Data-Driven Edge Resource Provisioning for Inter-Dependent Microservices with Dynamic Load

Ruozhou Yu, Szu-Yu Lo, Fangtong Zhou, Guoliang Xue

***Abstract*—This paper studies how to provision edge computing and network resources for complex microservice-based applications (MSAs) in face of uncertain and dynamic geo-distributed demands. The complex inter-dependencies between distributed microservice components make load balancing for MSAs extremely challenging, and the dynamic geo-distributed demands exacerbate load imbalance and consequently congestion and performance loss. In this paper, we develop an edge resource provisioning model that accurately captures the inter-dependencies between microservices and their impact on load balancing across both computation and communication resources. We also propose a robust formulation that employs explicit risk estimation and optimization to hedge against potential worst-case load fluctuations, with controlled robustness-resource trade-off. Utilizing a data-driven approach, we provide a solution that provides risk estimation with measurement data of past load geo-distributions. Simulations with real-world datasets have validated that our solution provides the important robustness crucially needed in MSAs, and performs superiorly compared to baselines that neglect either network or inter-dependency constraints.**

***Keywords*—Edge computing, microservice, load balancing, resource provisioning, robustness, data-driven**

I. Introduction

Modern digital applications have grown significantly in scale and complexity. Consequently, the traditional monolithic architecture is no longer suitable for modern applications that require big data processing, rapid development, and dynamic deployment. By breaking a monolithic application into loosely coupled microservices, an application can achieve scalable and flexible distributed processing of external data with high performance and reliability. Furthermore, microservices can prospectively be combined with *edge computing* to enable low-latency, high-throughput and reliable computing that satisfies the stringent performance requirements of modern applications, including but not limited to autonomous vehicles, social media, smart city, smart healthcare, etc. Due to their great potential, microservices have received significant attention from both academia and industry in recent years.

In a microservice-based application (MSA), a user request is processed by a set of inter-dependent microservices. More precisely, each microservice may need to issue API calls to other microservices when processing an in-coming user request. When microservices are deployed distributedly, processing of a request will involve both computation by microservice instances on distributed nodes, and communications between these instances for data transfer. A critical challenge in microservice management is thus to disentangle and accurately model these inter-dependencies, to satisfy various performance goals of application owners and/or edge computing provider.

Yu, Lo and Zhou ({ryu5, slo4, fzhou}@ncsu.edu) are with North Carolina State University, Raleigh, NC 27606, USA. Xue (xue@asu.edu) is with Arizona State University, Tempe, AZ 85287, USA. This research was supported in part by NSF grants 1704092, 2007391, 2007469 and 2045539. The information reported here does not reflect the position or the policy of the funding agency.

This paper studies microservice management in the view of edge resource provisioning and microservice load balancing. To ensure high performance of an MSA, edge provider needs to jointly provision computing and network resources, and guide the MSA's load balancing behavior, to minimize congestion when processing user requests. This is non-trivial due to microservice inter-dependencies, which will lead to imbalanced load if naive load balancing is employed [1], [2]. Additionally, real-world demands are both dynamic and time-varying. Static load balancing as in existing work [2] could result in high congestion if demand dynamics are not tackled proactively.

To address these challenges, we first propose an accurate model that formulates microservice load balancing as a linear program (LP) with static inputs, which captures the impact of inter-dependencies on resource sharing between co-located microservice instances, as well as between inter-microservice communications on shared network links. The ability to capture both computing resource and network bandwidth sharing is lacking in existing work [2]. Then, we further propose a novel robust formulation for proactive risk estimation and optimization against the dynamic demands, based on *conditional value-at-risk (CVaR)* which is a popular risk management tool in economics and finance. To address the robust resource provisioning problem, we propose a data-driven approach for approximating CVaR with historical demand data for the target MSA, with convergence guarantee when the amount of available data approaches infinity. To evaluate its performance, we conducted simulations based on profiling of a real-world MSA and real-world demand traces. Results have shown the superior performance of our robustness solution when facing adverse demand fluctuations, and when compared to two baselines that neglect certain network- or inter-dependency-constraints. To summarize, our main contributions include:

- We propose a computing-network load balancing model for inter-dependent microservices in a general edge network, filling a gap in existing network-oblivious solutions.
- We propose a robust edge resource provisioning formulation based on the load balancing formulation, which explicitly hedges against worst-case demand fluctuations with proactive resource provisioning. The formulation enables flexible trade-off between solution robustness and resource consumption controlled by the edge provider.
- We present extensive simulation results with real-world datasets to show the superior performance of our solution.

The rest of this paper is organized as follows. Sec. II discusses related work. Sec. III describes our system model and the basic MSA load balancing formulation. Sec. IV explains the concept of CVaR and details our robust formulation. Sec. V shows evaluation results. Sec. VI concludes this paper.

II. Background and Related Work

Microservices. Microservices are widely used in modern digital applications, ranging from large-scale, cloud-based applica-

tions such as Amazon AWS, Netflix and Twitter [3] to internet-of-things, smart city, or connected industry scenarios [4], [5]. The cloud container technology is behind the migration from a monolithic to a microservice-based application architecture by many companies [6], and substantial work has been done on microservice management in the cloud [1], [7], [8]. Recently, the combination of edge computing and microservices has been proposed, showing strong support for performance-stringent applications [2], [9]–[11]. Among these efforts, several have recognized the complexity brought by microservice inter-dependencies and its impact on resource management [2], [8], [10]. A majority of such efforts, though, neglect networking as an important factor affecting the performance of microservices.

Edge computing. Edge resource management is a proliferating area due to the limited and expensive edge resources, with many efforts addressing mobile offloading [12], [13], resource allocation [14], resource provisioning [15], service deployment [9], edge caching [16], task scheduling and dispatching [17], [18], etc. Among these, [2] employs a model similar to ours when addressing load balancing for microservices with complex inter-dependencies. However, [2] considers neither network load balancing (it only focuses on load balancing across computing nodes), nor demand dynamics in the edge network, both being unique and critical aspects of edge computing. This motivates us to develop a joint computing-network resource provisioning solution for microservice load balancing, and explicitly incorporate robustness into the solution.

Robustness. Both *proactive* and *reactive* mechanisms have been proposed for robust system design, with the former offering better robustness than the latter at the cost of increased resource redundancy. As edge applications are performance-oriented, proactive mechanisms ensure that applications incur minimal performance degradation due to adverse events such as outages or demand fluctuations. *Robust optimization* is a common technique for optimizing system performance under *the worst possible scenario*; unfortunately, it has been criticized to be too *conservative*, resulting in significant resource cost and wastage [15]. This paper employs CVaR, a risk management tool that is naturally defined upon the trade-off between solution conservativeness and overhead [19]. The same concept has been predominantly used in risk analysis in economics and finance [19], but has additionally found application in power grid optimization [20] and security management [21], and most recently, in network traffic engineering [22], traffic offloading [23] and edge resource management [15].

III. System Model and Problem Statement

A. Network and Application Model

The edge network is modeled as a directed graph $G = (N, L)$, where N is the set of nodes including access points (APs) $A \subseteq N$ and edge computing nodes (hosts) $H \subseteq N$, and L is the set of inter-connecting links. Each edge node $h \in H$ has computing capacity $c_h > 0$, and each link $l \in L$ has bandwidth capacity $b_l > 0$. We use $L_{\mathsf{in}}(n), L_{\mathsf{out}}(n) \subseteq L$ to denote the sets of in-coming and out-going links of node $n \in N$ respectively.

An MSA is also modeled as a directed graph $\Gamma = (V, E)$, where V is the set of microservices, and E denotes the processing inter-dependencies (*i.e.*, API calls) between microservices — an edge $(u, v) \in E$ means microservice u may issue API calls to microservice v for processing user requests. The MSA serves multiple types of requests from its users, which enter the edge network through the APs in A. For instance, a ride sharing application needs to process passenger ride requests, driver updates, passenger updates, trip completion, etc. Each type of request is processed by a different set of microservices via different processing paths. Specifically, a request is first processed at an entrance microservice (*e.g.*, a load balancer or NGINX web server), which then issues API calls to successor microservices for processing; each successor may further issue calls to its own successors. Formally, assume the application has K types of requests, and let $[K] = \{1, 2, \ldots, K\}$. For a request type $k \in [K]$, the processing inter-dependencies between microservices (API calls) are represented by a *directed acyclic graph (DAG)* $\Gamma_k = (V_k, E_k)$ — a subgraph of Γ.

Microservices are deployed as distributed instances in the edge network. Hence an API call involves 1) forward transmission of call parameters (*e.g.*, request data and meta-data) from predecessor to successor, 2) processing at successor including further API calls, and 3) backward transmission of results to predecessor. Let $H_v \subseteq H$ be the set of nodes deploying microservice v, and $V_h \subseteq V$ be the set of microservices with instances on h. To characterize the communication and computing loads incurred by processing a request, we define $\rho^k_{e,1}, \rho^k_{e,2} \geq 0$ as the *forward* and *backward communication load ratios* respectively on inter-dependency edge $e = (u, v) \in E_k$, and $\rho^k_v \geq 0$ as the *computation load ratio* incurred on microservice v, for request type k. In practice, both the subgraphs $\{\Gamma_k\}$ and the load ratios can be obtained by profiling the application with sample workloads in a controlled environment [24], [25].

B. Resource Provisioning Problem Statement

For load balancing, an MSA utilizes distributed microservice instances deployed across the edge network. A basic provisioning problem in this scenario is to provision sufficient computing resources on edge hosts and bandwidth in the network, and properly assign computation and communication loads to microservice instances and network routes respectively, to ensure that data requests from all APs are served with sufficient end-to-end throughput [2]. We next formalize this process.

1) Decision variables: To characterize the load on nodes and links, we define two sets of decision variables: $\{f^k_a(v, h)\}$ and $\{f^k_a(e, l)\}$. For a request type $k \in [K]$ from AP $a \in A$, $f^k_a(v, h) \geq 0$ defines the *allocated demand* to microservice $v \in V_k$ on edge node $h \in H_v$ for processing, and $f^k_a(e, l) \geq 0$ defines the *allocated demand* for inter-dependency $e \in E_k$ on communication link $l \in L$; both variables are defined in terms of *source demand*, *i.e.*, the volume of type-k requests in-coming from a that are assigned to a node/link. These variables are later translated into actual computation (*e.g.*, GHz for CPU frequency) and communication load (*e.g.*, bps for bandwidth) via the load ratios. Additionally, a single variable $\psi \geq 0$ represents the maximum incurred load on any node/link, defined as the total load divided by the capacity of a node/link.

2) Capacities: The following constraints couple demand allocations with the maximum load factor ψ, using microservice load ratios and node/link capacities.

$$\sum_{k\in[K]} \sum_{a\in A} \sum_{e\in E_k} \left(\rho^k_{e,1} \cdot f^k_a(e, l) + \rho^k_{e,2} \cdot f^k_a(e, l'_l)\right) \leq b_l \cdot \psi, \ \forall l \in L; \tag{1}$$

$$\sum_{k\in[K]} \sum_{a\in A} \sum_{v\in V_k \cap V_h} \rho^k_v \cdot f^k_a(v, h) \leq c_h \cdot \psi, \ \forall h \in H. \tag{2}$$

Note that in Eq. (1), both the forward and backward communications are accounted for for each edge and link, where l'_l is the opposite-direction link of $l \in L$.

3) Processing and flow conservation: The computation demand of a microservice comes from either external users, or a predecessor microservice in the processing DAG. At a microservice, each received user request (unit demand) must be processed and then forwarded to every successor microservice via some API call(s). This motivates us to define the following constraint for the processing and forwarding process:

$$\sum_{l\in L_{\mathsf{out}}(n)} f_a^k(e,l) - \sum_{l\in L_{\mathsf{in}}(n)} f_a^k(e,l) = \mathbf{1}_{\{n\in H_u\}}\cdot f_a^k(u,n) - \mathbf{1}_{\{n\in H_v\}}\cdot f_a^k(v,n), \quad \forall k\in[K], a\in A, n\in N, e=(u,v)\in E_k, \tag{3}$$

where $\mathbf{1}_{\{\mathbf{X}\}}$ is the indicator function of some condition $\mathbf{X}$. The left-hand-side of Eq. (3) defines the *excessive (out-going) flow* incurred by inter-dependency e on node n. If both u and v do not have any instance on node n, then the excessive flow should be exactly 0, as n only forwards traffic for inter-dependency e. If u is on n but v is not, then the excessive flow should be equal to the amount of demand that u processes on n; in other words, u has to forward all its processed demand to any instance(s) of successor v located on some other nodes. Similarly, if v is on n but u is not, then v should receive the same amount of demand that it is assigned to process from any instance(s) of u somewhere else, *i.e.*, its negative excess flow should equal $f_a^k(v,n)$. Finally, when both u, v are on n, then the excessive out-going flow simply equals the demand that u processes minus the demand that v processes. If the excessive flow is positive, then u processes more than v on n and hence must send some of its processed demand to some other instance(s) of v; if the excessive flow is negative, then v is processing more than u on n, and some of v's demand must come from some other instance(s) of u. In summary, Eq. (3) defines 1) a multi-source multi-destination network flow for data transmission for each inter-dependency e, and 2) correlation between each microservice instance's processing demand allocation and its input/output demand flow.

4) Demand: Assume the demand for each request type k at each AP a is known, denoted as δ_a^k. The demand allocation should cover all external demands to avoid exceeding the provisioned resources. For simplicity, we assume that each request's G_k has a single entrance microservice v_k, and that v_k is deployed at every AP instead of on edge nodes; this assumption is for mathematical convenience only, and does not affect generality of our model. The demand constraint is:

$$f_a^k(v_k,n) \geq \delta_a^k, \quad \forall k\in[K], n=a\in A. \tag{4}$$

5) Objective: With the above, the edge provider's goal is to minimize the maximum load incurred on any node/link. The ***microservice load balancing (MLB)*** problem is defined as:

$$\min_{f,\psi} \quad \psi \qquad \text{s.t. (1)–(4).} \tag{5}$$

Remark: We note that although Program (5) is similar to the microservice load balancing problem studied in existing work [2], they differ in at least three aspects. First, the formulation in [2] assumes existence of dedicated communication channels between microservice instances, while Program (5) more practically models a shared edge network between computation nodes. Second, our model balances load across both nodes and links, while [2] only considers computation load balancing but neglects the network load. Third, both forward and backward communications are considered in our model, while [2] only considers forward communications. These factors make our model a non-trivial generalization of [2]. Further, we will extend our model to address the dynamic demands of microservice-based applications, as we show next.

IV. Robust Provisioning with Dynamic Loads

A. Load Dynamics in Microservice Load Balancing

If demands $\{\delta_a^k\}$ are known and fixed, Program (5) is an LP with a polynomial size, and can be solved optimally in polynomial time [26]. However, real-world user demands are heterogeneous and dynamic across geographical areas. For instance, for a taxi service, demand distribution in different city areas can be drastically different in workdays versus weekends, or during busy hours versus off hours, as have been verified in large-scale datasets [15]. This requires constantly adjusting load balancing in the network, or otherwise congestion will happen and will lead to degraded application performance.

At application level, it is easy to employ adaptive load balancing across microservice instances, through orchestrators such as Kubernetes [27]. However, load balancing is limited by the provisioned resources on edge nodes and in the network, which are much more difficult to adjust in the real time. For instance, Kubernetes does not generally allow adding or removing resources such as CPU cores or memory to microservice instances during run-time, and only allows adding more instances on a limited number of platforms. Meanwhile, real-time network reconfiguration is also prohibitively expensive even with software-defined networking (SDN). The difficulty in real-time resource adjustment thus requires edge provider to specifically tackle the potential load dynamics proactively.

B. Stochastic Modeling of Load Dynamics

We tackle dynamics via stochastic modeling. To start with, we use random variables to model demand dynamics, where $\delta_a^k \in \mathbb{R}^*$ now denotes the *random demand* of type k at AP a here and hereafter; the minimum maximum load ψ Program (5) then also becomes a random variable that depends on $\{\delta_a^k\}$. Replacing δ_a^k with the expectation $\mathbb{E}[\delta_a^k]$ yields a stochastic version of MLB. Nevertheless, while this stochastic version considers load dynamics, optimizing for the expected case does not provide robustness: it is likely that δ_a^k will frequently exceed $\mathbb{E}[\delta_a^k]$, in which case the provisioned resources cannot serve all demands and congestion would rise.

This motivates us to explicitly model worst-case scenarios in provisioning, utilizing tools in risk modeling and management that were originally developed for finance. Formally, given a random variable $\mathbf{R}$ denoting the potential investment loss due to market variation, and given a target confidence level $\alpha \in [0,1]$, we can define two *risk measures* [19]:

$$\mathrm{VaR}_\alpha(\mathbf{R}) \triangleq \min\{r \mid \Pr[\mathbf{R}\leq r] \geq \alpha\}; \tag{6}$$

$$\mathrm{CVaR}_\alpha(\mathbf{R}) \triangleq \mathbb{E}[\mathbf{R} \mid \mathbf{R} \geq \mathrm{VaR}_\alpha(\mathbf{R})]. \tag{7}$$

In the above, $\mathrm{VaR}_\alpha(\mathbf{R})$ denotes the minimum threshold value r of investment loss, such that the actual loss will not exceed r with at least α probability. $\mathrm{CVaR}_\alpha(\mathbf{R})$ then further defines the expected loss when the loss actually exceeds the threshold $\mathrm{VaR}_\alpha(\mathbf{R})$, *i.e.*, the expected loss in the worst $(1-\alpha)$ scenarios.

CVaR gives us an excellent tool for integrating robustness into Program (5) with controlled robustness. Unlike *robust optimization* which solely optimizes for the absolute worst

case, CVaR allows tuning the level of robustness via the α parameter, which should match with the level of confidence that edge provider wants to provide to the MSA regarding its throughput guarantee. Integrating CVaR into Program (5) yields the following ***robust MLB (RMLB)*** problem:

$$\begin{aligned} \min_{f,\psi} \quad & \psi \\ \text{s.t.} \quad & (1)\text{–}(3); \\ & f_a^k(v_k, n) \geq \text{CVaR}_\alpha(\delta_a^k),\ \forall k \in [K], n = a \in A. \end{aligned} \tag{8}$$

C. Sample Average Approximation of CVaR

Solving the stochastic constrained formulation in (8) has at least two challenges. First, one has to derive a closed-form formulation of the CVaR to be integrated into Program (8), as the computation of CVaR crucially depends on the computation of VaR as in Eq. (7). Second, optimization with CVaR commonly requires knowing the exact distributions of random variables $\{\delta_a^k\}$. In reality, the exact probabilistic distributions of demands are unknown, and statistical estimation with known distributions can be both costly and inaccurate.

In this subsection, we address the above challenges with an approximation approach. First, to integrate CVaR into Program (8), we apply the following equivalent CVaR formulation:

$$\text{CVaR}_\alpha(\mathbf{R}) = \min_r \left\{ r + \frac{1}{1-\alpha}\mathbb{E}[(\mathbf{R} - r)^+] \right\}, \tag{9}$$

where $(\cdot)^+ = \max\{0, \cdot\}$. The equivalence between Eqs. (7) and (9) is proved in Theorem 1 in [19]. To further tackle the unknown load distributions, we apply the idea of *sample average approximation (SAA)* in stochastic optimization. In plain words, SAA approximates each expectation term in a stochastic program with the average value of a set of *random samples* that are drawn from the underlying distribution. Specifically, assume for each δ_a^k we obtain a set of samples denoted by $\{\tilde{\delta}_a^{k,1}, \tilde{\delta}_a^{k,1}, \cdots, \tilde{\delta}_a^{k,N}\}$ where N is the sample size. Then Eq. (9) can be approximated with

$$\text{CVaR}_\alpha(\delta_a^k) \approx \min_r \left\{ r + \frac{1}{1-\alpha}\frac{1}{N}\sum\nolimits_{i=1}^{N} (\tilde{\delta}_a^{k,i} - r)^+ \right\}. \tag{10}$$

Eq. (10) is a convex program involving a single variable r. Based on standard *i.i.d.* assumptions, it can be shown that Eq. (10) converges to the true CVaR value with probability 1 when the sample size $N \to \infty$; see [28, Chapter 9] for details.

D. A Data-Driven Approach for RMLB

We wrap up description of our approach with a discussion on sampling and management. Traditionally, SAA is complemented by Monte Carlo experiments to obtain i.i.d. samples for approximation, which could be expensive in reality. Fortunately, modern MSAs are commonly equipped with measurement and monitoring modules, which can automatically collect *observed demand* data in the network. Since the external demands are generated by users, they are generally not affected by the measurement and resource management decisions, and hence can be empirically regarded as *i.i.d.* samples. With sufficient data on observed demand, the edge provider can first apply Eq. (9) to derive the CVaR value of each δ_a^k, and then solve Program (8) to derive a resource provisioning plan.

One important choice that the edge provider needs to make is on the trade-off between SAA accuracy and resource management overhead. The underlying demand distribution across geographical locations can be regarded as a mixture of demand distributions over the temporal domain. For instance, the demand distribution at 8am on Monday is generally quite different from the demand distribution at 8pm on the same day, or at 8am on a weekend day. Given a limited amount of data samples, it is generally beneficial to approximate the underlying distributions than directly approximating the mixture distribution, as the former would have lower variance than the latter due to the time-varying and repeating demand patterns validated in real-world datasets [15].

Getting back to RMLB, the edge provider can choose to approximate the demand distribution across all times using the entire dataset, and derive a *static resource provisioning* plan that stays the same for most of the time. It can also choose to divide the dataset into different time slots (*e.g.*, per hour on weekdays and weekends), and derive resource provisioning for each time slot. The latter is expected to improve provisioning performance (*e.g.*, reducing the per-time slot maximum load in RMLB), though with increased overhead for switching between provisioning plans across time slots. Our evaluations show that the benefit of employing time-varying provisioning is appreciable, and could likely outweigh the overhead.

V. Performance Evaluation

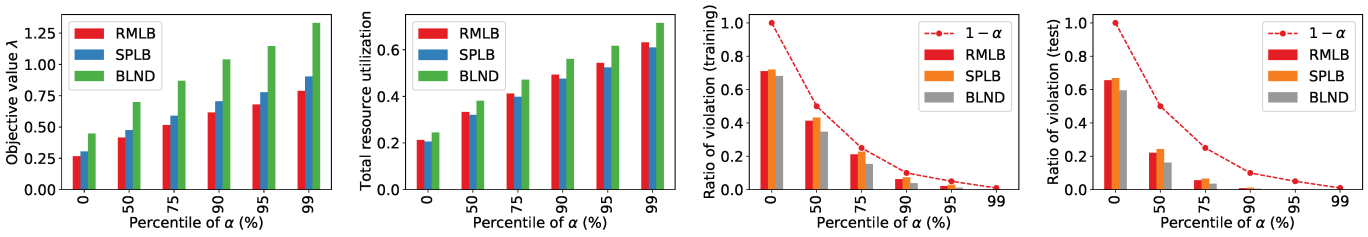
A. Simulation Dataset and Setting

Due to lack of an overall dataset containing both application and network data, our evaluation dataset was synthesized from real-world and randomly generated data. For application data, we profiled a real-world MSA: the *social network* from DeathStarBench [3]. The application contained 23 inter-dependent microservices. To profile the application, we generated three types of workloads: *compose-post*, *read-home-timeline*, and *read-user-timeline*, and then used Wireshark to obtain the full packet traces between microservices for each type of request. We then derived the communication load ratios by summing up the number of bytes transmitted from/to each microservice and between each pair of inter-dependent microservices respectively; for computation load ratios, we empirically summed communication load ratios of all in-coming edges, and assigned computation capacities correspondingly in the same unit. Some statistics of the profiled MSA are shown in Table I.

TABLE I: Workload Types, Statistics and Demand Data

Workload Type	# MSs	# Edges	Data Source
compose-post	21	25	pick-up drivers
read-home-timeline	4	3	pick-up passengers
read-user-timeline	5	4	drop-off passengers

For the edge network and the geo-distributed demands, we obtained the NYC 2018 Yellow Taxi Trip Data from NYC Open Data [29], including pick-up/drop-off times and taxi zones for 112 million taxi trips. We picked 20 most popular taxi zones in terms of total pick-up and drop-off traffic, which accounted for more than 50% of the total traffic over all 265 zones. Each picked taxi zone was regarded as one AP associated with one edge node. For each zone (AP), we extracted three types of demands, corresponding to the 10-minute total numbers of *pick-up drivers*, *pick-up passengers*, and *drop-off passengers* averaged in each hour as a data point. We then mapped the taxi demands to the application's workload types as in Table I, with each driver/passenger count simulating 6250 corresponding mapped requests per second, distributed in the area of an AP. Finally, to build the edge network, we assumed the APs/edge nodes were connected by a Watts–Strogatz random graph with 4 neighbors and 0.3 rewiring



(a) Max load factor λ in training (b) Total edge resource consumption (c) Ratio of violation (training) (d) Ratio of violation (testing)

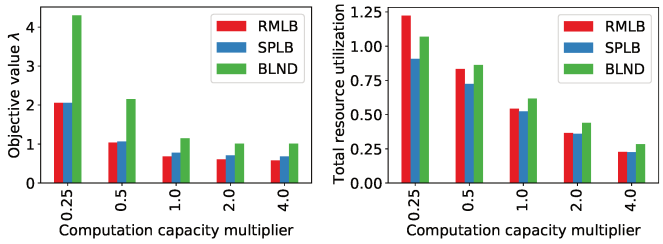
Fig. 1: Results with varying confidence α (percentile). Total resource consumption in (b) includes both computing and network resources. The $(1-\alpha)$ curve in (c)–(d) shows the expected fraction of violations guaranteed by VaR/CVaR.

probability for each node. We assumed each link had 1Gbps capacity, and each edge node could process up to 2.5Gbps of in-coming data. Each microservice was deployed on 20% randomly selected edge nodes. To average out random noise, we generated network topologies with 50 random seeds in each simulation setting, and averaged each evaluation metric by running the comparison algorithms on the generated networks.

The one-year demand data was divided into a *training* and a *test* set. The training set included the first 20% (73 days) of the entire dataset, while the test set included the rest 80%. We used the training set as historical demand data to derive the estimated CVaR demands, and solve the RMLB problem. We then used the rest 80% to test the performance of the RMLB resource provisioning solution. The default confidence α was set as 95%. We compared our RMLB solution to two of its variants: 1) a shortest path-based load balancing (SPLB) solution which solves RMLB with fixed shortest path routing between any pair of nodes, and 2) a blind load balancing (BLND) solution in which each microservice evenly distributes its load to all successor instances without considering inter-dependencies and microservice locations, but bandwidth provisioning is solved with the same multi-source multi-destination flow as in our formulation. The two baseline variants were picked to show the impact of network-(un)awareness and inter-dependency-(un)awareness respectively. All solutions were implemented in Python code with Gurobi [30] as the LP solver.

B. Evaluation Results

1) Impact of α on robustness: In Fig. 1, we evaluated RMLB, SPLB and BLND with different α values in $[0\%, 99\%]$ for CVaR computation. Note that $\alpha = 0$ means the CVaR is equal to the expected demands. In Fig. 1(a), with higher α, the training objective value λ increased, as more worst-case demands were taken into account. Consequently in Fig. 1(b), the total resource utilization (defined as the sum of provisioned communication and computing resources over sum of all capacities) increased to deal with the worst-case demands. Among all algorithms, RMLB always achieved the best λ in training. SPLB consumed slightly less resources than RMLB at the cost of a higher λ, due to its less flexibility with route selection; naive load balancing with BLND had significantly worse λ and resource consumption. Meanwhile in Figs. 1(c) and (d), the fraction of cases where real demands violated provisioned resources drastically decreased with increased α for both the training and testing datasets, showing the great advantage of adding robustness. Note that for all algorithms, the number of cases that violated the provisioned resources are significantly less than the $(1-\alpha)$ guarantee provided by VaR/CVaR in both training and testing phases, demonstrating that CVaR provides the desired level of robustness to resource provisioning. Overall, by solving RMLB with different α values, the edge provider can clearly see the trade-off between resource consumption and robustness, and make the best decision based on its optimization goal.

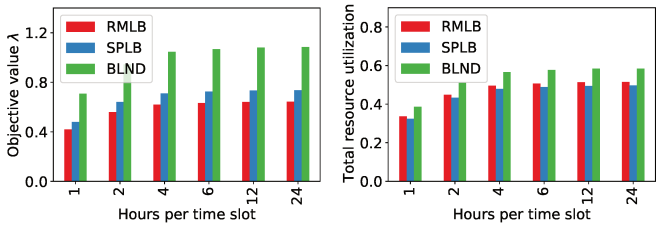


(a) Max load factor λ in training (b) Total edge resource consumption

Fig. 2: Results with varying node capacities.

2) Computing and communication bottlenecks: In Fig. 2, we evaluated RMLB with different resource bottlenecks. Specifically, we left link capacities unchanged, while multiplying node capacities by a scaling factor in $[0.25, 4]$. This effectively changed the ratio of computing versus communication capacities in the network. In Fig. 2(a), the objective value λ decreased with diminishing return when the node-link capacity ratio increased, showing the transitioning of resource provisioning from computing-bottlenecked to communication-bottlenecked. Comparing RMLB to the two baselines, first, RMLB outperformed both SPLB and BLND in almost all cases in terms of λ, though this may be at the cost of more resource consumption than the baselines sometimes. Second, the gap between SPLB and RMLB was smaller when computing was the bottleneck, and the gap between BLND and RMLB was smaller when communication was the bottleneck. This is intuitive, as SPLB is generally *network-agnostic*, while BLND is to some extent *computing-agnostic* and does not consider the inter-dependencies between microservices. RMLB's superior load balancing performance comes from its joint planning of computation and communication resources, and explicit consideration of microservice inter-dependencies.

3) Time-varying resource adjustment: In Fig. 3, we evaluated the impact of time-varying resource provisioning. We first divided all days into weekdays and weekends, and then further divided each day into 24 hours. Then, we aggregated the hours into different time slots, ranging from 1 hour to 24 hours per slot. In total, this resulted in 48 to 2 time slots. We then divided the demand dataset into the dataset for each time slot, and solved the RMLB problem for each time slot independently. This simulates the case when the edge provider can adjust the resource provisioning across time slots, but cannot do it



(a) Max load factor λ in training (b) Total edge resource consumption

Fig. 3: Results with time-varying resource provisioning.

more frequently due to the involved cost and performance overhead. In Fig. 3(a), we showed the average λ value across *all time slots* by each algorithm. We can see that with finer-grained time-varying resource provisioning (fewer hours per slot), the resulted λ is smaller. In Fig. 3(b), a similar trend can be observed for total resource consumption. Both results have suggested that employing time-varying resource provisioning can better adjust to the different demand distributions at different time slots, resulting in better load balancing performance.

VI. Conclusions

In this paper, we studied robust resource provisioning for MSA load balancing. To address microservice inter-dependencies and accurately model the required resources for serving geo-distributed application demands, we developed a network-aware resource provisioning formulation called MLB. To address dynamic geo-distributed demands, we then introduced stochastic modeling into the formulation, and formulated a robust version of the problem called RMLB, utilizing a risk estimation and optimization tool from economics and finance called CVaR. The robust formulation focuses on optimizing resource provisioning with respect to worst-case demand fluctuations up to a certain confidence level chosen by the edge provider, and can flexibly trade-off between model conservativeness (robustness) and resource consumption. Based on the robust formulation, we further developed a data-driven approach for approximating CVaR with data samples regarding real-world demand distributions. We evaluated RMLB with simulations based on real-world application and demand data, and showed its flexibility in robustness-resource trade-off and superior performance compared to baseline solutions.

References

[1] Y. Niu, F. Liu, and Z. Li, "Load Balancing Across Microservices," in *IEEE INFOCOM*, 2018.

[2] R. Yu, V. T. Kilari, G. Xue, and D. Yang, "Load Balancing for Interdependent IoT Microservices," in *IEEE INFOCOM*, 2019, pp. 298–306.

[3] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," in *ACM ASPLOS*, 2019, pp. 3–18.

[4] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a Smart City Internet of Things Platform with Microservice Architecture," in *IEEE FiCloud*, 2015, pp. 25–30.

[5] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices Approach for the Internet of Things," in *IEEE ETFA*, 2016, pp. 1–6.

[6] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, may 2016.

[7] H. Kang, M. Le, and S. Tao, "Container and Microservice Driven Design for Cloud Infrastructure DevOps," in *IEEE IC2E*, 2016, pp. 202–211.

[8] A. U. Gias, G. Casale, and M. Woodside, "ATOM: Model-Driven Autoscaling for Microservices," in *IEEE ICDCS*, 2019, pp. 1994–2004.

[9] M. Villari, A. Celesti, G. Tricomi, A. Galletta, and M. Fazio, "Deployment Orchestration of Microservices with Geographical Constraints for Edge Computing," in *IEEE ISCC*, 2017, pp. 633–638.

[10] Z. Song and E. Tilevich, "Win with What You Have: QoS-Consistent Edge Services with Unreliable and Dynamic Resources," in *IEEE ICDCS*, 2020, pp. 530–540.

[11] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, mar 2021.

[12] L. Wang, L. Jiao, T. He, J. Li, and M. Muhlhauser, "Service Entity Placement for Social Virtual Reality Applications in Edge Computing," in *IEEE INFOCOM*, 2018, pp. 468–476.

[13] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading Dependent Tasks in Mobile Edge Computing with Service Caching," in *IEEE INFOCOM*, 2020, pp. 1997–2006.

[14] G. Castellano, F. Esposito, and F. Risso, "A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees," in *IEEE INFOCOM*, 2019, pp. 2548–2556.

[15] R. Yu, G. Xue, Y. Wan, J. Tang, D. Yang, and Y. Ji, "Robust Resource Provisioning in Time-Varying Edge Networks," in *ACM Mobihoc*, 2020.

[16] Z. Xu, L. Zhou, S. Chi-Kin Chau, W. Liang, Q. Xia, and P. Zhou, "Collaborate or Separate? Distributed Service Caching in Mobile Edge Clouds," in *IEEE INFOCOM*, 2020, pp. 2066–2075.

[17] C. Zhang, H. Tan, H. Huang, Z. Han, S. H. Jiang, N. Freris, and X.-Y. Li, "Online Dispatching and Scheduling of Jobs with Heterogeneous Utilities in Edge Computing," in *ACM Mobihoc*, 2020, pp. 101–110.

[18] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the Starting Line: Joint Network Selection and Service Placement for Mobile Edge Computing," in *IEEE INFOCOM*, 2019, pp. 1459–1467.

[19] R. T. Rockafellar and S. Uryasev, "Optimization of Conditional Value-at-Risk," *Journal of Risk*, vol. 2, pp. 21–41, 2000.

[20] R. Farajifijani, S. Ahmadian, S. Ebrahimi, and E. Ghotbi, "Wind Farm Layout Optimization Problem Using Joint Probability Distribution of CVaR Analysis," in *Proc. IEEE CCWC*, 2019, pp. 0007–0012.

[21] R. Yu, G. Xue, V. T. Kilari, and X. Zhang, "Deploying Robust Security in Internet of Things," in *IEEE CNS*, 2018.

[22] J. Bogle, N. Bhatia, M. Ghobadi, I. Menache, N. Bjørner, A. Valadarsky, and M. Schapira, "TeaVaR: Striking the Right Utilization-Availability Balance in WAN Traffic Engineering," in *ACM SIGCOMM*, 2019, pp. 29–43.

[23] D. Xu, Y. Li, T. Xia, J. Li, S. Tarkoma, and P. Hui, "Portfolio Optimization in Traffic Offloading: Concept, Model, and Algorithms," *IEEE Transactions on Mobile Computing*, p. 1, 2019.

[24] A. Jindal, V. Podolskiy, and M. Gerndt, "Performance Modeling for Cloud Microservice Applications," in *ACM/SPEC ICPE*, 2019, pp. 25–32.

[25] H. Chang, M. Kodialam, T. Lakshman, and S. Mukherjee, "Microservice Fingerprinting and Classification Using Machine Learning," in *IEEE ICNP*, 2019, pp. 1–11.

[26] Y. Ye, "An O(n3L) Potential Reduction Algorithm for Linear Programming," *Mathematical Programming*, vol. 50, no. 1-3, pp. 239–258, mar 1991.

[27] "Kubernetes." [Online]. Available: https://kubernetes.io/

[28] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, 2011.

[29] NYC Open Data, "2018 Yellow Taxi Trip Data," 2018. [Online]. Available: https://data.cityofnewyork.us/Transportation/2018-Yellow-Taxi-Trip-Data/t29m-gskq

[30] Gurobi, "Gurobi Optimizer." [Online]. Available: http://www.gurobi.com/products/gurobi-optimizer