Adaptive mobile Web server framework for Mist computing in the Internet of Things

Mohan Liyanage, Chii Chang and Satish Narayana Srirama
Mobile Cloud Lab, Institute of Computer Science, University of Tartu, Tartu, Estonia

Received 28 March 2018
Accepted 11 April 2018

Abstract

Purpose – The distant data centre-centric Internet of Things (IoT) systems face the latency issue especially in the real-time-based applications, such as augmented reality, traffic analytics and ambient assisted living. Recently, Fog computing models have been introduced to overcome the latency issue by using the proximity-based computational resources, such as the computers co-located with the cellular base station, grid router devices or computers in local business. However, the increasing users of Fog computing servers cause bottleneck issues and consequently the latency issue arises again. This paper aims to introduce the utilisation of Mist computing (Mist) model, which exploits the computational and networking resources from the devices at the very edge of the IoT networks.

Design/methodology/approach – This paper proposes a service-oriented mobile-embedded Platform as a Service (mePaaS) framework that allows the mobile device to provide a flexible platform for proximal users to offload their computational or networking program to mePaaS-based Mist computing node.

Findings – The prototype has been tested and performance has been evaluated on the real-world devices. The evaluation results have shown the promising nature of mePaaS.

Originality/value – The proposed framework supports resource-aware autonomous service configuration that can manage the availability of the functions provided by the Mist node based on the dynamically changing hardware resource availability. In addition, the framework also supports task distribution among a group of Mist nodes.

Keywords Internet of Things, Edge computing, Mobile services, Fog computing, Mist computing

Paper type Research paper

1. Introduction

Emerging Internet of Things (IoT) (Gubbi *et al.*, 2013) applications interconnect heterogeneous devices to the internet towards providing various advanced ubiquitous services to users. However, the classic design of IoT systems (Chang *et al.*, 2017b), which relies on distant central processing servers, faces the issue of latency. Specifically, in many real-time ubiquitous applications (e.g. augmented reality, environmental analytic and ambient assisted living), users demand rapid responses.

To address the issue, researchers have introduced numerous edge computing methods. For examples, European Telecommunications Standards Institute introduced multi-access edge computing (MEC) model (Patel *et al.*, 2014) that enables the integration between the mobile application service and the cellular network co-located VM servers. Furthermore, the Fog computing architecture (Bonomi *et al.*, 2012) introduces the embedded virtualisation technology in the gateways (e.g. router, switch and set-top box) of the front-end IoT devices to support the data pre-processing at the edge networks. Indie Fog architecture applies

This research was supported by Estonian Research Council grant IUT20-55.

International Journal of Pervasive Computing and Communications

1742-7371
DOI 10.1108/IJPCC-D-18-00023

consumer-as-provider model in which the small businesses and individuals offer edge computing platforms from their own equipment (Chang *et al.*, 2017a).

Although the edge computing methods are promising, the increasing number of IoT devices (e.g. wireless sensors, actuators, smartphones and vehicles.) will overload the capability of the edge computing servers. Hence, the edge computing servers will face bottleneck issues and the latency will still arise (Preden *et al.*, 2015).

To reduce the burden of edge computing servers, researchers have introduced Mist computing model (Mist) (Preden *et al.*, 2015; Pulli *et al.*, 2011). In Mist, the computing moves to the extreme edge of the IoT environment, where there are a vast number of heterogeneous mobile devices and IoT devices that are capable of providing certain forms of services to assist the improvement of computational processes needed in IoT applications (Preden *et al.*, 2015).

Figure 1 illustrates the role of Mist in IoT environment.

As Figure 1 shows, Mist servers host on various objects, such as single board computers, mobile devices, embedded servers of vehicles and so forth. Essentially, Mist servers (or Mist nodes) are different to the regular embedded Web servers (EMS) (Nicoloudis and Pratistha, 2003; Srirama *et al.*, 2006a, 2006b; Liyanage *et al.*, 2015) that provide static software services to their clients. For example, an EMS can simply provide its current location as a Web service via mobile internet with common request–response interaction. On the other hand, Mist nodes provide a more flexible environment in which they can execute customised program methods sent from their requesters.

> Mist Computing (Mist) represents a paradigm in which edge network devices, which have predictable accessibility and have sensing or actuating capabilities, provide their computational and communicative resources as services to their vicinity via Device-to-Device communication protocols. Requesters in Mist can distribute software processes to Mist service providers for execution.

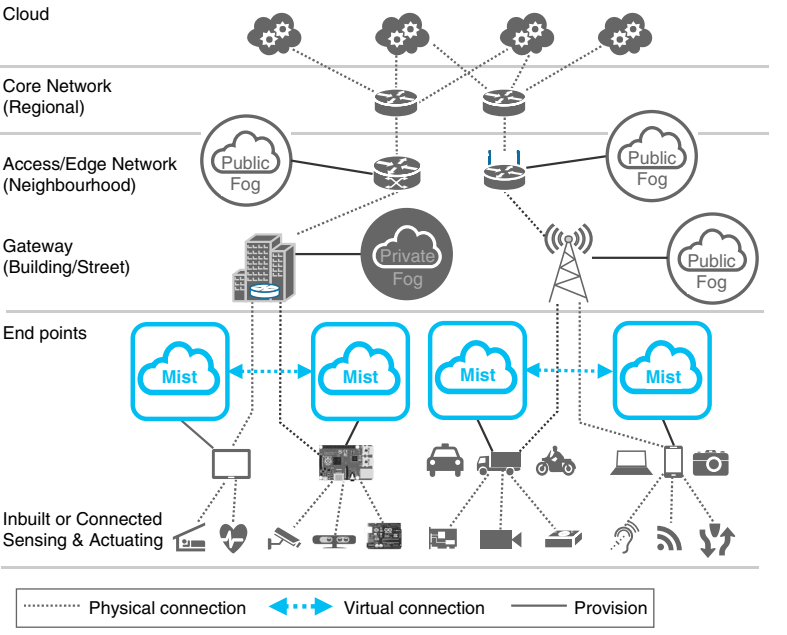

Figure 1.
The role of Mist in IoT

The fundamental concept of Mist may not be new. In past, several proposals (Chu and Humphrey, 2004; Murray *et al.*, 2010; Loke *et al.*, 2015) have introduced the solution that uses the constraint Acorn RISC Machine (ARM) CPU-based devices to perform the collaborative distributed computing. However, the gap is, there is no particular description or proposal that has described how to establish such kind of environment in a generic, platform-independent approach. Past works assume that a specific application or a standalone platform will be pre-installed so that the process can be distributed among the devices. Usually, the proposals focus on one particular use case ([Wang *et al.*, 2013] that is for message forwarding only), and the prototype may not be feasible for different use cases. If the intention moves forward to platform-independent MAGC solutions, which require loosely coupled interoperability, it faces the following issues.

In the real world, the less flexible OS-based devices (e.g. Android OS and iOS) have been applied in various domains, such as mobile phones, tablets, home appliances and wearable devices. The hardware components of these devices cannot be accessed by sandbox environment freely unless they use customised ROMs. Hence, the computational process distributed to them is very limited. However, the high density of these devices in urban areas will play an important role in Mist computing. There is a need for a more generic model for enabling the less flexible OS devices to participate in the Mist.

The usage of mobile devices is quite dynamic. Although it is possible to predict the accessibility of the device based on their users' daily activities, the hardware resource availability is less predictable as the users may randomly use their devices for different purposes even when they are not moving out of the accessible range. Hence, there is a need for autonomously reconfiguring the functional availability of Mist nodes based on the dynamic contextual factors.

To address the issues, we are proposing mobile-embedded Platform as a Service (mePaaS). As the technology evolved, today's ARM CPU-powered mobile devices can outperform an entry-level virtual machines (Chang *et al.*, 2015), and thus they are capable of supporting such a platform. mePaaS nodes can execute customised computational processes defined by their requesters. mePaaS nodes loan their hardware resources to the others based on certain service level agreement (SLA). The incentive can be receiving credit or discount from their cellular network providers who intend to provide Fog computing services. The requester who uses mePaaS-formed Mist may have the contract with the same cellular network provider as part of their monthly package. By using the Mist, cellular network providers can reduce the usage of their Fog nodes so that one Fog node may be able to serve more requests.

This paper is an extension of Liyanage *et al.*'s study (2016), and provides an overview of the recent technologies towards proposing a generic software framework for Mist computing. Moreover, the framework consists of a new module that addresses the hardware usage relative to the temporal space, especially for handling the complex type of services. The framework aims to address the two issues described before:

(1) a flexible program execution environment in the mobile device; and

(2) self-adaptive resource management. A prototype has been developed on the basis of mePaaS, and has been thoroughly tested, as a proof-of-concept.

The rest of the paper is organised as follows: Section 2 provides an overview of related works and Section 3 presents the design details of the proposed framework followed by the description of the main components. Section 4 describes the performance evaluation of the proposed framework with the prototype implementation. The paper concludes in Section 5 along with future research directions.

2. Background and related work

Using the proximity-based computational resources is not new and had attracted significant attention from both industry and research communities. This section explores the applications of mist computing and reviews the relevant approaches proposed during the recent years.

2.1. Applications of mist computing

Mist can enhance IoT in various applications. Below, we propose three scenarios to illustrate the usefulness of Mist. Imagine an ambient-assisted living (AAL) (Dar *et al.*, 2015) service that requires service composition with environmental IoT devices to support the real-time data analytic towards providing useful information to the mobile user while the user is in the outdoor area. In a classic system, it is expected that either most of the IoT devices are interoperable via the standard service/resource discovery schemes or they are operated within the same information system. However, in reality, even though the organisations who own the devices are willing to share the sensory data they collect, they may be operating in different protocols and it will cause high latency if the entire service is relying on the distant data centre to communicate with all the IoT devices' backend or their discovery servers for identifying which of them are providing the information that is of interest by the AAL service. With Mist, the AAL server can directly offload certain tasks to the user's mobile device dynamically to improve the agility. For example, some IoT devices may support Google Physical Web, as Mist allows the AAL server to deploy the matchmaking algorithm and the communication processes to the mobile device, it can dynamically configure the interaction between the mobile device and the environmental IoT devices by sending and executing the workflow on the devices. Hence, the overall process agility can be improved.

Mist can also improve mobile crowd-sensing services. In mobile crowd-sensing scenarios, sensing tasks usually are deployed from the distant data centre to the front-end mobile devices (collectors). Collectors will continue collecting and streaming the data to the distant data centre for a period of time. To save energy of mobile devices, sometimes collectors may use the mobile devices in their close proximity for brokering the data. However, it may consume a lot of extra resources for the collectors to compute, which is the best path to forward the data continuously. On the other hand, if Mist is enabled, the distant data centre can continue calculating the best path for each collector, and then dynamically deploy the software-defined networking (Nunes *et al.*, 2014) as a program for the collectors and Mist-enabled brokers without the participating node to pre-install the common applications.

Context as a Service (CaaS) is another promising scenario for Mist. CaaS represents an information service that is providing the on-demand context information to its requester. The context information is the interpreted information computed by executing the algorithm defined by the requester on the CaaS node collected sensory information (the customised algorithm is to achieve the need of 'context can be subjective' (Penco, 1999). In Fog computing environment, a mobile application can request a Fog node to perform real-time context-aware service for the mobile user. Instead of performing all the tasks on the Fog node itself, which may result in bottleneck and latency issues (Preden *et al.*, 2015), the Fog node can use the Mist nodes located in the same area to collect sensory data and pre-process them into semantic context information, then send the context information back to the Fog node. Such process can reduce the bandwidth and computational resource usage on the Fog node towards improving the accessibility of the Fog node, when it needs to serve more context-aware applications in its vicinity.

2.2. Proximity-based mobile grid computing

Stantchev *et al.* (2015) proposed a smart healthcare framework that uses three-tier architecture, including a fog layer, which provides low latency. The proposed healthcare and elderly care application is modelled using business process modelling notation (BPMN) and the fog layer enhances the framework by providing low latency and location awareness.

The mobile edge computing (MEC)-based Elastic Android Applications for Computation Offloading (CloudAware) (Orsini *et al.*, 2015) is a framework that offloads tasks to edge devices. Moreover, the proposed approach provides duplication and administration of the application business logic, even under the un-trusted and unpredictable dynamic environments. The MEC application is broken into components, which simplify the offloading decision and facilitate the development of elastic and scalable applications for *ad hoc* mobile clouds. Taleb *et al.* (2017) presented an approach called "Follow Me Edge", which is also based on the concept of MEC to enhance the QoS. The framework ensures the high quality of the services by reducing the core network traffic in which the applications/services also move with the users. Moreover, the authors showed how high quality of service can be maintained in an augmented reality use case. In this use case, a high-definition video is caught at the edge of an attractive location and is streamed to the people visiting that location on demand.

Cloudlet (Satyanarayanan *et al.*, 2009) is a framework that can be used to realise MEC and Fog computing. The main concept is to host virtual machines on the computers of local business and provide the VM as a service to proximity-based users. Hyrax (Marinelli, 2009) is a framework that aims to apply MapReduce to the grid computing environment formed by a group of mobile devices. The result of Hyrax shows that the approach is too heavy weight. To overcome the deployments encounters and limitations of the Cloudlets system, a hierarchical model of MEC–Cloudlet integrated architecture that uses the MEC system and Cloudlets infrastructures was proposed by Jararweh *et al.* (2016). The framework is designed for large-scale cooperative Cloudlets deployments that provide optimisation for the power consumption and delay.

The Honeybee (Fernando *et al.*, 2012) is a framework that approaches the same purpose as Hyrax. However, instead of using the heavyweight MapReduce, Honeybee applied the active work-stealing scheme in which participants in a mobile grid-computing environment will actively be taking the work from the other nodes that have more tasks to do. Other similar works can be found in Fernando *et al.*'s study (2016).

To minimise the transfer delay between the fog and the cloud and to maximise the resource utilisation of fog nodes, Skarlat *et al.* (2016) presented a conceptual framework for fog resource provisioning. The authors introduced the concept of fog colonies, which are micro data centres made up from an arbitrary number of fog cells. Moreover, within a fog colony, task requests and data can be distributed and shared between the single cells, which facilitate decentralised processing at the edge of the network.

Different to the above-related works, the framework proposed in this paper focuses more on how to provide a flexible program execution platform as a service from the high density, resource shareable mobile devices based on loosely coupled service oriented architecture. Hence, our work can be considered as a complement to the previous mobile grid computing frameworks in which mePaaS can be used to realise the environment in a more flexible way.

2.3. Mobile service provisioning

The provisioning of Web service-oriented server on mobile devices has already been studied from the early 2000s. For examples, a micro Web server that hosts SOAP Web server on Windows CE (Nicoloudis and Pratistha, 2003) and SOAP Web server for Symbian OS

(Srirama *et al.*, 2006a) are mentioned. Today, the trend of mobile Web servers and provisioning has shifted from SOAP to RESTful service provisioning (Liyanage *et al.*, 2015; Mohamed and Wijesekera, 2012). The service description, which originally followed W3C's WSDL and WADL, is slowly moving to RESTful service description such as JSON-LD, which combines service/resource description and semantic description.

Today, various embedded servers already exist, and are being used in small devices for Wireless Sensor and Actuators and IoT. They follow standards such as CoAP and MQTT for constrained devices to provision services. Service provisioning (SP) from constrained devices is no longer a challenge. Moreover, modern smartphones can outperform the low-end IaaS Cloud instances (Chang *et al.*, 2015; Chang *et al.*, 2015) in terms of computational process performance. If we also consider the mobile internet transmission required for using Cloud, the computational offloading to low-end Cloud instance is very inefficient. This is one of the factors driving the rise of proximity-based edge computing.

As described in the previous section, various edge computing models have also been introduced by both industry and academia. The key common idea is to use the proximity-based hardware resources for the computational tasks that cannot be effectively done locally with the mobile requester's own device. As the idea is given by Mobile Crowd Computing (MCrC), the mobile hosts are used for different computational needs. The interoperability and scalability were not the focus of MCrC prototype-Honeybee (Fernando *et al.*, 2012). The mePaaS framework can be seen as the complement of MCrC.

3. Proposed framework

3.1 Architecture overview

The proposed mePaaS framework enables mobile devices to provide a flexible way of sharing their computational and networking mechanisms as services. The core of mePaaS framework is based on Enterprise Service Bus (ESB) (Robinson, 2004) architecture. ESB is a software infrastructure that can easily connect resources by combining and assembling services to achieve a loosely coupled service-oriented architecture. mePaaS uses a plugin module-based approach to mediate native computational and networking components to services that can be invoked by the process execution engine. Requester of mePaaS can send a request package, which consists of input parameters and the flow of processes described in the standard workflow model (e.g. BPMN [www.bpmn.org]) with customised algorithm defined in the supported script language of the process execution engine. mePaaS can execute the workflow that facilitates the available service modules to complete the tasks from the requester.

Figure 2 illustrates the main elements of mePaaS framework, and the elements are described below.

3.1.1 Controller. Controller is the core element of mePaaS framework. Its main tasks include mediating and managing the computational and networking modules to coordinate with the service-provisioning component. Moreover, the controller analyses the requests from the clients, and updates the service description according to the availability of resources. It includes the following important components:

- *Service availability controller*. The service availability controller updates the service availability information when it progresses a new request. As each service module execution consumes a certain amount of hardware and other resources of the device, the service availability controller notifies the controller about the conflicts of services and the overhead usage of the system resource (i.e. CPU, RAM and network bandwidth). For example, let Req1 and Req2 are two requests that use the camera for a sensing task. Assume Req1 has been requested for 20-s video capturing. If Req2 also wanted a picture from the camera at the same time, the Req2 will be

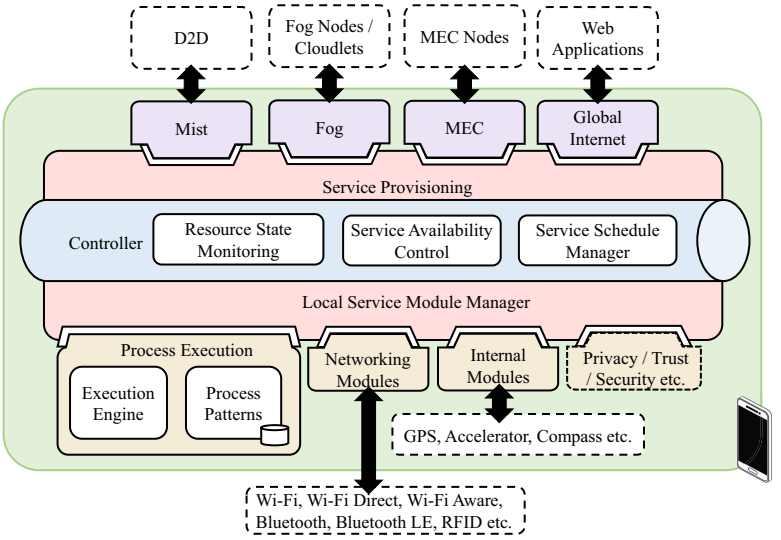

Figure 2.
mePaaS architecture

marked as a conflicting service and is available only after 20 s. Under this situation, the controller will notify the SP component to disable the corresponding service (e.g. image-based sensing) from the service description until the corresponding resources are released.

- *Resource state monitoring*. This component is responsible for monitoring the resource usages continuously, for example, monitoring the device hardware status, CPU load percentage, RAM usage, incoming and outgoing network transmission status, etc. Based on the information arrived from this component, the controller can autonomously make a decision about which service modules are available based on the required usage of resources. The decision-making scheme is described in the next section.
- *Service schedule manager*. This is the component that can predict the availability of resources in the temporal domain. More details of this component are discussed in the later section.

3.1.2 Service provisioning. Service provisioning is the component that handles which service modules can be included in the service description metadata (SDM). mePaaS can publish or advertise its SDM in different networks, depending on the application use case. In general, the SDM follows the W3C's recommendation for RESTful services in the Web of Things (WoT; [www.w3.org/WoT/]), in which SDM is described on the basis of JSON for Linked Data (JSON-LD [www.w3.org/TR/json-ld]) format. JSON-LD is a format that aims to represent the SDM in a JSON representation. For instance, a location sensor data can be presented in JSON-LD format (Su *et al.*, 2015) as follows:

```
{
"@context":
{
"i": "http://iot.fi/o#",
"ownerID": "i:ownerID",
"longitude": "i:longitude", "latitude": "i:latitude"
```

```
},
"@id": "i:locaSensor767",
"@type": "i:LocationSensor",
"ownerID": "Alice",
"longitude": "25.468", "latitude": "65.058"
}
```

SP associates with different service provisioning adaptors (SPA), such as Mist, Fog, MEC and the Web, to publish or advertise mePaaS's SDM. Also, the SPAs are responsible for handling the incoming request packages from their field. For example, Mist SPAs are responsible for advertising SDM on the D2D network (e.g. Wi-Fi Direct, Physical Web [https://google.github.io/physical-web]) and handling the requests from D2D network. Note that, distinguished from mobile *ad hoc* network, the D2D uses existing infrastructure for the communication within a one-hop range (Lin *et al.*, 2014).

3.1.3 Local service module manager. Local service module manager (LSMM) is responsible to launch, terminate and manage the local service modules of mePaaS. Local service modules can be seen as independent software components that can be installed as plugins of mePaaS. Initially, mePaaS should have at least one process execution module and a number of corresponding modules that can use the inbuilt functions of the device (e.g. access GPS data). mePaaS provides a flexible way for developers or users to add more modules for supporting different needs. For example, a user can install additional modules for performing semantic sensory data reasoning. LSMM will inform the controller about newly installed module, along with the corresponding descriptions. The controller will pass the information to SP to include the module as a new service in SDM.

3.1.4 Process execution consists of following main elements.

- *Execution engine* is a software to execute the workflow, which is included in the request package. The execution engine needs to associate with LSMM to invoke the corresponding service modules involved in the workflow tasks.
- *Process patterns* manage a number of predefined workflow patterns. The predefined patterns can be used to replace the inexcusable tasks in the workflow as substitutions. When it receives a request that contains the goal of the process, a corresponding abstract workflow model is executed. A flow relation pattern defines the structure of a set of workflow nodes. The definition of abstract workflow model and approach will be described in a later section.

Networking modules represent the service modules that can invoke the functions of networking requests (e.g. sending HTTP or CoAP requests, sending Bluetooth GET request, retrieving Bluetooth Beacon's data or reading RFID or data from ISO/IEC 20248 devices). These modules can also be used to fetch the SDM of the other service providers in proximity.

Internal modules represent the service modules that only involve the functions from inbuilt hardware resources and sensors, such as GPS, accelerometer, compass and brightness.

Privacy, trust, security, etc. are the modules that involve the management of privacy and SLA, the trustworthiness and quality of SP, cryptography and other security-involved mechanisms. We have studied them earlier in other contexts, such as service discovery (Chang *et al.*, 2014), and they will be integrated later.

3.2 Self-configured service provisioning

Mist node publishes the available services in SDM that the clients will receive in the JSON-LD format. Due to the dynamic nature of the resource usage of service modules, the

availability of the SP is also unpredictable. For example, if the Mist node is currently serving a continuous data-streaming task, then it is unlikely to serve a new request that also requires a heavy bandwidth usage. Hence, the corresponding service may need to be disabled from SDM, as the Mist node cannot handle any more such kind of requests. To dynamically update the SDM, we propose the service-scheduling scheme. First, we explain the terminologies used in the scheme.

As described earlier, the Mist node-handled process is modelled as a workflow. Here, we refer to the terms described in der Aalst's study (1998) where a task that is to be accomplished is called a work item. A work item in mePaaS is executed by a service module.

Definition 1. (Installed service module collection) It is defined as a tuple $< S, \beta >$

where:

$S = \{s_l : 1 \leq l \leq \mathbb{N}\}$ is a set of service modules. Each $s_l \in S$ is defined as a tuple <ID, type, status> corresponding to identification, type of the service module (e.g. CPU intensive and bandwidth intensive) and the availability status.

$\beta : S \rightarrow \mathcal{U}$ is a function that maps the service modules to the required hardware usages.

Definition 2. (Scheduled work items) It is defined as a tuple $< W, F, \zeta, \gamma, \delta >$

where:

$\mathcal{W}$ is a finite set of work items. $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_n\}, n \in \mathbb{N}$

$F \subseteq \mathcal{W} \times \mathcal{W}$ is a set of flow relation rules.

ζ: $\mathcal{W} \rightarrow Z$ is a function that maps work items to status.

γ: $\mathcal{W} \rightarrow \Gamma$ is a function that maps work items to start times.

δ: $\mathcal{W} \rightarrow D$ is a function that maps work items to estimated execution duration.

Let $\cdot\omega = \{\omega | (\omega, \upsilon) \in F\}$ be the pre-set of ω, $\omega\cdot = \{\omega | (\upsilon, \omega) \in F\}$ be the post-set of ω.

Definition 3. (Device hardware usages (H^{Glo})). $H^{Glo} = \left\{h_1^{Glo}, h_2^{Glo}, \ldots, h_n^{Glo}\right\}, n \in \mathbb{N}$. Each $h^{Glo} \in H^{Glo}$ is defined as a tuple $< ID^{Glo}, cu^{Glo}, xu^{Glo} >$ where:

ID^{Glo} denotes the identification of the hardware.

cu^{Glo} denotes the current assigned usage of the hardware based on the scheduled work items.

xu^{Glo} denotes the maximum availability of the hardware.

Initially, we can apply Algorithm 1 in which the SP component can decide whether to keep or remove the services from SDM based on the availability of hardware resources.

Algorithm 1. Pseudocode for our algorithm to decide whether to keep or remove the services from SDM

```
for s ∈ S do
    hardwareUsageSet ← β(s)
    for hus ∈ hardwareUsageSet
        for h^Glo ∈ H^Glo
            if ID_hus ≡ ID_h^Glo ∧ (value_hus + cu_h^Glo) > xu_h^Glo) then
                add s to removeList
            end if
        end for
    end for
end for
```

However, Algorithm 1 does not consider the hardware usage relative to the temporal space. For example, there may be a process that will terminate and release the resources in very near future. To overcome this limitation, we combined a service schedule manager into the controller to enhance the decision-making process and to ensure the unnecessary removing of services. For example, there may be a service request that takes some images from the

surrounding and uploads them to a server periodically. Assume that the request has started at 10.00 a.m. and continues in every 10 min as at 10.10, 10.20, ..., until 11.00 a.m. In that situation, availability of the camera resource is almost free except only at some occasions. Moreover, the system already knows the availability of resources in advance and the framework can make optimal decisions accordingly.

The following section describes the service schedule that optimises the SP.

3.3 Service scheduler

The following section describes a detailed overview of the service scheduling that enhances the SP. The service scheduler combines the available resources with the domain in advance to minimise the service conflicts.

Definition 4. (Local services pool – LSP) It describes the information about local services provided by the Mist node. It is defined as a tuple $< S, \varsigma, \kappa, \varepsilon >$

where:

$\mathcal{S}$ is a set of service modules.

ς: $\mathcal{S} \rightarrow \mathcal{R}$ maps services to hardware components (e.g. GPS sensor, accelerometer, camera, audio recorder and network signal browser).

κ: $\mathcal{S} \rightarrow \mathcal{S}$ maps services to conflict services.

ε: $\mathcal{S} \rightarrow$ E maps services to the system resource usage sets (e.g. CPU, RAM and network transmission bandwidth usage).

Example 1 (Conflict service). Let s_1 and s_2 be two services. Let $\mathcal{T}' \subseteq \mathcal{T}$. Assume s_1 has been requested by a real-time sensing request (e.g. noise-level sensing), which has its timestamp period within $\mathcal{T}'$, and s_2 is not requested by any client. Suppose $s_2 \in \kappa(s_1)$, then s_2 will be marked as unavailable during $\mathcal{T}'$ because it cannot be executed at the same time.

3.4 Scalable computational resources

In Mist, it is expected that the SDM of a Mist node also describes the computational and networking capabilities (CPU, RAM, bandwidth etc.) it can provide. As such information is available, a Mist node can form a grid-computing group centred by itself with other Mist nodes that are within one-hop range from it. Hence, when the Mist node cannot perform a task by itself or it cannot achieve the performance requirement for the task execution, it is possible to distribute the work (by executing a predefined substitute workflow pattern) to the other Mist nodes as long as it generates a more efficient result. However, it raises a question about how does mePaaS makes the decision on which Mist node and when it should distribute the work to?

Here, we propose a work distribution scheme, which is used when a computational offloading node needs to be defined for the work substitution purpose at runtime.

Step 1. The resources required for executing the work item depend on the usage of the corresponding hardware components.

Let RES be the resource for the work item. RES consumes a set of hardware (CPU, memory, bandwidth, etc. based on the historical record and input parameters). Let H be the hardware usage by RES, $H = \{h_k : 1 \leq k \leq \mathbb{N}\}$, where each $h \in H$ is defined as a tuple $(< id, value >)$ corresponding to the identification of the hardware usage and the hardware usage value consumed for executing the RES.

Step 2. The weight of hardware usage required for the work item influences the performance ranking.

Based on the resource for the work item, the weight of hardware usage is different. We categorise them into following types based on the hardware usage considered by Qin *et al.* (2003) and Kondo *et al.* (2009):

CPU intensive task (e.g. customised complex algorithm scripts).

CPU + RAM intensive task (e.g. I/O data processing; large data volume loading involved tasks).

Bandwidth intensive task (e.g. data forwarding process; e.g. send/receive tasks).

Hybrid, where multiple resources have higher weight.

Step 3. Identify availability of the system resources.

Schedule manager updates the service availability information when it progresses a new request. The service availability is influenced by two factors: service conflicts and system resource availability (i.e. CPU, RAM and network bandwidth).

Let $\mathcal{S}^{t_x}$ be a set of scheduled service executions at timestamp $t_x \in \mathcal{T}$, $\mathcal{S}^{t_x} = \{s_m^{t_x} | 1 \leq m \leq \mathbb{N}\}$. $\kappa(s_m^{t_x})$ denotes a set of conflict services of $s_m^{t_x}$. Hence, a set of conflict services at timestamp – t_x (denoted by $\mathcal{K}^{t_x}$) will be:

$$\mathcal{K}^{t_x} = \bigcup_{m \in |\mathcal{S}^{t_x}|} \kappa(s_m^{t_x}) \tag{1}$$

and the available services at timestamp t_x (denoted by $\overline{\mathcal{S}^{t_x}}$) will be $\overline{\mathcal{S}^{t_x}} = \mathcal{S} \backslash \mathcal{K}^{t_x}$.

The above process has only filtered the services based on conflicts. Following is the process that considers the system resource availability.

Let $E^{sys} = \{e_o^{sys} | 1 \leq o \leq \mathbb{N}\}$ be a set of the available system resources (Note that available system resources are different to the hardware specification of the mobile device. User can set the availability in percentage to avoid the SP affecting the normal use of the mobile device).

Let $\mathcal{S}^{t_x}$ be a set of services assigned at $t_x \in \mathcal{T}$. For each service – $s_z \in \mathcal{S}^{t_x}$, its system resource consumption is found in $\varepsilon(s_z)$.

Let $E^{s_z} = \varepsilon(s_z)$ and $E^{s_z} = \{e_o^{s_z} | 1 \leq o \leq \mathbb{N}\}$ in which the system resources denoted by $e_o^{s_z}$ and e_o^{sys} are the same, and let $ve_o^{s_z}$ be the usage value of $e_o^{s_z}$ and ve_o^{sys} be the remaining value of e_o^{sys}. For each t_x, the ve_o^{sys} after assigning $\mathcal{S}^{t_x}$ (denoted by $ve_o^{t_x}$) will be:

$$ve_o^{t_x} = ve_o^{sys} - \sum_{s_z \in \mathcal{S}^{t_x}} ve_o^{s_z} \tag{2}$$

Let $E^{t_x} = \{ve_o^{t_x}\}$. Referring to previous result, $\overline{\mathcal{S}^{t_x}}$ is a set of services that has been identified as available at t_x. For a service – $s_y \in \mathcal{S}^{t_x}$, let E^{s_y} be the system resource usage required by the service. If $\exists e_o^{s_y} \in E^{s_y}$ such that $ve_o^{s_y} > ve_o^{t_x}$, which indicates that the service – s_y requires higher usage value than the actual available resource value. Hence, the s_y is considered as unavailable at the timestamp – t_x.

Step 4. The ranking of candidate is based on the weight of resource and the resource availability.

Let $\mathcal{M}$ be a set of candidate Mist nodes where $\mathcal{M} = \{\mu_i : 1 \leq i \leq \mathbb{N}\}$. Each $\mu \in \mathcal{M}$ has a set of available hardware usage $\mathcal{A} = \{\alpha_j : 1 \leq j \leq \mathbb{N}\}$. Each α_j is defined as a tuple <*id, value*>, and $value_j^i$ denotes the value of available hardware usage α_j of Mist node μ_i. The score of a candidate Mist node $\mu_x \in \mathcal{M}$ is computed by equation (3):

$$score_x = \sum_{j \in |\mathcal{A}|} \left(\frac{value_j^x}{\sum_{i \in |\mathcal{M}|} value_i} \times w_j'^x \right) \tag{3}$$

where $w_j'^x$ denotes the normalised weight of the hardware usage α_j at μ_x, which is derived from equation (4):

$$w'^{x}_{j} = \frac{w^{x}_{j}}{\sum_{k \in |H^{x}|} \mathrm{wk}_{i}} \quad (4)$$

where w^{x}_{j} is the initial assigned weight value (see Example 1) and w_k is the weight of a $h_k \in H^x$. H^x is the H^{Glo} of $\mu_x \in \mathcal{M}$.

4. Evaluation

The proof-of-concept mePaaS prototype has been implemented on an Android OS mobile device (LG G4C). Following is the summary of the main components that have been implemented for the prototype:

- *Controller, LSMM, SP and schedule manager*. These components are the core elements of mePaaS. They have been implemented as one local service on Android OS.
- *Service modules* were implemented as independent local applications. These are managed by LSMM component in which LSMM dynamically launches them as plugin-based services for fulfilling the requests and they are automatically terminated when they are no longer needed. Since the service modules are a plugin-like software components, it is easy to extend the framework by installing more modules (Android application package). The current version of the prototype has seven service modules: Video, Image, GPS, Temperature, HTTP, MQTT and CoAP.
- *Program execution engine* is an extension of the Android-ported Activity BPM engine [http://activiti.org] derived from Dar *et al.* (2015). It can execute the program that has been modelled as BPMN with script language support. The details and performance testing of the process engine can be found in Dar *et al.*'s study (2015). Although the ported Activity BPM may not be the best option for other kind of IoT devices (e.g. Raspberry Pi), at this stage, it is sufficient for the proof-of-concept of mePaaS.

4.1 Self-configured service description

First, we have developed several service modules and installed with the mePaaS. Initially, in the idle state, the mobile device is not processing any request and local service module manager has published the SDM as given below:

```
{
"@context": "http://schema.org/geo","@id": "i:locaSensor1","@
type":
"i:LocationSensor",
"name": "Current Location of the Server", "url":
"http://172.19.28.237:8765/location",
"@context": "http://schema.org/temp","@id":"i:tempSensor1","@
type":"i:TemperatureSensor",
"name":"Current Ambient Temperature",
"url":"http://172.19.28.237:8765/temp",
"@context": "http://schema.org/image","@id": "i:camera1","@type":
"i:ImageSensor",
"name": "Image Sensing Service", "url":"http://172.19.28.237:8765/
image",
"@context": "http://schema.org/video","@id": "i:camera1","@type":
"i:ImageSensor",
"name": " Video Sensing Service ", "url":"http://172.19.28.237:8765/
video",
```

```
"@context": "http://schema.org/upload","@id": "i:upload","@type":
"i:DataUpload1",
"name": "Data Uploading Service", "url":"http://172.19.28.237:8765/
dataForwader"
}
```


As there are no resource-intensive services that are running, mePaaS published all service modules as available to clients. To confirm the dynamic nature of the SP, now we send a client request to the mePaaS node, which requests a periodic video sensing task that captures a 30-s video clip in every minute for 10-min duration and uploads to a distant server. This task requires using the camera as a hardware component and considerable amount of mobile internet bandwidth. Moreover, the controller coordinates with the LSMM, schedule manager and SP to update the SDM. In our previous prototype, the controller just removes the other services from the SDM; those also require the camera to provide the services. However, with the information from the schedule manager, this new prototype can reconfigure the SDM more effectively. Because the controller provides exact timeframes when the camera hardware is available in advance, the LSMM can inform the SP to reconfigure the SDM accordingly. Moreover, in our framework, the SP component temporally removed two services from the SDM that used the camera hardware as shown below, to avoid receiving another service request that also needs to access the camera at the same time slot.


```
{
"@context": "http://schema.org/geo","@id": "i:locaSensor1","@type":
"i:LocationSensor",
"name": "Current Location of the Server","url":
"http://172.19.28.237:8765/location",
"@context":
"http://schema.org/temp","@id":"i:tempSensor1","@type":"i:
TemperatureSensor",
"name":"Current Ambient Temperature", "url":"http://172.19.28.237:
8765/temp",
"@context": "http://schema.org/upload","@id": "i:upload","@type":
"i:DataUpload1",
"name": "Data Uploading Service","url":"http://172.19.28.237:8765/
dataForwader"
}
```


However, this is only for 30 s and thereafter the SP component reconfigures SDM that will make available video and image sensing until the next time slot.

4.2. Mist vs Fog vs distant data Centre

In this experiment, we compared the communication delay between three paths as cloud, fog and Mist. As shown in Figure 3, we deployed a temperature sensor, which connected to the Arduino board. The sensor advertises ambient temperature with the current time stamp in every second via the BLE communication protocol. In the first test case, mePaaS host collects temperature data in the proximity (just from the sensor via BLE), and provide to other clients as a service. Moreover, the client requests the current temperature information from the mePaaS host and it just forwards the temperature information, including the timestamp, which was recorded at the sensor node. At the client, we calculate the average time difference between the original and received timestamp for one thousand transactions.

Second, we measure the communication delay along the fog path. In this case, as the temperature sensor uses BLE as a local communication protocol, we need a proxy device that connects BLE to IP network. Here, we developed a simple Bluetooth application and installed it on a mobile phone that receives data over BLE and forwards to the IP network over the Wi-Fi link. Thereafter, the Fog node receives the temperature from the proxy device and makes it available to the clients within the local Wi-Fi network. Here, the Fog node is a laptop (HP EliteBook G3), which connected to a TP-Link (TL-WR940N) Wi-Fi router that sets up a local network. In this test case, the client connected to the local Wi-Fi network and received the ambient temperature over the established fog path. We measured the average delay same as that mentioned earlier. Finally, in the cloud path, the temperature sensor sends data to the proxy device, and then it moves to the cloud server over the local Wi-Fi network. Moreover, as the cloud server, we developed a Heroku Web app, which receives temperature data from the local sensor and published as a service to other clients. In this situation, the client connected to the cloud server via a mobile internet connection (TELE2 LTE connection-Estonia) with the average of 50 Mbps and 33.87 Mbps for download and upload speed, respectively. As of other test cases, we measured the average time delay for the cloud path also. As the amount of data is very small, note that we neglected the processing time at the respective node in all test cases.

According to Figure 4, the Mist path provides the least delay time (97.75 ms) for client requests, whereas the cloud path has highest communication delay (1287.28ms). Moreover, the fog path has the 719.08 ms delay because the sensor data travel through the proxy device and the fog server too.

4.3 Dynamic service module execution

As mentioned earlier, as the mePaaS node launches service modules on demand, we performed the bootstrapping test for each implemented service module. Moreover, we were concerned that the bootstrapping process of service modules can influence the overall performance and also add extra cost (e.g. energy, which is important if the Mist node is running in uncharged mode). Hence, we measured the latency and the energy consumption, which are caused by bootstrapping the mePaaS service modules (Figure 5).

First, we performed a test to get the average boot-up time for each module. According to Figure 5(a), MQTT and CoAP service modules have explicit latency due to establishing the underlying protocol stack. Also, here the HTTP server is based on AndroidAsync [https://github.com/koush/AndroidAsync] that provides lightweight Web server running on Android. However, we can see a little bit higher delay in the temperature-sensing module. The reason is that there is no inbuilt temperature sensor; the module should fetch the data

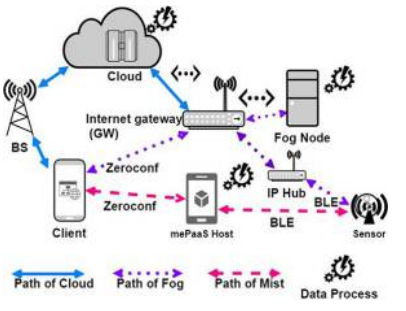

Figure 3.
Experiment setup: Mist vs Fog vs distant data centre

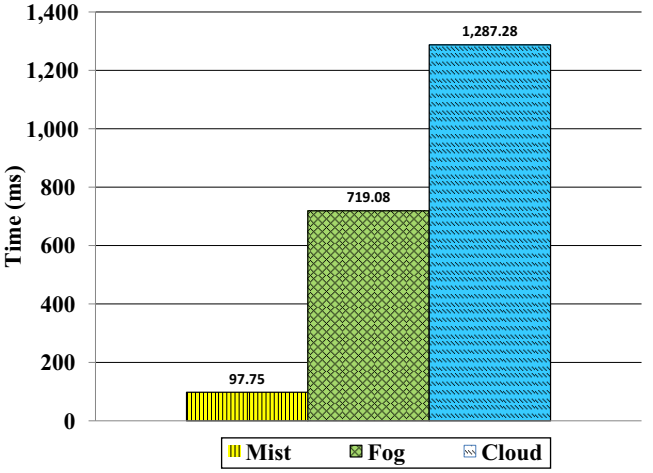

Figure 4. Communication latency

from the proximity sensor. First, the module should establish a BLE connection with the proximity sensor device that caused a few milliseconds delay.

Second, we performed a test to get the average power consumption of modules during the bootstrapping. According to the test results [Figure 5(b)] that were measured by PeakTeck 3430 Digital Multimeter, the MQTT module consumes the highest power (212.2 mA), whereas CoAP and the temperature-sensing modules consume a reasonable amount of power during the bootstrapping.

The test results indicate the need for improving the bootstrapping process in mePaaS, especially for reducing the bootstrapping time. As an alternative, we discovered that the Node.JS-based framework can reduce the burden of the bootstrapping process up to a certain level. However, most of the Android compatible modules are still in the early stage and could not just integrate into the current mePaaS framework.

4.4 Process substitution performance

As described in Section 3.4, mePaaS node can use nearby nodes for process substitution by selecting the best node for offloading the process. In this selecting process, there are two factors that would be affected by the performance of the offloading process.

First, the decision-making algorithm, which is based on comparing the performance and service availability of each candidate node. When the numbers of potential candidates are increasing, the algorithm should evaluate SDM from all candidates that will take the longer latency in the decision-making.

Second, as the decision-making requires the SDM of each candidate, the performance is also influenced by the SDM retrieval. However, this may be purely related to the fundamental wireless network protocol speed. As the upcoming IEEE 802.11ax reaches 10 Gbps speed, this concern may be solved by the underlying hardware.

Hence, the time for the overall process can be measured based on:

$$\mathcal{T} = max\left\{\mathcal{T}_m^{getSDM}\right\} + \sum_{m\in|\mathcal{D}|} \mathcal{T}_m^{readSDM} + \mathcal{T}_m^{runAlg} \tag{5}$$

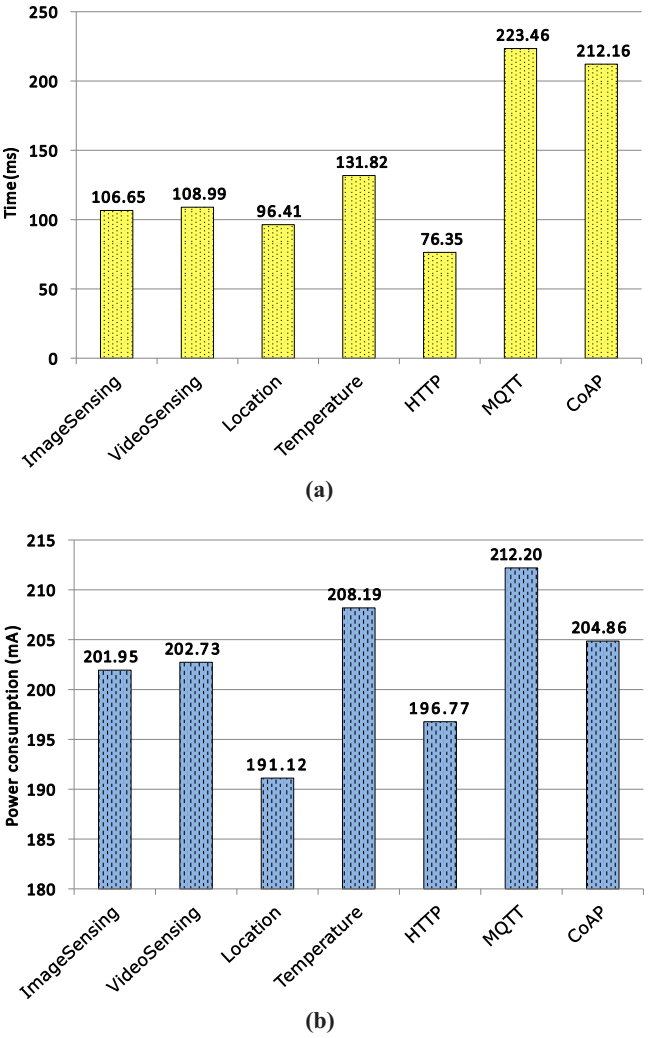

Figure 5.
Service module bootstrapping cost and performance

Notes: (a) Time consumption for bootstrapping; (b) average power consumption for bootstrapping

where:

m denotes one candidate for offloading. The environment has a set of m, which may be denoted by M.

$\mathcal{T}_m^{getSDM}$ is the time consumed for retrieving candidate m's SDM asynchronously.

$\mathcal{D}$ is a set of retrieved SDM in local memory.

$\mathcal{T}_m^{readSDM}$ is the time consumed for reading m's SDM in local memory.

$\mathcal{T}_m^{runAlg}$ is the time consumed for applying m in the candidate selection algorithm.

Figure 6 shows the average time consumption for identifying the best offloading node from many potential candidates, which include both reading SDM and applying the parameters from SDM to the matchmaking algorithm. It is clearly indicated that when the numbers of candidates are increasing, the latency is also highly increasing due to reading all SDMs. However, the processing time does not explicitly cause much latency, which indicates that today's smartphones are quite capable of performing computational tasks.

4.5. Cost and performance testing

In this experiment, we compared the performance of the process distribution using three cases as:

(1) *Case 1*: The Mist node owner is not using the device while the device is performing the program from the requester that involves a video sensing task. According to the request, the mePaaS node should record a video for 10 s and split it into two video files and upload to a distant server. This case is mentioned as "Normal" in Figure 7.

(2) *Case 2*: In this case, the Mist node owner is using the device (e.g. playing a game) while the device is executing the program from the requester. Here, we performed our own CPU-intensive task (average CPU usage about 46%) and at the same time do the video-sensing task that was requested by the client. This case is denoted by "In use, not offload" in Figure 7.

(3) *Case 3*: The Mist node owner is performing the same task and due to the lower resources (high CPU usage), the mePaaS has distributed the process to another Mist node (here we use LG Nexus 5 smartphone). This case is mentioned as "In use, do offload" in Figure 7.

Figure 7(a) shows the time comparison of the three cases. As the figure shows, video sensing and uploading task consumes 157.89 s, which includes the time for recording, splitting a video into two files and uploading. Here, we observed that still the Android system takes more time for video processing than other alternatives, such as Open-CV. Moreover, still offloading consumed considerable amount of time as the Android Wi-Fi connection between the two devices was not quite fast.

Figure 7(b) shows the CPU consumption comparison among the three cases. Initially, it consumes about 18 per cent of the CPU for the given video-sensing task. Next time, without offloading, the CPU usage can go over 72 per cent for both the processes. Also, we observed that sometimes the Android system kills the sensing task due to the high CPU use. However, once the mePaaS node made a decision to offload the process to the proximity Mist node, the figure shows that the CPU usage has reduced to 51 per cent that includes the video recording and offloading.

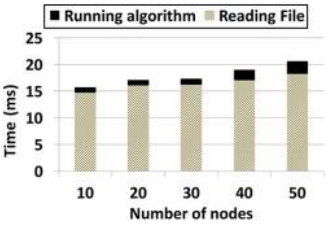

Figure 6.
Average time consumption for identifying offloading node

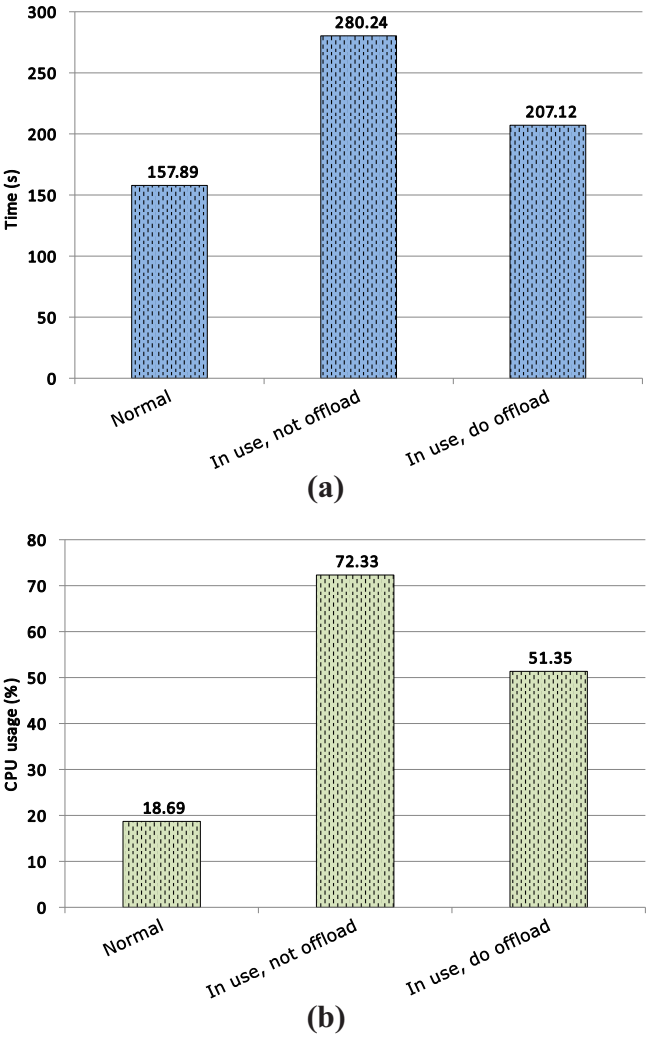

Figure 7.
Process offloading testing results

Notes: (a) Average time consumption comparison; (b) average CPU usage comparison

5. Conclusion and future work

In this paper, we have presented a mePaaS framework. The framework follows the service-oriented ESB architecture, which adapts the mechanism supported by the device (e.g. Wi-Fi communication, Bluetooth communication and GPS) into service. We extended our original work by adding service scheduling and the management of conflicts to improve the quality of SP. With the combination of the service scheduling and script language supported BPMN workflow engine as the program execution engine, mePaaS

allows the mobile device to provide a flexible platform for proximal users to offload their computational or networking program to mePaaS-based Mist computing node. The prototype of mePaaS has been implemented on a real-world mobile device together with performance evaluation. The evaluation results have shown the promising nature of mePaaS.

In the future, we will work on improving the performance of the bootstrapping of service modules. We have already identified the suitability of Node.js base modules and plan to integrate them into the mePaaS in future.

References

Bonomi, F., Milito, R., Zhu, J. and Addepalli, S. (2012), "Fog computing and its role in the internet of things", *Proceedings of the first edition of the MCC workshop on Mobile cloud Computing*, pp. 13-16.

Chang, C., Ling, S. and Srirama, S. (2014), "Trustworthy service discovery for mobile social network in proximity", PerCom '14 Workshops. pp. 478-483.

Chang, C., Srirama, S.N. and Buyya, R. (2017a), "Indie fog: an efficient fog-Computing infrastructure for the internet of things", *Computer*, Vol. 50 No. 9, pp. 92-98.

Chang, C., Srirama, S.N. and Buyya, R. (2017b), "Mobile cloud business process management system for the internet of things: a survey", *ACM Computing Surveys (Surveys)*, Vol. 49 No. 4, p. 70.

Chang, C., Srirama, S.N. and Mass, J. (2015), "A middleware for discovering proximity-based service-oriented industrial internet of things", *Proceedings of the IEEE International Conference on Services Computing (SCC)*, pp. 130-137.

Chu, D.C. and Humphrey, M. (2004), "Mobile ogsi.net: grid computing on mobile devices", I5th IEEE/ACM International Workshop on Grid Computing. pp. 182-191.

Dar, K., Taherkordi, A., Baraki, H., Eliassen, F. and Geihs, K. (2015), "A resource oriented integration architecture for the Internet of things: a business process perspective", *Pervasive and Mobile Computing*, Vol. 20, pp. 145-159.

der Aalst, W.M.P. (1998), "The application of petri nets to workflow management", *Journal of Circuits, Systems, and Computers*, Vol. 8 No. 1, pp. 21-66.

Fernando, N., Loke, S.W. and Rahayu, W. (2012), "Honeybee: a programming framework for mobile crowd computing", MobiQuitous '12, *Springer, New York*, pp. 224-236.

Fernando, N., Loke, S.W. and Rahayu, W. (2016), "Computing with nearby mobile devices: a work sharing algorithm for mobile edge-Clouds", *Transactions on Cloud Computing* No. 99, pp. 1-14.

Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013), "Internet of things (IoT): a vision, architectural elements, and future directions", *Future Generation Computer Systems*, Vol. 29 No. 7, pp. 1645-1660.

Jararweh, Y., Doulat, A., AlQudah, O., Ahmed, E., Al-Ayyoub, M. and Benkhelifa, E. (2016), "The future of mobile cloud computing: integrating cloudlets and mobile edge computing", Telecommunications (ICT), 23rd International Conference on, pp. 1-5.

Kondo, D., Javadi, B., Malecot, P., Cappello, F. and Anderson, D.P. (2009), "Cost-benefit analysis of cloud computing versus desktop grids", Parallel & Distributed Processing, IPDPS, IEEE International Symposium on, pp. 1-12.

Lin, X., Andrews, J., Ghosh, A. and Ratasuk, R. (2014), "An overview of 3GPP device-to-device proximity services", *IEEE Communications Magazine*, Vol. 52 No. 4, pp. 40-48.

Liyanage, M., Chang, C. and Srirama, S.N. (2015), "Lightweight mobile web service provisioning for sensor mediation", *Mobile Services (MS), IEEE International Conference on*. pp. 57-64.

Liyanage, M., Chang, C. and Srirama, S.N. (2016), "mePaaS: mobile-embedded platform as a service for distributing fog computing to edge nodes", 17th international conference on parallel and distributed computing, applications and technologies (PDCAT-16).

Loke, S.W., Napier, K., Alali, A., Fernando, N. and Rahayu, W. (2015), "Mobile computations with surrounding devices: proximity sensing and MultiLayered work stealing", *ACM Transactions on Embedded Computing Systems*, Vol. 14 No. 2, pp. 22:1-22:25.

Marinelli, E.E. (2009), "Hyrax: cloud computing on mobile devices using MapReduce".

Mohamed, K. and Wijesekera, D. (2012), "A lightweight framework for web services implementations on mobile devices", Mobile Services (MS), IEEE First International Conference on, pp. 64-71.

Murray, D.G., Yoneki, E., Crowcroft, J. and Hand, S. (2010), "The case for crowd computing", 2nd ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds. pp. 39-44.

Nicoloudis, N. and Pratistha, D. (2003), ".net compact framework mobile web server architecture".

Nunes, B.A.A., Mendonca, M., Nguyen, X.N., Obraczka, K. and Turletti, T. (2014), "A survey of software-defined networking: past, present, and future of programmable networks", *IEEE Communications Surveys & Tutorials*, Vol. 16 No. 3, pp. 1617-1634.

Orsini, G., Bade, D. and Lamersdorf, W. (2015), "Computing at the mobile edge: designing elastic android applications for computation offloading", IFIP Wireless and Mobile Networking Conference (WMNC), 8th, pp. 112-119.

Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S. and Neal, A. (2014). "Mobile-edge computing introductory technical white paper", White Paper, Mobile-edge Computing (MEC) industry initiative.

Penco, C. (1999), "Objective and cognitive context", *Modeling and Using Context*, Springer, New York, NY, pp. 270-283.

Preden, J.S., Tammemäe, K., Jantsch, A., Leier, M., Riid, A. and Calis, E. (2015), "The benefits of self-awareness And attention in fog and mist computing", *Computer*, Vol. 48 No. 7, pp. 37-45.

Pulli, P., Martikainen, O., Zhang, Y., Naumov, V., Asghar, Z. and Pitkänen, A. (2011), "Augmented processes: a case study in healthcare", ISABEL '11, p. 137.

Qin, X., Jiang, H., Zhu, Y. and Swanson, D.R. (2003), "Dynamic load balancing for I/O-intensive tasks on heterogeneous clusters", High Performance Computing-HiPC, *Springer*, New York, NY, pp. 300-309.

Robinson, R. (2004), "Understand enterprise service bus scenarios and solutions in service-oriented architecture, part 1: the role of the enterprise service bus".

Satyanarayanan, M., Bahl, V., Caceres, R. and Davies, N. (2009), "The case for vm-based cloudlets in mobile computing", *IEEE Pervasive Computing*, Vol. 8 No. 4.

Skarlat, O., Schulte, S., Borkowski, M. and Leitner, P. (2016), "Resource provisioning for IoT services in the Fog", Service-Oriented Computing and Applications (SOCA), IEEE 9th International Conference on, pp. 32-39.

Srirama, S.N., Jarke, M. and Prinz, W. (2006a), "Mobile host: a feasibility analysis of mobile web service provisioning", UMICS.

Srirama, S.N., Jarke, M. and Prinz, W. (2006b), "Mobile web service provisioning", Telecommunications, AICT-ICIW'06. International Conference on Internet and Web Applications and Services/ Advanced International Conference on, p. 120.

Stantchev, V., Barnawi, A., Ghulam, S., Schubert, J. and Tamm, G. (2015), "Smart items, fog and cloud computing as enablers of servitization in healthcare", *Sensors & Transducers*, Vol. 185 No. 2, p. 121.

Su, X., Riekki, J., Nurminen, J.K., Nieminen, J. and Koskimies, M. (2015), "Adding semantics to internet of things", *Concurrency and Computation: Practice and Experience*, Vol. 27 No. 8, pp. 1844-1860.

Taleb, T., Dutta, S., Ksentini, A., Iqbal, M. and Flinck, H. (2017), "Mobile edge computing potential in making cities smarter", *IEEE Communications Magazine*, Vol. 55 No. 3, pp. 38-43.

Wang, L., Zhang, D. and Xiong, H. (2013), "effSense: energy-efficient and cost-effective data uploading in mobile crowdsensing", *Proceedings of the ACM conference on Pervasive and Ubiquitous Computing Adjunct Publication*, pp. 1075-1086.

Corresponding author
Mohan Liyanage can be contacted at: ldmohan@yahoo.com