Load Balancing for Microservice Service Meshes

Ratnadeep Bhattacharya
George Washington University
Email: ratnadeepb@gwu.edu
Advisor: Dr. Timothy Wood

***Abstract*—Microservices break down monolithic applications into smaller components and have become a popular model for application deployment. In this model, load balancing has moved from the server to the request side where a load balancer resides with each upstream service.**

Least Connection, a derivative of the Join the Shortest Queue (JSQ), is a popular algorithm used in the microservice architecture. Despite JSQ being proven to be nearly optimal in certain scenarios, Least Connection significantly underperforms in microservices and edge environments.

My thesis aims to adapt load balancing to the microservices environment such that they can autonomously choose from a collection of approaches to mitigate widening of response time distributions.

1. Introduction

Microservices have increasingly become the accepted solution for deploying modern applications. This has led to the rise of the "service mesh", like Istio [1], to connect the different microservices. The service mesh typically has a "control plane", managing configuration and metadata for the cluster, and a "data plane", managing the movement of requests through the cluster. In the service mesh, requests from the outside world land on an ingress gateway acting as a gatekeeper to the entire cluster. In the service mesh the load balancer has moved to a "sidecar" container deployed with the upstream service, through reverse proxies like Envoy [2].

Least Connection is a popular algorithm used in service meshes both in the ingress gateway and the sidecars. The Least Connection algorithm is a derivative of the Join the Shortest Queue (JSQ) algorithm. In the JSQ algorithm, the load balancer sends the next incoming request to the backend with the lowest queue size. JSQ keeps track of queue size at each backend by keeping a count of the number of outstanding requests. JSQ has been mathematically proven to be nearly optimal [3] under certain conditions. Least Connection adapts the JSQ algorithms for microservices which is a popular distributed systems pattern to deploy applications.

Least Connection keeps a count of the number of requests to each backend in the downstream service. This count is a reflection of the amount of load a particular client side server has sent to the individual backend and is not representative of the total load on each of the backends. Thus the Least Connection lacks a global perspective. When routing requests to a particular service, Least Connection uses power of two random choices algorithm (P2C) [4], which proves that selecting the lesser of two randomly chosen queues is almost as performant as choosing the lowest queue size. The P2C algorithm also ensures that multiple clients do not choose the same backend leading to a herding problem.

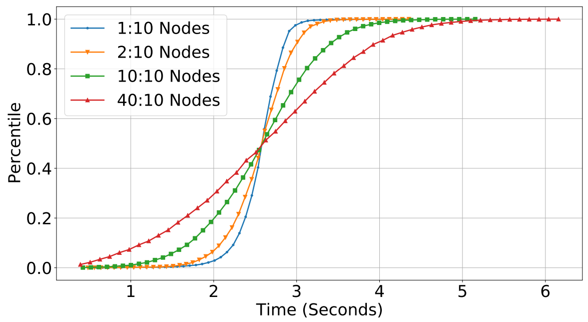

Figure 1. Changing from 1 to 40 frontends causes a significant increase in the range of response times and tail latencies.

My work shows that the performance of Least Connection can deteriorate severely unless carefully managed, Figure 1. The ingress gateway on an edge cluster has a global perspective but also a strong assumption of homogeneity in the backend cluster. This assumption has a strong possibility of being untrue in edge clusters [5], [6]. In the sidecar clusters too, as the service layers scale independently of each other, the response time distribution can broaden significantly while the mean remains the same. In both cases, I have shown that by incorporating feedback from the backends and estimating service capacities, Least Connection can be significantly improved narrowing the gap between the performances of JSQ and Least Connection.

2. Proposal

In my thesis, I am working to incorporate feedback from backend servers into load balancing decisions. This problem involves determining the correct feedback, like service capacity and queue length, and appropriate indicators to assist in the selection of the best possible backend server.

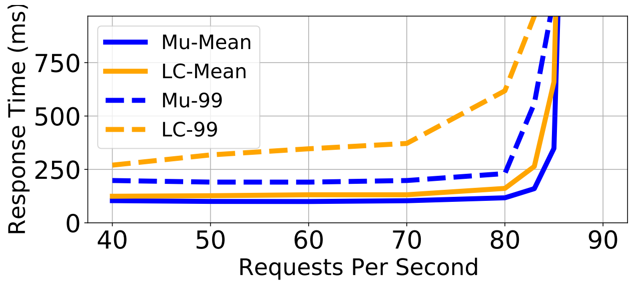

Figure 2. Mu's load balancer vs. Least Connection load balancer

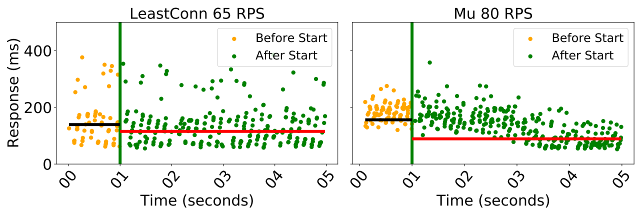

Figure 3. Mu takes advantage of a newly added pod more quickly: shifting load, improving both mean (horizontal lines) and variance in response time more

2.1. Results

In edge environments, pods can have different service capacities due mismatched hardware and/or interference due to dense packing. In such environments, my research has focused on Least Connection's assumption of homogeneity and its ability to rebalance load when new pods are added.

In my first paper, about the Mu system [7], we saw that even when Least Connection has a global perspective there is significant widening of the response time distribution, especially when scaling up the cluster. Least Connection protects against overloading new pods using P2C, so there is only a 2/N chance of the new pod being picked, leading to underutilization of resources in a resource constrained environment.

In Mu, we developed an algorithm that uses piggybacked values for capacity and queue length to pick a pod most likely to provide the fastest response time. Thus the algorithm automatically adjusts to different service capacities in the backend by sending more requests to the faster pods.

In Figure 2, we see that at 80RPS (Requests Per Second), the 99%ile, "Mu-99" and "LC-99", decreases from 618ms to 230ms since our algorithm sends more requests to the faster pods compared to Least Connection. In Figure 3, we see the result of pod addition on response time. Within three seconds of adding a new pod, our algorithm causes a marked downward shift in response times. Also, Mu exhibits significantly lower variability than Least Connection by accounting for the difference service capacities.

In my next paper, [8], we built a feedback system, BLOC, through which downstream pods could influence load balancing at the upstream service through 1) Active Queue Management (AQM) on the downstream pods and

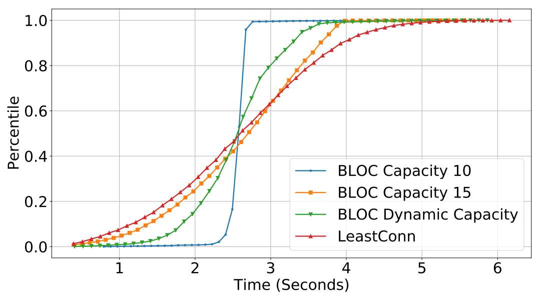

Figure 4. Sensitivity of BLOC to Capacity

2) an estimate, "chip", of the remaining capacity on the downstream pods. The backend server only accepts requests if its capacity, number of requests that can be handled concurrently, is not exceeded. A rejected request is then requeued at a different backend whose selection is guided by the chips.

The capacity can be manually configured or a running average of concurrent requests handled can be used. In Figure 4, we see that both approaches perform better than Least Connection which has no feedback from the backend.

2.2. Future Work

My work indicates that defining current capacity and estimating it accurately for backend servers can significantly improve load balancing in distributed systems. Defining capacity as number of requests completed per unit time and number of concurrent requests handled while maintaining an SLO (Service Level Objective) are both useful approaches. However, both approaches are plagued by the problem of identifying and accounting for idle server time in the backends and remains the prime detriment in capacity estimation. Another problem is that service cost may vary between requests impacting any capacity definition. A simple way to work around this is to consider the "average" service cost.

The second major issue is to develop a robust estimate of the remaining capacity of a backend server when routing requests. While the "chip" indicator has had some success, it remains a problem that needs to be investigated further.

A third issue is that overload control techniques, while successful at providing feedback for requests with high service costs, become a significant overhead when the backend service cost is low.

Finally, there is also a need for load balancers in distributed systems that can handle job priorities, variable job sizes and so on.

My goal is to build load balancers that can distinguish between different operating conditions and autonomously deploy a bag of techniques that provide optimal results under each of those conditions.

References

[1] Istio. [Online]. Available: https://istio.io/latest/

[2] Envoy proxy - home. [Online]. Available: https://www.envoyproxy.io/

[3] V. Gupta, M. Harchol Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farms," vol. 64, no. 9, pp. 1062–1081. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0166531607000624

[4] M. Mitzenmacher, "The power of two choices in randomized load balancing," vol. 12, no. 10, pp. 1094–1104. [Online]. Available: http://ieeexplore.ieee.org/document/963420/

[5] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012*, M. Gerla and D. Huang, Eds. ACM, 2012, pp. 13–16. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[6] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the Leading Edge of Mobile-Cloud Convergence," in *Proceedings of the 6th International Conference on Mobile Computing, Applications and Services*. Austin, United States: ICST, 2014. [Online]. Available: http://eudl.eu/doi/10.4108/icst.mobicase.2014.257757

[7] V. Mittal, S. Qi, R. Bhattacharya, X. Lyu, J. Li, S. G. Kulkarni, D. Li, J. Hwang, K. K. Ramakrishnan, and T. Wood, "Mu: An efficient, fair and responsive serverless framework for resource-constrained edge clouds," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. Association for Computing Machinery, pp. 168–181. [Online]. Available: https://doi.org/10.1145/3472883.3487014

[8] R. Bhattacharya and T. Wood, "BLOC: Balancing Load with Overload Control In the Microservices Architecture," in *3rd IEEE International Conference on Autonomic Computing and Self-Organizing Systems*, ser. ACSOS '22, p. 10.