Challenges and Solutions in Fog Computing Orchestration

Yuxuan Jiang, Zhe Huang, and Danny H. K. Tsang

Abstract

Fog computing, complementary to cloud computing, has recently emerged as a new paradigm that extends the computing infrastructure from the center to the edge of the network. This article explores the design of a fog computing orchestration framework to support IoT applications. In particular, we focus on how the widely adopted cloud computing orchestration framework can be customized to fog computing systems. We first identify the major challenges in this procedure that arise due to the distinct features of fog computing. Then we discuss the necessary adaptations of the orchestration framework to accommodate these challenges.

Fog Computing: Why, What and How

As we enter the era of the Internet of Things (IoT), there is a significant trend to connect ubiquitous smart end devices to the Internet for running diverse applications, such as smart metering, smart transportation, augmented reality (AR), and wearable computing. Facing typical limitations in computational resources and power supply (e.g., batteries and renewable energy) at the end devices, researchers have developed offloading middleware to migrate part of the application processing from the resource-poor end devices to powerful backend clouds [1].

Solely relying on backend clouds, however, cannot completely accommodate current IoT applications because of four important reasons: *First*, in view of the huge amount of dynamically generated data nowadays, it is impractical to transfer all the data from end devices, where they are created, to the backend cloud, where they are processed, due to bandwidth limitations and excessive transmission costs. *Second*, some users do not want their data to traverse a long distance from end devices to the cloud because of risks to privacy [2]. *Third*, the noticeable round-trip delay from end devices to the cloud can easily degrade the performance of delay-sensitive applications, such as AR, online gaming, social media, and video applications that require online analytics. *Finally*, for localized applications that dynamically adapt to local network states and user contexts, e.g., dynamic adaptive video streaming to mobile users, it is difficult for remote clouds to rapidly respond to local contextual changes. In fact, the offloaded application segments do not necessarily have to be executed in the cloud. Instead, edge facilities, such as off-the-shelf commercial routers and WiFi gateways, edge data centers, and home servers, can be leveraged to facilitate the processing. To this end, the concept of fog computing, in which some offloaded application segments are pushed to edge facilities that are closer to the end devices, has recently been proposed [3].

To handle the highly heterogeneous nature of fog nodes[1] that participate in a fog computing eco-system, a *universal orchestration platform* on top of the fog nodes is the technical enabler [4], which brings interoperability, software-programmability, and virtualization. Interoperability allows heterogeneous fog nodes to operate under the same architecture. Software-programmability eases the way for application developers to program based on general virtualized hardware, where low-level hardware details of fog nodes are shielded. Virtualization divides the resources on fog nodes into resource units, such as kernel-based virtual machines (VMs) and containers, shared by multiple IoT applications without mutual interference. In the rest of this article, we use VMs and containers interchangeably as the resource units in the resource provisioning process.

As a natural extension of cloud computing to the network edge, fog computing inherits major characteristics of cloud computing, including resource orchestration, elastic provisioning, and multi-tenancy. Interconnected fog nodes form a shared pool of reconfigurable computational resources. Resource-rich fog nodes can lease their computational resources out to execute IoT applications [3]. A well-designed orchestration framework that supports *affinity-aware offloading* enables an application to be partitioned into segments and offloaded onto different fog nodes. Therefore, segments processing delay-sensitive tasks can be offloaded onto closer edge facilities to meet the delay requirements. We take the cognitive assistance application [5] shown in Fig. 1 as an example to demonstrate affinity-aware offloading. The application assists people affected by cognitive decline by displaying context-aware real-time scene interpretation with AR on a Google Glass, based on captured first-person images. The Google Glass, as the end device, only runs a portion of the application. The remaining application segments are offloaded onto different in-network fog nodes.

In addition to the universal orchestration platform and affinity-aware offloading, the distinct features of fog computing listed below also present fresh research challenges and opportunities.

Large-Scale and Distributed: Fog nodes can be owned by separate parties at different geographic locations, forming a massive-scale computing network (a.k.a. a fog network). Therefore, the architecture that supports fog computing has to be scalable and give consideration to the individual preferences and security concerns of fog infrastructure owners.

Y. Jiang and D. H. K. Tsang are with The Hong Kong University of Science and Technology.

Zhe Huang is with Princeton University.

[1] In this article, fog nodes refer to the nodes in a fog computing communication network, including end devices, computing equipment in close proximity to end devices, and backend clouds. Fog computing focuses on offloading the computation from resource-poor fog nodes to resource-rich fog nodes.

Digital Object Identifier: 10.1109/MNET.2017.1700271

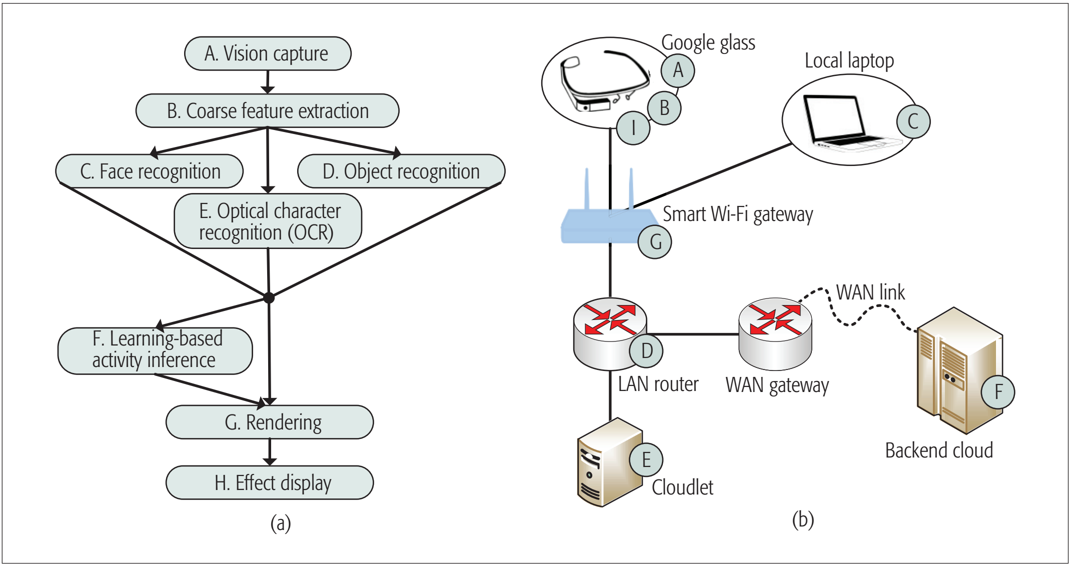

FIGURE 1. The cognitive assistance application is partitioned into segments, shown in a), and offloaded onto fog nodes in the network, shown in b). Note that Cloudlet [1] is a resource-rich server at the edge for low-latency computing. Generally, segments running delay-sensitive tasks (e.g., recognition, rendering) are offloaded to closer facilities.

Dynamic and Autonomous: The state of a fog network is dynamically changing due to the on-off switching of IoT applications and the mobility of fog nodes, as well as the unreliable access links of some fog nodes to the network. Fog computing should be autonomous to tackle the dynamics.

Quality-of-Service Required: IoT applications can specify their Quality-of-Service (QoS) requirements, such as delay, throughput (e.g., streaming rates for video applications), and data locality, to be satisfied in the affinity-aware offloading process. Therefore, determining how multiple applications should be simultaneously deployed in a shared fog network is non-trivial.

Fog computing has attracted increasing attention in recent years, with the fundamental issues under extensive discussion. Similar to cloud computing, research on fog computing can be viewed from three perspectives: the underlying networking infrastructure, resource orchestration, and applications. When designing the networking architecture, most studies used the prevalent Software-Defined Networking (SDN) approach. For example, Baktir *et al*. [6] illustrated the motivation for this procedure, and surveyed its advantages, challenges, and research directions. Liang *et al.* [7] presented a conceptual SDN-based fog networking architecture in a single-ISP wireless network. Another stream of research targets fog-computing-based applications that leverage the distinct features of this new computing paradigm. Fadlullah *et al.* [8], for instance, discussed the vision, benefits, and research directions of using fog computing to conduct deep learning, while Rodrigues *et al.* [9] investigated the optimal design of delay-critical services in fog computing.

Different from the above two perspectives, in this article our focus is on the design of an orchestration framework that bridges the gap between the infrastructure and the applications. Specifically, orchestration refers to the processes of managing and coordinating the physical computational resources provided by the underlying infrastructure to serve the applications. Since fog computing is an extension of cloud computing, existing solutions in cloud computing serve as important references to tackle similar issues in fog computing. Motivated by the widely adopted three-layer orchestration framework in cloud computing, we are particularly interested in how applicable this design is to the context of fog computing. In the rest of this article, we first identify the major challenges that arise in customizing the cloud computing orchestration framework to fog computing due to the distinct features of fog computing. We next discuss the adaptations that should be made in the conventional three-layer orchestration framework to accommodate these challenges. This article is among the initial attempts to discuss the fog computing orchestration issue. Our aim is to deliver some insights toward a complete design and implementation of a fog computing orchestration framework.

Fog computing has attracted increasing attention in recent years, with the fundamental issues under extensive discussion. Similar to cloud computing, research on fog computing can be viewed from three perspectives: the underlying networking infrastructure, resource orchestration, and applications.

Challenges in Designing a Fog Computing Orchestration Framework

Figure 2 depicts a classical three-layer orchestration framework for cloud computing [10]. The framework was initially suggested by the National Institute of Standards and Technology (NIST) and then widely adopted by modern cloud computing systems, such as OpenStack (https://www.openstack.org/). In the framework, the top service layer provides abstractions for cloud applications to access the computing services. Typically, the services can be classified into three types, Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), depending on the level of

Matching application segments and fog computing facilities is a non-trivial task due to the fact that the effectiveness of the application deployment depends on the understanding of both the states of the underlying fog infrastructure in different controller domains and the affinities among the application segments.

abstraction that is provided. The central controller in the control layer allocates the resources in the lower physical resource layer to the cloud applications in the upper service layer.

We can employ the three-layer orchestration framework for fog computing, with fog nodes residing in the physical resource layer and IoT applications running in the service layer. Segmented IoT applications are deployed down to the fog computing infrastructure by the intermediate control layer. However, the distinct features of fog computing present two new challenges to the conventional orchestration framework that need to be tackled: scalability of the control layer and support for affinity-aware offloading. In this section, we illustrate these challenges and survey their state-of-the-art advances.

Scalability of the Control Layer

The control layer of the orchestration framework for fog computing coordinates a massive number of fog nodes in the physical resource layer. It is anticipated that there will be billions of IoT devices connected to the Internet by the year 2020 (http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated). However, the design of a central controller in the conventional orchestration framework does not scale with the rapid growth of fog nodes.

Methods to cope with control layer scalability presented in the literature of cloud computing orchestration serve as important references for fog computing. These can be classified into two categories, the *hierarchical* controller approach, e.g., OpenStack Cells (https://docs.openstack.org/developer/nova/cells.html), and the *flat* controller approach, e.g., [11]. Both of these approaches employ the notion of "divide and conquer" to divide the devices in the physical resource layer into multiple non-overlapped domains. A domain is managed by its internal central controller. Particularly, the hierarchical controller approach deploys extra higher-level controllers on top of the local domain controllers and takes advantage of multi-level workload allocation to mitigate the processing stress in the control layer. The flat controller approach, on the other hand, relies on periodic peer gossiping to synchronize the global view of the computing system, based on which an individual controller makes decisions for its locally managed devices.

In fog computing, the above two methods can also be applied under different practical settings. The hierarchical controller approach fits into a single-operator system, e.g., an LTE-based mobile fog network. In a more general large-scale future fog network, however, domains participating in fog computing are owned and operated by different parties, such as homeowners, universities, and institutes, with heterogeneous access networks. In this case, finding jointly trusted higher-level aggregate controllers becomes difficult, and the flat

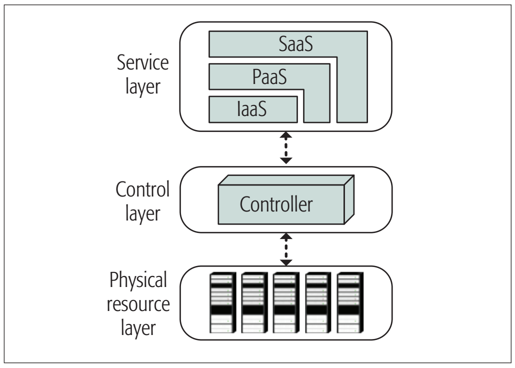

FIGURE 2. The orchestration framework in cloud computing [10].

controller approach is more suitable. However, it is not practical for a fog network to be treated as a gigantic cloud computing system with a flat control layer, in which each controller maintains the view of the system via periodic gossiping of its internal information to its peers. This is because the inter-controller messaging volume of a massive-scale fog network with enormous domains can be huge, and adding a new domain leads to exponential growth in this volume. On the other hand, a domain, as an independent operation entity, intends to disclose limited information about itself, instead of its full internal information, to its peers due to privacy concerns [2]. Therefore, the design of inter-controller communications becomes a critical issue that newly arises in fog computing. A good trade-off should be achieved so that the necessary information to conduct fog computing is transmitted, while the privacy of domains is maintained as much as possible.

Support for Affinity-Aware Offloading

Application offloading in fog computing involves operations at the holistic level across multiple controller domains. Matching application segments and fog computing facilities is a non-trivial task due to the fact that the effectiveness of the application deployment depends on the understanding of both the states of the underlying fog infrastructure in different controller domains and the affinities among the application segments. The main challenges in this affinity-aware offloading process can be listed as follows.

How to Abstract and Model the Affinities among Application Segments Posted by their QoS Requirements: New abstractions are needed for the applications to specify their segmentation schemes, with the QoS requirements to be satisfied in the offloading process. The difficulty lies in extracting and analyzing various affinity constraints among the segments based on the QoS requirements.

How to Obtain the Holistic State Information from Distributed Domains in a Consistent and Scalable Manner: Large-scale fog networks call for a scalable mechanism to gather the state information from the underlying infrastructure for potential offloading. In the flat control layer model, the mechanism includes efficient peer messaging strategies with clearly defined message semantics to ensure that the control layer still functions adequately in a distributed manner. In the hierarchical control layer model, on the other hand, determining how to alleviate and distribute the workload of higher-level controllers that oversee lower-level controllers is crucial.

How to Technically Handle the Segment-to-Facility Matching Problem in the Scheduling Process for Affinity-Aware Offloading: The decision making should comprehensively consider the factors from both the fog infrastructure side and the application side. How this decision making should be technically performed is a complicated yet pivotal issue to be discussed.

Existing Proposals

Studies of the orchestration framework in fog computing are currently in the initial stage, with most efforts put toward examining the practicality and discussing the use cases. The simulation-based study by Wen *et al.* [12] demonstrated the increase in the processing stress of the control layer with the growth of the fog infrastructure scale. Recently, Brito *et al.* [13] developed a small-scale fog network prototype in the three-layer orchestration framework with a central controller where affinity-aware offloading was not supported. Yet how to resolve the two major issues, scalability of the control layer and support for affinity-aware offloading, still remains unclear in the literature. In the rest of this article, we discuss possible solutions to the issues. In particular, we focus on the flat control layer model, as we believe it can better serve the general case of large-scale fog networks with distributed ownerships and operations. (Note that the hierarchical controller approach is an alternative solution for the control layer that fits into some specific scenarios with its corresponding advantages. It is beyond the scope of this article, but merits future discussion.) Next, we discuss how adaptations should be made to the three-layer orchestration framework with a flat control layer to tackle the two major design challenges presented by the distinct features of fog computing.

From Cloud Computing to Fog Computing: Adaptations in the Orchestration Framework

Architecture Overview

The three-layer fog computing orchestration framework with a flat control layer is depicted in Fig. 3. Fog nodes in the physical resource layer are clustered into domains based on the nature of their ownerships and operations. For example, in the figure, domains are formed by fog nodes connected to local LTE networks, vehicular networks, or local area networks of houses and institutes. Similar to cloud computing, fog nodes within a domain are managed by a central controller. The southbound interface between the fog nodes and the controller collects the underlying hardware information and distributes the offloaded application segments onto the fog nodes for physical execution. Federated domain controllers form the control layer and communicate with each other via the westbound interface in a peer-to-peer (P2P) manner. The top service layer provides abstractions for IoT applications to access the fog computing services, and talks to the control layer via the northbound interface. The flat control layer is suitable for large-scale fog networks operated by heterogeneous access networks with distributed operations. In the rest of this section, we discuss the design principles of the components in the fog computing orchestration framework.

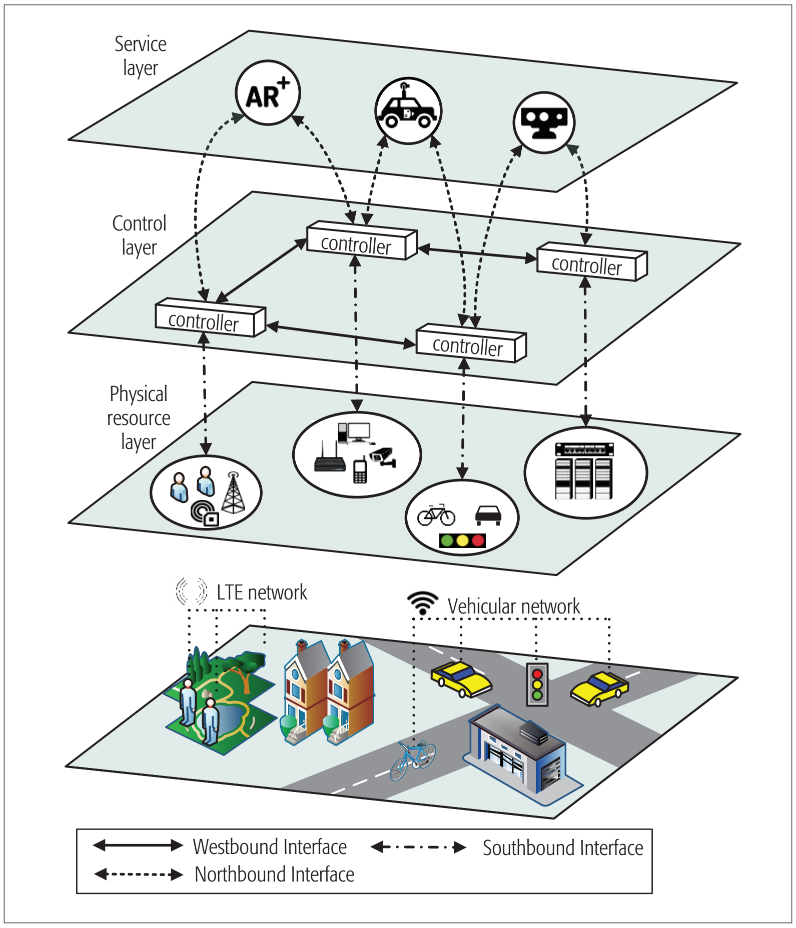

FIGURE 3. Fog computing orchestration framework with a flat control layer. The bottom plane of the figure depicts the physical scenario of a fog network. The upper planes show the three-layer orchestration framework.

Westbound Interface

The westbound interface in the control layer interconnects peer controllers, and helps the controllers maintain the knowledge of their edge networks via P2P control message exchanges. In cloud computing, a controller broadcasts its internal information to its peers to maintain the view of the whole system. However, this strategy incurs a huge traffic volume for large-scale fog networks with tremendous numbers of domains. Adding a new domain also leads to exponential traffic growth. We suggest that in the context of fog computing, each controller should maintain the view of its own detected edge network instead of the entire fog network, because fog computing aims to offload computation to edge facilities. IoT applications naturally prefer to be offloaded to nearby fog nodes rather than fog nodes far away. Therefore, it is not necessary for a controller to maintain a view of the entire network, but only a local view of its nearby fog nodes. As a result, the region of P2P messaging in the control layer is limited, and the control message volume is alleviated.

The message format in westbound communications should also be carefully defined. The distributed domain ownerships and operations make it unrealistic for a controller to fully disclose

The service layer provides abstractions for IoT applications to access the computing services. Such abstractions include not only a virtualized environment for application development, but also a flow engine for application segmentation and an optional interface for application self-scheduling.

its internal information to its peer domains. A better strategy is to disclose limited, but still sufficient, state information for potential application offloading from peer domains. Thus, instead of the full information, a domain announces only the resource availability (e.g., VM profiles that can be provisioned) of its internal fog nodes as a whole to other peers. We define the edge network detected by a domain controller through P2P messaging as its control overlay, which distinguishes the available region where applications submitted and hosted by this domain can be deployed. A domain appears in the *control overlay* of another peer domain only if its sent message successfully reaches that peer domain. The sent message from a domain carries its necessary state information, including the following:

- **General VM profile:** the hardware features, resource capacities, provisioning latency, availability, and pricing information of the VMs that a domain can provide.
- **Time-to-live stamp:** the messaging range within which a domain determines its P2P messages to be spread.
- **Reachability rules:** rules to prevent untrusted peer domains from receiving the messages.
- **Self-preferences:** declaration of other self-preferences of a domain, e.g., a certain kind of application to which the domain does not prefer to be offloaded.

Self-Measurement and Analytics-Driven Management in the Control Layer

In order for an individual controller to better understand a highly dynamic fog network and form its control overlay, we introduce the following two extra modules, supplementary to westbound communications.

The Self-Measurement Module: The self-measurement module inside a controller periodically measures its transmission delay toward peer domains in the control overlay. The delay information is of significant importance to satisfy the delay-related QoS requirements of IoT applications.

The Analytics-Driven Management Module: As an increasingly popular approach to accommodate large-scale dynamic systems [14], the analytics-driven management module analyzes the historical data of the control overlay dynamics using data analytics techniques, from which the stability, reliability, and long-term resource availability of peer domains can be learned. Such information supports more reliable offloading, as an application does not prefer to be offloaded to a domain that is frequently offline or encountering failures.

Service Layer and Northbound Interface

The service layer provides abstractions for IoT applications to access the computing services. Such abstractions include not only a virtualized environment for application development, but also a flow engine for application segmentation and an optional interface for application self-scheduling.

The flow engine assists the partitioning of applications into segments. Depending on the levels of abstraction that are provided, there can be different forms of flow engines. The first form is to ask the application developers to explicitly define the workflows, such as the example in Fig. 1b. To relieve the burden of the application developers, the second form is to provide pre-defined and widely-acknowledged programming frameworks to the application developers. For example, data analytics applications developed based on the MapReduce framework can be naturally decomposed into Map and Reduce tasks. The third form is to enable automated segmentation that is completely transparent to the applications. Thanks to recent research progress in application profiling [1], leading projects, such as CloneCloud, MAUI, and ThinkAir, have been able to transparently analyze and partition applications in standard granularities (e.g., threads, methods) that can be offloaded for execution. Along with the segmentation schemes, the application-specific QoS requirements are also defined for the underlying controllers to make offloading decisions. Depending on the forms of the flow engines, the specified QoS requirements may be segment-wise, workflow-wise, or on the overall application level, as follows.

Delay Requirements: Requirements on the processing or transmission delay of a segment, a certain part of the workflow, or the overall application processing flow can be specified. Fine-grained segment-wise delay requirements are necessary for applications with heterogeneous delay requirements among their segments. (Take the cognitive assistance application in Fig. 1 as an example. The face recognition segment requires stringent delay, while the learning-based activity inference segment is less sensitive to delay.)

Performance-Based Requirements: Applications are allowed to specify other quantified performance-based requirements, such as video or audio streaming rates, image resolution, and encoding throughput.

Locality Requirements: Locality restricts certain segments' executions to specific fog nodes, or some segments to be co-located on the same fog node, which may stem from source data issues or security policies.

Upon submission of an IoT application, likely from a resource-poor fog node, the application segments with the specified QoS requirements go through the northbound interface to the controller. The application is then deployed based on the control overlay detected by the controller. We define the scheduling result of an application as its *application overlay*. The design of our orchestration framework reserves the flexibility of application self-scheduling, enabled by the application self-scheduling interface defined in the service layer abstractions. Based on the control overlay provided by the controller through the northbound interface, an application can run its customized scheduling algorithm and inform the controller of its self-derived application overlay. This is a "push-based" procedure, in which the scheduling result is simply "pushed" to the controller. It requires appli-

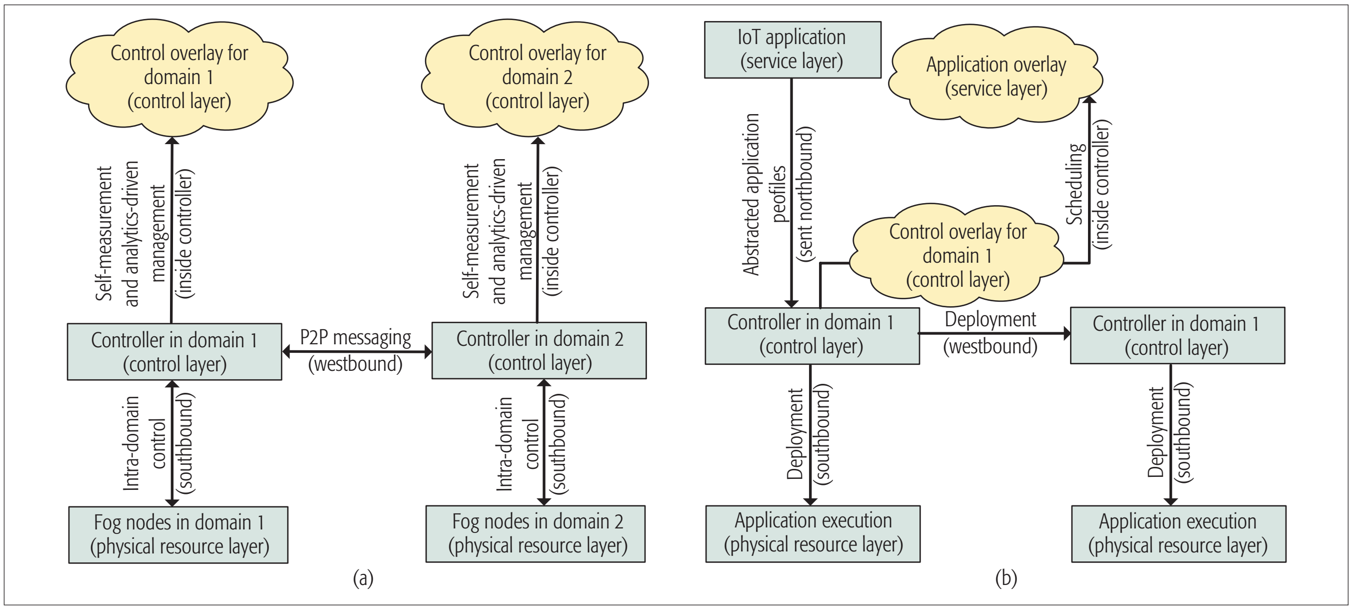
Control overlay for domain 1 (control layer)
Control overlay for domain 2 (control layer)
Self-measurement and analytics-driven management (inside controller)
Self-measurement and analytics-driven management (inside controller)
Controller in domain 1 (control layer)
P2P messaging (westbound)
Controller in domain 2 (control layer)
Intra-domain control (southbound)
Intra-domain control (southbound)
Fog nodes in domain 1 (physical resource layer)
Fog nodes in domain 2 (physical resource layer)
(a)
IoT application (service layer)
Application overlay (service layer)
Abstracted application profiles (sent northbound)
Scheduling (inside controller)
Control overlay for domain 1 (control layer)
Controller in domain 1 (control layer)
Deployment (westbound)
Controller in domain 1 (control layer)
Deployment (southbound)
Deployment (southbound)
Application execution (physical resource layer)
Application execution (physical resource layer)
(b)

FIGURE 4. Summary of the interactions among key components in the fog computing orchestration framework with a flat control layer: a) The flow in forming the control overlay. The controller oversees the intra-domain fog nodes in a centralized manner. Communicating with peer controllers from other domains via the westbound interface and supplemented with self-measurement and analytics-driven management modules, the controller derives its control overlay. b) The flow in forming the application overlay. The service layer provides abstractions for applications. The abstracted application profiles are sent to the controller via the northbound interface. Together with its control overlay, the controller determines the application overlay for affinity-aware offloading.

cation developers' prior knowledge and extra programming work for scheduling. In contrast, the default policy in fog computing is a "pull-based" procedure that is inherited from cloud computing.[2] The controller "pulls" the application profiles from the service layer and schedules the applications accordingly. Reserving the flexibility of application-customized scheduling is a trend in modern distributed computing systems, such as Hadoop.[3] In this article, we focus on the default "pull-based" procedure, which requires more effort in the design.

Application Scheduling Module in the Control Layer

With the dynamically maintained control overlay and the application profiles collected from the northbound interface, the controller schedules the applications to be deployed onto the fog network. The scheduling process should be optimized in the following aspects.

Quantified QoS: Quantified application QoS includes delay and performance-based requirements. It can be an objective to be maximized, or a constraint to be guaranteed above a certain required threshold.

Operational Cost: Incurred by renting on-demand resources from peer domains for offloading, the total operational cost can be set by a controller as a minimization objective or a constraint to be limited within a range.

Social Welfare: When a controller jointly optimizes multiple applications, the social welfare calculated by the quantified application QoS minus the total operational cost can be an objective to be maximized.

Application Segmentation Scheme: Profiles of the segments (e.g., specification of required resources, resource-to-achieved-QoS relationship) reflect the physical composition of an application that must be obeyed.

Current State of the Control Overlay: The control overlay, including the resource capacities and specifications, communication delay, and preference policies of peer domains, are physical constraints in application offloading.

Locality Considerations: The special location requirements should be regarded as constraints to be satisfied.

Implementation Considerations

To bridge the gap between theory and practice, this sub-section illustrates two technical aspects to be considered in implementing the fog computing orchestration framework in accordance with the newly introduced adaptations.

Control Messaging Intervals and Ranges: An optimized design that considers the trade-offs in control messaging through the communication interfaces can be introduced. Take the P2P messaging in the westbound interface as an example. A fine-grained exchange with small heartbeat intervals updates the network dynamics in a more timely manner. However, it increases the control traffic volume. On the other hand, a broader area for the P2P messaging of a domain potentially enlarges its control overlay, and equivalently provides a larger and more flexible potential range for application deployment. However, the expansion of the messaging range may impair the convergence speed in forming the control overlay. An optimized design can benefit the westbound communications by achieving the best trade-off.

Local Hierarchy in the Control Layer: In the P2P file sharing literature, *super-peers* [15] are stable peers with good connectivities and powerful computational capabilities. Super-peers coordinate the local P2P network and have been shown to effectively facilitate file lookup among peers. Likewise, if some peers in the

[2] In cloud computing, the control layer is responsible for resource scheduling to fulfill the requirements from cloud applications in the service layer.

[3] In Hadoop, analytics applications are scheduled by the Hadoop cluster by default ("pull-based"). Meanwhile, applications can also run their custom-written scheduling algorithms utilizing the hardware resources provided by the cluster ("push-based")

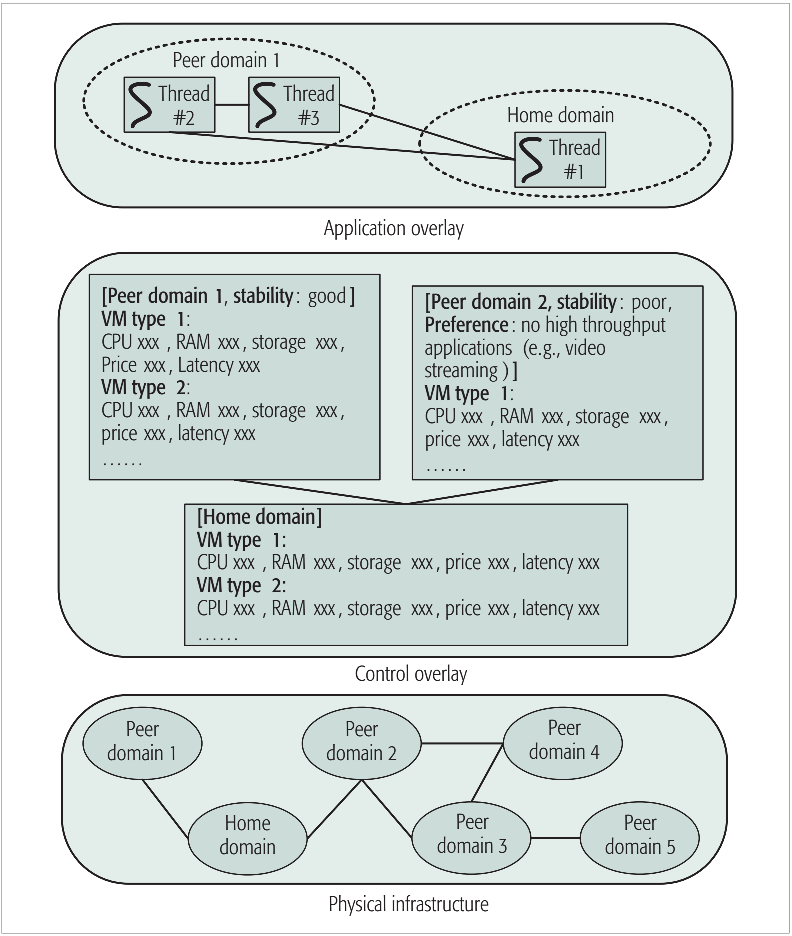
Peer domain 1
Thread #2
Thread #3
Home domain
Thread #1
Application overlay
[Peer domain 1, stability: good]
VM type 1:
CPU xxx, RAM xxx, storage xxx,
Price xxx, Latency xxx
VM type 2:
CPU xxx, RAM xxx, storage xxx,
price xxx, latency xxx
……
[Peer domain 2, stability: poor,
Preference: no high throughput applications (e.g., video streaming)]
VM type 1:
CPU xxx, RAM xxx, storage xxx,
price xxx, latency xxx
……
[Home domain]
VM type 1:
CPU xxx, RAM xxx, storage xxx, price xxx, latency xxx
VM type 2:
CPU xxx, RAM xxx, storage xxx, price xxx, latency xxx
……
Control overlay
Peer domain 1
Peer domain 2
Peer domain 4
Home domain
Peer domain 3
Peer domain 5
Physical infrastructure

FIGURE 5. Simple examples of a control overlay and an application overlay from the perspective of the Home Domain. In the formed control overlay, the Home Domain successfully detects two neighboring domains. An application submitted to the Home Domain is partitioned into three threads. According to the derived application overlay, two threads are offloaded to fog nodes in Peer Domain 1.

flat control layer of the fog computing orchestration framework are resource-rich and stable with good connectivities, they can also volunteer to be super-peers and help coordinate the local network. Super-peers cache nearby data in a centralized manner and help local domains with faster convergences in forming the control overlays. By volunteering to be super-peers, they can also benefit by potentially attracting more applications to be executed on them due to their good reliability and availability.

Architecture Summary

Figures 4a and 4b summarize two flows showing how the key components in the fog computing orchestration framework interact with each other. Overseeing intra-domain fog nodes in a centralized manner, as in the first flow depicted in Fig. 4a, the controller maintains full knowledge of its managed domain and communicates with peer domains by westbound P2P messaging. Together with the self-measurement and analytics-driven management modules, the controller forms a view of its edge network, namely, the control overlay. The orchestration framework achieves *control layer scalability* with distributed controllers that are easy to scale. The carefully defined controller functionalities and westbound interface guarantee the control layer still functions adequately, though it is distributed instead of centralized.

Figure 4b depicts the second flow to support affinity-aware offloading for IoT applications. The controller jointly considers the application profiles collected from the northbound interface and the state of the control overlay to derive the application overlay, which describes how the applications are deployed. On the other hand, applications can also choose to determine the deployment schemes by themselves (i.e., self-scheduling). Examples of a control overlay and an application overlay are illustrated in Fig. 5. The second flow is clearly defined to *support affinity-aware offloading*.

Concluding Remarks

This article focuses on the design of a fog computing orchestration framework, which is currently an open issue that calls for extensive discussion. In particular, we investigate how the three-layer cloud computing orchestration framework can be adapted to fog computing. We first identify the critical challenges presented by the distinct features of fog computing that need to be tackled in this procedure. To overcome the challenges, we discuss the necessary adaptations in the three-layer orchestration framework to fit into the fog computing scenario. With this discussion, we aim to shed light on the ongoing investigation of fog computing orchestration.

References

[1] F. Liu *et al.*, "Gearing Resource-Poor Mobile Devices with Powerful Clouds: Architectures, Challenges, and Applications," *IEEE Wireless Commun.*, vol. 20, no. 3, 2013, pp. 14–22.
[2] S. Yi, Z. Qin, and Q. Li, "Security and Privacy Issues of Fog Computing: A Survey," *Springer Wireless Algorithms, Syst., and Applicat.*, 2015, pp. 685–95.
[3] L. M. Vaquero and L. Rodero-Merino, "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *ACM SIGCOMM Comput. Commun. Review*, vol. 44, no. 5, 2014, pp. 27–32.
[4] OpenFog Reference Architecture White Paper; available: http://www.openfogconsortium.org/resources/white-papers/
[5] K. Ha *et al.*, "Towards Wearable Cognitive Assistance," *Proc. 12th ACM Annu. Int. Conf. Mobile Syst., Applicat., and Services*, 2014, pp. 68–81.
[6] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions," *IEEE Commun. Surveys & Tutorials*, vol. PP, no. 99, 2017.
[7] K. Liang *et al.*, "An Integrated Architecture for Software Defined and Virtualized Radio Access Networks with Fog Computing," *IEEE Network*, vol. 31, no. 1, 2017, pp. 80–87.
[8] Z. Fadlullah *et al.*, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems," *IEEE Commun. Surveys & Tutorials*, vol. PP, no. 99, 2017.
[9] T. G. Rodrigues *et al.*, "Hybrid Method for Minimizing Service Delay in Edge Cloud Computing Through VM Migration and Transmission Power Control," *IEEE Trans. Comput.*, vol. 66, no. 5, 2017, pp. 810–19.
[10] F. Liu *et al.*, "NIST Cloud Computing Reference Architecture," *NIST Special Publication*, vol. 500, 2011, p. 292.
[11] O. Babaoglu, M. Marzolla, and M. Tamburini, "Design and Implementation of a P2P Cloud System," *Proc. 27th Annu. ACM Symp. Applied Comput.*, 2012, pp. 412–17.
[12] Z. Wen *et al.*, "Fog Orchestration for Internet of Things Services," *IEEE Internet Comput.*, vol. 21, no. 2, 2017, pp. 16–24.
[13] M. S. de Brito *et al.*, "A Service Orchestration Architecture for Fog-Enabled Infrastructures," *Proc. 2nd IEEE Int. Conf. Fog and Mobile Edge Comput.*, 2017, pp. 127–32.
[14] L. Cui, F. R. Yu, and Q. Yan, "When Big Data Meets Software-Defined Networking: SDN for Big Data and Big Data for SDN," *IEEE Network*, vol. 30, no. 1, 2016, pp. 58–65.

[15] E. K. Lua *et al.*, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Commun. Surveys & Tutorials*, vol. 7, no. 2, 2005, pp. 72–93.

Biographies

Yuxuan Jiang [S'14] received his B.Eng. degree in electronics and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2013. He is now pursuing the Ph.D. degree at the Hong Kong University of Science and Technology. His current research interests include resource management for networked systems, with an emphasis on cloud/edge computing and big data systems.

Zhe Huang [M'13] is a postdoctoral researcher in the EDGE Lab of the Department of Electrical Engineering at Princeton University. He received his M.Phil. and Ph.D. degrees from the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology, Hong Kong. His research interests include fog computing, cloud computing, big-data analytic systems and smart grid systems.

Danny H.K. Tsang [M'82, SM'00, F'12] received the Ph.D. degree in electrical engineering from the Moore School of Electrical Engineering at the University of Pennsylvania, U.S.A., in 1989. He has been with the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology since the summer of 1992 and is now a professor in that department. He was a Guest Editor for the *IEEE Journal on Selected Areas in Communications* special issue on Advances in P2P Streaming Systems, an associate editor for the *Journal of Optical Networking* published by the Optical Society of America, and a Guest Editor for the *IEEE Systems Journal*. He currently serves as technical editor for *IEEE Communications Magazine*. He was nominated to become an IEEE Fellow in 2012 and an HKIE Fellow in 2013. During his leave from HKUST in 2000-2001, he assumed the role of principal architect at Sycamore Networks in the United States. He was responsible for the network architecture design of Ethernet MAN/WAN over SONET/DWDM networks. He invented the 64B/65B encoding (U.S. Patent No. US 6,952,405 B2) and contributed it to the proposal for Transparent GFP in the T1X1.5 standard that was advanced to become the ITU G.GFP standard. The coding scheme has now been adopted by the International Telecommunication Union (ITU) Generic Framing Procedure Recommendation GFP-T (ITU-T G.7041/Y.1303)). His current research interests include cloud computing, cognitive radio networks and smart grids.