Load Balancing for Interdependent IoT Microservices

Ruozhou Yu, Vishnu Teja Kilari, Guoliang Xue, Dejun Yang

***Abstract*—Advances in virtualization technologies and edge computing have inspired a new paradigm for Internet-of-Things (IoT) application development. By breaking a monolithic application into loosely coupled microservices, great gain can be achieved in performance, flexibility and robustness. In this paper, we study the important problem of load balancing across IoT microservice instances. A key difficulty in this problem is the interdependencies among microservices: the load on a successor microservice instance directly depends on the load distributed from its predecessor microservice instances. We propose a graph-based model for describing the load dependencies among microservices. Based on the model, we first propose a basic formulation for load balancing, which can be solved optimally in polynomial time. The basic model neglects the quality-of-service (QoS) of the IoT application. We then propose a QoS-aware load balancing model, based on a novel abstraction that captures a realization of the application's internal logic. The QoS-aware load balancing problem is NP-hard. We propose a fully polynomial-time approximation scheme for the QoS-aware problem. We show through simulation experiments that our proposed algorithm achieves enhanced QoS compared to heuristic solutions.**

Keywords*—IoT, microservice, application graph, load balancing, fully polynomial-time approximation scheme

I. Introduction

The Internet-of-Things (IoT) has drastically grown in size and capability in the recent years, owing to advances in broadband access networks, cloud/edge computing, big data analytics, machine learning, etc. In the near future, the global IoT can expand to tens of billions of devices, powering up numerous applications such as smart city, smart home, smart health, connected vehicles, etc. The global economic impact of IoT can be more than several trillion dollars in early 2020s [1].

Due to IoT's rapid development, the traditional monolithic architecture is no longer suitable for IoT applications. Instead, the microservice architecture is gaining support from both industry and academia. The architecture is built upon loosely coupled microservices, each with compact logic and well-defined interfaces. An IoT application is built as a collection of microservices with inter-microservice communications. In real deployment, an application can employ multiple instances of each microservice to achieve elasticity and robustness.

A key difficulty in microservice management is the interdependencies among microservices. Specifically, the input data to one microservice may depend on the output data of other microservices. To capture this, existing works adopt a graph-based approach, modeling an application as a directed graph where vertices represent microservices, and edges represent data flows between microservices. Fig. 1 shows an example.

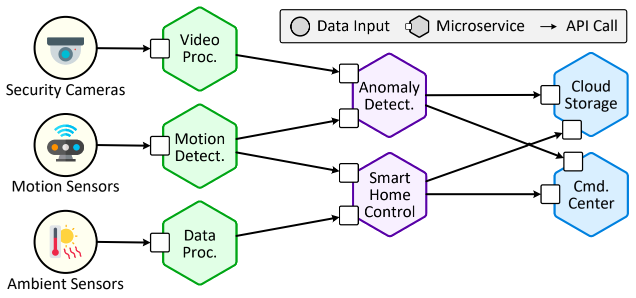


Fig. 1: Graph representation of a smart home application.

In this paper, we study an important problem in microservice management: load balancing across microservice instances. Lower load on instances can lead to better robustness and elasticity when facing instance failures or demand changes. Yet microservice load balancing is a complex problem due to the interdependencies. Specifically, changing the load distribution at one microservice instance could cause changes to the load on many other microservice instances. This situation is aggravated by the heterogeneous connectivity between microservice instances, which are distributed in the edge network.

Existing efforts have adopted simplified models to make the load balancing problem tractable, *e.g.*, by abstracting the application's processing logic as a chain of microservices [23]. Such a model, however, is insufficient to capture the rich interdependencies in modern IoT applications. In this paper, we aim to provide a general solution to load balancing for interdependent microservices. We start with a directed acyclic graph (DAG)-based model for describing microservice interdependencies, which is rich enough to abstract a majority of IoT applications. We then propose a linear program formulation for the basic load balancing problem, where the application aims to minimize the maximum load among all instances.

The basic model neglects the quality-of-service (QoS) goal of the application, and may result in arbitrarily large end-to-end delay when answering user requests. This motivates us to study the more complex QoS-aware load balancing problem, where the application aims to optimize the end-to-end QoS while satisfying a desired load balancing goal. We first present a method to characterize the QoS of a load balancing solution, by abstracting a realization structure for the application graph. We show through a decomposition theorem that the realization structure can precisely represent QoS-aware load balancing solutions. Unfortunately, the QoS-aware problem is NP-hard. Hence we propose a fully polynomial-time approximation scheme (FPTAS) for the problem. We show through simulation experiments that our proposed algorithm indeed outperforms heuristic solutions in terms of QoS of the application.

Our main contributions are summarized as follows:

- To our knowledge, we are the first to study microservice load balancing with DAG-based interdependencies.

Yu, Kilari and Xue ({ruozhouy, vkilari, xue}@asu.edu) are all with Arizona State University, Tempe, AZ 85287. Yang (djyang@mines.edu) is with Colorado School of Mines, Golden, CO 80401. This research was supported in part by NSF grants 1717197, 1704092, 1461886, and 1717315.

- We formulate both a basic and a QoS-aware load balancing problem. We prove the latter to be NP-hard.
- We show that the basic problem can be solved optimally, while the QoS-aware problem admits an FPTAS.
- Simulation experiments have shown that our algorithm can improve application QoS compared to baselines.

The rest of this paper is organized as follows. In Sec. II, we introduce background and related work. In Sec. III, we present system model and the basic load balancing model. In Sec. IV, we describe our QoS-aware model, a formal statement of the problem, and its complexity. In Sec. V, we propose the FPTAS for QoS-aware load balancing. In Sec. VI, we show our simulation results. In Sec. VII, we conclude this paper.

II. Background and Related Work

A. Microservices and Application Graph Models

The microservice architecture, originally proposed for complex business applications in enterprises such as Amazon [2] and Netflix [3], has rapidly gained attention in the IoT domain, where it can largely reduce the cost and complexity of building IoT applications. With the help of distributed edge computing, a microservice-based application can achieve a number of benefits over a monolithic application, including robustness [17], elasticity [26], security [21], evolvability [12], and many more.

Graph-based approaches are commonly adopted to model and manage such decomposed IoT applications. Belli *et al.* [7] proposed an application architecture, which uses the processing graph to characterize QoS and improve infrastructure usage efficiency. Akkermans *et al.* [4] built a system for application orchestration based on application graphs. Erbs *et al.* [11] built another graph-based distributed processing system, which, in addition to orchestration across logical components, also considers the chronological dependencies among components. Lee *et al.* [19] used *tenant application graphs* to model cloud applications, and proposed bandwidth-aware application embedding in tree-like cloud networks.

Resource allocation for application graphs can be hard due to the complex structures of such graphs. Many heuristic solutions exist without performance bound [6], [14]. To address this, existing approaches used simplified graph models. For example, Li *et al.* [20] and Niu *et al.* [23] used a chain-based model. Li *et al.* [20] proposed a heuristic for bandwidth-aware application chain deployment. Niu *et al.* [23] proposed a game theoretical approach to coordinate resources among competing application chains, in order to minimize their response times. A related area considers the embedding of *service function chains* (SFC), where similarly one or multiple chains of service functions are to be deployed in the network. Approximation algorithms exist, *e.g.*, due to Cao *et al.* [8], Kuo *et al.* [18], and Yu *et al.* [31]. Yu *et al.* [32] studied a different model where the application has a star structure, and proposed an FPTAS for network load balancing. Yet these simplified models greatly limit the expressiveness of this approach, and hence are not suitable for general IoT applications.

Outside of the IoT domain, there are problems with similar abstractions. An early area of research is virtual network embedding (VNE), where a request is given as a virtual network graph that is to be embedded on a physical network [10]. A related area is data center virtualization, where a tenant request is also given as a graph, which is to be embedded in the physical cloud network [33]. Since these problems are mainly studied in the network domain, they are different from resource allocation for application graphs. As an example, these problems commonly request that a virtual node is mapped to one and only one physical node, while a vertex (microservice) in an application graph can be implemented by a number of distributed instances at the same time.

B. Application-level Load Balancing

The load balancing problem has been studied in many different contexts, such as parallel computing [27], web applications [9], cloud applications [24], network load [5], [15], [25], microservices [23], etc. Due to the large body of work, we only do a brief review on the methods used.

Many existing works are based on the principle of *randomized load balancing*, where in-coming load is randomly assigned to different entities based on load information. For example, Equal-Cost Multi-Path (ECMP) [25] is one of the most widely used Layer-3 load balancing technique that has a lot of variants. In the computing domain, it has also been shown that using randomized load balancing can reduce the queueing delay at servers [30]. A drawback of the randomized approach is that it is difficult to consider the load interdependencies among entities, hence it may result in skewed load distributions at entities that significantly depend on others.

Contrary to randomized load balancing, *deterministic load balancing* can make decisions based on many different factors, such as entity interdependencies [6], [14], [23], QoS information [23], [32], energy consumption [24], failures and network asymmetry [15], etc. These solutions differ from the randomized ones in that commonly the former need global coordination to obtain system-wide information and to enforce load balancing policies. In network, this either requires a centralized network controller [15], or complex peer-to-peer information aggregation and dissemination [5]. In the computing domain, the task is easier, as most computing platforms already employ hypervisors to make centrally coordinated decisions. Among problems with different considerations, load balancing with complex interdependencies seems to be one of the most difficult, where only heuristic solutions exist [6], [14].

III. System Model and Basic Formulation

A. Application Model

An IoT application is built by selecting a number of microservices, and establishing proper inter-connections between their output and input APIs. Specifically, an application is modeled as a directed acyclic graph (DAG), denoted as $G = (V, E)$, where V is the set of *vertices* denoting microservices, and E is the set of *edges* denoting direct API calls between microservices. A microservice v is called a successor of microservice u if there is a directed edge $(u, v) \in E$; v is a predecessor of u if the opposite edge $(v, u) \in E$ exists. A microservice with no successor is called a *sink microservice*. For simplicity, we use $V_{\text{in}}(v)$ and $V_{\text{out}}(v)$ to denote the subsets of predecessors and successors of microservice v, respectively. G is called the *application graph* (app-graph) hereafter.

The set E defines how IoT data flow across microservices. For load balancing, it is important to capture how data are distributed at each microservice. For each edge $e = (u, v) \in E$, a *data distribution ratio* r_e is defined, which denotes the input data volume that microservice v would receive from u if u is fed with 1 unit of input data. The input data of a microservice thus depends on both the external data it directly receives, and the data distributed from its predecessors. r_e can be obtained via analysis of historical measurements. An app-graph example is shown in Fig. 2(a).

B. Infrastructure Model

The IoT application must be instantiated with microservice instances in the network. For each microservice $v \in V$, we define N_v as the set of *nodes* (microservice instances) that implement v. The physical connectivities between instances of a pair of microservices $(u, v) \in E$ are defined by *link* set $L_{uv} \subseteq N_u \times N_v$. Since we only care about connectivities between microservices that have direct API calls, the set L_{uv} is only well-defined for $(u, v) \in E$. The *infrastructure graph* (inf-graph) is denoted as $\Gamma = (N, L)$, where $N = \bigcup_{v \in V} N_v$ and $L = \bigcup_{(u,v) \in E} L_{uv}$. For simplicity, we let $v_n \in V$ be the microservice that node n belongs to; we then use the same notation, "predecessor/successor" of node n, to denote the corresponding predecessor/successor microservice of v_n. We further let $L_{\text{in}}(n)$ or $L_{\text{out}}(n)$ be the subset of links in-coming or out-going node n, respectively, and $L_{\text{out}}(n, v)$ be the subset of out-going links of node n that point to nodes belonging to microservice v. The inf-graph is also a DAG.

We next define a number of attributes for the inf-graph. First, each node $n \in N$ has a capacity c_n, which is the maximum load it can process to avoid congestion. In stream-analysis applications, c_n is commonly measured in terms of input data volume. Second, each node may have a processing delay d_n. The delay values can be obtained, *e.g.*, using the method outlined in [16]. Each link $l \in L$ may also have a delay d_l, denoting the data transmission latency between the two instances. For a path p, its delay is defined as the sum of node and link delays on p: $d(p) = \sum_{n \in p} d_n + \sum_{l \in p} d_l$. The application receives input data from external sources such as IoT devices, and the data may be fed into a certain node based on the data source locations, types of data, frontend distribution policies, etc. We use δ_n^{ext} to denote the volume of external data fed into node n, also called its *external demand*. A node with $\delta_n^{\text{ext}} > 0$ is called a *source node*. Note that node n may also receive input data from other nodes through API calls, which is different from its external demand, and is called the *internal demand* instead. An inf-graph example is shown in Fig. 2(b), corresponding to the app-graph in Fig. 2(a).

Note that for clarity of illustration, we use different terms for different graphs. We use "vertex" and "edge" for entities in the app-graph. We use "node" and "link" for entities in the inf-graph. In the next section, we will use "point" and "arc" for entities in the realization graph, to be explained later.

C. Basic Load Balancing Model

In our scenario, an application is instantiated by allocating external and internal demands to the microservice instances. In choosing how to instantiate the application, its owner aims to balance the load across different microservice instances, in order to achieve the best performance as well as to leave room for elastic scaling in the future. As a first step, we establish a formal model for the basic load balancing problem, which neglects the QoS requirement of the application.

Let δ_n be the total demand into node $n \in N$, which is the summation of both its external and internal demands: $\delta_n = \delta_n^{\text{ext}} + \delta_n^{\text{int}}$. The external demand δ_n^{ext} is regarded as a constant value, but the internal demand δ_n^{int} depends on the demands distributed from predecessor microservice instances of n, which in turn depends on the input demands of the predecessor instances, their corresponding data distribution ratios, and how they allocate their own output demands. Define $f(n_1, n_2)$ as the *demand allocation* from n_1 to n_2 if $(n_1, n_2) \in L$. We then have $\delta_n^{\text{int}} = \sum_{l \in L_{\text{in}}(n)} f(l)$, and hence:

$$\delta_n = \delta_n^{\text{ext}} + \sum\nolimits_{l \in L_{\text{in}}(n)} f(l), \quad \forall n \in N. \tag{1}$$

For a demand allocation function $f : L \mapsto \mathbb{R}^*$ ($\mathbb{R}^*$ is the non-negative real number set) to be feasible, it must satisfy the following two constraints:

1) The total demand at each node should not exceed its desired capacity multiplied by a load factor ψ:
$$\delta_n \leq \psi \cdot c_n, \quad \forall n \in N. \tag{2}$$
2) The demand distributed from a node to all nodes of a successor microservice should satisfy the data distribution ratio between the two microservices:
$$\sum\nolimits_{l \in L_{\text{out}}(n,w)} f(l) = r_{(v_n,w)} \delta_n, \ \forall n, w \in V_{\text{out}}(v_n). \tag{3}$$

The load factor ψ essentially specifies the maximum load of any microservice instance. Commonly, the application would have a desired bound Ψ, such that the load on any instance does not exceed this bound. We then define the following problem:

Definition 1. *Given app-graph G with inf-graph Γ, and load bound $\Psi > 0$, the* ***Basic Load Balancing (BLB)*** *problem seeks for a demand allocation function $f : L \mapsto \mathbb{R}^*$, which satisfies Eqs.* (1), (2) *and* (3) *while ensuring $\psi \leq \Psi$. Its optimization version, instead of giving a load bound Ψ, is to minimize ψ, and is named* ***O-BLB*** *hereafter.* □

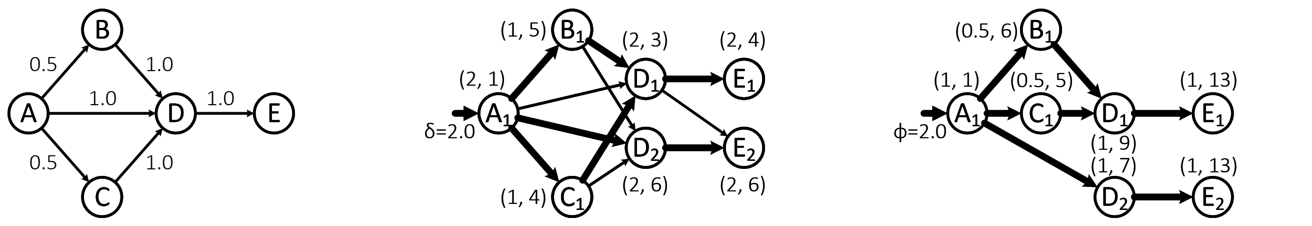
The following linear program (LP) formulates O-BLB:

$$\min_{f \geq 0} \quad \psi \tag{4}$$
$$\text{s.t.} \quad (1), (2) \text{ and } (3).$$

Theorem 1. *O-BLB can be solved in $O(|L|^3 \cdot \mathbb{L})$ time.* □

Proof: Program (4) is an LP with $(|L| + 1)$ variables. Based on [29], the LP can be solved optimally in $O(|L|^3 \cdot \mathbb{L})$ time ($\mathbb{L}$ is the input size). ■

In Fig. 2(b), we also show a feasible solution to the load balancing problem. Bold links show links with positive demand allocation, and each bold link's allocation is equal to the cumulative load imposed on the link. For example, link $A_1 \rightarrow B_1$ has an allocation of $\delta \cdot 0.5 = 1.0$, while link $D_1 \rightarrow E_1$ has an allocation of $\delta \cdot (0.5 \cdot 1.0 + 0.5 \cdot 1.0) \cdot 1.0 = 2.0$ with the first half of demand coming from the path $A_1 \rightarrow B_1 \rightarrow D_1$, and the second half from $A_1 \rightarrow C_1 \rightarrow D_1$. A clearer view of the solution is shown in Fig. 2(c), which will be detailed using the real-graph abstraction in the next section. This basic model, however, neglects the QoS of the application, and may lead to arbitrarily long delay for serving user demands. In the next section, we propose a novel model for characterizing QoS.



(a) App-graph $G = (V, E)$ (Sec. III). Symbols in circles are microservices. Values on lines are distribution ratios.

(b) Inf-graph $\Gamma = (N, L)$ (Sec. III). Symbols in circles are microservice instances. Values beside circles are (capacity, delay). Links have 0 delay. Bold links show a feasible load balancing solution.

(c) Real-graph $\pi = (X_\pi, A_\pi)$ (Sec. IV). Symbols in circles are instances mapped from the corresponding points. Values beside circles are (impact ratio, max cumulative delay).

Fig. 2: App-graph (a), inf-graph (b) and a real-graph (c) of an example application with load bound $\Psi = 1$. Bold links in (b) show a feasible load balancing solution with max delay of 13, which is further shown in (c) as a single decomposed real-graph.

IV. QoS-aware Load Balancing

The above outlines a basic formulation of the load balancing problem. The model, however, merely reflects the numerical relationship between different instances, without describing the richer structural relationship in the app-graph. It is therefore intrinsically difficult to incorporate QoS information into the formulation. In this section, we model the QoS of an application through a novel realization graph abstraction.

We consider the following QoS goal of the application. In IoT, users usually ask for a guaranteed response time to ensure timely reception and handling of IoT events, such as traffic status or emergency events. We hence assume that the application's QoS goal is to bound or minimize the *maximum end-to-end delay* that any external demand would experience. To characterize this, we propose the following abstraction:

Definition 2 (Real-graph)**.** *Given app-graph $G = (V, E)$, inf-graph $\Gamma = (N, L)$ and a source node $n \in N$, a* ***realization graph (real-graph)*** *is defined as a DAG $\pi = (X_\pi, A_\pi)$, coupled with a mapping $\sigma : X_\pi \mapsto N$, which satisfies that:*
1) x_π with $\sigma(x_\pi) = n$ is the only point with 0 in-degree;
2) $\forall x \in X_\pi$ and $\forall v \in V_{\text{out}}(v_{\sigma(x)})$, there is exactly one $y \in X_\pi$, such that $\sigma(y) \in N_v$, $(\sigma(x), \sigma(y)) \in L$, and $(x, y) \in A_\pi$. $\square$

X_π is the set of *points* in π, and A_π is the set of *arcs*. We call x_π the root point of real-graph π. Real-graphs with roots mapped to source node $n \in N$ are denoted by set Π_n, and real-graphs of all source nodes are denoted by set $\Pi = \bigcup_n \Pi_n$. For simplicity, we use the same notation σ to map entities (nodes, links, paths or subgraphs) in π to the corresponding entities in Γ. We also use a point x to represent the node $\sigma(x) \in N$ when no ambiguity is introduced. $N_\pi \subseteq N$ and $L_\pi \subseteq L$ denote the subsets of nodes and links that are mapped from some points and arcs in π, respectively.

We now explain the intuition behind Definition 2. A real-graph can be viewed as a unitary structure that realizes every possible processing path starting with a source microservice (a microservice that has source nodes) in the app-graph. Each path is realized by a sequence of physical instances. To do this, we start from a source node, and recursively instantiate every successor microservice of the current node by assigning an instance to it, until we reach the sinks. Note that in this process, we may choose different instances of the same microservice, each as the successor of a different predecessor instance. Hence there can be multiple points in the real-graph mapped to the same node. We show an example of a real-graph in Fig. 2(c), which will be explained later.

We can define a number of attributes for π. First, each point/arc inherits the delay of its mapped node/link in Γ. Second, since each point x has exactly one neighbor y for each successor $v \in V_{\text{out}}(v_x)$, we can define the distribution ratio $r_{x,y} = r_{(v_{\sigma(x)}, v_{\sigma(y)})}$ for $(x, y) \in A_\pi$. We can further derive the *impact ratio* ρ^π_x for each point $x \in X_\pi$, defined as the *demand on x when one unit of demand is input at root x_π*. This can be computed by initially letting $\rho^\pi_{x_\pi} = 1$, and then traversing π from x_π, with each point adding its own impact ratio times the distribution ratio of each out-going arc to the impact ratio of the corresponding out-going neighbor. Similarly, the impact ratio ρ^π_a for each arc $a \in A_\pi$ can be computed. Then, we sum the impact ratios of all points mapped to a node $m \in N_\pi$ to compute the impact ratio ρ^π_m imposed on m by π, and similarly the impact ratio ρ^π_l on each link $l \in L_\pi$. The unitarity of π is guaranteed by assigning exactly one instance to every point's every successor microservice; in other words, each processing path in the app-graph corresponds to exactly one path in π.

Based on these, we can define a *source demand allocation* function $\phi : \Pi \mapsto \mathbb{R}^*$. For $\pi \in \Pi_n$, the value $\phi(\pi)$ denotes the external demand at the source node n that is allocated to be carried on real-graph π. Each node/link's demand under π can then be computed by multiplying $\phi(\pi)$ with the node/link's impact ratio. We highlight the importance of this real-graph abstraction in the following theorem, which is an analogy to the Flow Decomposition theorem for traditional network flow:

Theorem 2. *Any demand allocation f satisfying Eqs.* (1) *and* (3) *can be decomposed into at most $|N| + |L|$ real-graphs Π^{sel} with $\phi(\pi) > 0$ for $\forall \pi \in \Pi^{sel}$, such that the total demand incurred by ϕ on any link $l \in L$ is no more than $f(l)$.* $\square$

Proof: We decompose f as follows. We first find an arbitrary real-graph $\pi \in \Pi_n$ for $n \in N$ with $\delta^{\text{ext}}_n > 0$, such that $f(l) > 0$ for $\forall l \in L_\pi$. We then calculate the maximum acceptable demand of π as $\delta(\pi) = \min\{\delta^{\text{ext}}_n, f(l)/\rho^\pi_l \text{ for } \forall l \in L_\pi\}$, *i.e.*, the minimum among δ^{ext}_n, and the total allocated demand on every link l that appears in π factored by the inverse of its impact ratio $1/\rho^\pi_l$. We let $\phi(\pi) = \delta(\pi)$. We then visit every link $l \in L_\pi$, deducting $\delta(\pi) \cdot \rho^\pi_l$ from $f(l)$; we also deduct $\delta(\pi)$ from δ^{ext}_n. After the deduction, the remaining f and demands still satisfy Eqs. (1) and (3), since we deduct the same amount from the lefthand and righthand sides of Eqs. (1) and (3). Due to our calculation of $\delta(\pi)$, at least one link's allocated demand is fully taken away during the deduction, or the total demand at a source node is deducted. We need at most $|N| + |L|$ steps to deduct all allocations from f. Also, we never deduct more than the demand allocated on any link, hence any capacity constraint satisfied by f is also satisfied by ϕ.

We now prove that we can always find a π with $f(l) > 0$ for $\forall l \in L_\pi$, if f is feasible and $\exists n \in N$ s.t. $\delta_n^{\text{ext}} > 0$. Let n be a node with $\delta_n > 0$ where δ_n comes from either external or internal demands. By Eq. (3), there exists $m \in N_v$ for $\forall v \in V_{\text{out}}(v_n)$ such that $f(n, m) > 0$, and hence $\delta_m > 0$. Therefore, we can start from any n with positive external demand, arbitrarily select a node $m \in N_v$ with $f(n, m) > 0$ for each successor microservice v of v_n, and then follow this process at each selected m until no successor to work on. This clearly generates a real-graph whenever f is feasible and has positive external demand, which completes our proof. ■

Fig. 2(c) shows a real-graph, which is also a decomposition of the load balancing solution shown in Fig. 2(b) (in practice, a solution may be decomposed into multiple real-graphs; in our example, only one is needed). We compute both the impact ratio and the cumulative delay for each point as shown in the figure. For example, the point mapped to D_1 has impact ratio $(r_{\text{AB}} \cdot r_{\text{BD}} + r_{\text{AC}} \cdot r_{\text{CD}}) = 1.0$. Delay is computed as the maximum delay from the root, *e.g.*, the delay at the point mapped to D_1 is $d_{\text{A}_1} + \max\{d_{\text{B}_1}, d_{\text{C}_1}\} + d_{\text{D}_1} = 9$. Note that although the real-graph realizes the app-graph, it may not be isomorphic to the app-graph, as it allows instantiating a microservice by multiple instances, such as microservice D instantiated by D_1 and D_2.

Theorem 2 is fundamental, as it enables us to use real-graphs as a basic structure for characterizing a load balancing solution. In other words, instead of defining a per-link allocation function f, we can define the allocation ϕ over the real-graphs from each source node, with the end-to-end delay of the application defined as the maximum delay from the source point to the leaf points of any real-graph with positive allocation. We can then define the QoS-aware load balancing problem. Let D be the application's delay bound. Define $d(\pi)$ as the maximum path delay from root x_π to any leaf point in π: $d(\pi) = \max\{d(p) \mid p \in \pi\}$. For brevity, we let $\Pi^m = \{\pi \in \Pi \mid m \in N_\pi\}$ be the subset of all real-graphs that include points mapped to node m.

Definition 3. *Given app-graph G, inf-graph Γ, load bound $\Psi > 0$, and delay bound $D > 0$, the **QoS-aware Load Balancing (QLB)** problem seeks for a subset of real-graphs Π_n^{sel} for each source node n, with $\Pi^{\text{sel}} = \bigcup_{n \in N} \Pi_n^{\text{sel}}$, and a source demand allocation function $\phi : \Pi^{\text{sel}} \mapsto \mathbb{R}^*$, s.t.:*
1) $\psi \le \Psi$,
2) for node $\forall m \in N$, $\sum_{\pi \in \Pi^m} \rho_m^\pi \cdot \phi(\pi) \le \psi \cdot c_m$,
3) for source node $\forall n \in N$, $\sum_{\pi \in \Pi_n^{\text{sel}}} \phi(\pi) = \delta_n^{\text{ext}}$, and
4) for real-graph $\forall \pi \in \Pi^{\text{sel}}$, $d(\pi) \le D$.
*The optimization version, denoted as **O-QLB** hereafter, is to minimize the maximum delay of all selected real-graphs.* □

Proof of the following theorem is deferred to the appendix.

Theorem 3. *Both QLB and O-QLB are NP-hard.* □

V. Approximation Scheme Design

Due to the NP-hardness of O-QLB, we seek to develop an approximation algorithm. Below, we first show that if all delay values are positive integers, the QLB problem can be solved in pseudo-polynomial time. Such a problem is defined as Integral QLB (IQLB), with O-IQLB being its optimization version to minimize maximum delay. The pseudo-polynomial time algorithm is then used as a building block in the design of an approximation scheme for the general non-integral problem.

A. Pseudo-Polynomial Time Optimal Algorithm

Our pseudo-polynomial time algorithm for IQLB is based on a layered graph technique [22], [28]. Given inf-graph Γ and an integral delay bound D, we define an auxiliary inf-graph $\Gamma^D = (N^D, L^D)$. The node set $N^D = \{n^0, n^1, \ldots, n^D \mid n \in N\}$, *i.e.*, a node has $(D+1)$ copies each belonging to a layer. For source node $n \in N$, we let $\delta_{n^{d_n}}^{\text{ext}} = \delta_n^{\text{ext}}$; all other nodes have 0 external demand. Let $d_{nm}^+ = d_n + d_{(n,m)}$ be the delay of link $(n, m) \in L$ plus the delay of m. The link set $L^D = \{(n^i, m^{i+d_{nm}^+}) \mid (n, m) \in L, i = 0, 1, \ldots, D - d_{nm}^+\}$, *i.e.*, each original link in L has $(D - d_{nm}^+ + 1)$ copies. Each link copy inherits the distribution ratio of the original link. Due to page limit, we refer the reader to [22] (p. 4, Fig. 4) for an illustrative figure of the layered graph technique.

It is easy to see that each path or real-graph in Γ^D corresponds to exactly one path or real-graph in Γ, respectively. On the opposite direction, each path or real-graph starting with one source node n in Γ also corresponds to exactly one path or real-graph in Γ^D (since the external demand of each source node in Γ enters at exactly one node in Γ^D). As our focus is only on paths or real-graphs starting with source nodes, we use p or π to denote both a path or real-graph in Γ, and its correspondence in Γ^D, without introducing ambiguity.

The intuition behind this construction is to enforce that going through a node n or link l in Γ is equivalent to "climbing" d_n or d_l layers in the auxiliary inf-graph, respectively. The processing delay of source node n is encoded such that its external demand enters in the respective d_n-th layer. Since only $(D+1)$ layers (from 0 to D) present, any processing path must reach a sink node within D layers, thus bounding the maximum delay. Formally, we have the following observation:

Observation 1. *Any path p or real-graph π in Γ^D has delay $d(p) \le D$ or $d(\pi) \le D$ in Γ, respectively.* □

Combining Theorem 2 and Observation 1, we are motivated to study the basic load balancing problem on the auxiliary inf-graph, which is formulated as the following LP:

$$\min_{f \ge 0} \quad \psi \tag{5a}$$

$$\text{s.t.} \quad \delta_n = \sum\nolimits_{l \in L_{\text{in}}^D(n)} f(l) + \delta_n^{\text{ext}},\ \forall n \in N^D; \tag{5b}$$

$$\sum\nolimits_{i=0}^{D} \delta_n \le \psi \cdot c_n,\ \forall n \in N; \tag{5c}$$

$$\sum_{l \in L_{\text{out}}^D(n,w)} f(l) = r_{(v_n,w)} \delta_n, \forall n \in N^D, w \in V_{\text{out}}(v_n). \tag{5d}$$

Program (5) is almost the same as Program (4), except that the capacity constraint (5c) now considers the demands entering all copies of the same node n. We then have the following:

Theorem 4. *IQLB can be solved in $O(D^4|L|^3\mathbb{L})$ time.* □

Proof: Program (5) is an LP with at most $D|L|$ variables and input size of $O(D\mathbb{L})$, and hence it can be solved in $O(D^4|L|^3\mathbb{L})$ time. By Theorem 2, the feasible solution to Program (5) can be decomposed into at most $D(|N|+|L|)$ real-graphs on Γ^D. By Observation 1, each real-graph has delay bounded by D, thus the solution is feasible to IQLB if the optimal value of Program (5) satisfies $\psi \le \Psi$. Now, assume IQLB has a feasible solution, by reversing the decomposition, we can construct a demand allocation f that satisfies all constraints in Program (5). The theorem follows. ■

B. Approximation Scheme for O-QLB

The basic idea of our algorithm is to find a sufficiently fine-grained discretization of the real-valued delays, such that the discretized solution is a good approximation of the optimal solution, and yet the time complexity is polynomial to the input size (and the inverse of an approximation factor ϵ). For this reason, we use a factor α to represent the granularity of discretization. We then define the discretized delay values given α: $d^\alpha_n = \lfloor \alpha \cdot d_n \rfloor + 1$ for $n \in N$, and $d^\alpha_l = \lfloor \alpha \cdot d_l \rfloor + 1$ for $l \in L$; we use similar symbols $d^\alpha(p)$ or $d^\alpha(\pi)$ to denote the delay of a path or a real-graph after discretization, respectively. The discretized delays satisfy the following lemma:

Lemma 1. *For any path p in Γ, we have*

$$\alpha \cdot d(p) \le d^\alpha(p) \le \alpha \cdot d(p) + 2|N| - 1. \qquad \square$$

Proof: The left inequality is clear. The right one is because each path has at most $|N|$ nodes and $|N| - 1$ links. ■

By selecting a proper factor α, we discretize the O-QLB problem to have only integral delay values. We want to solve the resulting O-IQLB problem to obtain an approximation to the original O-QLB problem. Let $\Delta^{\text{O-QLB}}$ be the optimal value to the original O-QLB problem, and let (UB, LB) be a pair of bounds such that $\text{UB} \ge \Delta^{\text{O-QLB}} \ge \text{LB}$. We define the discretization factor as $\alpha = \frac{2|N|-1}{\epsilon \text{LB}}$, given a small approximation factor ϵ. Let $\Delta^{\text{O-IQLB}}$ be the optimal value to the corresponding O-IQLB instance. We have the following:

Lemma 2. $\alpha \cdot \text{LB} \le \Delta^{\text{O-IQLB}} \le \lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1.$ $\square$

Lemma 3. $\alpha \cdot \Delta^{\text{O-QLB}} \le \Delta^{\text{O-IQLB}} \le \alpha \cdot (1 + \epsilon) \cdot \Delta^{\text{O-QLB}}.$ $\square$

Proof of Lemmas 2 and 3: Let (Π^{sel}, ϕ) be an optimal solution to O-QLB. Since it is also feasible to O-IQLB, with Lemma 1, we have

$$\begin{aligned}\Delta^{\text{O-IQLB}} &\le \max\{d^\alpha(p) \mid \pi \in \Pi^{\text{sel}}, p \in \pi\} \\ &\le \alpha \cdot \max\{d(p) \mid \pi \in \Pi^{\text{sel}}, p \in \pi\} + 2|N| - 1 \\ &\le \alpha \cdot \Delta^{\text{O-QLB}} + 2|N| - 1 \\ &\le \alpha \cdot \text{UB} + 2|N| - 1.\end{aligned} \tag{6}$$

This implies the right inequality in Lemma 2, as $\Delta^{\text{O-IQLB}}$ is always an integer due to the discretization. From the third inequality in Eq. (6), we have

$$\begin{aligned}\Delta^{\text{O-IQLB}} &\le \alpha \cdot (\Delta^{\text{O-QLB}} + \frac{2|N|-1}{\alpha}) \\ &= \alpha \cdot \left(\Delta^{\text{O-QLB}} + \epsilon \cdot \text{LB}\right) \\ &\le \alpha \cdot (1 + \epsilon) \cdot \Delta^{\text{O-QLB}}\end{aligned} \tag{7}$$

Based on the left inequality of Lemma 1, we have the left inequality in Lemma 3, which implies the left inequality in Lemma 2. This proves Lemmas 2 and 3. ■

We now talk about the implications of Lemmas 2 and 3. Lemma 3 states that $\Delta^{\text{O-IQLB}}$ divided by α provides a $(1+\epsilon)$-approximation to the value $\Delta^{\text{O-QLB}}$. Lemma 2 further provides a method to compute the value $\Delta^{\text{O-IQLB}}$, given a pair of bounds (UB, LB) on $\Delta^{\text{O-QLB}}$. A bisection method can be used to search the space $[\alpha \cdot \text{LB}, \lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1]$ for the delay value $\Delta^{\text{O-IQLB}}$, each time solving an instance of IQLB by Program (5). The entire search requires $O(\log \frac{\lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1}{\alpha \cdot \text{LB}}) = O(\log \frac{|N|\text{UB}}{\epsilon \text{LB}})$ runs, each with time complexity of $O(|L|^3 (\frac{|N|\text{UB}}{\epsilon \text{LB}})^4 \mathbb{L})$. To summarize, Lemmas 2 and 3 give us a method for computing a $(1+\epsilon)$-approximation of the optimal solution to O-QLB within time polynomial to the input size, the parameter $1/\epsilon$, and the ratio $\frac{\text{UB}}{\text{LB}}$ between the upper and lower bounds.

Algorithm 1: FPTAS-O-QLB(G, Γ, Ψ)

```
1  Solve Program (4) on $\Gamma$ to test feasibility;
2  Ascendingly sort all delay values as $\mathbf{d} = (d_0, \ldots, d_K)$;
3  $lo \leftarrow 0, hi \leftarrow K$;
4  while $lo < hi - 1$ do
5      $mi \leftarrow \lfloor (hi + lo)/2 \rfloor$;
6      Construct sub-graph $\Gamma_{-d_{mi}}$;
7      Solve Program (4) on $\Gamma_{-d_{mi}}$ for optimal load $\psi$;
8      if $\psi > \Psi$ then $lo \leftarrow mi$;
9      else $hi \leftarrow mi$;
10 end
11 $d_{\text{bot}} \leftarrow d_{hi}, \text{LB} \leftarrow d_{\text{bot}}, \text{UB} \leftarrow (2|N|-1)d_{\text{bot}}$;
12 Discretize delays with $\alpha = (2|N|-1)/\epsilon\text{LB}$;
13 $D_{lo} \leftarrow \lfloor \alpha \cdot \text{LB} \rfloor, D_{hi} \leftarrow \lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1$;
14 while $D_{lo} < D_{hi} - 1$ do
15     $D_{mi} \leftarrow \lfloor (D_{lo} + D_{hi})/2 \rfloor$;
16     Construct auxiliary inf-graph $\Gamma^{D_{mi}}$;
17     Solve Program (5) on $\Gamma^{D_{mi}}$ for optimal load $\psi$;
18     if $\psi > \Psi$ then $D_{lo} \leftarrow D_{mi}$;
19     else $D_{hi} \leftarrow D_{mi}$;
20 end
21 Do real-graph decomposition on the solution for $\Gamma^{D_{hi}}$;
22 return $(\Pi^{\text{sel}}, \phi)$ obtained by decomposition.
```

To establish a polynomial-time algorithm, our next step is to find a pair of bounds (UB, LB) that is within a polynomial factor of each other. This can be done by finding a *bottleneck delay* that "determines" the feasibility of the problem instance. Let Γ_{-d} be the sub-graph of Γ where all nodes and links with delays larger than d are pruned. We wish to find the minimum value d_{bot} such that $\Gamma_{-d_{\text{bot}}}$ still admits a feasible BLB solution satisfying the load bound Ψ. Note that a source node n having delay $d_n > d$ directly eliminates the existence of a feasible solution in Γ_{-d}. Naturally, if the original O-QLB instance is feasible, which can be checked by solving Program (4) on the original Γ, then d_{bot} exists. Since there are at most $|N| + |L|$ distinct delay values, d_{bot} can be found also using bisection in $O(\log(|N| + |L|))$ iterations, each spending $O(|L|^3 \cdot \mathbb{L})$ time solving Program (4). We then have the following:

Lemma 4. $(2|N| - 1) \cdot d_{\text{bot}} \ge \Delta^{\text{O-QLB}} \ge d_{\text{bot}}.$ $\square$

Proof: By the definition of d_{bot}, it follows that the QLB instance is feasible on $\Gamma_{-d_{\text{bot}}}$ but is infeasible on $\Gamma_{-d_{\text{bot}}}$ with all nodes/links with delay equal to d_{bot} removed. This means that any feasible solution to QLB includes at least one node/link with delay no less than d_{bot}, which means d_{bot} is a lower bound of $\Delta^{\text{O-QLB}}$. On the other hand, since there is a feasible solution in $\Gamma_{-d_{\text{bot}}}$, and any path in $\Gamma_{-d_{\text{bot}}}$ has at most $|N|$ nodes and $(|N| - 1)$ links, the longest path delay in the feasible solution cannot exceed $(2|N| - 1) \cdot d_{\text{bot}}$. Hence $(2|N| - 1) \cdot d_{\text{bot}}$ is an upper bound of $\Delta^{\text{O-QLB}}$. This completes the proof. ■

Lemma 4 provides us a pair of bounds $\text{UB} = (2|N| - 1) \cdot d_{\text{bot}}$ and $\text{LB} = d_{\text{bot}}$ with $\text{UB}/\text{LB} = 2|N| - 1$. Putting together the bounds with Lemmas 2 and 3, we now have an approximation scheme for O-QLB, shown in Algorithm 1.

Lemma 5. *Given any $\epsilon > 0$, Algorithm 1 outputs a $(1+\epsilon)$-approximation of the optimal O-QLB solution, within $O(\frac{1}{\epsilon^4}|L|^3|N|^8\mathbb{L}\log\frac{|N|}{\epsilon} + |L|^3\mathbb{L}\log|N|)$ time.* □

Proof: The approximation ratio is due to Lemma 3. The dominant time complexity comes from the two bisection searches along with an LP solving per search iteration. The first search takes $O(\log(|N|+|L|))$ iterations each with $O(|L|^3 \cdot \mathbb{L})$ time complexity. The second search takes $O(\log\frac{|N|}{\epsilon})$ iterations each with $O(\frac{1}{\epsilon^4}|L|^3|N|^8\mathbb{L})$ time complexity. Since $|L| < |N|^2$, we have the final complexity by adding them together. ■

C. Efficiency Enhancement

While Algorithm 1 runs in time polynomial to input size and $1/\epsilon$, it has a time complexity as high as $O(|N|^{12}\log|N|)$ assuming $|L| = \Omega(|N|)$ and ϵ is a constant. Observe that the high complexity mainly comes from solving LPs with $D|L|$ variables and input size $D\mathbb{L}$, where $D = D_{mi}$ is in the order of $\frac{|N|\text{UB}}{\epsilon\text{LB}}$ and UB/LB is in the order of $|N|$. If we can reduce UB/LB to a constant, a reduction of order $|N|$ can be achieved on the program size, resulting in orders of reduction in the overall time complexity. We use a technique called *approximate testing* to achieve such a reduction [22].

Specifically, we define a test procedure $\text{TEST}_\omega(\mathcal{D})$. Given $\omega > 0$ and $\mathcal{D} > 0$, we define a new discretization factor $\alpha = \frac{2|N|-1}{\omega\mathcal{D}}$ and discretize all delay values in Γ. We then define an instance of IQLB as (G, Γ, Ψ, D') where $D' = \lfloor\frac{2|N|-1}{\omega}\rfloor + 2|N| - 1$ and all delay values in Γ are discretized by α. Let $\text{TEST}_\omega(\mathcal{D}) = \text{True}$ if the discretized IQLB instance (G, Γ, Ψ, D') has a feasible solution, and $\text{TEST}_\omega(\mathcal{D}) = \text{False}$ otherwise. We then have the following lemma:

Lemma 6. *Given any $\omega > 0$ and $\mathcal{D} > 0$, we have*

$$\text{TEST}_\omega(\mathcal{D}) = \text{True} \quad \Rightarrow \quad \Delta^{\text{O-QLB}} \le (1+\omega)\cdot\mathcal{D};$$
$$\text{TEST}_\omega(\mathcal{D}) = \text{False} \quad \Rightarrow \quad \Delta^{\text{O-QLB}} > \mathcal{D}.$$ □

Proof: Assume $\text{TEST}_\omega(\mathcal{D}) = \text{True}$, then we have a feasible solution (Π^{sel}, ϕ) for discretized $\text{IQLB}(G, \Gamma, \Psi, D')$, which also translates to a feasible solution for $\text{O-QLB}(G, \Gamma, \Psi)$. Let p be any processing path in any $\pi \in \Pi^{\text{sel}}$, we have $d^\alpha(p) \le D'$. Based on Lemma 1, we then have

$$\begin{aligned} d(p) &\le d^\alpha(p)/\alpha \le D'/\alpha \\ &\le \left(\frac{2|N|-1}{\omega} + 2|N| - 1\right)\cdot\frac{\mathcal{D}\cdot\omega}{2|N|-1} \\ &= \mathcal{D}(1+\omega) \end{aligned} \tag{8}$$

Since $\Delta^{\text{O-QLB}} \le \max_{\pi\in\Pi^{\text{sel}}, p\in\pi}\{d(p)\}$ (as the solution is feasible to O-QLB), we have $\Delta^{\text{O-QLB}} \le (1+\omega)\cdot\mathcal{D}$.

To prove the second statement, we can prove its contraposition, *i.e.*, if $\Delta^{\text{O-QLB}} \le \mathcal{D}$ then $\text{TEST}_\omega(\mathcal{D})$ must output True. Let π be any real-graph in the optimal O-QLB solution, and let p be any path in π. Based on Lemma 1, we then have

$$\begin{aligned} d^\alpha(p) &\le \alpha\cdot d(p) + 2|N| - 1 \\ &\le \frac{2|N|-1}{\omega\mathcal{D}}\Delta^{\text{O-QLB}} + 2|N| - 1 \\ &\le \frac{2|N|-1}{\omega} + 2|N| - 1. \end{aligned} \tag{9}$$

This implies that $d^\alpha(p) \le \lfloor\frac{2|N|-1}{\omega}\rfloor + 2|N| - 1$ since again the delays are discretized to be integers. Therefore the optimal solution to O-QLB also translates to a feasible solution to the discretized IQLB instance, and hence $\text{TEST}_\omega(\mathcal{D})$ must output True based on Theorem 4. This completes the proof. ■

```
Algorithm 2: RefineBound($G, \Gamma, \Psi, \text{LB}, \text{UB}$)
  // Between Lines 11 and 12 in Algo. 1
1 while $\text{UB} > 4\cdot\text{LB}$ do
2     $\mathcal{D} \leftarrow \sqrt{\frac{\text{UB}\cdot\text{LB}}{2}}$;
3     if $\text{TEST}_1(\mathcal{D}) = \text{True}$ then $\text{UB} \leftarrow 2\cdot\mathcal{D}$;
4     else $\text{LB} \leftarrow \mathcal{D}$;
5 end
6 return $(\text{LB}, \text{UB})$.
```

Based on Lemma 6, we can use TEST_ω with $\omega = 1$ as a test procedure for carrying out a bisection search on the pair (UB, LB). Given an initial pair with $\text{UB}/\text{LB} = 2|N| - 1$ (after Line 11 of Algorithm 1), we insert the procedure described in Algorithm 2, to obtain a pair of bounds within a constant factor of each other, before proceeding to Line 12 of Algorithm 1.

Theorem 5. *Given any $\epsilon > 0$, Algorithm 1 plus Algorithm 2 outputs a $(1+\epsilon)$-approximation for O-QLB within time $O(\frac{1}{\epsilon^4}|L|^3|N|^4\mathbb{L}\log\frac{|N|}{\epsilon} + |L|^3|N|^4\mathbb{L}\log\log|N|)$.* □

Proof: First, observe that in Lines 3–4 of Algorithm 2, if $\text{TEST}_1(\mathcal{D})$ outputs True, then $2\cdot\mathcal{D}$ is still a valid upper bound; if $\text{TEST}_1(\mathcal{D})$ outputs False, then $\mathcal{D}$ is still a valid lower bound; both due to Lemma 6. By the choice of $\mathcal{D}$, it satisfies that $\frac{\text{UB}}{\mathcal{D}} = \frac{2\cdot\mathcal{D}}{\text{LB}} = \sqrt{\frac{2\text{UB}}{\text{LB}}}$ in each iteration. Let $\beta = \frac{\text{UB}}{\text{LB}}$, and let $\beta[i]$ be the value of β after the i-th iteration. It is clear that $\beta[i] = \sqrt{2\beta[i-1]} = \cdots = 2^{1/2+1/4+\cdots+1/2^i}\beta[0]^{1/2^i} \le 2\cdot\beta[0]^{1/2^i}$. Therefore Algorithm 2 needs $O(\log\log\beta[0]) = O(\log\log|N|)$ iterations, each solving an IQLB instance with $D' = \lfloor\frac{|N|}{1}\rfloor + |N| = 2|N|$. It follows that Algorithm 2 has a time complexity of $O(|N|^4|L|^3\mathbb{L}\log\log|N|)$. Since we reduce UB/LB to be within 4 after Algorithm 2, the complexity of the second bisection search in Algorithm 1 now becomes $O(\frac{1}{\epsilon^4}|L|^3|N|^4\mathbb{L}\log\frac{|N|}{\epsilon})$. Combining these with the time complexity of the first bisection search in Algorithm 1, we have the claimed time complexity. ■

VI. Performance Evaluation

In this section, we present performance evaluation of our proposed algorithm with simulation experiments. Our algorithm is denoted as QLB. We implemented two heuristic algorithms. In the first algorithm denoted as BLB, the BLB formulation in Program (4) is solved, followed by a simple decomposition as in Theorem 2. In the second algorithm denoted as QHU (QoS-aware Heuristic), we solve a modified

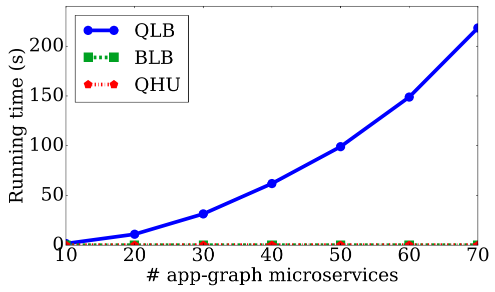
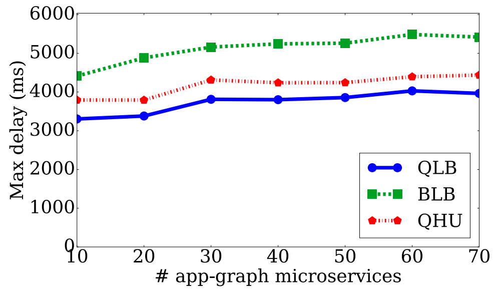
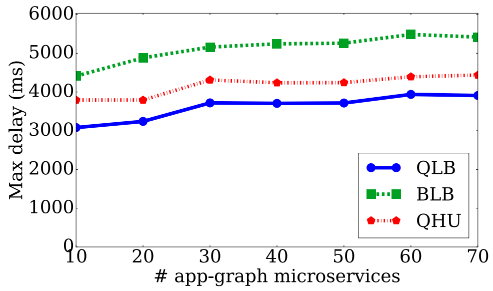


Fig. 3: Running time vs. app-graph size



(a) End-to-end delay with $\Psi = \psi^*$



(b) End-to-end delay with $\Psi = 2\psi^*$

Fig. 4: Objective value vs. app-graph size

version of Program (4) where the objective is changed to minimizing the demand-weighted sum of delays of all nodes and links, meanwhile the load factor ψ has to be bounded by Ψ; we then use the same decomposition method. Therefore the first heuristic corresponds to load balancing that neglects QoS at all, and the second corresponds to a heuristic formulation where the total delay instead of the maximum delay of the solution is minimized in the modified LP.

We used randomly generated app-graphs and inf-graphs. The app-graph was organized as a random 5-layer DAG, with the lowest layer having 10% of the vertices and the other layers sharing the rest. Edges were generated from lower layers to all upper layers, with each vertex in the upper four layers having 4 in-coming edges on average. Distribution ratios of edges were also randomly generated, such that all out-going edges of a vertex had their ratios sum to 1. The inf-graph was generated such that each lowest-layer microservice had only one instance, as a source node, while each microservice in other layers had $[5, 15]$ instances. Links were built between instances of interconnected microservices with a base probability of 0.3, while additional links were added to ensure that each instance could find at least one out-going neighbor for every successor microservice. Each source node had a random demand in $[100, 900]$ and a large capacity such that it would not be a load balancing bottleneck. For all other nodes, capacities were generated in $[10, 90]$. Node and link delays were generated in $[0, 1000]$ ms and $[0, 500]$ ms respectively. Finally, we assigned the load bound Ψ to be one of the two values: it was either the optimal value ψ^* to BLB (Program (4)) or $2 \cdot \psi^*$. We used them to test the delay performance in heavy- or moderate-load scenarios respectively.

We set the accuracy parameter $\epsilon = 0.5$. Experiments were run on a Ubuntu PC with i7-2600 CPU and 16GB memory. To average-out random noise, we conducted 20 random experiments under each setting and took their average.

Fig. 3 shows the running time of the implemented algorithms with growth in the app-graph size. The running time of QLB is longer than the heuristics as expected. The growth in the running time is nevertheless polynomial to the growth in app-graph size (and hence the inf-graph size).

Fig. 4 shows the comparison of the delay values achieved by all three algorithms. We show the comparisons under two settings: heavy load ($\Psi = \psi^*$) and moderate load ($\Psi = 2\psi^*$). We note that QLB achieves the lowest delay in all scenarios, regardless of load. The QoS-aware heuristic QHU can improve delay performance over QoS-agnostic load balancing BLB, but cannot achieve an improvement as significant as QLB. Comparing Figs. 4(a) and 4(b), QLB has a larger advantage in terms of delay when the load is less (Fig. 4(b)). This is partly because when the load is moderate, QLB has more room for selecting nodes and links with lower delays, and thus it can further improve QoS. On the other hand, the results for BLB and QHU are almost the same with different loads, since their formulations commonly generate the same solutions. Differences may result from the decomposition process, which, as shown in the results, has little impact on the QoS of the final solutions.

To summarize, our algorithm always achieves advantageous QoS performance over the heuristics, with a polynomially bounded complexity. This supports our theoretical analysis that QLB has guaranteed performance while both heuristics do not.

VII. Conclusions

In this paper, we studied basic and QoS-aware load balancing across interdependent IoT microservices. A DAG-based model was used to abstract microservice interdependencies. We proposed an LP formulation for the basic problem, which can be solved in polynomial time. For the QoS-aware problem, we proposed a decomposition-based model where a realization of the application is expressed as a realization graph. Since the QoS-aware problem is NP-hard, we proposed an FPTAS, along with an efficiency enhancement technique that achieves several orders of speed-up. Simulations showed that our algorithm achieves enhanced QoS compared to heuristic solutions. We believe that the proposed method, aside from making theoretical contribution to the problem studied, more importantly provides insight in extending chain- or star-based application models to the more general DAG-based model, which is much more expressive and flexible in real-world scenarios.

References

[1] "IoT Market Forecasts." URL: https://www.postscapes.com/internet-of-things-market-size/

[2] "What Led Amazon to Its Own Microservices Architecture." URL: https://thenewstack.io/led-amazon-microservices-architecture/

[3] "Why You Can't Talk About Microservices Without Mentioning Netflix." URL: https://smartbear.com/blog/develop/why-you-cant-talk-about-microservices-without-ment/

[4] S. Akkermans, N. Small, W. Joosen, and D. Hughes, "Niflheim: End-to-End Middleware for Applications Across All Tiers of the IoT," in *Proc. ACM SenSys*, 2017, pp. 1–2.

[5] M. Alizadeh, N. Yadav, G. Varghese, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, and R. Pan, "CONGA: Distributed Congestion-Aware Load Balancing for Datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 503–514.

[6] G. Attiya and Y. Hamam, "Two Phase Algorithm for Load Balancing in Heterogeneous Distributed Systems," in *Proc. IEEE EMPDP*, 2004, pp. 434–439.

[7] L. Belli, S. Cirani, G. Ferrari, L. Melegari, and M. Picone, "A Graph-Based Cloud Architecture for Big Stream Real-Time Applications in the Internet of Things," in *Proc. ESOCC*, 2014, pp. 91–105.

[8] Z. Cao, M. Kodialam, and T. V. Lakshman, "Traffic Steering in Software Defined Networks: Planning and Online Routing," in *Proc. ACM DCC*, 2014, pp. 65–70.

[9] V. Cardellini, M. Colajanni, and P. Yu, "Dynamic Load Balancing on Web-server Systems," *IEEE Internet Comput.*, vol. 3, no. 3, pp. 28–39, 1999.

[10] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in *Proc. IEEE INFOCOM*, 2009, pp. 783–791.

[11] B. Erb, D. Meißner, J. Pietron, and F. Kargl, "Chronograph–A Distributed Processing Platform for Online and Batch Computations on Event-sourced Graphs," in *Proc. ACM DEBS*, 2017, pp. 78–87.

[12] B. Familiar, *Microservices, IoT and Azure: Leveraging DevOps and Microservice*, 2015.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1990.

[14] T. Hagras and J. Janecek, "A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems," in *Proc. IEEE IPDPS*, 2004.

[15] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based Load Balancing for Fast Datacenter Networks," in *Proc. ACM SIGCOMM*, 2015, pp. 465–478.

[16] Y. Jiang, L. R. Sivalingam, S. Nath, and R. Govindan, "WebPerf: Evaluating What-If Scenarios for Cloud-hosted Web Applications," in *Proc. ACM SIGCOMM*, 2016, pp. 258–271.

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

[18] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, "Service Chain Embedding with Maximum Flow in Software Defined Network and Application to the Next-Generation Cellular Network Architecture," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[19] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven Bandwidth Guarantees in Datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 467–478.

[20] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghayi, D. Li, G. Wilfong, Y. R. Yang, and C. Guo, "PACE: Policy-Aware Application Cloud Embedding," in *Proc. IEEE INFOCOM*, 2013, pp. 638–646.

[21] D. Lu, D. Huang, A. Walenstein, and D. Medhi, "A Secure Microservice Framework for IoT," in *Proc. IEEE SOSE*, 2017, pp. 9–18.

[22] S. Misra, G. Xue, and D. Yang, "Polynomial Time Approximations for Multi-Path Routing with Bandwidth and Delay Constraints," in *Proc. IEEE INFOCOM*, 2009, pp. 558–566.

[23] Y. Niu, F. Liu, and Z. Li, "Load Balancing Across Microservices," in *Proc. IEEE INFOCOM*, 2018, pp. 1–9.

[24] A. Paya and D. C. Marinescu, "Energy-Aware Load Balancing and Application Scaling for the Cloud Ecosystem," *IEEE Trans. Cloud Comput.*, vol. 5, no. 1, pp. 15–27, jan 2017.

[25] O. Tilmans, S. Vissicchio, L. Vanbever, and J. Rexford, "Fibbing in Action: On-demand Load-Balancing for Better Video Delivery," in *Proc. ACM SIGCOMM*, 2016, pp. 619–620.

[26] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," *IEEE Cloud Comput.*, vol. 3, no. 6, pp. 76–83, nov 2016.

[27] M. Willebeek-LeMair and A. Reeves, "Strategies for Dynamic Load Balancing on Highly Parallel Computers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 9, pp. 979–993, 1993.

[28] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 656–669, jun 2008.

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

[30] L. Ying, R. Srikant, and X. Kang, "The Power of Slightly More Than One Sample in Randomized Load Balancing," in *Proc. IEEE INFOCOM*, 2015, pp. 1131–1139.

[31] R. Yu, G. Xue, and X. Zhang, "QoS-Aware and Reliable Traffic Steering for Service Function Chaining in Mobile Networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2522–2531, nov 2017.

[32] ——, "Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective," in *Proc. IEEE INFOCOM*, 2018, pp. 1–9.

[33] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable Virtual Data Center Embedding in Clouds," in *Proc. IEEE INFOCOM*, 2014, pp. 289–297.

APPENDIX

Proof of Theorem 3: We derive a reduction from the Partition problem to QLB, the former of which is a well-known NP-hard problem [13]. Given a set of objects $X = \{x_1, \ldots, x_\kappa\}$ and a positive integer a_x for $\forall x \in X$, the Partition problem seeks for a subset $Y \subset X$ such that $\sum_{x \in Y} a_x = \sum_{x \in X \setminus Y} a_x$. Given an instance X of Partition, we construct an instance (G, Γ, Ψ, D) of QLB as follows. $G = (V, E)$ has the vertex set $V = \{v_0, u_1, v_1, u_2, v_2, \ldots, u_\kappa, v_\kappa\}$, and edge set $E = \{(v_{i-1}, u_i), (u_i, v_i) \,|\, i = 1, \ldots, \kappa\}$. $\Gamma = (N, L)$ has the node set $N = \{n_i \,|\, \forall v_i \in V\} \cup \{m_i^0, m_i^1 \,|\, \forall u_i \in V\}$, and link set $L = \{(n_{i-1}, m_i^0), (n_{i-1}, m_i^1), (m_i^0, n_i), (m_i^1, n_i) \,|\, i = 1, \ldots, \kappa\}$. In summary, the app-graph is a line graph with $(2\kappa+1)$ vertices and 2κ edges, and the inf-graph has $(3\kappa+1)$ nodes and 4κ links. For the attributes, all edges in E have distribution ratio of 1. Node n_0 has external demand of 2, while all other nodes have no external demand. All n_i nodes have capacity of 2, while all m_i^j nodes have capacity of 1. All links, all n_i nodes and all m_i^0 nodes have delay of 0, while each m_i^1 node has delay $d_{m_i^1} = a_{x_i}$, for $i = 1, \ldots, \kappa$. Finally, we set the application delay bound $D = \frac{1}{2}\sum_{i=1}^{\kappa} a_{x_i}$, and the load bound $\Psi = 1$. An example is shown in Fig. 5.

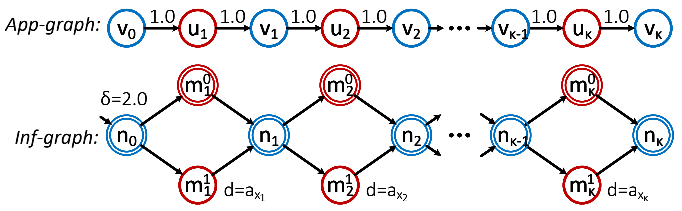


Fig. 5: App-graph values are distribution ratios. Delays of m_i^1 nodes are shown beside them. Double lined nodes and all links have 0 delay. All n_i nodes have capacity of 2. All m_i^j nodes have capacity of 1. $\Psi = 1$. $D = \frac{1}{2}\sum_{i=1}^{\kappa} a_{x_i}$.

Since the app-graph G is a line graph, every real-graph of n_0 is also a line graph from n_0 to n_κ, and hence its delay is precisely the sum of delays of m_i^1 nodes included in the path. Now, suppose instance X of Partition has feasible solution Y. Then instance (G, Γ, Ψ, D) of QLB also has a feasible solution, which has two paths, one taking m_i^1 for $x_i \in Y$ and m_i^0 for $x_i \in X \setminus Y$, and the other taking the opposite m_i^j nodes, both having a demand allocation of 1. On the reverse direction, suppose instance (G, Γ, Ψ, D) has a feasible solution. The solution must also contain two paths since each m_i^j node can only accept half of the demands coming from the predecessor n_{i-1} node. A feasible solution to Partition instance X is then constructed by picking x_i corresponding to all m_i^1 nodes taken by one of the paths. This proves the NP-hardness of QLB, and the NP-hardness of O-QLB follows. ■