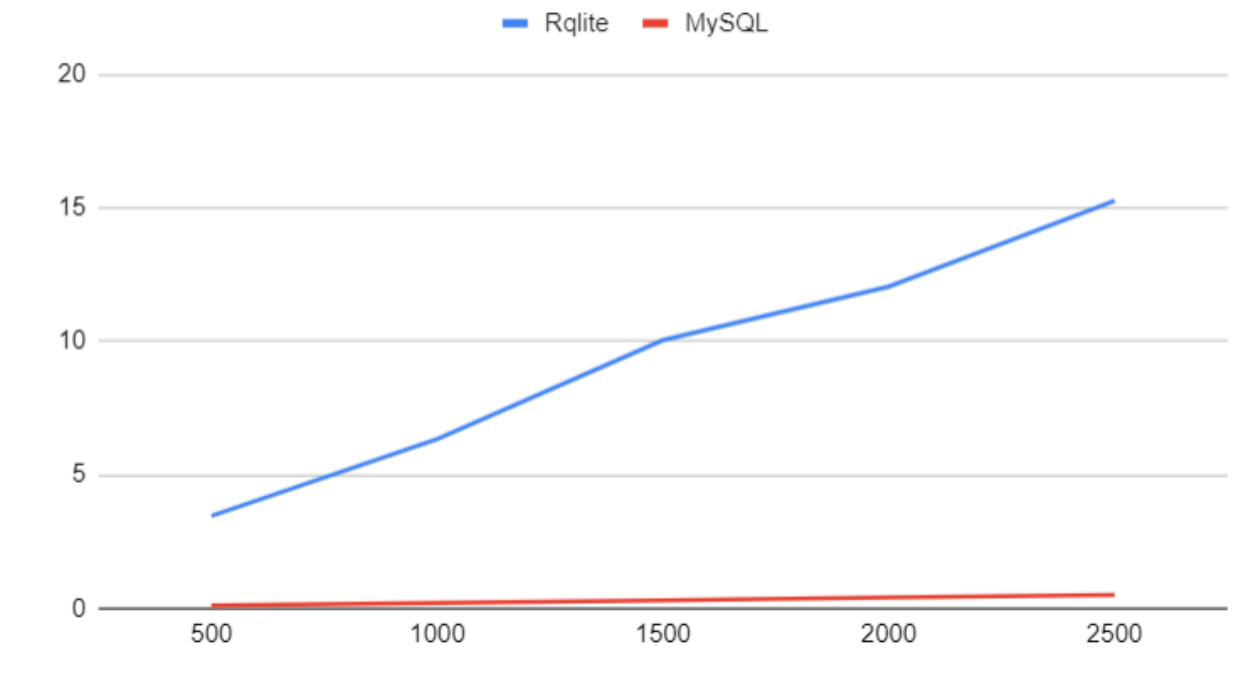
```
1 2024-05-18 02:10:49,174 - INFO - Inserting 500 records...
2 2024-05-18 02:10:49,282 - INFO - Time taken to insert 500 records: 0.11 seconds
3 2024-05-18 02:10:49,285 - INFO - Number of records fetched: 500
4 2024-05-18 02:14:22,715 - INFO - Inserting 500 records...
5 2024-05-18 02:14:22,819 - INFO - Time taken to insert 500 records: 0.11 seconds
6 2024-05-18 02:14:22,821 - INFO - Number of records fetched: 500
7 2024-05-18 02:14:24,763 - INFO - Inserting 500 records...
8 2024-05-18 02:14:24,868 - INFO - Time taken to insert 500 records: 0.11 seconds
9 2024-05-18 02:14:24,869 - INFO - Number of records fetched: 500
10 2024-05-18 02:15:50,826 - INFO - Inserting 1000 records...
11 2024-05-18 02:15:51,051 - INFO - Time taken to insert 1000 records: 0.22 seconds
12 2024-05-18 02:15:51,053 - INFO - Number of records fetched: 1000
13 2024-05-18 02:15:52,529 - INFO - Inserting 1000 records...
14 2024-05-18 02:15:52,736 - INFO - Time taken to insert 1000 records: 0.21 seconds
15 2024-05-18 02:15:52,738 - INFO - Number of records fetched: 1000
16 2024-05-18 02:15:53,897 - INFO - Inserting 1000 records...
17 2024-05-18 02:15:54,106 - INFO - Time taken to insert 1000 records: 0.21 seconds
18 2024-05-18 02:16:10,638 - INFO - Number of records fetched: 1000
19 2024-05-18 02:16:48,441 - INFO - Inserting 1500 records...
20 2024-05-18 02:16:48,748 - INFO - Time taken to insert 1500 records: 0.31 seconds
21 2024-05-18 02:16:48,752 - INFO - Number of records fetched: 1500
22 2024-05-18 02:16:49,619 - INFO - Inserting 1500 records...
23 2024-05-18 02:16:49,924 - INFO - Time taken to insert 1500 records: 0.30 seconds
24 2024-05-18 02:16:49,927 - INFO - Number of records fetched: 1500
25 2024-05-18 02:16:50,955 - INFO - Inserting 1500 records...
26 2024-05-18 02:16:51,253 - INFO - Time taken to insert 1500 records: 0.30 seconds
27 2024-05-18 02:16:51,256 - INFO - Number of records fetched: 1500
28 2024-05-18 02:17:47,867 - INFO - Inserting 2000 records...
29 2024-05-18 02:17:48,284 - INFO - Time taken to insert 2000 records: 0.42 seconds
30 2024-05-18 02:17:48,289 - INFO - Number of records fetched: 2000
31 2024-05-18 02:17:49,124 - INFO - Inserting 2000 records...
32 2024-05-18 02:17:49,537 - INFO - Time taken to insert 2000 records: 0.41 seconds
33 2024-05-18 02:17:49,541 - INFO - Number of records fetched: 2000
34 2024-05-18 02:17:50,598 - INFO - Inserting 2000 records...
35 2024-05-18 02:17:51,020 - INFO - Time taken to insert 2000 records: 0.42 seconds
36 2024-05-18 02:17:51,024 - INFO - Number of records fetched: 2000
37 2024-05-18 02:18:45,101 - INFO - Inserting 2500 records...
38 2024-05-18 02:18:45,601 - INFO - Time taken to insert 2500 records: 0.50 seconds
39 2024-05-18 02:18:45,606 - INFO - Number of records fetched: 2500
40 2024-05-18 02:18:47,071 - INFO - Inserting 2500 records...
41 2024-05-18 02:18:47,582 - INFO - Time taken to insert 2500 records: 0.51 seconds
42 2024-05-18 02:18:47,587 - INFO - Number of records fetched: 2500
43 2024-05-18 02:18:48,969 - INFO - Inserting 2500 records...
44 2024-05-18 02:18:49,479 - INFO - Time taken to insert 2500 records: 0.51 seconds
45 2024-05-18 02:18:49,484 - INFO - Number of records fetched: 2500
```



The below table shows the avg between RQLite and MySQL:

Insert	Rqlite	MySQL
500	3.473333333	0.11
1000	6.346666667	0.2133333333
1500	10.05666667	0.3033333333
2000	12.06333333	0.4166666667
2500	15.27333333	0.5066666667

Rqlite Vs MySQL



Conclusion:

In this experiment, we observe that MySQL outperforms RQlite in handling parallel writes. As the write load increases, MySQL consistently demonstrates superior performance, completing the write operations in significantly less time compared to RQlite.

For 500 writes, RQlite takes around 3.47 seconds, while MySQL takes only 0.11 seconds. As we increase the number of writes to 1000, 1500, 2000, and 2500, MySQL consistently takes less time compared to RQlite. For the largest test size of 2500 writes, RQlite takes around 15.27 seconds, while MySQL takes only about 0.50 seconds, which is significantly faster.

This is due to MySQL's optimized architecture and advanced mechanisms for handling concurrent write operations across multiple nodes in a distributed environment. MySQL employs more efficient techniques for managing concurrency control, locking, and write scalability, enabling it to process a large number of parallel write requests more effectively than RQlite. On the other hand, RQlite's performance degrades more rapidly as the number of parallel writes increases.

In conclusion, for distributed database systems that require handling a large number of parallel write operations, MySQL would be the better choice compared to RQlite.

Learning:

From this assignment, I learned how to set up RQlite and MySQL as distributed systems on a three-node setup. I understood that, even though MySQL outperformed RQlite, RQlite is ideal for simpler, smaller-scale distributed systems because it is very easy to set up and connect to the nodes. For testing a small prototype, RQlite is really easy to use. However, if the task demands large volumes of writes and significant scalability, MySQL is definitely the better option, as suggested by the test results.


References:

[1] https://www.scs.stanford.edu/17au-cs244b/labs/projects/muindi_stern.pdf

[2] <https://rqlite.io/docs/quick-start/>

[3] <https://www.mdpi.com/2076-3417/11/24/11590>

[4] <https://www.digitalocean.com/community/tutorials/how-to-create-a-multi-node-mysql-cluster-on-ubuntu-18-04>

[5]  Distributed Databases

[6] <https://www.scs.stanford.edu/20sp-cs244b/projects/Distributed%20SQLite.pdf>