Enabling Workload Engineering in Edge, Fog, and Cloud Computing through OpenStack-based Middleware

GIOVANNI MERLINO, University of Messina
RUSTEM DAUTOV, Kazan Federal University
SALVATORE DISTEFANO, University of Messina and Kazan Federal University
DARIO BRUNEO, University of Messina

To enable and support smart environments, a recent ICT trend promotes pushing computation from the remote Cloud as close to data sources as possible, resulting in the emergence of the Fog and Edge computing paradigms. Together with Cloud computing, they represent a stacked architecture, in which raw datasets are first pre-processed locally at the Edge and then vertically offloaded to the Fog and/or the Cloud. However, as hardware is becoming increasingly powerful, Edge devices are seen as candidates for offering data processing capabilities, able to pool and share computing resources to achieve better performance at a lower network latency—a pattern that can be also applied to Fog nodes. In these circumstances, it is important to enable efficient, intelligent, and balanced allocation of resources, as well as their further orchestration, in an elastic and transparent manner. To address such a requirement, this article proposes an OpenStack-based middleware platform through which resource containers at the Edge, Fog, and Cloud levels can be discovered, combined, and provisioned to end users and applications, thereby facilitating and orchestrating offloading processes. As demonstrated through a proof of concept on an intelligent surveillance system, by converging the Edge, Fog, and Cloud, the proposed architecture has the potential to enable faster data processing, as compared to processing at the Edge, Fog, or Cloud levels separately. This also allows architects to combine different offloading patterns in a flexible and fine-grained manner, thus providing new workload engineering patterns. Measurements demonstrated the effectiveness of such patterns, even outperforming edge clusters.

CCS Concepts: • **Computer systems organization → Embedded systems**; *Redundancy*; • **Networks →** Network reliability;

Additional Key Words and Phrases: Fog, edge, cloud computing, offloading, Stack4Things, smart environment, IoT, big data

ACM Reference format:
Giovanni Merlino, Rustem Dautov, Salvatore Distefano, and Dario Bruneo. 2019. Enabling Workload Engineering in Edge, Fog, and Cloud Computing through OpenStack-based Middleware. *ACM Trans. Internet Technol.* 19, 2, Article 28 (April 2019), 22 pages.
https://doi.org/10.1145/3309705

Authors' addresses: G. Merlino and D. Bruneo, University of Messina, Contrada di Dio n. 1, 98166, Messina, Italy; emails: {gmerlino, dbruneo}@unime.it; R. Dautov, Kazan Federal University, Kazan, Russia; email: rdautov@it.kfu.ru; S. Distefano, University of Messina, Contrada di Dio n. 1, 98166, Messina, Italy and Kazan Federal University, Kremlevskaya 35, 420008, Kazan, Russia; emails: sdistefano@unime.it, s_distefano@it.kfu.ru.

1533-5399/2019/04-ART28 $15.00
https://doi.org/10.1145/3309705

1 INTRODUCTION

Despite the recent ICT advances in Cloud, Fog, and Edge computing, as well as in the Internet of Things (IoT) and Big Data, there is still an untapped potential to be unleashed through innovative convergences and previously unknown correlations of these technologies. This becomes particularly evident in some innovative ICT domains, such as *Smart Cities*, *e-health*, *Industry 4.0*, and *e-government*, to name a few. In such smart environments, a common cyber-physical pattern is emerging based on a synergistic fusion of several existing solutions. This pattern usually comprises a pervasive network of smart objects and things, serving to monitor and control physical systems (i.e., IoT), collect, store, and process related data through computing infrastructures and platforms (i.e., Edge, Fog and Cloud computing), adopting a wide range of software frameworks, solutions, and algorithms for analytics (i.e., Big Data). So far, however, such smart environment systems have been implemented following a "vertical," stand-alone approach, based on *ad-hoc* infrastructures, platforms, and frameworks, specifically conceived for a problem at hand. This siloed and unscalable pattern typically relies on proprietary infrastructure with dedicated hardware and software, which will render unsustainable if, as expected, demand for smart services will increase in the near future. This identifies smart environments as fragmented *archipelagos* of isolated, local *islands* of sensors and actuators, (static and mobile) devices (cameras, environmental stations, personal portable devices, vehicles, etc.), network facilities (WiFi, 3G/4G, fiber optic, routers, base stations, cells, etc.), and basic mechanisms for data management (collection, storage, aggregation, fusion, processing, etc.).

Accordingly, the next evolutionary step to be taken is to efficiently combine these existing approaches and technologies into an all-encompassing solution able to boost up new social, technological, and commercial avenues for pervasive and ubiquitous smart environment applications, thus establishing novel markets and economies. The convergence among the IoT, Edge, Fog, and Cloud technologies will not only simplify and speed up this process but will make it more scalable and flexible–i.e., underlying IoT infrastructures are supposed to be shared and reused by multiple users/applications, thus significantly reducing up-front investments and time-to-market for new services. By embracing heterogeneous technologies and services, the envisioned overarching approach will be able to address smart environment challenges and support related software design, development, and deployment activities.

However, the Edge/Fog paradigm is still not seen as an alternative to the Cloud but rather as a complementary addition, able to support distributed (remote) processing of smart environment applications. Thus, the cross-cutting convergence of the Edge, the Fog, and the Cloud to address time, performance, and security requirements is yet to be explored. To fill this gap, this article proposes a novel OpenStack-based middleware platform that embraces the Edge and Fog paradigms for (Big) data processing, supported by hierarchical Cloud-driven coordination. This approach relies on equipping co-located smart Edge and Fog devices with middleware agents to allow Cloud-mediated discovery. Upon discovery, Edge devices are then able to establish local-area clusters to share computational workload (i.e., horizontal offloading) to minimize the amount of data transfers over public networks and thus achieve better performance, as well as to avoid unnecessary security risks associated with public networks. If needed, Cloud/Fog nodes can also be involved (i.e., vertical offloading), thus combining both offloading patterns to meet the application requirements, paving the way for a discipline of *(offloading) workload engineering*. As demonstrated by experimental results, such offloading patterns based on clustering techniques at the very network edge have the potential to outperform the currently established and dominant Cloud-centric/vertical offloading data processing, especially in the context of time-critical application scenarios. Thus, the main contributions of the article are the following:

- ***horizontal offloading*** is a novel pattern for computational task offloading to Edge/Fog nodes through clustering of collocated devices;
- a degree of freedom in combining horizontal and vertical offloading to Edge, Fog, and Cloud nodes, enabling orchestrated hierarchical clustering, is endowed, providing ***new avenues for workload engineering***;
- ***middleware platform*** provides an OpenStack-based framework implementing basic mechanisms for horizontal and vertical offloading and their orchestration through workload engineering.

The rest of the article is organized as follows. Section 2 introduces preliminary concepts, outlining the research context, and positioning the proposed research with respect to the existing works. Section 3 provides an initial high-level description of the proposed solution based on several functional requirements. Section 4 further elaborates on the architecture of the proposed middleware by explaining its key components and their relations. Section 5 sums up the theoretical materials and details how the proposed system addresses the requirements. Section 6 puts theory into practice and demonstrates how the proposed approach can be applied to a smart vision system, evaluating the experimental results. Section 7 summarizes the article with some concluding remarks and an outlook for future work.

2 PRELIMINARY CONCEPTS

2.1 Big Picture and Vision

Cloud Computing and the IoT are nowadays seen among the main drivers of the new ICT wave, giving rise to smart environments, cities, and communities. As a result, avalanches of data are being generated, raising the challenge of minimizing network latency and increasing the computational speed (Gubbi et al. 2013). To this purpose, the concept of Fog Computing emerged as a way of bringing processing capabilities closer to the data source by enabling processing on networking and communication units, such as routers, switches, IoT gateways, "cloudlets," and so on. A further step toward keeping computation even closer was made by the Edge Computing paradigm that enabled end devices to execute relatively simple data aggregation and filtering as part of a more complex data processing workflow, the rest of which was expected to be accomplished by the Fog and Cloud through vertical offloading. As suggested by the existing literature (Botta et al. 2016; Copie et al. 2013; Pflanzner and Kertész 2016), a promising way of implementing data processing in smart environments is to converge Cloud, Fog, and Edge Computing capabilities into an all-encompassing framework in an efficient and resource-optimized manner. With this research effort, we aim to advance the existing technological baseline, mostly driven by the elastic resource provisioning of the Cloud. At this baseline, the IoT still primarily acts as a passive data generator, only capable of pushing collected data to the Cloud for processing. However, as further explained below, a truly balanced and efficient convergence of the Edge, Fog, and Cloud paradigms is possible with a goal to enable faster execution and data management. Given that IoT devices are equipped with sufficient networking and processing capabilities, it is valid to assume that they can build up distributed clusters to share computational tasks, introducing a novel form of processing named ***Clustered Edge Computing (CEC)***. As computation is pushed closer to the extreme edge from the Cloud and network-level processing units, latency and processing time decrease. Arguably, since a minimum amount of sensitive data is transferred over the public network, associated security threats are also reduced. Figure 1 depicts our comprehensive view of this three-level processing architecture:

- ***Edge Computing/CEC*** assumes that data processing takes place directly on Edge nodes, thus reducing network latency and overheads. CEC extends the existing scope of Edge Computing enabling (peer-to-peer) cooperation among Edge nodes through clustering.

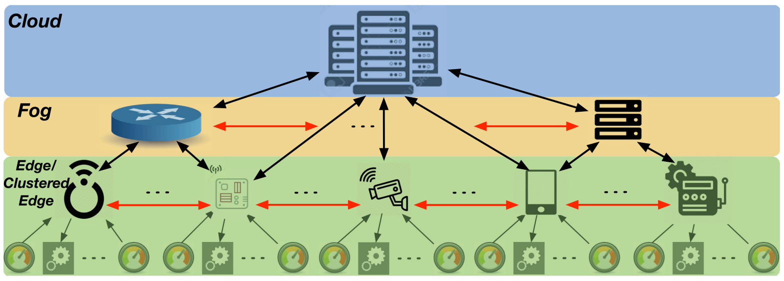


Fig. 1. Three-level (Cloud, Fog, Edge/Clustered Edge Computing) processing in smart environments.

— ***Fog Computing*** on the network/communication processing units in close proximity to data sources serves to provide additional support to Edge devices. Similarly to Edge Computing, Fog nodes could also be involved in *Fog Clustering*.
— ***Cloud Computing*** may provide even more resources to lower-level devices when pooled resources are not sufficient.

A fundamental underpinning of this view is the notion of *computational workload offloading*. Broadly speaking, offloading is a process of transferring certain computing tasks to an external platform, such as a cluster, a grid, or a Cloud (Kumar and Lu 2010). In this three-level architectural view of smart environment computing, we identify two main patterns of computation offloading, extending its original definition as follows.

— ***Vertical offloading*** (black arrows in Figure 1) is the traditional and established way of sending computing workload over the network to a nearest server/datacenter. Sensor-enabled Edge devices are treated and managed as resource-constrained devices, only capable of collecting and pre-processing (e.g., filtering, aggregating) raw data that are then transferred for remote processing and storage to communication and processing units, such as routers, servers, or "cloudlets" (i.e., Fog computing) or to higher-level datacenters (i.e., Cloud computing). In this case, no computation takes place on Edge devices, but rather all the raw datasets are transferred from the bottom of the topology "upward" (i.e., vertically offloaded). This way, IoT systems can benefit from (seemingly infinite) computing and storage capabilities of a remote cloud platform at the cost of increased network latency and reaction time.
— ***Horizontal offloading*** (red arrows in Figure 1) is a novel type of offloading, enabled by CEC and seen as one of its most significant contributions, relies on the principles of cluster computing, where multiple computational nodes (i.e., a distributed cluster) share the workload among themselves to increase the performance. Horizontal offloading lies at the core of the proposed approach, thereby allowing it to minimize the amount of data sent over the network to the Fog/Cloud and time delays due to network latency. There are three main factors underpinning the horizontal offloading pattern to be clarified in this respect:

(1) Edge (and especially mobile) devices are getting increasingly powerful in terms of their hardware specifications (i.e., CPU and RAM). They have advanced beyond the simplistic notion of collecting and transferring raw data, and nowadays act as fully functional processing units in their own right, widely used as effective computing systems.
(2) The world is experiencing a continuously growing number and usage of embedded and mobile devices in all aspects of people's daily activities. The IoT world is flooded with

all kinds of "smart" things, which can be seen as potential contributors to the shared pool of clustered resources.

(3) Despite the growing computational capabilities of edge devices, these are typically not yet equipped with full-featured storage facilities, thus mainly suitable for in-memory data processing.

A policy-driven convergence among horizontal and vertical offloading patterns opens promising opportunities for optimizing QoS and SLA requirements based on task complexity through balanced workload distribution across participating devices at the Edge and Fog levels. This way, multiple degrees of freedom are offered to architects for customizing the smart environment infrastructure according to current requirements. In these circumstances, orchestration becomes crucial, as it concerns how distributed computation is allocated, balanced across nodes, and managed at different levels. This calls for novel solutions to support system architects in *workload engineering* with mechanisms to define, design, deploy, mash up, orchestrate, monitor, and evaluate offloading policies and strategies.

2.2 Related Works

Both academic researchers and industrial practitioners have been putting their efforts into developing efficient and scalable approaches to enable transparent elastic provisioning of Edge-Fog-Cloud computational resources to end users, aiming to comply with established SLA and guaranteed QoS, taking into consideration such factors as energy consumption, network latency, and geographic distribution (Manvi and Shyam 2014; Vinothina et al. 2012). Admittedly, Cloud Computing has established itself as the *de-facto* standard for provisioning remote ICT resources to customers through the Internet following the vertical offloading approach. Fog Computing, albeit a less mature domain, is experiencing a rapid growth of interest and demanding for similar container-based resource allocation and orchestration mechanisms. As opposed to the Cloud, the main challenge here is not in energy efficiency and network latency but rather in a relatively limited hardware capacity of Fog nodes (Pahl and Lee 2015). In this light, a balanced resource allocation across the two layers—namely, Fog and Cloud—has been widely explored. As a result, existing works aim to enable a resource allocation and orchestration architecture, which would transparently provision containerized resources, finding a right balance between low network latency of the Fog and increased computational capacities of the Cloud (Masip-Bruin et al. 2016; Skarlat et al. 2016; Vilalta et al. 2016; Wen et al. 2017).

More recently, there is preliminary work (Dautov et al. 2017a; Manzalini and Crespi 2016; Markakis et al. 2017; Pahl et al. 2016) demonstrating how Edge devices can be used for processing, also clustered and managed through common middleware, thereby achieving even lower latency for relatively simple computations. Similarly to the Cloud and Fog orchestration, these approaches rely on equipping Edge nodes with agent-like virtual containers to enable orchestration and management. This way, communicating Edge devices can split and delegate computational tasks.

As a natural development to this background, workload engineering for Cloud-Fog-Edge environments is then covered in some recent works. In Wang et al. (2018), the authors survey offloading algorithms for workloads within an Edge-Cloud space according to the type of application partitioning schemes, static/dynamic architecture, granularity, mobility, and destination. As far as containerization is concerned, a prominent solution is *microservices* (Balalaie et al. 2016)—an architectural style for application development, where a software system is designed as a collection of loosely coupled services, each implementing some specific business logic. An example of an architecture for microservice offloading at the Edge, and specifically in vehicular networks, is presented in Taherizadeh et al. (2018), where the workload unit is the container, but the solution is not based on any infrastructure-oriented platform. As such, many facilities, such as resource monitoring, are custom designed. A lightweight Cloud platform for mobile (i.e., Android) offloading is presented in Wu et al. (2017), where the mobile OS is emulated on the Cloud side to support mobile-compatible containers. This way, same workload can be hosted on either side (Cloud or mobile), according to requirements. Albeit an interesting development for workload engineering with regard to mobiles, the work tackles neither the architecture of the infrastructure orchestration system nor any offloading workflows.

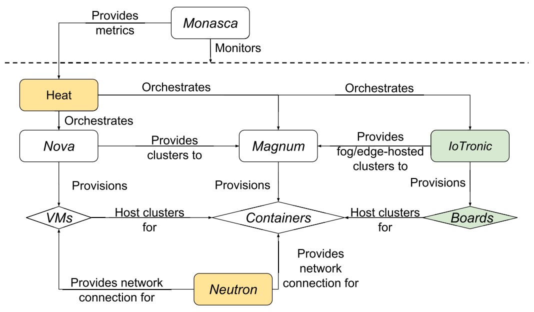
With a sufficient amount of works focusing on vertical offloading to the Fog and the Cloud and associated resource management, as well as first steps to enable collaborative computation among Edge devices by means of horizontal offloading, there is, however, a lack of evidence of convergence between the three paradigms. More specifically, to the best of our knowledge, there has been no intelligent and efficient approach proposed in the literature that would take into account computational resources at all three levels to fully utilize both local resources of collocated Edge devices and remote capacities of Fog and Cloud infrastructures. Driven by intelligent policies, such a solution would enable optimised resource provisioning and orchestration across all three levels through horizontal-vertical offloading mash-up and workload engineering, thereby achieving better performance with lower latency. Accordingly, the research effort presented in this article makes a first step toward addressing this gap, also taking as a reference existing tools and middleware for IoT–Cloud integration, overviewed in Pflanzner and Kertész (2016), Botta et al. (2016), and Bruneo et al. (2018). All these works are somewhat similar on their goals and scopes to this article and are, therefore, also taken as a starting point for further exploring the domain of workload engineering in Edge–Fog environments.

3 SOLUTION

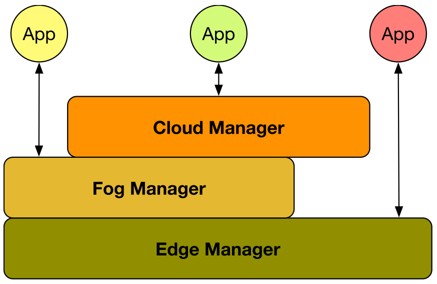
3.1 Assumptions and Requirements

Data management is a broad and complex research area covering such aspects as data generation, collection, fusion, processing, archival, and curation. Admittedly, discussing all these issues, albeit important in the context of data-intensive IoT scenarios, goes beyond the scope of this article, which primarily focuses on the processing aspects of the data management lifecycle. Therefore, it is assumed that corresponding tools and mechanisms preceding and following the data processing step are already in place and are not further discussed here. As far as data processing itself is concerned, there are a number of requirements that can be quite broadly defined and are not specific to the workload engineering domain, unless otherwise noted. First, available computing, storage, and I/O (i.e., sensors and actuators) resources have to be suitably managed at different layers. If the management of Cloud resources can be easily done with off-the-shelf solutions (i.e., Cloud platforms), then resources belonging to the Fog and Edge layers call for *ad-hoc* mechanisms able to deal with their specific nature. In fact, such resources are not part of a single datacenter and cannot guarantee stable connections and reliable QoS. Specifically, the following aspects require particular attention:

- ***Resource Discovery***: Fog and Edge resources can unpredictably join/leave the system, calling for mechanisms able to dynamically discover and select available resources according to current requirements (e.g., type of resources, geographical area, availability, etc.)—an essential requirement for workload engineering.
- ***Placement and Orchestration***: In a Cloud-Fog-Edge environment, nodes are highly heterogeneous, making allocation of tasks to available resources challenging. Next, upon task placement, it is also important to monitor of both application and infrastructure layers, as well as to evaluate offloading policies and react to potential issues, such as the node churn.



(a) Conceptual architecture for a Fog/Edge-enhanced OpenStack middleware.



(b) Cloud/Fog/Edge framework high-level architecture.

Fig. 2. Proposed solution architectures.

- ***Accounting, Authorization, and Authentication (AAA)***: Corresponding mechanisms should be implemented to manage resources offered by different providers and enable secure and trustworthy resource delegation and exploitation.
- ***(Overlay and Virtual) Networking***: A key concept in Cloud-based application development is the capability of establishing virtual networks that allow application components to interact one with another. Creation and management of Cloud-mediated or fully peer-to-peer virtual networks among remote resources is a strong requirement of the proposed Fog–Edge vision. In fact, this feature will allow the deployment of standard (i.e., not specifically ported) applications on top of Fog–Edge devices, as if they are on the same LAN without caring about NAT/firewall issues.
- ***Programmability, Customizability, and Isolation***: Pushing computation to the Fog–Edge layers means decentralizing business logic and scheduling its execution on remote resources, whenever conceivable. To this end, suitable mechanisms have to be designed to manage software execution and reconfiguration over a pool of distributed resources. The heterogeneous nature of technologies and resources, their limited reliability, harder reachability, constrained capabilities and fragmentation, as well as the diversity in software systems result in a fragmented landscape from the developer's perspective and leads to a lack of standardized programmability and multi-device integration facilities. A possible solution is to treat applications as if they are composed of several micro-services that can run in isolation (e.g., through the containerization technology) in distributed hypervisors, thus allowing on-the-fly mechanisms for suspending/migrating/resuming application components in a seamless, non-intrusive way.

3.2 Middleware

Based on the identified requirements, we propose infrastructure-oriented middleware allowing to provide a full-fledged suite of tools for Cloud-Fog-Edge orchestration, co- and inter-operation, whilst leveraging the progress and pace of the relevant developers and researchers communities. OpenStack[1] is an Infrastructure-as-a-Service (IaaS) framework, a widely recognized open source effort with several contributing companies and communities, making it a natural choice in terms of widespread support and potential for enhancement and collaborations. The main OpenStack components, relevant in the context of this research, are the following (also depicted in Figure 2(a))[2]:

[1]https://www.openstack.org/.
[2]Other notable subsystems are Keystone, the authentication and authorization framework, and Horizon, the Web-based UI (dashboard).

— ***Nova*** provisions compute instances (e.g., virtual servers), at its core creating virtual machines (VMs);
— ***Magnum*** is functionally similar to Nova but rather oriented toward the provisioning of containers, instead of VMs;
— ***Neutron*** provides networking facilities to compute instances;
— ***Heat*** orchestrates virtual resource instantiation, manages the lifecycle of resource ensembles, and auto-scales allocated resources to match dynamic application requirements;
— ***Monasca*** collects infrastructure usage and performance metrics at scale.

A first attempt to move the OpenStack ecosystem toward the IoT world has been made through ***Stack4Things*** (S4T) (Longo et al. 2016), trying to implement the basic mechanisms for IoT nodes to join Edge-first IaaS Clouds. It can be seen as an IoT operating system for deployment of smart applications, adopting a software-defined approach. S4T provides infrastructure-enabling facilities for the management and pooling of Edge nodes in clusters, including smart, networked objects, and any kind of (programmable) nodes belonging to smart environments, providing support for:

— ***Tenants and projects***, by offering mechanisms (Bruneo et al. 2016b) to delegate and revoke authorizations for users and groups to access and manage selected resources with tunable granularity.
— ***Customizability***, by enabling on-demand reconfiguration of smart objects at runtime, from low-level firmware/OS configuration (i.e., also interacting with the initialization and package management subsystem) up to the business logic through mechanisms for injection of pluggable software components from the Cloud.
— ***Virtual networking***, by instantiating and managing (Cloud-initiated) virtual networks among smart cameras, thus deploying standard applications on IoT devices as if they were on the same LAN. This way, network barriers and issues can be overcome through (reverse) tunneling, while setting up the functional equivalent of private, isolated, secure VPN environments with the flexibility of low-level, composable tools for smart environment nodes communication.

In terms of components, S4T comprises ***IoTronic***, which is the core OpenStack subsystem for the IoT, and ***Lighthing-Rod***—the board-side S4T agent. The IoTronic service enables end users to manage smart boards remotely, whereas Lightning-Rod is the point of contact with the Cloud infrastructure, through which board resources can be managed even when they are behind a NAT or a strict firewall. The latter is ensured by Secure WebSocket– (WSS) powered messaging, as well as WSS-enabled tunneling, between the agent and its Cloud counterpart.

As a next step from this existing baseline, we are extending the S4T framework to enable the aforementioned Edge/Fog infrastructure-enabled concepts and offloading patterns. To this extent, as discussed above, Figure 2(a) presents key subsystems and resource categories in OpenStack, highlighting the existing S4T elements in green and the newly introduced ones in yellow. Indeed, as depicted by the arrows, IoTronic provides Fog–Edge nodes to Magnum as "clusters," which are classes of instances (e.g., VMs when provided by Nova) able to host containers. Similarly to Nova, IoTronic also provisions Fog–Edge machines (here referred to as "boards"), providing native support and management for boards. Differently from Nova, resource provisioning takes place on the bare metal, akin to the Ironic subsystem.[3]

Next, we also adopt the recently introduced OpenStack Nova concept of *cell* and tailor it to the context of S4T. A *cell* is a (relatively) autonomous domain comprising a subset of the pool of

[3]https://wiki.openstack.org/wiki/Ironic.

managed resources within a larger OpenStack deployment. The purpose of the cell partitioning is to allow large deployments to shard their controlled nodes into smaller (and more easily manageable) domains, which may let the system scale even when dealing with massive amounts of nodes. This way, cells are especially useful in the Edge computing context, as well as when creating a topological/hierarchical domain (e.g., a Fog node is a root of a tree, whereas its leaves are Edge nodes using the Fog node as a gateway) that also becomes a failure domain, able to survive extended downtimes involving either other domains or datacenter-hosted (centralized) services.

In terms of modifications to pre-existing OpenStack subsystems, Heat needs to be adapted to include support for IoTronic, and the additional resource types it deals with. In this sense, the Heat Orchestration Template (HOT) specification has to be amended to include these resources, and to encode Fog–Edge-exclusive patterns in its directives for scaling, to deal with vertical/horizontal/mixed offloading also allowing to take into account eventual application mobility. By contrast, it is important to remark that, albeit Magnum could be enhanced as well to support IoTronic natively, in principle it may be kept agnostic as long as IoTronic provides another endpoint for a Nova-compatible subset of APIs, those relevant for cluster initialization. Same considerations apply to Neutron, as its native adaptation to Fog–Edge workflows is possible: Indeed, IoTronic already features a custom-developed (Merlino et al. 2015) solution for tunnel-based overlay networking and a preliminary integration with Neutron (Benomar et al. 2018).

4 ARCHITECTURE

Given the requirements and specifications identified in Section 3, a middleware for IaaS is just part of the answer: In particular, a *distributed* framework architecture and deployment is needed to manage the underlying system complexity. This framework spans across the Cloud/Fog–Edge levels and therefore can be viewed as a three-layer architecture, as depicted in Figure 2(b), comprising the Cloud Manager, the Fog Manager, and the Edge Manager.

— ***Cloud Manager*** is mainly composed of a Cloud platform that allows to control datacenter resources (i.e., computing, storage, networking). It provides typical Cloud services, while enabling a new dimension by introducing the possibility to include resources provided by the Fog and Edge levels. Some subsystems, such as Heat (orchestration), Magnum (container provisioning), and Monasca (monitoring), to name a few, fully reside within the realm of the Cloud Manager.

— ***Fog Manager*** acts as an intermediary between Cloud and Edge resources. It manages the pool of underlying Edge resources, located on the same network, as well as its own resources. It hosts a subset of typical Cloud facilities (e.g., orchestration, networking, and churn management) to take autonomous decisions without reverting to the Cloud.

— ***Edge Manager*** acts as an Edge resource controller and is able to run application components, also offering its sensing and actuating facilities. It interacts with higher-level counterparts for vertical offloading at the Fog and/or Cloud levels.

This way, an application (as a whole or partially) can run at different layers exploiting vertical (Cloud and Fog), horizontal (CEC), and combined offloading patterns, as discussed in Section 6. In the following, we present the IoTronic-based design of the Cloud, Fog, and Edge Managers according to the topological layering in Figure 2(b).

4.1 Cloud Manager

The design of the Cloud-side IoTronic subsystem, belonging to the Cloud Manager, is presented in Figure 3, where the architecture and (centralized) deployment of *Heat*, *Magnum*, and *Monasca* is unmodified for the sake of the Fog–Edge adaptations to S4T. We will discuss their roles in more

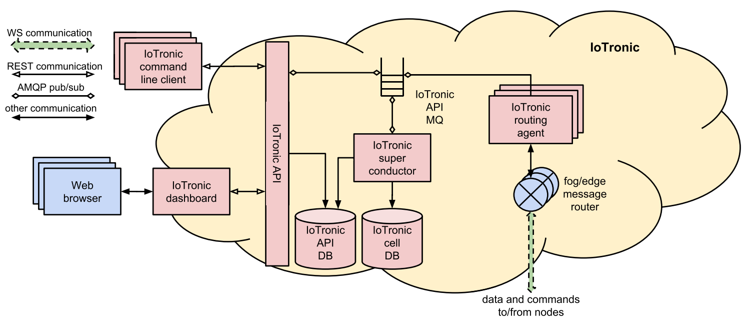


Fig. 3. IoTronic cloud-side architecture.

detail when describing the workflows in the following section. In general, the IoTronic service features standard architectural traits of an OpenStack service, thus making it compliant with best practices and agreed guidelines among OpenStack stakeholders. More specifically, the *IoTronic super-conductor* represents the core of the service, managing Fog–Edge resources, storing nodes' metadata, and dispatching remote procedure calls among other components. What makes the conductor "*super*" is the established hierarchy for interacting with a number of (plain) conductors to support a specific clustering configuration (i.e., a *cell*-based configuration in the OpenStack parlance, as anticipated in Section 3.2). Each cell has a dedicated database and a message queue.

The super-conductor stores the required information either globally in the *IoTronic API DB* (e.g., for board-unique identifiers, board properties, and hardware/software characteristics) or in multiple local *IoTronic cell DBs* (e.g., for temporary allocations of board resources to users and tenants). The *IoTronic API* service exposes a REST interface to end users that may interact with the service via the *IoTronic command line client* or a Web browser. Furthermore, the OpenStack Horizon dashboard was extended with an *IoTronic panel* to expose the functionality provided by IoTronic.

The *IoTronic routing agent* controls the Fog–Edge message router, which "speaks" the Web Application Messaging Protocol (WAMP), a WebSocket sub-protocol, and acts as a bridge between other components and boards. It translates Advanced Message Queuing Protocol (AMQP) messages into WAMP messages and vice versa. Following the standard OpenStack philosophy, all the communication among IoTronic components is performed over the network (regardless of the transport, as explained in Section 5) via an AMQP queue. This allows the whole architecture to be as scalable as possible, given the facts that (i) all the components can be deployed on different machines without affecting the service functionalities and (ii) more than one *IoTronic routing agent* can be instantiated to deal with a subset of the IoT devices. This way, redundancy and high availability are also guaranteed. A prominent reason for choosing WAMP as the protocol for node-related interactions, apart from possibly leaner implementations and smoother porting, lies in WAMP being a WebSocket sub-protocol, supporting two application messaging patterns, *Publish & Subscribe* and *Remote Procedure Calls* (RPC), whereas the latter is not natively supported by AMQP.

4.2 Fog Manager

The design of the Fog Manager, the head of an IoTronic Cell, is outlined in Figure 4. The *IoTronic conductor* is the core component of a Cell head. It stores instance information in the local *cell DB*, interacts with Cloud and Edge Managers via a cell-local message queue, mapped through (coupled) message routers on each side, to a remote queue either on the (Cloud/Edge) side, sending metrics, or other application-specific data (e.g., sensor values), exchanging messages to/from the Cloud/Edge,

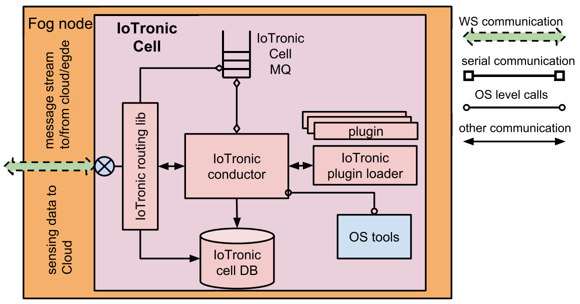


Fig. 4. IoTronic fog-side architecture.

and executing instructions out of the command stream. Signalling with the Cloud and the Edge is facilitated by (*IoTronic routing libraries*) implementing the client-side functionality of the message routing protocol.

The Fog Manager also implements a *plugin* loader. In the IoTronic model, plugins are the smallest self-contained units of execution (either in Node.js or Python), carrying the business logic as a payload. Once uploaded to the Cloud, the incoming payload is validated only by checking its syntax with a linter. Custom plugins can be injected from the Cloud through a dedicated RPC, exposed just for enabling the transfer of plugin code in its payload, and run on top of the plugin loader to implement specific user-defined commands. The lifecycle of a plugin, once loaded on the board, is fully managed by the loader itself, through a generic wrapper. New REST resources are automatically created exposing any (plugin-provided) user-defined commands on the Cloud side. Once such resources are invoked, the corresponding code is run on the board. In this respect, we can distinguish two kinds of plugins:

— ***asynchronous plugin***: once invoked (i.e., started), typically long running (e.g., a monitoring job) until the user stops its execution, and its status (whether still running or stopped) can be queried;

— ***synchronous plugin***: called on demand to execute its code (e.g., a system call) and return a result, in pure Function-as-a-Service (Hendrickson et al. 2016) (FaaS) fashion. Indeed, this plugin *model* for user-defined behavior in IoTronic has been devised independently from the FaaS paradigm and, in its synchronous flavour implementation, already provides equivalent functionality by exposing through HTTP requests the corresponding (WAMP-enabled) RPCs.

In fact, it is actually this plugin-oriented approach, featuring its own wrappers and full lifecycle management, originally developed for other scenarios, which has been leveraged as a hosting platform for business logic, in place of proof-of-concept integration design of container instantiation by Magnum. Aligning with the roadmap of the OpenStack community, we expect to refactor the IoTronic plugin facilities to integrate them with the official OpenStack subsystem for FaaS, Qinling.[4] Specifically, a plugin wrapper instance exposes a subset of the APIs to Magnum (belonging to the Cloud Manager) that is able to invoke the container manager (i.e., the wrapper mimics the container hypervisor interface). The plugin code is expected to have enough privileges to invoke the container manager to actually bring up container instances and report back status/progress to the wrapper, in turn relaying this information to Magnum.

[4]https://docs.openstack.org/qinling/latest/index.html.

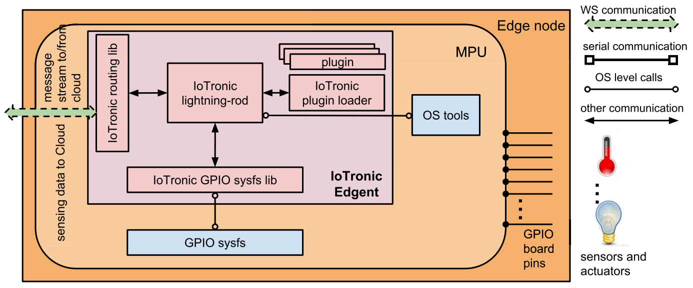


Fig. 5. IoTronic edge-side architecture.

4.3 Edge Manager

The design of the Edge Manager, *IoTronic Edgent*, which is somewhat similar to the Fog Manager, is introduced in Figure 5. Similarly to the conductor in Fog nodes, the *IoTronic Lightning-Rod* plays a central role in the Edgent, with a difference that it is meant to be (relatively speaking) stateless, thus not featuring any cell-specific DB, and just serializing limited state information (e.g., loaded plugins) in a local JSON settings file. Indeed, the Edgent has been field tested in Smart City (Bruneo et al. 2016a) deployments, featuring Arduino YÚNs and Raspberry Pis, and been found very lightweight with a small footprint. In particular, the YÚN is a single-board computer on the (extreme) network edge edge, considering that its specifications are 64MB of RAM only, and a MIPS as a microprocessor unit, yet it is enough to host a minimal Linux environment that features a Python runtime hosting the Edgent. Beyond that, such a board has still room to let the Edgent spin up a few plugins, each of which operating on transducers (sampling sensors, triggering actuators) and, in some cases, even pre-processing samples on-board before sending data to the Cloud. Communication with a cell is ensured by the same set of client-side message routing facilities (*IoTronic routing libraries*) as in the Fog Manager. Ultimately, access to the board digital/analog I/O pins, and thus with the connected sensing and actuation resources, is achieved, even remotely, still resorting to the same message queuing/routing facilities.

5 IMPLEMENTATION, SETUP, AND OPERATION

5.1 Implementation and Deployment

IoTronic[5] has recently been integrated with other core subsystems, such as Neutron (Benomar et al. 2018), and it is in the process to fully interoperate with the container-oriented subsystems in the OpenStack ecosystem. Moreover, the IoTronic project has recently been recognized by the OpenStack community at large and more specifically promoted to the shortlist[6] of projects the OpenStack *Edge Computing Group* is agreeing upon to have OpenStack ecosystem fully support Edge computing use cases in the near future. As such, IoTronic, albeit currently an unofficial project, is expected to be gradually incubated in OpenStack. The Cell-related modifications are at the design and early proof-of-concept implementation stages and as such unreleased yet.

Figure 6 depicts a hierarchy featuring the (datacenter-hosted) centralized Cloud services at the top, and Fog–Edge nodes below, respectively one (in the middle) for the former category and two (at the bottom) for the latter one. As anticipated, the Cloud Manager components, including other OpenStack services (Keystone, Neutron, Heat) essential for a minimal deployment, are hosted in the datacenter, whereas the Fog Manager is deployed in the Fog node and the two Edge nodes

[5]http://git.openstack.org/cgit/openstack/iotronic.
[6]https://wiki.openstack.org/wiki/Edge_Computing_Group#Related_OSF_Projects.

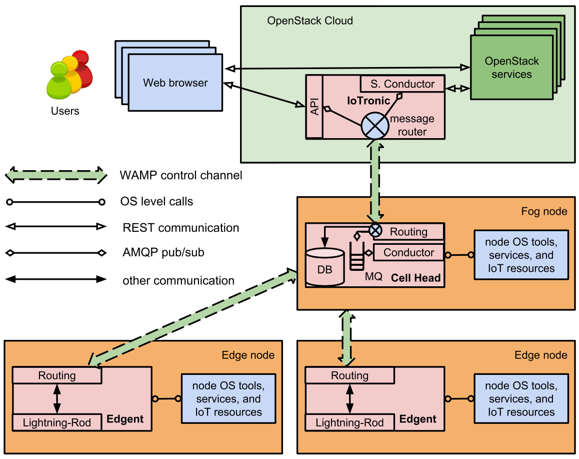


Fig. 6. IoTronic deployment.

host the Edgents. On Fog nodes, apart from IoTronic-provided components, only select Neutron agents may be required for deployment, according to specific requirements (Benomar et al. 2018), whereas Edge nodes are not expected to host any Neutron component.

The depiction of the Managers at all levels reflects the structure we have already seen in Figures 3, 4, and 5, albeit simplified in the diagram, to better focus on the big picture. Noteworthy, the message routers act as a transport conduit for Cloud-native intra-service traffic—i.e., AMQP messages between any of the relevant Cloud Manager entities (API, Super Conductor) and the Fog-side counterpart (Conductor) get encoded as WAMP messages, as it also happens for SQL commands, when the Fog-side counterpart interacts with the (Cloud) API, and the Super Conductor is the DB. According to the topology in Figure 6, one that can be identified by the set comprising the Fog node (*cell head*) and the two Edge nodes (*edgents*), horizontal offloading (e.g., from the edge node on the left to the one on the right) and vertical offloading (e.g., from one of the Edge nodes to the Fog one) inside a single cell can then be enabled by a combination of the following facilities:

— ***container snapshotting/upload***—to support live migration in the presence of certain QoS-constrained workflows;
— ***orchestration hooks***—i.e., triggers for the orchestrator to initiate the offloading process;
— ***overlay networks***—to support cooperation of tasks in the workload (regardless of the underlying network infrastructure).

Even across (multi-)cells (not depicted in the figures) similar considerations apply, albeit additionally requiring cooperation among conductors, with the supervision of the super-conductor for overlay networks across multiple cells.

5.2 Workflow

Execution of complex, computationally intensive applications goes beyond capabilities of a single edge device, which requires this edge device to partially offload computation to peer nodes. Accordingly, as a pre-requisite to initiate an offloading procedure, it is first required to decompose the complex application into several simpler tasks that fit functional and non-functional properties of edge devices. In this light, task decomposition—i.e., identifying individual steps, as part of a more complex application workflow and their interconnections—becomes an important challenge, especially in the light of resource-constrained and heterogeneous nature of IoT/Edge environments. This is, however, a research topic going beyond the scope of this article, where task partitioning is assumed to be performed manually by a system administrator, familiar with the nature and the internal organization of the application. Accordingly, it is assumed that edge devices are already manually equipped with required knowledge of decomposed tasks, as well as policies defining when to initiate offloading.

In the proposed offloading scenarios, five phases can be identified to enable CEC and successfully complete offloading processes. These stages are discussed in the following and shown in Figure 8.

— ***Discovery***: To tap into the idle potential of ubiquitous Fog and Edge devices, it is important to discover them first, thus facilitating their integration into an (overlay network-enabled) cluster. Node discovery provides useful details on Fog–Edge nodes (e.g., role in the topological hierarchy, geographical location, network configuration, type of power supply, operating system, available software, etc.) for the next phase (node selection).
— ***Selection***: Albeit discovered nodes might be functionally suitable for processing, problems may emerge during aggregation or clustering due to, for example, networking or performance/dependability/security issues. Network barriers can partially be overcome by the availability of firewall-agnostic tunneling mechanisms for instantiating overlays, as is the case in S4T. With regard to the latter, performance can be impaired by runtime changes, such as disconnecting Edge nodes (e.g., due to unstable connectivity or power supply). In this case, the "setting capacity" functionality built into Heat can be exploited as is by correspondingly defining the capacity, i.e., the minimum number of required resources/nodes, in one or more HOT templates. The special case is thus tackled by modeling clustering itself as an "application" of its own. More specifically, this means aligning software requirements with the number of cluster nodes and their computational resources. This is performed according to specific policies and strategies, which may be expressed through filters, as Nova does with its default ("filter") scheduler, e.g., by supporting filtering and weighting to decide where a new instance should land. This may also result (at a later phase) in a hierarchical clustering, where orchestration and coordination is somehow delegated to lower levels. It is important to note that the cluster could hierarchically span across cells as well as topological layers (Edge, Fog, Cloud).
— ***Placement***: Once all the nodes and their capabilities are identified, it is time to actually assign proper roles and duties by matching tasks to resource instances (e.g., containers) and allocating the latter. This is particularly challenging in the hierarchical clustering case, where tasks and duties differ and should be properly orchestrated. In fact, if in small scenarios simple *if-then* rules can be applied in a best effort fashion, complex environments require the use of specific heuristics or AI-based approaches. In particular, due to the high dynamics of the system, as a future work we plan to adopt reinforcement learning techniques to find the optimal policy with respect to the actual system conditions.
— ***Configuration***: After placement, it is time for the orchestration subsystem to actually setup the cluster and for any task deployment subsystem (e.g., a container provisioning subsystem) to deploy tasks in the instances (e.g., containers), hosted by the resulting cluster, and request the Fog–Edge subsystem overlays among instances.
— ***Orchestration/Lifecycle Management***: Once configured, the resource instances (e.g., containers) on the selected cluster nodes are ready to proceed with the processing step. The tasks then run in parallel on the hosting instances.

From a behavioral perspective, the dynamics of such Fog–Edge clustering/offloading is shown in Figure 7. It highlights the workflows involved, i.e., the actual clustering process (Discovery, Selection) and the (horizontal/vertical/mixed) offloading one (Placement, Configuration), respectively. In the described scenario, the activity is commenced by an *initiator* (e.g., any Fog–Edge node agent,

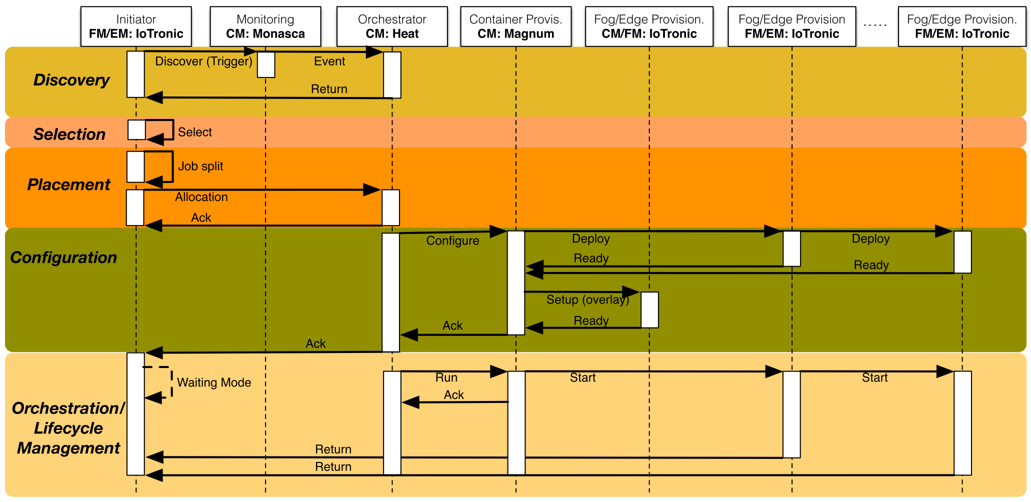


Fig. 7. Offloading workflow.

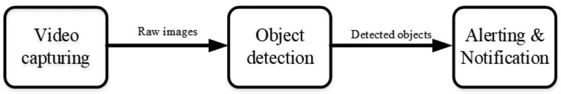


Fig. 8. An ISS process using neural networks for object detection.

configured to offload when stimulated by the application passing a reference to a HOT template to IoTronic *offload* APIs, in particular an offload "trigger" request (i.e., endpoint where to POST an URL), where details of the application components are encoded, especially in terms of (separate) *scaling groups*, as a crude form of component segregation. There are no special implications for the design of the application, apart from the "reflective" property of the application having a pointer (e.g., URL) to its own template manifest, and the logic to engage the offload interfaces. This process starts the *Discovery* of nodes by signaling reduced performance through *synthetic* manipulation of certain metrics fed to Cloud Manager monitoring subsystem (Monasca) as a way to trigger (via a corresponding event) the Heat orchestration subsystem of the Cloud Manager. This approach, e.g., misleading the monitoring system, is just meant for a proof-of-concept implementation and should be actually redesigned for adherence to recent OpenStack developments, e.g., Placement APIs. This way, the Cloud Manager identifies potential candidates—either across cells and within cells, as well as across the hierarchy (i.e., vertically)—for auto-scaling and intercepts the output, i.e., a list of available Fog–Edge nodes with relevant metadata (e.g., node capabilities/peripherals), from the message bus, actually restraining such output from reaching its natural destination, i.e., the Magnum container provisioning subsystem of the Cloud Manager.

The initiator can then compare the main functional requirements each node has to meet to be part of the computation to the aforementioned information about nodes. Available nodes can then be ranked accordingly during the *Selection* phase. The list of selected (worker) nodes is then fed back (either in its entirety or split in a number of subsets) to Cloud Manager-hosted Heat (*Placement* phase). This is done for the latter to build the cluster out of the listed nodes to be handed off to the relevant subsystem (i.e., Cloud Manager's Magnum in this case, considering we are exploiting containers to encapsulate tasks) for two additional steps (*Configuration* phase). These steps include provisioning the instances for the offloaded workloads and requesting IoTronic (i.e., the Cloud Manager—in the case of inter-cell nodes, or the Fog Manager—in the case of intra-cell nodes) to setup overlays among instances. Once the configuration is finalized, the processing starts running within the instances hosted on the nodes, in parallel to *Orchestration/Lifecycle Management*, the latter delegated to its natural recipient, Heat. Hosted workloads can also enjoy *mobility* by virtue of migration mechanisms, also explored in Puliafito et al. (2018).

6 CASE STUDY

An application domain, where the increased amount of data, generated by multiple wireless sources, has to be processed in a timely manner is Intelligent Surveillance Systems (ISS) (Qian et al. 2011). Widely present in smart urban environments, ISS minimize the involvement of human operators to avoid such shortcomings as, for example, high labor cost or limited capability for multiple screens (Dautov et al. 2017b). Usually, an ISS consists of a number of Internet-connected cameras, installed in a public location, constantly streaming video to a central location equipped with more powerful computing and storage capabilities. Video streams are then processed using existing image/face recognition techniques to detect suspect citizens and prevent potential crimes by alerting police officers nearby. ISS rely on existing technological achievements in computer vision and machine learning, used to identify certain patterns (abnormal behavior, suspicious objects, missing people, etc.) in video streams. The recent development of neural networks found a successful application in object detection and recognition scenarios, widely present in ISS. It is worth noting that face detection/recognition using neural networks is considered a computationally intensive task (often requiring computation to take place on a GPU), which cannot be performed by an Edge device (i.e., a smart CCTV camera) on its own, and therefore raw data are required to be transferred to an external computational service for analysis. In this respect, a typical ISS process can be conceptually split into main three steps, as depicted in Figure 8:

— ***Video capturing (VC)*** is undertaken by CCTV cameras, which continuously capture raw video and transfer it for processing as a continuous stream or as a sequence of sampled static images. Given the increased adoption of wireless CCTV cameras and the ever-growing image resolution, this may result in extreme amounts of network data transfers.
— ***Object detection (OD)*** is performed by a neural network framework, which applies sophisticated object detection and feature extraction techniques to detect/recognize objects in the input video based on a pre-trained model.
— ***Alerting and Notification (AN)*** is required to notify interested parties (e.g., the police or system administrators) via an e-mail, message queue, API, and so on, upon object detection.

6.1 ISS Offloading Workflow

It is assumed that the scenario is running in a smart urban environment with multiple Internet-connected devices in close proximity to each other (albeit not necessarily located on the same LAN) and equipped with either an IoTronic Edgent (Edge Manager)—in the case of devices at the Edge or an IoTronic Cell Head (Fog Manager)—in the case of nodes higher up in the topology hierarchy. Below, we revisit the phases described in Section 5.2 and explain how the three-step ISS process of Figure 8 can be implemented using the proposed approach and the S4T framework.

— ***Discovery***: The whole process is initiated by a smart CCTV camera, configured to detect a human and equipped with a corresponding neural network framework. The camera is also equipped with internal behavioral policies for situations when emerging application requirements go beyond its capabilities—e.g., the frame sampling frequency needs to be increased, in which case image processing cannot be supported by the camera on its own. In these circumstances, the camera initiates an offloading procedure: Having received the offloading request, the Cloud Manager orchestrator is able to (re-)evaluate basic requirements and identify a set of potential (Cloud, Fog, and/or Edge) nodes for task offloading, according to suitable resource segregation metadata (e.g., "host aggregates," "availability zones," etc.) exposed by nodes.

— ***Selection***: The CCTV camera finds suitable nodes to support image processing by their participation in a cluster, ranking them according to their suitability and availability, as well as their (possibly) close proximity to the camera. To support device discovery and selection in the heterogeneous IoT ecosystem, it is important to provide a common semantic vocabulary to provide abstract definitions for heterogeneous devices, model the Edge environment, and express concepts related to data processing in a unified manner. To this purpose, we adopt the established Semantic Web technology stack. Following the principle of *reusability*, we are extending the existing M3-Lite (Agarwal et al. 2016), IoT-Lite (Bermudez-Edo et al. 2017), and SSN (Compton et al. 2012) ontologies to provide a high-level taxonomy of concepts to describe IoT devices, as well as their sensing and actuation capabilities. On the one hand, these ontologies serve to define a common semantic vocabulary of terms (i.e., an ontology consisting of concepts, properties, relationships, and constraints) for modelling the IoT domain. On the other, the underpinning capabilities for formal reasoning based on Description Logics provide facilities to perform automatic match-making—i.e., evaluating whether specific characteristics of an Edge device "fit" the current task requirements (Dautov et al. 2014, 2017c). Taken together, the semantic ontological vocabulary and the set of match-making rules constitute a common knowledge base that underpins the overall approach.

— ***Placement***: Once selected nodes and tasks have been paired, it is required to deploy the workflow topology on the cluster. The camera keeps an (internal) representation of the assignment of tasks to be offloaded to suitable nodes and allocates matching tasks to the latter thus establishing the cluster, then passing the node-task list to the orchestrator.

— ***Configuration***: Next, the orchestrator accordingly asks a manager entity (at the Fog or Cloud level) to set up the cluster, and directs the container subsystem to provision application-specific logic (and corresponding software dependencies if required) as container instances on the clustered nodes. It also instructs the container provisioning subsystem to request an overlay network among containers to be fulfilled by the Fog–Edge subsystem. As a result, the instances can now directly exchange information and cooperate.

— ***Orchestration and Lifecycle Management***: Finally, the instances hosted by the clustered devices are now able to receive and process object detection tasks and are requested to continuously run the object detection procedure over the images captured by the CCTV camera and notify interested parties as required.

6.2 Setup of Experiments and Results

As far as horizontal and vertical offloading are concerned, four different patterns, depicted in Figure 9, can be identified:

— ***Vertical offloading to the Cloud (VOC)*** is an existing pattern, where raw video images are transferred to the Cloud for object detection. CCTV cameras are only responsible for video capturing, whereas more complex and resource-intensive detection and recognition operations are performed on the Cloud side. In the experiments, the object detection component was deployed on an AWS EC2 instance (T2 Medium: Intel Xeon Haswell CPU@2.4GHz, 4GB RAM).

— ***Vertical Offloading to the Fog (VOF)*** is also an existing pattern, where computation is offloaded to more powerful devices in close proximity the CCTV camera. Even though such

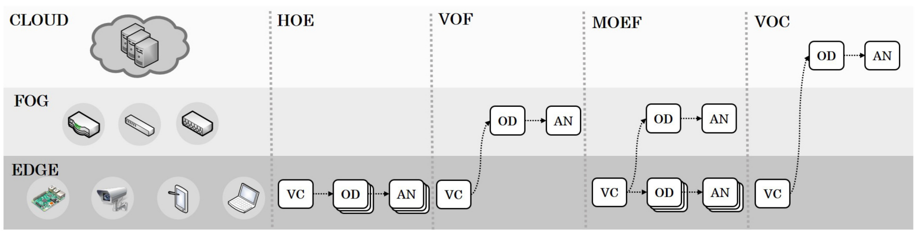


Fig. 9. Allocation of ISS tasks to computing nodes in each setup.

devices are less capable than the Cloud ones, the network latency is minimized, thereby resulting in faster processing results. In the current experiments, the Fog environment is represented by a server machine (Intel Core i3-2100 CPU@3.10 GHz, 4GB RAM) located on the same LAN.

— ***Horizontal offloading to clustered Edge devices (HOE)*** is a novel proposed pattern, where local Edge devices located at the same level of the network topology are able to pool their resources and thereby enable image processing, not feasible for a single Edge device otherwise. In the current experiments, Edge devices are represented by five Raspberry Pi 3 boards (ARM Cortex-A53 CPU@1.2GHz, 1GB RAM), connected into an Edge cluster, as described above.
— ***Mixed offloading to clustered Edge devices and the Fog (MOEF)*** is a pattern that combines the previous two to enable even faster processing. Raw images are offloaded to both Edge devices nearby (i.e., Raspberry Pis) and the network-connected server. As a result, the resulting pool of shared computational resources is larger than the two patterns would potentially offer independently.

In all four offloading patterns, the source CCTV camera was able to capture and transfer five MPixel images of 1.2MB on average. Object detection was implemented using an existing neural network framework Darknet (Redmon et al. 2016) (executable on both Intel and ARM processors). As part of the underlying object detection system, the YOLO model and its more "lightweight" and faster sub-model, TinyYOLO, were used. In all experiments, the system was configured to send an e-mail alert upon detection of a human in an incoming image. Three micro-services (as such suitable to be hosted in containers) were implemented—namely, `CaptureVideo`, `DetectObject`, and `SendNotification`—to design the reference ISS process in Figure 8. As shown in Figure 9, in the vertical offloading scenarios (i.e., **VOC** and **VOF**), object detection is executed on a single machine, whereas in the horizontal offloading scenarios (i.e., **HOE** and **MOEF**), it runs in parallel on multiple clustered devices. The **MOEF** architecture was achieved by integrating the server machine into an existing cluster of Raspberry Pi boards, such that image processing tasks were distributed among six nodes in total—i.e., 5 Raspberry Pi boards and 1 (much more powerful) server machine. Network communication was implemented using the standard TCP/IP stack. In the cases of **HOE**, **MOEF**, and **VOF**, all processing nodes are located on the same LAN and are able to communicate using either Wi-Fi or Ethernet channels. Network bandwidth and latency of the four testbeds are summarized in Table 1. To benchhmark these metrics, standard Linux commands for measuring network bandwidth and latency were used.

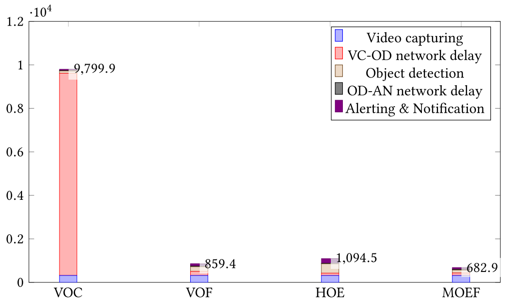
The primary goal of the experiments was to demonstrate how the performance of ISS object detection could be improved. The experimental results are summarized in Figure 10. More specifically, Figure 10(a) depicts the total amount of time required to process a single video frame by each setup, including latency delays due to transferring intermediate results from one processing step to another. In all setups, the video capturing step has a flat value of 317.8ms, whereas

Table 1. Testbed Network Bandwidth and Latency

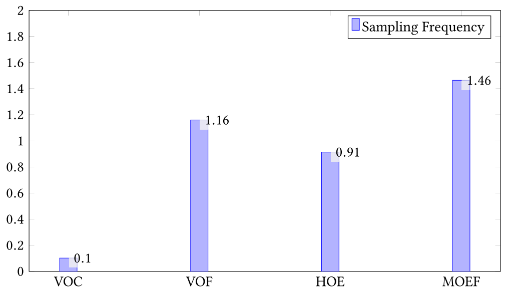
	Uplink, Mbit/s	Downlink, Mbit/s	Round Trip Time, ms
HOE	21.45	22.41	50
MOEF	22.33	23.76	54
VOF	22.09	23.11	51
VOC	2.73	5.18	144

Table 2. CPU and RAM Utilization of Computing Nodes

	HOE	**MOEF**		**VOF**	**VOC**
		Edge devices	**Fog server**		
CPU load	100%	44%	10%	15%	6%
Memory use	83%	38%	12%	19%	9%



(a) Processing time per frame (ms)



(b) Saturation sampling frequency (frames per second)

Fig. 10. ISS image processing benchmarking results.

the next two steps in the **HOE** and **MOEF** setups run in parallel. As it follows from the diagram, the slowest processing results (9,780ms per image) are demonstrated by the **VOC** setup, despite the powerful hardware specification of Cloud machines that remained under-utilized. This can be easily explained by the increased latency associated with transferring separate frames over the Internet to a remote Cloud service, as evidenced by Table 1. Workload offloading to the Fog environment could successfully address this limitation by avoiding data transfers over congested public networks. This way, the **VOF** setup is able to process incoming images at a rate of 859.4ms per image. Similarly, by keeping computation locally, the **HOE** setup was able to accomplish an image processing task in 1094.5ms on average. Finally, a promising result was demonstrated by the hybrid **MOEF** setup able to process a single image in 682.9ms.

By knowing the processing rate of each setup, it is possible to calculate the maximum throughput for each setup—i.e., the highest number of continuously processed frames, at which the rate of image generation or sampling frequency is less than or equal to the rate of image processing (i.e., the bottleneck). Increasing the sampling frequency beyond this threshold will lead to a constantly growing number of queuing images, which will eventually saturate the system forcing it to drop frames (i.e., *saturation sampling frequency*). According to the queueing theory flow balance assumption (Lazowska et al. 1984), the saturation sampling frequency corresponds to the maximum throughput the system can accommodate, depicted in Figure 10(b). This way, the **VOC** setup cannot satisfy a demand higher than 0.1 frames per second and therefore is not expected to be considered for time-critical ISS scenarios. The **VOF** and **HOE** setups independently can achieve 1.16 and 0.91 frames per second, respectively, whereas by combining the two offloading types, it is possible to achieve an even higher frequency of 1.46 frames per second, as demonstrated by the hybrid **MOEF** setup. In the case of mixed workloads, i.e., multiple video streams are pushed from several cameras to one of the ISS setups (assuming that processing demands are the same), the overall saturation sampling frequency is still the one shown in Figure 10(b), requiring the sum of the sampling frequencies of involved cameras to be lower than the saturation thresholds.

Despite the potential benefits of the horizontal offloading in terms of performance and network latency, certain issues and assumptions are to be considered. Most importantly, in practice Edge devices should contribute to a shared pool of clustered resources only upon availability on a volunteering basis, since they are supposed to be running their own dedicated tasks. In these circumstances, combining the horizontal and vertical offloading patterns has the potential to enable a balanced and fair task distribution, so that Edge devices are not overloaded with processing. To demonstrate this, we measured the CPU and RAM utilization of the Raspberry Pis at a maximum image sampling frequency of 2.33 frames per second for the **HOE** and **MOEF** setups. The results, summarized in Table 2, indicate that by introducing a network server to the cluster of Edge boards and thus implementing the mixed offloading pattern, it is possible to decrease the average CPU and RAM utilization up to 44% and 39%, respectively. This benefit becomes particularly important in the presence of battery-powered Edge devices, when the offloading priority is given to Fog nodes to avoid draining the battery of Edge devices. For comparison, Table 1 also contains corresponding values for the **VOC** setup. Noteworthy, the cloud machine stays almost idle, while waiting for incoming frames to be processed.

7 DISCUSSION AND CONCLUSION

As the world is becoming increasingly digital, more and more spaces are turned into smart environments, ranging from individual houses and offices to factories and hospitals and even to whole cities. One of the main challenges to be addressed in this respect is related to enabling efficient data processing, which may span across three different levels, supported by the technological advances in Edge, Fog, and Cloud computing domains. A combination of these paradigms allows, when possible, to push intelligence closer to the data source avoiding transferring and disclosing potentially sensitive data on a public network. However, there is still a lack of methodological and technical approaches to support these processes, able to identify patterns for computational task offloading and related orchestration. This article makes a first step toward offloading engineering as a scientific discipline and aims at filling the gap by identifying main offloading patterns (i.e., vertical and horizontal) and their combinations, which can be applied to meet the smart environment application requirements at hand. To this extent, a middleware platform to enable offloading workload design and engineering, exploiting Edge, Fog, and Cloud facilities, is proposed, extending the current functionality of the existing platforms OpenStack and Stack4Things with support for multiple infrastructure levels on Cloud, Fog, and Edge nodes and introducing and exploiting the novel concept of cells at the lower two levels.

From this perspective, the proposed mechanism for dynamic discovery and integration of computing resources at various network levels to meet fluctuating data processing demands exhibits *software-defined* characteristics. That is, the infrastructure to enable computation is not fixed but can rather adjust with respect to current workload, available resources, network latency, security constraints, and so on. This non-trivial behavior, however, should be driven by policies and a corresponding policy enforcement mechanism. To this end, future work will focus on application of existing software-defined computing techniques. Moreover, in terms of (Big) data management, the OpenStack ecosystem already covers most use cases through the Sahara[7] project, which provides users with a simple means to provision data processing frameworks, such as Apache Hadoop, Spark, and Storm, on OpenStack. In particular, Big *Streaming* Data frameworks, such as Apache NiFi, are naturally more amenable to Edge/IoT scenarios (Dautov et al. 2017a, 2017b).

[7]https://wiki.openstack.org/wiki/Sahara.

Results obtained from the evaluation of a case study on an intelligent surveillance system demonstrated the viability of the proposed solution in time/resource-constrained smart environment scenarios, fueling future work on developing more complex orchestration policies and strategies. In particular, reinforcement learning can be used to define autonomic mechanisms allowing the infrastructure, e.g., to optimize computing and storage resources by opportunely moving the application across the different layers (Cloud, Fog, Edge) according to mobility patterns and non-functional requirements. Other important aspects, albeit not yet covered by this article, are privacy, trustworthiness, and security of resources and data in a heterogeneous Cloud–Fog–Edge environment. Admittedly, mechanisms for offloading computation to external devices and delegation of rights to manage local resources by third parties requires an unprecedented level of trustworthiness—a challenging requirement that has to be tackled from both technological and sociological perspectives.

REFERENCES

Rachit Agarwal, David Gomez Fernandez, Tarek Elsaleh, Amelie Gyrard, Jorge Lanza, Luis Sanchez, Nikolaos Georgantas, and Valerie Issarny. 2016. Unified IoT ontology to enable interoperability and federation of testbeds. In *Proceedings of the IEEE 3rd World Forum on Internet of Things (WF-IoT'16)*. IEEE, 70–75.

Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices architecture enables DevOps: Migration to a cloud-native architecture. *IEEE Softw.* 33, 3 (2016), 42–52.

Zakaria Benomar, Dario Bruneo, Salvatore Distefano, Khalid Elbaamrani, Noureddine Idboufker, Francesco Longo, Giovanni Merlino, and Antonio Puliafito. 2018. Extending openstack for cloud-based networking at the edge. In *Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData).* IEEE Computer Society. DOI: https://doi.org/10.1109/Cybermatics_2018.2018.00058

Maria Bermudez-Edo, Tarek Elsaleh, Payam Barnaghi, and Kerry Taylor. 2017. IoT-Lite: A lightweight semantic model for the internet of things and its use with dynamic semantics. *Pers. Ubiq. Comput.* 21, 3 (2017), 475–487.

Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. 2016. Integration of cloud computing and internet of things: A survey. *Fut. Gener. Comput. Syst.* 56 (2016), 684–700. https://doi.org/10.1016/j.future.2015.09.021

Dario Bruneo, Salvatore Distefano, Francesco Longo, and Giovanni Merlino. 2016a. An IoT testbed for the software defined city vision: The #SmartMe project. In *Proceedings of the 2016 IEEE International Conference on Smart Computing, SMARTCOMP 2016* (2016). DOI: https://doi.org/10.1109/SMARTCOMP.2016.7501678

Dario Bruneo, Salvatore Distefano, Francesco Longo, Giovanni Merlino, and Antonio Puliafito. 2016b. IoT-cloud authorization and delegation mechanisms for ubiquitous sensing and actuation. In *Proceedings of the IEEE 3rd World Forum on Internet of Things (WF-IoT'16)*. 222–227. DOI: https://doi.org/10.1109/WF-IoT.2016.7845494

Dario Bruneo, Salvatore Distefano, Francesco Longo, Giovanni Merlino, and Antonio Puliafito. 2018. I/Ocloud: Adding an IoT dimension to cloud infrastructures. *Computer* 51, 1 (Jan. 2018), 57–65. DOI: https://doi.org/10.1109/MC.2018.1151016

Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, et al. 2012. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semant.* 17 (2012), 25–32. https://doi.org/10.1016/j.websem.2012.05.003

Adrian Copie, Teodor-Florin Fortis, Victor Ion Munteanu, and Viorel Negru. 2013. From cloud governance to IoT governance. In *Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA'13)*. IEEE, 1229–1234.

Rustem Dautov, Salvatore Distefano, Dario Bruneo, Francesco Longo, Giovani Merlino, and Antonio Puliafito. 2017a. Pushing intelligence to the edge with a stream processing architecture. In *Proceedings of the 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE.

Rustem Dautov, Salvatore Distefano, Giovani Merlino, Dario Bruneo, Francesco Longo, and Antonio Puliafito. 2017b. Towards a global intelligent surveillance system. In *Proceedings of the 11th International Conference on Distributed Smart Cameras (ICDSC'17)*. 119–124.

Rustem Dautov, Iraklis Paraskakis, and Mike Stannett. 2014. Utilising stream reasoning techniques to underpin an autonomous framework for cloud application platforms. *J. Cloud Comput.* 3, 1 (2014), 13.

Rustem Dautov, Symeon Veloudis, Iraklis Paraskakis, and Salvatore Distefano. 2017c. Policy management and enforcement using OWL and SWRL for the internet of things. In *Proceedings of the International Conference on Ad-Hoc Networks and Wireless*. Springer, 342–355.

Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of things (IoT): A vision, architectural elements, and future directions. *Fut. Gener. Comput. Syst.* 29, 7 (2013), 1645–1660.

Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. Serverless computation with OpenLambda. In *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing*. USENIX Association, Berkeley, CA, 33–39.

Karthik Kumar and Yung-Hsiang Lu. 2010. Cloud computing for mobile users: Can offloading computation save energy? *Computer* 43, 4 (2010), 51–56.

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. 1984. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Upper Saddle River, NJ.

Francesco Longo, Dario Bruneo, Salvatore Distefano, Giovanni Merlino, and Antonio Puliafito. 2016. Stack4Things: A sensing-and-actuation-as-a-service framework for IoT and cloud integration. *Ann. Telecommun.* 72, 1–2 (2016), 1–18.

Sunilkumar S. Manvi and Gopal Krishna Shyam. 2014. Resource management for infrastructure as a service (IaaS) in cloud computing: A survey. *J. Netw. Comput. Appl.* 41 (2014), 424–440. https://doi.org/10.1016/j.jnca.2013.10.004

Antonio Manzalini and Noel Crespi. 2016. An edge operating system enabling anything-as-a-service. *IEEE Commun. Mag.* 54, 3 (2016), 62–67.

Evangelos K. Markakis, Kimon Karras, Nikolaos Zotos, Anargyros Sideris, Theoharris Moysiadis, Angelo Corsaro, George Alexiou, Charalabos Skianis, George Mastorakis, Constandinos X. Mavromoustakis, et al. 2017. EXEGESIS: Extreme edge resource harvesting for a virtualized fog environment. *IEEE Commun. Mag.* 55, 7 (2017), 173–179.

Xavi Masip-Bruin, Eva Marín-Tordera, Ghazal Tashakor, Admela Jukan, and Guang-Jie Ren. 2016. Foggy clouds and cloudy fogs: A real need for coordinated management of fog-to-cloud computing systems. *IEEE Wireless Commun.* 23, 5 (2016), 120–128.

Giovanni Merlino, Dario Bruneo, Francesco Longo, Salvatore Distefano, and Antonio Puliafito. 2015. Cloud-based network virtualization: An IoT use case. In *Proceedings of the International Conference on Ad Hoc Networks*. Springer, 199–210.

Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin, and Brian Lee. 2016. A container-based edge cloud PaaS architecture based on Raspberry Pi clusters. In *Proceedings of the IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloudW'16)*. IEEE, 117–124.

Claus Pahl and Brian Lee. 2015. Containers and clusters for edge cloud architectures–A technology review. In *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud'15)*. IEEE, 379–386.

Tamas Pflanzner and Attila Kertész. 2016. A survey of IoT cloud providers. In *Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'16)*. IEEE, 730–735. DOI: https://doi.org/10.1109/MIPRO.2016.7522237

Carlo Puliafito, Enzo Mingozzi, Carlo Vallati, Francesco Longo, and Giovanni Merlino. 2018. Companion fog computing: Supporting things mobility through container migration at the edge. In *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP'18)*. IEEE, 97–105.

Huihuan Qian, Xinyu Wu, and Yangsheng Xu. 2011. *Intelligent Surveillance Systems*. Vol. 51. Springer Science & Business Media, New York, NY.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 779–788.

Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. 2016. Resource provisioning for IoT services in the fog. In *Proceedings of the IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA'16)*. IEEE, 32–39.

Salman Taherizadeh, Vlado Stankovski, and Marko Grobelnik. 2018. A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers. *Sensors* 18, 9 (2018), 2938.

Ricard Vilalta, Arturo Mayoral, David Pubill, Ramon Casellas, Ricardo Martínez, Jordi Serra, Christos Verikoukis, and Raul Muñoz. 2016. End-to-end SDN orchestration of IoT services using an SDN/NFV-enabled edge node. In *Proceedings of the Optical Fiber Communications Conference and Exhibition (OFC'16)*. IEEE, 1–3.

V. Vinothina, R. Sridaran, and Padmavathi Ganapathi. 2012. A survey on resource allocation strategies in cloud computing. *Int. J. Adv. Comput. Sci. Appl.* 3, 6 (2012), 97–104.

Jianyu Wang, Jianli Pan, Flavio Esposito, Prasad Calyam, Zhicheng Yang, and Prasant Mohapatra. 2018. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–23.

Zhenyu Wen, Renyu Yang, Peter Garraghan, Tao Lin, Jie Xu, and Michael Rovatsos. 2017. Fog orchestration for internet of things services. *IEEE Internet Comput.* 21, 2 (2017), 16–24.

Song Wu, Chao Niu, Jia Rao, Hai Jin, and Xiaohai Dai. 2017. Container-based cloud platform for mobile computation offloading. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Symposium (IPDPS'17)*. IEEE, 123–132.

Received November 2018; revised January 2019; accepted January 2019