Software-Defined Fog Orchestration for IoT Services

Renyu Yang[12]***, Zhenyu Wen***[3]***, David McKee***[1]***, Tao Lin***[4]***, Jie Xu***[12]***, Peter Garraghan***[5]

[1]*UNIVERSITY OF LEEDS, UK*

[2]*BEIHANG UNIVERSITY, CHINA*

[3]*UNIVERSITY OF EDINBURGH, UK*

[4]*EPFL, SWITZERLAND*

[5]*LANCASTER UNIVERSITY, UK*

Abstract

The Internet of Things (IoT) interconnects physical objects including sensors, vehicles, and buildings into a virtual circumstance, resulting in the increasing integration of Cyber-physical objects. The Fog computing paradigm extends both computation and storage services in Cloud computing environment to the network edge. Typically, IoT services comprise of a set of software components running over different locations connected through datacenter or wireless sensor networks. It is significantly important and cost-effective to orchestrate and deploy a group of microservices onto Fog appliances such as edge devices or Cloud servers for the formation of such IoT services. In this chapter, we firstly discuss new characteristics and open challenges of realizing Fog orchestration for IoT services before summarizing the fundamental requirements. We propose a software-defined orchestration architecture that decouples software-based control policies with the dependencies and operations of heterogeneous hardwares. This design can intelligently compose and orchestrate thousands of heterogeneous Fog appliances. Specifically, we provide a resource filtering based resource assignment mechanism to optimize the resource utilization and fair resource sharing among multi-tenant IoT applications. Additionally, we exploit a component selection and placement mechanism for containerized IoT microservices to minimize the latency by harnessing the network uncertainty and security whilst considering different application requirement and appliance capabilities. We finally describe a Fog simulation scheme to simulate the above procedure by modeling the entities, their attributes and actions. Our practical experiences show that the proposed parallelized orchestrator can reduce the execution time by 50% with at least 30% higher orchestration quality. We believe that software-defined orchestration is a promising paradigm and will play an increasingly important role in future Fog ecosystem.

1. Introduction

The proliferation of the Internet and increasing integration of physical objects spanning sensors, vehicles, and buildings have resulted in the formation of Cyber-physical environments that encompass both physical and virtual objects. These objects are capable of interfacing and interacting with existing network infrastructure, allowing for computer-based systems to interact with the physical world, thereby enabling novel applications in areas such as smart cities, intelligent transportation, and autonomous vehicles. Explosive growth in global data generation across all industries has led to research focused on effective data extraction from objects to gain insights to support Cyber-physical system design. IoT services typically comprise a set of software components running over different geographical locations connected through networks (i.e. 4G, wireless LAN, Internet etc.) that exhibit dynamic behavior in terms of workload internal properties and resource assumption. Systems such as datacenters and wireless sensor networks underpin data storage and compute resources required for the operation of these objects.

A new computing paradigm − Fog computing − further evolves Cloud computing by placing greater emphasis of computation and data storage at the edge of the network, allowing for reduced latency and response delay jitter for applications[1][25]. These characteristics are particularly important for latency-sensitive applications such as gaming and video streaming. In this way, the data processing can be greatly decentralized by exploiting compute capacities from not only Cloud infrastructures, but from the IoT network itself. In this environment, existing applications and massive physical devices can be leveraged as fundamental services and appliances respectively. They are composed in a mash-up style (i.e., applications are developed using contents and services available online [56]) in order to control development cost and reduce maintenance overhead. IoT services which involve a great number of data-stream and control flows across different regions that require real-time processing and analytics are especially suitable to this style of construction and deployment. In this context, *orchestration* is a key concept within distributed systems, enabling the alignment of deployed applications with user business interests.

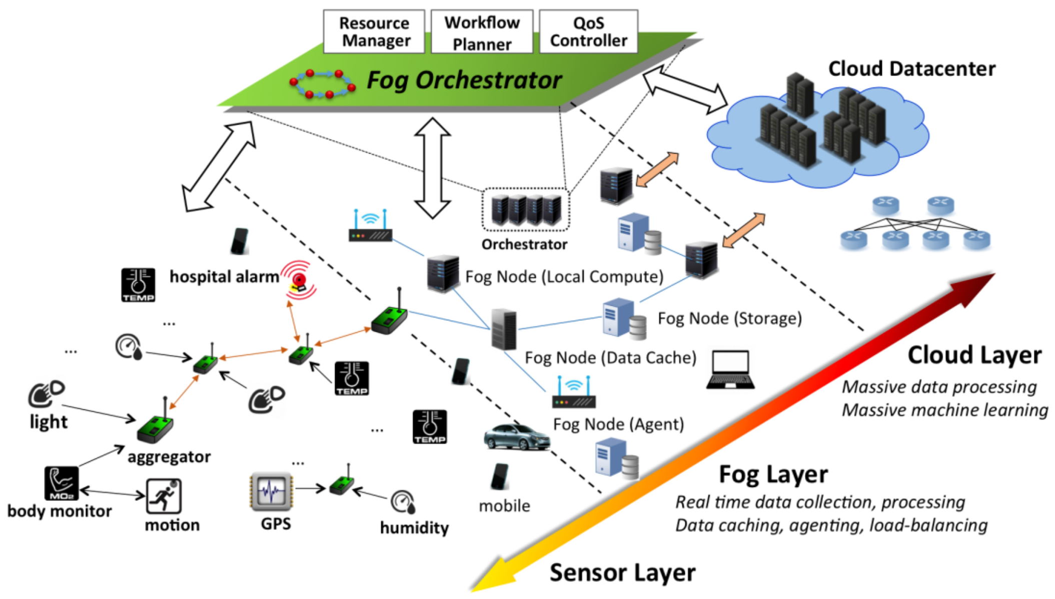

Figure 1. An orchestration scenario for an e-Health service: different IoT appliances (diverse types of sensors and Fog nodes) are orchestrated as a workflow across all layers of Fog architecture. Several candidate objects can potentially provision similar functionality. The Fog orchestrator acts as a controller deployed on a workstation or Cloud datacenter and across all organization layers based on global information. Its primary responsibility is to select resources and deploy the overall service workflow according to data security, reliability, system efficiency requirements. It is noteworthy that the orchestrator is a centralized controller only at a conceptual level and might be implemented in a distributed and fault-tolerant fashion, without introducing a single point of failure.

A Motivating Example. *Smart cities* aim to enhance the quality of urban life by using technology to improve the efficiency of services to meet the needs of residents. To this end, multiple information and communication technology (ICT) systems need to be integrated in a secure, efficient and reliable way in order to manage city facilities effectively. Such systems consist of two major components: (1) sensors integrated with real-time monitoring systems, and (2) applications integrated with the collected sensor (or device) data. Currently, IoT services are rudimentary in nature, and only integrate with specific sensor types. This is resultant of no existing universally agreed standards and protocols for IoT device communication, and represents a challenge towards achieving a global ecosystem of interconnected *things*.

To address this problem, an alternative approach is to use an IoT service orchestration system to determine and select the best IoT appliances for dynamic composition of holistic workflows for more complex functions. As shown in Figure 1, the proposed orchestrator manages all layers of an IoT ecosystem to integrate different standalone appliances or service modules into a complex topology. An appropriate combination of these standalone IoT services can be used to facilitate more advanced functionality, allowing for reduced cost and improved user experience. For example, *mobile health* sub-systems are capable of remote monitoring, real-time data analysis, emergency warning, etc. Data collected from wearable sensors that monitor patient vitals can be continuously sent to data aggregators and, in the event of detection of abnormal behavior, hospital personnel can be immediately notified in order to take appropriate measures.

While such functionality can be developed within a standalone application, this provides limited scalability and reliability. The implementation of new features leads to increased development efforts and risk of creating a monolithic application incapable of scaling effectively due to conflicting resource requirements for effective operation. For reliability, increased application complexity leads to tedious, time-consuming debugging. The use of orchestration allows for more flexible formation of application functionality to scale and it also decreases the probability of failure correlation between application components.

At present, orchestration within Cloud computing environments predominantly address issues of automated interconnection and interaction in terms of deployment efficiency and resource satisfaction from the perspective of the Cloud provider [8][11]. However, these works do not consider the effects of network transmission characteristics outside the operational boundary of the datacenter. In reality, the heterogeneity, mobility and the dynamicity introduced by edge devices within Fog environments are greater than those found within Cloud environments. Additionally, the emergence of 4G or 5G techniques are still far from mature in terms of response latency and energy efficiency. This has resulted in increasing network uncertainty which may incur tailed execution and security hazards. In this context, it is significantly important to take all these factors into account within the automated resource provisioning and service delivery. Therefore, the *Fog orchestrator* should provide (a) a centralized arrangement of the resource pool, mapping applications with specific requests and an automated workflow to physical resources in terms of deployment and scheduling, (b) workload execution management with runtime QoS control such as latency and bandwidth usage; and (c) time-efficient directive operations to manipulate specific objects.

In this chapter, we propose a scalable software-defined orchestration architecture to intelligently compose and orchestrate thousands of heterogeneous Fog appliances (devices, servers). Specifically, we provide a resource filtering based resource assignment mechanism to optimize the resource utilization and fair resource sharing among multi-tenant IoT applications. Additionally, we propose a component selection and placement mechanism for containerized IoT microservices to minimize the latency by harnessing the network uncertainty and security whilst considering different applications' requirement and appliances' capabilities. We describe a Fog simulation scheme to simulate the above procedure by modeling the entities, their attributes and actions. We then introduce the results of our practical experiences on the orchestration and simulation.

2. Scenario and Application

2.1 Concept Definition

Prior to discussing technical details of orchestration, we first introduce a number of basic terms and concepts.

***Appliance*:** Appliance is the fundamental entity in the Fog environment. Appliances include Fog *Things*, *Fog nodes* and Cloud servers. *Things* are defined as networked devices including sensors and devices with built-in sensors which can monitor and generate huge amount of data. Cloud servers store the data and provide parallelized capability of computation. It is noteworthy that a Fog node is defined as a particular equipment or middleware residing within the midst of edge things and the remote Cloud. It serves as an agent that collects data from a set of sensors, which is then transmitted to a centralized computing system that locally caches data and performs load balancing.

IoT Microservice: It is a software unit that provisions a specific type of functionality. For instance, there are a number of demands for data collecting, data streaming gateway or routing, data pre-processing, user data caching, load balancing, firewall services, etc. These functionalities are independently executed, encapsulated into a container and then placed onto an appliance (except for sensors that simply generate data). Additionally, several candidate objects potentially provision similar functionality and one of them will be eventually selected and deployed as the running instance.

IoT Service (IoT Application): A complete IoT application typically consists of a group of IoT microservices. All microservices are inter-connected to form a function chain that best serve user's requirements. Formally, an IoT application can be depicted as a DAG workflow, where each node within the workflow represents a microservice. An example is illustrated in Figure 2, where the aforementioned e-Health application can be divided into many independent but jointly-working microservices.

***Fog Orchestration*:** The orchestration is a procedure that enables the alignment of deployed IoT services with users' business interests. Fog orchestration manages the resource pool; provides and underpins the automated workflow with specific requests of IoT service satisfied; and conducts the workload execution management with runtime QoS control. A full discussion of this concept can be found within Section 4.

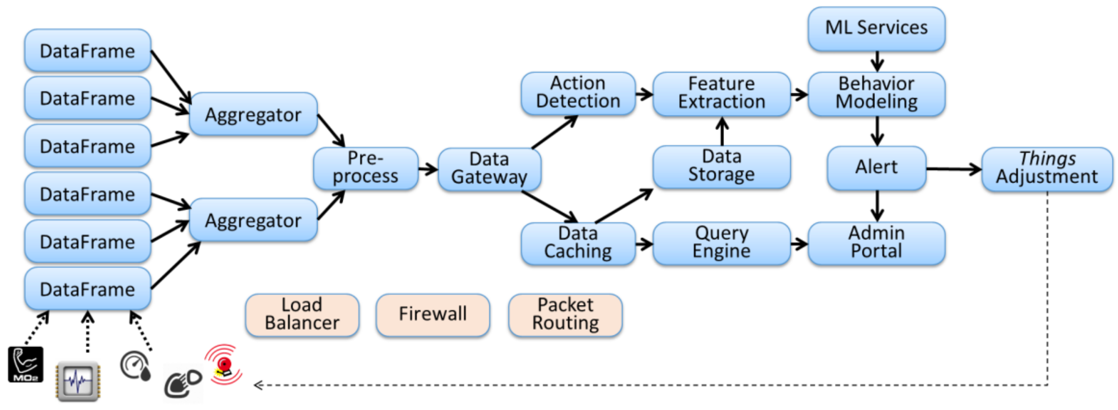

Figure 2. e-Health system workflow and containerized microservices in the workflow

2.2 Fog-enabled IoT Application

Traditional Web-based service applications are deployed on servers within Cloud datacenters that are accessed by end devices such as tablets, smart phones and desktop PCs. Similarly to Web-based service applications, the Cloud provisions centralized resource pools (compute, storage) in order to analyze collected data and automatically trigger subsequent decisions based on a pre-defined system logic. The most significant difference, however, is the use of *Fog nodes* that transmit data to Cloud datacenters. For example, the vast majority of wearable sensor data is collected and pre-processed by smart phones or adjacent workstations. This can either significantly reduce transmission rates or improve their reliability.

Table 1: Comparison between web-based application and Fog-enabled IoT application

Attributes	Web-based	Fog-enabled
Architecture	Cloud + devices	Cloud + Fog + Things
Communication	Centralized	Hybrid
Interfaces	WSDL/SOAP protocol web service	MQTT protocol [40] Lightweight API
Interoperability	Loosely-decoupled	Extremely Loosely-decoupled
Reliability	Medium	Low

We summarize the main differences between Web-based and IoT applications in Table 1.

First, IoT communication is performed using a hybrid centralized-decentralized approach depending on context. Most message exchanges between sensors or between a sensor and the cloud are performed using fog nodes. Purely centralized environments are ill-suited for applications that have soft and hard real-time requirements. For example, neighboring smart vehicles need to transfer data between other vehicles and traffic infrastructure to prevent collisions. Such a system was piloted in New York City using Wi-Fi to enable real-time interactions to assist drivers in navigating congestion and to communicate with pedestrians or oncoming vehicles [13]. Furthermore, given the huge number of connected devices, the data volume generated and exchanged over an IoT network is predicted to become many orders of magnitude greater than that of conventional Web-based services, resulting in significant scalability challenges.

Interoperability is another aspect where Web-based and IoT applications diverge. Software-defined networking technologies enable the decoupling of software control and heterogeneous hardware operations. This approach provides an opportunity to dynamically achieve different quality levels for different IoT applications in heterogeneous environments [14]. Moreover, application-level interoperability benefits from Web technologies, such as the RESTful architecture, that provide a high level of interoperability. Using these technologies and the MQTT messaging protocol [40], an abundance of programming APIs can be distributed across entire fog domains and utilized to increase the flexibility of loosely coupled management [15]. Lightweight APIs, such as RESTful interfaces, result in agile development and simplified orchestration with enhanced scalability when composing complex distributed workflows.

A third aspect is reliability. Physical systems make up a significant part of IoT applications, thus the assumptions that can be made regarding fault and failure modes are weaker than those for Web-based applications. IoT applications experience crash and timing failures stemming from low-sensor battery power, high network latency, environmental damage, etc.[57][58]. Furthermore, the uncertainty of potentially unstable and mobile objects increases difficulties in predicting and capturing system operation. Therefore, an IoT application workflow's reliability needs to be measured and enhanced in more elaborate ways.

2.3 Characteristics and Open Challenges

The diversity among Fog nodes is a key issue - location, configuration, and served functionalities of Fog nodes all dramatically increase this diversity. This raises an interesting research challenge, namely how to optimize the process of determining and selecting the best software components onto Fog appliances to compose an application workflow whilst meeting non-functional requirements such as network latency, QoS, etc. We outline and elaborate on these specific challenges as follows:

Scale and complexity. With the increase of IoT manufacturers developing heterogeneous sensors and smart devices, selecting optimal objects becomes increasingly complicated when considering customized hardware configurations and personalized requirements. For example, some applications can only operate with specific hardware architectures (e.g., ARM, Intel) or operating systems, while applications with high security requirements might require specific hardware and protocols to function. Not only does orchestration cater to such functional requirements, it must do so in the face of increasingly larger workflows that change dynamically. The orchestrator must determine whether the assembled systems comprising of Cloud resources, sensors, and Fog nodes coupled with geographic distributions and constraints are capable of provisioning complex services correctly and efficiently. In particular, the orchestrator must be able to automatically predict, detect, and resolve issues pertaining to scalability bottlenecks which may arise from increased application scale.

Security criticality. In the IoT environment, multiple sensors, computer chips, and communication devices are integrated to enable the overall communication. A specific service might be composed of a multitude of objects, each deployed within different geographic locations, resulting in an increased attack vector of such objects. Fog nodes are the data and traffic gateway that is particularly vulnerable to such attacks. This is especially true in the context of network-enabled IoT systems, whose attack vectors can range from human-caused sabotage of network infrastructure, malicious programs provoking data leakage, or even physical access to devices. A large body of research focuses on cryptography and authentication towards enhancing network security to protect against Cyber-attacks [16]. Furthermore, in systems comprising of hundreds of thousands electronic devices, how to effectively and accurately evaluate the security and measure its risks is critically important in order to present a holistic security and risk assessment [17]. This becomes challenging when workflows are capable of changing and adapting at runtime. For these reasons, we believe that approaches capable of dynamically evaluating the security of dynamic IoT application orchestration will become increasingly critical for secure data placement and processing.

Dynamicity. Another significant characteristic and challenge for IoT services is their ability to evolve and dynamically change their workflow composition. This is a particular problem in the context of software upgrades through Fog nodes or the frequent join-leave behavior of network objects which will change its internal properties and performance, potentially altering the overall workflow execution pattern [50]. Moreover, handheld devices inevitably suffer from software and hardware aging, which will invariably result in changing workflow behavior and its properties. For example, low-battery devices will degrade the data transmission rate; and unexpected slowdown of read/write operations will manifest due to long-time disk abrasions. Finally, the performance of applications will change due to their transient and/or short-lived behavior within the system, including spikes in resource consumption or data generation [57]. This leads to a strong requirement for automatic and intelligent re-configuration of the topological structure and assigned resources within the workflow, and importantly, that of *Fog nodes*.

Fault diagnosis and tolerance. The scale of a Fog system results in increased failure probability. Some rare-case software bugs or hardware faults which do not manifest at small-scale or testing environments have a debilitating effect on system performance and reliability. For instance, the straggler problem [18] occurs when a small proportion of these tasks experience abnormally longer execution compared with other sibling tasks from the same parallel job, leading to extended job completion time. At the scale, heterogeneity, and complexity we are anticipating, it is very likely that different types of fault combinations will occur [19]. To address these, redundant replications and user-transparent fault-tolerant deployment and execution techniques should be considered in orchestration design.

2.4 Orchestration Requirements

According to the discussed user cases within Fog environments, a user firstly provides a specification of their requirement that explicitly describes the basic topological workflow (e.g., from the data collection to the final monitoring system) and the detailed requirements for each workflow node in terms of data locality, response latency, reliability tolerance level, minimum security satisfactory level, etc. In this context, the ultimate objective of the Fog orchestration is to transform the logical workflow design from the user perspective into the physically executable workflow over different resources of Fog appliances. In this procedure, the requirements that should be at least satisfied can be primarily summarized as follows:

1) Exploit Fog appliance heterogeneity. The orchestrator should recognize the diversity of edge devices, Fog nodes and Cloud servers, and fully exploit the capabilities of CPU, memory, network and storage resources over the Fog layers. At present, neither the conventional cluster management systems [43]-[48] nor the container management frameworks [1][2][3] can efficiently detect and leverage the edge resources due to the deficient design of current inter-action protocol and state management mechanism.

2) Enable IoT appliance and application operation. Unawareness of resource availability and IoT application status make it unfeasible to manipulate any instructions of resource allocation or parameter tuning at runtime. This is also a fundamental step for realizing the interoperations among different appliances in the workflow. Loosely-coupled functions or APIs should be designed and accessed via pre-defined interfaces over the network, which enables the re-use and composition to form a chain of functions.

3) Conduct workflow planning optimization and network latency-aware container placement. For general purposes, the orchestrator is expected to support topology-based orchestration standard TOSCA [6]. Afterwards, according to the topological workflow, how to choose the most suitable microservice from the candidates and how to choose the most suitable Fog appliances for hosting the selected containerized microservices are two research problems. Due to the physically widespread in a local or wide area network of Fog appliances, the software services are ideally deployed close to the data sources or data storage in order to reduce the transmission latency. With other factors considered, the orchestrator must support a comprehensive placement strategy whilst being aware of appliances' characteristics such as physical capabilities, locations, etc.

4) Leverage real-time data and learning techniques for optimization and simulation. Performance-centric and QoS-aware learning can significantly steer the effectiveness and efficiency of resource allocation, container placement and the holistic orchestration. This is highly dependent of data-driven approach and machine learning techniques.

3. Architecture: A Software-Defined Perspective

3.1 Solution Overview

To fulfill the aforementioned requirements, the initial steps are resource allocation and microservice-level planning before those microservices are deployed and launched. An exemplified construction problem is to firstly find a suitable microservice instance into container, and then find a physical entity with adequate resources to host those containers. Namely, after obtaining a candidate *i* that can serve the functionality from a candidate list **I** for a specific type of microservice *t* (which is the node within the whole topology **T** of IoT application), we deploy the selected instance into a container which is hosted by a physical machine or portable device *r* from the resource set **R**. The objective is to maximize a utility function (*utilFunc*) that describes the direction of resource selection and container placement (such as minimizing the performance interference whilst maximizing the security and reliability) under QoS and capacity constraints.

maximize:

$$\sum_{t \in T} utilFunc(i, r), i \in C_t, r \in R$$

***subject to**: QoS(i, r),* $\mathrm{i} \in C_t,\ \mathrm{r} \in R$

Cap(r), $\mathrm{r} \in R$

To satisfy the application-specific needs with hard or soft constraints, and the platform-level fairness of allocations among different IoT applications, it is highly preferable to accurate sort out the appliances that can best serve each IoT application. Also, for online decision making, real-time or sometimes faster-than real time is urgently required. In some cases, orchestration would be typically considered computationally intensive, as it is extremely time-consuming to perform combination calculation considering all specified constraints and objectives.

After resource selection and allocation, we can obtain an optimal or near-optimal placement scheme based on current system status before the application deployment. After the IoT application is deployed, workload running status and system states should be timely monitored and collected to realize dynamic orchestration at run-time with QoS guaranteed. Meanwhile, with the huge amount of data generated, data-driven optimization and learning-based tuning can facilitate and drive the orchestration intelligence.

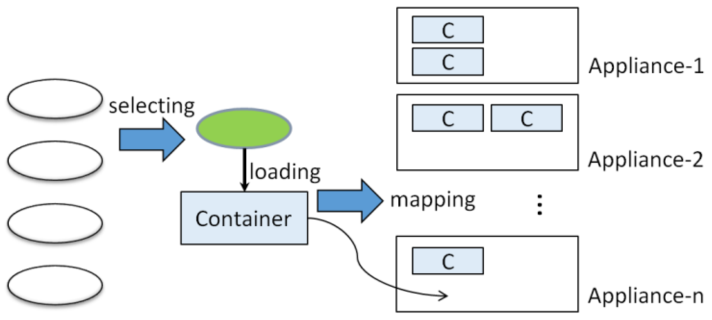

Figure 3. mapping between microservice candidate, containerized microservice instance, and the physical appliances.

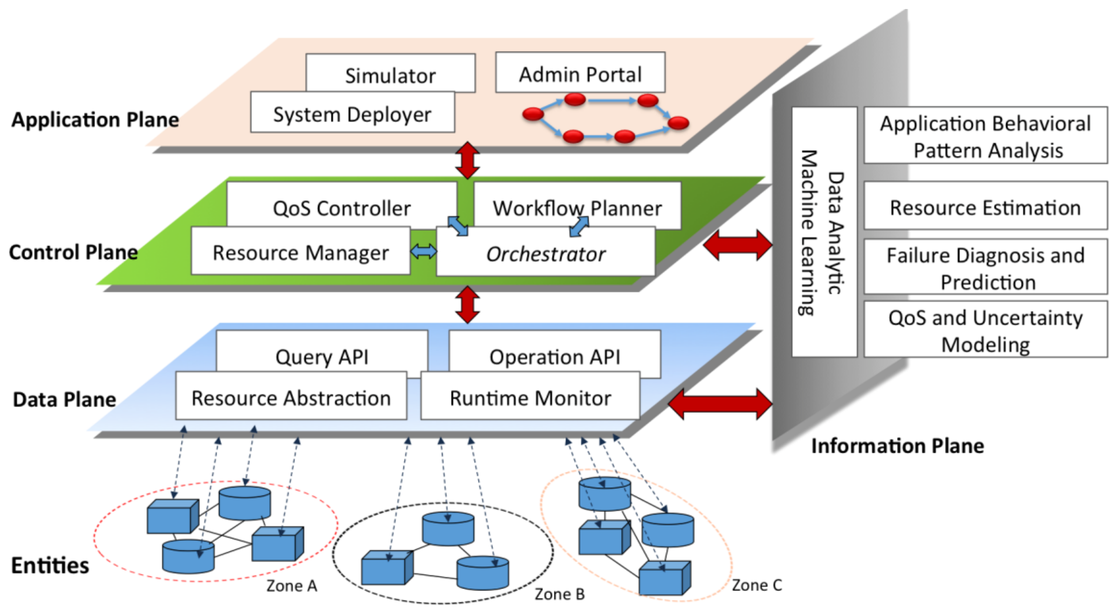

Figure 4. Fog Orchestration Architecture

3.2 Software-Defined Architecture

A significant advantage of a software-defined solution [60] is the de-coupling of the software-based control policies and the dependencies on heterogeneous hardware. On one hand, along with the rapid development of mobile and embedded operating system, the programming API and virtualization techniques can be greatly utilized to increase the flexibility of manipulation and management. Virtualization, through the use of containerization, can provide minimized granularity of resource abstraction and isolated execution environment. Resource operations are exposed as interoperable system APIs and accessible to upper frameworks or administrators. On the other hand, the orchestration controls the software-defined architecture. In order to mitigate the overloaded functionalities of control plane in previous architecture, the information plane is de-coupled from the control plane. The independent information plane can therefore provision more intelligent ingredients into the orchestration and resource management by integrating pluggable libraries or learning frameworks. Additionally, we adopt container technology to encapsulate each task or IoT microservice. Containers can ensure the isolation of running microservices and create a development and operation environment that is immune to system version revision, sub-module updates.

As shown in Figure 4, the Fog orchestration framework is incorporated with the emerging networking and resource management technologies. We design the layered architecture according to the popular SDN reference architecture [60][61]. The main components are described as follows:

Data Plane. The first responsibility of data plane is to regulate and abstract the resources from heterogeneous Fog entities. It also provisions an easily accessed Application Programming Interface (API) for resource management and application runtime control across the entire Fog system. Furthermore, the monitored system states such as resource usage, application-specific metrics etc. are collected and maintained in the data plane. The build-in query APIs are provided for visualization or administration.

Control Plane. The control plane is the decision making plane that works on the basis of control logic in the overall architecture. It dominates both data flow and control flow, and inter-connects with the deployment module and operations of underlying entities and running appliances. The orchestrator mainly takes charge of resource management, workflow planning and runtime QoS control:

- *Resource Manager.* The resource manager is responsible for the resource pre-filtering according to the basic demands and constraints in the requests and available resources in the Fog environments. In addition, after the final decision made by the planning step, the resource manager also takes the responsibility of resource binding and isolation against other applications. It also takes charge of the elastic resource provisioning during the appliances' execution. They are depicted in Section 4.1.
- *Workflow Planner.* The planner calculates the optimal mapping of candidate micro-services, containerized appliances and the hosting entities. We will detail the relevant techniques in Section 4.2.
- *QoS Controller.* The controller dynamically tunes the allocated resource, the orchestration strategy at run-time with the QoS guaranteed. They are detailed in Section 4.3.

The control module can be implemented in distributed (with each sub-orchestrator managing its own resource partitions without global knowledge) or centralized (with all resource statuses in the central orchestrator), or a hybrid way for the consideration of scalability and dependability.

Information Plane. The information plane lends itself as a vertical within this architecture, provisioning data-driven supporting and intelligent solutions. By exploiting the stored sensed information and system real-time statuses, the data analytic and machine learning sub-module can abstract and analyze the application's behavior pattern and give more accurate resource estimation and location preference in the resource allocation. Also, with the aid of big data analytics, this module can build the performance model based on the QoS and network uncertainty modeling, and diagnose system failures preventing them from the regular orchestration.

Application Plane. The application-level plane firstly contains an administration portal that aggregates and demonstrates the collected data, and allows for visualized interaction. Additionally, a containerized deployer is integrated in this plane, providing cost-effective Fog service deployment. It automatically deploys the planned IoT application or services into the infrastructure and continently upgrade current services. The simulation module by leveraging the collected data, modeling the user and appliance's characteristics, resource allocation and placement policies, and the fault patterns etc.

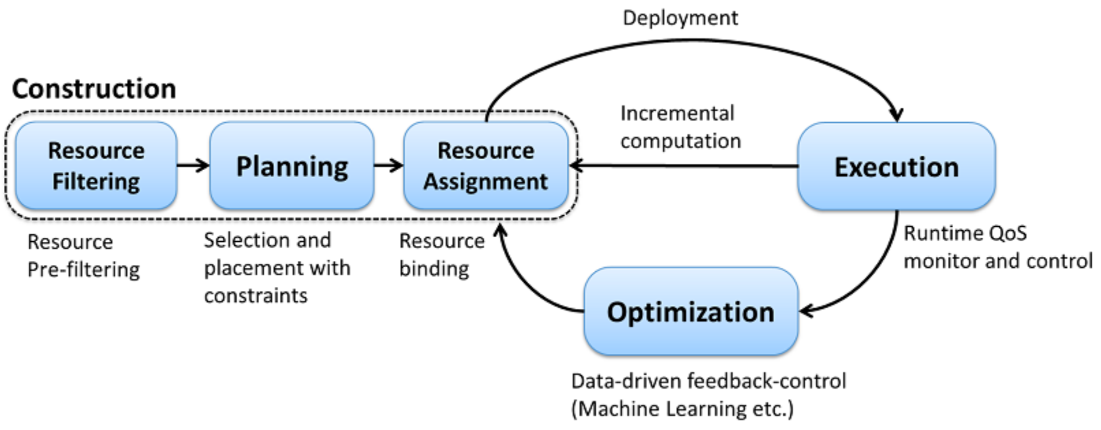

Figure 5. Orchestration within the life-cycle management: Main functional elements in our Fog Orchestrator: resource allocation for filtering and assigning the most suitable resources to launch appliances; the planning step for selection and placement; runtime monitoring and control during execution; and the optimization step to make data-driven decision based on adaptive learning techniques.

4. Orchestration

In this section, we discuss the detailed research sub-topics that we believe are key to tackling the challenges outlined above. As shown in Figure 5, within the life-cycle management, these include the resource filtering and assignment in resource allocation phase; the optimal selection and placement in planning phase; dynamic QoS monitoring and guarantees at runtime through incremental processing and re-planning; and big data driven analytics and optimization approaches that leverage adaptive learning such as machine learning or deep learning to improve orchestration quality and accelerate the optimization for problem solving. The functionality decomposition based on the life-cycle perspective is orthogonal to the software-defined architecture. In particular, the construction and execution part are mainly implemented in the control plane and all functionalities are underpinned by the information plane and data plane. The data-driven optimization is associated with the information plane. The application deployment and overall administration will manifest in the application plane.

4.1 Resource Filtering and Assignment

Fog infrastructures typically consist of heterogeneous devices with diverse capacity of compute, memory and storage size. Therefore, resource allocation is a very fundamental procedure for system entities to be launched and executed. One of the responsibilities of Fog orchestrator is to optimize the use of both Cloud and Fog resources on top of Fog applications. There are two main tasks in the Fog eco-systems: containers which encapsulate the microservices and run across tiers in Fog eco-system and computation-intensive tasks that run in parallel to process the huge volume of data. The resource requests proposed by both sides needs to be timely dispatched and handled in the resource manager. Meanwhile, the resource manager will trigger new iterations of resource allocation by leveraging recently aggregated resources (such as CPU, memory, disk, network bandwidth, etc.). Allocated resources will be guaranteed and reserved for the requesting application. Additionally, the resource manager keeps track of the task progress and monitors the service status.

In essence, the procedure of resource allocation is the matchmaking or mapping between the requirements from the applications that are waiting for execution and the available resources that are dispersed over the Fog environment. Therefore, the resource allocation sub-system should fully exploit the diversities and dynamicity of computing clusters at massive scale to improve throughput of computation jobs and reduce the negative impact of unexpected latencies stemming from the jitter of network and occurrence of ineffective queuing. Only through recognizing the accurate targets for placement can the scheduler mitigate the computation straggler or promote resource utility and compaction. Considering heterogeneity in Fog [50] is extremely important when conducting the mapping and such heterogeneity leads to different resource capacities and unique machine characteristics. We need to find out the machines that are determined to be most suitable for specific purposes. In Cloud datacenter, this can be typically done through a multi-step filtering process that comprehensively considers estimated load, correlative workload performance, and queue states. Similarly, fog resource allocation should be conducted from the following three aspects:

Label-based resource representation. In the Fog environment, there is a considerably growing trend of the resource heterogeneity stemming from the rapid development of IoT devices and new types of hardware. This growth provisions more choices for upper applications. For example, hardware such as GPU, FPGA, TPU (Tensorflow Process Unit, NVM, etc.) make it possible to accelerate the computations and data processing in deep learning, vision recognition etc. Moreover, for those applications that involve a great many of geo-distributed data access and processing, it is preferable to require the affinities between tasks and the stored raw data. Therefore, consideration of such data and resource affinity is extremely meaningful especially for latency-sensitive applications. In the procedure of resource filtering, we should firstly sort out the collections of destinations that have sufficient available resources and satisfy all specific requirements. To this end, we can adopt the label-based matchmaking between the requests and resources. Formally, the request can be expressed as an n-tuple: $\mathbf{ResReq_i} = (\mathbf{Req_i}, \mathit{LatBound}, \mathit{LocPref})$ where $\mathbf{Req_i} = \{Req_i^1, \ldots Req_i^d\}$ represents the requested resource amount of different labels. The label represents a specific description of resource dimension or a certain constraint. Latency requirement *LatBound* specifies the detailed acceptance level of the latency and response time and the *LocPref* indicates the preferable execution locations according to the data distribution and processing requirements. On the other hand, the Fog resources existing in an Fog appliance *e* can be described as $\mathbf{Res}_e = \{Res_e^1, \ldots Res_e^d, \mathit{Priority}\}$ where Res_e^i represents the value of ith label and *Priority* attribute implies the prioritized level according to the appliance type.

Candidate filtering and resource assignment. Combined all requests with available resources from all active entities, the resource manager tries to rank the candidate Fog appliances according to system metrics such as resource status, device load, queuing states, etc. An intuitive latency-aware resource allocation strategy is to firstly allocate resources of edge devices to microservices requiring lower-delay, and microservices with lower level of delay requirement are then allowed onto entities such as Fog node or Cloud resources. The resultant collection of candidate entities will be further considered in the next phase; and the final resource binding is conducted once the component selection and placement is determined. It is worth noting that the pre-filtering and candidate selection can dramatically reduce the aimless and unnecessary calculations. Therefore this step cannot be ignored for Fog orchestration.

Node and executor management. Node Manager in conventional cluster management systems is an agent that runs on each individual node, and serves two purposes: the resource monitoring and the executor (worker process, VM or Docker container) control. The latter is made possible through the aid of process launch, monitor, and isolation etc. Compared with the clusters in Cloud data centers, the network condition, vulnerability of physical devices and communication stability are entirely different, resulting in the disability to directly apply all current methods of node management in the resource management. For example, to handle frequent variations of node status, the resource manager should reserve the allocated resources instead of directly killing all running processes in face of frequent node joining-in or departing and the node anomaly stemming from temporary network delays or transient process crashes etc. Additionally, the high-rate data exchange between the Cloud and edge devices is fundamental to underpin the IoT applications. Long-thin connections between mobile users and remote cloud have to be long-lived maintained and isolated for the sake of network resource reservation.

4.2 Component Selection and Placement

The recent trend in composing Cloud applications is driven by connecting heterogeneous services deployed across multiple datacenters. Similarly, such a distributed deployment helps improve IoT application reliability and performance. However, it also exposes appliances and microservices to new security risks and network uncertainty. Ensuring higher levels of dependability is a considerable challenge. Numerous efforts [20][21] have focused on QoS-aware composition of native VM-based Cloud application components, but neglect the proliferation of uncertain execution and security risks among interactive and interdependent components within the DAG workflow of an IoT application.

Cost model. As we discussed in section 3.1, the composition and placement of components can be regarded as an optimization problem. To be precise, the optimization includes two main factors in order to capture the increasing characteristics in Fog environment - the network uncertainties and service dependability (such as security and reliability risks). We assume that the uncertainty and security of microservice s_i are defined as Unc_i and Sec_i respectively. Importantly, there are parameters to represent the dependency relation between two adjacently chained microservices. For example, $DSec_{ij}$ represents the risk level of interconnecting s_i and s_j. Similarly, the uncertainty level between s_i and s_j is described as $DUnc_{ij}$. Thus, the optimization objectives can be formalized as:

$$\begin{cases} argmax_{s_i \in S, s_j \in S} \sum Sec_i + \sum DSec_{ij} \\ argmin_{s_i \in S, s_j \in S} \sum Unc_i + \sum DUnc_{ij} \end{cases}$$

As the equation shows that we need to maximize the security whilst minimizing the impact of uncertainties on the services. There are two ways to solve this problem: 1) optimize a utility function which includes both objectives with different weights; 2) set a constraint for one of the objective and then optimize the other one. For some multi-objective problems, it is unlikely to find a solution that has optimal values for all objective functions simultaneously. Alternatively, a feasible solution is the Pareto optimal [38] where none of the objectives can be improved without degrading an objective. Therefore, IoT service composition is to find a Pareto optimal solution which meets users' constraints.

Parallel computation algorithm. Optimization algorithms or graph-based approaches are typically both time-consuming and resource-consuming when applied into a large-scale scenario, and necessitate parallel approaches to accelerate the optimization process. Recent work [22] provides possible solutions to leverage an in-memory computing framework to execute tasks in a Cloud infrastructure in parallel. However, how to realize dynamic graph generation and partitioning at runtime to adapt to the shifting space of possible solutions stemming from the scale and dynamicity of IoT services remains unsolved.

Late calibration. To ensure near-real-time intervention during IoT application development, a potential approach could be *correction mechanisms* that could be applied even when sub-optimal solutions are deployed initially. For example, in some cases, if the orchestrator finds a candidate solution that approximately satisfies the reliability and data transmission requirements, it can temporarily suspend the search for further optimal solutions. At runtime, the orchestrator can then continue the improvement of decision results with new information and a re-evaluation of constraints, and make use of task and data migration approaches to realize workflow re-deployment.

4.3 Dynamic Orchestration with Runtime QoS

Apart from the initial placement, the workflow dynamically changes due to internal transformations or abnormal system behavior. IoT applications are exposed to uncertain environments where variations in execution are commonplace. Due to the degradation of consumable devices and sensors, capabilities such as security and reliability that initially were guaranteed will vary accordingly, resulting in the initial workflow being no longer optimal or even totally invalid. Furthermore, the structural topology might change in accordance to the task execution progress (i.e. a computation task is finished or evicted) or will be affected by the evolution of the execution environment. Abnormalities might occur due to the variability of combinations of hardware and software crashes, or data skew across different management domains of devices due to abnormal data and request bursting. This will result in unbalanced data communication and subsequent reduction of application reliability. Therefore, it is essential to dynamically orchestrate task execution and resource reallocation.

QoS-aware control and monitoring. To capture the dynamic evolution and variables (such as dynamic evolution, state transition, new operations of IoT, etc.), we should predefine the quantitative criteria and measuring approach of dynamic QoS thresholds in terms of latency, availability, throughput, etc. These thresholds usually dictate upper and lower bounds on the metrics as desired at runtime. Complex QoS information processing methods such as hyper-scale matrix update and calculation would give rise to many scalability issues in our setting.

Event streaming and messaging. Such performance metric variables or significant state transitions can be depicted as system events, and *event streaming* is processed in the orchestration framework through an event messaging bus, real-time publish-subscribe mechanism or high-throughput messaging systems (e.g., Apache Kafka[4]), therefore significantly reducing the communication overheads and ensuring responsiveness. Subsequent actions could be automatically triggered and driven by Cloud engine (e.g., Amazon Lambda service[5]).

Proactive recognition. Localized regions of self-updates become ubiquitous within Fog environments. The orchestrator should record staged states and data produced by Fog appliances periodically or in an event-based manner. This information will form a set of time series of graphs and facilitate the analysis and proactive recognition of anomalous events to dynamically determine such hotspots [23]. The data and event streams should be efficiently transmitted among Fog appliances, so that system outage, appliance failure, or load spikes will rapidly feedback to the central orchestrator for decision making.

4.4 Systematic Data-Driven Optimization

IoT applications include numerous geographically distributed devices that produce multidimensional, high-volume data requiring different levels of real-time analytics and data aggregation. Therefore, data-driven optimization and planning should have a place in the orchestration of complex IoT services.

Holistic cross-layer optimization. As researchers or developers select and distribute applications across different layers in the fog environment, they should consider the optimization of all overlapping, interconnected layers. The orchestrator has a global view of all resource abstractions, from edge resources on the mobile side to compute and storage resources on the cloud data center side. Pipelining the stream of data processing and the database services within the same network domain could reduce data transmission. Similar to the data-locality principle, we can also distribute or reschedule the computation tasks of fog nodes near the sensors rather than frequently move data, thereby reducing latency. Another potential optimization is to customize data relevant parameters such as the data-generation rate or data-compression ratio to adapt to the performance and assigned resources to strike a balance between data quality and specified response-time targets.

Online tuning and History-Based Optimization (HBO). A major challenge is that decision operators are still computationally time consuming. To tackle this problem, online machine learning can provision several online training (such as classification and clustering) and prediction models to capture the constant evolutionary behavior of each system element, producing time series of trends to intelligently predict the required system resource usage, failure occurrence, and straggler compute tasks, all of which can be learned from historical data and a history-based optimization (HBO) procedure. Researchers or developers should investigate these smart techniques, with corresponding heuristics applied in an existing decision-making framework to create a continuous feedback loop. Cloud machine learning offers analysts a set of data exploration tools and a variety of choices for using machine learning models and algorithms [24].

4.5 Machine-learning for Orchestration

Although current deployment of orchestration has been explored by human experts and optimized by some hand-crafted heuristics algorithms, it is still far from meeting the challenge of automated management and optimization. Learning-based methods, or more precisely, machine learning (ML), open a new door to tackle the challenges raised from IoT orchestration. ML approaches automatically learn underlying system patterns from historical data and explores the latent space of representation. It not only significantly reduces human labor and time, but is capable of dealing with multi-dimension and multi-variety data in dynamic or uncertain environments.

Metric Learning. The current evaluation of a given workflow normally involves the knowledge of human experts as well as the numerical characteristic, quality of hardware, etc. However, the dynamicity within heterogeneous environments makes it infeasible and inaccurate to handcraft standard metrics for the evaluation over different orchestrations. Instead, Metric Learning[62] aims to automatically learn the metric from data (e.g., hardware configuration, historical records, runtime logs), providing convenient proxies to evaluate the distance among objects for better complex objects manipulation. Regarding orchestration scenarios, it is interesting if the algorithm can consider the topology layout of data during the learning.

Graph Representation Learning. Connecting Metric Learning with the graph structure provides an orthogonal direction for current methodologies of resource filtering and resource allocation. However, traditional orchestration approaches normally use user-defined heuristics to explore the optimal deployment over the original graph with structural information. Those summary statistics again significantly involve hand-engineered features which are inflexible during learning process and design phase. By using Graph Representation Learning (GRL), we can represent or encode the complex structural information of a given workflow [63]. Furthermore, we can either use it for better exploitation of the machine learning models, or provide more powerful workflow metrics for better orchestration. For example, the current label-based resource representation may easily encounter the issue of sparse one-hot representation, and it would be more efficient to represent different hardware/services in a low- and dense- latent space [64].

Reinforcement Learning. Design of good heuristic or approximation algorithms for NP-hard combinatorial optimization problems often requires significant specialized domain knowledge.

However, traditional algorithms are often insufficient in such knowledge when extreme complicated IoT applications are orchestrated. Given the efficient representation of the workflow, graph embedding[65] shows the potentiality of using neural network with Reinforcement Learning (RL) methods to incrementally construct an optimal solution in dynamic environments. There are a great number of research opportunities since current Deep RL solutions of combinatorial optimization merely focus on the standard traveling salesman problem (TSP) whose scenario is much simpler than the IoT application orchestration.

5. Fog Simulation

5.1 Overview

Simulation is an integral part of the process of design and analyzing systems in engineering and manufacturing domains. There is also a growing trend to analyze distributed computing systems using technologies such as CloudSim[26] or SEED[27] for example to study resource scheduling or analyze the thermodynamic behavior of a data center[28]. In these contexts, it is essential to understand the categories of simulation[29]:

- **Discrete event simulation** *(DES)*[30][31] in which the system events are modeled as a discrete sequence.
- **Continuous simulations** [29] which are typically constructed based on ordinary differential equations (ODEs) which represent properties of physical systems.
- **Stochastic simulation** [32] such as monte-carlo methods.
- **Live, Virtual and Constructed (LVC) simulation** providing interactive simulations often supported by technologies as the IEEE HLA 1516 [33].

The LVC category of simulation introduces the concept of co-simulation whereby two or more simulations are run concurrently to explore interactions and complex emergent behaviors. In the domain of engineering, co-simulation is typically limited to a handful of simulations due to the complexity of integrating simulations with differing time-steps and simulations of differing types. With the rise of IoT, Industry 4.0, and the Internet of Simulation (IoS) paradigm [34][35] there is a growing trend to explore the use of simulation as a technology for facilitating online decision making. There are several key factors that must be considered in this context:

- Inter-tool compatibility between simulations and also between simulations and other tools/systems.
- Performance and scalability in terms of the size of the simulations that need to be run and the time needed to do so. For example the use cases for simulation with IoT may require simulation to perform in near real-time.
- Understanding the complexity of the models involved in the simulations and understanding the trade-offs between complexity and abstraction [36].

The remainder of this section explores the application of simulation across two key areas: simulation for IoT and also online analysis as part of a decision making system such as an orchestrator.

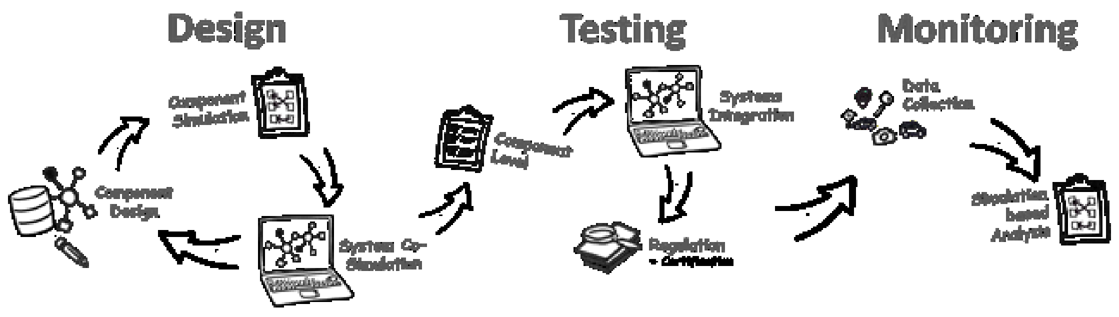

Figure 6. The workflow of system simulation

5.2 Simulation for IoT application in Fog

Within the traditional IoT sector there are two broad situation categories in which simulation is typically used:

(a) Design and analysis of devices which may include control systems or 3D modeling during the design phases of the engineering process. This may occur at the component, system, or system of systems level. During the early stages the simulations are typically abstract concepts of functional behavior which are then iteratively, ideally using co- simulation, expanded upon to provide detailed insight into specific behaviors and emergent interactions. This process in the context of traditional engineering lifecycles is depicted in Figure 6 where the traditional V-model of component and system-level design is integrated with component and systems level testing.

(b) Analysis of data as described by the Industry 4.0 movement [37]. In this context, data is collected by IoT sensors and systems and fed into analysis systems which increasingly involve simulation. An example is the automotive industry where data is collected from vehicles as they are used and fed back to the manufacturer. In the automotive space this is typically limited to periodic data collection during servicing but there is a growing trend with connected vehicles to provide more frequent or even a continuous data feedback to the manufacturer. As Figure 6 shows, data can be collected from deployed systems or devices and fed into simulations for further analysis which may or may not occur at the design phase of another engineering lifecycle iteration.

Figure 7 depicts the abstract layers of IoS: virtual or federated cloud, traditional service layers (IaaS, PaaS, SaaS, FaaS, etc.). On top of this are the simulation layers with virtual things deployed as simulations (SIMaaS) and then virtual things becoming virtual systems at the workflow service layer (WFaaS). IoT lends itself as another vertical within this model providing physical things that can be connected to the virtual system workflows.

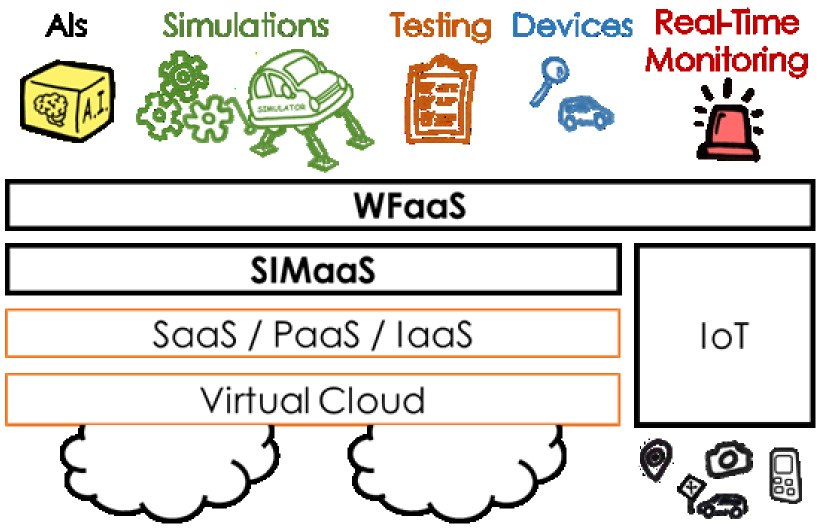

Figure 7. The architecture of Simulation as a Service

In order to precisely model and build the Fog simulator, we need to depict the attributes and behaviors of Fog appliances and the services. For a specific appliance, we include the appliance type, physical capacities such as CPU cores, RAM, storage, up/downlink bandwidths, and the connection status such as which appliances it is inter-connected with and the latency information of connections among different appliances. The attributes also contains the hardware specifications (i.e., GPU, FPGA, TPU, etc.), software specifications (i.e., OS version, libraries, etc.), and other machine attributes that comprehensively described in [7]. All these are implemented as different labels. To simulate the service, we should provide the interfaces to define the IoT service DAG topology and the dependencies among different microservices. The detailed resource requests and other requirements of each microservice (the vertex in the DAG) are also determined as inputs within the simulator. For example, the main attributes of action detection microservice in Figure 2 can be depicted as follows:

```
{"ActionDetection": {
    "Resource": {
      "CPU": 2 vcores,
      "RAM": 2GB,
      "DISK": 10GB
    },
    "Priority": "Medium",
    "Security": "High",
    "Computation": "Medium",
    "Latency": "Low"
}}
```

5.3 Simulation for Fog Orchestration

Moving away from the traditional IoT sector and the common uses of simulation there are two growing trends for simulation adoption as part of online decision making systems. The first trend is the automated parameterization and deployment of simulations based on data to provide immediate data analytics to decision makers. Secondly, the use of real-time simulation is in-the-loop with other systems. There are two key challenges with both trends which are the need for timeliness whilst dealing with the scale of the systems being modeled.

An example in the context of orchestration is the use of simulation as part of both the optimization and planning phases. During system execution, the collected data is used to update relevant simulation models in terms of system behavior (this could include network latencies, server performance, etc.). The optimization process is able to use the simulation as a data representation for the machine learning algorithms which in turn feeds into the planning phase. For example, using the genetic algorithm (GA) based approaches used in Section 6, simulations can be run with each individual and generation to provide a more detailed and informed fitness function. Although this has the ability to significantly increase the capability of the system, there remains a significant trade-off in deciding the complexity of the simulation versus the performance that is required.

6. Early Experience

6.1 Simulation-based Orchestration

Design Overview: Based on the design philosophy and methods discussed, we propose a framework that can efficiently orchestrate Fog computing environments. As demonstrated in Figure 8, in order to enable planning and adaptive optimization, a preliminary attempt was made to manage the composition of applications in parallel under a broad range of constraints. We implement a novel parallel genetic algorithm based framework (GA-Par) on Spark to handle orchestration scenarios where a large set of IoT microservices are composed. More specifically, in our GA-based algorithm, each chromosome represents a solution of the composed workflow and the gene segments of each chromosome represent the IoT microservices. We normalize the utility of security and network QoS of IoT appliances into an objective fitness function within GA-Par to minimize the security risks and performance degradation.

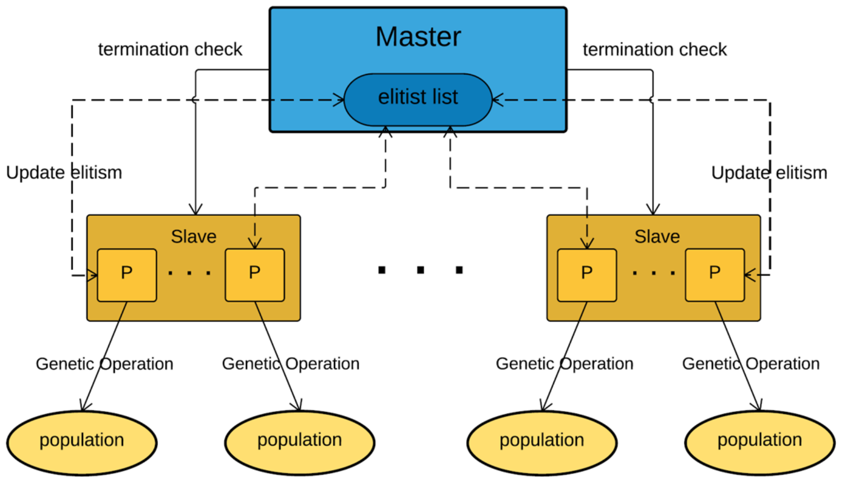

Figure 8. A parallel GA solver to accelerate the handling of optimization issues raised in the planning and optimization phase

To strike a balance between accuracy and time efficiency, we separate the total individual population into parallel compute partitions dispersed over different compute nodes. In order to maximize parallelism, we set up and adjust the partition configuration dynamically to make partitions fully parallelized whilst considering data shuffling and communication cost with the topology change. To guarantee optimal results can be gradually obtained, we dynamically merge several partitions into a single partition and then re-partition it based on runtime status and monitored QoS. Furthermore, the quality of each solution generation can be also maintained by applying an elitist method, where the local elite results of each partition will be collected and synthesized into global elite. The centralized GA-Par master will aggregate the full information at the end of each iteration, and then broadcasts the list to all partitions to increase the probability of finding a globally optimal solution.

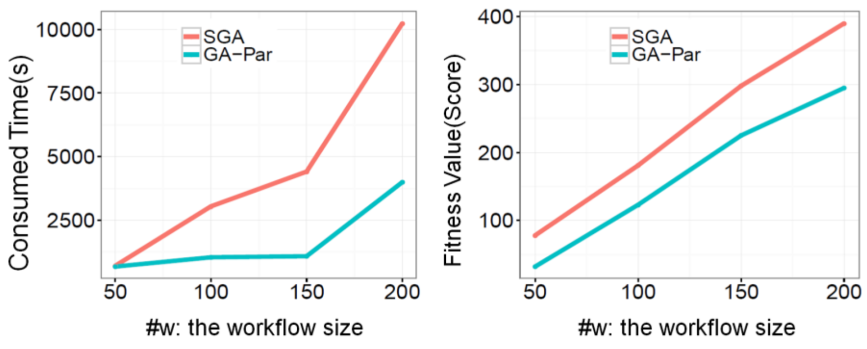

Figure 9. Initial results demonstrate the proposed approach can outperform a standalone genetic algorithm in terms of both time and quality aspects

Experiment Setup: To address data skew issues, we also conduct a joint data-compute optimization to repartition the data and reschedule computation tasks. We perform some initial experiments on 30 servers hosted on Amazon Web Services (AWS) as the Cloud datacenter for the Fog environment. Each server is hosted as an r3.2xlarge instance with 2.5GHz Intel Xeon E5-2670v2 CPUs, 61GB RAM, and 160GB disk storage. We use simulated data below to illustrate the effectiveness of composition given IoT requirements. For this, we randomly select four types of orchestration graphs with 50, 100, 150, and 200 workflow nodes, respectively. For each node within a workflow, we stochastically prepare 100 available IoT appliances as simulated agents. The security levels and network QoS levels are randomly assigned to each candidate agent. We compare our GA-Par with a standalone genetic algorithm (SGA). The metrics quality, execution time and fitness score (with lower values indicating better results) are used to evaluate SGA and GA-Par.

Evaluation: As can be observed in Figure 9, GA-Par outperforms SGA. The time consumption of GA-Par has been significantly reduced to nearly 50% of that of SGA, while the quality of appliance selection in GA-Par is always at least 30% higher than that of SGA. However, the scalability of our current approach is still slightly affected by increasing numbers of components and requests, indicating that we still need to explore opportunities for incremental re-planning and on-line tuning to improve both time-efficiency and effectiveness of IoT orchestration.

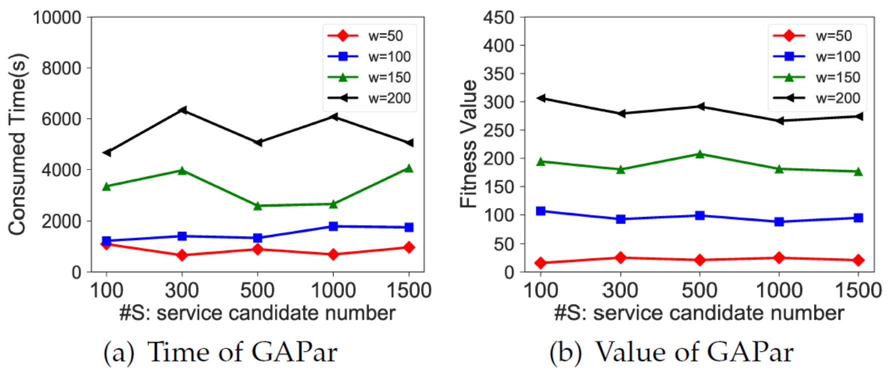

Figure 10. Initial results of GA-Par in terms of both time and quality aspects

Figure 10 demonstrates the experimental results under different workflow size and candidate number of microservice by using GA-Par. We can observe that with the increment of workflow size, the time consumption increase accordingly. The linear increase demonstrates that the growth of task number in the workflow will augment the searching range to find optimal solution, thereby taking longer time to finish the overall computation. In Figure 10(a), the number of microservice candidate number is not an obvious factor that influence on the time consumption. The consumed time slightly fluctuates when the topology and size of the workflow is determined. Apparently, given the workflow size w and each node in the orchestrated workflow has s candidates, the searching space is s^w. Thus, the impact of s on the consumed time will not be as significant as that of w. Likewise, a similar phenomenon can be observed in terms of the fitness calculation. In particular, the increased workflow size will naturally degrade the optimization effectiveness given the fixed setting of the total population. Compared with a smaller-scale workflow, larger workflows with soaring number are less likely to converge and obtain the optimal result once the population is set up.

Discussion: IoT services are choreographed through workflow or task graphs to assemble different IoT microservices. Therefore it is very worthwhile if we intend to obtain a precise decision and deploy IoT services in a QoS guaranteed, context- and cost-aware manner in spite of the magnitude of consumed time. In the context of pre-execution planning, static models and methods can deal with the submitted requests when the application workload and parallel tasks are known at design time. In contrast, in some domains, the orchestration is supplied with a plethora of candidate devices with different geographical locations and attributes. In the presence of variations and disturbances, orchestration methods should typically rely on incremental orchestration at runtime (rather than straightforward complete re-calculation by re-running static methods) to decrease unnecessary computation and minimize the consumed time.

Based on the time series of graphs, the similarities and dependencies between successive graph snapshots should be comprehensively studied to determine the feasibility of incremental computation. Approaches such as memorization, self-adjusting computation, and semantic analysis could cache and reuse portions of dynamic dependency graphs to avoid unnecessary re-computation in the event of input changes. Intermediate data or results should also be inherited as far as possible, and the allocated resources that have been allocated to the tasks should also be reused rather than be requested repeatedly. Through graph analysis, operators can determine which sub-graphs changes within the whole topology by using sub-graph partitioning and matching as an automated process that can significantly reduce overall orchestration time.

Another future work is to further parallelize the simulation and steer the complexity of GA-Par to achieve better scalability over large-scale infrastructures. Potential means include using heuristic algorithm or approximated computing into some key procedures of algorithm execution and value estimation.

6.2 Orchestration in Container-Based Systems

There are numerous research efforts and system works that address the orchestration functionality in Fog computing infrastructures. Most of them are based on the open source container orchestration tools such as Docker swarm[1], Kubernetes[2], and Apache Mesos marathon[3]. For instance, [39][40] gave an illustrative implementation of a Fog-oriented framework that can deliver containerized application onto datacenter nodes and edge devices such as Raspberry Pi. [41] comprehensively compared how these tools can meet the basic requirement to run IoT applications and pointed out Docker swarm is the best fit to seamlessly running IoT containers across different Fog layers. Docker swarm provisions robust scheduling that can spin up containers on hosts provisioned by using Docker-native syntax.

Based on the evaluation conclusions drawn by [41], we developed the proposed orchestrator as a standalone module that can be integrated with the existing Docker Swarm built-in modules. The orchestrator will overwrite the Swarm Scheduler and take over the responsibility of orchestrating containerized IoT services. Other sensors or Raspberry Pi devices are regarded as Swarm workers (Swarm Agents) and managed by the Scheduler. We integrate the proposed techniques such as label-based resource filtering and allocation, microservice placement strategy and parallelized optimization solver GA-Par with the provided *filter* and *strategy* mechanisms in Swarm Scheduler. As a result, whenever a new deployment request from the client is received, the Swarm Manager will forward it to the orchestrator, and the planner in the orchestrator will try to find the optimal placement to place and run containers on the suitable appliances. For the scalability and adaptability, we also design the planner as a pluggable module which can be easily substituted by different policies.

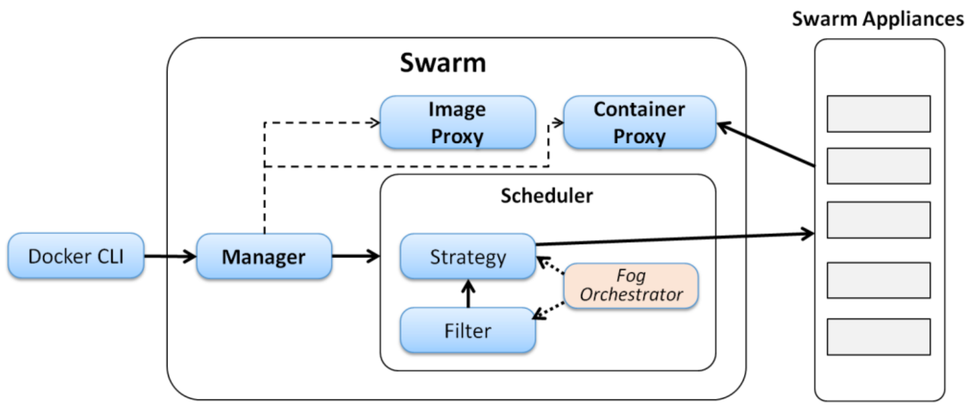

Figure 11. Fog Orchestrator with Docker Swarm

7. Discussion

The emergent of fog computing is one of particular interest within computer science. Within the coming decades the concept of the exascale system will become increasingly commonplace, interconnecting billions and tens of billions of different devices, objects, and things across a vast number of industries which will likely co-exist in some form of Fog eco-system. The challenges pertaining to security, reliability and scalability will continue to play a critical concern for designing these systems, as well as a number of additional characteristics:

Emergent behavior: Systems operating at scale have begun to increasingly operational characteristics not envisioned at system design-time conception. This is particularly true due to the massive heterogeneity and diversity of orchestrating various IoT services in tandem. Such emergent behavior encompasses both positive aspects such as emergent software [51], but also encompasses failures [18][52][53][54][55] unforeseen at design time. As a result, we will likely see increased use of meta-learning in order to dynamically adapt workflow orchestration in response to user demand and adversarial system conditions.

Energy usage: 10% of global electricity usage stems from ICT [59], and coupled with technological innovations and massive demands for services will likely see this electricity demand grow in both quantity and proportion. Given that these systems will operate services which produce vast quantities of emissions and economic cost, the environmental impact of these services executing within IoT will likely become increasingly important in the coming years. This is particularly true if legal measures are created and enforced to control and manage such power demand and carbon emissions.

Centralization vs. Decentralization: Within the past two decades, distributed systems have seen paradigms spanning clusters, web services, grid computing, Cloud computing, and Fog computing. It is noticeable that these paradigms appear to pivot between centralized [43][44][45][46] and decentralized architectures [47][48][49] in response to technological breakthroughs, combined with demands for new types of applications. We foresee that this pattern will continue to evolve, and potentially see the realization of massive-scale Fog eco-systems that are capable of both centralized and decentralized architectures combined together in response to demand.

Conclusion

Most recent research related to Fog computing explore architectures within massive infrastructures [25]. Although such work advances our understanding of the possible computing architectures and challenges of new computing paradigms, there are presently no studies of composability and concrete methodologies for developing orchestration systems that support composition in the development of novel IoT applications. In this chapter, we have outlined numerous difficulties and challenges to develop an orchestration framework across all layers within the Fog resource stack, and have described a prototype orchestration system that makes use of some of the most promising mechanisms to tackle these challenges.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program (2016YFB1000103), EPSRC (EP/P031617/1), National Natural Science Foundation of China (61421003), and the Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC).

REFERENCES

[1] Docker Swarm. https://github.com/docker/swarm

[2] Google Kubernetes. https://kubernetes.io/

[3] Mesos Marathon. http://mesos.apache.org/

[4] Apache Kafka. http://kafka.apache.org/

[5] AWS Lambda. https://aws.amazon.com/lambda/

[6] TOSCA: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

[7] Google Cluster Trace. https://github.com/google/cluster-data/

[8] P.Manh, A.Tchana, D.Donsez, N.Palma, V.Zurczak, P. Gibello. "Roboconf: a hybrid cloud orchestrator to deploy complex applications." In IEEE Cloud 2015.

[9] X.Wang, Z.Liu, Y.Qi, and J.Li. "Livecloud: A lucid orchestrator for cloud datacenters." In IEEE Cloudcom, 2012.

[10] C.Liu, B.Loo, and Y.Mao. "Declarative automated cloud resource orchestration." In ACM SoCC 2011.

[11] R. Ranjan, B.Benatallah, S.Dustdar, and M.Papazoglou. "Cloud resource orchestration programming: overview, issues, and directions." IEEE Internet Computing 19, no. 5,2015, pp 46-56.

[12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in ACM MCC, 2012, pp. 13–16.

[13] Kim M. D. Dikaiakos, A. Florides, T. Nadeem, and L. Iftode, "Locationaware services over vehicular ad-hoc networks using car-to-car communication," in IEEE JSAC, vol. 25, no. 8, pp. 1590–1602, 2007.

[14] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-ofthings," in IEEE NOMS, 2014, pp. 1–9.

[15] S. Nastic, S. Sehic, D. H. Le, H. L. Truong, and S. Dustdar, "Provisioning software-defined iot cloud systems," in IEEE FiCloud, 2014, pp. 288–295.

[16] R. Roman, P. Najera, and J. Lopez, “Securing the internet of things,” in IEEE Computer, vol. 44, no. 9, pp. 51–58, 2011.

[17] A. Riahi, Y. Challal, E. Natalizio, Z. Chtourou, and A. Bouabdallah, “A systemic approach for iot security,” in IEEE ICDCSS, 2013, pp. 351–355.

[18] P. Garraghan, X. Ouyang, R. Yang, D.Mckee, and J. Xu, “Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters,” [Online] in IEEE TSC, vol. PP, no. 99, pp. 1–1, 2016.

[19] R. Yang, Y. Zhang, P. Garraghan, Y. Feng, J. Ouyang, J. Xu, Z. Zhang and C. Li, “Reliable computing service in massive-scale systems through rapid low-cost failover,” [Online] in IEEE TSC, vol. PP, no. 99, pp. 1–1, 2016.

[20] Z. Zheng, Y. Zhang, and M. R. Lyu, “Investigating qos of real-world web services,” in IEEE TSC, vol. 7, no. 1, pp. 32–39, Jan 2014.

[21] Z. Wen, J. Cala, P. Watson, and A. Romanovsky, “Cost effective, reliable and secure workflow deployment over federated clouds,” [Online] in IEEE TSC, vol. PP, no. 99, pp. 1–1, 2016.

[22] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, “Graphx: Graph processing in a distributed dataflow framework,” in USENIX OSDI, 2014, pp. 599–613.

[23] K. Yamanishi and J. ichi Takeuchi, “A unifying framework for detecting outliers and change points from non-stationary time series data,” in ACM SIGKDD, 2002, pp. 676–681.

[24] Cloud machine learning. [Online]. Available: http://www.infoworld.com/article/3068519 /artificialintellegence/review-6-machine-learning-clouds.html

[25] S.Yi, C.Li, and Q.Li. A Survey of Fog Computing: Concepts, Applications, and Issues. In ACM MBDW, 2015, pp. 37-42

[26] Calheiros, Rodrigo N., et al. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." Software: Practice and experience 41.1 (2011): 23-50.

[27] P. Garraghan P, D. McKee, X. Ouyang, D. Webster, J.Xu. “SEED: A Scalable Approach for Cyber-Physical System Simulation.” IEEE Transactions on Services Computing, 9 (2), 2016, pp. 199-212.

[28] S. Clement, D. McKee, J.Xu. “A Service-Oriented Co-Simulation: Holistic Data Center Modelling Using Thermal, Power and Computational Simulations”. in IEEE/ACM UCC 2017.

[29] Law, Averill M., W. David Kelton, and W. David Kelton. Simulation modeling and analysis. Vol. 3. New York: McGraw-Hill, 2007.

[30] Fujimoto, Richard M. "Parallel discrete event simulation." In Communications of the ACM 33.10 (1990): 30-53.

[31] Varga, András. "Discrete event simulation system." In European Simulation Multiconference (ESM'2001). 2001.

[32] Ripley, Brian D. Stochastic simulation. Vol. 316. John Wiley & Sons, 2009.

[33] J.-R. Martinez-Salio, J.-M. Lopez-Rodriguez, D. Gregory, and A. Corsaro, "A comparison of simulation and operational architectures," in 2012 Fall Simulation Interoperability Workshop (SIW), Simulation Interoperability Standards Organization (SISO), 2012

[34] D.McKee, S.Clement, X. Ouyang, J.Xu, R. Romano, J. Davies. "The Internet of Simulation, a Specialisation of the Internet of Things with Simulation and Workflow as a Service (SIM/WFaaS)" in IEEE SOSE 2017, pp. 47-56.

[35] S. Clement; D. McKee, R. Romano; J. Xu; J. Lopez; D. Battersby. The Internet of Simulation: Enabling agile model based systems engineering for cyber-physical systems. In IEEE SoSE 2017.

[36] Singer, Mihaela. "Modelling both complexity and abstraction: A paradox?." Modelling and applications in mathematics education. Springer, Boston, MA, 2007. 233-240.

[37] Lasi, Heiner, et al. "Industry 4.0." Business & Information Systems Engineering 6.4 (2014): 239-242.

[38] Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". IEEE Transactions on Evolutionary Computation. 6 (2): 182. doi:10.1109/4235.996017.

[39] C.Pahl, S.Helmer, L.Miori, J.Sanin, B.Lee. "A container-based edge cloud PaaS architecture based on Raspberry Pi clusters." In IEEE FiCloud, 2016.

[40] P.Bellavista, and A.Zanni. "Feasibility of fog computing deployment based on docker containerization over raspberrypi." In ACM ICDCN, 2017.

[41] S.Hoque, M.Brito, A.Willner, O.Keil, T. Magedanz. "Towards container orchestration in fog computing infrastructures." In IEEE COMPSAC, 2017

[42] Message Queuing Telemetry Transport (MQTT) http://mqtt.org/

[43] Z. Zhang, C. Li, Y. Tao, R. Yang et al., "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," in VLDB, 2014.

[44] V. K. Vavilapalli, A. C. Murthy, C. Douglas et al., "Apache hadoop yarn: Yet another resource negotiator," in ACM SoCC, 2013.

[45] B. Hindman, A. Konwinski, M. Zaharia et al., “Mesos: A platform for fine-grained resource sharing in the data center.” in USENIX NSDI, 2011.

[46] A. Verma, L. Pedrosa, M. Korupolu et al., “Large-scale cluster management at google with borg,” in ACM EuroSys, 2015.

[47] E. Boutin, J. Ekanayake, W. Lin et al., “Apollo: scalable and coordinated scheduling for cloud-scale computing,” in USENIX OSDI, 2014.

[48] K. Karanasos, S. Rao et al., “Mercury: hybrid centralized and distributed scheduling in large shared clusters,” in USENIX ATC, 2015.

[49] Sun, X., Hu, C., Yang, R., Garraghan, P., Wo, T., Xu, J., Zhu, J. and Li, C., 2018. ROSE: Cluster Resource Scheduling via Speculative Over-subscription.

[50] B. Flavio, R.Milito, J.Zhu, and S.Addepalli. "Fog computing and its role in the internet of things." In ACM MCC, 2012.

[51] B Porter, M Grieves, RV Rodrigues Filho, D Leslie, REX: A Development Platform and Online Learning Approach for Runtime Emergent Software Systems, in USENIX ODSI, 2016.

[52] J. Dean, L. A. Barroso, The Tail at Scale, Communications of the ACM 56, 2013.

[53] B. Schroeder, G. A. Gibson, A Large-scale Study of Failures in High-Performance Computing Systems, IEEE Transactions on Dependable and Secure Computing, 2010

[54] P.Garraghan, R.Yang, Z.Wen, A.Romanovsky, J.Xu, R.Buyya, R.Ranjan. Emergent Failures: Rethinking Cloud Reliability at Scale. IEEE Cloud Computing, 2018

[55] Ouyang, X., Garraghan, P., Yang, R., Townend, P. and Xu, J., 2016, May. Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters. in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)

[56] J.Yu, B.Benatallah, F.Casati, and F.Daniel. "Understanding mashup development." IEEE Internet computing 12, no. 5 (2008).

[57] H.Madsen, B.Burtschy, G. Albeanu, and F.Vladicescu. "Reliability in the utility computing era: Towards reliable fog computing." In IEEE IWSSIP, 2013.

[58] L.Atzori, A.Iera, and G.Morabito. "The internet of things: A survey." Computer networks 54, no. 15 (2010): 2787-2805.

[59] M.P.Mills. The Cloud Begins with Coal. Big data, Big Networks, Big Infrastructure, and Big Power. Digital Power Group Report.

[60] D.Kreutz, F.Ramos, P.Verissimo, C.Rothenberg, S.Azodolmolky, and S.Uhlig. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103, no. 1 (2015): 14-76.

[61] Y.Jarraya, T.Madi, and M.Debbabi. "A survey and a layered taxonomy of software-defined networking." IEEE Communications Surveys & Tutorials 16.4 (2014): 1955-1980

[62] A.Bellet, A.Habrard, and M.Sebban. "A survey on metric learning for feature vectors and structured data." arXiv preprint arXiv:1306.6709 (2013).

[63] W. Hamilton, R.Ying, and J.Leskovec. "Representation Learning on Graphs: Methods and Applications." arXiv preprint arXiv:1709.05584 (2017).

[64] T.Mikolov, I.Sutskever, K.Chen, G.Corrado, and J.Dean. "Distributed representations of words and phrases and their compositionality." In NIPS 2013.

[65] I.Bello, H.Pham, Q.Le, M.Norouzi, and S.Bengio. "Neural combinatorial optimization with reinforcement learning." arXiv preprint arXiv:1611.09940(2016)