Building Scalable Distributed Systems

# Assignment 4

Git Repository: <https://github.com/akksshah/building-scalable-distributed-systems>

# Server Design Description

I rolled back the server code to what I had in assignment 2. With that I also replaced my database from a MySQL hosted on an RDS to DynamoDB. The biggest advantage that I had with this design was that I was able to achieve single digit millisecond latencies for almost all of my requests when I tried to save them directly at the server end. There is a static dynamoDbMapper object available to all the threads that are servicing the requests. It is thread safe and so all threads can use it to write the requests to the database.

Database design

I have a single table that has the key made out a combination of id as the primary hashkey and date as the sort key. I also two GSI (Global Secondary index): One with the index hashKey of customerId, and one with the index with storeId as the hashkey.

This database design makes it very easy to query for purchases made in a store or purchases made by a customer as we won’t have to run a scan on the whole table. We can make effective use of partitions to reduce lookup times as we would be reading different partitions when querying for purchases.

# Runs:

256 threads

Text

Description automatically generated

Run with 512 threads

Text

Description automatically generated

1024 threads

Text

Description automatically generated

# Experiments and Inferences

Comparing the results that I had from assignment 2, what I essentially saw for the throughput for the client run for 256 threads is that the server simply provided a far better baseline throughput.

Chart, line chart

Description automatically generated

Simply switching out to a better database yield me more throughput. This was expected because Dynamo provides single digit millisecond latencies for all write request. For almost all of my request the time taken for a DynamoDB write was almost always less than 5ms.

Comparisons to Assignment 3.

My runs from assignment 3 showed the following results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Threads | Throughput (RabbitMQ) | Mean Response Time ( RabbitMQ) | Throughput (Dynamo) | Mean Response Time( Dynamo) |
| 256 | 5619.51 | 45.086 | 5276.33 | 47.01 |
| 512 | 5421.17 | 92.84 | 5296.55 | 92.33 |

What we can see that the results from Assignment 3 mimic very close to that of Dynamo inserts in Assignment 4. The reason is that the cost to write the message to the RabbitMQ broker in Assignment 3 is almost very close enough to the write latencies provided by Dynamo.

So, what we can actually see is that by choosing the server design as Assignment 2, but replacing the database to dynamo, provides us with close enough throughput as having a RabbitMQ broker as in Assignment 3. The biggest advantage with the new system would be is that now this design would provide us with strong consistency of the database rather than that of Assignment 3 providing eventual consistency.

With the newer design, we can minimize the cost because in total we would be able to reduce the cost of having 2 separate ec2 instances that were otherwise running as in Assignment3. That is, we would not need an ec2 running the rabbit MQ broker and another to consume messages out of the broker. Also, we would have better fault tolerance because the RabbitMQ broker can become a single Point of failure if we don’t have a backup broker which the servers can try in case of failures.

Depending on workload it is extremely easy for us to scale the system horizontally by replicating our servers in Assignment4 and registering them with the load balancer to ease out the load on each individual server. It is difficult to scale the Assignment 3 system, as the rabbit MQ broker can easily become a bottleneck. So, we would have to scale ourselves both the RabbitMQ broker and add individual servers at the same time and the consumers of the messages. Which in plain words simply would mean a far greater increase in cost than in Assignment4. It would be easy enough to say that scaling Assignment3 systems would incur a more significant increase in cost as compare to Assignment4.

# Further testing (Scaling even further)

So, I tried to scale up even further by deploying an additional of 3 more servers and registering them with the load balancers and see the effect on throughput

I was able to achieve the following throughput

256 Threads

Text

Description automatically generated

512 Threads

Text

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Factors | 256 threads | | 512 threads | |
| 4 server | 7 server | 4 server | 7 server |
| Throughput | 5276.33 | 7125.77 | 5296.55 | 8976.62 |
| Max Response Time | 47.01ms | 34.81ms | 92.33ms | 53.92ms |
| Median Response Time | 208ms | 9.8ms | 92.5ms | 12ms |

Scaling up the servers for the same load, we can see that the max response time is quite impacted and that doubling the load from 256 to 512 does not have a huge impact. In fact, the max response time almost halves with the additional servers

The most impacted performance indicator is that the median time for servicing a request in both the loads, get served within single digit millisecond latencies which is an astonishing improvement compared to the load when they were served using 4 servers. We get an improvement of 95% percent under 256 thread load and about 90% improvement under the load of 512 threads.