Project - Phase 5 Stress Testing

Group 14
Tiago Carvalho fc51034
Diogo Lopes fc51058
Miguel Saldanha fc51072
João Roque fc51080
João Afonso fc51111

14/04/2021

1 Introduction

To perform an evaluation on the system by stress testing, we used three different technologies: Prometheus, Grafana and Locust.

Prometheus is a monitoring system that retrieves data directly from the cluster as it is executing, while Grafana displays and amalgamates the data in a way that can be analyzed by the human eye (plots, etc.).

To perform the stress tests on itself, we used Locust, a stress testing tool.

All these tools were configured automatically with scripts to further automate the deployment of the system (see locust.sh and add_plugins.sh).

2 Evaluation

We ran Locust, the tool to perform the stress testing, in another cluster to avoid affecting the results.

It was tested with up to 800 users, which at that point it was getting close to 100% CPU usage, according to Grafana. At the time, we were using weak machines (12 CPUs) as Google Cloud's free trial did not allow any higher than that. We later got access to better machines and so we tested with up to 6800 users, which was still limited by quotas since we couldn't spawn more users to do so.

Every aspect of the system was being tested: getting entries, putting new entries, get pages from the database, get a recommendation (and even the login and logout which was later changed and so there is no need for load testing as it is handled by an exterior service, auth0). We did stress testing with 100 users,

500 users, 800 users, and later, when we had access to better machines, up to 6800 users.

3 Validation

With 800 users the system reached around 75% CPU usage and 2000 to 2500% system load, which we assume to be a somewhat comfortable limit for the resources we have, it can go higher, but it could probably start running into performance problems after a certain point (images in appendix **??**).

The latency remained good nonetheless, around 2 to 4ms. Disk usage was always very low as most of the data was not stored in the cloud but rather on an exterior database (MongoDB and MySQL). There was a high usage of memory though, probably to store all the requests (around 14gb).

When we later tested with 6800 users, we got to around 600% CPU usage, which is equivalent to 6 CPUs running at their full potential. System load had spikes of up to 400% and the memory usage was around 35%. Since most of the load was concentrated on the api-gateway, we increased it's scaling capabilities immensely, with it getting to 100 pods, give or take. As mentioned before, our system could be tested with a bigger load, but unfortunately we were limited by the quotas.

4 results

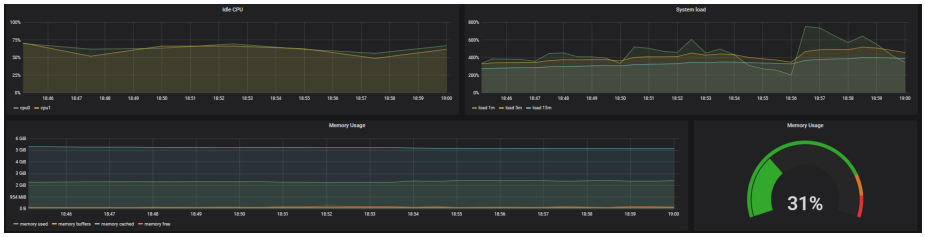

Figure 1: Stress testing with 800 users. CPU usage and system load plots

Figure 2: Stress testing with 800 users. CPU usage and system load plots

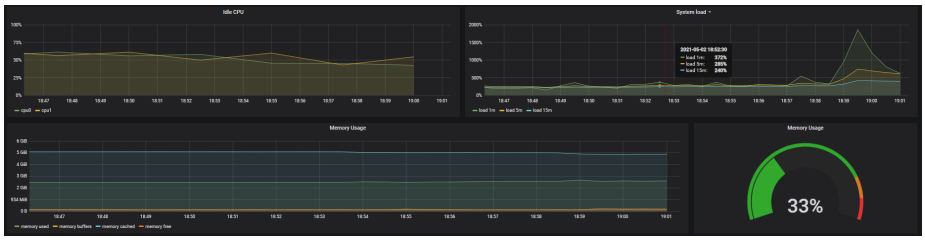

Figure 3: Stress testing with 800 users. CPU usage and system load plots

Figure 4: Stress testing with 800 users. CPU usage and system load plots

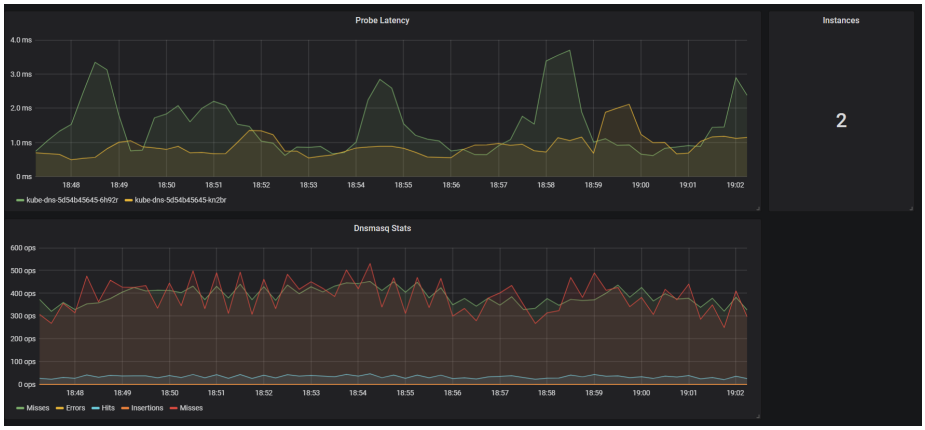

Figure 5: Stress testing with 800 users. DNS latency

Type	Name	# Requests
GET	/item/BOOK/607615b3aeb60e0f26f7c1df	2190
PUT	/item/BOOK/607615b3aeb60e0f26f7c1df/like	2229
PUT	/item/BOOK/607615b3aeb60e0f26f7c1df/seen	2230
GET	/lib/1	2358
POST	/suggest	2285
GET	/user/login	2237
GET	/user/logout	2234
PUT	/user/search/saldanha	2253
	Total	**18016**

Figure 6: Stress testing with 800 users, requests and count of each

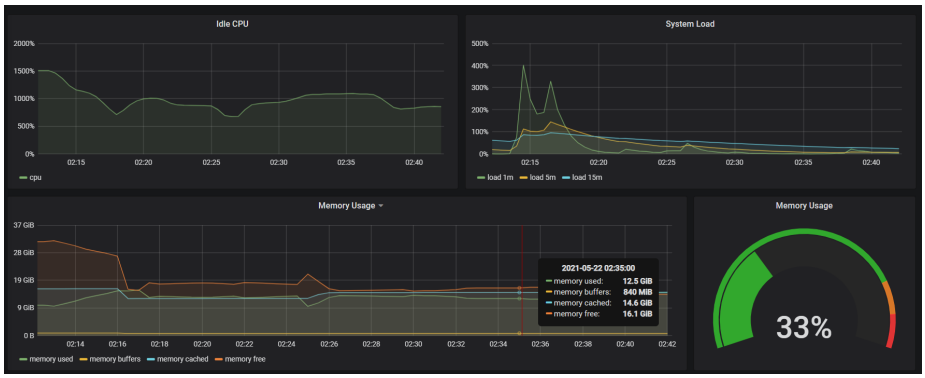

Figure 7: Stress testing with 6800 users. Idle CPU and system load plots

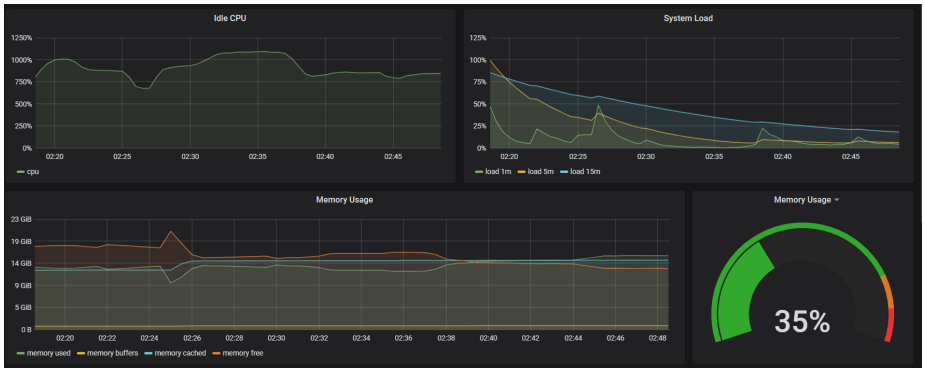

Figure 8: Stress testing with 6800 users after the system stabilized. Idle CPU and system load plots