Progress Report -- Checkpoint 1

Rongrong Miao, Jingwu Xu, Saurabh Goyal

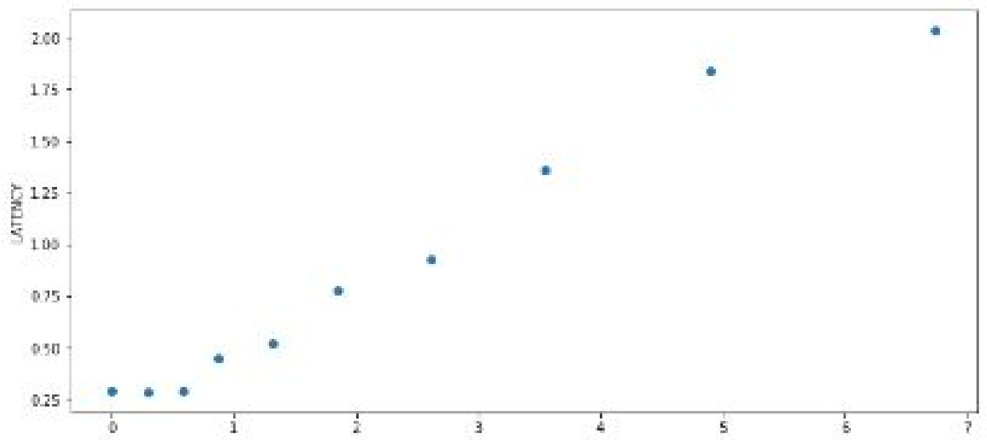
Work Related to the Proposal Designing Experiment

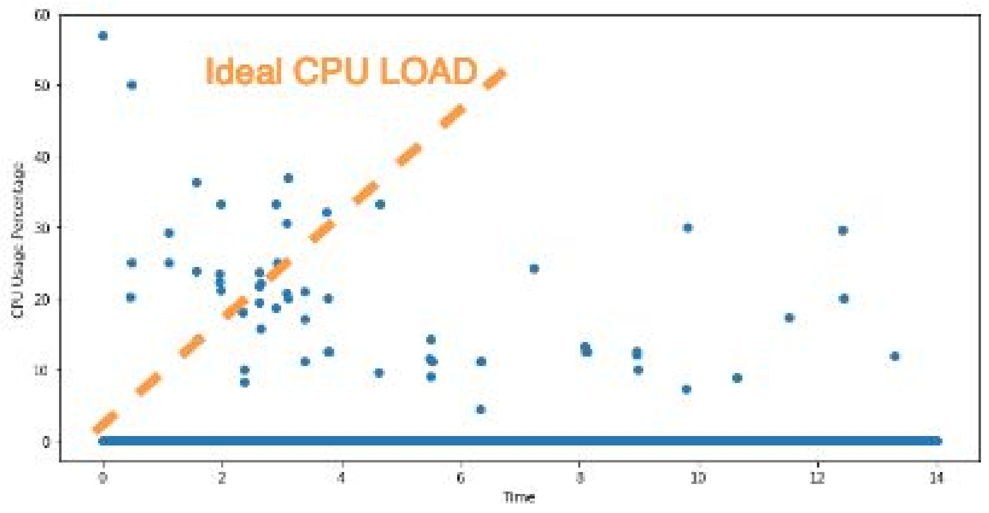
1. Wrote a proposal and discussed high-level plans.
2. Met with professor and TA, got feedback on the proposal, restricted our load balancer implementation on HTTP level, rewrote proposal.
3. Researched on how to implement load balancer on HTTP level, more specifically, we need to measure the performance by comparing average response time with different request distribution algorithms, handle server failure and redirect requests, consider maximum connection limit at server side, maximize server utilization. More details in this area are described in our revised project proposal

Work Related to the Experiment Setups

Server side implementations:

1. Had endpoint(s) that can accept GET request and return JSON response.
2. Simulated server load by spawning threads to run dummy computational task
3. Used open-source HTTP load generator VEGETA to measure the latency. We wrote shell scripts integrating with this tool that can generate HTTP requests that will hit the endpoints at user-specified request per second, and we record the latency of each request sent, as well as the status code. We can take the measurement and generate plots in real time. We analyzed the results by plotting in graphs and histograms. See results section for more information.
4. Tried to measure performance statistics (CPU Load, RAM Usage, etc) on a server
5. Explored the relationship between server CPU load and request response time, (see the figure below), but it turned out we could not capture correspondence between these two relationships. (ideally, CPU load is high when response latency is high)
6. Designed different server load pattern by using dummy workers/threads. By server load pattern we meant we manufactured the load distribution of server given time. In this project, we mainly explore the server load that changes periodically. We will design our own load balancing algorithm based on this assumption.





In this graph we can see even though when the server is busier, the latency increases, we don't see a corresponding pattern in cpu load increase.

Work Related to Load Balancing

We have accomplished:

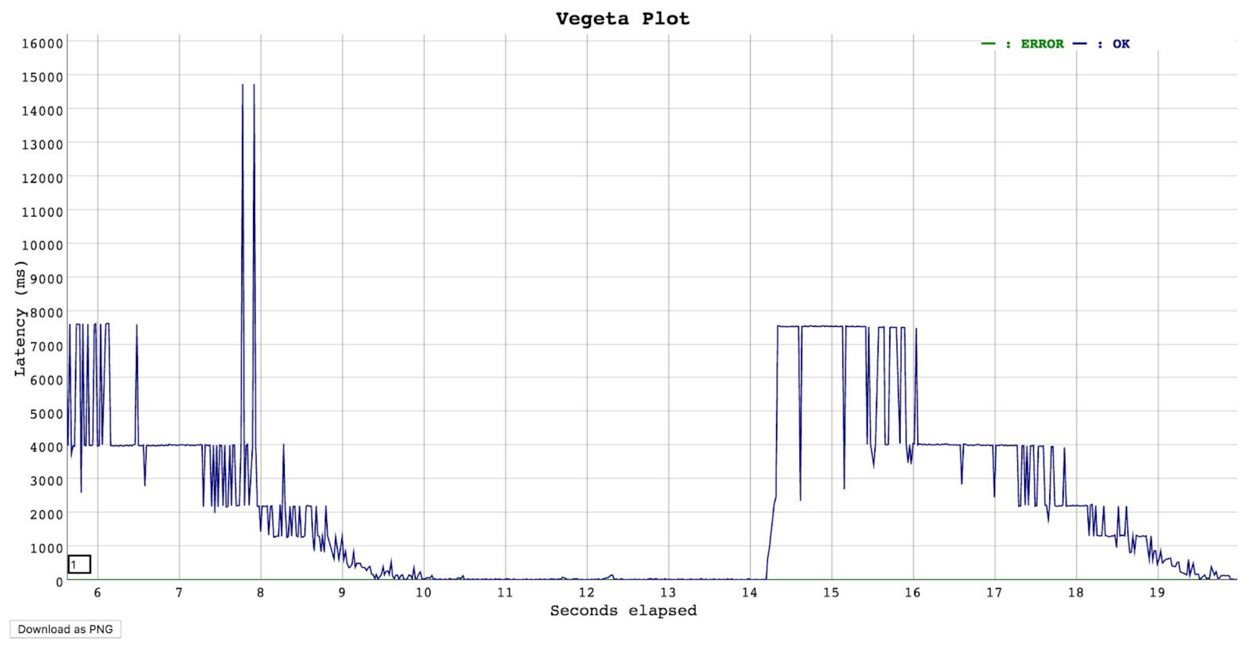
- Implementing request redirection on the load balancer
- Implementing random redirection and round robin load balancing algorithm and obtaining baseline performance stats on these two algorithms

See results section for more details.

Results

In this section, we show some of the graphs we get at this stage of the project.

(1) In this graph below we show the server load pattern we implemented. HTTP requests are sent to a single server continuously and uniformly for 20s at the rate of 50 requests per second, and we measure the response time for each request. This shows the periodical server load pattern we implemented on server side.



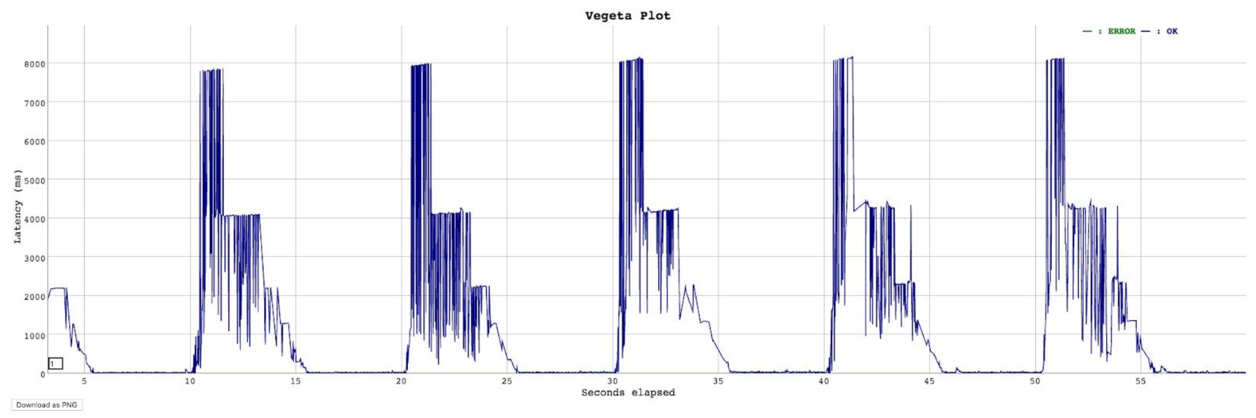
(2) In this graph below, we show the response time for *random load balancing*. We have two servers running and the load balancer randomly chooses which server to send the request to. HTTP requests are sent continuously and uniformly for 60s at the rate of 50 requests per second.

Here are some statistics:

Total number of requests sent: 3000

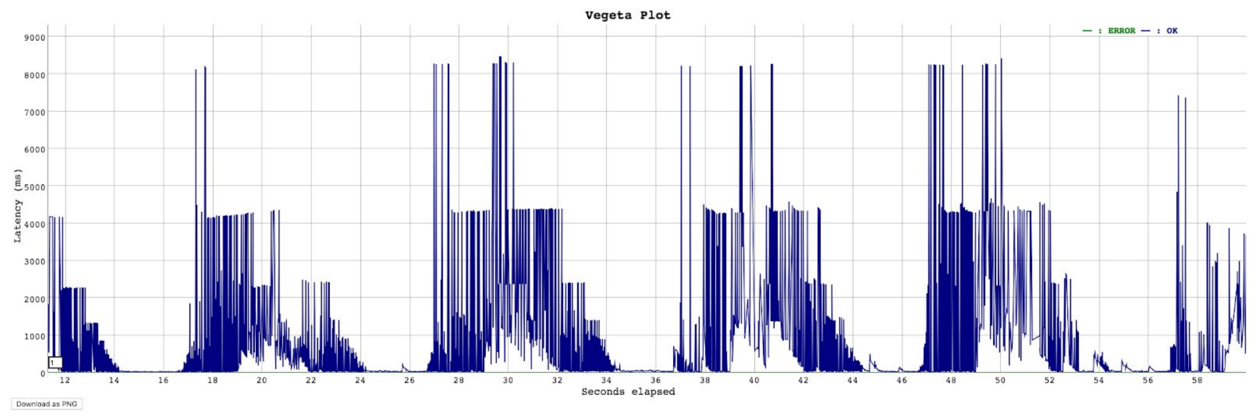
Latencies [mean, P50, P90, P99, max]: 1.21s, 33.83ms, 7.73s, 8.11s, 8.16s

Number of successful requests (ratio): 83.13%



(3) In this graph below, we show the response time for *round robin load balancing*. We have two servers running and the load balancer randomly chooses which server to send the request to. HTTP requests are sent continuously and uniformly for 60s at the rate of 50 requests per second.
Here are some statistics:

Total number of requests sent: 3000
Latencies [mean, P50, P90, P99, max]: 976ms, 62.59ms, 4.34s, 8.24s, 8.47s
Number of successful requests (ratio): 92.6%


Vegeta Plot
— : ERROR — : OK
Latency (ms)
Seconds elapsed
Download as PNG