Team Garuda
CSCI-B-649
Spring 2022

Optimization using Object-Store

Architecture Analysis Case Study

We think that the scientist using this tool to understand weather patterns will need to compare multiple plots of different variables, date and time to make some observations. To make this possible we need to retrieve multiple data points quickly.

Previous implementation stored the Nexrad dataset in MongoDB and when a user requests to plot the data, the fetch API retrieves it from MongoDB and plots it on the map on the web app.

Following were the downside of the previous implementation:

- We were restricted to reducing the time range in the request as MongoDB had a 16MB limit for a document.
- The backend had a lot of load as it had to transfer in the worst-case scenario 16MB of data for 1 request to the web app. 16MB for one request is a huge number.
- In the case of multiple concurrent requests it would lead to unavailability of resources or high response time for the user.

Solution

- To overcome the problem mentioned above we introduced Object Store. Now, the system stores the Nexrad dataset downloaded for a request to the object-store rather than in MongoDB.
- For object-store, we are using an S3 compatible API from Jetstream. Now, we are only storing the S3 bucket object's URL in MongoDB, and on request from the user to plot the map we retrieve and return the S3 bucket URL.
- Then, on the web app, render the data directly from the object store and plot it(The same flow as in the previous version).
- The good thing about this decision is that we didn't have to do any major architectural changes.

Advantages/Improvements on architecture

- Reduced load on the backend which in turn leads to more availability and low response time for the user
- We can increase the NEXRAD/MERRA2 data time range in the request, ie. large data set in one request.
- Later on, we can use CDNs layers to cache objects in Jetstream object store

Analysis

In order to analyze the approach we made 100 concurrent requests with the same parameters to the architecture with and without the object-store feature.

The results were far better than we expected.

Requests result without the object-store

No. of requests	Average time(ms)	Minimum time(ms)	Maximum time(ms)
100	14194.34	2082	22946

Requests result with the object-store

No. of requests	Average time(ms)	Minimum time(ms)	Maximum time(ms)
100	399.63	141	566

*The average time was reduced by around **135%***

As you see from the graphs below the response time varies from 2000 ms to 22000 ms without the object-store feature (Ref. Fig 1.1) and in this case, with the object-store feature, the response time varies from 100ms to 600ms (Ref. Fig 1.2).

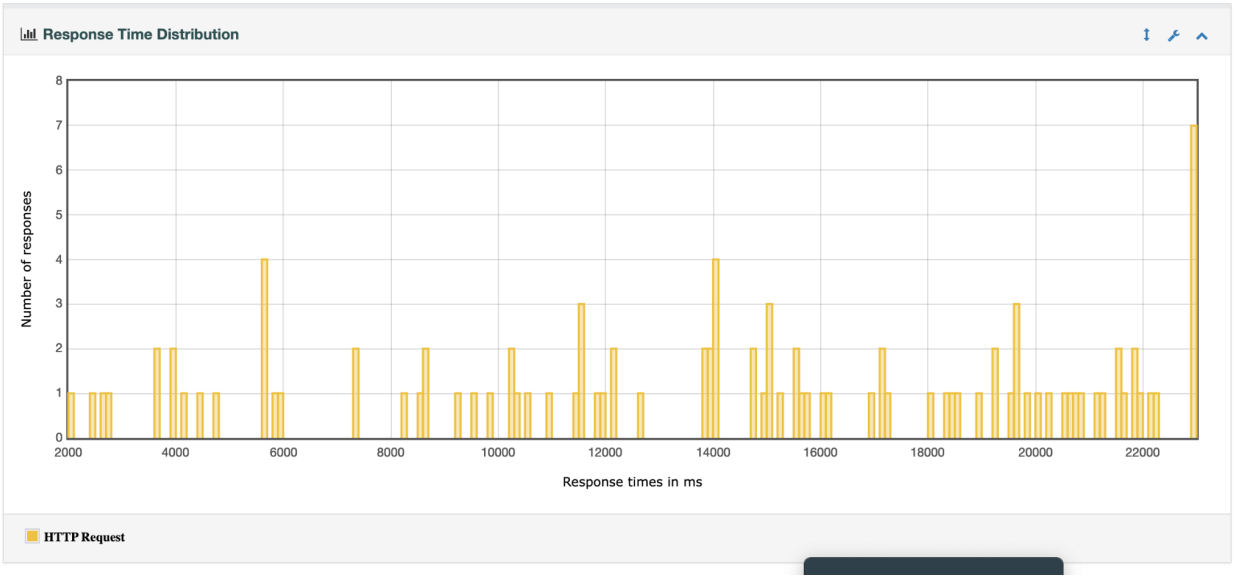

Fig 1.1 - Response time distribution without object-store

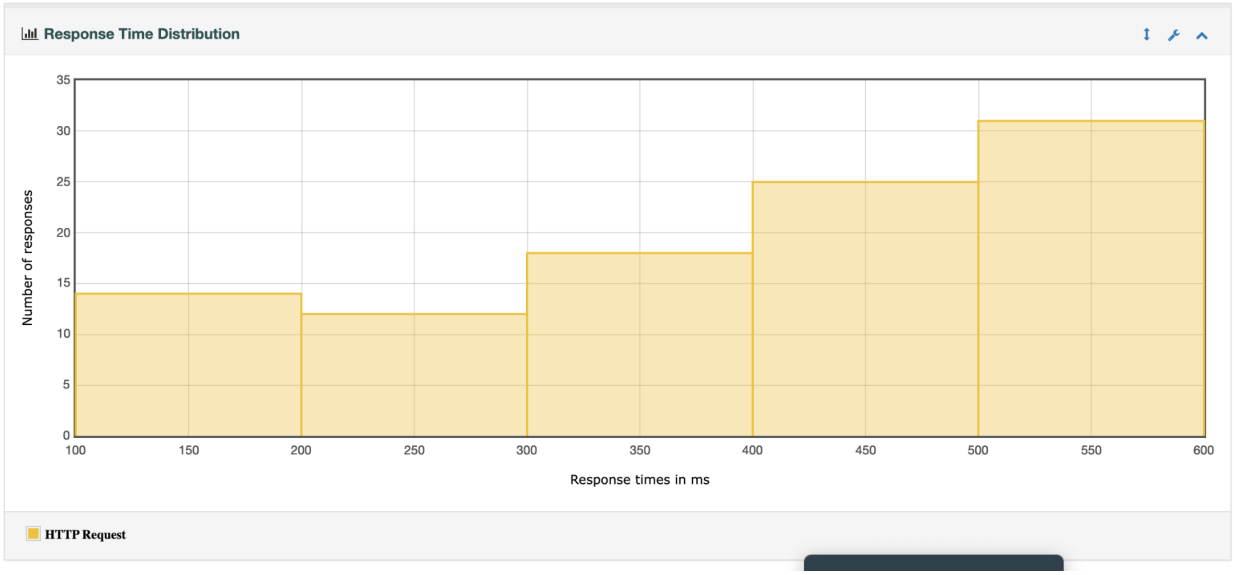

Fig 1.2 - Response time distribution with object-store

In the case without the object-store feature, the response time of all the requests is more than 1500ms (Ref. Fig 2.1) whereas with the object-store feature around 70% of the requests response time is less than 500ms (Ref. Fig 2.2).

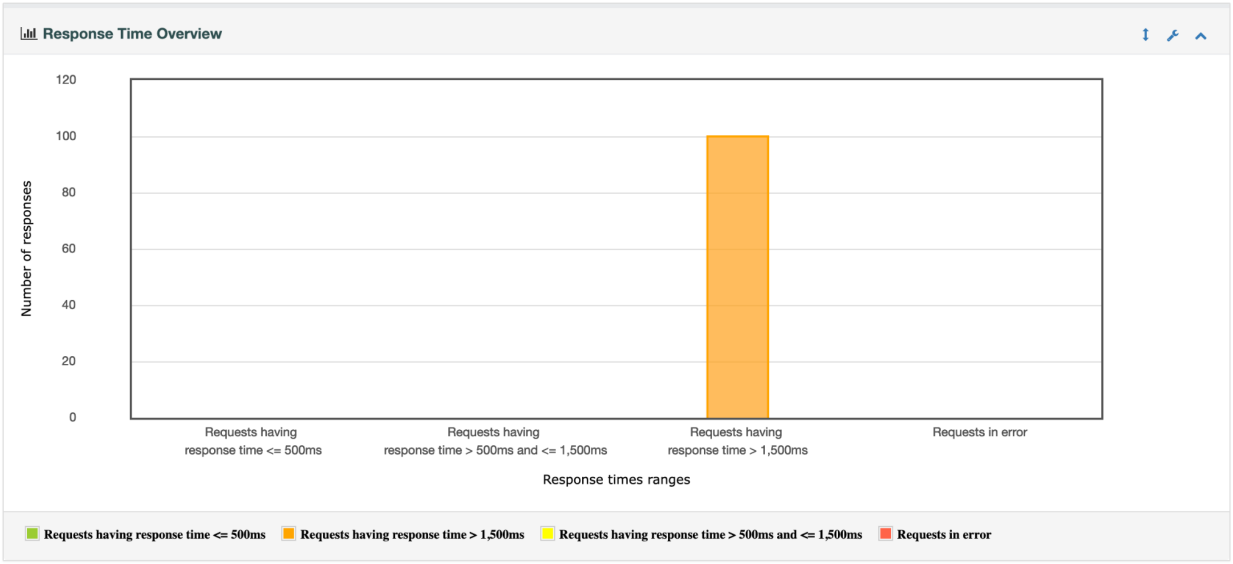

Fig 2.1 - Response time overview without object-store

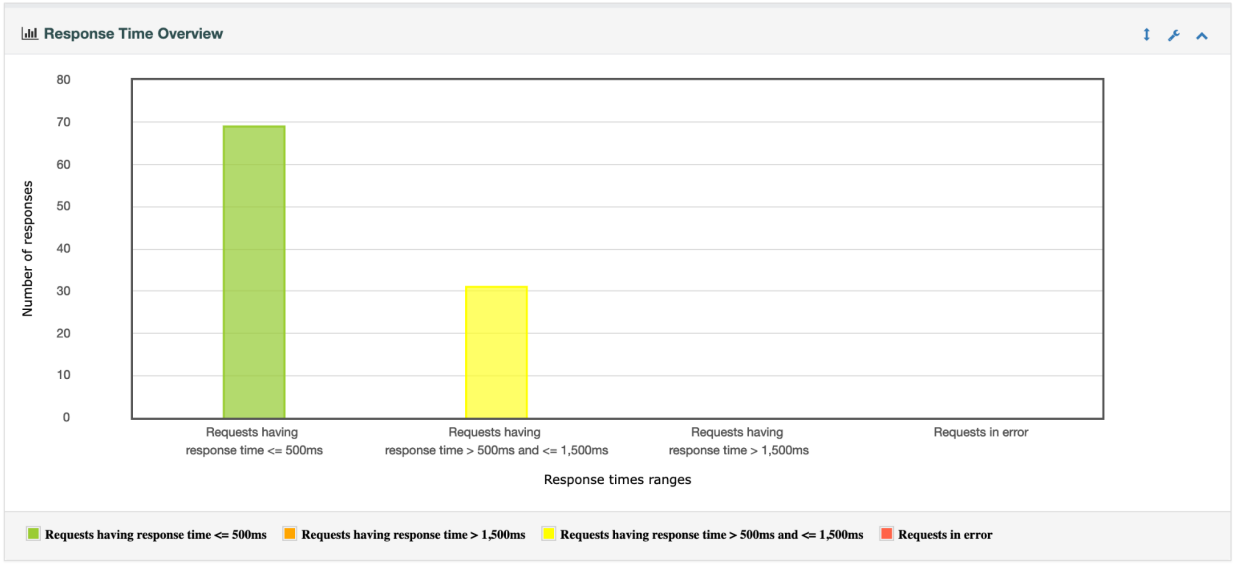

Fig 2.2 - Response time overview with object-store

Disadvantages of the Object store solution

- Processing of data becomes difficult after upload. If processing is required, the object needs to be downloaded on the server, processed and uploaded again.
- Management of objects in object stores. Need to set access layers and view policies.

As the data is available publicly we didn't have to worry about data access layers and authorisation and since, the advantages outweigh the disadvantages we decided to move forward with integrating object store to our architecture.

For a detailed report please follow the path mentioned below on the github repository:
./**tests/load_test/object-store_optimization**