A Learning-based Dynamic Load Balancing Approach for Microservice Systems in Multi-cloud Environment

Jieqi Cui
School of Systems Science and Engineering
Sun Yat-sen University, Guangzhou, China
cuijq5@mail2.sysu.edu.cn,

Pengfei Chen*, Guangba Yu
School of Data and Computer Science
Sun Yat-sen University, Guangzhou, China
chenpf7@mail.sysu.edu.cn, yubg5@mail2.sysu.edu.cn

2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS) | 978-1-7281-9074-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICPADS51040.2020.00052

***Abstract*—Multi-cloud environment has become common since companies manage to prevent cloud vendor lock-in for security and cost concerns. Meanwhile, the microservice architecture is often considered for its flexibility. Combining multi-cloud with microservice, the problem of routing requests among all possible microservice instances in multi-cloud environment arises. This paper presents a learning-based approach to route requests in order to balance the load. In our approach, the performance of microservice is modeled explicitly through machine learning models. The model can derive the response time from request volume, route decision, and other cloud metrics. Then the balanced route decision is obtained from optimizing the model with Bayesian Optimization. With this approach, the request route decision can adjust to dynamic runtime metrics instead of remaining static for all different circumstances. Explicit performance modeling avoids searching on an actual microservice system which is time-consuming. Experiments show that our approach reduces average response time by 10% at least.**

***Keywords*-multi-cloud; microservices; load balancing; performance modeling; request routing;**

I. Introduction

As cloud-computing technology continues to develop, multi-cloud becomes a common deployment scheme for large-scale systems like Microsoft Azure and OpenStack [1]. Multi-cloud solution can prevent vendor lock-in by distributing the system to each independent cloud service provider. If any of the clouds or services goes down, replica or backup servers on other clouds can still guarantee availability in case of this partial fault. Multi-cloud also allows the enterprise to combine various cloud services to meet the demand of both cost and performance [2]. To better accommodate the cloud environment, microservice architecture is leveraged to construct a system with individual service components [3]. Unlike the monolithic system that requires build, deploy and run all components simultaneously, a microservice system is decoupled into separate components, usually deployed in form of containers, enabling distributed deployment and running in a multi-cloud environment. Currently with the help of service mesh and container orchestration platform like Istio[1] and Kubernetes[2], multi-cloud microservice architecture could be built with little effort.

Though leveraging microservice in multi-cloud environment offers much benefit, the problem balancing load among services across clouds arises. Microservice system is composed of multiple independent running servers. For each of these microservices, replicas can reside in different clouds and provide identical service together. Requests aimed at these services needed to be route among all replicas appropriately for best performance. Meanwhile, all microservices load in the system should be balanced simultaneously for optimizing the overall response time of external requests to the microservice system.

While traditional load balancing focuses on the static mapping of request to server and preconfigured route, the increasingly complex multi-cloud environment and microservice system have deeply impacted the performance of this method [4]. In multi-cloud microservice environment, there are following challenges.

- **Varied latency.** The latency between clouds often fluctuates significantly caused by the instability of network connection. The latency is also affected by the geographical distance of clouds.
- **Dynamic replicas.** The number of service replica is not static due to replica scaling or container crashes.
- **Fluctuated requests.** The number of user requests to each cloud also rises and falls frequently.
- **Complexity of microservice system.** Microservice system has heterogeneous services which differ greatly in performance and rather complicated service dependence relationship that can cause a chain reaction, dampening the overall response time.

The above challenges pose a great challenge for load balancing in multi-cloud microservice environment. In this scenario, an approach that can adjust load balancing strategy dynamically and efficiently is demanded to cope with the constantly changing environment. In order to tackle these

*Corresponding author.

[1]Istio [https://istio.io/]
[2]Kubernetes [https://kubernetes.io/]

2690-5965/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPADS51040.2020.00052

challenges, we propose a learning based load balancing approach. Our approach consists of two major parts:

1) The performance of each microservice under various conditions is modeled explicitly with machine learning. An optimal load balancing route for individual microservice can be obtained by running Bayesian Optimization [5] search algorithm on the model.
2) Providing the optimal route of individual microservice, our approach merges all request chains into request tree and adjust the route of each microservice down the tree whenever condition change is detected.

Our approach breaks down the microservice system into individual microservices and optimizes them separately, avoiding optimizing the whole system at once which is comparatively more challenging. Load balancing of every microservice along the request path also guarantees the overall response time is optimized. On top of our approach, a system containing running environment, monitor and controller are built for validation, in which controller responsible for load balancing implements our approach. Experiments show that our approach achieves a 10% response time reduction in the evaluation.

This paper makes the following contributions:

- Our approach considers dynamic runtime factors such as cross-cloud latency, service replica number, and user requests to each cloud for load balancing.
- Our approach can find out the best load balancing route every quickly with explicit modeling of microservice performance which consumes little time.
- Our approach combines the optimal load balancing route of individual microservice for the overall response time optimization.

The remainder of this paper is organized as follows. Section II shows an overview and details of our system. Section III presents our experiments and discussions The related work is summarized in Section IV. Finally, we conclude this paper in Section V.

II. System Design

The whole system consists of a multi-cloud environment, monitor, and controller as illustrated in Fig. 1. All clouds host the microservice system together and generate metrics for monitoring. User requests are issued to the clouds directly. The monitor is responsible for runtime metrics collection from clouds which is used as a data source for controller. Runtime metrics are mostly conditions of clouds and microservice system that affect performance and quality of service that reflects performance. These metrics are provided to controller for load balancing decision. All metrics are collected and provided dynamically at runtime. Controller is the main implementation of our approach. At any moment, controller will check runtime metrics for environment condition changes. When distinct changes are detected, controller will start load balancing regarding the whole multi-cloud microservice system by optimizing the route of each microservice.

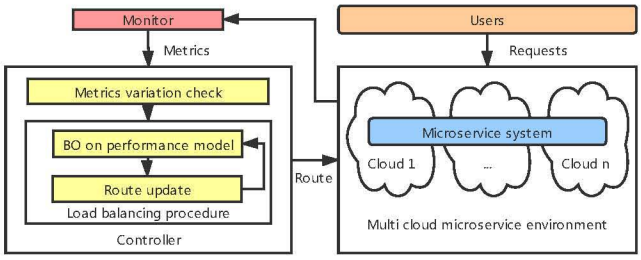

Figure 1. System overview.

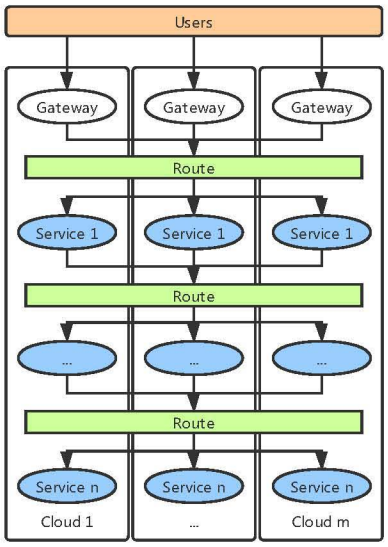

Figure 2. Multi-cloud microservice environment.

A. Multi-cloud Microservice Environment

In multi-cloud microservice environment, all clouds coordinate to run exactly one microservice system as shown in Fig. 2. For each microservice, there exists multiple replicas in each cloud and the number of replicas may differ. Along the request chain of microservice system, requests for each microservice are routed among all available clouds for load balancing.

Environment setting Each cloud is orchestrated by Kubernetes and all microservices are deployed on it as Pods which are collection of containers. The number of service replica cloud be scaled and discovered with Kubernetes API. Istio as service mesh connects all microservices and clouds. Routing is implemented by Virtual Service and Destination Rule of Istio. If the destination microservice is at other clouds, requests will be proxied to the dedicated Istio gateway of that cloud and route to the corresponding service.

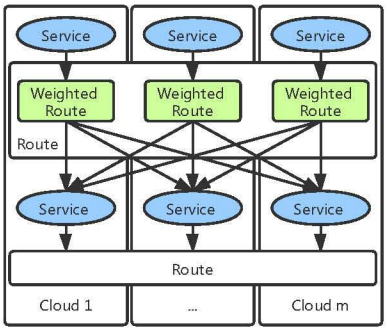

Figure 3. Microservice route.

User requests. Similar to real-life multi-cloud environment, user requests to the microservice system can arrive at any of the cloud depending on region and preference. Also, the number of requests often fluctuates considerably as time goes by. Therefore significant variation exists in request, implying the need for dynamic load balancing.

Microservice route As illustrated in Fig. 3, route for microservice is done at each cloud separately. Each route contains n weighted destinations, corresponding to microservices in all n clouds. All route works together for individual microservice load balancing. The weight of the destination represents the probability of routing requests to that cloud.

When a user request hits any one of the clouds, it will first be received by the gateway. Next, gateway will issue internal requests aiming at Service 1. This request will be routed to service 1 at the cloud specified by route. Then service 1 will send requests for service 2, which is also routed the same way described above. Any request in the request chain is routed in this manner. Requests routed to microservice will be evenly distributed to all replicas of it.

B. Monitor

The monitor collects dynamic runtime metrics of three aspects: environment, microservice, and quality metrics.

- **Environment metrics.** Environment metrics include the number of each microservice replica, latency (RTT) among all clouds and user requests to each cloud.
- **Microservice metrics.** Microservice metrics include service request per second (RPS), service request count sum, service response time sum, and children services that are invoked. These metrics are collected at each cloud separately. Service RPS is averaged within a specified period. Service response time and request count sum are accumulated within the same specified period. Service response time sum here contains processing time and network latency of both service itself and its children services.
- **Quality metrics.** Quality metrics are the response time of external user requests.

The monitor in our system is implemented based on APIs provided by Istio and Prometheus[3]. Due to lack of tracing, the service response time metric is collected in the form of the sum of all requests instead of each request individually. Most runtime metrics like RPS (Request Per Second), request count, and response time are collected within a specified period because of scrape interval setting of Prometheus. In our approach, the period is 10 seconds. Service response time which does not include children response time is computed by removing the latter one. Metrics of environment, quality, and microservice including average response time are provided to controller for searching the optimal route on model and experiments for benchmarking.

C. Controller

The controller is composed of four components: metrics variation check, BO on performance model, route update and load balancing procedure. Notice that BO on performance model implements our approach. While metrics variation check takes care of detecting the changes of metrics, load balancing procedure will run BO on performance model and route update for each microservice along request tree to optimize the overall workload.

- **Metrics variation check.** The metrics variation check is invoked every preconfigured cycle. When the metrics variation check is invoked, it will retrieve environment metrics mentioned in Section II-B and compare these metrics to metrics retrieved at the last invocation. If more than a preconfigured degree of changes are detected in any of these metrics, load balancing will be started to accommodate environment changes. In our approach, the cycle is set to 20 seconds and the degree of changes is configured to 20%.
- **BO on performance model.** This component contains both BO and performance modeling, which are the core of our approach. Performance model takes environment and microservice metrics as input and outputs microservice response time. BO treats the route as parameters and searches for the minimal response time based on the performance model. Details of performance model and BO are stated in the following.
- **Route update.** Route update is responsible for individual microservice route update. Route update configures the Virtual Service settings of microservice at each cloud.
- **Load balancing procedure.** The load balancing procedure is illustrated in Fig. 4. Notice that in our approach, a request path is the union of all request chains of external requests. In other words, it is the architecture of microservice system. The procedure starts from the first service, usually frontend service, and execute route

[3]Prometheus [https://prometheus.io/]

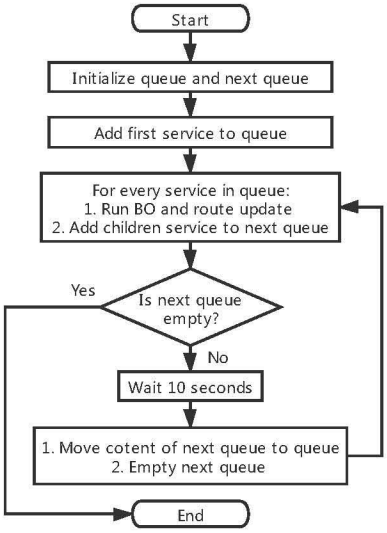

Figure 4. Load balancing procedure.

update layerwise. This is because when the route of parent service changes, the load status of it at each cloud will change correspondingly. Then this will lead to changes in the sub request for children service at each cloud. To sum up, changing the route of parent service will result in RPS changes of children at each cloud. Therefore, parent service is needed to update before children services and services at the same layer on the request tree can be updated simultaneously. When one layer of route update finishes, controller has to wait for 10 seconds to guarantee that metrics are accurate after route update. This is necessary due to latency in route update and monitor scrape interval mentioned in Section II-B. The procedure ends when all microservice load are balanced.

1) Performance Modeling: The performance modeling is a common technique to model performance of all sorts of situations including server and network [6], [7]. Briefly speaking, performance modeling explicit construct the numerical relationship between factors that impact performance and performance metric. In our approach, model depicts the performance of individual microservice under all sorts of environments where replica at each cloud, clouds latency, and route are different. The chosen machine learning model is neural network because it excels at regression task. The training of model is detailed in Section III. Model details are stated as following:

Explicit performance modeling. Notice that BO actually can search route on the system directly, but each step could take up 10 seconds because from applying route to retrieving response time requires such a long time. On the one hand, this long optimization step could lead to extremely low convergence. On the other hand, environment status may

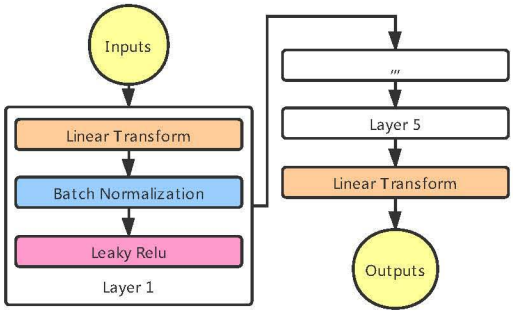

Figure 5. Network architecture.

vary from the ones when BO just starts. Based on this discussion, explicit performance modeling is adopted in our approach.

Inputs and Output. All inputs and output belong to the same microservice, thus reflecting single microservice performance. Output of model is average service response time, a critical metrics that reflect performance. Inputs to model includes environment metrics, microservice metrics, and request shift:

- Environment metrics include service replica at each cloud and latency (RTT) among clouds, representing capacity of microservice and cost to route request to other clouds respectively.
- Microservice metrics are RPS at each cloud and standard processing times. They carry the information of load status and service capacity characteristics.
- Request shift supplements route status to model.

Standard processing time. Standard processing time is service response time under a range of RPS while other factors remain identical. Therefore it acts as a performance characterization of microservice. This metric will be fed to performance model as input. Standard processing time is used to indicate the capacity of different services, which can be regarded as performance modeling in a static environment.

Request shift. Request shift is the compact version of route, which is the actual parameters to search with BO. Each shift represents the amount of traffic shift between a pair of clouds. Postive and negative values indicates the direction of shift. This is because for n clouds, there will be $n * n$ weight variables since the route is done at each cloud separately as mentioned in II-A.

Neural networks model. Artificial neural networks [8] is mathematical abstraction of biological neural networks. The key of neural networks is multiple layers of linear transform, convolution, activation, and other structures [9]. This multilayer architecture large empowers the modeling ability of neural networks.

In our approach, a neural network of 5 layers is proposed as shown in Fig. 5. Each layer of network consists of linear

transform, batch normalization [10] and Leaky ReLU. All these linear transforms have same 100 hidden nodes. While linear transform and Leaky ReLU capture the mapping, batch normalization prevents overfit and accelerates learning. The ending linear transform converts activation of the final layer to scalar output.

To showcase effectiveness of our neural networks model, Support Vector Regression (SVR) [11], [12], which is also a well known machine learning model for regression task, is compared in III.

2) Bayesian Optimization: Bayesian Optimization is a technique to perform optimization without knowledge of the target function except for input and output. BO is often applied when the function is expensive to evaluate because it attempts to find optimal in the minimal number of steps and requires much fewer steps than grid search. Usually the function is black-box, meaning that the analytic expression and derivatives are unknown. Hyperparameters tunning of machine learning model is in this scope [13]. BO also worked well in optimizing target function directly [5]. In our approach, BO is leveraged to optimize performance model by searching request shift, which is then converted back to route and applied to environment. Notice that there

The main idea of BO is building a surrogate model of black-box target function and predicting optimal point with acquisition function based on the model. Gaussian process (GP) is often considered for the surrogate model as well as Expected Improvement (EI) [14] for acquisition function. Both of them are utilized in our approach. The process of BO contains three steps. The first is to optimize the acquisition function over GP to find the next best sampling point. The second is to retrieve the value of target function with the sampling point. The final step is updating the GP with the new sample for a better approximation of the target function. This process is repeated for a specified steps. Details of Gaussian Process and Expected Improvement are stated as following:

Gaussian Process. Gaussian Process is a collection of random functions which is subject to multivariate Gaussian distribution. In BO, GP is used as prior distribution of target function and can be formulated as following:

$$f(x_{1:n}) \sim N(\mu(x_{1:n}), K(x_{1:n}), x_{1:n}))) \tag{1}$$

where $f(x_{1:n})$ and $x_{1:n}$ are sample points. This distribution is defined by the mean function μ and the positive definite covariance function k.

Given n samples, the posterior distribution of $f(x_{n+1})$ providing x_{n+1} can be computed with Bayes' theorem as:

$$f(x_{n+1})|f(x_{1:n}) \sim N(\mu_n(x_{n+1}), \sigma_n^2(x_{n+1})) \tag{2}$$

Expected Improvement. Expected Improvement finds next sample with greatest expectation. It is defined as:

$$EI(x) = E(\max f(x) - f(x_{best}), 0) \tag{3}$$

where s and x_{best} are next sample point and the sample point with best value so far respectively.

III. Evaluation

A. Experimental Settings

Environment setup. We set up three Kubernetes clusters in the same local network. Two of them have three nodes and the rest one has four nodes. All ten nodes are KVM virtual machines with the same specification, which is 2-core CPU, 4GB of memory, and Ubuntu 18.04 OS. Each of these Kubernetes clusters represent a cloud. Multi-cloud microservice environment is built by Istio multi-cluster setup that enables cross-cluster requests. Cloud latency is simulated by tcconfig[4], a tc command wrapper.

Benchmarks. Bookinfo[5] and Online Boutique[6] are benchmarks leveraged in our experiments. Bookinfo is an example microservices of Istio. It has four microservices, which are all stateless and only one type of external request. Load generation for this benchmark only issues the unique external request. Online Boutique is another demo microservices provided by GoogleCloudPlatform which has twelve components. Among these components, there are ten stateless microservices, one load generator microservice, and one storage service. We only run experiments on those stateless microservice and the storage service is not load balanced. There are six different requests, so the load generator will issue all requests simultaneously at the specified RPS.

Offline performance model training. Unlike BO that can predict online, performance model in our approach requires training offline first. Both SVR and neural networks are trained with a pre-collected offline dataset. The dataset consists of standard processing time and other metrics according to the inputs and output mentioned in Section II-C1. For standard processing time, it is collected under the environment where only one cloud is available and all replicas are set to one. The range of RPS for it is [10, 20, 30, 40, 50]. For other metrics, they are collected under all sorts of conditions of the environment. We set up environment with random service replica, clouds latency, and RPS and then test the performance of all services with various request shift. 2500 sample points of other metrics are collected for each benchmark. The dataset is split into train and test part with train ratio of 70%. Evaluation of performance model on dataset is detailed in Section III-B.

For SVR, the penalty term C is tuned to 960.77. For neural networks, it is trained with learning rate 0.5 and weight decay 10. for 150 epochs. Adam optimizer and mean square error loss are used. These parameters are tuned on Bookinfo and used on Online Boutique directly.

[4]tcconfig [https://github.com/thombashi/tcconfig]

[5]Bookinfo [https://istio.io/latest/docs/examples/bookinfo/]

[6]Online Boutique [https://github.com/GoogleCloudPlatform/microservices-demo/]

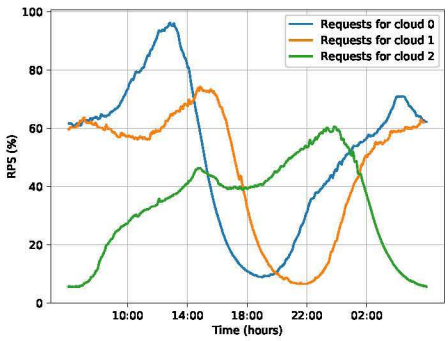

Figure 6. User requests per second at each cloud.

Online test settings. Online test is the standard testing setup of our experiments. For all following evaluations that are under online test settings, the test setup is identical. Online test settings include clouds latency, service replica and request. The former two are set to change randomly as time goes by. Notice that though the changes are random, the sequence is identical in every evaluation. Requests to clouds are simulated according to RPS curves from facebook [4]. Curves are shown in Fig. 6 All settings are limited to preconfigured range. For latency, the random range is 10ms to 40ms. For replica, two services are selected to change randomly while the rest services remain one replica at each cloud due to limited memory of virtual machines. As for services selected, the replica of them at each cloud will be scaled randomly. The scaled ranges are $[1, 3]$ and $[1, 2]$ for Bookinfo and Online Boutique respectively also due to memory limitation. The request ranges are $[0, 40]$ and $[0, 10]$ for both benchmarks. Notice that the latter one is RPS for a set of requests, hence smaller. The number of requests to clouds is obtained by multiplying the percentage of RPS curves sample point to the max request of range.

B. Performance Model Evaluation

The performance model is evaluated in two aspects. One is the test part of offline dataset. The other is online test settings. For the test part of offline dataset, the performance of SVR and neural networks is listed in Table I for both benchmarks. For online test settings, it is listed in Table II. Evaluation metrics include R^2, MSE and MAE. R^2 is coefficient of determination, a metric that represents the proportion of variance of y explained by the model. The best possible value of it is 1.0 which means all variance of y is explained. MSE, and MAE are mean square error and mean absolute error. Both of them calculated the average error of prediction. While the former one calculates the error by the square difference between prediction and true value, the latter one does by absolute difference.

From the tables, we can notice that all evaluations have

Table I
PERFORMANCE ON BOOKINFO AND ONLINE BOUTIQUE OFFLINE DATASET.

	Bookinfo			Online Boutique		
	R^2	MSE	MAE	R^2	MSE	MAE
SVR	0.84	156.56	6.20	0.68	215.38	9.00
NN	0.85	144.04	6.42	0.67	220.73	8.85

Table II
PERFORMANCE ON BOOKINFO AND ONLINE BOUTIQUE UNDER ONLINE TEST SETTINGS.

	Bookinfo			Online Boutique		
	R^2	MSE	MAE	R^2	MSE	MAE
SVR	0.82	136.34	7.51	0.58	291.32	9.99
NN	0.86	102.68	6.29	0.52	328.93	9.69

R^2 over 0.5, indicating that both SVR and neural networks can model performance effectively under heterogeneous conditions of the environment. In terms of benchmark, the performance on Bookinfo is better than Online Boutique under both offline and online. The gap is resulted form higher complexity of microservice system of Online Boutique and direct usage of hyperparameters from Bookinfo. Another thing worth noting is R^2 of online test settings decreases slightly from the one of offline dataset. This is because environment conditions behind metrics are not the same, but the tiny gap between R^2 indicates that model has good generalization ability. The metrics of SVR and neural networks are close in all perspectives. However, it can be recognized that neural networks is better than SVR on Bookinfo but in a reversed way on Online Boutique. This is also the result of direct usage of parameters. While SVR is more robust with parameters, neural networks is more sensitive. Given the difference between two benchmarks, the slight performance reduction is acceptable.

MSE and MAE reveal more insights. While MAE varies little among all evaluations, MSE nearly doubles from Bookinfo compared to Online Boutique. This indicates that there are some predtions differs from true values greatly. Notice that the size of training set are the same for both benchmarks. This exhibits performance pattern of a complex microservice system is harder to cover given the same number of training samples.

C. Bayesian Optimization Evaluation

The convergence of BO is evaluated. In detail, one online test moment is sampled randomly under online test settings for each benchmark. This moment contains replica, latency, and request settings at that time and only one service is picked for evaluation. Then, BO is run on SVR and neural networks under these two moments with maximum step set to 20. Fig. 7 illustrates the convergence of BO. In the figure, the minimum value among all currently found observations

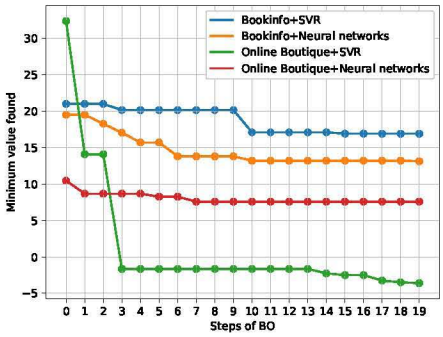

Figure 7. Minimum among current observations of BO at each step.

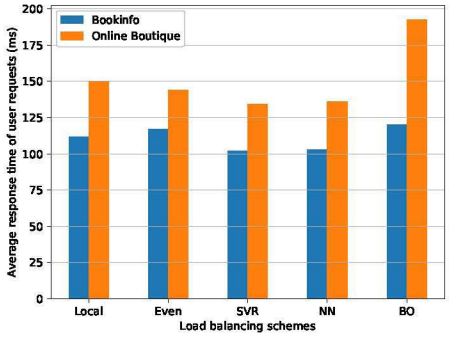

Figure 8. Average response time of user requests.

at each step are marked. It can be noticed that all cases reach global minimum within 10 steps, suggesting the convergence is relatively fast. With Bookinfo, BO finds the minimum at the same pace on both models. The gap is small given that it is close MAE of models. As for Online Boutique, BO on neural networks gives a negative minimum which is impossible in reality considering the value is response time. It shows that neural networks does not cover the performance pattern correctly.

D. Overall Performance Evaluation

Overall performance is evaluated regarding response time of user requests. Every part of our system is involved in this evaluation. As the environment is running according to online test settings, the monitor oversees it and controller balances the load of it. In this evaluation, other load balancing schemes are added for comparison. Local scheme is to route all requests to local microservice and do no load balance. Even scheme distributes the requests among all clouds evenly. BO is running BO on system directly without performance modeling, in which case, parameters are still request shift but the target function is system itself. SVR and NN (neural networks) are load balance strategy of our approach. For all BO, the step is set to 10.

The average response time of user requests under different load balancing scheme is shown in Fig. 8. SVR and neural networks has similar time and it is lower than Local and Even scheme by around 10ms, which exhibits the effectiveness of our approach. BO, however, is worst among all schemes because the execution step of it is to low to find the global minimum. For both benchmarks, the conclusion is similar on the whole. On the whole, the response time reduction is around 10% compared to other schemes, which is a considerable reduction given the time is at millisecond scale.

The execution time of different schemes is illustrated in Fig. 9. It is noticeable that BO has a much larger execution time than any other schemes. This is because each step

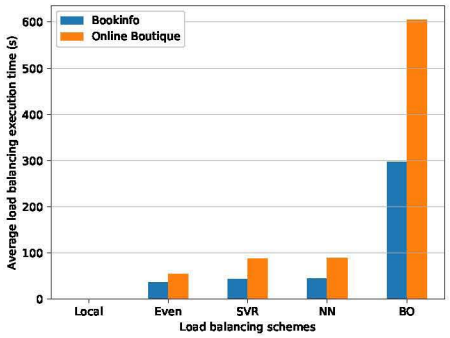

Figure 9. Load balancing execution time.

of BO requires retrieving target value from system, which takes 10 seconds in our setup. Therefore it is unsuitable for searching load balance under a highly dynamic environment. The time for SVR and neural networks are comparatively much smaller and both under 100 seconds. Notice that the time for Online Boutique is almost double of Bookinfo, due to double size of microservices of former one.

IV. Related Work

Extensive research on multi-cloud, microservice, performance modeling, and load balancing exists currently. Most of them focus on the intersection of these areas.

Modeling performance of microservice system for further capacity planning is popular. [6] approximates the capacity of a microservice under by fitting a regression model on data collected from load tests. The underlying infrastructure of microservice is considered in [15] and a model of both micro and macro details is proposed. [16] formulate a microservice-based application workflow scheduling problem and design a multi-layer performance model for minimal end-to-end delay. Modeling burstable instances of clouds for cost saving is presents in [17].

Performance optimization under multi-cloud is studied frequently. Taiji [4] focuses on edge to datacenter traffic

routing and proposes an approach to optimize the route for balancing the utilization of data centers. [18] builds a global queue for job scheduling under multiple geo-distributed datacenters. In [19], jobs are balanced by predicting the bottlenecks based on network latency, CPU, and memory of all available nodes. A microservice brokering approach regarding cost and performance by selecting and leasing virtual machines (VMs) [20] is proposed. The future directions of multi-cloud computing are discussed in [21].

There is much research related to load balancing. FastRoute [22] improved the performance of online services by a operational anycast-based system. A system that can deliver online services over integrated infrastructure efficiently [23] is proposed. [24] gives analysis of a decentralized load balancing algorithm, showing that it outperforms existing centralized approach. Ananta [25] is a load balancer runs on commodity hardware and meets the performance requirements of multi-tenant cloud computing environments.

V. Conclusion

In this paper, a learning-based load balancing approach is introduced and evaluated. Our approach leverages Bayesian Optimization for searching optimal route decision on performance model. A system for load balancing overall microservice system is designed and implemented. Experiments show that our approach out offers a reduction on average response time for about 10%.

Acknowledgments

The work was supported by the Key-Area Research and Development Program of Guangdong Province (2020B010165002), the Basic and Applied Basic Research of Guangzhou (202002030328), the Natural Science Foundation of Guangdong Province (2019A1515012229), and the Natural Science Foundation of China (61802448). The corresponding author is Pengfei Chen.

References

[1] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "Charm: A cost-efficient multi-cloud data hosting scheme with high availability," *ieee international conference on cloud computing technology and science*, vol. 3, no. 3, pp. 372–386, 2015.

[2] G. Liu and H. Shen, "Minimum-cost cloud storage service across multiple cloud providers," *IEEE ACM Transactions on Networking*, vol. 25, no. 4, pp. 2498–2513, 2017.

[3] M. Villamizar, O. Garces, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *2015 10th Computing Colombian Conference (10CCC)*, 2015, pp. 583–590.

[4] D. Chou, T. Xu, K. Veeraraghavan, A. Newell, S. Margulis, L. Xiao, P. M. Ruiz, J. Meza, K. Ha, S. Padmanabha, K. Cole, and D. Perelman, "Taiji: managing global user traffic for large-scale internet services at the edge," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 430–446.

[5] P. I. Frazier, "A tutorial on bayesian optimization," *ArXiv*, vol. abs/1807.02811, 2018.

[6] A. Jindal, V. Podolskiy, and M. Gerndt, "Performance modeling for cloud microservice applications," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 25–32.

[7] K. Salah, K. Elbadawi, and R. Boutaba, "Performance modeling and analysis of network firewalls," *IEEE Transactions on Network and Service Management*, vol. 9, no. 1, pp. 12–21, 2012.

[8] D. Specht, "A general regression neural network," *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568–576, 1991.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of The ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of The 32nd International Conference on Machine Learning*, 2015, pp. 448–456.

[11] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[12] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[13] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 2951–2959.

[14] D. J. C. Mackay, "Introduction to gaussian processes," *NATO advanced study institute on generalization in neural networks and machine learning*, pp. 133–165, 1998.

[15] H. Khazaei, C. Barna, and M. Litoiu, "Performance modeling of microservice platforms considering the dynamics of the underlying cloud infrastructure." *arXiv preprint arXiv:1902.03387*, 2019.

[16] L. Bao, C. Q. Wu, X. Bu, N. Ren, and M. Shen, "Performance modeling and workflow scheduling of microservice-based applications in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2114–2129, 2019.

[17] Y. Jiang, M. Shahrad, D. Wentzlaff, D. H. Tsang, and C. Joe-Wong, "Burstable instances for clouds: Performance modeling, equilibrium analysis, and revenue maximization," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1576–1584.

[18] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 2015, pp. 111–124.

[19] H. Wang and B. Li, "Lube: mitigating bottlenecks in wide area data analytics," in *Proceedings of the 9th USENIX Conference on Hot Topics in Cloud Computing*, 2017, pp. 1–1.

[20] T. Shi, H. Ma, and G. Chen, "A genetic-based approach to location-aware cloud service brokering in multi-cloud environment," in *2019 IEEE International Conference on Services Computing (SCC)*, 2019, pp. 146–153.

[21] R. Buyya and J. Son, "Software-defined multi-cloud computing: A vision, architectural elements, and future directions," in *International Conference on Computational Science and Its Applications*, 2018, pp. 3–18.

[22] A. Flavel, P. Mani, D. A. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev, "Fastroute: a scalable load-aware anycast routing architecture for modern cdns," in *NSDI'15 Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, 2015, pp. 381–394.

[23] H. H. Liu, R. Viswanathan, M. Calder, A. Akella, R. Mahajan, J. Padhye, and M. Zhang, "Efficiently delivering online services over integrated infrastructure," in *NSDI'16 Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, 2016, pp. 77–90.

[24] M. Rusek and J. Landmesser, "Time complexity of an distributed algorithm for load balancing of microservice-oriented applications in the cloud," *ITM Web of Conferences*, vol. 21, p. 18, 2018.

[25] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri, "Ananta: cloud scale load balancing," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, vol. 43, no. 4, 2013, pp. 207–218.