Assignment 4

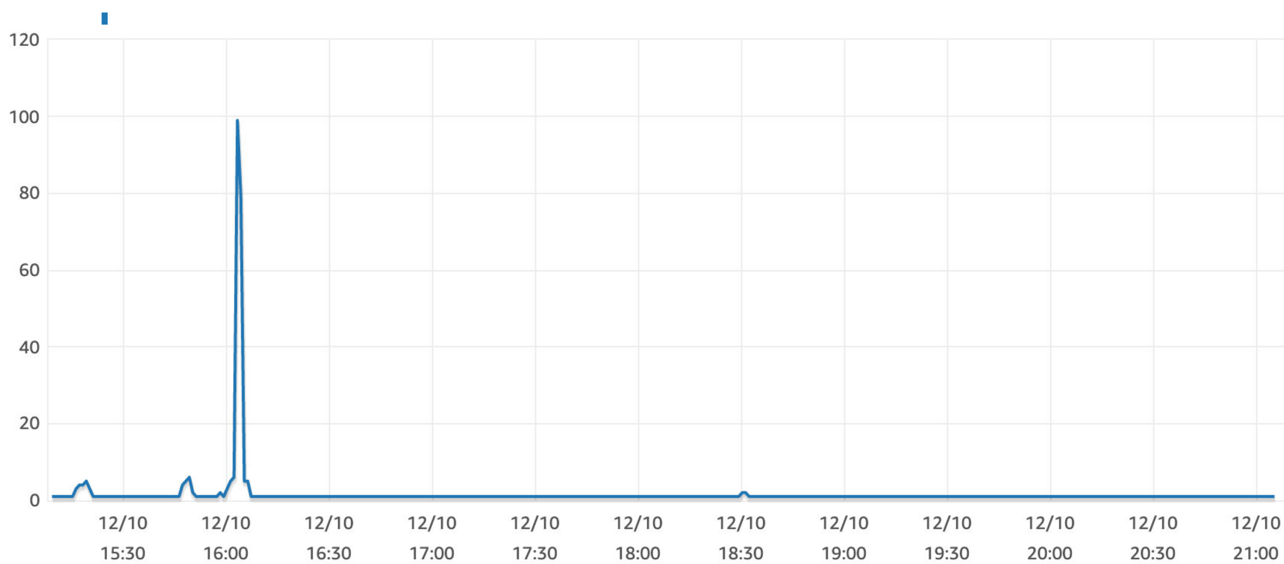
Github Repo

https://github.com/duskcloudxu/bsds2020fall_Assignment

Database update

> In this assignment, I switched the database from `MySQL` to `Aurora` on *AWS*.

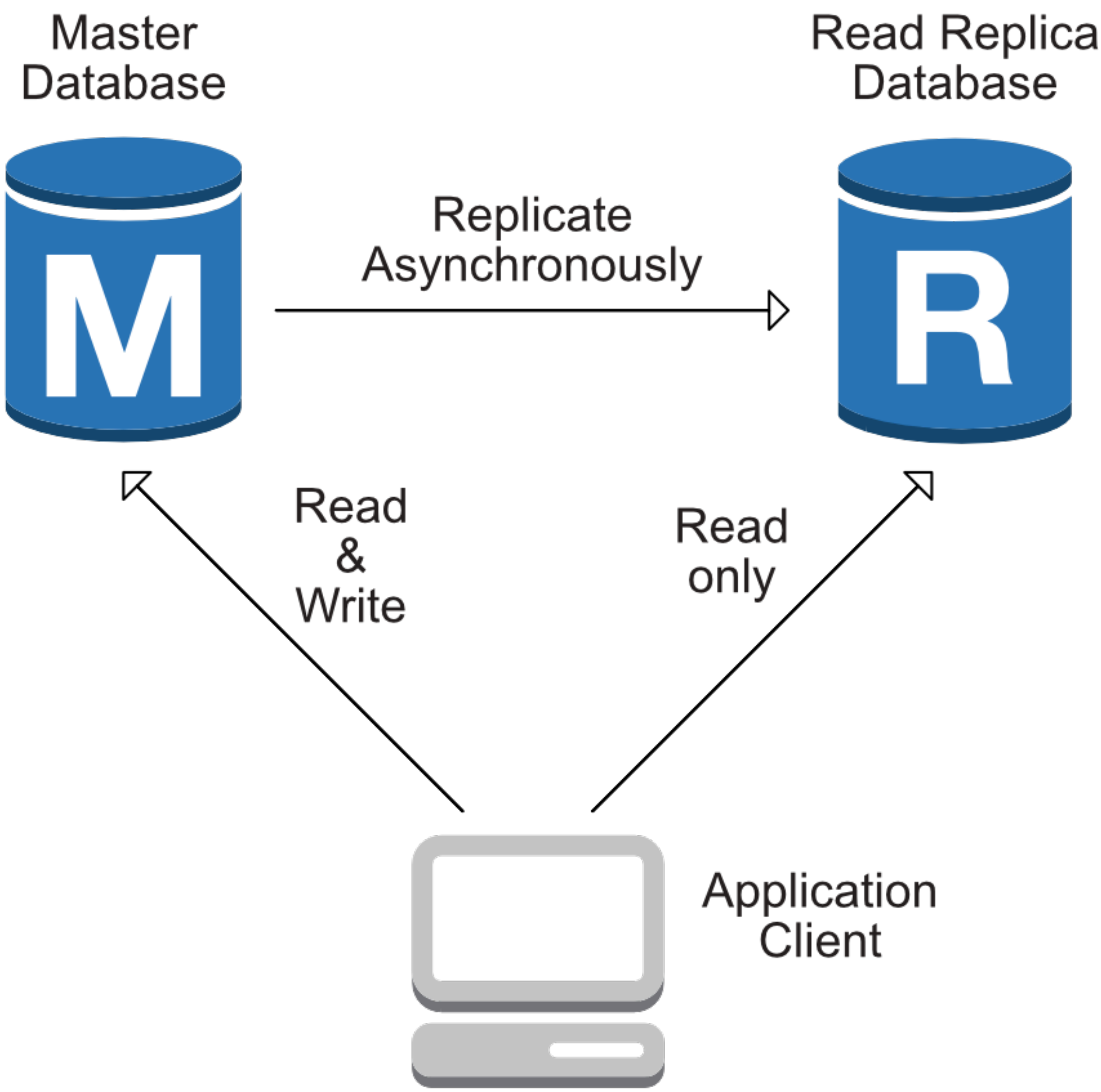
It's obvious that after we start testing on scale of 256~512 threads, the database became the bottleneck for our system, as suggested by the following screenshot of database CPU monitoring:



Thanks to the **message queue** we added in our last assignment, the mean latency of *POST* and *GET* request have increased in great amount. However, when we look into the consuming rate of ready message in the RabbitMQ, the metric is not very promising: the `ack rate` running on my local machine is only around 100/s. Which means for a test of 256 threads, which could produce ~200k messages, the server needs **half an hour** to consume all the messages cached in the *rabbitMQ*.

There are several solutions we could use. The first one is to run the consumers on several instance (as consumer node), MQ in a high performance instance(as MQ node). The consumer nodes consume message from the MQ node and it's size varies by the workload of MQ node. In the meanwhile, we could optimize our database.

we could ***separate writing and reading in the database***. For most database, the reading request is way more large than the writing requests, so the indexing could help to optimize the performance yet it require more time on writing. But if we could have a *reading replica* that the reading request would be redirected to and synchronize with the *primary database* (where the writing replica would be sent to) between a very short time (like 100ms), we could improve the reading spend without impacting the writing request latency.



Also, by creating replicas, we now are having a ***database cluster***. *With more instances, comes more resource.*

It would be a hard tasks to write such a database cluster decades ago, but now we have AWS and they provide aurora db, which supports MySQL instances. So instead of learning MySQL configuration to synchronize databases and dealing with all concurrency trouble, my work is only configuration and adjusting my back-end script daze.

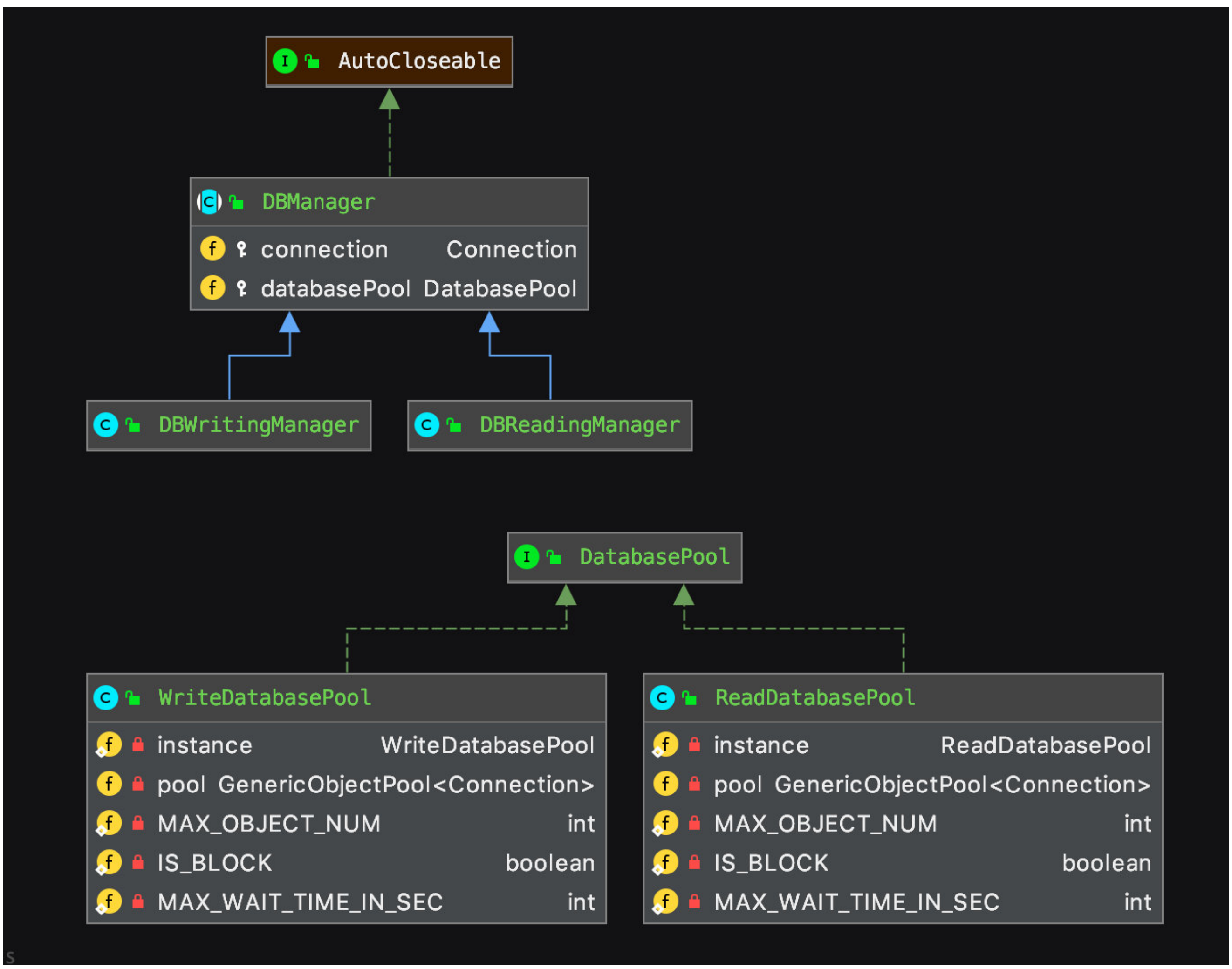
Related

Filter databases

	DB identifier	Role	Engine	Region & AZ	Size	Status
◉	database-2	Regional	Aurora MySQL	us-east-1	2 instances	Available
○	database-2-instance-1	Writer	Aurora MySQL	us-east-1c	db.r5.large	Available
○	database-2-instance-1-us-east-1d	Reader	Aurora MySQL	us-east-1d	db.r5.large	Available

New Added Modules

In previous assignments, we have `DatabasePool` to manage *database connections* in singleton pattern, and for this assignment, we should have two different Database connection Pool, one for connections to primary database, and another for connections to the read-only database. The same thing applies to the `DBManager` class, and the new class diagram as below:



As the UML shows, we made `DatabasePool` as an interface and `DBManager` as an abstract class, by doing that, we keep differnent configuration in different implementation class of `Databasetool` and use two subclass of `DBManager` to use those implementations.

Performance Comparation

numThread\metric	Mean Post Latency(ms)	Mean Get Latency(ms)	Median Post Latency(ms)	Median Get Latency(ms)	Wall Time(Sec)	Throughput(per Sec)	P99 of Post	P99 of Get	Max Response of Post	Max Response of Get
256 assignment2	1151	1506	1222	961	2341	165	2594	4472	11032	6369
256 assignment3	201	192	175	259	332	1155	5021	6326	6031	8185
512 assignment3	221	302	205	291	493	1325	6091	8941	9014	10011
256 assignment4	94	129	75	35	245	1579	247	636	344	661
512 assignment4	192	339	200	173	472	1638	394	6003	661	8035

Analysis

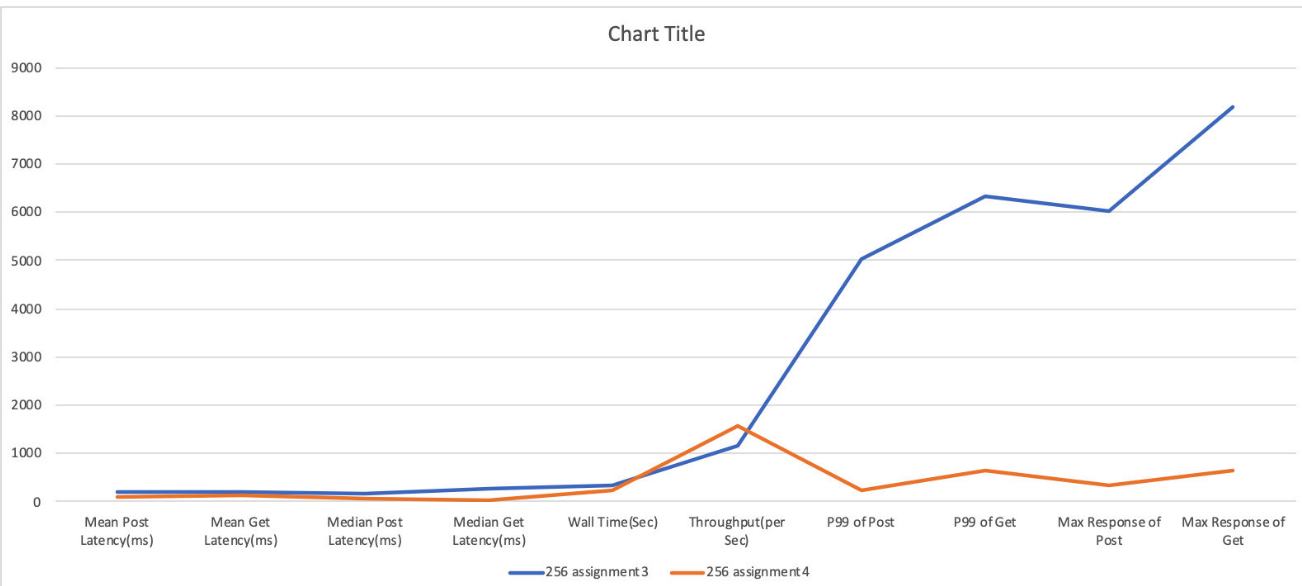


chart comparation of 256 threads test

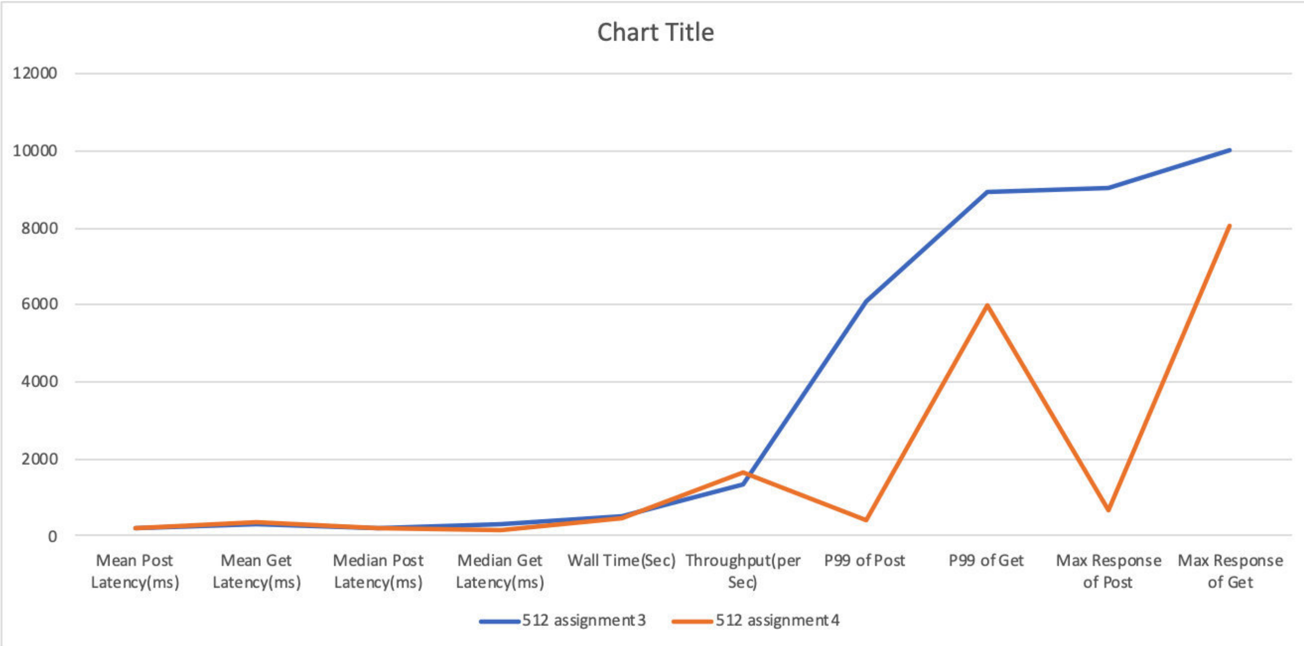
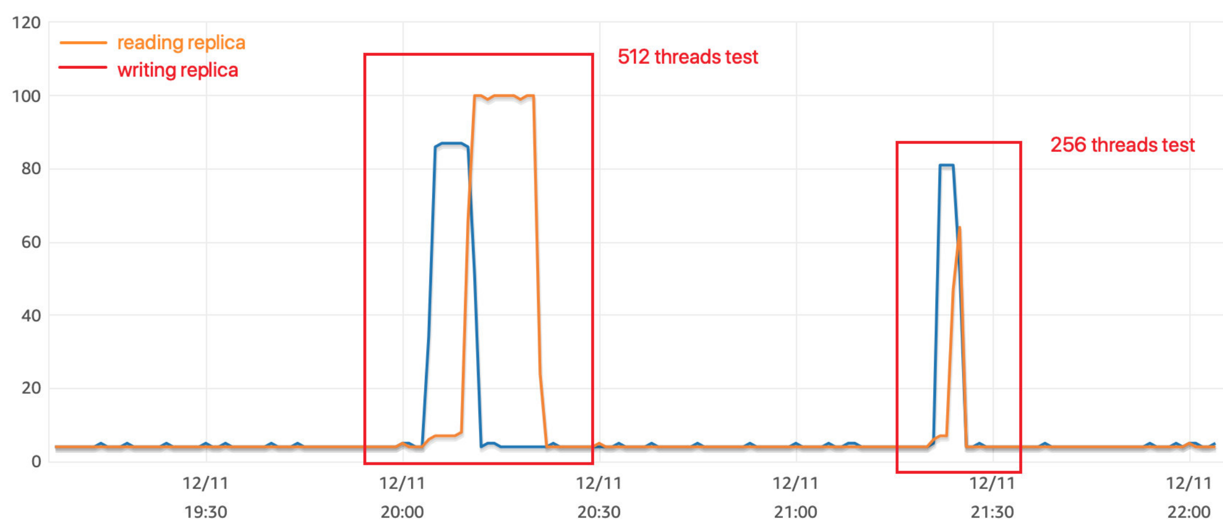


chart comparation of 512 threads test

By comparing metrics of assignment3 and assignment4 in 256-threads-test and 512-threads-test, we could approach the conclusion about database optimization:

- largely reduced the P99 time and max latency of the POST requests, which could be explained as we have a powerful primary database).
- For GET requests, in 256-threads-test, the assignment4 server has better metric than server implementation in assignment3. I guess it's because of write-read spliting. However, in the 512-threads-test, those metric do not differ as much as they do in 256-threads-test, which could be explained as the network traffic of 512-threads-test is still overwhelming for our reading replica. The hypothesis above could be tested by the CPU usage monitoring of Aurora database.



P.S.

In order to prevent costing too much aws credits (I still have a database instance hosting in that AWS account that necessary for another course), I will shutdown all the extra ec2 instance in the cluster after I submit this report, so please let me know if you want to test my server performance.