Gerhard van Andel
CSCI 5273 Network systems

gMonitor

Spring 2020

Overview

Just another service level indicator (SLI), an SLI is a carefully defined quantitative measure of some aspect of the level of service that provides an idea on the health of the service. With this a SLI can be as simple as a web page load time. If a website does not load within three seconds the chance that the user gives up increases by 32% and goes to a different page. So monitoring traffic and response time from servers can be an important data point for web experiences across all industry sectors.

gMonitor will take a request for an endpoint that will send a list of urls to search. The service will then create a task for each url and using a pub sub model the web_bots will pull the web sites for this data and store it into google cloud storage.

Redis - used as a datastore for search data, and a lock manager

RabbitMQ - give tasks (web-site urls) to then read and parse

controller - web service to request website crawling to and search requests

spider - takes a task off of RabbitMQ (web-site urls) to get

scanner - takes a task off of RabbitMQ (web-site urls) to get

cleaner - takes the entries out of the cache and uploads them to google cloud storage

Goals

1. Monitor http get load times between web domains
2. Use google cloud platform to host in different regions around the united states to be able to test connections speeds on the same domain and endpoints from different regions to simulate east coast and west coast traffic.
3. Test a variety of industry web sites from sports to news to popular sites.
4. Pool the data to analyze the results of accessing websites from different regions.

Milestones

Checkpoint 1:

Have a working prototype pulling http/https based metrics. I will use python requests to pull data from a website then using beautiful soup to parse the web site, pull all the links from the site and then follow the links to determine site speed on leaf sites. Using redis to lock on the domain so workers can follow robots.txt and then store this data into google cloud storage. To test this ecosystem I used docker-compose this allowed me to make changes locally. This increased my productivity a lot as I no longer had to push the changes to the docker registry. I am still working on how to do the correlation across requests. To compare load times of the web sites.

Checkpoint 2:

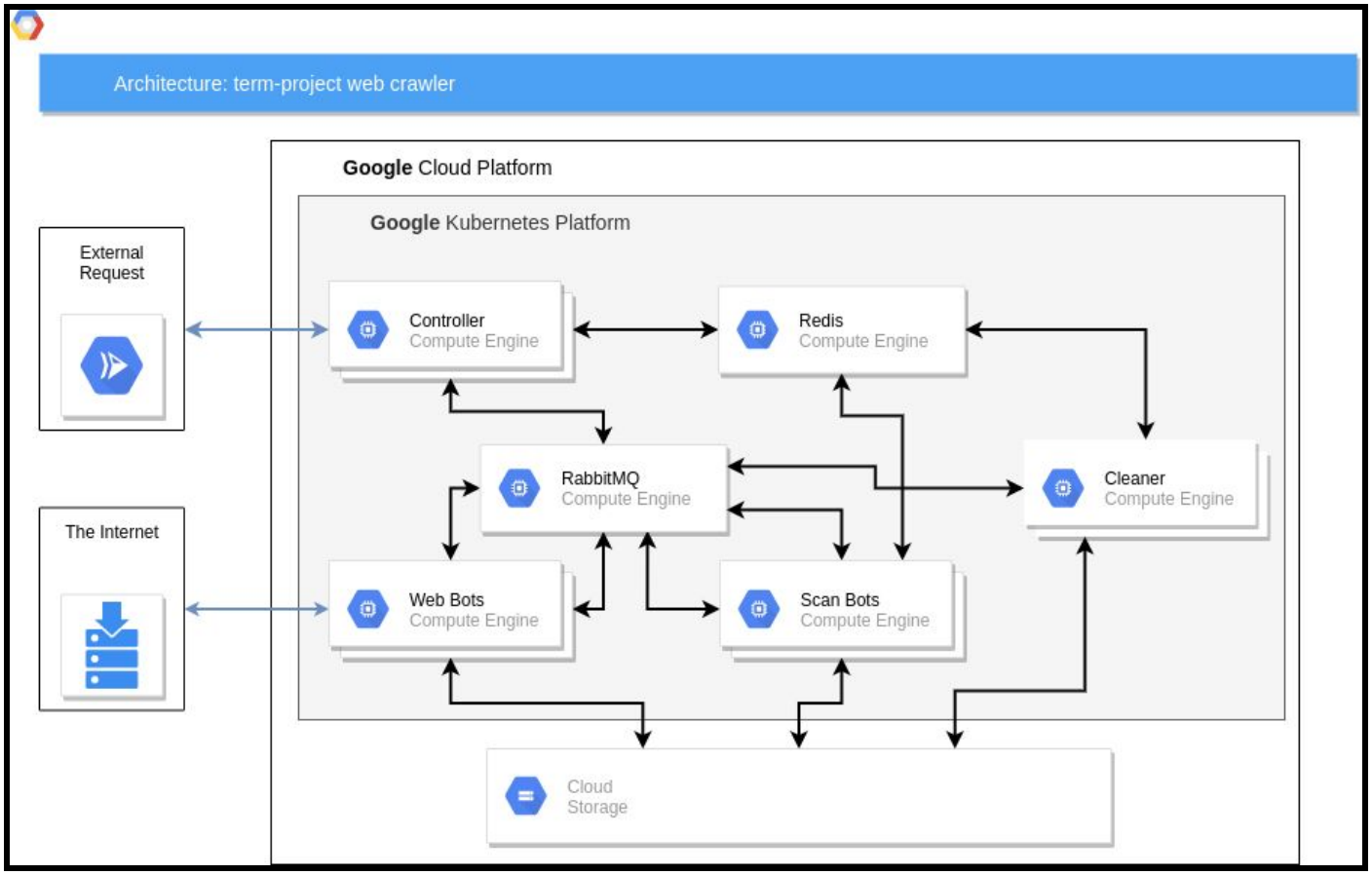
The redis database works well but I can fill it very fast as the crawler grows exponentially. To reduce the number of tasks in the database by pushing completed jobs back into cloud storage and on request load it out of cloud storage. I didn't think that link information would take up as much space as I thought but I crawled 70 web pages and that linked to more than 1200 other pages so I may have to think how to store more information elsewhere. Have two or more regions working to poll web traffic from more than one location. Then figure a way to collectively pool the data sources. I would like to keep the data locality, something that I don't need to share between regions.

Checkpoint 3:

I found the Redis database has an expiry when you add an entry into the database then will expire the data field. I was able to reduce the domain database from growing very large using this as the domains are generally grouped as you will have links from one page linking to pages within the same domain. This also allowed me to then get an updated version of the robots.txt after the domain would expire from the database. I also used the expiry of the domain data to ensure the release of the domain lock. If I did a deploy in the middle of a search I could have a deadlock on a message, so putting an expiry on that would ensure domain locks would be released at some point. The domain locks and rabbitmq found a separate issue in release of the message I was looking for a way to put the message at the back of the queue but from what I saw if a domain lock was being used the message would be rejected and then placed back on the message queue only to get picked back up right away. I think if I used priority in the queue I could then lower the priority of a message along with track the number of times I have seen that message would be techniques I could use to optimize around this. Offloading completed requests from the redis database into cloud storage was also a big help as a small amount of jobs are pulled after they complete. I did look at using Google's big-table service but found that redesigning the platform for the use of big-table was more than I wanted in the last two weeks of the assignment. Along with this I found I could do a better job loading content from a website so that I could also index the loading of other content being style sheets, images, and javascript.

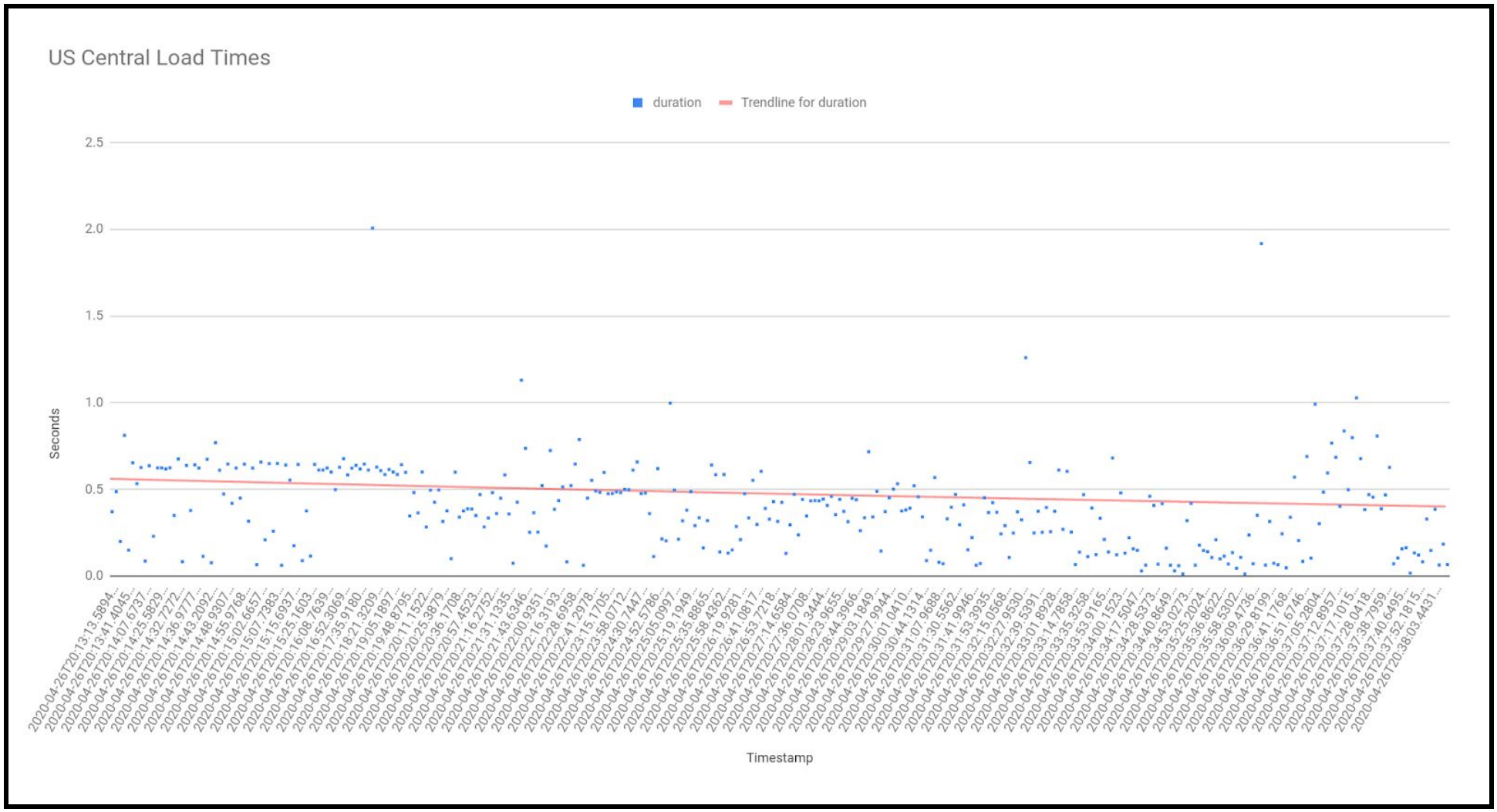
Architectural Diagram

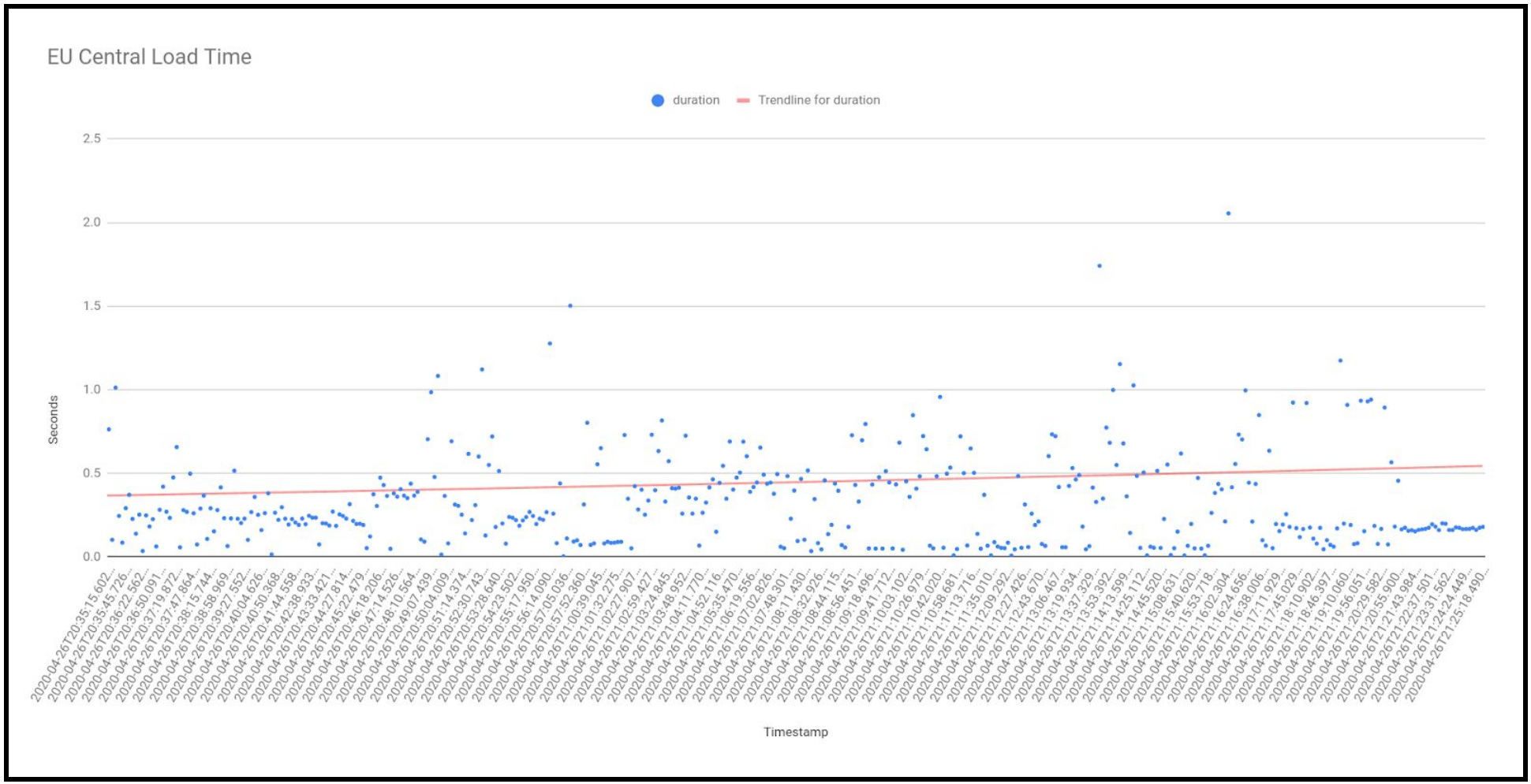
Google Kubernetes Engine (GKE) provides a managed environment for deploying, managing, and scaling your containerized applications using Google infrastructure.

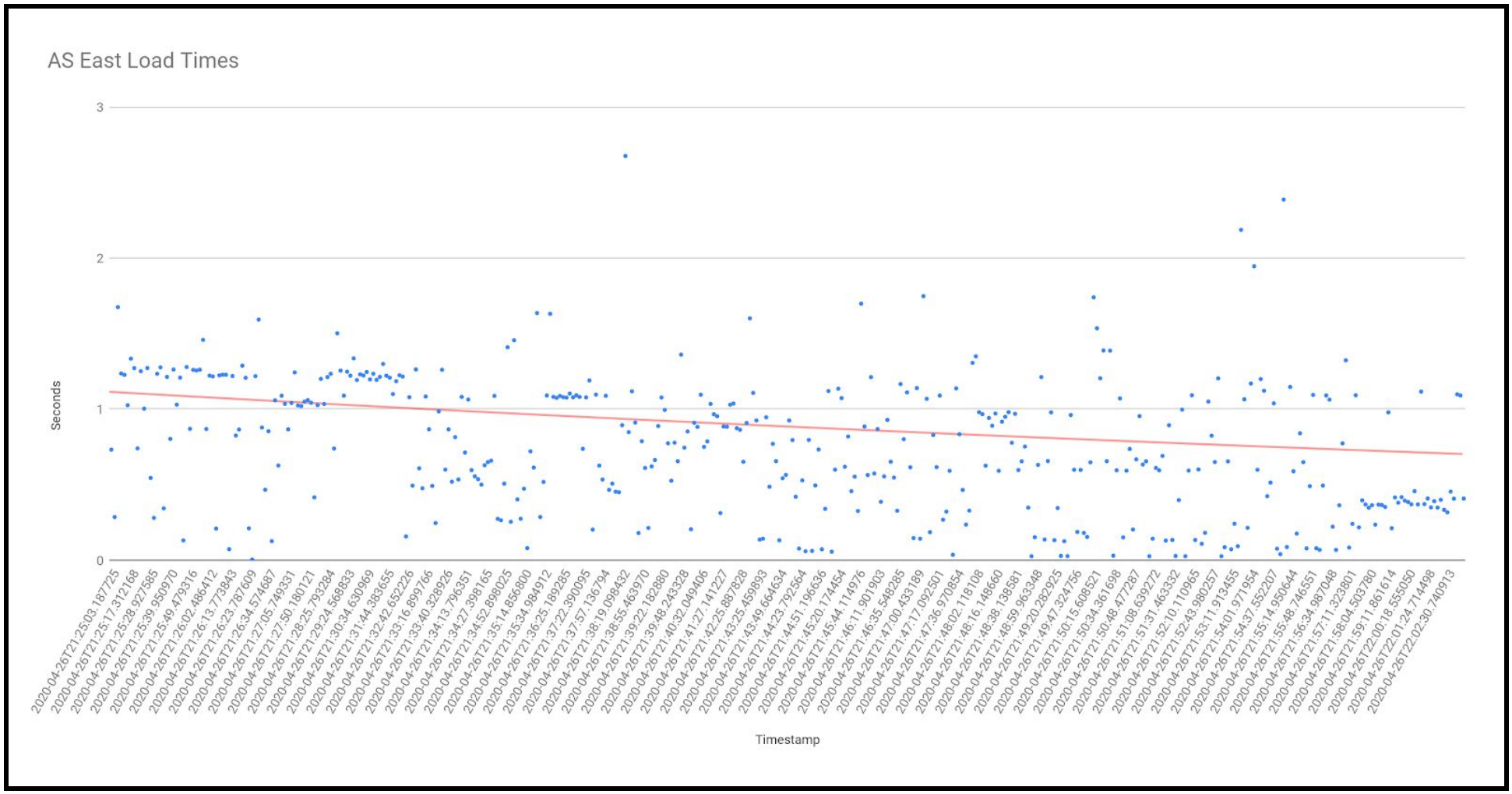


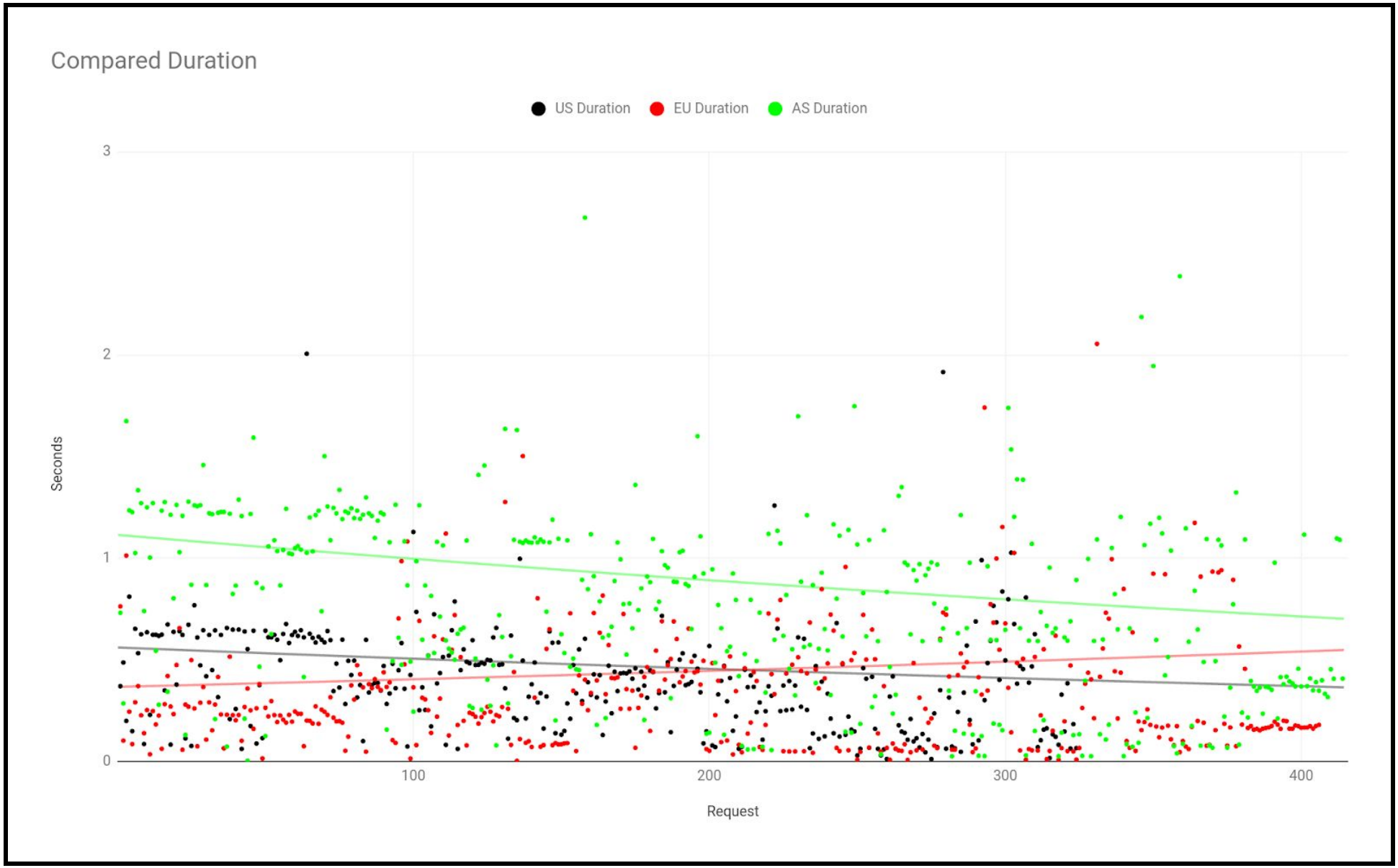
With this same template in three regions, united-states central, europe central, and asia east. In these three regions I then tried to keep my requests similar but with the nature of the distributed system I did not control the queues and what jobs were being worked.

Results


US Central Load Times
duration
Trendline for duration
2.5
2.0
1.5
1.0
0.5
0.0
Seconds
Timestamp


EU Central Load Time
duration
Trendline for duration
2.5
2.0
1.5
1.0
0.5
0.0
Seconds
Timestamp


AS East Load Times
Seconds
3
2
1
0
Timestamp


Compared Duration
US Duration
EU Duration
AS Duration
3
2
1
0
Seconds
100
200
300
400
Request