Elba Experiment Data Processing Orchestration

A Framework for Batch Processing Logs

CS4365 Spring 2018 Project Report

Team Members: Neel Jha

I. Motivation

The Elbalang Interpreter (Elbalang) is a recent development in the Elba project. Elbalang is an interpreter designed to parse non-standardized output logs from the benchmarking and logging tools used during an experiment. The goal of this interpreter is to make data easily digestible for researchers looking to publish the findings of their own experiments, as well as the academic community looking to run a pure quantitative analysis on the experiment data.

Presently Elbalang is a single-threaded interpreter with a runtime of approximately ten seconds per log file. There are over ten thousand logs from previous experiments stored on-premises at Georgia Tech. Given this constraint there is a need for a cloud-based parallelized batch processing system.

II. Design

The system consists of two cloud storage buckets, a cloud function, and a Kubernetes cluster.

Files to be processed will be uploaded to a cloud storage bucket with a lifecycle of 24 hours configured. An anonymous cloud function will trigger once a file has been uploaded. This cloud function will make an HTTP request against Elba-API, a RESTful endpoint, which is deployed to the Kubernetes cluster. Elba-API will then process the log and save the output in a cloud storage bucket.

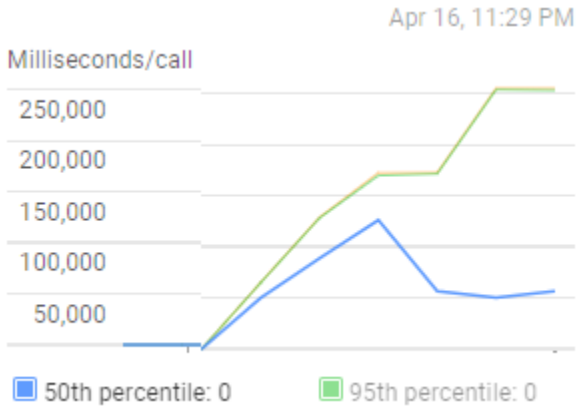
The Kubernetes cluster exposes the Elba-API via a load balancer and has autoscaling configured. The node pool contains at least one node and can scale up to five nodes. These nodes can run anywhere from two to thirty pods. Kubernetes handles the instantiation of additional nodes and pods to scale up and the decommissioning of nodes and packing of pods to scale down automatically. The target CPU utilization for this workload is 60%.

These cluster min/max scaling sizes come from a best guess estimate of the workload

and the capacity of an individual pod. The CPU utilization target comes from a metric Turtle Rock Studios found to be successful in their autoscaling deployment [1].

III. Results

The median processing time per file during a benchmarking run of uploading 100 files was about 10 seconds once the cluster had scaled up. The figure below shows the median and 95th percentile processing time with respect to time.



The execution time caps out at 240000 milliseconds because of a hard cap on the execution time of the cloud functions. The cloud functions making HTTP requests against Elba-API would hang until they received a response. That time was used to measure processing time. However, under load and as the cluster scaled, some requests were delayed.

I had initially thought that autoscaling would be able to handle the load seamlessly and that is why I opted to remove the queue from my initial design. Perhaps with greater finetuning autoscaling could seamlessly handle the load. The more correct approach appears to use a queue and let more workers spawn in response to high CPU utilization by active workers.

While some requests had longer than expected response times, all the requests were processed as evidenced by the processed output arriving successfully in the bucket.

IV. Future Work

Future work would be to optimize for the long running time of log file processing requests against the Elba-API. The easiest way that directly ties into this work would be to put a queue between the Elba-API and cloud functions. This would eliminate cloud function timeouts for long running log processing since enqueuing a file path is a faster operation than waiting for an HTTP response on a potentially long-running request. This reduction in running time translates to a cost savings as runtime and invocation count factor into the cost of Cloud Functions.

The second method would consist of modifying all of the existing infrastructure and turning the Elba-API cluster into a MapReduce job. Batches of files could then be submitted to this cluster for rapid, distributed processing. This upgrade path would be quite involved, required a rewrite of Elbalang and a complete rearchitecture of the entire application to move from Kubernetes to a MapReduce cluster.

References

[1] Wes Macdonald, “Hunting Monsters in a Low-latency Multiplayer Game on Amazon EC2”, In Proceedings of AWS re:Invent, Las Vegas, NV, USA. October 2015