Cloud and Edge Computing

Docker Project – Performance evaluation

Ari Vesalainen

ari.vesalainen@kolumbus.fi

010670459

Performance analysis

Figure 1 shows the setup for Stage 1 where Web load balancer is based on HTTP redirect. In the test client send a batch of HTTP requests to web balancer which in turns redirects the requests randomly to web workers. Test consists of several iterations: First iteration uses one thread to send all requests sequentially. Second iteration uses two threads in parallel and so on. The last iteration uses 20 threads to send requests out simultaneously. Web server balancer is used as a broker to select one of the worker servers for the actual calculation.

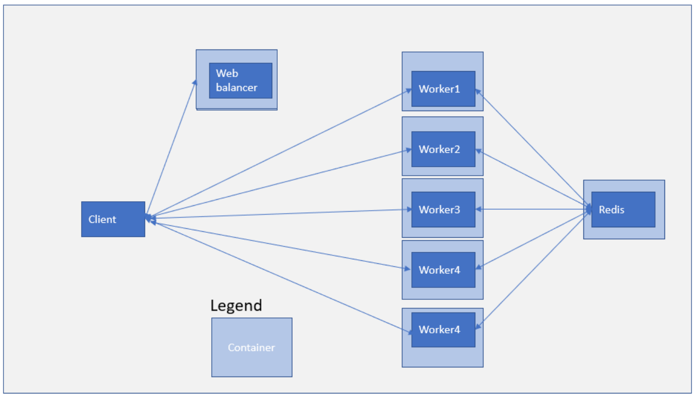

Figure 1. Setup for own web server load balancer

Figure 2 show the setup for Stage 2 which uses Docker Swarm service. In this case a port is published (port 5000) to enable service to be available outside the swarm. All workers participate in an ingress routing mesh. The routing mesh enables each worker node in the swarm to accept connections on published ports and provide replies to required tasks. The routing mesh routes all incoming requests on round-robin fashion to available workers.

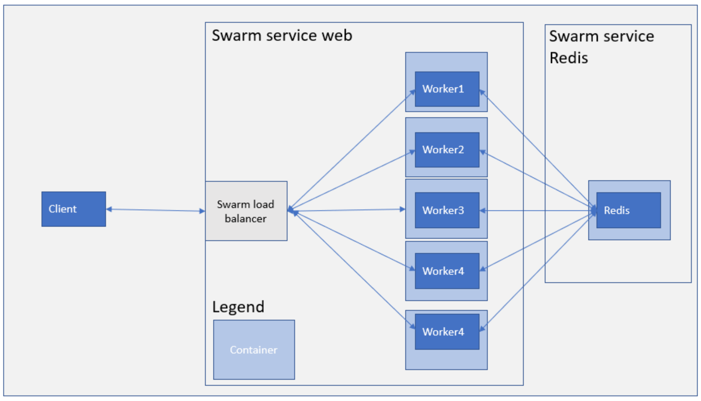

Figure 2. Setup using Docker Swarm service

The measurements are based on restricting CPU to 25% of maximum load. Using 5% CPU limit was not possible as the execution time for 1 worker increased significantly. The test was stopped after 10 hours of executions. Figure 3 shows the mean latency for own load balancer when we have 1, 3 or 5 worker nodes. Figure 4 shows the mean latency for Swarm balancer also when we have 1, 3 or 5 worker nodes.

The figure 3 shows how the latency increases when the number of parallel requests per thread is increasing. In case of one worker node it increases linearly from 27 ms to 660 ms. Latency for 3 workers is increasing from 14 ms to 312 ms and for 5 workers latency is increasing from 12 ms to 252 ms. As expected, the addition of new worker servers affects the latency positively when the number of parallel requests per thread is higher. The latency for 3 workers is 53% lower than the latency of 1 worker and latency for 5 workers is 61% lower than with 1 worker. The difference between 3 workers and 5 workers is also significant, for smaller number of simultaneous threads the difference is very small but it increases to 20% when number of simultaneous threads reaches 20 (3 workers: 312 ms vs. 5 workers: 252 ms).

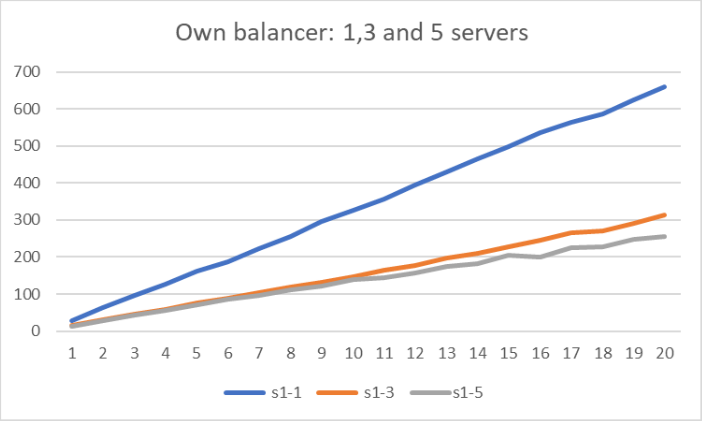

Figure 3. Own balancer: Mean latencies for 1, 3 and 5 servers

The figure 4 shows very similar latency trends as the previous test. In case of one worker node latency increases linearly from 34 ms to 674 ms. Latency for 3 workers is increasing from 14 ms to 221 ms and for 5 workers from 11 ms to 213 ms. As earlier, the addition of new worker servers affects the latency positively when the number of parallel requests per thread is higher. The latency for 3 workers is 67% lower than the latency of 1 worker and latency for 5 workers is 68% lower than with 1 worker. The difference between 3 workers and 5 workers is more than 10% for small number of simultaneous threads (n <= 4), but the difference reduces when n grows and is eventually only few percent (3 workers: 222 ms vs. 5 workers: 213 ms).

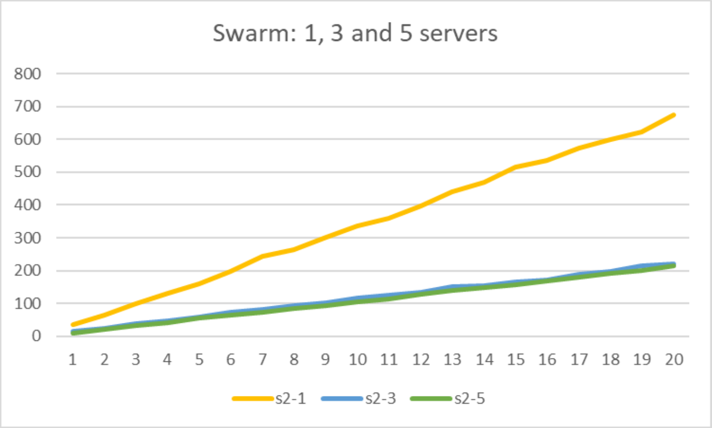

Figure 4. Swarm balancer: Mean latencies for 1, 3 and 5 servers

For the reference I have also included a comparison between these two different load balancing approaches. I have selected the case when number of workers is three. The Figure 5 shows how the Swarm balancer outperforms the simple load balancer. Already with 2 simultaneous threads the latency is 20% longer for own load balancer than for Swarm balancer and it increases to 40% when n reaches 20. Similar results are visible also for case of 5 workers. In case of 1 worker there is no real performance difference between approaches. This is understandable as there is only one request at the time and the load balancing doesn’t provide any advantage.

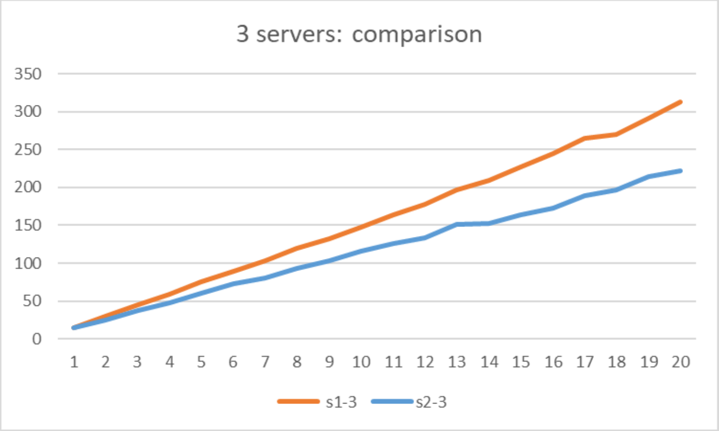

Figure 5. Comparison between load balancing approaches when number of workers is 3.

Analysis of results

Test measures the application latency and it consists of two components: network latency and server execution delay.

Server execution delay

When the number of simultaneous threads is one the execution is sequential, and the parallel execution is not helping. However, when the number of simultaneous threads is increasing, and tasks can be executed in parallel. This shortens the server execution delay as we can see when the number of workers is increased to 3 and 5. However, when number of simultaneous threads is increasing and the CPU is limited in worker nodes they are not able to complete tasks when new tasks are arriving and this causes queuing and increased delays in execution.

To analyze possible causes for delays some execution time measurements were gathered.

Resource usage when number of workers is three (own load balancer)

```
sudo docker stats

CONTAINER ID   NAME         CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O   PIDS

e31b46c646bf   web-server   6.94%     38.05MiB / 1.949GiB   1.91%     53.9MB / 67.4MB   4.02MB / 0B       3

c50d8d31e51c   worker3      4.75%     34.87MiB / 1.949GiB   1.75%     16MB / 15.8MB     0B / 0B           3

712e1c31a7d2   worker2      8.27%     34.91MiB / 1.949GiB   1.75%     16.1MB / 15.9MB   0B / 0B           3

2a3dbd82d6c6   worker1      14.28%    41.83MiB / 1.949GiB   2.10%     15.8MB / 15.6MB   15.9MB / 0B       3

d6678ed2a852   redisdb      0.65%     636MiB / 1.949GiB     31.87%    12MB / 6.15MB     130MB / 0B        5
```

The execution time data shows the resource constraints are most probably CPU and networking in worker and web-server nodes. The CPU utilization in redis server and load balancer is low and it doesn’t impact the latency. Also, memory utilization for all nodes is low and therefore it is not a critical resource and there is no impact on latency.

Networking latency

Latency is increasing when the tasks are competing on the same resources and the critical resource is not available the process needs to wait until it is released. For networking the requests are sent in parallel and they are competing on the same networking resources (e.g. access to same data structures, web server). This causes additional waiting time and delays are increasing when number of simultaneous threads is increasing.

The networking latency is most probably the reason for performance difference between different approaches. As the load balancer is based on HTTP redirect the client connects first the load balancer servers, i.e. it connects to HTTP server in the node. This redirects the connection to the worker node, i.e. there is two application level connections before the calculation is done. In case of Swarm balancer, the routing is done on a lower level and most probably this is more efficient way to resolve the task and therefore the Swarm performs better.