Member: Ibrahim Fazili
utorid: faziliib

How does it work?
For the software and hardware I used Golang to write my url-shortner and url-writer applications, Cassandra for the scalable database, Redis for caching, Docker for creating the swarm and routing the http requests, and bash and Python for scripting.

The url-shortner has all the necessary code to accept HTTP requests and has connections to the primary and secondary Redis and Cassandra clusters. This code gets packaged by Docker as images that can be deployed as replicas in the swarm.

The url-writer is also golang code which is active (Q2) when the url-shortner publishes the short and long to a channel in the primary Redis which it is listening to. It creates anonymous functions for each message that it sees when it awakes, this allows it to spawn multiple threads to handle multiple messages at once.

I leverage Docker' swarm mechanism to run multiple services together so the different services can interact with one another. Otherwise, when writing the code to connect the Go to Redis and Cassandra, I would need to pass the IP, but here I can just pass the DNS name which Docker does for us.

I have two kinds of instances for Redis, primary and secondary. We use the primary for writing to the cache as well as to publish messages. I chose this design decision because we already having so many different kinds of services on one machine that it did not make sense to have another Redis application purely for publishing and reading messages from. The secondary Redis is used purely to read data from the cache.

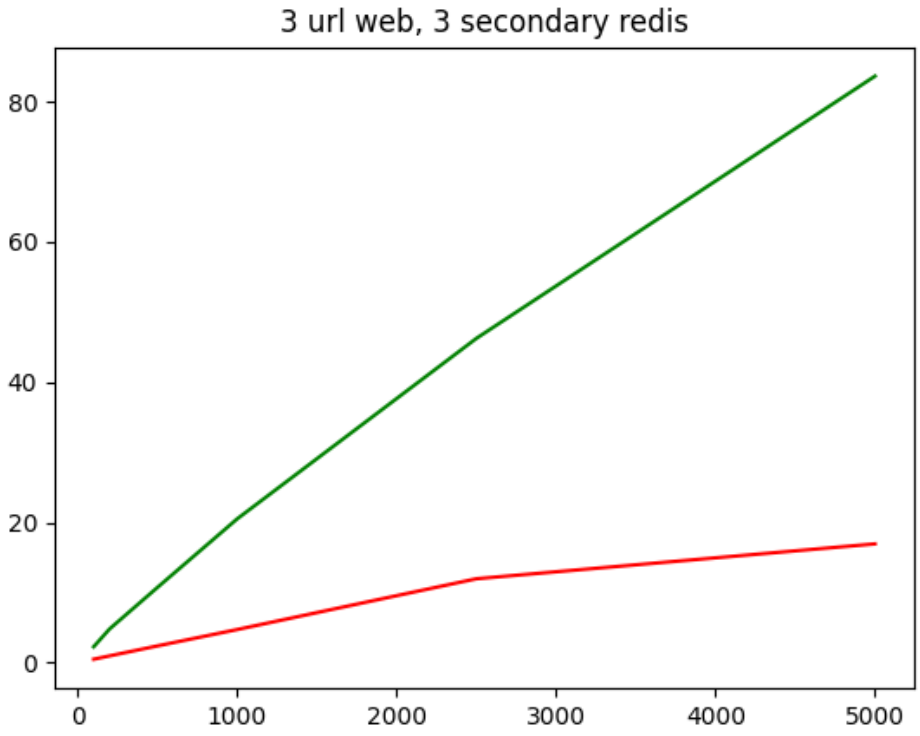
The Cassandra cluster is one service that is run outside the Docker swarm so it has to be stopped and started independently of the other services. This is because Arnold said so in the assignment specification. I only try to connect to one when intiializing the url-writer and url-shortner applications because once its able to connect to that one node, then it will be able to discover all and make requests to any of the Cassandra nodes.

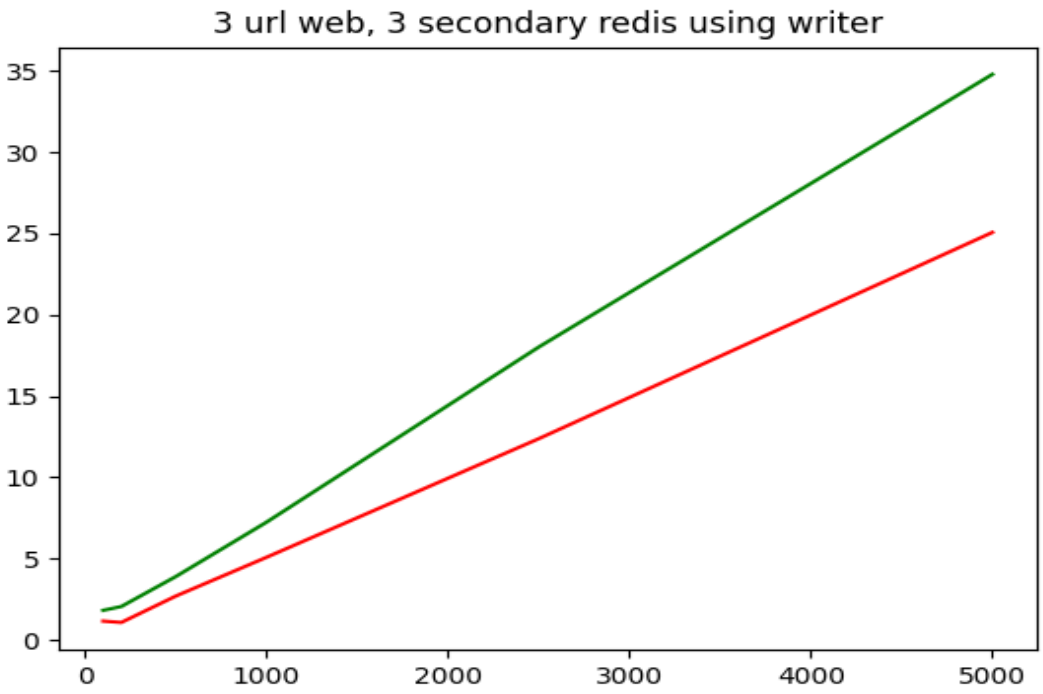
Strengths and weaknesses:

The strength of the system is that it is highly scalable, all it requires is a few commands to add/remove replicas to the swarm or more nodes to evenly distribute the services to or add/remove nodes to the Cassandra cluster. THis can be done on the fly and there is no need to take the whole system down to make the necessary changes. You are able to switch between using an external writer or not simply by changing an environment variable flag. A weakness I found was that in some startups there would be a few issues that could not be resolved, such as the first Cassandra node could not be started or in some cases when trying to execute PUT

requests it seems to timeout even though it just happened a few hours before the submission. The only ways that it could be resolved was through restarting the vm nodes and then launching the start script. A poor design in my part was the start script, if there was any errors, I did not get time to error handle so it was causing the entire interactive-ness to crash and so when trying to reboot you can face issues.

Reds are GETs and green is PUTs





As you can see from the graphs, using the writer definitely speeds up the execution time. My understanding is because in the first graph, where it executes without the writer, whenever it has to make a write request it needs to wait for a response from Cassandra before continuing and this can be expensive. On the other hand using the writer, the url-shortner just publishes it to the Redis channel where the url-writer can read from and continue its operations. The url-writer itself is multi-threaded so it can execute multiple writes to Cassandra and Redis on the fly.