Load Balancing across Microservices

Yipei Niu[1] Fangming Liu*[1] Zongpeng Li[2]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology
[2]University of Calgary, Canada

***Abstract*—With the advent of cloud container technology, enterprises develop applications through microservices, breaking monolithic software into a suite of small services whose instances run independently in containers. User requests are served by a series of microservices forming a chain, and the chains often share microservices. Existing load balancing strategies either incur significant networking overhead or ignore the competition for shared microservices across chains. Furthermore, typical load balancing solutions leverage a hybrid technique by combining HTTP with message queue to support microservice communications, bringing additional operational complexity. To address these challenges, we propose a chain-oriented load balancing algorithm (COLBA) based solely on message queues, which balances load based on microservice requirements of chains to minimize response time. We model the load balancing problem as a non-cooperative game, and leverage Nash bargaining to coordinate microservice allocation across chains. Employing convex optimization with rounding, we efficiently solve the problem that is proven NP-hard. Extensive trace-driven simulations demonstrate that COLBA reduces the overall average response time at least by 13% compared with existing load balancing strategies.**

I. Introduction

Driven by latest container technology, the microservice architecture decomposes a monolithic cloud application into a collection of small services, and has been adopted by IT giants such as Amazon, Netflix, and Uber [1]–[3].

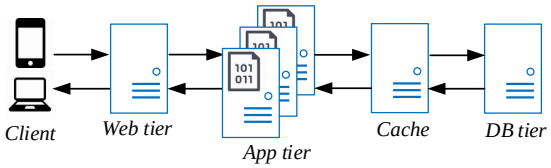

Fig. 1. Overview of a traditional monolithic project architecture.

As shown in Fig. 1, web applications are typically developed in a monolithic fashion. Services provisioned by the application are tightly coupled, making them complex to maintain, upgrade and test. All the modules and functionalities are developed in one piece of monolithic code, which is then deployed across multiple hosts in the application tier. A small change of the code requires redeployment across the entire infrastructure, incurring significant maintenance overhead.

*This work was supported in part by the National Key Research & Development (R&D) Plan under grant 2017YFB1001703, in part by NSFC under Grant 61722206 and 61761136014 (NSFC-DFG) and 61520106005, in part by the National 973 Basic Research Program under Grant 2014CB347800, and in part by the Fundamental Research Funds for the Central Universities under Grant 2014YQ001, 2014TS006 and 2017KFKJXX009. (Corresponding author: Fangming Liu)

Different from the monolithic architecture, in a microservice system, an application is developed as a suite of microservices, each running independently on containers. Fig. 2 plots partial functionalities of an ecommerce website developed in a microservice fashion, where each container, i.e., microservice instance, can be dynamically created and removed.

However, apart from the high agility and scalability brought by microservices, service providers are required to orchestrate hundreds of microservices efficiently [4]. To this end, a critical problem needs to be addressed: how to efficiently balance load across microservices?

In a microservice system, homogeneous user requests traverse a set of microservices in succession, thereby served in a chain. The instances of a microservice may serve multiple chains. As shown in Fig. 2, chains A and B traverse the product microservice simultaneously. Hence, it is inevitable that chains compete for a microservice against each other. Furthermore, user requests of chains have different QoS requirements, as well as different processing times. Existing load balancing solutions fail to take either request heterogeneity or inter-chain competition into consideration.

To address the above issues, instances of a microservice should be isolated across chains, and requests of a chain should be carefully steered to pass through instances that belong to them. As a result, chain-oriented load balancing is required to judiciously balance requests across their exclusive microservice instances. Unfortunately, existing communication pattern, which combines *HTTP* with *message queue*, complicates interconnection management across microservice instances, bringing extra challenges towards achieving chain-oriented load balancing.

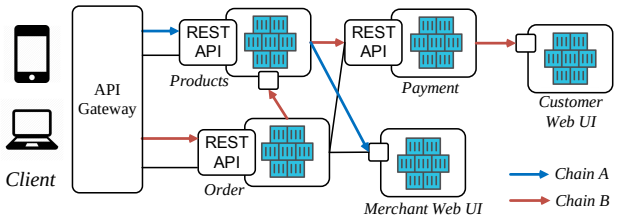

Fig. 2. A microservice architecture of an e-commerce website.

Purely employing the message queue technique, we propose in this work a Chain-Oriented Load Balancing Algorithm (COLBA) to address the above challenges. On one hand, for homogeneous requests of a chain, COLBA should determine how many instances of a microservice to serve the requests in

a chain, so as to minimize the average response time. On the other hand, concerning request heterogeneity and competition, COLBA needs to decide how many instances of a microservice are sufficient to serve chains that pass through.

The challenges of designing COLBA are three folds. First, pure message communication fails to support the synchronous pattern, providing no guarantee in serving interactive requests. As a result, it is challenging for a pure message solution to bound average response time of each microservice chain. Second, the scale of microservice population is much larger than that of VMs, *e.g.*, Netflix employs over 600 microservices to support its business [4]. Therefore, for each round of the balancing strategy, our algorithm is required to determine instance assignment across chains promptly to meet fluctuating loads. Third, jointly determining instance scale within a chain and assigning instances across chains are challenging due to chain heterogeneity.

Our contributions of this work are summarized as follows.

- We model the load balancing problem as a non-cooperative game, and leverage Nash bargaining solution to bound the average response time of every chain and coordinate microservice allocation across chains.
- By employing the convex optimization technique with rounding, we (approximately) solve the problem that is proven to be NP-hard within dozens of iterations. Theoretical analysis proves that the solution is within a constant gap of the optimum.
- We carry out extensive trace-driven simulations to demonstrate that COLBA can reduce the overall average response time by 13% and 41% respectively, compared with microservice-oriented and instance-oriented load balancing solutions.

II. Background and Motivation

A. Load Balancing across Microservices

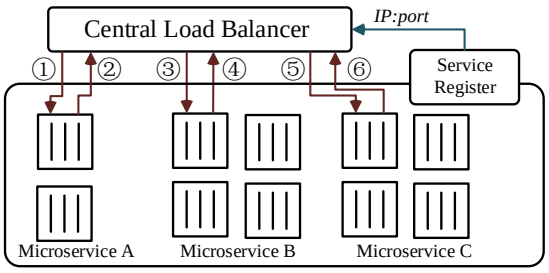

Fig. 3. An overview of instance-oriented load balancing strategy.

With respect of load balancing in the context of microservice systems, since instances of microservices independently run in containers, operators are supposed to determine the balancing scope, i.e., the basic unit chosen to balance load across. For example, an aggressive strategy chooses instance as the balancing scope, in which a central load balancer balances requests across instances [5]. As shown in Fig. 3, suppose that requests of a chain traverse microservices A, B, and C. Lacking request redirection from one microservice to another, the load balancer has to send the request to a specific instance, receive response, and send it to the next microservice instance. Such an instance-oriented solution can balance load across instances well, but at the price of significant overhead in communicating with instances back and forth.

To alleviate the overhead of the aggregative strategy, an alternative is a microservice-oriented solution. As shown in Fig. 4, instances of the same microservice are managed by a service registery, and load is balanced across them evenly. However, instances of a microservice may serve multiple chains, whose requests have different requirements of performance and various service times. For example, requests that query a product's specifications are more urgent than those that insert a log record, and it takes longer to update product specification than to insert a log entry. Furthermore, it is common that different chains are served by the same microservice, competing for microservice instances against each other. In short, the microservice-oriented solution is oblivious to the heterogeneity of requests and competition across chains. To overcome the disadvantages of both solutions, a chain-based load balancing solution is required.

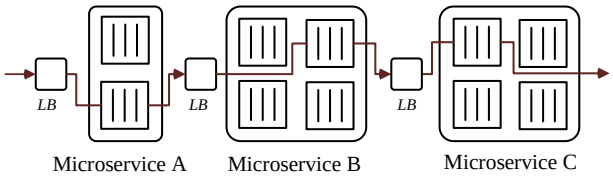

Fig. 4. Overview of microservice-oriented load balancing strategy.

B. Communication across Microservices

Communication techniques used by a microservice system fall into two categories, i.e., *HTTP* and *message queue*. Using **HTTP**, a microservice instance can initiate a connection to anyone without relying on an intermediate broker. However, since HTTP communication requires exact locations (URLs) of microservice instances, it highly depends on service registries such as Zookeeper [6], to discover instances and API gateways to dispatch requests. Furthermore, designed and intended for one-to-one request/response, HTTP fails to support either one-to-many or asynchronous patterns. Compared with HTTP, **message queue** supports all communication patterns except the synchronous mode. A hybrid technique may combine HTTP and message queue to support microservice communication; however, such a hybrid method comes with additional operational complexity.

Modern web applications are developed using the Task Asynchronous Paradigm (TAP) [7], [8], making it possible to use a pure message queue technique to support microservice communication. Direct TCP connection between message producers and consumers is absent in message queues, and consequently no guarantee is provided on response time and request status. That further compromises user-side QoS. To overcome the disadvantages above, this work aims to propose a

pure message queue based pattern with performance guarantee. A detailed comparison among HTTP, message queue, and our solution is shown in Table I.

TABLE I
COMPARISON BETWEEN HTTP AND MESSAGE QUEUE

	HTTP	*Message Queue*	*Our Solution*
One-to-One	✓	✓	✓
One-to-Many	✗	✓	✓
Synchronous	✓	✗	Guaranteed
Asynchronous	✗	✓	✓

C. Pure Message Chain-Oriented Load Balancing

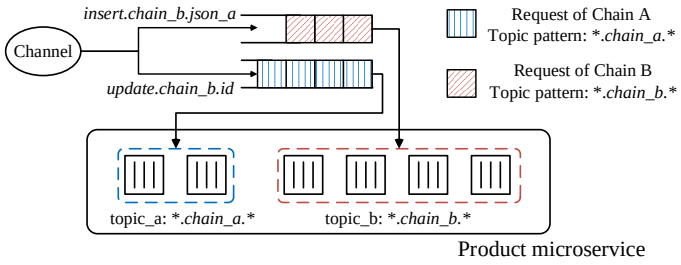

Fig. 5. The implementation of COLBA across chains in one microservice.

To address the issues above, we propose a Chain-Oriented Load Balancing Algorithm (COLBA) using message queues. As illustrated in Fig. 5, we leverage the *topic* mode of RabbitMQ [9] to balance loads across instances based on different chains. The upstream microservice declares a channel and binds two queues to it. The messages are routed to the corresponding queues based on topics. By updating the topic of each downstream instance, we are able to change the instance assignment to adapt to workload fluctuation. As such, we can balance load based on request heterogeneity and competition across chains.

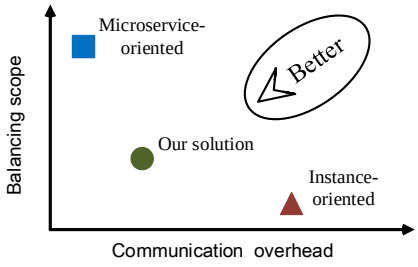

Fig. 6. The positioning of COLBA against instance-oriented and microservice-oriented solutions.

Fig. 6 illustrates the positioning of COLBA by comparing with instance-oriented and microservice-oriented solutions. We evaluate load balancing solutions in two dimensions, namely *balancing scope* and *communication overhead*. As shown in Fig. 6, the instance-oriented solution is most promising in handling heterogeneity and competition of requests, but incurs significant overhead due to frequent interaction between the central load balancer and instances. On the contrary, the microservice-oriented solution evenly balances load across instances of each microservice without bringing extra overhead, yet fails to take request heterogeneity and competition across chains into consideration.

III. System Model

TABLE II
KEY PARAMETERS

	Definition
C	The total number of microservice chains
M	The number of microservice types
I^m	Number of instances of microservice m
C^m	The total number of chains that passes microservice m
M^c	The total number of microservices which chain c passes
$\lambda_{c,m}$	The request arrival rate of chain c at microservice m
$\mu_{c,m}$	The request service rate of chain c at microservice m
U_c^b	an upper bound for response time of each chain

A. The Microservice Architecture

For an application developed in a microservice fashion, its architecture can be represented as a matrix $R = (r_{c,m})_{C\times M}$, where $r_{c,m} \in \{0, 1\}$ and indicates whether chain c traverses microservice m. The total number of chains that passes microservice m is then $C^m = \sum_{c=1}^{C} r_{c,m}$.

Moreover, the overall assignment strategy of the microservice architecture is $S = (s_{c,m})_{C\times M}$, where $s_{c,m}$ denotes the number of microservice m instances assigned to chain c.

The microservice system has a total capacity limit in the number of microservice instances that can be provisioned:

$$\sum_{c=1}^{C} r_{c,m} \cdot s_{c,m} \leq I^m,$$

where I^m is the total instances of microservice m.

B. Performance of Microservice System

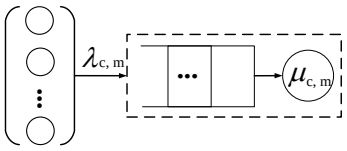

Fig. 7. A queue model for one microservice.

1) Response time of a microservice: Following existing literature, in the microservice system, we assume that requests belonging to a chain arrive at the API gateway as a Poisson process [10]–[12]. Moreover, we assume that the service

time of requests in a microservice instance follows general distribution. We represent the request arrival rate of chain c at microservice m as $\lambda_{c,m}$. Meanwhile, let $\mu_{c,m}$ be the average rate that a microservice m instance serves requests of chain c. We model the serving process of each microservice as an M/G/1/PS queue [13], as depicted in Fig. 7. Hence, the average response time of chain c in microservice m is:

$$u_{c,m}(s_{c,m}) = \int_0^\infty \frac{z^{c,m}}{1-s_{c,m}} dF(z^{c,m}) = \frac{E[Z^{c,m}]}{1-\rho_{c,m}(s_{c,m})}, \tag{1}$$

where $\rho_{c,m}(s_{c,m}) = \frac{\lambda^{c,m}}{\mu_{c,m} \cdot s_{c,m}}$. $F(z^{c,m})$ is the probability distribution function of service time, and $E[Z^{c,m}]$ is the expectation of service time. Hence, the following constraint must be satisfied, to keep the traffic intensity $\rho_{c,m}(s_{c,m})$ below 1.

$$s_{c,m} \geq s^b_{c,m} > \frac{\lambda^{c,m}}{\mu_{c,m}}.$$

2) Response time of a chain: In a microservice chain, as shown in Fig. 8, requests traverse a set of microservices. Based on Equation (1), we then prove the following theorem to evaluate the average response time of a chain.

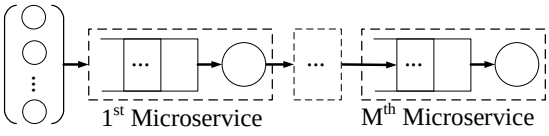

Fig. 8. A queue-based model for a microservice chain.

Theorem 1. *Suppose chain c passes M^c microservices, $\forall m \in \{1, M^c-1\}$, the request departs from microservice m at a rate of $\lambda_{c,m}$ and arrives at microservice $m+1$ at a rate of $\lambda_{c,m+1}$. Hence, the request arrival process of microservice $m+1$ is a Poisson process, and $\lambda_{c,m+1} = \lambda_{c,m}$ when the microservice system is stable.*

Proof: Let $\mathcal{Z}$ be the set of the user requests who depart from the microservice m during $(z, z+t)$. Hence, at the y^{th} time slot, the probability that a request belongs to set $\mathcal{Z}$ arrives at microservice m is denoted as:

$$F(y) = \begin{cases} Q(z+t-y) - Q(z-y), & y < z \\ Q(z+t-y), & z < y < z+t \\ 0, & y > z+t \end{cases}$$

where $Q(x)$ is the probability distribution function of the duration spent on serving a request.

Since requests that arrive at the same time slot in an M/G/1/PS queue share the capacity of microservice instances, the output of an M/G/1/PS queue is still a Poisson process. Let $E[N(t)]$ be the expected number of the requests which departs from microservice m to $m+1$ at t^{th} time slot. Inspired by [14], we hence calculate $E[N(t)]$ as follows.

$$\begin{aligned} E[N(t)] &= \lambda_{c,m} \cdot \int_0^\infty F(y)dy \\ &= \lambda_{c,m} \cdot \int_0^s (Q(z+t-y) - Q(z-y))dy \\ &\quad + \lambda_{c,m} \cdot \int_z^{z+t} Q(z+t-y)dy \\ &= \lambda_{c,m} \cdot \int_z^{z+t} Q(y)dy. \end{aligned}$$

As a result, requests which depart from microservice m follow a Poisson process, with an expectation of $\lambda_{c,m} \cdot \int_z^{z+t} Q(y)dy$. Hence, we have:

$$P_n(t) = e^{-\lambda_{c,m} \cdot \int_z^{z+t} Q(y)dy} \cdot \frac{\left(\lambda_{c,m} \cdot \int_z^{z+t} Q(y)dy\right)^n}{n!} \tag{2}$$

According to the definition of Poisson process, we have:

$$\begin{aligned} &P_n(t+h) = P\{N(t+h) = n\} \\ &= P\{N(t) = n-1, N(t+h) - N(t) = 1\} \\ &\quad + P\{N(t+h) = n, N(t+h) - N(t) \geq 2\}, \end{aligned}$$

where $h > 0$. Given $P\{N(h) = 1\} = \lambda_{c,m+1}h + o(h)$, and $P\{N(h) \geq 2\} = o(h)$, we hence have:

$$P_n(t+h) = (1 - \lambda_{c,m+1}h)P_n(t) + \lambda_{c,m+1}hP_{n-1}(t) + o(h),$$

where $\lambda_{c,m+1}$ is the arrival rate of requests from microservice m to $m+1$.

Dividing both sides of the equation above by h leads to:

$$\frac{P_n(t+h) - P_n(t)}{h} = -\lambda_{c,m+1}P_n(t) + \lambda_{c,m+1}P_{n-1}(t) + \frac{o(h)}{h}$$

When $h \to 0$, it derives:

$$P_n^{'}(t) = -\lambda_{c,m+1} \cdot P_n(t) + \lambda_{c,m+1} \cdot P_{n-1}(t). \tag{3}$$

Applying Equation (2) to the left side of Equation (3), we have:

$$\lambda_{c,m+1} = \lambda_{c,m} \cdot Q(t).$$

When $t \to \infty$, i.e., the microservice system approaches stable, $Q(t) \to 1$. We hence prove that $\lambda_{c,m+1} = \lambda_{c,m}$. ■

Based on Theorem 1, it is reasonable to compute the average response time by summing up that of each microservice. Hence, we have the following proposition.

Proposition 1. *In chain c, the average response time u_c can be calculated as:*

$$u_c(\boldsymbol{s}_c) = \sum_{m=1}^{M} r_{c,m} \cdot u_{c,m}(s_{c,m}),$$

where $r_{c,m}$ indicates whether chain c passes microservice m.

Different from the instance-oriented and microservice-oriented load balancing solutions, our solution is dedicated to balancing load at the chain level. Considering that chains

have unique requirements of QoS as well as different capacity of serving their requests, we introduce a predefined upper bound U_c^b for response time of each chain. Since our solution employs pure message queue to enable communications across microservices, the QoS of each chain should be carefully guaranteed. To this end, the average response time should be controlled within this predefined deadline. To bound and further minimize the average response time of each chain, we formulate an objective function as follows.

$$\max_{u^c} \quad \prod_{u^c \in U} \frac{U_c^b - u_c}{U_c^b}, \tag{4}$$

where $U_c^b = u_c(\boldsymbol{s}_c^b)$, and $\boldsymbol{s}_c^b$ is the initial instance assignment of each chain.

From the perspective of chains, each chain competes for microservice instances against other chains. Thus an instance allocation algorithm is required to coordinate the resources across chains. Game theory is known as a promising approach to solve such resource competition among agents.

IV. A Nash Bargaining Based Solution

Let $\mathcal{R}^N$ be the set of all the available instance allocation strategies. Define G as the subset of allocation strategies in which response times of each chain is within its deadline, denoted as $G = \{\boldsymbol{s}_c | \boldsymbol{s}_c \in S, u_c(\boldsymbol{s}_c) \leq u_c(\boldsymbol{s}_c^b) = U_c^b, c \in \{1, ..., C\}\}$. Each chain in the system uses an initial strategy $\boldsymbol{s}_c^b$ to bargain with others, and finally the chains reach agreement upon an allocation strategy in G.

Definition 1. *A mapping $\mathcal{M} : (G, \boldsymbol{s}_c^b) \rightarrow \mathcal{R}^N$ is a Nash bargaining solution if it satisfies $\mathcal{M} : (G, \boldsymbol{s}_c^b) \in G$, Pareto optimality, symmetry, invariant to affine transformations, and independent of irrelevant alternatives [15].*

Let $J = \{G = \{\boldsymbol{s}_c | \boldsymbol{s}_c \in G, u_c(\boldsymbol{s}_c) < u_c(\boldsymbol{s}_c^b) = U_c^b, c \in \{1, ..., C\}\}$, so J is nonempty. Given that S is a convex and compact subset of $\mathcal{R}^N$, we have Theorem 2 as follows.

Theorem 2. *There exists a Nash bargaining solution and the elements of the vector $\boldsymbol{s}_c = \mathcal{M}(G, u_0)$ solve the following optimization problem:*

$$\max_{\boldsymbol{s}_c} \quad \prod_{\boldsymbol{s}_c \in J} \frac{U_c^b - u_c}{U_c^b}. \tag{5}$$

Theorem 2 implies that there exists a Nash bargaining solution to the instance allocation problem, which minimizes and bounds average response time of each microservice chain. Note that the objective function (5) is equivalent to the function $\max_{\boldsymbol{s}_c} \ln(U_c^b - u_c), \forall \boldsymbol{s}_c \in J$. Hence we can formulate the optimization problem as follows:

$$\max_{\boldsymbol{s}_c} \quad \sum_{c=1}^{C} \ln \frac{U_c^b - u_c}{U_c^b} \tag{6}$$

$$\text{s.t.} \quad \sum_{c=1}^{C} r_{c,m} \cdot s_{c,m} \leq I^m, \quad m \in \{1, ..., M\} \tag{7}$$

$$s_{c,m} > \frac{\lambda^{c,m}}{\mu_{c,m}}, \quad m \in \{1, ..., M\} \tag{8}$$

The problem above is a nonlinear integer program, where nonlinearity is confined to the objective function. It is proven that such a problem is NP-hard in general [16]. Given that user requests are varying and unpredictable, the problem should be solved efficiently so as to obtain a latest instance allocation strategy timely. By relaxing integrality constraints, the original problem is reduced to convex optimization. Based on the optimal solution to the relaxed version, we leverage a rounding strategy to compute the solution to problem (6), with integrality gap analyzed.

Theorem 3. *There exist $\delta_m > 0, m \in \{1, ..., M\}$ such that*

$$\hat{s}_{c,m} > -\frac{A_2}{2A_1}, \tag{9}$$

where $A_1 = U_c^b \mu_{c,m}^2 + E[Z_{c,m}]\mu_{c,m}^2$, $A_2 = 2u_{c,m}^b \lambda_{c,m}\mu_{c,m} + E[Z_{c,m}]\lambda_{c,m}\mu_{c,m}$.

Proof: Because Constraints (7) and (8) are linear and the objective function (6) is differentiable, the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality [17].

Define the Lagrangian function $\mathcal{L}(s, \delta, \eta)$ as follows.

$$\mathcal{L}(s, \delta, \eta) = f(s) - \sum_{m=1}^{M} \delta_m (s_{c,m} - \frac{\lambda^{c,m}}{\mu_{c,m}}) - \sum_{m=1}^{M} \eta_m (\sum_{c=1}^{C} r_{c,m} \cdot s_{c,m} - I^m),$$

where $\delta_m \leq 0 (m = 1, ..., M)$ and $\eta_m \leq 0 (m = 1, ..., M)$ are the Lagrange multipliers.

We hence derive the first-order necessary and sufficient conditions as follows.

$$\nabla_s \mathcal{L}(s, \delta, \eta) = \frac{\partial f(s_{c,m})}{\partial s_{c,m}} - \delta_m - \sum_{m=1}^{M} \eta_m \cdot r_{c,m} = 0, \tag{10}$$

$$(\sum_{c=1}^{C} r_{c,m} \cdot s_{c,m} - I^m) \cdot \delta_m = 0; \quad \delta_m \geq 0; m \in \{1, ..., M\} \tag{11}$$

$$(s_{c,m} - \frac{\lambda^{c,m}}{\mu_{c,m}}) \cdot \eta_m = 0; \quad \eta_m \geq 0; m \in \{1, ..., M\} \tag{12}$$

Based on Constraint (8), we have $\eta_m \equiv 0$.

If $\delta_m = 0$, we have $\nabla_s \mathcal{L}(s, \delta, \eta) = 0$, which contradicts the fact that $f(\cdot)$ is increasing.

Hence, we have $\delta_m > 0$, indicating that $\sum_{c=1}^{C} r_{c,m} \cdot s_{c,m} - I^m = 0$, which ensures that load is balanced across all the instances. Based on Equation (10), we have:

$$\nabla_s \mathcal{L}(s, \delta, \eta) = \frac{\partial f(s_{c,m})}{\partial s_{c,m}} - \delta_m = 0.$$

Applying Equation (6) to the above equation, we have:

$$A_1 \cdot s_{c,m}^2 - A_2 \cdot s_{c,m} + A_3 = 0 \tag{13}$$

where $A_1 = U_c^b \mu_{c,m}^2 + E[Z_{c,m}]\mu_{c,m}^2$, $A_2 = 2u_{c,m}^b \lambda_{c,m}\mu_{c,m} + E[Z_{c,m}]\lambda_{c,m}\mu_{c,m}$, and $A_3 = U_c^b \lambda_{c,m}^2 - \frac{E[Z_{c,m}]\mu_{c,m}\lambda_{c,m}}{\delta_m}$.

As a result, $\exists \delta_m > 0$, such that Equation (13) has at least one solution $\hat{s}_{c,m} > -\frac{A_2}{2A_1}$, which is the unique Nash bargaining solution to the optimization problem. ■

By employing convex optimization techniques, we obtain the optimal solution $\hat{s}_{c,m}, c \in \{1, ..., C\}, m \in \{1, ..., M\}$ to the relaxed version of problem (5). However, $\hat{s}_{c,m}$ is fractional, and is hence infeasible for the original problem. We apply a simple rounding up (factional part > 0.5) and down (otherwise) strategy. We then have the following theorem, which quantifies the integrality gap between the optimal value of problem (6) and the value of problem (6).

Theorem 4. *Let ϕ be the value of problem (6) under relaxation with rounding and ϕ^* be the optimal value of problem (6), respectively. We hence are able to bound the value of $\tilde{\phi}$ as follows.*

$$\phi > \phi^* - \sum_{c=1}^{C} \ln(1 + M \cdot Q_c)$$

where $Q_c = \frac{1}{U_c^b - \sum_{m=1}^{M} u_c(\hat{s}_{c,m}+1)}$ and M is the total number of microservices.

Proof: Let $\tilde{\boldsymbol{S}} = (\tilde{s}_{c,m})_{C\times M}$ be the optimal solution to problem (5) under relaxation, and let $\tilde{\phi}$ be the optimal value of the problem. Moreover, let $\boldsymbol{S}^* = (s_{c,m}^*)_{C\times M}$ denote the optimal solution to problem (5), whose value is defined as ϕ^*. Hence, we have:

$$\tilde{\phi} > \phi^*. \tag{14}$$

Then, under the rounding strategy of $\hat{s}_{c,m} = \lfloor \tilde{s}_{c,m} \rfloor$, $\hat{\boldsymbol{S}} = (\hat{s}_{c,m})_{C\times M}$ is a solution to problem (5), whose objective function value is $\hat{\phi}$. Therefore, $\exists \theta_{c,m} \in \{0, 1\} (c \in \{1, ..., C\}, m \in \{1, ..., M\})$, such that

$$\hat{s}_{c,m} = \tilde{s}_{c,m} + \theta_{c,m}. \tag{15}$$

Since $\hat{s}_{c,m}$ is larger than $\tilde{s}_{c,m}$, we have:

$$\phi > \hat{\phi}, \tag{16}$$

where ϕ is the optimal value of problem (6) after rounding.

Based on Eq. (14), Eq. (15), and (16), we have:

$$\begin{aligned}
&\phi^* - \phi < \tilde{\phi} - \hat{\phi} \\
&= \sum_{c=1}^{C} \ln \left(\frac{U_c^b - u_c(\hat{\boldsymbol{s}}_c)}{U_c^b} - \ln \frac{U_c^b - u_c(\tilde{\boldsymbol{s}}_c)}{U_c^b} \right) \\
&= \sum_{c=1}^{C} \ln \frac{U_c^b - \sum_{m=1}^{M} u_c(\hat{s}_{c,m})}{U_c^b - \sum_{m=1}^{M} u_c(\hat{s}_{c,m} + \theta_{c,m})} \\
&= \sum_{c=1}^{C} \ln \left(1 + \frac{\sum_{m=1}^{M} u_c(\hat{s}_{c,m} + \theta_{c,m}) - \sum_{m=1}^{M} u_c(\hat{s}_{c,m})}{U_c^b - \sum_{m=1}^{M} u_c(\hat{s}_{c,m} + \theta_{c,m})} \right) \\
&< \sum_{c=1}^{C} \ln \left(1 + Q_c \sum_{m=1}^{M} u_c(\hat{s}_{c,m} + \theta_{c,m}) - u_c(\hat{s}_{c,m}) \right) \\
&= \sum_{c=1}^{C} \ln \left(1 + Q_c \cdot \sum_{m=1}^{M} (1 - \Phi) \right)
\end{aligned}$$

where $Q_c = \frac{1}{U_c^b - \sum_{m=1}^{M} u_c(\hat{s}_{c,m}+1)}$, and $\Phi = \frac{\theta_{c,m}\lambda_{c,m}}{\mu_{c,m}\hat{s}_{c,m}^2 + (\mu_{c,m}\theta_{c,m} - \lambda_{c,m})\hat{s}_{c,m}}$

Noting that $\Phi \geq 0$, we have:

$$\phi^* - \phi < \sum_{c=1}^{C} \ln(1 + M \cdot Q_c),$$

where M is the total number of microservice types. ■

V. Performance Evaluation

A. Simulation Setup

We simulate a microservice system with $M = 6$ microservices, 600 instances in total, serving $C = 7$ chains.

1) Workload data: In the simulations, we use the online traffic activity of the world's top e-retailers in the U.S. on Cyber Monday collected by Akamai [18]. Data were reported by Akamai's Net Usage Index and Real User Monitoring, which is plotted in Fig. 9. For requests of chains, we randomly determine each chain's share of the overall traffic, to examine whether COLBA adapts well to fluctuating workload.

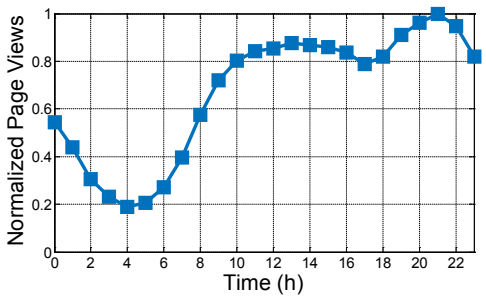

Fig. 9. Normalized online traffic of ecommerce retailers in the U.S. on Cyber Monday.

2) Baselines: To evaluate the performance of COLBA, we compare it with two baselines: microservice-oriented and instance-oriented solutions.

Instance-oriented load balancing. Recall that instance-oriented solution is promising in handling request heterogeneity and competition, since it balances load across instances with a central load balancer. For fair comparison, we apply the chain-oriented load balancing strategy to it.

Microservice-oriented load balancing. Compared with instance-oriented solution, microservice-oriented solution evenly balances requests across instances of a microservice, ignoring request heterogeneity and competition.

3) Worst response time $\boldsymbol{U}^b$: In Sec. III, to bound response time of a chain, we introduced $U_c^b (c \in \{1, ..., C\})$, representing the worst response time of requests in chain c, i.e., the upper bound of response times. U_C^b can also be interpreted as the performance requirement of user requests in chain c.

B. Comparing to Baselines

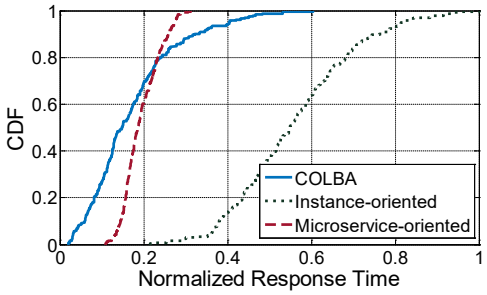

Fig. 10. CDF of response time, COLBA *vs.* baselines.

Overall performance. We take the average response time of all requests in the microservice system as a metric to demonstrate the balancing performance of the three solutions. Fig. 10 plots the CDF of overall response time of COLBA, compared with baselines. In Fig. 10, we observe that among the three solutions, performance of the instance-oriented solution is the worst due to significant communication overhead. Furthermore, 80% of response times are almost the same for microservice-oriented solution and COLBA. However, compared with COLBA, the response time of the microservice-oriented solution spans a smaller range. It is because that COLBA balances load jointly concerning the worst response time U_c^b and competition across chains, meaning that COLBA prefers allocating resources to urgent chains and compromising with others.

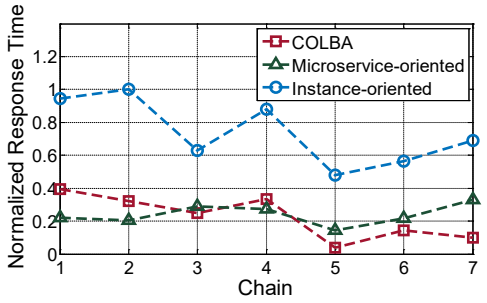

Fig. 11. The normalized response time of COLBA against instance-oriented and microservice-oriented solutions.

Individual performance of chains. Fig. 11 depicts the respective normalized response time of chains under the worst response time configuration of $[10, 8, 6, 7, 2, 4, 2]$. Under the microservice-oriented solution, performance of each chain is almost the same. In comparison, under COLBA, the response time of chains are different from each other and are in accordance with the worst response time configuration. Furthermore, a chain with a lower U_c^b has better performance, reflecting that COLBA can guarantee QoS based on U_c^b of a chain, and coordinate instance allocation across chains.

C. Convergence of COLBA

Fig. 12 plots the CDF of the number of iterations to achieve convergence. It shows that our algorithm COLBA can converge within 79 iterations in 80% of the respective total runs. Furthermore, the minimum number of iterations is 16 while the maximum is 109. Such results demonstrate that our solution can achieve fast convergence, making it possible to adjust the instance allocation strategy promptly to adapt unpredictable workload.

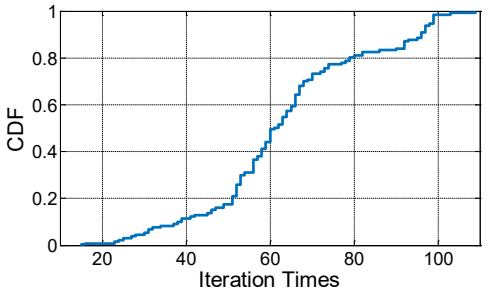

Fig. 12. The iterations of COLBA.

D. Impact of Parameters

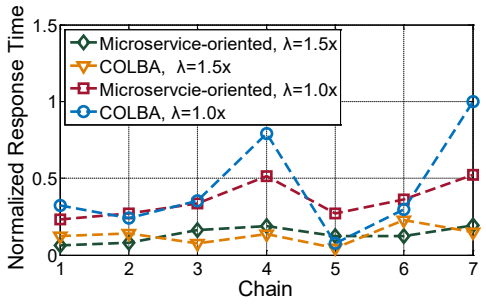

Fig. 13. Normalized response time of chains in the microservice system.

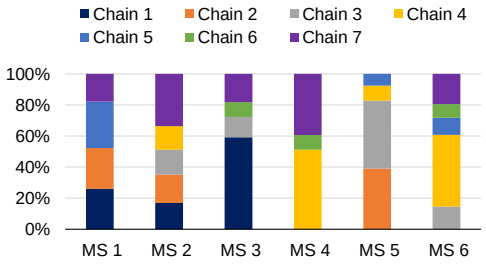

Fig. 14. The instance allocation across chains under request arrival rate of λ_{max}.

Request arrival rate. To further evaluate the performance of COLBA, we tune the maximum request arrival rate λ from 1.0x to 1.5x of the maximum page view in Fig. 9. Fig. 13 illustrates the normalized response time of chains when λ increases from 1.0x to 1.5x. In Fig. 13, as λ increases, the performance of both microservice-oriented and COLBA degrades. Specifically, response times of chains under the microservice-oriented solution roughly have similar increments. In comparison, performance of Chains 5 and 6 almost remains unchanged

under COLBA. However, response time of Chains 4 and 7 increases substantially.

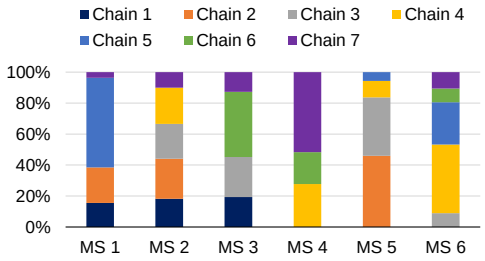

Fig. 15. The instance allocation across chains under request arrival rate of $1.5\lambda_{max}$.

To further evaluate the impact of request arrival rate, we plot instance allocation of COLBA under two values of 1.0x and 1.5x in Fig. 14 and 15, respectively. We make the following observations. For Chain 5 and 6, the instances allocated to them increase significantly so as to overcome the bursty workload, finally guaranteeing response time under the worst value U_c^b. Obviously, since the scale of instances is limited, other chains who compete against Chains 5 and 6 have to compromise on microservice instances. As such, we then observe that number of instances assigned to Chains 1, 4 and 7 decreases sharply, leading to increase in response time. Such phenomenon indicates that COLBA can exploit the heterogeneity of performance requirements across chains, therefore better adapt to fluctuating workload and meet users' expected QoS.

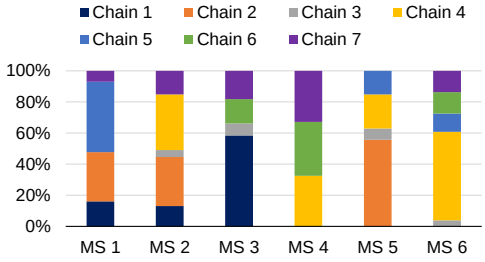

Fig. 16. The instance allocation of COLBA when $\boldsymbol{U}^b = [3:3:2:5:4:2:9]$.

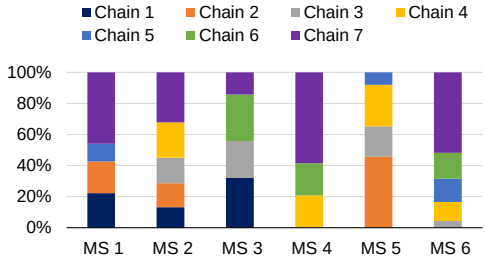

Fig. 17. The instance allocation of COLBA when $\boldsymbol{U}^b = [3:3:2:9:4:2:5]$.

Worst response time $\boldsymbol{U}^b$. We evaluate the response time of COLBA under different configuration of $\boldsymbol{U}^b$, with results depicted in Fig. 18. By switching the worst response time between Chains 4 and 7, we change $\boldsymbol{U}^b$ from $[3:3:2:5:4:2:9]$ to $[3:3:2:9:4:2:5]$. In Fig. 18, we can observe that response time of requests in Chain 7 decreases while those in Chain 4 increase, because Chains 4 and 7 exchange the worst response time with each other. Furthermore, performance of Chains 3 and 6 remains stable, yet response time of Chains 1, 2 and 5 increases. The reason behind such increase can be found in Fig. 16 and 17.

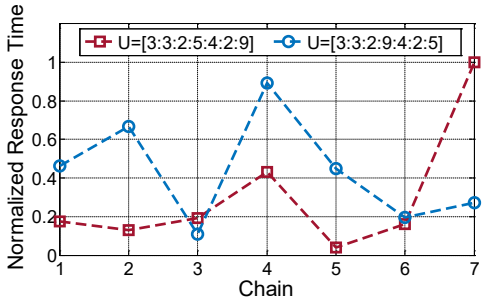

Fig. 18. The normalized response time of COLBA under different values of U^b.

Fig. 16 and 17 plot the instance allocation of COLBA under different $\boldsymbol{U}^b$. As we observe, the scale of instances allocated to Chain 7 increases remarkably, especially for Microservice 1 and 6, ensuring response time within $\boldsymbol{U}^b$. In the meantime, instance scale of Microservice 6 assigned to Chain 4 shrinks significantly, incurring increase in response time, meaning that $\boldsymbol{U}^b$ successfully changes the preference of instance allocation.

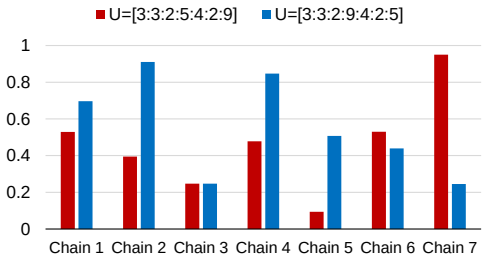

Fig. 19. The respect ratio of response time to upper bound u_c^b across chains under different $\boldsymbol{U}^b$.

Apart from Chain 4, response time of Chains 1, 2, and 5 also goes down, due to compromising on instances with others. To investigate whether response time of Chains 1, 2, and 5 stays within the upper bound $\boldsymbol{U}^b$, we plot Fig. 19 to demonstrate the respect ratio of response time to upper bound u_c^b across chains under different $\boldsymbol{U}^b$. As we observe, response time in Chains 1, 2, and 5 goes up, yet without exceeding the upper bound $\boldsymbol{U}^b$.

VI. Related Work

In this section, we briefly review existing work on cloud containers and microservices.

Cloud containers. Recently, container technology, which provisions lightweight isolation, has drawn extensive attention

from researchers. Burns *et al.* explicitly discuss three typical design patterns for container-based systems, i.e., single-container pattern, single-node pattern, and multi-node pattern [19]. In [20], three container management systems, i.e., Borg [21], Omega [22], and Kubernetes [23] are carefully compared based on lessons learned by Google when managing Linux containers. Yi *et al.* propose a group buying mechanism to improve cloud resource utilization on cloud containers [24]. Moreover, motivated by the lightweight and flexible characteristics of containers, the IT industry shows a growing interest in deploying network functions in containers. Zhang *et al.* design an NFV platform which enables every single NF to process a specific flow by running VNFs in containers [25]. Yu *et al.* propose FreeFlow, which leverages shared memory and RDMA to improve performance and portability of container networking [26]. Jointly optimizing the communication overhead and overall throughput, Zhang *et al.* propose to re-distribute containers across hosts, to alleviate the communication overhead in container system in [27]. Different from and compliment to the above studies, COLBA focuses on enterprise applications, e.g., web applications, which are deployed in containers.

Microservices. With the popularity of container technology growing, many applications are developed in a microservice fashion, facilitating frequent update and deployment. As a result, testing and troubleshooting become critical issues to address. With respect of testing, Heorhiadi *et al.* propose a framework for systematically testing the failure-handling capabilities of microservices [28]. In terms of troubleshooting, Rajagopalan *et al.* propose an autonomous tool to troubleshoot and repair such software issues in production environments [29].

Different from the above works, our solution focuses on the load balancing problem across microservices. To the best of our knowledge, little literature is dedicated to handling such a problem in the context of microservice systems.

VII. Conclusion

This work proposes COLBA to balance load across microservices, jointly taking heterogeneity of requests and inter-chain competition into consideration. By employing convex optimization with rounding, COLBA efficiently solves the chain-oriented load balancing problem which is NP-hard. Theoretical analysis proves that COLBA can promptly converge to the unique Nash solution which is within a constant gap of the optimum. Moreover, statistics of iterations times indicate that COLBA can converge within 79 iterations in 80% of the respective total runs, facilitating adapting unpredictable workload. Trace-driven simulation shows that COLBA can reduce average response time by about 13% and 41% compared with instance-oriented and microservice-oriented load balancing solutions, respectively. In addition to minimizing response time, simulation results demonstrate that COLBA can coordinate instance allocation, concerning expected QoS of chains as well as competition across chains.

References

[1] What Led Amazon to its Own Microservices Architecture. [Online]. Available: https://thenewstack.io/led-amazon-microservices-architecture/

[2] Adopting Microservices at Netflix: Lessons for Architectural Design. [Online]. Available: https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

[3] Service-Oriented Architecture: Scaling the UBER Engineering Codebase As We Grow. [Online]. Available: http://zookeeper.apache.org/

[4] Designing and Deploying Microservices. [Online]. Available: https://www.nginx.com/resources/library/designing-deploying-microservices/

[5] E. Wolff, *Microservices: Flexible Software Architectures*. CreateSpace Independent Publishing Platform, 2016.

[6] Apache Zookeeper. [Online]. Available: http://zookeeper.apache.org/

[7] Y. Jiang, L. R. Sivalingam, S. Nath, and R. Govindan, "Webperf: Evaluating what-if scenarios for cloud-hosted web applications," in *Proc. of SIGCOMM*, 2016.

[8] Asynchronous Programming with Async and Await. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh191443.aspx

[9] RabbiMQ. [Online]. Available: http://www.rabbitmq.com/

[10] A. Kamra, V. Misra, and E. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites," in *Proc. of IWQOS*, 2004.

[11] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance management for cluster-based web services," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 12, pp. 2333–2343, Dec 2005.

[12] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Internet Technol.*, vol. 7, no. 1, Feb. 2007.

[13] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[14] Y. Niu, F. Liu, X. Fei, and B. Li, "Handling flash deals with soft guarantee in hybrid cloud," in *Proc. of INFOCOM*, May 2017.

[15] J. Nash, "Non-cooperative games," *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[16] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel, *Nonlinear Integer Programming*. Springer Berlin Heidelberg, 2010, pp. 561–618.

[17] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[18] "Akamai's 2014 online holiday shopping trends and traffic report," Report, 2016 Akamai Technologies, Inc., 2014. [Online]. Available: https://content.akamai.com/PG2112-Holiday-Recap-Report.html

[19] B. Burns and D. Oppenheimer, "Design patterns for container-based distributed systems," in *Proc. of HotCloud*, 2016.

[20] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, Apr. 2016. [Online]. Available: http://doi.acm.org/10.1145/2890784

[21] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. of EuroSys*, 2015.

[22] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proc. of EuroSys*, 2013.

[23] Kubernetes. [Online]. Available: https://kubernetes.io/

[24] X. Yi, F. Liu, D. Niu, H. Jin, and J. C. S. Lui, "Cocoa: Dynamic container-based group buying strategies for cloud computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 2, no. 2, pp. 8:1–8:31, Feb. 2017.

[25] W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, and T. Wood, "Flurries: Countless fine-grained nfs for flexible per-flow customization," in *Proc. of CoNext*, 2016.

[26] T. Yu, S. A. Noghabi, S. Raindel, H. Liu, J. Padhye, and V. Sekar, "Freeflow: High performance container networking," in *Proc. of HotNet*, 2016.

[27] Y. Zhang, Y. Li, K. Xu, D. Wang, M. Li, X. Cao, and Q. Liang, "A communication-aware container re-distribution approach for high performance vnfs," in *In Proc. of ICDCS*, 2017.

[28] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar, "Gremlin: Systematic resilience testing of microservices," in *Proc. of ICDCS*, June 2016.

[29] S. Rajagopalan and H. Jamjoom, "App–bisect: Autonomous healing for microservice-based apps," in *Proc. of HotCloud*. USENIX Association, 2015.