


# Exploring microservices for enhancing internet QoS

Deval Bhamare<sup>1</sup>  | Mohammed Samaka<sup>1</sup> | Aiman Erbad<sup>1</sup> | Raj Jain<sup>2</sup> | Lav Gupta<sup>2</sup>

<sup>1</sup>Qatar University, Doha, Qatar

<sup>2</sup>Washington University, St. Louis, USA

## Correspondence

Deval Bhamare, Qatar University, Doha, Qatar.

Email: devalb@qu.edu.qa

## Funding information

Qatar National Research Fund,  
Grant/Award Number: NPRP 8-634-1-131

## Abstract

With the enhancements in the field of software-defined networking and virtualization technologies, novel networking paradigms such as network function virtualization and the Internet of Things are rapidly gaining ground. The development of Internet of Things and 5G networks and explosion in online services has resulted in an exponential growth of devices connected to the network. As a result, application service providers and Internet service providers are being confronted with the unprecedented challenge of accommodating increasing service and traffic demands from the geographically distributed users. To tackle this problem, many ISPs, such as Netflix, Facebook, and AT&T, are increasingly adopting microservices application architecture. Despite the success of microservices in the industry, there is no specific standard or research work for service providers as guidelines, especially from the perspective of basic microservice operations. In this work, we aim to bridge this gap between the industry and the academia and discuss different microservice deployment, discovery, and communication options for service providers as a means to forming complete service chains. In addition, we address the problem of scheduling microservices across multiple clouds, including microclouds. We consider different user-level service level agreements, such as latency and cost, while scheduling such services. We aim to reduce the overall turnaround time and costs for the deployment of complete end-to-end service. In this work, we present a novel affinity-based fair weighted scheduling heuristic to solve this problem. We also compare the results of the proposed solution with standard greedy scheduling algorithms presented in the literature and observe significant improvements.

## 1 | INTRODUCTION

With the explosion of online services and mobile and sensory devices, the demand for new services and consequently data traffic is growing rapidly. The popularity of Internet of Things (IoT) and 5G networks have contributed significantly to this trend, with millions of new sensing devices and online services exchanging data. According to Wireless World Research Forum (WWRF), the number of connected wireless devices is expected to be 100 billion by 2025.<sup>1</sup> Cloud computing has been considered as a major enabler for such novel networking paradigms.<sup>2</sup> The online services, sensing devices, and end-users are generally spread across geographically distributed areas. This motivates the application service providers (ASPs) and Internet service providers (ISPs) to deploy the services over multiple clouds for scalability, redundancy, and

\*This is an extended version of a paper presented at IEEE ICC 2017. We have extended the work significantly (50% changes) for submission to Journal of Transactions on Emerging Telecommunications Technologies (ETT).

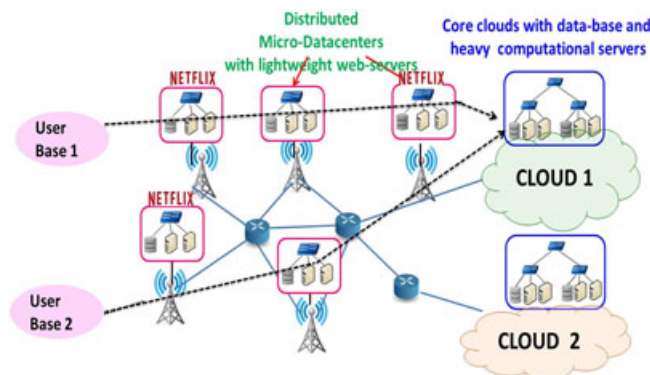
quicker response to the users.<sup>1,3,4</sup> The increasing use of virtualization technologies also helps ASPs and ISPs to deploy their services over standard high-volume infrastructures to accommodate such high volume of user demands.

Large services, which were monolithic software in the past, are being replaced by a set of lightweight services called microservices (MSs),<sup>5,6</sup> which are being deployed in distributed virtualized environments. Monolithic applications are complex, hard to scale, and difficult to upgrade and innovate. On the contrary, MSs, where the functionalities of the application are segregated, are lightweight and easy to deploy and scale. Instead of building a single, monolithic application, the idea is to split the application into a set of smaller, interconnected services, called microservices (or simply services).<sup>6,7</sup> Such services are lightweight and perform distinct tasks independent of each other. Hence, they can be deployed quickly and independently as user demands vary. The MSs can be easily upgraded or scaled without affecting much of the other functionality. Spreading MSs across multiple clouds allows having ASP's points-of-presence close to the distributed mobile users. The services are then chained through a process called service function chaining (SFC)<sup>8</sup> to create a complete end-to-end service. The goal is to enable the traffic to flow smoothly through the network, resulting in an optimal quality of experience to the users.

Microservices can be easily deployed over physical machines (PMs) or virtual machines (VMs) using novel techniques such as containers, allowing service providers to easily deploy, scale, and load balance their applications.<sup>5-10</sup> Microservices deployed using containers benefit from lower maintenance, lower costs, and more scalability as compared to directly deploying the VMs. The ASPs such as Google, and Netflix are already using containers extensively. The ISPs like AT&T and international communities like IETF are actively proposing the use of MSs to bolster software-defined networking (SDN) and network functions virtualization (NFV) efficiency.<sup>11-13</sup> For example, AT&T has started an “*open container initiative*”.<sup>11</sup> The users benefit from the quick response and lower costs, whereas ASPs and ISPs benefit from quicker and cheaper deployment options. Microservices are usually scaled dynamically depending on the user demands. Many service providers are opting for MSs to deploy their services. With the advancements in the virtualization technology, the MSs are being deployed over VMs as well. The ASPs and ISPs send requests to cloud service providers (CSPs) and obtain the resources to deploy the MSs as per their requirements at the time.<sup>3</sup> We discuss more about MSs in Section 3.

Another recent trend is the movement of MSs from host-centric to data-centric model in which the computational resources move closer to the end users. This results in further reduction in response time to the end users and lower costs to ASPs and ISPs because of shorter access links. This has led service providers to the concept of microclouds at the cellular base stations.<sup>3,12</sup> The technology is called as micro edge computing. A sample scenario is demonstrated in Figure 1. We consider the example of *Netflix*, a multinational entertainment content provider, which specializes in streaming media and video on demand. As a result of an explosion in mobile devices,<sup>1</sup> *Netflix* would benefit from locally relevant content cached at microclouds served to users through an MS. This will reduce the user latencies and result in a better user experience. In addition, it may result in lower operational expense (OpEx) to *Netflix* by reducing the usage of expensive wide area network (WAN) bandwidth.

Due to the nature of the contemporary telecommunications applications, the services need to be highly available, almost as much as 99.999%.<sup>12</sup> Additionally, most of the contemporary applications are sensitive to the delays, jitter, and packet-loss (such as online games, healthcare applications, video streaming and others). Many of these services are required to support millions of subscribers and meet the rigorous performance standards.<sup>12,14</sup> Proper scheduling of these services is important for reducing total delays, total required resources, and overall deployment costs.<sup>13</sup> These requirements mandate the optimal placement and scheduling of the service instances and proper interconnection among them.



**FIGURE 1** Microclouds at the base station for quicker response

Although the VM placement problem has already been studied in other works,<sup>10,15-21</sup> the MS architecture and its scheduling are a relatively novel problem. The instances of the MSs are generally short lived and dynamic in nature. Researchers are working on innovative schemes to design efficient algorithms for appropriately placing and scheduling the services,<sup>9,12</sup> splitting the load across instances on multiple clouds, and chaining them to improve performance parameters. However, we argue that there is a lack of research work in the domain of MS scheduling across multiple clouds for optimal SFCs, for both ASPs and ISPs.<sup>6</sup>

Despite the widespread acceptance of the MSs in the industry, there is a huge gap between the industry and the academia in this field.<sup>6,22-24</sup> Hence, in this work, we aim to bridge this gap by discussing the MSs and some of the options for deployment and discovery of MSs and the communication among different instances of MS. In addition, we formally discuss the problem of scheduling MSs. The service providers may benefit from such work to understand the limitations, challenges, and advantages of this novel networking architecture. Such work may motivate ASPs and ISPs to leverage the advantages of MS and multicloud platforms.

The rest of the paper is organized as follows. In the next section, we discuss the state-of-the-art MSs and the scheduling problem in the SFC context to show the limitations of existing approaches. In Section 3, we discuss the options for

**TABLE 1** List of Acronyms

<i>Acronym</i>	<i>Description</i>
API	Application program interface
ASP	Application service provider
CAPEX	Capital expenditures
CSP	Cloud service providers
DB	Database
DPI	Deep packet inspector
EC2	Elastic Compute 2
ESB	Enterprise service bus
FWS	Fair weighted affinity-based scheduling
IaaS	Infrastructure as a service
IoT	Internet of Things
ISP	Internet service provider
IT	Information Technology
LFDT	Least-full first with decreasing time
LFFF	Least-full first with first finish
MFDT	Most-full first with decreasing finish
MFFF	Most-full first with first time
MORSA	Multi-objective resource scheduling
MS	Microservice
NFV	Network function virtualization
OF	OpenFlow
OPEX	Operational expenses
OSGi	Open Service Gateway Initiative
PM	Physical machine
REST	Representational State Transfer
SDN	Software-defined networking
SLA	Service level agreement
SFC	Service function chaining
SOA	Service-oriented architecture
VF	Virtual function
VM	Virtual machine
VNF	Virtual network function
WAN	Wide area network
WWRF	Wireless World Research Forum

deployment and discovery of MSs and the communication among different instances of the MSs to form end-to-end service chains. Section 4 formalizes the MS scheduling problem. In Section 5, we propose a novel fair weighted scheduling (FWS) algorithm for MS scheduling and explain the experimental setup, and in Section 6, we present the comparison results. Finally, Section 7 concludes the paper. A list of the acronyms used throughout this paper is given in Table 1.

## 2 | RELATED WORK

Microservices are being extensively used in the industry. However, the research community lacks the intensity with respect to various MS aspects. There are only a few works<sup>25-30</sup> that discuss the MSs. Most of the works on MSs are available on blogs and online communities, mostly in a scattered manner.<sup>5,16</sup> Dmitry et al<sup>6</sup> discussed the MS platform architectures in brief. Balalaie et al provide the MS architecture for clouds.<sup>25-32</sup> Newman discussed different options for building MSs.<sup>12</sup> Garderen also discussed the MS architecture in brief.<sup>24</sup>

Recently, researchers have been studying SDN and NFV integration with MSs as well. Jahromi et al<sup>33</sup> leveraged the NFV and MS architectural style to propose an architecture for on-the-fly CDN component, provisioning to tackle issues such as flash crowds. In the proposed architecture, CDN components are designed as sets of MSs, which interact via RESTful Web services and are provisioned as virtual network functions, which are deployed and orchestrated on-the-fly. Luong et al<sup>34</sup> presented an MS platform to target the flexibility of telecom networks and the automation of its deployment. Using Docker orchestration, this demo paper shows the flexibility and the rapid deployment of wireless network infrastructure. In the work of Soenen et al,<sup>35</sup> authors introduced tunable and scalable mechanisms that provide NFV MANO with high availability and fault recovery using MSs. Fazio et al<sup>36</sup> argued that developers can engineer applications that are composed of multiple lightweight, self-contained, and portable runtime components deployed across a large number of geo-distributed servers and discussed open issues in MS scheduling. García-Pérez and Merino<sup>37</sup> and Farris et al<sup>38</sup> investigated further into fog computing to reduce the latencies in LTE and 5G networks, respectively, where MSs can be a candidate solution for service deployment. Yaseen et al<sup>39</sup> and de Brito et al<sup>40</sup> discussed leveraging fog computing and SDN for the security of the mobile wireless sensor networks and smart factories.

The MS architecture is being implemented by large enterprises such as banks, financial institutions, and global retail stores, to build their services in an incremental, flexible, and cost-effective manner. Recently, there has been a trend to use containers to deploy MSs across geographically distributed clouds. Containers are the lightweight version of the virtual machines. They are gaining significant traction in the industry recently since they are lightweight as compared to VMs. They can be easily downloaded and quickly deployed.<sup>32,41</sup> Platforms such as Docker and Solaris Zones<sup>27,42</sup> are available to ASPs for the deployment of their services using containers.<sup>43</sup> Researchers have identified the importance of MSs as an enabler for novel networking paradigms such as IoT and 5G. They have started identifying and addressing various problems in this context.<sup>5,6</sup>

One important problem in the context of MSs is their scheduling over the available and scattered resources to form complete SFCs. Optimal scheduling of MSs is necessary for faster deployments and minimum expenses to the service providers and better quality of experience to the end users, such as lower latencies. The problem of placing and scheduling the virtual functions (VFs) has been actively pursued in the industry and academia for years. However, researchers argue that the problem needs to be revisited from the perspective of MSs over service chains, as SFCs has some unique features.<sup>4</sup> For example, SFC is an ordered chain of services so the order in which the service instance needs to be visited is defined dynamically by the traffic flows.<sup>7,8</sup> The SFC scheduling problem has been in focus recently and other works<sup>9-11,14-16,44-48</sup> provide a wide range of VM scheduling strategies in a single cloud or across multiple clouds forming efficient SFCs.

Due to the time-sensitive nature of contemporary applications, VM placement alone is not sufficient to yield acceptable performance in the deployment of MSs over microclouds. Especially from the perspective of the short-lived MSs, scheduling is more important than the placement problem. Furthermore, mobile users have strict SLAs as far as tariffs and delays are concerned. This mandates ASPs to create points of presence close to the mobile users, reducing access latency and the overall cost. Merely efficiently placing the MSs is not sufficient to obtain optimal results. Recently, researchers have become aware of the importance of scheduling problem for MSs in SFCs, especially for the microclouds at the edges to guarantee carrier-grade performance.<sup>44</sup> However, there is a dearth of research works that address the scheduling problem in the context of MSs.

In this work, we propose a novel fair weighted affinity-based scheme for scheduling MSs and compare results with four different variants of the greedy strategies, which are common in the literature. We show significant improvements

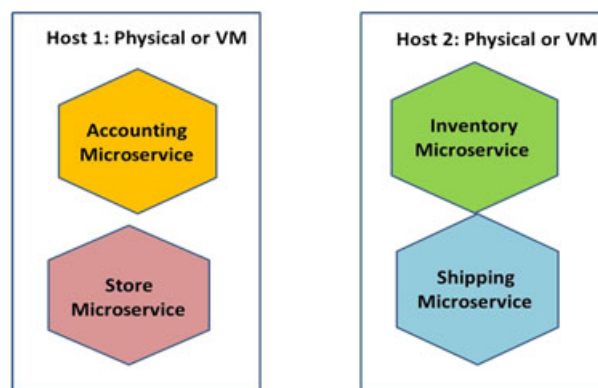
with the proposed heuristic. In the next section, we discuss the options for deployment, the discovery of MSs, and the communication among the different instances of the MSs.

### 3 | MICROSERVICES AND OPTIONS

As defined in the works of Namiot and Sneps-Sneppé<sup>6</sup> and Bhamare et al.,<sup>20</sup> the MS architecture is a specialization of an implementation approach for service-oriented architectures (SOAs) used to build flexible and independently deployable software systems. Generally, software applications become easier to build and maintain when they are divided into smaller pieces, which cooperate to perform one particular complex task. For this discussion, we consider the example of an ASP who provides e-commerce-based services. This may apply to ASPs such as Netflix, Uber, and Amazon, who provide their services through the Internet. A single user request for some online service may comprise of a set of functionalities, which are accounting, storage, inventory, and shipping.<sup>5,16</sup> Each such functionality may be deployed as an MS (Figure 2). The point to be noted here is that the scope of the MS architecture is not only limited to the ASPs but also is equally important for the transport services, multimedia services, and network services.<sup>12</sup>

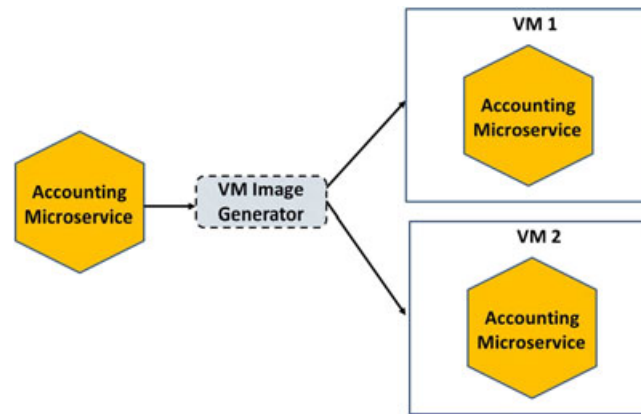
Microservices communicate with each other to provide the desired functionality to end users. For example, the user request for online service or product has to travel through the MS managing inventory, then accounting and storage, and finally shipping to complete the order. Microservices perform the allocated task and then exchange the related data to update the other MSs. For example, in this case, inventory MS will update the accounting MS regarding the total quantities ordered of the goods or services to prepare the bill. The same task may be performed using traditional monolithic services or even VMs. However, MSs are more agile, lightweight, and easy to scale up or down as per the user demands vary. In the remaining section, we discuss various ways to deploy different ways in which MSs can be deployed to form complete service chains for an ASP's services.

1. **MS deployment.** With the advent of the virtualization technology, many MS deployment options are becoming available to the service providers. In the following, we discuss these options and the pros and cons associated with each one.
  - a. *Multiple service instances per host.* This is the simplest way of MS deployment, where multiple MS instances are deployed on a single host. Though the host could be a PM or a VM, PMs are preferred in this simpler way of deployment. This scheme benefits from the high resource utilization. However, it suffers from some significant drawbacks. The major drawback is that there is little isolation for different service instances. A single service instance may consume a significant amount of resources starving other service instances. In addition, security becomes a major threat in such an environment. This option is shown in Figure 2.
  - b. *Single service instance per host.* In this approach, a separate host is selected for each MS. Generally, VMs are selected as hosts. The VM types are selected as per the system requirements of the service. Multiple hosts such as VMs are then deployed over a single or multiple servers as per the capacity and other constraints. Recently, this is the primary approach used by a majority of the ASPs. The CSPs, such as Amazon, make the resources available through the Infrastructure as a Service (IaaS) model. The benefit of this approach is that each service instance is executed



**FIGURE 2** Multiple service instances per host (generally physical machine). VM, virtual machine

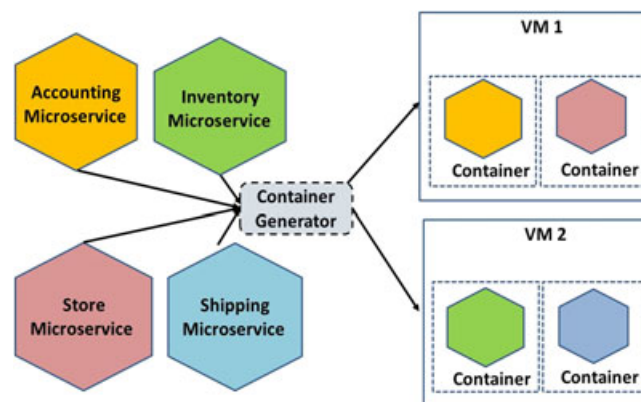




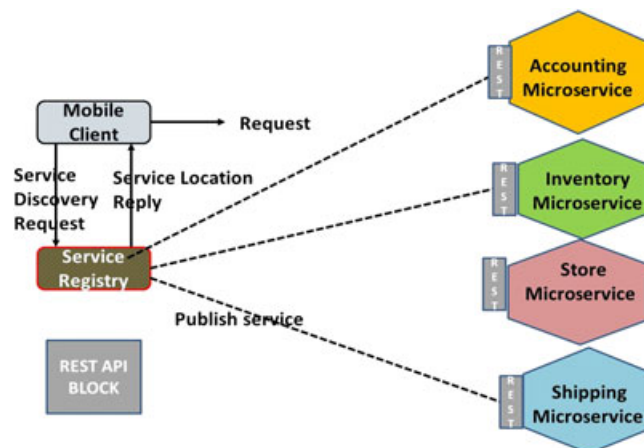
**FIGURE 3** Single service instance per host (generally virtual machine (VM))

within a completely isolated environment. It has a fixed amount of CPU and memory and does not have to share resources with other MSs. With this model, ASPs can leverage mature cloud infrastructure. Single service instance per VM deployment is shown in Figure 3 with account service as an example. However, since VMs are available in fixed sizes, it is possible that some VMs will be underutilized. Moreover, shutting down or restarting a particular VM instance, as user demands vary, is time consuming and may affect the user-latencies adversely.

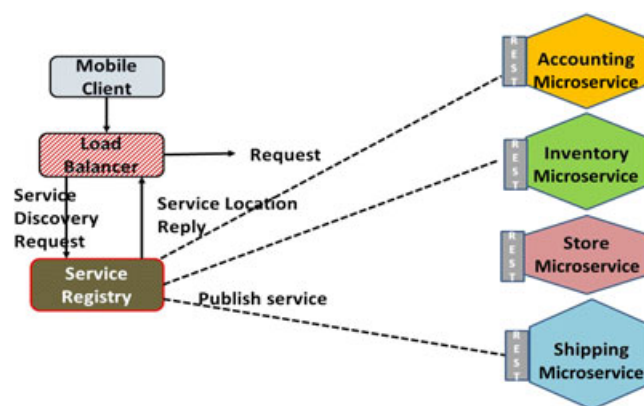
- c. **Single service instance per container.** Containers are the lightweight version of the VMs. A container is an OS-level virtualization to deploy and run applications without launching an entire VM and is suitable for the deployment of the MSs. A container consists of the libraries required to run the entire MS. A container may host multiple MSs. More details about the containers may be found in other works<sup>23,29,30</sup> and online resources.<sup>5,31</sup> Containers are gaining significant traction in the industry recently because they are lightweight as compared to VMs. They can be easily downloaded and quickly deployed. Containers may be deployed over PMs or VMs quickly as per the choice of the service providers. The platforms such as *Docker* are generally used for the deployment of containers.<sup>27</sup> The MS deployment option using containers is shown in Figure 4.
2. **Discovery of MSs.** Once the MSs are deployed, the next important stage is the discovery of such services for the end users so that the user requests can be guided through the underlying network properly. In an SOA, the interservice communication among the service instances and service discovery module are implemented with an enterprise service bus (ESB).<sup>49</sup> For the efficient discovery of MSs, various platforms such as Kubernetes and Marathon,<sup>31</sup> which implement the service registry, have been developed in the industry. A service registry is nothing but a database (DB) of available service instances. Service registry module translates user requests into the appropriate message types and routes them to the appropriate provider by enabling users to interconnect with the different services. Examples of the service registry are *Apache Zookeeper*, *Netflix Eureka*, and others. The *OSGi* architecture<sup>49</sup> also provides a similar platform for service registry, discovery, and deployment of the services. These specifications enable a



**FIGURE 4** Single service instance per container. VM, virtual machine



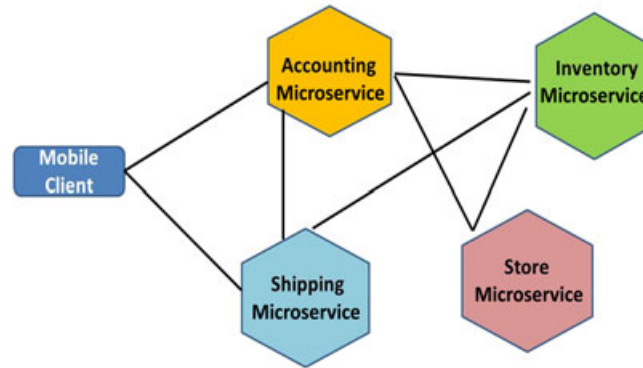
**FIGURE 5** Client-side service discovery. API, application program interface; REST, representational state transfer



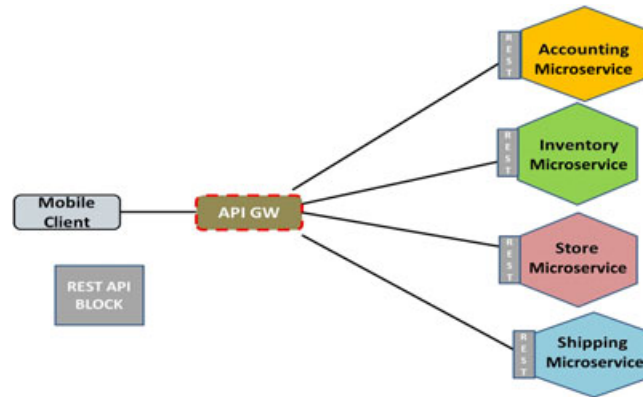
**FIGURE 6** Server-side service discovery

development model where applications are composed of many different reusable components. Contemporary service discovery architectures can be divided into two types, which are discussed as follows.

- a. *Client-side discovery.* In this approach, the client or the API-GW is responsible for obtaining the location of a service instance by querying a service registry. The user is also responsible for load balancing among the service instances. A sample client-side discovery model is shown in Figure 5. We assume that the MS instances implement representational state transfer (REST) APIs<sup>50,52</sup> for the communication purpose. The service provider is responsible for implementing the service registry; however, it is the responsibility of the clients or end users to determine the location of the MS instance from the service registry. *Netflix OSS* is a good example of client-side discovery model.<sup>31</sup> Though this model is simple to implement, it couples the client code with the service registry.
  - b. *Server-side discovery.* With this approach, clients/API-GWs send the request to a component, such as a load balancer, that runs in a well-known location. The load balancer is responsible for calling the service registry and determining the absolute location of the MS, as shown in Figure 6. That component has an entry for the port and IP address for the service registry. Amazon web service elastic load balancer (*AWS ELB*)<sup>31</sup> is a good example of a server-side discovery. The major advantage of this model is that the details of service registry are abstracted from the client.
3. **MS communication.** In monolithic applications, different components invoke one another using language-level function calls. In contrast, in MS-based applications, each service instance is typically a process. There are synchronous and asynchronous modes of communication among processes and MSs use combination of these interaction styles. Communication among various MS instances is important for a complete service to the end users. There should be a proper mechanism to guide the user packets through proper instances of the MSs in the given order.<sup>18</sup> For example, with the given hypothetical ASP in this work, the order of service instances through which the user request should



**FIGURE 7** Point-to-point MS communication



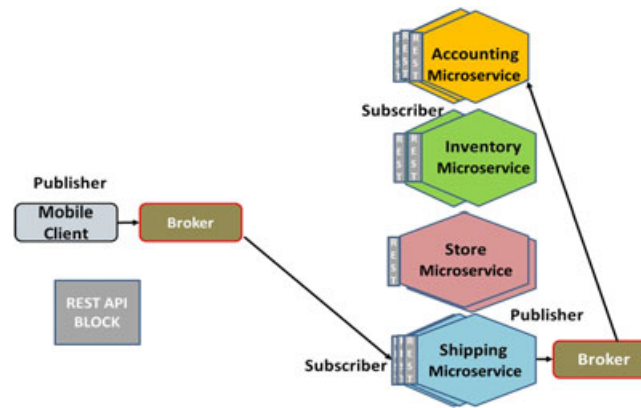
**FIGURE 8** Communication through application program interface gateway (API GW). REST, representational state transfer

be routed is (*inventory* → *account* → *shipping* → *store*). From the design perspective, the following communication options are available for the ASPs in their MS setup.

- Point to point.* This is the simplest approach in which each service instance directly communicates with another using the APIs such as REST. The user directly communicates with the first service instance, as shown in Figure 7. As the number of functionalities and instances of each subservice increase, the performance of systems based on point-to-point communication degrades. In addition, it gets unmanageable with a large number of service instances.
- Communication through the API gateway.* In this option, an application program interface gateway (API-GW) is installed in-between the end users and the MS instances, as shown in Figure 8. The API-GW has a well-known port-IP combination, to which, clients send the requests and API-GW forwards the requests to the appropriate service instance. The decision is taken based on parameters such as possible delay and load balancing. Service instances do not have to bother with the communication among each other, and they just forward their replies to the gateways. This is more scalable as compared to the previous option. However, a single gateway may become a single point of failure. This disadvantage can be easily eliminated by having multiple instances of such gateways for load balancing and redundancy (see Figure 8).
- Message broker style.* This is an asynchronous mode of communication. A given MS can be a message producer and can asynchronously send messages to a queue. On the contrary, the consuming MS takes messages from the queue. Such style of communication decouples message producers from message consumers and the intermediate message broker buffers messages until the consumer is able to consume or process them. Producer MSs are completely unaware of the consumer MSs, hence, are said to be in asynchronous communication.<sup>22,23,31</sup> The MS communication using message broker style is shown in Figure 9.

In the next section, we discuss one particular and important problem in the context of MSs, that is, scheduling of MSs. This is an important problem for optimal placement of service chains. Optimal scheduling is necessary to satisfy the SLAs and QoS to end users (such as minimum latencies) and the minimum expenses to the service providers while deploying





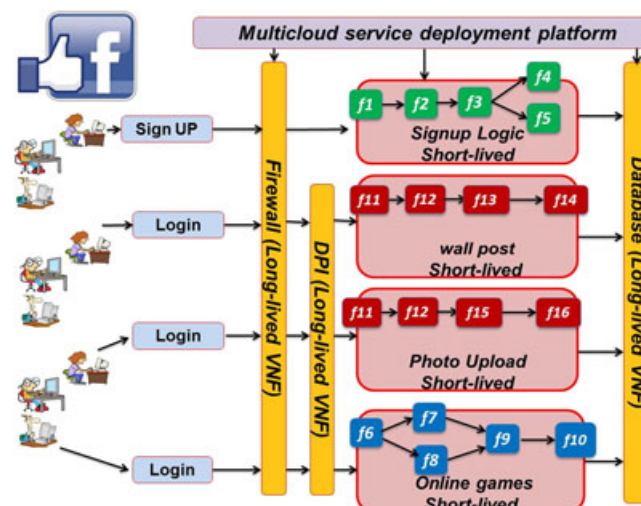
**FIGURE 9** Message-broker style communication option. API, application program interface; REST, representational state transfer

the instances of MSs over the available resources. As pointed out earlier, though the problem of placement of VMs is studied quite extensively in the literature, scheduling of MSs along with relevant MS options is severely under researched.

#### 4 | MICROSERVICES SCHEDULING PROBLEM

In this section, we discuss the MS scheduling problem in the context of constitution and placement of SFCs. The problem generally comprises three subproblems, ie, (1) selecting types and numbers of the service instances to be scheduled, (2) selecting PMs or VMs on which the services should be scheduled, and (3) deciding the time slot for which a particular service instance needs to be executed. Common heuristics used in the state-of-art systems for these tasks are “greedy with bias.”<sup>9,29,46</sup> The bias is toward some factor such as (1) select a service with earliest finish time or (2) select service with the longest execution time. Similarly, the bias while selecting VMs/PMs are (1) select most-loaded machine or (2) select least-loaded machine.<sup>14,16</sup> We start our discussion with a particular use case. We consider an ASP such as *Facebook (FB)* and take up a hypothetical example of the services offered by FB to explain the problem under consideration. It is important to note that the scope of the problem under consideration is not only limited to the application services but is equally important for the telecommunication services, multimedia services, and network services.<sup>7-9</sup> For example, the network-slicing problem for the network service providers.<sup>12,33</sup>

As shown in Figure 10, different groups of users from various user-bases may send different types of web requests to *FB* webserver(s). For example, some users may be interested in signing up for the service and others may log in to check their messages or posts on the wall or scan through their friend requests. The sign-up requests, after passing through the



**FIGURE 10** Service function chaining for different services offered by an application service provider (such as Facebook (*FB*)). DPI, deep packet inspection

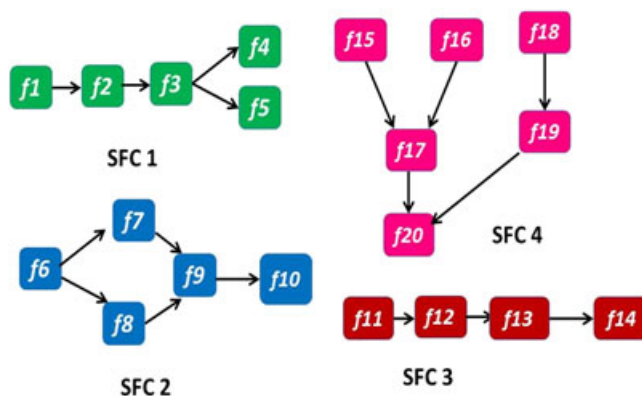
firewall, are passed to a set of services, which handle user registration logic (in this case,  $firewall \rightarrow f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_5 \rightarrow database$ ). However, login requests may have to be passed through deep packet inspection (DPI) in addition to the firewall to distinguish among user demands (such as wall post, photo upload, or online *FB* integrated games). A complete service chain may comprise of a combination of IT and telecommunication services. This example is just for an illustration purpose of the service flows, and it may be different in actual *FB* implementation of the services. The important point to be noted here is the dynamic formation of complex and hybrid service chains, which comprise a different set of MSs implemented at the application layer. Some of the long-lived services in the process such as the firewall, the DPI and the DB may stay for longer durations compared to other short-lived services such as function specific MSs. While placement would be enough for such long-lived services, the short-lived MSs need to be scheduled for efficient service chains.<sup>33</sup>

In this example, if we consider some specific functionality, such as user registration (sign-up), wall post on *FB*, or other integrated game applications, a specific set of service instances needs to be executed. Such sets of service instances may be switched on/off as user demands vary, especially at the microclouds, since the capacities are limited. Scheduling these service instances over the available resources is an important problem. In this work, we have considered four SFCs comprising 20 MSs in total. The SFC shapes and graphs are shown in Figure 11. Note that the topologies of the SFCs also indicate their execution order. For example, in SFC 1, service  $f_2$  has to be executed after  $f_1$ . This maybe because of the business logic dependence or some mandatory network traffic flow demand. For example, web-service logic handling service has to be executed before the service handling DBs; or firewall must be executed before the business logic.

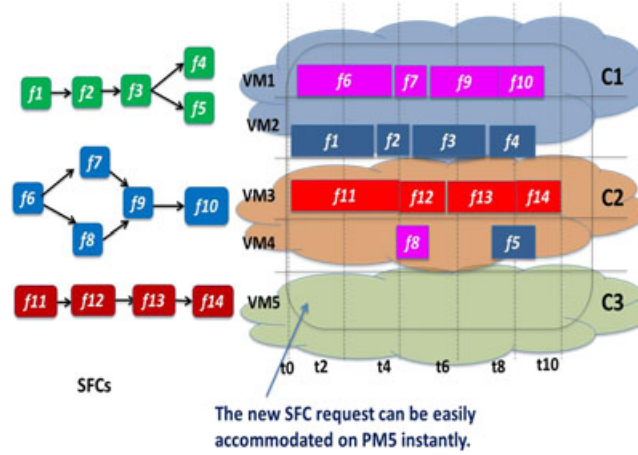
However,  $f_4$  and  $f_5$  may be executed in parallel after  $f_3$  because they are independent of each other. Similarly, in SFC 2,  $f_7$  and  $f_8$  may be executed at the same time after  $f_6$ . However,  $f_9$  has to be executed only after both  $f_7$  and  $f_8$  have finished their execution. This mandatory ordered flow of services in SFCs makes scheduling a complex problem. For the sake of simplicity, we assume that the VFs are visited in the numerical order. There may exist different service flows following different chains. However, the numbers for the VFs are in numerical order. For example, different chains consisting of different VFs may exist, such as (1, 2, 3, 4), (1, 2, 3, 5), (6, 7, 9, 10), and (15, 17, 20), as shown in Figure 11.

Let us now consider the scheduling problem of MS by considering three SFCs from the aforementioned example displayed on the left of Figure 12. On the right-hand side, we show the Gantt chart for the scheduling of the MSs over available resources using VMs ( $VM_1$  to  $VM_5$ ), deployed across three clouds  $C_1$ ,  $C_2$ , and  $C_3$ . Vertical lines indicate the time slots and each service needs different time to finish the execution. We assume that three user requests for these three SFCs arrive at the same time. The widths of the MSs indicate the total time needed to execute the services (longer services mean longer time for execution). A possible scheduling to optimize the total time and the resources required for the three SFCs on the available resources is shown in Figure 12.

We observe that, with optimal scheduling, all the executions finish before time-slot  $t_{10}$ , keeping  $VM_5$  free and ready to serve another incoming request. We argue that a sophisticated heuristic is needed to solve the large-scale MS scheduling problem within acceptable time limits. In the next section, we propose our novel affinity-based FWS scheme and explain the experimental setup. We implement all the four combinations of SFCs along with our proposed FWS approach. Our proposed novel heuristic performs scheduling of MSs on multiple VMs/PMs spread across multiple clouds. We consider different user-level SLAs, such as traffic affinity among services,<sup>9</sup> user delays, and cost constraints. In addition, we consider network parameters such as link loads and network traffic. We aim to reduce the overall turnaround time for the service and reduce the total inter-VM traffic generated.



**FIGURE 11** Four service function chainings (SFCs) with 20 virtual functions used for evaluation



**FIGURE 12** Gantt chart for optimal scheduling. SFC, service function chaining; VM, virtual machine

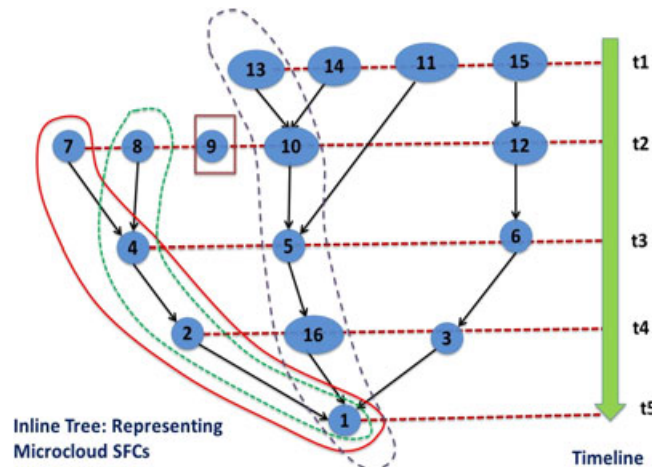
## 5 | HEURISTICS AND EXPERIMENTAL SETUP

In this section, we propose a novel affinity-based FWS for the scheduling problem under consideration. The heuristic can be divided into two distinct parts, ie, (1) selection of next service instance to be scheduled and (2) selection of next machine (VM or PM) on which the service instance should be scheduled. Heuristic starts at the time  $t = t_0$ . User requests arrive dynamically with interarrival time exponentially distributed, ie, the arrival process is *Poisson*.<sup>14,54</sup> Let  $U$  be the set of users waiting for the service or being served at any time  $t$ . Initially, we prepare the graphs for each SFC for each user  $u$  in  $U$ . It is to be noted that the graph may have disjoint sets of subgraphs.

A sample inline service graph is shown in Figure 13. Solid, dotted, and dashed lines highlight three possible service chains (there may be several other SFCs as well). In addition, the users may demand a single functionality, such as  $F_9$  shown in the figure. Again, these graphs can be of any shape and size, depending on the service provided by a specific ASP and the types of end-user demands. We have used various resource combinations (from Amazon EC2<sup>48</sup>) mentioned in Table 2 to simplify configurations so that resource requirements can be easily mapped to the nearest available configuration. Depending on the user resource demands, a particular VM is chosen from Table 2 such that the requirements are the closest match. Initially, we assign labels to the services using the Coffman-Graham algorithm.<sup>10</sup>

It ensures that the service instance that needs to be executed first for the particular SFC (starting service) gets a priority as per the arrival time. The service instance with the highest value of the label is scheduled first. Furthermore, we assign weight  $w$  to the services, such that:

- $w \propto$  (number of dependent services in that chain); and
- $w \propto$  (time spent by the services in the waiting queue).



**FIGURE 13** An inline graph for services forming different service function chainings (SFCs)

**TABLE 2** Resource configuration taken from Amazon EC2

Name	API Name	Memory	Cores	Max Bandwidth	Hourly Cost
T2 Small	t2.small	2.0 GB	1	25 MB/s	0.034
T2 Medium	t2.medium	4.0 GB	2	25 MB/s	0.068
T2 Large	t2.large	8.0 GB	2	25 MB/s	0.136
M4 Large	m4.large	8.0 GB	2	56.25 MB/s	0.14

Abbreviations: API, application program interface.

If there are ties between two services for scheduling (that is, services having the same labels), the service with higher weight is selected. This step ensures fair scheduling as it makes sure that the longer SFCs and the SFCs, which have waited longer in the queue, get a fair chance for their scheduling.

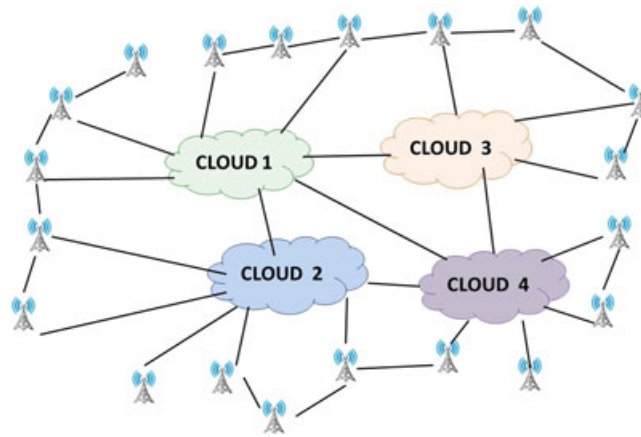
While selecting the VMs/PMs for service deployment, the affinity between services is taken into consideration. Two services belonging to the same instance of an SFC are considered to have higher affinity, and we try to place them on the same machine. This ensures minimum delays and less intermachine traffic overhead. This step ensures that the services for the same SFC are scheduled on the same machine, if possible, to minimize the total traffic generated. Otherwise, it tries to schedule the service on the machine with which intermachine traffic will be minimized, and all capacity constraints are satisfied. We may combine two or more services and deploy them on a single machine as well, provided a machine of that capacity is available. The availability of the machines depends on the cloud capacity. If a service instance is not serving any user demands, it is buffered in the cloud. In the buffered stage, the service uses fewer resources (such as storage only to save the state). However, it can be brought up quickly whenever relevant user demand arrives, saving resources and time.<sup>14</sup> For simplicity, we assume that clouds have infinite buffering capacity. The steps for the FWS algorithm are given in detail in Table 3.

We consider a 20-node topology out of which 16 are the microclouds deployed at the edges, such as cellular base stations, closer to the end users and four are core public clouds, with larger capacities, as shown in Figure 14. Computation and/or data intensive services that need more processing and/or storage capacities and that tend to run for longer times, such as firewall and DB services, are generally deployed at core clouds. We assume that each service instance produces data in the range of 5 kB to 20 kB. Moreover, the number of user requests each MS instance can handle at average load is selected from a range of 20 to 100 requests/sec. Time needed for execution of each service is chosen from the range of 10 to 100 milliseconds (ms).<sup>44</sup>

All the values are selected randomly from the given ranges. In addition, we assign each user request with some delays and cost constraints it may tolerate. We also make sure these constraints are satisfied while scheduling the MSs on the

**TABLE 3** Fair weighted scheduling algorithm for microservice (MS) scheduling<sup>33</sup>

1. Let $\{1, 2, 3, 4, \dots, N\}$ be the set of MSs to be scheduled on $M$ machines. Let $\{T_1, T_2, T_3, \dots, T_n\}$ be their finish times.
2. If $T_i < T_j$ , then MS $j$ is said to be immediate successor of task $i$ .
3. Let $S(i)$ be the set of all immediate successors of MS $i$
4. Let $L_i$ be the label assigned to MS $i$ .
5. Choose MS $i$ from the arrived request s.t. $S(i) = 0$ . Let $L_i$ be 1.
6. For $l = 2$ to $N$
7. Let $C$ be the set of unlabeled MS s.t. there is no unlabeled successor.
8. Let $s$ be the MS in $C$ s.t. $T_s < T_{s^*}$ for all other MS $s^*$ in $C$
9. Let $L_s = l$
10. Once labels are assigned, we assign weights $\{w_1, w_2, w_3, \dots, w_n\}$ to the services, s.t.: $w_i \propto$ (number of dependent services in that chain) and $w_i \propto$ (time spent by the services in the waiting queue).
11. Foreach service $i$
12. if $l_i = l_{i+1}$
13. select $i$ for scheduling if $w_i > w_{i+1}$
14. else select $i + 1$
15. Select the machine from the sorted list as per the remaining capacity for deployment.



**FIGURE 14** 20-node topology with 16 microclouds and four core clouds

clouds. In the next section, we compare the results of the proposed FWS solution with four variants of standard biased greedy scheduling strategy, which are common in the literature and observe significant improvements.

## 6 | RESULTS AND ANALYSIS

We now present the results obtained through the experimental setup. In addition to our FWS approach, we have implemented four additional algorithms based on the greedy biased approach for comparison. Service labeling step is common for all the heuristics. Table 4 displays the basic steps for the following strategies:

1. least-full first with first finish (LFFF);
2. most-full first with first finish (MFFF);
3. least-full first with decreasing time (LFDT);
4. most-full first with decreasing time (MFDT).

We execute the algorithms for a certain number of times, and then we take an average. Graphs in Figure 15 show the comparison of the four approaches mentioned earlier and our FWS approach in terms of the total inter-VM traffic generated. The FWS approach (thick yellow line) performs the best with the least inter-VM traffic. For example, with 3000 user demands, greedy algorithms produce more than 20 MB of data, whereas FWS only produces less than 10 MB data, which is an improvement of 50%.

Similarly, Figure 16 shows the average turnaround time for each user where FWS again performs the best. For example, with 4000 user demands, FWS results in a turnaround time of less than 220 ms, whereas the other algorithms need around 330 ms. We also present bar charts for the aforementioned results, that is, for total traffic generated and average turnaround time in Figure 17 and Figure 18, respectively.

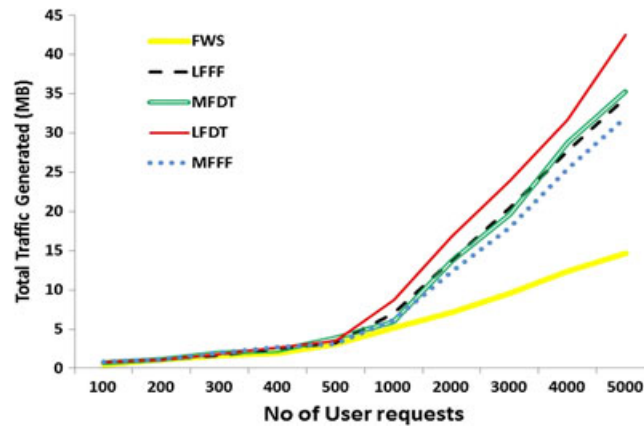
However, the average turnaround time alone is not sufficient to measure the performance, especially in the context of the time-sensitive applications. Most of the time, if the user demands are not satisfied within a given time constraint, it is as bad as service denied. Hence, we also find out the percentage of user demands, which got satisfied in the given time

**TABLE 4** The selection criterion for greedy biased heuristics

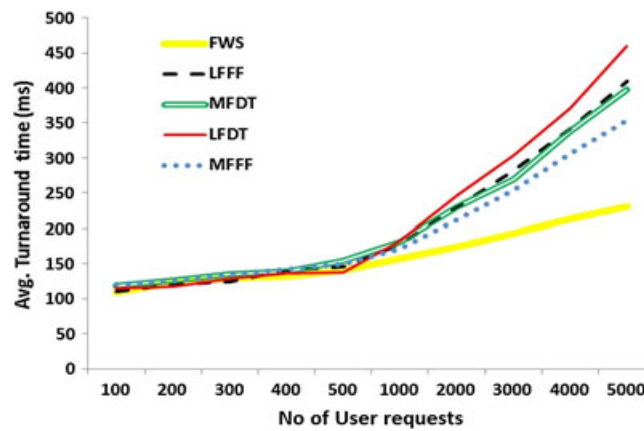
Microservice	Machine Selection	
	LF	MF
FF	Select Least full machine first, Select the service of SFC having first finish time	Select Most full machine first, Select the service of SFC having first finish time
DT	Select Least full machine first, Select the service of SFC with longest finish time	Select Most full machine first, Select the service of SFC with longest finish time

Abbreviations: SFC, service function chaining.

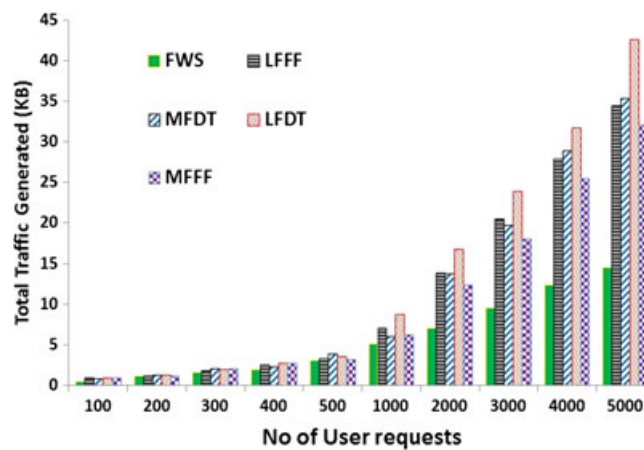




**FIGURE 15** Total traffic generated (in KB). FWS, fair weighted scheduling; LFDT, least-full first with decreasing time; LFFF, least-full first with first finish; MFDT, most-full first with decreasing time; MFFF, most-full first with first finish



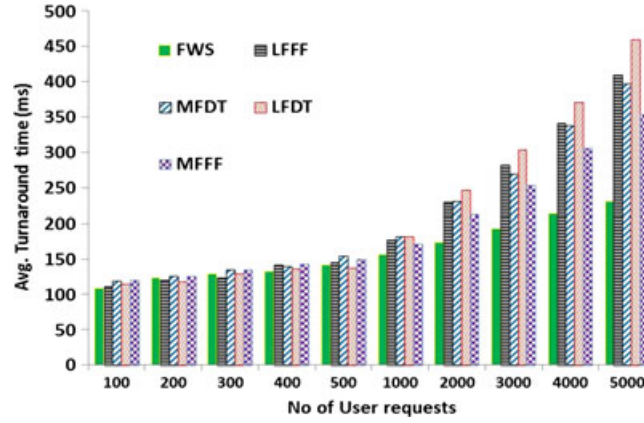
**FIGURE 16** Total turnaround time (in milliseconds). FWS, fair weighted scheduling; LFDT, least-full first with decreasing time; LFFF, least-full first with first finish; MFDT, most-full first with decreasing time; MFFF, most-full first with first finish



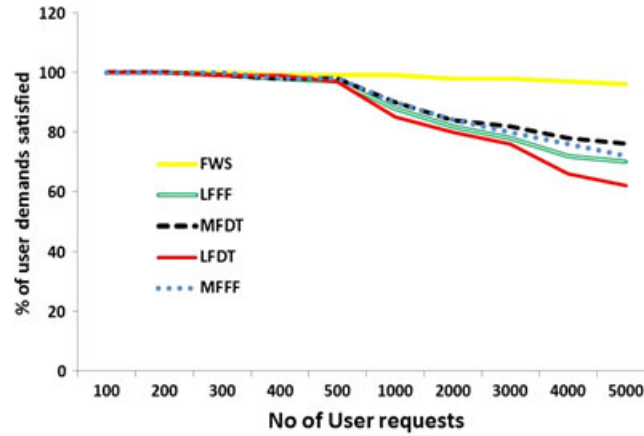
**FIGURE 17** Bar chart for total traffic generated (in KB). FWS, fair weighted scheduling; LFDT, least-full first with decreasing time; LFFF, least-full first with first finish; MFDT, most-full first with decreasing time; MFFF, most-full first with first finish

constraints (Figure 19). We observe that a significantly higher percentage of the user demands get satisfied with the FWS approach. The total percentage varies from 100% to 96% as user demands vary from 100 to 5000. On the contrary, the percentage drops to 70% for LFFF, 62% for LFDT and MFFF, and 74% for MFDT.

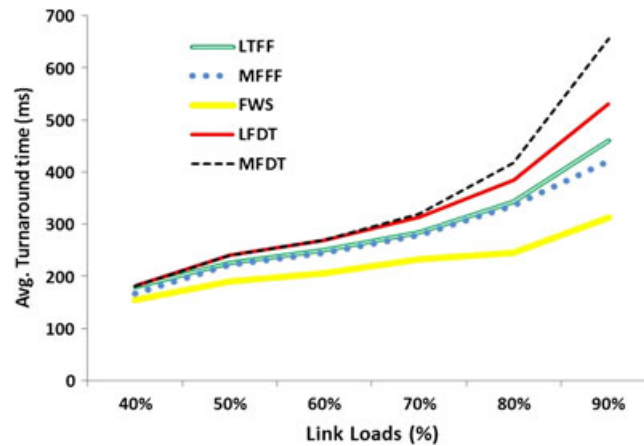
We have also analyzed the effect of traffic loads on average turnaround time or average time to schedule all the services. We observe the exponential growth in the total turnaround delays as traffic loads in the network grow. We generated dummy traffic to obtain different average traffic loads. The links were modeled as M/D/1 queues, and by the standard



**FIGURE 18** Bar chart for total turnaround time (in milliseconds). FWS, fair weighted scheduling; LFDT, least-full first with decreasing time; LFFF, least-full first with first finish; MFDT, most-full first with decreasing time; MFFF, most-full first with first finish



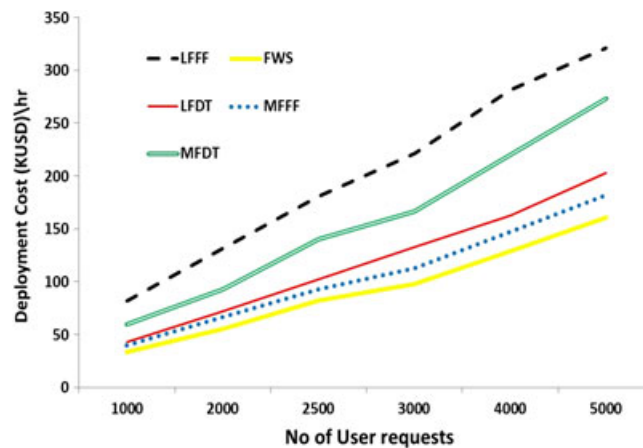
**FIGURE 19** Percentage of user demands satisfied. FWS, fair weighted scheduling; LFDT, least-full first with decreasing time; LFFF, least-full first with first finish; MFDT, most-full first with decreasing time; MFFF, most-full first with first finish



**FIGURE 20** Average turnaround time. FWS, fair weighted scheduling; LFDT, least-full first with decreasing time; LFFF, least-full first with first finish; MFDT, most-full first with decreasing time; MFFF, most-full first with first finish

formula, we calculate the delays in the links as given in Equation (1).<sup>54</sup> We note that  $T_{ij}$  is the total delay on the link  $(i, j)$ .  $\lambda_{ij}$  is the arrival rate of packets and  $\mu_{ij}$  is the processing rate of the same link.

$$T_{ij} = \frac{1}{2\mu_{ij}} \times \frac{2 - (\lambda_{ij}/\mu_{ij})}{1 - (\lambda_{ij}/\mu_{ij})} \quad (1)$$



**FIGURE 21** Cost comparison (fair weighted scheduling (FWS) vs. Greedy approaches). LFD, least-full first with decreasing time; LFFF, least-full first with first finish; MFD, most-full first with decreasing time; MFFF, most-full first with first finish

In Figure 20, we observe that even at 90% traffic load, the total delays with the proposed FWS scheme remain within the range of 250 ms, which is within acceptable limits for the contemporary real-time applications.<sup>56</sup> For other schemes, however, it varies from 400 to more than 600 ms. In Figure 21, we plot the graphs for the total costs of the resources needed to satisfy all the given demands using all the approaches. The cost has been calculated for an hour to host the required services for all the users.

We assume the Amazon pricing model, as shown in Table 2, to calculate the costs. We observe that the proposed affinity-based FWS approach performs better than the greedy approaches in terms of the total cost as well. The cost difference goes on increasing with an increase in the total number of users. This may be attributed to the fact that, in the affinity-based FWS approach, we try to accommodate the VMs, hosting MSs with affinity, on a single machine with the closest match for the required capacities. This reduces the required number of the resources and eventually the cost. From the results, we observe that the proposed FWS scheme outperforms the contemporary greedy approaches in terms of the total traffic overhead, total turnaround time, the total number of the services satisfied, and the total deployment cost.

## 7 | CONCLUDING REMARKS AND FUTURE WORK

In this paper, we have discussed the MSs and addressed important problems such as the deployment and discovery of MSs and the communication among the different instances of the MSs to form end-to-end service chains. In addition, we discussed the problem of scheduling MSs. We pointed out that this is an important problem to be addressed for optimal service chains and pointed out the gap between the work done for the VM placement problem and the MS scheduling problem. In addition, we pointed out that link loads and network delays while minimizing the total turnaround time and total traffic generated needs to be considered.

In this work, we aimed to bridge the gap between the academia and the industry to help the service providers to deploy the MSs more efficiently. In addition, we proposed a novel FWS approach for MS scheduling in the multicloud scenario to form optimal SFCs. We took into account different delay and cost related SLAs. Furthermore, we considered link loads and network delays while minimizing the total turnaround time and total traffic generated. The proposed approach demonstrates significant improvement compared to the standard biased greedy approaches. However, there is still a wide area open for the research in developing novel scheduling algorithms considering the different delay and cost related SLAs. Advancements in the field of machine learning may be applied, such as proactive scheduling. The MS architecture brings in more challenges, such as distributed data management, failure recovery, security, monitoring, network latency, message formats, load balancing, and fault tolerance, which need to be investigated further.

## ACKNOWLEDGEMENT

This publication was made possible by the National Priorities Research Program award [NPRP 8-634-1-131] from the Qatar National Research Fund (a member of The Qatar Foundation). The statements made herein are solely the responsibility of the author(s).

## ORCID

Deval Bhamare  <http://orcid.org/0000-0002-8925-6859>

## REFERENCES

1. Sørensen LT, Skouby KE, Dietterle D, Jhunjunwala A, Fu X, Wang X. User Scenarios 2020: A Worldwide Wireless Future. <http://www.wwrf.ch/files/wwrf/content/files/publications/outlook/Outlook4.pdf>. July 2009. Accessed December 14, 2017.
2. Miorandi D, Sicari S, De Pellegrini F, Chlamtac I. Internet of things: vision, applications and research challenges. *Ad Hoc Netw.* 2012;10(7):1497-1516.
3. Zhang Q, Cheng L, Boutaba R. Cloud computing: state-of-the-art and research challenges. *J Internet Serv Appl.* 2010;1(1):7-18.
4. Bhamare D, Jain R, Samaka M, Erbad A. A survey on service function chaining. *J Netw Comput Appl.* 2016;75:138-155.
5. Indrasiri K. Microservices in Practice: From Architecture to Deployment. <https://dzone.com/articles/microservices-in-practice-1>. Accessed December 14, 2017.
6. Namiot D, Sneps-Snepp M. On micro-services architecture. *Int J Open Inf Technol.* 2014;2(9):24-27.
7. Kecskemeti G, Marosi AC, Kertesz A. The ENTICE approach to decompose monolithic services into microservices. Paper presented at: IEEE International Conference on High-Performance Computing & Simulation (HPCS); 2016; Innsbruck, Austria.
8. Halpern J, Pignataro C. Service function chaining (SFC) architecture. RFC 7665. <https://tools.ietf.org/html/rfc7665>. Published October 2015. Accessed December 14, 2017.
9. Mouat A. *Using Docker: Developing and Deploying Software with Containers*. Sebastopol, CA: O'Reilly Media Inc; 2015.
10. Mehraghdam S, Keller M, Karl H. Specifying and placing chains of virtual network functions. Paper presented at: IEEE 3rd International Conference on Cloud Networking (CloudNet); 2014; Luxembourg.
11. Open Container Initiative. <https://www.opencontainers.org/>. Accessed December 14, 2017.
12. Newman S. *Building Microservices*. Sebastopol, CA: O'Reilly Media Inc; 2015.
13. Yoshida M, Shen W, Kawabata T, Minato K, Imajuku W. MORSA: a multi-objective resource scheduling algorithm for NFV infrastructure. Paper presented at: IEEE 16th Asia-Pacific Network Operations and Management Symposium (APNOMS); 2014; Hsinchu, Taiwan.
14. Riera JF, Hesselbach X, Escalona E, Garcia-Espin JA, Grasa E. On the complex scheduling formulation of virtual network functions over optical networks. Paper presented at: IEEE 16th International Conference on Transparent Optical Networks (ICTON); 2014; Graz, Austria.
15. Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur J Oper Res.* 2007;177(3):2033-2049.
16. Wolff E. *Microservices: Flexible Software Architectures*. New York, NY: Pearson Education Inc; 2016. ISBN:978-0134602417.
17. Montesi F, Weber J. Circuit breakers, discovery, and API gateways in microservices. 2016. arXiv preprint arXiv:1609.05830.
18. Gupta L, Samaka M, Jain R, Erbad A, Bhamare D, Metz C. COLAP: a predictive framework for service function chain placement in a multi-cloud environment. Paper presented at: IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC); 2017; Las Vegas, NV.
19. Luizelli MC, Bays LR, Buriol LS, Barcellos MP, Gaspary LP. Piecing together the NFV provisioning puzzle: efficient placement and chaining of virtual network functions. Paper presented at: IFIP/IEEE International Symposium on Integrated Network Management (IM); 2015; Ottawa, ON.
20. Bhamare D, Samaka M, Erbad A, Jain R, Gupta L, Chan HA. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Comput Commun.* 2017;102:1-16.
21. Bhamare D, Samaka M, Erbad A, Jain R, Gupta L, Chan HA. Multi-objective scheduling of micro-services for optimal service function chains. Paper presented at: IEEE International Conference on communications; 2017; Paris, France.
22. Kratzke N. About microservices, containers and their underestimated impact on network performance. 2015. arXiv:1710.04049.
23. Thönes J. Microservices. *IEEE Softw.* 2015;32(1):116-116.
24. Garderen V. Archivematica: using micro-services and open-source software to deliver a comprehensive digital curation solution. In: Proceedings of the 7th International Conference on Preservation of Digital Objects; 2010; Vienna, Austria.
25. Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables DevOps: migration to a cloud-native architecture. *IEEE Softw.* 2016;33(3):42-52.
26. Stubbs J, Moreira W, Dooley R. Distributed systems of microservices using Docker and Serfnode. Paper presented at: IEEE 7th International Workshop on Science Gateways (IWSG); 2015; Budapest, Hungary.
27. Davies M, Gil G, Maknavicius L, Narganes M, Urdiales D, Zhdanova AV. m: ciudad: an infrastructure for creation and sharing of end user generated microservices. In: Proceedings of the Poster and Demonstration Paper Track of the 1st Future Internet Symposium (FIS), CEUR Workshop Proceedings, Vol 399; 2008; Vienna, Austria.
28. Kwok Y, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput Surv.* 1999;31(4):406-471.
29. Mohamed M, Yangui S, Moalla S, Tata S. Web service micro-container for service-based applications in cloud environments. Paper presented at: IEEE 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE); 2011; Paris, France.

30. IBM Cloud Products. <http://www.ibm.com/cloud-computing/products/hybrid-integration/enterprise-service-bus-esb/>. Accessed December 14, 2017.
31. Elastic Load Balancing. <https://aws.amazon.com/elasticloadbalancing/>. Accessed December 14, 2017.
32. Balalaie A, Heydarnoori A, Jamshid P. Migrating to cloud-native architectures using microservices: an experience report. Paper presented at: European Conference on Service-Oriented and Cloud Computing; 2015; Taormina, Italy.
33. Jahromi NT, Glitho RH, Larabi A, Brunner R. An NFV and microservice based architecture for on-the-fly component provisioning in content delivery networks. Paper presented at: 15th IEEE Conference in Consumer Communications & Networking (CCNC); 2018; Las Vegas, NV.
34. Luong DH, Thieu HT, Outtagarts A, Mongazon-Cazavet B. Telecom microservices orchestration. IEEE Conference on Network Softwarization (NetSoft); 2017; Bologna, Italy.
35. Soenen T, Tavernier W, Colle D, Pickavet M. Optimising microservice-based reliable NFV management & orchestration architectures. Paper presented at: IEEE 9th International Workshop on Resilient Networks Design and Modeling (RNDM); 2017.
36. Fazio M, Celesti A, Ranjan R, Liu C, Chen L, Villari M. Open issues in scheduling microservices in the cloud. *IEEE Cloud Comput.* 2016;3(5):81-88.
37. García-Pérez CA, Merino P. Experimental evaluation of fog computing techniques to reduce latency in LTE networks. *Trans Emerging Tel Tech.* 2017;29(4):e3201.
38. Farris I, Taleb T, Flinck H, Iera A. Providing ultra-short latency to user-centric 5G applications at the mobile network edge. *Trans Emerging Tel Tech.* 2017;29(4):e3169
39. Yaseen Q, Albalas F, Jararwah Y, Al-Ayyoub M. Leveraging fog computing and software defined systems for selective forwarding attacks detection in mobile wireless sensor networks. *Trans Emerging Tel Tech.* 2017;29(4):e3183
40. de Brito MS, Hoque S, Steinke R, Willner A, Magedanz T. Application of the fog computing paradigm to smart factories and cyber-physical systems. *Trans Emerging Tel Tech.* 2017;29(4):e3184
41. Viennot N, Lécuyer M, Bell J, Geambasu R, Nieh J. Synapse: a microservices architecture for heterogeneous-database web applications. In: ACM Proceedings of the Tenth European Conference on Computer Systems; 2015; Bordeaux, France.
42. Introduction to Microservices. [https://www.nginx.com/blog/introduction-to-microservices/?utm\\_source=building-microservices-using-an-api-gateway&utm\\_medium=blog](https://www.nginx.com/blog/introduction-to-microservices/?utm_source=building-microservices-using-an-api-gateway&utm_medium=blog). Accessed December 14, 2017.
43. Mijumbi R, Serrat J, Gorricho JL, Bouten N, De Turck F, Davy S. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. Paper presented at: 1st IEEE International Conference on Network Softwarization (NetSoft); 2015; London, UK.
44. Lucrezia F, Marchetto G, Risso F, Vercellone V. Introducing network-aware scheduling capabilities in OpenStack. Paper presented at: IEEE 1st International Conference on Network Softwarization (NetSoft); 2015; London, UK.
45. Lopez-Pires F, Baran B. Virtual Machine Placement Literature Review. Technical Report. San Lorenzo, Paraguay: Polytechnic School National University of Asuncion; 2015.
46. Xia M, Shirazipour M, Zhang Y, Green H, Takacs A. Network function placement for NFV chaining in packet/optical datacenters. *J Lightwave Technol.* 2015;33(8):1565-1570.
47. Lakkakorpi J, Sayenko A, Moilanen J. Comparison of different scheduling algorithms for WiMAX base station: deficit round-robin vs. proportional fair vs. weighted deficit round-robin. Paper presented at: IEEE Wireless Communications and Networking Conference (WCNC); 2008; Las Vegas, NV.
48. EC2Instances.info. Easy Amazon EC2 Instance Comparison. <http://www.ec2instances.info/>. Accessed December 14, 2017.
49. The Dynamic Module System for Java. <https://www.osgi.org/developer/architecture/>. Accessed December 14, 2017.
50. Verborgh R, Harth A, Maleshkova M, et al. Survey of semantic description of REST APIs. In: *REST: Advanced Research Topics and Practical Applications*. New York, NY: Springer; 2014:69-89.
51. Representational State Transfer. [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer). Accessed December 14, 2017.
52. Anderson C. Docker [software engineering]. *IEEE Softw.* 2015;32(3):102-c3.
53. Bhamare D, Jain R, Samaka M, Vaszkun G, Erbad A. Multi-cloud distribution of virtual functions and dynamic service deployment: open ADN perspective. IEEE International Conference on Cloud Engineering (IC2E) March 2015;299-304.
54. ITU-T Recommendation Y.1541. Network Performance Objectives for IP-Based Services 2011. <https://www.itu.int/rec/T-REC-Y.1541/en>. Accessed December 14, 2017.
55. Jain R, ed. In: *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York, NY: Wiley Interscience; 1991.
56. Riera JF, Escalona E, Batalle J, Grasa E, Garcia-Espin JA. Virtual network function scheduling: concept and challenges. Paper presented at: IEEE International Conference on Smart Communications in Network Technologies (SaCoNeT); 2014; Vilanova i la Geltru, Spain.

**How to cite this article:** Bhamare D, Samaka M, Erbad A, Jain R, Gupta L. Exploring microservices for enhancing internet QoS. *Trans Emerging Tel Tech.* 2018;e3445. <https://doi.org/10.1002/ett.3445>