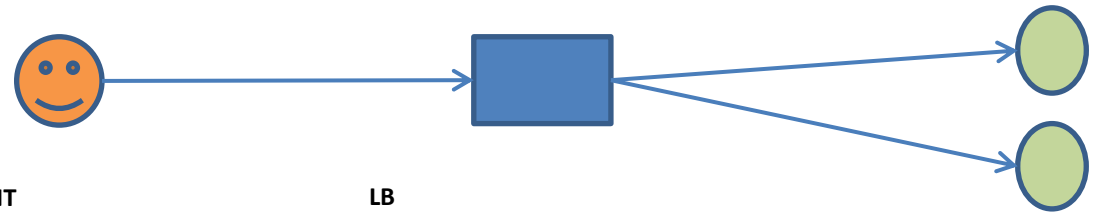
Benchmarking SW load-balancers

WHY

There are two major design decisions for the upcoming ELB implementation. The first is whether to run the load-balancers on the native OS or inside the VMs that Eucalyptus manages. The second is the choice of the underlying SW load-balancer. We're considering either HAProxy or Nginix, based on the feature set they cover. The goal of this benchmark is to compare the possible options from performance perspectives.

HOW



CLIENT

- Apache Benchmark tool (ab)
- Concurrency level: 50 threads
- # of REQ: 50,000 – 500,000
- Native OS (CENTOS 6.3)

LB

- HAProxy 1.5-dev 14(latest dev version w/ ssl support)
- Nginx 1.0.15 (default package in Centos 6.3)
- Round-robin distribution
- Host vs. Eucalyptus VM (m1.xlarge)
- Centos 6.3 for both host and vm
- HTTP (Https will be later)
- Keep-alive On and Off

Backend web servers

- 2 VM instances (m1.large)
- Apache
- No keep-alive
- HTTP
- 1KB, 10KB, 100KB files

The picture above illustrates the benchmark's setup. Across the tests, the client and backend web servers remain the same. This benchmark result covers the HTTP and HTTPS listeners and DOES NOT cover the arbitrary TCP listeners. They will be covered in the continuing benchmark.

RESULT - HTTP

Because everyone's time is precious, we will begin with the conclusion:

- **HAproxy outperforms Nginx significantly (especially for smaller messages: 67% better throughput for 1KB download)**
- **There is some performance loss due to VMs (23% loss for small objects with HAproxy)**

The following graphs illustrate the result in detail. During the test, the client downloads objects from web servers in varying size (1K, 10K, 100K files). The first set of graphs shows the number of requests a load-balancer process per second. The four bars represent the four cases: Haproxy on VMs, Nginx on

VMs, Haproxy on native host, Nginx on native host. The graphs represent the conclusions above. The Haproxy outperforms Nginx on both VMs and native host. The difference becomes less apparent when the object size is bigger, and almost negligible for 100KB objects.

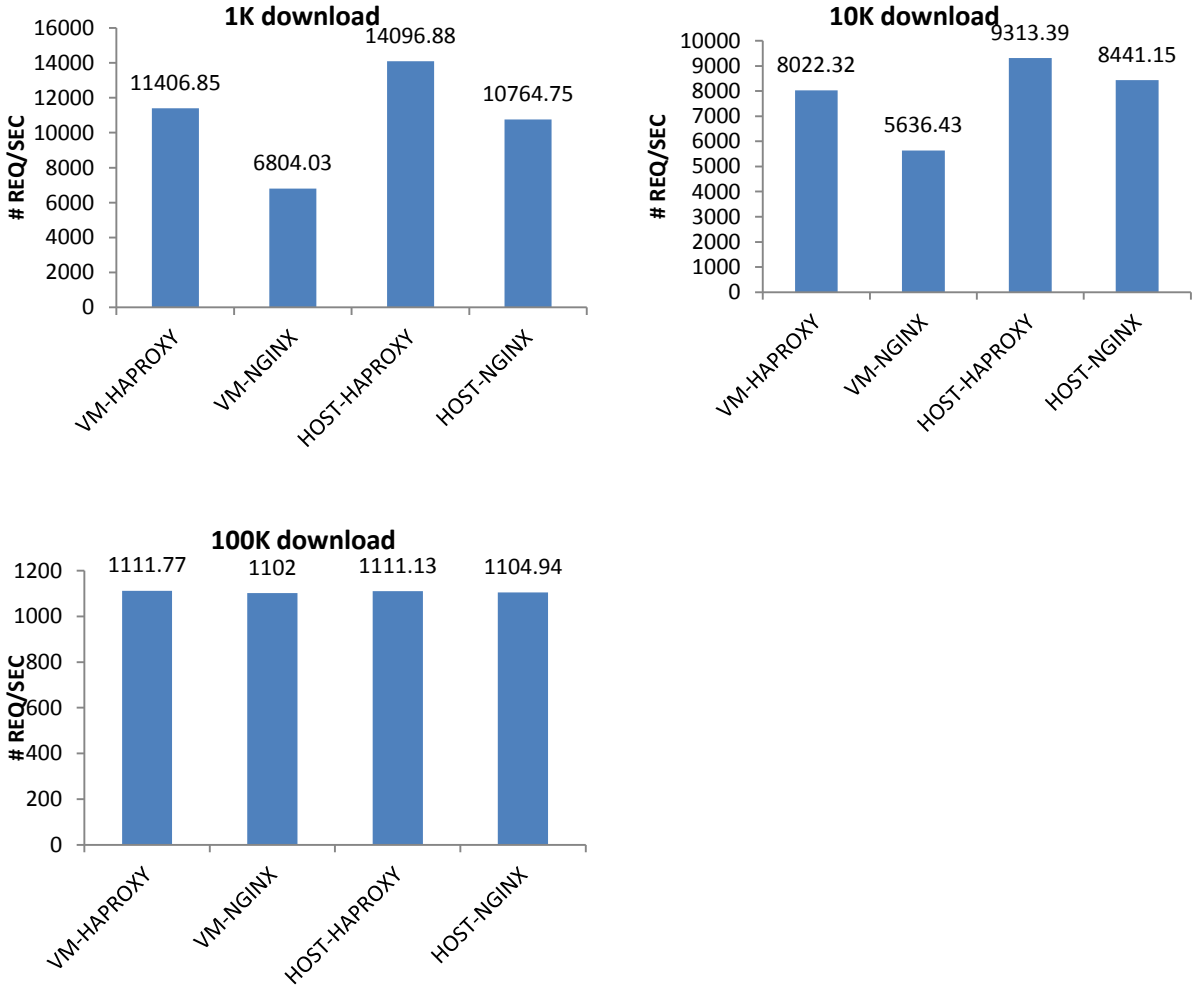


Figure 1: Haproxy and Nginx, on virtual machines and native OS

The next set of graphs show the distribution of response times for each request. The x-axis represents the percent of requests completed within the time in y-axis. For instance, when downloading 1KB object, 95 % of requests completed within 8 ms, for both Nginx and Haproxy. The graphs indicate that both the Haproxy and Nginx show stable response time (not too many spikes) across many requests. In general, Haproxy shows better distribution of the response time than Nginx. For example, 99% of the requests finish within 11ms for Haproxy, compared to 35ms of the Nginx during the 10K file download. For Nginx, the distribution becomes worse when object is bigger (i.e., small number of requests downloading large object will experience very large delay).

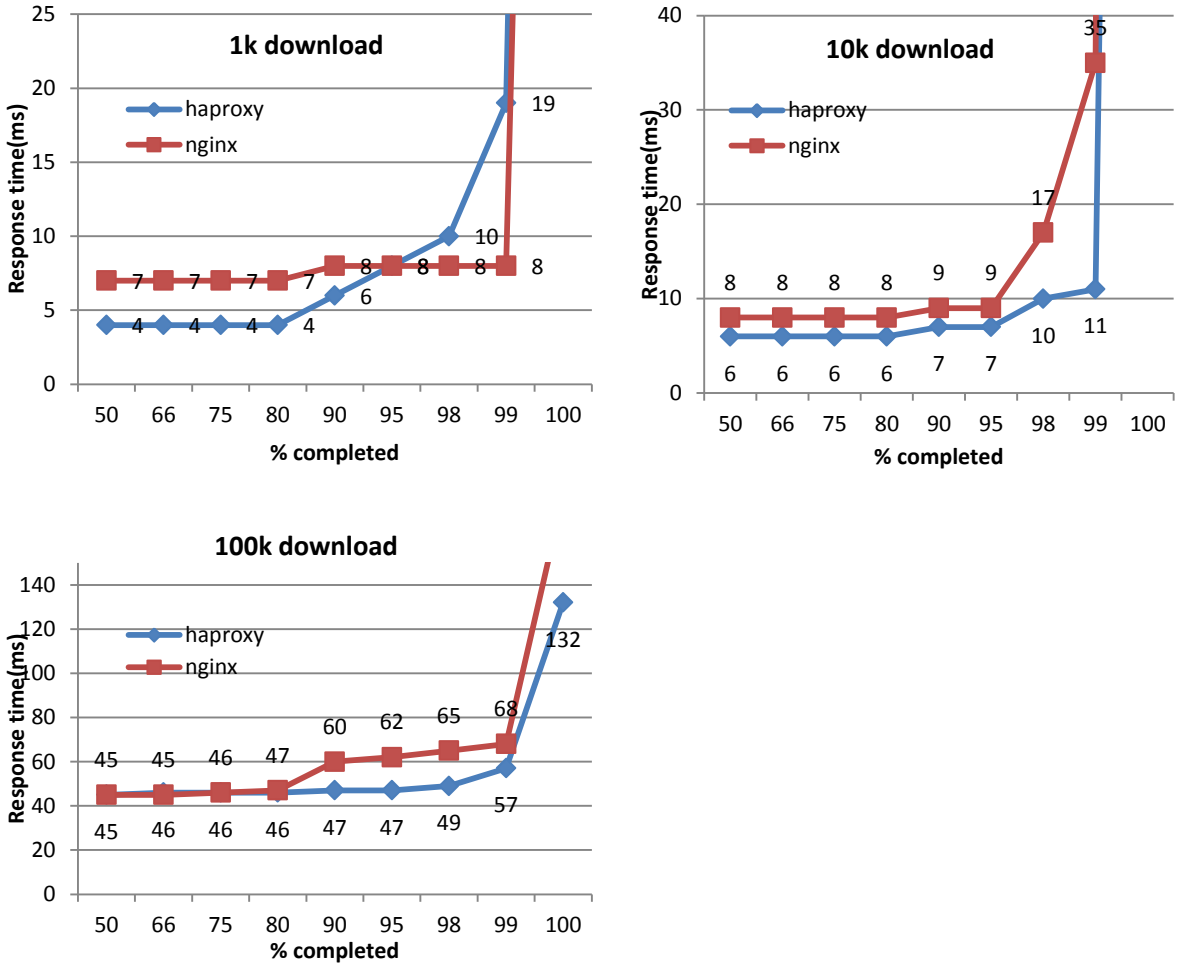


Figure 2: distribution of HTTP response time

Finally, the next three graphs show the result when keep-alive is turned on for both load-balancers. Note that in this setup, the client requests keep-alive to the load-balancer (keep-alive is optional for http clients), which will keep the socket open during the entire test run (50K-500K download). It is known that AWS has keep-alive turned on their ELB. For brevity, we ran the tests only on VMs, comparing the Haproxy and the Nginx. The result shows that keep-alive on the LB increases the throughput by about 15 %. The Haproxy performs better than Nginx, by roughly 20 %, when the object is small (1KB). Their difference becomes negligible for bigger objects (10K and 100K) and actually Nginx for the first time perform better for 10KB object than Haproxy.

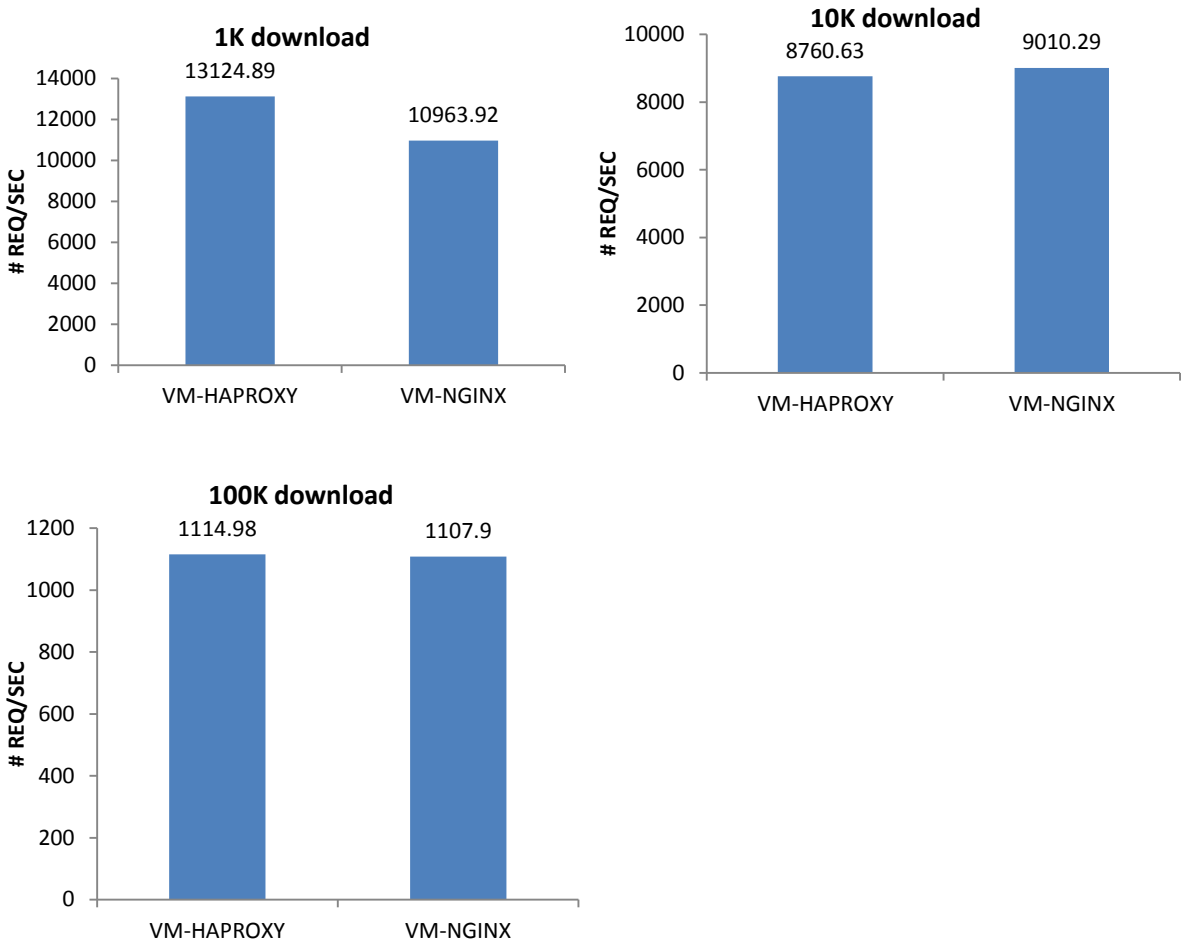
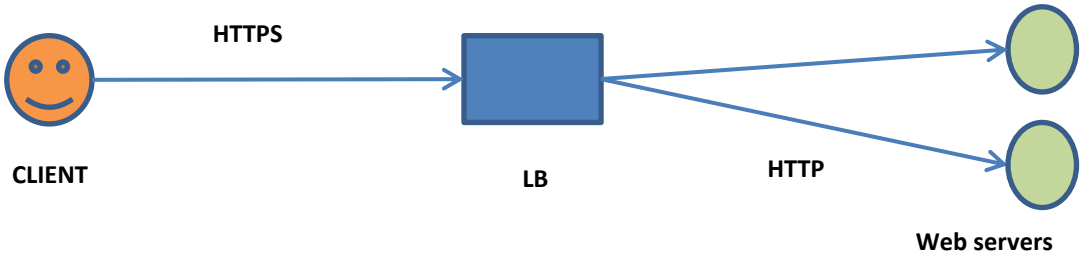


Figure 3: Haproxy and Nginx with Keep-alive

RESULT - HTTPS

Conclusion: Haproxy performs slightly better than Nginx.

The following benchmark tests SSL termination on load-balancers. In this test, while the client communicates with the load-balancers over SSL, the communication between load-balancers and the back-end web servers are over HTTP (so we terminate SSL at the load-balancer). The parameters remain the same as the previous HTTP benchmark. We used AES256-SHA as the cipher algorithm. Both HAProxy and Nginx are run inside the VMs in this test (we do not test HTTPS on native host).



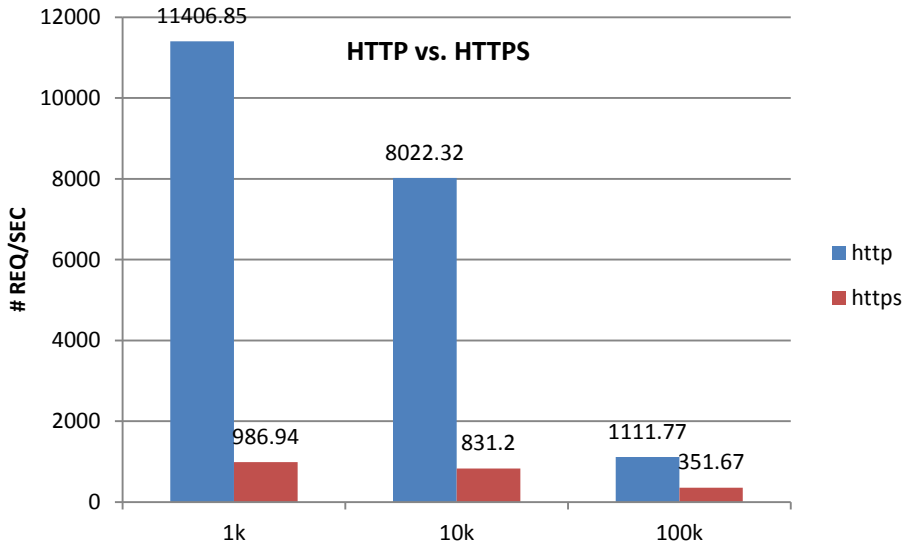
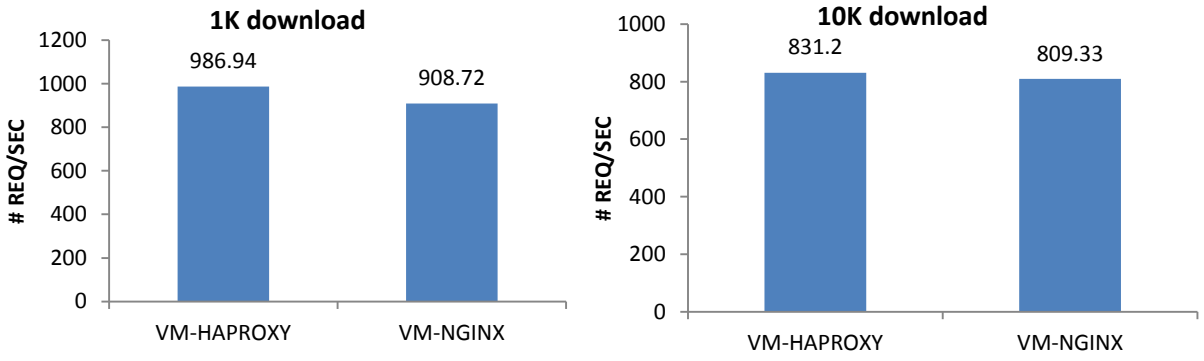


Figure 4:HTTP vs. HTTPS (HAproxy)

The figure 4 illustrates the difference in the throughput of HTTP and HTTPS. In the figure we compare the number of requests processed per second for HAProxy (similar results were obtained for Nginx). We can see roughly 90% less throughput with HTTPS for small objects.



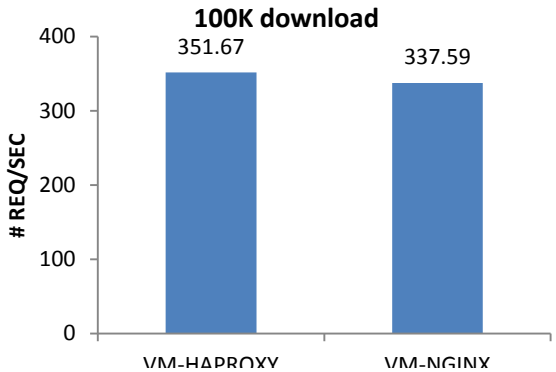


Figure 5: HTTPS throughput: HAProxy vs. Nginx

The figure 5 compares the HTTPS throughputs between HAProxy and Nginx. In all three object sizes, we can see HAProxy slightly outperforms Nginx. The biggest difference is when the object size is smallest (1KB): HAProxy achieves 8.6% better throughput than Nginx.

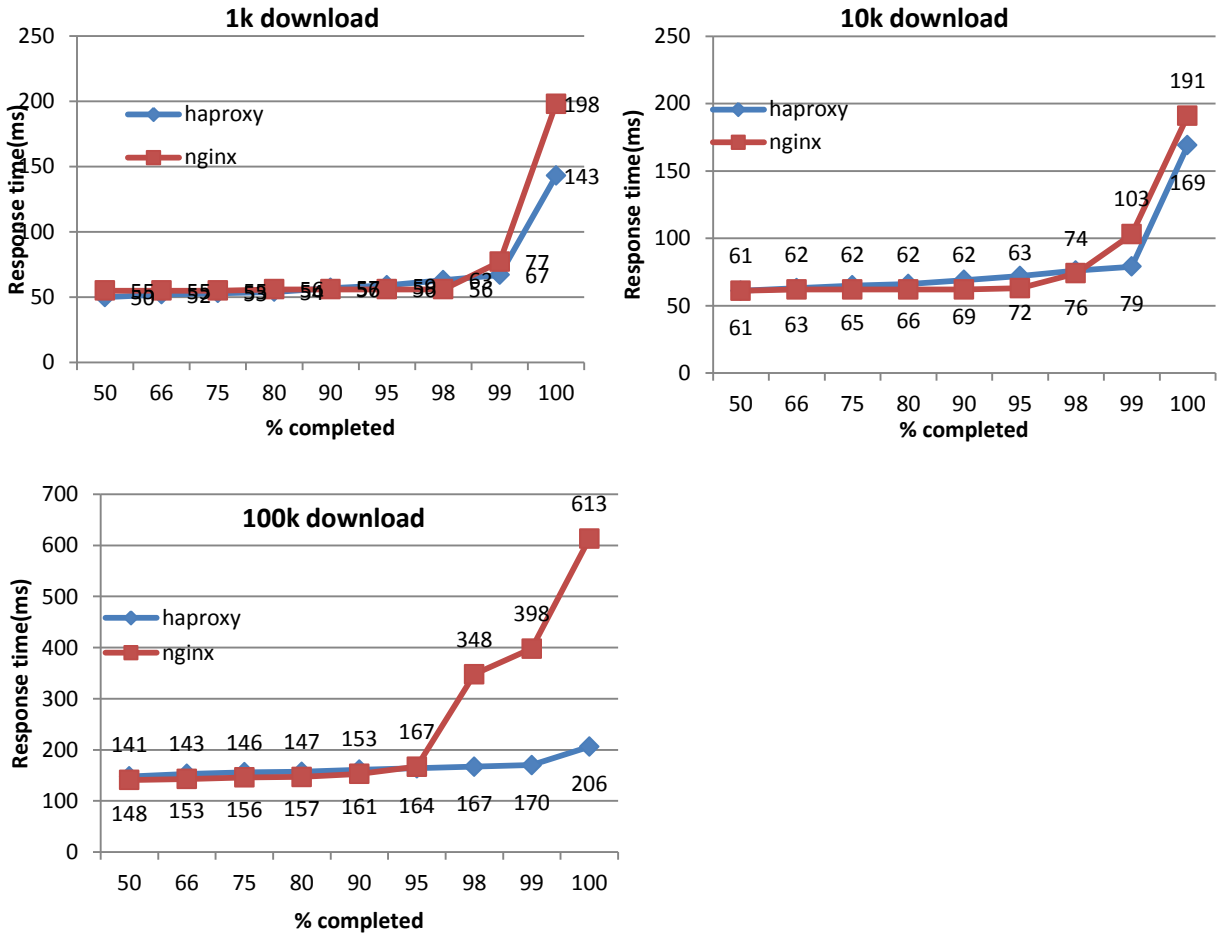
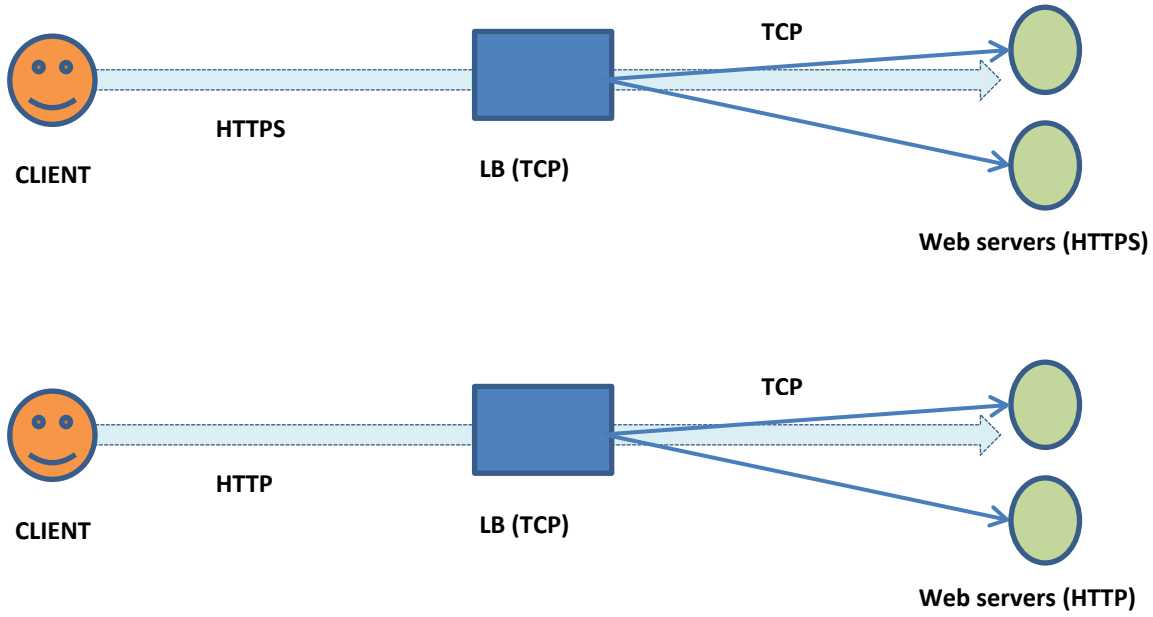


Figure 6: HTTPS response time distribution: Haproxy vs. Nginx

Figure 6 presents the distribution of the HTTPS response time, comparing Haproxy and Nginx. The x-axis represents the percent of requests completed within the time in y-axis. In all three graphs, we can see that both Nginx and Haproxy exhibit fairly stable distribution. 95% of requests complete within the reasonable time for all object sizes. However, we can see one issue with Nginx. For the request that's in the highest 5% of response time, the Nginx shows significantly worse response time than Haproxy. The difference is most apparent when the object size is bigger (100KB). This means that with Nginx there will be small number of requests that show much higher spikes than Haproxy.

RESULT – TCP

To benchmark the TCP load-balancing, we've come up with two test configurations.



In both configurations, the Haproxy and Nginx are setup as TCP load-balancers, communicating to the backend web servers using TCP. The client takes the data as HTTP or HTTPS depending on the protocol on the backend web servers (so the load-balancers are dumb TCP tunnel).

Figure 7 represent the results. In the figure, we compare HAproxy and Nginx, both running on VMs, with the throughput from direct connection to one of the web server (no LB involved). Note that there are two web servers behind the LB, so the comparison with the direct connection would give us insight on how well the LB would perform as SSL pass-through balancers to backend HTTPS servers. There are two findings in the graph:

- Haproxy and Nginx are virtually tie. This an expected result, as only with two web servers running HTTPS, we can't saturate the throughput capacity of both LBs.

- For smaller objects, we can expect that SSL pass-through balancing would be horizontally scalable. However, for bigger objects (when workload is more data-bound) it appears that simply adding more HTTPS backend would not result in better throughput.

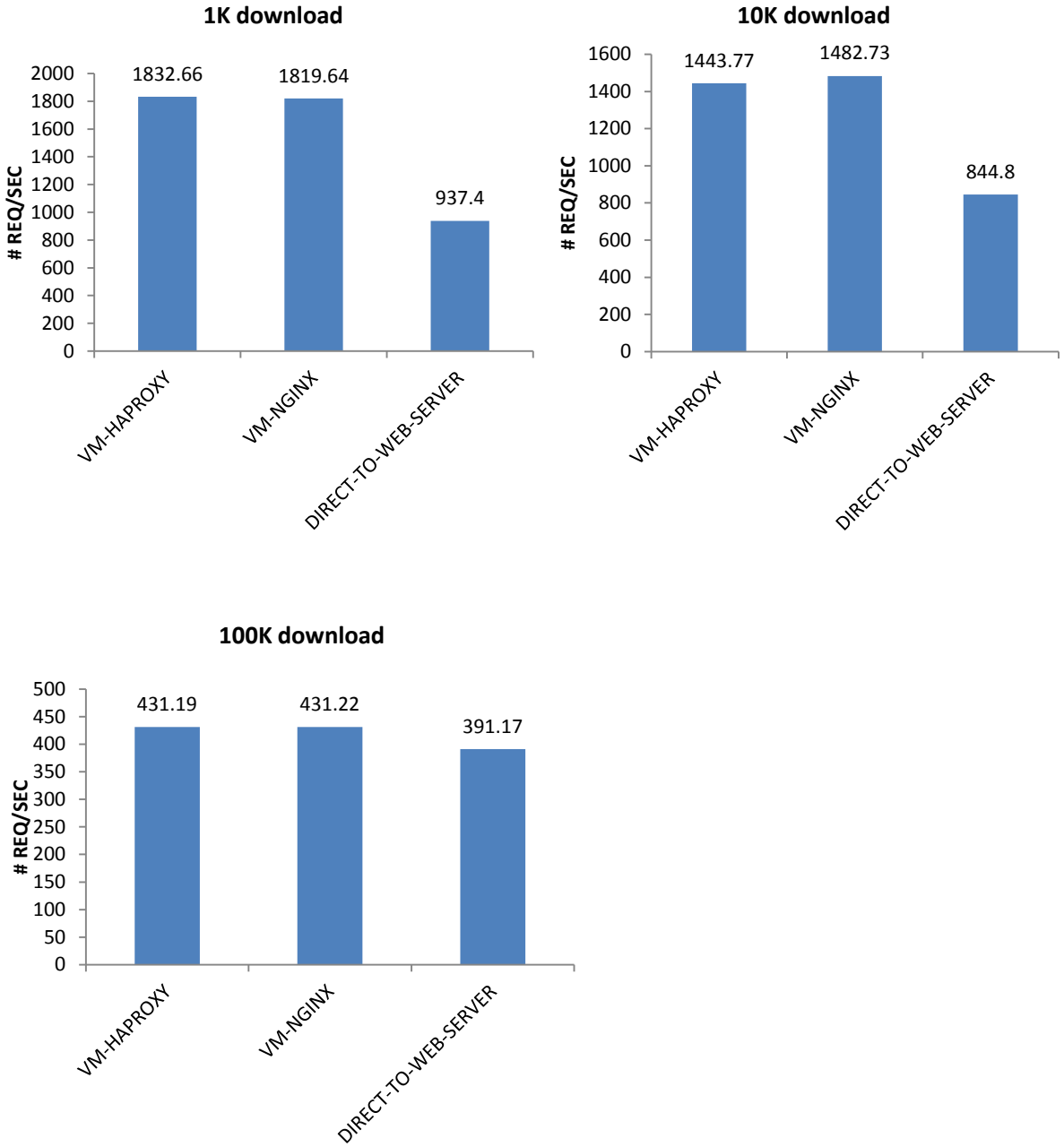


Figure 7: HTTPS through TCP load-balancer

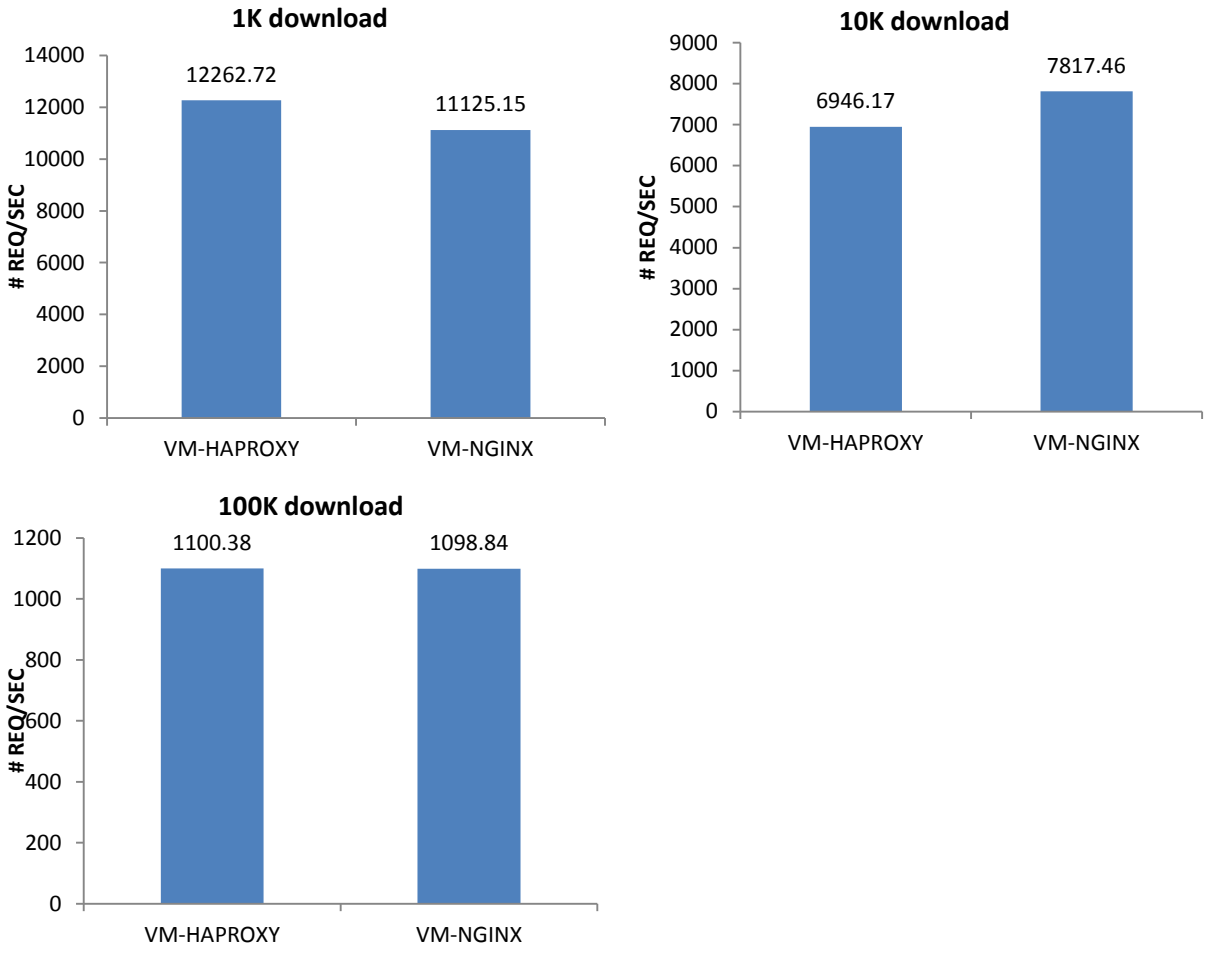


Figure 8: Figure 7: HTTP through TCP load-balancer

Finally figure 8 illustrate the results when load-balancers are for pass-through HTTP traffic. In this setup, we can assume that load-balancers are saturated and thus we can compare the maximum attainable TCP throughput for Haproxy and Nginx. In the figure we can see Haproxy and Nginx are roughly tie in the TCP load-balancing. For 1K objects, Haproxy slightly outperforms Nginx, while 10K result is the opposite.

SUMMARY

We summarize the result of the benchmark as follows:

- There are some performance losses due to LBs running inside the VMs (as much as 23%). We can anticipate that tuning the VM parameters and the hypervisors could lessen the loss.
- For HTTP and HTTPS, HAproxy apparently outperforms Nginx (by as much as 67%).
- For TCP, HAproxy and Nginx show about the same performance.
- Through the benchmark, we found that both Haproxy and Nginx can cover the feature set mandated by ELB specification (HTTP, HTTP-KA, HTTPS, TCP, Reconfigurability, etc).