A Survey on Fog Programming: Concepts, State-of-the-Art, and Research Challenges

Dapeng Lan, Amir Taherkordi, Frank Eliassen, and Geir Horn
University of Oslo, Norway
{dapengl,amirhost,frank,geir.horn}@ifi.uio.no

ABSTRACT

With the rapid evolution of the Internet of Things (IoT) and the growth of IoT-generated data, cloud computing platforms have been widely used to store and process this type of data. However, cloud computing cannot handle rapidly emerging smart applications with latency-sensitive, high throughput, and high availability and reliability requirements, such as virtual reality and autonomous vehicles. The fog and mobile edge computing paradigms have been proposed to bring the cloud capacity along the network to end devices to address the above concerns. A key development aspect of fog systems is their programming models. Fog programming models and frameworks face challenges that originated from the unique characteristics of fog systems, such as heterogeneity, scalability, and mobility. In this paper, we first study the main characteristics of fog programming, as well as the design requirements of fog programming models with respect to the fog architecture and application types. Then, we survey fog programming models both from research and industry perspectives. Finally, we point out the issues and future challenges in fog programming. The survey framework, presented in this paper, provides useful insights and outlook for the fog programming research and development, inspires further research on fog programming models, and sheds light on the future of programming in this fast-growing computing paradigm.

KEYWORDS

Fog Computing, Internet of Things, Fog Programming Models and Frameworks, Survey

ACM Reference Format:
Dapeng Lan, Amir Taherkordi, Frank Eliassen, and Geir Horn. 2019. A Survey on Fog Programming: Concepts, State-of-the-Art, and Research Challenges. In *2nd International Workshop on Distributed Fog Services Design (DFSD '19), December 9–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3366613.3368120

1 INTRODUCTION

The rapid evolution of the Internet of Things (IoT) devices, including sensors, actuators, and gateways, produces massive volume, velocity, and variety of IoT big data. In this case, central clouds have been the main infrastructure to deal with the IoT data and services like fetching the data from sensors, analyzing and storing the information, and sending back the control signals to end devices [17]. However, they hardly fulfill the performance requirements of rapidly emerging smart applications, such as virtual reality (VR)/augmented reality (AR), autonomous vehicles, and smart grid automation, all demanding high throughput, ultra-low latency, and high availability and reliability [12]. In light of this, the fog computing and mobile edge computing (MEC) paradigms have been proposed to address the low latency and high throughput challenges by migrating cloud services to the proximity of local infrastructures in order to enable real-time applications and to elevate quality of service (QoS) [4]. As shown in Figure 1, a general architecture model for fog computing contains three layers: the cloud layer, the fog devices and networks layer, and the IoT devices layer.

Fog computing faces many challenges concerning programming models and frameworks. Fog computing brings many features that are different from cloud computing, such as location awareness, low latency, geographical distribution, and mobility [7]. The requirements for fog programming are more dynamic and challenging than for cloud platforms. For example, one challenge is the heterogeneity of fog devices which have various computation and storage resources. Fog programming frameworks need to provide unified interfaces and tools to develop, deploy, and manage services in such a heterogeneous system. Another challenge in fog programming is to provision and orchestrate dynamic and hierarchical resources to build and manage applications on diverse fog platforms.

Approaches for fog programming have been partly surveyed in some previous works. In most of them, the main focus of the studies has been on the architectural aspects of developing fog systems. Mouradian *et al.* [10] categorize fog architectures into two classes: end-user application-agnostic architectures and application-specific architectures. They further divide the research scopes into application provisioning, resource management, communication, and federations. Mukherjee *et al.* [19] discuss a three-tier fog architecture, software-defined fog architectures, and Fog Radio Access Networks (F-RAN). Yousefpour *et al.* [6] also investigate fog frameworks and programming models from concepts, architectures, and data perspectives. Nath *et al.* [21] mainly discuss two different architectural models: service-oriented architectures and application-specific architectures. Liu *et al.* [15] present architectures of mobile edge cloud (MEC) systems, including concepts, technical enablers, service models, and deployment scenarios. In a similar work, C. Puliafito *et al.* [22] survey the mobility aspect of fog computing.

Compared to the works above, we first explore the characteristics and requirements of fog programming in more detail, which are needed for a better understanding of developing fog applications. Second, we not only provide a survey for fog programming solutions from a research perspective but also extend the investigation

DFSD '19, December 9–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7031-8/19/12…$15.00
https://doi.org/10.1145/3366613.3368120

to the industrial field. Lastly, we discuss research challenges and open issues in fog programming.

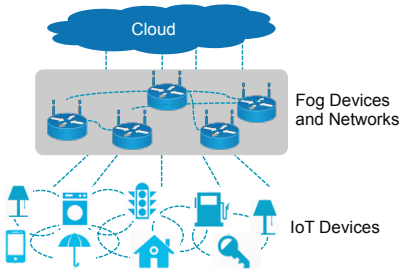

Figure 1: A general architecture model for fog computing [24]

The rest of the paper is organized as follows. In Section 2, we discuss the characteristics of fog programming and the programming design requirements considering the fog architecture model and its applications. Then, in Section 3, we survey the state-of-the-art fog programming models. In Section 4, we point out the issues and open research challenges in developing fog programming models and frameworks. We make the concluding remarks in Section 5.

2 FOG PROGRAMMING: CHARACTERISTICS AND REQUIREMENTS

Fog programming frameworks are influenced by their typical characteristics, architecture models, and application types. In this section, we provide a high-level view of fog programming and compare it with programming in cloud computing. Then, we identify the critical characteristics of fog programming, followed by the programming requirements with respect to different architectural perspectives and fog application types.

2.1 Comparison with Cloud Programming

Similar to cloud computing, fog computing can also provide three levels of services: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Therefore, many cloud techniques and algorithms can be adapted for fog systems, such as resource provisioning and orchestration [25]. However, there exist some differences. The most typical feature of fog computing platforms is their distance to the end IoT devices, providing computation and storage resources at the edge of the network to support latency-sensitive applications and services [16] within several milliseconds. Fog computing also brings the location-awareness and context-awareness properties, while cloud computing is far from users lacking such information. Besides, fog computing clients are mobile and services are hosted on mobile devices, while cloud computing clients are usually stable, and services are hosted on fixed servers. Geo-distributed fog nodes can extract their locations and track end-users to support mobility [24]. Compared to cloud computing, fog programming frameworks should provide migration, dynamic service provision, and orchestration modules to support such dynamic systems.

2.2 Characteristics

In this section, the main characteristics of programming fog platforms are presented.

Heterogeneity. In contrast to dedicated data centers with well-planned devices and networks, fog nodes are heterogeneous with varying processing and storage resources, and varying network connectivity and bandwidth. In this case, the target programming framework should provide development tools for designing services and tasks according to the different capacities of devices, such as computing and storage resources. The framework should also decide which application components should be deployed and where.
Scalability. Fog devices are expected to host millions of IoT/end-user services belonging to multiple applications running simultaneously on shared fog platforms. Handling millions of IoT devices at the fog layer while satisfying the latency guarantees of the applications is a big challenge. Thus, fog systems need to adapt the operation to scale services elastically. Fog programming models should provide modules to support scalability, such as elasticity engines that allocate resources in terms of VM or containers according to the resource requirements of user applications.
Quality of Service. Each application deployed in the fog system needs to meet the Quality of Service (QoS) requirements. One crucial factor is latency. Fog systems are envisioned to be used in real-time applications due to the proximity of fog nodes to IoT/end-user devices. Therefore, fog programming frameworks should provide some modules supporting QoS. For example, the migration engine helps applications move from the Cloud to the fog stratum or move between fog nodes depending on service level agreement (SLA). On the other hand, QoS management algorithms can also help manage QoS, enabling decisions in the system to optimize resource usage.
Local user context awareness. Fog applications can leverage local user context information and locations. As Fog nodes are deployed at the edge, they can access the real-time wireless network and channel information. These features enable fog nodes to monitor users' requirements and provide appropriate services to each of the users in time. Context analysis modules should be developed, which are in charge of acquiring, analyzing, and managing context information.
Mobility and unreliable access. Fog services may be hosted on mobile devices, even across different fog systems. Besides, mobile devices get access to fog services through unreliable wireless access and often change their points of attachment to the network. Therefore, mobility support is critical for the Fog. Architectural modules such as mobility engines are needed to solve this issue.
Openness. Fog computing is suitable for various use cases with participants from different industrial fields. An open-standard environment offers a bright future for fog computing, as it is evident by the IEEE Standard's adoption of OpenFog Reference Architecture [5]. Openness is a significant feature for the development and adoption of fog computing. In this case, the fog programming framework should be standardized and open, guaranteeing the *interoperability* among different fog devices, even different fog systems.

2.3 Requirements: Architectural Viewpoint

Fog architectures are not a simple extension to the cloud architecture because of their distinctive characteristics and requirements such as heterogeneity, latency-sensitive, and contextualized services. New fog computing architectures, along with new programming models, are needed. For example, the Openfog consortium has

proposed a fog computing reference architecture (RA) based on the Openfog pillars: Security, Scalability, Open, Autonomy, RAS (Reliability, Availability, and Serviceability), Agility, Hierarchy, and Programmability [5]. Programmability means highly adaptive deployment, including support for programming the software and hardware layers, which can be completely automated. Programmability brings benefits, such as adaptive infrastructures for diverse IoT deployment scenarios and changing business needs; resource-efficient deployments by containerization; supporting multi-tenancy using isolated runtime environments. The other perspective is the diverse user scenarios leading to various application-specific architectures. The services and tasks distribution for such architectures are heterogeneous, such as distributing the tasks between IoT nodes and fog nodes, fog nodes and fog nodes, and fog nodes and the Cloud. The system can also be dynamic in terms of service migration during runtime. In this case, fog programming models need to have service provisioning and orchestration modules to support such heterogeneous and dynamic systems.

2.4 Requirements: Application Viewpoint

In this section, we discuss fog programming requirements from the application viewpoint. Applications benefit from being deployed at the edge, fully utilizing the features of fog computing.

Latency-sensitive applications. One of the prominent scenarios using fog computing platforms is latency-sensitive applications. As the services can be executed at the edge and the data does not have to travel over the Internet back and forth between end devices and remote data centers, the latency will be reduced, and the user experience of delay-sensitive applications such as real-time traffic control and networked gaming will be improved. Usually, the programming model requires QoS or SLA modules for this kind of application. For example, the forest fire detection application in [25] needs low latency and urgent notification, which is guaranteed by the SLA management module in the application management layer of the fog programming framework.

Autonomous applications. Some applications can not be maintained locally due to being installed in remote areas or the high cost of manual changes. For example, offshore oil and gas platforms are usually established in the middle of the ocean, lacking access to the cellular network and using costly satellite networks. Fog/edge computing platforms provide an ideal solution for this use case. Without the connection from the Cloud, the local application should continue working regularly. This kind of fog programming model must be augmented with self-configuration, management, healing, and energy awareness functionalities to process data and make autonomous decisions locally.

Privacy and security applications. Privacy concerns must be addressed in some applications, e.g., healthcare applications [22] contain sensitive data which is not suitable for sharing at the cloud side; camera as a sensor collects image data revealing a person's identity. This kind of sensitive data needs to be processed and stored locally at the fog side, and only the processed data needs to be uploaded to the Cloud. Fog nodes can do data filtering before uploading to the Cloud without revealing any sensitive information. The heterogeneity of connected fog devices and proximity to the users make their security challenging. Besides, site attacks

	Heterogeneity	QoS - Management	Scalability	Mobility	Openness
MobileFog (PaaS)	✓	✗	✓	✓ (but not context awareness)	✗
Distributed Dataflow	✓	✓	✓	✓(node duplication and pre-configuration)	✓
Hybrid Cloud/Fog	✗	✓	✓	✗	✓
FogFlow	✓	✓	✓	✗	✓
Fog05	✓	✓	✓	✗	✓
Cisco edge computing framework	✗	✓	✓	✗	✗
AWS Greengrass	✓	✓	✓	✗	✗
Azure IoT Edge	✓	✓	✓	✗	✓

Figure 2: Summary of fog programming frameworks

are more likely to happen on the fog nodes than in cloud data centers. This requires the fog programming model to provide strong access-control policies for fog nodes. New authentication and trust mechanisms should also be proposed to cope with heterogeneity of fog nodes and IoT nodes from different vendors.

Context-awareness applications. As fog nodes are closer to the end IoT devices, they can make use of this feature to develop context-aware applications. Context-aware applications, such as local content sharing, etc., can leverage location information, user context, and local computing capability to improve performance. This is because data processing and computation are conducted on smaller datasets locally with less overhead and latency. It also increases privacy as the location of mobile users is not sent out of the wireless access network. For instance, the healthcare application proposed by C. Puliafito *et al.* [22] can utilize the context information of the patients to provide better services. Besides, the work in [21] proposes an intelligent transportation system. As contextual information like traffic conditions are known to fog nodes, the vehicles do not need to upload, process, and download rerouting instructions, which can save much time. The fog programming module should support virtualization technology, computing offloading, and service migration to interact between the Cloud and fogs.

3 FOG PROGRAMMING: STATE-OF-THE-ART

3.1 Fog Programming Frameworks

Fog programming frameworks are in charge of handling the lifecycle of applications/services deployed over fog platforms, including development, deployment, execution, and management. In this section, we discuss the state-of-the-art fog programming frameworks, both in academia and industry. Figure 2 lists existing programming frameworks and their support of fog programming characteristics, explained in the previous section.

3.1.1 MobileFog. MobileFog is a programming architecture proposed by Hong *et al.* [14]. MobileFog is a high-level programming model for future Internet applications with the characteristics of geo-spatially distributed, large-scale, and latency-sensitive. An application in MobileFog is composed of distributed mobile fog processes mapped into different instances in the Fog and the Cloud. From the development perspective, it provides various event handlers, functions, and programming interfaces to develop applications. Once the code is written, the same programming code can be deployed in any of the nodes and devices. In this case, it solves

the heterogeneity problem. MobileFog also provides different kinds of APIs for managing applications distributed over IoT, the Fog, and the Cloud. For instance, applications can retrieve underlying information of the nodes about resources, capabilities, locations, and logical positions, using APIs such as *query_capability (type t)*. Based on this underlying information, applications can adjust the running processes and execute the appropriate sensing and actuation. Moreover, on-demand instances can be created when the computing instance is overloaded during the management phase. Later MobileFog was extended to Foglet [23], adding the implementation of the APIs, extending with discovery and migration algorithms, and conducting the experimental evaluation.

According to the fog programming characteristics, MobileFog supports heterogeneity, scalability, and partially mobility, but it does not meet the requirements of QoS as it lacks the QoS management module. It is not standardized and open source, lacking openness.

3.1.2 Distributed data flow (DDF) fog programming architecture. Giang *et al.* [13] propose a distributed data flow (DDF) programming model for IoT that utilizes computing resources across the Fog and the Cloud. DDF connects components like a flow chart, providing a flexible way to develop end-user applications. For development, this programming model considers two types of developers: node developers and IoT application developers. Node developers are responsible for developing the function of the components used for specific things and services. IoT application developers focus on creating flows and connecting things and services. The components can be deployed both in the Cloud and the fog strata. The fog stratum can be divided into three types: Edge, IO, and Compute nodes, corresponding to the appropriate capabilities of the nodes. For example, some nodes are suitable for running computing tasks due to relative abundant computing resources. In order to solve the mobility issue, some nodes are duplicated along with the flow beforehand. For the management, DDF applications do not rely on one centralized management; instead, they can decide how to execute the application logic themselves and communicate with each other based on the flow's design. The nodes also include modules for scaling the applications when it is needed. In order to evaluate the model, the authors implemented a DDF framework based on the open-source project Node-RED (Distributed Node-RED).

Therefore, DDF fog programming fulfills the need for heterogeneity, QoS, scalability, and partially mobility, but its mobility support needs duplicating the nodes, which costs extra resources and takes time for developers to pre-configure the duplication.

3.1.3 Hybrid Cloud/Fog. Yangui *et al.* [25] propose an architecture for a Platform-as-a-Service (PaaS) based on two principles: extending the existing PaaS, and using the REST paradigm for the interactions. Cloud Foundry is used as the basis for the implementation, allowing to automate the provisioning of applications in hybrid cloud/fog environments. This hybrid cloud/fog model is used for fire detection utilizing temperature sensors and fire fighting robots. In the Cloud, the architecture consists of four layers: application development layer, application deployment layer, application hosting, and execution layer, and application management layer. The application development layer provides an Integrated Development Environment (IDE) that allows developing and combining the components. In the application deployment layer, new modules, *Deployer* and *Controller* modules, are developed. The *Deployer* is responsible for discovering and instantiating the fog resources and executing the application's components. It also places the applications either in the Cloud or the Fog. The *Controller* module is responsible for interaction with the *Deployer* and setting the locations of the components. Besides, there is a novel module, called *Fog Resources Repository*, for storing the available hosting resources. For the hosting and execution layer, a novel module called *Orchestrator* is used for orchestrating the execution flow between the containers. Lastly, the management layer is arranged horizontally in the PaaS. It also contains several new modules. For example, the *SLA Manager* module is responsible for managing the application's QoS, the *Elasticity Engine* is responsible for scaling up/down the application components, and the *Migration Engine* is responsible for migrating the components among devices in different layers. In addition, some appropriate REST-based interfaces are proposed for signaling, controlling, operating, and managing the components.

As indicated above, Hybrid Cloud/Fog meets the characteristics of QoS, scalability, and openness. However, it does not support heterogeneity as it needs support from the Cloud all the time, and it does not contain the migration module to support mobility.

3.1.4 FogFlow. B. Cheng *et al.* [9] propose a standard-based programming model for fog computing by extending the dataflow programming model with *declarative hints* with the widely used standard *NGSI*, aiming the applications in IoT smart city platforms. The authors claim this brings two benefits to service developers: *i)* faster and easier development of applications as the *declarative hints* hide much complexity in task configuration and deployment, and *ii)* better openness and interoperability as NGSI is a standardized open data model which has been used for information sharing and data source integration in more than 30 cities. The architecture is vertically divided as Cloud, edge nodes, and devices. The FogFlow framework consists of three logically separated divisions: service management, data processing, and context management. The service management division is typically in the Cloud, including task designer, topology master (TM), and the Docker image repository. The service developers can design, submit, and manage specific services in docker image styles. TM is mainly for service orchestration and translating services and topology into a concrete deployment plan. The data processing division is responsible for performing data processing tasks assigned by TM, consisting of a set of workers. The context management division is usually deployed in the Cloud, including a set of IoT Brokers, a centralized IoT Discovery, and a Federated Broker. These modules decide the data flow across tasks through NGSI and also manage the system contextual data, such as the available resources of devices. These three divisions enable FogFlow to orchestrate dynamic data processing flows over the Cloud and edges.

In all, Fogflow supports heterogeneity, QoS, scalability, and openness. However, it still lacks modules supporting the mobility feature.

3.1.5 Fog05. A. Corsaro *et al.* [11] introduce Fog05, a fog infrastructure that unifies computing, networking, and storage fabrics across the Cloud, edge, and things. Fog05 adopts a novel server-less data-centric architectural approach that is claimed to be stable, secure, and reliable. Fog05 can run either on a traditional operating system

or on a Trusted Execution Environment (TEE). Only the software running on TEE can communicate with the external network. Fog05 proposes some key abstractions in order to provision, manage, and orchestrate applications. Fog05 entities are either atomic entities such as a virtual machines, containers, unikernels, or a directed acyclic graph of entities. Entities and atomic entities use Finite State Machine (FSM) to define the legal state transitions. Another key abstraction is Fog05 resources. Fog05 resources are expressed as URLs to represent everything in the system. All resources are stored in a distributed key-value store. This store provides an eventual consistency semantics with built-in versioning and provides the user with a way to end-to-end memory virtualization and storage. These key abstractions become the foundations for the Fog05 programming framework. The Fog05 agent/node is the core component representing a manageable resource in the system, having two separate stores, the actual state, and the desired state. The actual state can be revised by the local node, while the desired state can be written by anybody to cause state transitions, such as provisioning a VM. The functionalities of agents are controlled by plug-ins to manage things, such as entities, networks, and operating systems.

According to the discussion above, Fog05 supports heterogeneity, QoS, scalability, and openness, while no migration module exists to support the mobility function in the framework.

3.2 Fog Programming: Industry Attempts

The industrial companies independently propose their fog computing platforms. Some are extending their cloud platform to support edge services, while others are new players in this field. In this subsection, we investigate the prominent fog/edge programming models proposed by different companies, such as Cisco [1], Amazon [2], and Azure [3]. There are other fog programming solutions in industry, such as ioFog, Edgex, but they are not as popular as the above frameworks.

3.2.1 Cisco Edge Computing Framework. The authors from Cisco proposed the fog computing concept in 2012 [7]. The Cisco edge computing framework consists of three major architectural shifts. One is decomposition, which means separating the control/signaling and user/data in network functions for optimization of resources. The second one is disaggregation into software and hardware. The last one is infrastructure convergence: mobile networks share a common 5G core-based infrastructure for operational practices. Cisco also proposes Edge Fog Fabric (EFF) [1] that is an open architecture IoT platform for industrial customers. EFF is a modular microservice architecture platform to filter, aggregate, and compress data at the edge or the cloud level as appropriate for the operation. Cisco EFF key components contain system administrator, dataflow editor, system monitor, message broker, links, and historian database.

According to the discussed above, Cisco Edge Computing Framework supports scalability and QoS, but it does not support heterogeneity as it needs hardware devices from Cisco and has no mobility support with migration modules. Besides, it is not a standardized and open-source platform.

3.2.2 AWS IoT Greengrass. Amazon's Greengrass system also points out to a model in which the same development technology can be utilized both in the Cloud and in IoT devices. AWS IoT Greengrass brings local compute, messaging, data caching, sync, and machine learning (ML) interface capabilities to edge devices [2]. In Greengrass, the programming platform is Amazon's Lambda. The developers can develop the code in the Cloud and deploy it seamlessly to the devices with AWS Lambda. One relevant module is AWS IoT Greengrass Core, acting as a hub to communicate with other devices that run Amazon FreeRTOS or AWS IoT Device SDK. The core enables local execution of AWS Lambda code, messaging, data caching, and security.

Therefore, AWS IoT Greengrass supports heterogeneity, scalability, and QoS. However, there is no migration module to support mobility. It is also not an open-source project.

3.2.3 Azure IoT Edge. Azure IoT Edge is a complete edge management service built on Azure IoT Hub, which enables deploying Artificial Intelligence (AI) or other services on IoT edge devices via standard containers [3]. Azure IoT Edge consists of three components. The first one is IoT Edge modules, which are containers that run Azure services, third-party services, or users' code. Modules are deployed to IoT Edge devices and execute locally on those devices. The second one is the IoT Edge runtime, running on each IoT Edge device and managing the modules deployed on each device. The third one is a cloud-based interface enabling the user to monitor and manage IoT Edge devices remotely.

Azure IoT Edge is also an open-source project, and it supports heterogeneity, scalability, and QoS as well. It still does not support the mobility feature.

4 OPEN RESEARCH CHALLENGES

Fog computing brings various new characteristics, as well as several challenges. Regarding fog programming, there are many issues and research opportunities in this attractive field. In this section, we point out several future issues and challenges.

Proactive vs. Reactive Service Migration. The components and services of fog applications may need to move at runtime due to the mobility of devices dynamically. Mobile users offload intensive computations to adjacent fog nodes while they can move to a new fog system during the computation offloading period. How to move the services smoothly and guarantee QoS is also another open research question [22]. Due to the mobility patterns of IoT/end-user devices and fog nodes, the migration engine module is needed to keep the fog services closer to the corresponding end-user. Migration decisions (i.e., when and where) should be made, proactively or reactively. Proactive migration means to make them before the actual handover, based on the stochastically predicted user's mobility. Reactive migration means that the user only migrates the services when reaching the next fog access point. There is a trade-off between proactive and reactive migration. Indeed, the overall performance can be improved by moving the service to the next fog access point beforehand, and thus ready to run on the target node immediately. However, as the prediction of user mobility is a stochastic process, the success rate may be low, causing a possible risk to overall performance. Instead, reactive migration is determinism, and there are no unpredicted risks. Combined approaches may be a proper and relevant investigation in this field of research.

Fog Infrastructure Virtualization. Due to the heterogeneity of hardware devices in the infrastructure, there should be a systematic virtualization method applied in fog systems, exploiting the start-of-the-art virtualization technologies, such as VM, container, and unikernel [20]. Choosing appropriate virtualization techniques can substantially improve performance and applicability. Virtualization technologies should suit as many different types of physical nodes as possible to host fog services. Besides, designing communication interfaces and abstraction layers to leverage the interoperability between different virtualization technologies is a necessity.

Resource-aware Dynamic Reconfiguration. Fog systems contain numerous heterogeneous end devices with different computation and storage resources in different layers [23]. They are more dynamic than cloud systems due to device mobility and resource consumption. These devices need to interact with each other (the Cloud and fog nodes) to manage the resources to guarantee the SLA. In this case, the dynamic resource management mechanism should be included in fog programming frameworks, considering both centralized and distributed peer-to-peer resource management frameworks. Besides, resource management policies/algorithms are also highly demanded in fog programming for resource optimization. These policies/algorithms should not only fulfill the SLA at individual device level, but also for global resource optimization level considering saving the resources such as energy and bandwidth. To achieve that, novel dynamic resource-aware software reconfiguration techniques should be devised.

Artificial Intelligence (AI). AI has been widely used in the Cloud for image processing and natural language processing, which takes a huge amount of data and computation resources for training the models [8]. There is a transformation pushing intelligence towards the edge side, such as traditional base stations used for communication, data processing, and control services. The issue about how to coordinate the AI running model between the Cloud and fog sides remains to be solved. Moreover, AI can be used to manage and distribute the resources, and make the decision more efficiently at different levels. Utilizing different AI algorithms to solve the optimization problems, such as load offloading and resource usage optimization, is still unsolved. The fog programming framework can include some modules to support AI for easy development, deployment, and management. The modules supporting AI can either reside centrally in the Cloud or be distributed to the fog nodes (distributed AI), which is also a challenging research question.

P2P Fog Computing and Blockchain. Existing fog programming frameworks are mainly based on client-server models. The IoT/end users (clients) utilize cloud computing or fog computing (servers) to process their requests. The fog nodes can offload tasks among themselves or to the Cloud. However, there is a trend coming to the era of ubiquitous computing and storage. Using peer-to-peer (P2P) resource sharing frameworks can exploit the potential of fog systems. First, each fog node can share its resources with peers without involving third parties using P2P protocols. Second, a fog programming framework that supports P2P can handle the heterogeneity of fog nodes as fog nodes with the same P2P protocols can communicate with each other compatibly, regardless of heterogeneity of nodes. Besides, P2P fog programming can further leverage the benefits of blockchain. For example, resource pricing and incentive methods for resource sharing can be applied in P2P fog programming. Blockchain could be a choice for the resource pricing model and keeping track of fog resources transactions using smart contracts. Moreover, blockchain could also contribute to the security and privacy of the fog systems, such as device identity management [18]. The mining task of forming blockchain (i.e., Proof of Work) demands substantial computing resources. MEC can utilize distributed computing to solve the energy consumption issue.

5 CONCLUSIONS

In this paper, we discussed the characteristics of fog programming and its requirements based on the common fog architectural model and fog applications. In addition, we investigated state-of-the-art in fog programming frameworks, proposed both in academia and industry. We summarized studied fog programming frameworks according to the described characteristics. Finally, we presented the open research questions in fog programming. We argued that resource-aware reconfiguration, distributed intelligence, and P2P fog computing and blockchain would play an essential role in the future of fog programming research.

REFERENCES

[1] 2019. Retrieved August 27, 2019 from https://www.cisco.com/c/en/us/solutions/service-provider/edge.html
[2] 2019. Retrieved August 27, 2019 from https://aws.amazon.com/greengrass/
[3] 2019. Retrieved August 27, 2019 from https://docs.microsoft.com/en-us/azure/
[4] N. Abbas et al. 2018. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal* (Feb 2018).
[5] OpenFog Reference Architecture. 2017. OpenFog Consortium Technical Report v.1.0.
[6] A.Yousefpour et al. 2019. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. of Systems Architecture* (2019).
[7] Flavio Bonomi et al. 2012. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*.
[8] Z. Chen et al. 2019. An Artifitial Intelligence Perspective on Mobie Edge Computing. *in Proc. IEEE International Conference on Smart Internet of Things* (2019).
[9] B. Cheng et al. 2018. FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. *IEEE Internet of Things Journal* (April 2018).
[10] C.Mouradian et al. 2018. A Comprehensive Survey on Fog Computing:State-of-the-Art and Research Challenges. *IEEE Comm. Surveys Tutorials* (2018).
[11] A. Corsaro et al. 2018. fogO5: Unifying the computing, networking and storage fabrics end-to-end. In *2018 3rd Cloudification of the Internet of Things (CIoT)*.
[12] D.Lan et al. 2019. Latency Analysis of Wireless Networks for Proximity Services in Smart Home and Building Automation:The Case of Thread. *IEEE Access* (2019).
[13] N. K. Giang et al. 2015. Developing IoT applications in the Fog: A Distributed Dataflow approach. In *2015 5th International Conference on the Internet of Things*.
[14] Kirak Hong et al. 2013. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. ACM, 15–20.
[15] H. Liu et al. 2018. Mobile Edge Cloud System: Architectures, Challenges, and Approaches. *IEEE Systems Journal* (Sep. 2018).
[16] Lei Liu et al. 2019. Vehicular Edge Computing and Networking: A Survey. *arXiv preprint arXiv:1908.06849* (2019).
[17] Y. Liu et al. 2019. A Data-Centric Internet of Things Framework Based on Azure Cloud. *IEEE Access* (2019).
[18] Yunlong Lu et al. 2019. Blockchain and Federated Learning for Privacy-preserved Data Sharing in Industrial IoT. *IEEE Transactions on Industrial Informatics* (2019).
[19] M.Mukherjee et al. 2018. Survey of Fog Computing:Fundamental,Network Applications,and Research Challenges. *IEEE Comm. Surveys Tutorials* (2018).
[20] Morabito et al. 2018. Consolidate IoT edge computing with lightweight virtualization. *IEEE Network* (2018).
[21] Nath et al. 2018. A survey of fog computing and communication: current researches and future directions. (2018).
[22] C. Puliafito et al. 2017. Fog Computing for the Internet of Mobile Things: Issues and Challenges. In *2017 IEEE International Conference on Smart Computing*.
[23] Enrique S. et al. 2016. Incremental Deployment and Migration of Geo-distributed Situation Awareness Applications in the Fog. In *Proceedings of DEBS*.
[24] A. Taherkordi et al. 2018. Future Cloud Systems Design: Challenges and Research Directions. *IEEE Access* (2018).
[25] Yangui et al. 2016. A platform as-a-service for hybrid cloud/fog environments. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks*.