Resilience Enhancement of Container-based Cloud Load Balancing Service

Dongsheng Zhang[1]

[1]**Compsure.ai**

Corresponding author:
Dongsheng Zhang[1]

Email address: dzhang@composure.ai

ABSTRACT

Web traffic is highly jittery and unpredictable. Load balancer plays a significant role in mitigating the uncertainty in web environments. With the growing adoption of cloud computing infrastructure, software load balancer becomes more common in recent years. Current load balancer services distribute the network requests based on the number of network connections to the backend servers. However, the load balancing algorithm fails to work when other resources such as CPU or memory in a backend server saturates. We experimented and discussed the resilience evaluation and enhancement of container-based software load balancer services in cloud computing environments. We proposed a pluggable framework that can dynamically adjust the weight assigned to each backend server based on real-time monitoring metrics.

I. INTRODUCTION

With the exponential growth of internet services over the last decades, the web request rate for popular websites could increase in a massive scale within seconds. The internet services that fail to handle the high volume of web traffic within short period of time would suffer significant network congestion and delay thus deliver extremely poor user experience on the client side. For large-scale internet services, the solution to this problem in the new cloud world is horizontal scaling. One critical component of horizontal scaling is load balancer that could distribute incoming application traffic among a set of backend servers. Load balancer is becoming even more important in the service-oriented architecture that has become the main backend web service architecture in recent years [1].

There are two major categories of load balancers: hardware-based and software-based. Compared to hardware-based load balancers that require dedicated machines, software-based load balancers are more programmable, scalable, and resilient. The most popular software-based load balancer includes HAProxy [2] and NGINX [3]. We chose container-based [4] NGINX as the load balancer in our experiments.

They are several load balancer algorithms that would decide how to distribute the traffic. Round-robin algorithm distributes the traffic sequentially among the servers. Least-connection algorithm always sends the next request to the sever with least number of concurrent network connections. Both round-robin and least-connection algorithms have a weighted version that takes into account the capacity of the backend servers. However, weight assignment can only be determined in a static manner based on initial server capacities. NGINX plus allows the user to configure dynamic weight based on certain metrics. Still, the metrics that the user can choose are very limited. In the paper, we propose a flexible, pluggable, cloud-agnostic, and metric-agnostic dynamic algorithm for any cloud load balancer services. We model the resources in a server as a multidimensional vector, based on which we convert the relative resource availability of all backend servers to the weights assigned to them dynamically.

The rest of paper is arranged as follows. In Section II, we introduce the background of resilience and load balancing services. In Section III, we propose the model approach and algorithm to improve existing load balancer and introduce our experiment to evaluate load balancer performance. In Section IV, we present and discuss our experiment results. Finally, we summarize our work and discuss future steps in Section V.

II. BACKGROUND AND RELATED WORK

An architectural framework for network resilience and survivability has been presented in [5]. Network resilience has been extensively studied [6, 7, 8, 9, 10, 11, 12, 13, 14]. Various load balancing schemes have been compared in [15, 16]. Even though dynamic load balancing algorithm is difficult to implement, it provides better performance in heterogeneous cloud computing environment [15]. In this work, we will measure network resilience of dynamic load balancing services using container-based load balancers [17, 18]. Dynamic load balancing mechanism has been presented and extensively studied [1]. What distinguishes our work from previous work is that we provide a practical framework that can be easily integrated with any existing cloud load balancing infrastructure. In this work, we conducted our experiment using NGINX docker container running in AWS (Amazon Web Services) instance.

III. MODELING APPROACH

Figure 1 below presents the overall architecture of our dynamic and pluggable multidimensional load balancing service framework.

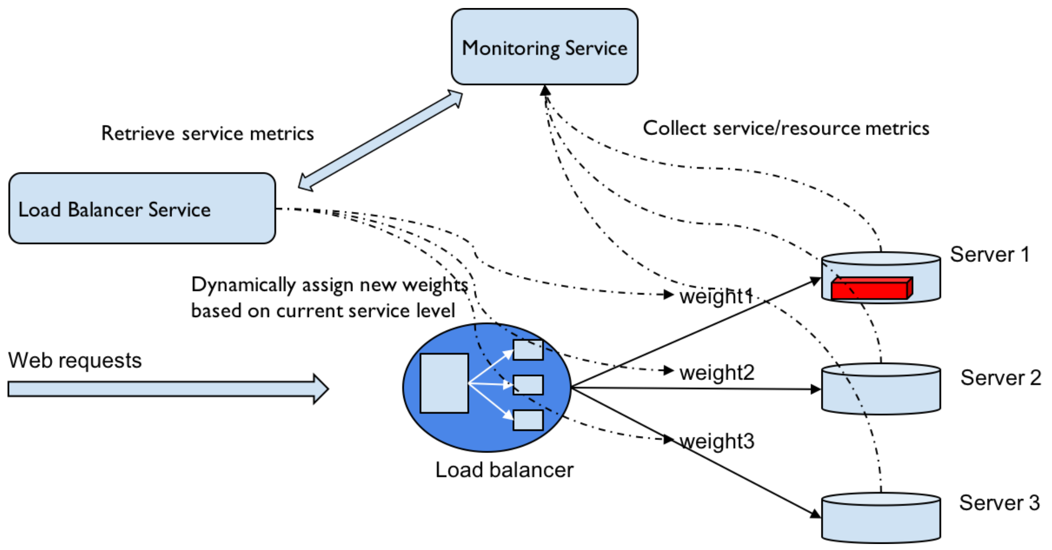

Figure 1. System architecture

We have an agent-based monitoring service that can collect various types of metric data of the backend servers. Since the user has the full control of their applications, they can decide what metrics they want to send to our monitoring service. Our monitoring service stores all the metric data in a time-series database. Our load balancing service acts as a plugin for NGINX load balancer and dynamically modified the weights assigned to each backend server based on the real-time metrics reading from the monitoring service. We measure the RTT (round trip time) of web requests via NGINX load balancer. For each backend server, we compare the RTT of the experiments with and without our dynamic multidimensional load balancer algorithm.

IV. EXPERIMENT RESULTS

We run container applications in a Ubuntu instance in AWS. Our load balancer policy can adjust the weight for each backend server dynamically based on service level metrics. The default NGINX load balancer with equally assigned weight to each back server cannot take into account the currently used resources such as CPU and memory. The metric used here is cpu_load, which can be obtained via /proc/loadavg in a Linux machine. This metric provides more accurate estimation of the CPU resource availability than cpu_used, as cpu_used cannot capture the number of queued jobs in the operation

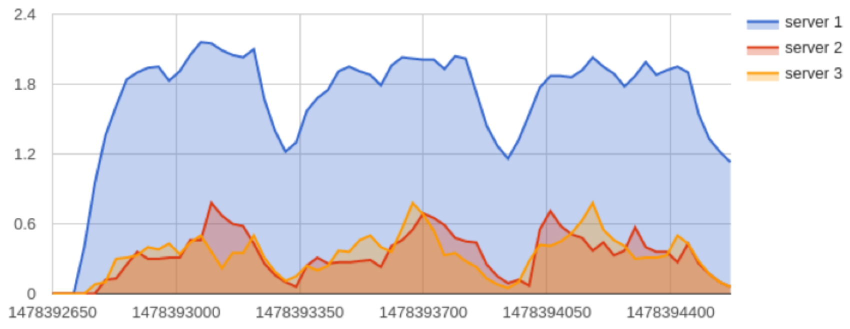

Figure 2. RTT from each backend server without applying our policy

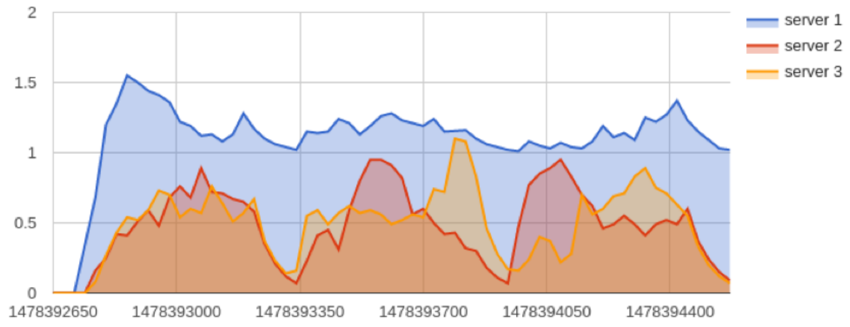

Figure 3. RTT from each backend server with our policy

system. In this experiment, we compare the average RTT for each request. In each backend server, we run a task that sorts a fixed number of random integers to simulate the CPU consumed to handle each coming web request. We put extra CPU load in server 1. The default NGINX policy still distributes requests to backend servers equally. Our load balancer policy will dynamically change the weight of distributing requests to the backend servers.

Figure 2 and Figure 3 display the time-series cpu_load of 3 backend servers with and without our load balancer policy. With the adjustment of backend server weights, the variances of cpu_load among 3 servers are much smaller than without adjustment.

From the performance point of view, we look at the average RTT from all backend servers. With cpu_load based load balancer policy, the average RTT is 0.0072 seconds as shown in Table 1; in contract, without any policy, the average RTT is 0.0658 seconds, which is almost an order of magnitude slower. The maximum RTT for both scenarios comes from server 1. The maximum RTT without our policy is more than 0.6 seconds as shown in Table 2, while with our policy the maximum RTT is only 0.0184 seconds. Our dynamic load balancing algorithm improves the RTT up to 9 times compared to default NGINX load balancer.

	Average	Min	Max
all	0.0072	0.0065	0.0184
server 1	0.0100	0.0069	0.0184
server 2	0.0070	0.0068	0.0085
server 3	0.0069	0.0065	0.0086

Table 1. RTT with dynamic load balancing policy

	Average	Min	Max
all	0.0658	0.0066	0.6019
server 1	0.1848	0.0068	0.6019
server 2	0.0070	0.0067	0.0087
server 3	0.0069	0.0066	0.0087

Table 2. RTT without dynamic load balancing policy

V. DISCUSSIONS

The computing power of the same AWS instance varies tremendously, which makes it difficult to conduct this experiment fairly. The factors that could affect the services of AWS instance observed so far include: a. the time of the day; b. the duration that the instance has been running for; c. the CPU usage of the application. The unpredictability of AWS instance makes it extremely difficult to control the variables in this experiment. In addition, docker stats provides very bizarre data after running CPU-intensive jobs for certain period of time. What can be observed repeatedly is that in a docker container running in a single-core AWS instance, docker stats reports up to 500% CPU usage which is obviously wrong. cpu_load could be a bad predictor, because for the same amount of cpu_load, there could be a significant difference between loads contributed by a couple of CPU-intensive jobs and a large number of small jobs. Other factors that could also affect this is the priority of the processes and the scheduling algorithm of operation system. For transactional scenario, the best metric in the feedback loop to capture resource usage in a server probably is RTT. The tables above show that cpu_load based policy provides a significant gain of performance for this particular scenario. However, as mentioned above, the experiment was done in a strictly-controlled environment. The performance gain might not be at the same level in general cases. A better way to calculate new weights for load balancer backend servers could be a machine learning based approach using historical data of application resource usage.

VI. CONCLUSIONS AND FUTURE WORK

In the work, we proposed a multidimensional pluggable dynamic cloud load balancing framework. The algorithm can be easily integrated with various cloud computing platforms such as Docker, OpenStack, Kubernetes, etc. In the future, we can plug service-level metrics into our algorithms. Machine learning based predictive weight adjustment could also significantly improve the system performance and user experience.

REFERENCES

[1] Valeria Cardellini, Michele Colajanni, and Philip S Yu. Dynamic load balancing on web-server systems. *IEEE Internet computing*, 3(3):28–39, 1999.

[2] HAProxy. http://www.haproxy.org/.

[3] NGINX. https://www.nginx.com/.

[4] Kyoung-Taek Seo, Hyun-Seo Hwang, Il-Young Moon, Oh-Young Kwon, and Byeong-Jun Kim. Performance comparison analysis of linux container and virtual machine for building cloud. *Advanced Science and Technology Letters*, 66(105-111):2, 2014.

[5] James P. G. Sterbenz, David Hutchison, Egemen K. Çetinkaya, Abdul Jabbar, Justin P. Rohrer, Marcus Schöller, and Paul Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245–1265, 2010.

[6] Dongsheng Zhang. *Resilience Evaluation and Enhancement in Mobile Ad Hoc Networks*. PhD thesis, University of Kansas, 2015.

[7] Dongsheng Zhang and James P.G. Sterbenz. Modelling critical node attacks in MANETs. In *Self-Organizing Systems*, volume 8221 of *Lecture Notes in Computer Science*, pages 127–138. Springer Berlin Heidelberg, 2014.

[8] Dongsheng Zhang and James P. G. Sterbenz. Analysis of Critical Node Attacks in Mobile Ad Hoc Networks. In *Proceedings of the 6th IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 171–178, Barcelona, Spain, November 2014.

[9] Dongsheng Zhang, Santosh Ajith Gogi, Dan S. Broyles, Egemen K. Çetinkaya, and James P.G. Sterbenz. Modelling Wireless Challenges. In *Proceedings of the 18th ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 423–425, Istanbul, August 2012. Extended Abstract.

[10] Dongsheng Zhang, Santosh Ajith Gogi, Dan S. Broyles, Egemen K. Çetinkaya, and James P.G. Sterbenz. Modelling Attacks and Challenges to Wireless Networks. In *Proceedings of the 4th IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 806–812, St. Petersburg, October 2012.

[11] Dongsheng Zhang and James P. G. Sterbenz. Measuring the Resilience of Mobile Ad Hoc Networks with Human Walk Patterns. In *Proceedings of the 7th IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, Munich, Germany, October 2015.

[12] Dongsheng Zhang and James PG Sterbenz. Robustness Analysis and Enhancement of MANETs using Human Mobility Traces. *Journal of network and systems management*, 24(3):653–680, 2016.

[13] Dongsheng Zhang and James P. G. Sterbenz. Robustness analysis of mobile ad hoc networks using human mobility traces. In *Proceedings of the 11th International Conference on Design of Reliable Communication Networks (DRCN)*, Kansas City, USA, March 2015.

[14] Dongsheng Zhang and James P. G. Sterbenz. Measuring the resilience of mobile ad hoc networks with human walk patterns. In *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 161–168, Oct 2015.

[15] Mayanka Katyal and Atul Mishra. A comparative study of load balancing algorithms in cloud computing environment. *arXiv preprint arXiv:1403.6918*, 2014.

[16] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi, and Jameela Al-Jaroodi. A survey of load balancing in cloud computing: Challenges and algorithms. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 137–142. IEEE, 2012.

[17] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[18] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.