2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies

mePaaS: Mobile-Embedded Platform as a Service for Distributing Fog Computing to Edge Nodes

Mohan Liyanage
Institute of Computer Science
University of Tartu, Estonia
liyanage@ut.ee

Chii Chang
Institute of Computer Science
University of Tartu, Estonia
chii.chang@acm.org

Satish Narayana Srirama
Institute of Computer Science
University of Tartu, Estonia
srirama@ut.ee

***Abstract*—The distant data centre-centric Internet of Things systems face the latency issue especially in the real-time-based applications. Recently, Fog Computing models have been introduced to overcome the latency issue by utilising the proximity-based computational resources. However, the increasing users of Fog Computing servers will cause bottleneck issues and consequently the latency issue arises again. This paper introduces the utilisation of Mist Computing (Mist) model, which exploits the computational and networking resources from the devices at the very edge of IoT networks. The proposed service-oriented mobile-embedded Platform as a Service framework enables the edge IoT devices to provide a platform that allows requesters to deploy and execute their own program models. The 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. Additionally, the framework also supports task distribution among a group of Mist nodes. The prototype has been tested and performance evaluated on the real world devices.**

I. Introduction

The information systems designed for integrating the Internet of Things (IoT) [1] are usually applying the global centralised model, in which the IoT devices rely on the distant management systems. Such a model is considered as a drawback in terms of agility [2], [3]. In many real-time ubiquitous applications (e.g., augmented reality, environmental analytics, ambient assisted living etc.), mobile device users require rapid responses. However, the latency caused by the distant centralised model is too high, even though the mobile Internet speed has improved a lot during the last few years. To address this problem, Fog Computing (Fog) [2] introduces offloading computational process to the computers in proximity. Such computers can be co-located with cellular base station [3], nearby Grid Routers, or personal computers that have implemented the compatible functions (e.g. Cloudlets [4]). The basic idea is to provide a sandbox environment for requesters to offload computational processes or to perform rapid processing on the data collected from the sensors in local area in order to achieve fast responses, instead of relying on the remote data centre to perform all the processes.

The concept of Fog is promising. However, it will increase the amount of data transmission in local networks and also consumes a lot of computational resources of the Fog nodes. Unless the companies are willing to deploy a large number of Fog nodes in the area, soon Fog nodes will face bottleneck issues and the latency issue will still arise [5].

In order to reduce the burden of Fog, researchers [5]–[7] have introduced Mist Computing model (Mist). In Mist, the computing moves to the extreme edge of the IoT environment, where 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 [5]–[7].

The Mist covers various types of devices such as Raspberry Pi (https://www.raspberrypi.org), Contiki (http://www.contiki-os.org) devices, TinyOS [8] devices, Android OS devices and heterogeneous mobile devices [6] such as smartphone, tablets, handheld entertainment devices etc. The participative device of Mist, in this paper, is called a Mist node. Mist nodes are different to the regular mobile Web servers (MWS) [9]–[11] that provide static software services to their clients. For example, a MWS can simply provide its current location as a Web service via the 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, that have predictable accessibility, 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."*

The idea of Mist may not be new. A similar research trend, which is termed—Mobile Ad hoc Grid Computing (MAGC) in this paper, has been studied for many years. MAGC is utilising the high density ARM CPU-powered mobile devices for computational process distribution. The motivation derives from the prediction of Gartner research in which smartphones, tablets and PCs will reach over 7.3 billion units globally in 2020 [12], and there will be 6.1 billion smartphones users globally [13]. Since many of the mobile devices are not in regular use (e.g. getting charged with AC power) and their accessibility is predictable based on their users (e.g. the user is working continuously in a shop for 4 hours), they can be utilised for grid computing purpose.

MAGC has been applied in several works [14]–[17] during

978-1-5090-5081-9/16 $31.00 © 2016 IEEE
DOI 10.1109/PDCAT.2016.29

the past years. 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 a specific mobile application or a standalone platform will be installed so that the process can be distributed among mobile devices. Usually, the proposals focus on one particular use case (e.g. effSense [18] 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.

1) In real world, the less flexible OS-based devices (e.g. Android OS, iOS) have been applied in various domains such as mobile phones, tablets, home appliances, wearable devices etc. 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 for Mist computing. There is a need for a more generic model for enabling the less flexible OS devices to participate in Mist.
2) 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 context factors.

To address the issues, we are proposing "**m**obile-**e**mbedded **P**latform **a**s **a** **S**ervice (**mePaaS**)". As the technology evolved, today's ARM CPU-powered mobile devices can outperform entry-level virtual machines [19], [20], 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 provides an overview on the recent technologies towards proposing a generic mePaaS software framework for Mist Computing. The framework aims to address the two issues described before: (1) a flexible program execution environment in mobile device; (2) self-adaptive resource management. A prototype has been developed based on mePaaS, and has been thoroughly tested, as a proof-of-concept.

The paper is organised as follows. Section 2 describes the architecture of the proposed framework, the self-configured service provisioning scheme and the scheme for extending the computational resources of the Mist node. Section 3 describes the evaluation of the prototype. Section 4 summarises the related works. Finally, the paper is concluded in Section 5 together with future research plans.

II. Proposed Framework

A. Architecture Overview

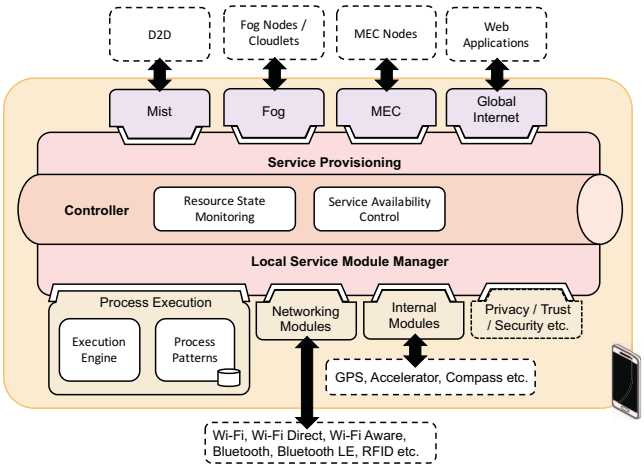


Figure 1: mePaaS architecture

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) [21] architecture. ESB is a software infrastructure that can easily connect resources by combining and assembling services to achieve a loosely-coupled Service-Oriented Architecture (SOA). mePaaS utilises 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 standard workflow model (e.g. BPMN [http://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 1 illustrates the main elements of mePaaS framework. The main elements are described as below.

Controller—is the core element of mePaaS. It mediates and manages the computational and networking modules to the service provisioning component. It contains two important functions:

- *Resource state monitoring*, which continuously monitors the states of the device hardware (e.g. CPU load percentage, RAM usage, incoming and outgoing network transmission status etc.). Controller can autonomously make decision about which service modules are available based on the required usage of resources. The decision making scheme is described in next section.

- *Service availability control*. As mentioned, mePaaS enables customisable program method to be executed based on workflow execution associated with invoking the local service modules. Since each service module execution consumes certain amount of hardware resources of the device, if one of the hardware resources is not available due to the overhead of usage or due to the faulty hardware, Controller will notify the Service Provisioning component to disable the corresponding service provisioning mechanism.

Service Provisioning (SP)—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 Web of Things (WoT; https://www.w3.org/WoT/), in which SDM is described based on JSON-LD (https://www.w3.org/TR/json-ld). SP associates with different Service Provisioning Adaptors (SPA) such as Mist, Fog, MEC, Web to publish or advertise mePaaS's SDM. Also, the SPA are responsible to handle the incoming request packages from their field. For example, Mist SPA are responsible to advertise SDM in 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, D2D utilises existing infrastructure for the communication within 1-hop range [22].

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 utilise 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, user can install additional module for performing semantic sensory data reasoning. LSMM will inform Controller the new installed module together with the corresponding descriptions. Controller will pass the information to SP to include the module as a new service in SDM.

Process Execution—consists of two main elements:

- *Execution Engine*—is a software to execute the workflow included in the request package. The Execution Engine needs to associate with LSMM in order to invoke the corresponding service modules involved in the workflow tasks.
- *Process Patterns*—manages a number of predefined workflow patterns. The pre-defined patterns can be used to replace the inexcusable tasks in the workflow as substitutions. The details of the task substitution will be described in the later section.

Networking Modules—represents the service modules that can trigger 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 etc.). These modules can also be used to fetch the SDM of the other service providers in proximity.

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

Privacy, Trust, Security etc.—are the modules that involve the management of privacy and service level agreement, the trustworthiness and quality of service provisioning, cryptography and other security involved mechanisms. We have studied them earlier in other contexts such as service discovery [23], and they will be integrated with mePaaS in our future work.

B. Self-configured Service Provisioning

Service Description Metadata (SDM) provides information about the hardware specification of the Mist node, and also the available service modules supported by the Mist node. As mentioned previously, the hardware usage of mobile devices is quite dynamic, the activities performed on the device will influence the availability of the service provisioning because each service module requires a certain amount of hardware usage. For example, if the Mist node is currently serving for a continuous data streaming work, then it is unlikely to serve a new request that involves heavy bandwidth usage. Hence the corresponding service may need to be disabled from SDM since the Mist node cannot handle any more such kind of requests. In order to dynamically update the SDM, we propose the service scheduling scheme. First, we explain the terminologies used in the scheme.

As described earlier, Mist node-handled process is modelled as a workflow. Here, we refer to the terms described in [24] 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* 3.1.** (*Installed Service Module Collection*)—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, bandwidth intensive etc.) and the *availability status*.
- $\beta : S \rightarrow \mathcal{U}$ is a function that maps the service modules to the required hardware usages.

***Definition* 3.2.** (*Scheduled Work Items*)—is defined as a tuple $< \mathcal{W}, F, \zeta, \gamma, \delta >$ where:

- $\mathcal{W}$ is a finite set of work items. $\mathcal{W} = \{\omega_1, \omega_2, ..., \omega_n\}$, $n \in \mathbb{N}$.
- $F \subseteq \mathcal{W} \times \mathcal{W}$ is a set of flow relation rules.
- $\zeta : \mathcal{W} \rightarrow Z$ is a function that maps work items to status.
- $\gamma : \mathcal{W} \rightarrow \Gamma$ is a function that maps work items to start times.
- $\delta : \mathcal{W} \rightarrow D$ is a function that maps work items to estimated execution duration.

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

***Definition* 3.3.** (*Device Hardware Usages* (H^{Glo})). $H^{Glo} = \{h_1^{Glo}, h_2^{Glo}, ..., h_n^{Glo}\}, n \in \mathbb{N}$. 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} denote the current assigned usage of the hardware based on the scheulded work items.
– xu^{Glo} denotes the maximum availability of the hardware.

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

Algorithm 1 Identifying Unavailable Service Modules

```
1:  removeList = newcollection
2:  for s ∈ S do
3:      hardwareUsageSet ← β(s)
4:      for hus ∈ hardwareUsageSet do
5:          for h^Glo ∈ H^Glo do
6:              if ID_hus ≡ ID_{h^Glo} ∧ (value_hus + cu_h^Glo) > xu_h^Glo)
                then
7:                  add s to removeList
8:              end if
9:          end for
10:     end for
11: end for
```

However, Algorithm 1 does not consider that the hardware usage may be released in a short time. Therefore, we add an additional information for the service description known as "delay allowance time ($daTime$)", which corresponds that the service may not be executable immediately but is expected to be available after a period of time. A customised value can be set for the maximum delay allowance. If the service module cannot be executed within the delay allowance, it will be confirmed for removal from SDM. Algorithm 2 is the decision making algorithm performed after the previous Algorithm 1 to ensure the unnecessary removing of service.

In Algorithm 2, an additional parameter—"expected start time ($expecST$)" was introduced. This is the value that needs to be calculated based on the dependency involved. For example, if the work item-A is scheduled to be started after work item-B, then the $expecST$ of work item-A will be the $endTime$ of work item-B.

C. 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. Since such information is available, a Mist node can form a grid computing group centred by itself with the other Mist nodes that are within 1-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 will generate a more efficient result. However, it raises a question about how does mePaaS makes the decision on which Mist node it should distribute the work to?

Algorithm 2 Delay-tolerant Service Availability

Require: $removeList$ from Algorithm 1
$daTime$ of each service module
Ensure: $currentTime \leftarrow currentsystemtime$
$expecST \leftarrow$ expected start time
$expecET \leftarrow$ expected end time

```
1:  finalRemoveList = newcollection
2:  for s ∈ removeList do
3:      maxTime = newTime
4:      for ω ∈ W do
5:          if status_ω ≡ "running" then
6:              endTime ← γ(ω) + δ(ω)
7:              DT ← daTime of s
8:              if (endTime − currentTime) > DT then
9:                  add s to finalRemoveList
10:             end if
11:         else if status_ω ≡ "pending" then
12:             expecST_ω ← max{expecET of ω ∈ •ω}
13:             endTime = expecST_ω + δ(ω)
14:             if (endTime − currentTime) > daTime then
15:                 add s to finalRemoveList
16:             end if
17:         end if
18:     end for
19: end for
```

Work distribution in mobile grid computing environment is an interesting research domain that has been studied for many years. A basic approach is the rule-based approach [25], in which the distribution decision is made by the static predefined rules based on certain criteria (e.g. computational power, speed of transmission). A more complex approach which was applied in Hyrax [15], utilises MapReduce framework to manage the works in the mobile grid. However, such an approach is inefficient since the heavyweight MapReduce framework consumes too much of hardware resources. The third approach was to apply the work stealing scheme [17], where the higher performance participative nodes are actively retrieving the works from the other nodes until all the works are completed.

The first approach is less robotic; the second approach may not yet be efficient for today's edge IoT devices due to the constrained resources; the third approach is suitable for the work that can be equally divided into multiple portions, and since the work portions are 'stolen' after they are assigned, it may take extra time for the work distribution.

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 3.1. 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 3.2. 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 in [26] and [27]:

(a) CPU intensive task (e.g. customised complex algorithm scripts).
(b) CPU+RAM intensive task (e.g. I/O data processing; large data volume loading involved tasks)
(c) Bandwidth intensive task (e.g. data forwarding process; e.g. send/receive tasks)
(d) Hybrid, where multiple resources have high weight.

Example 1: If (b) is the classification of the Work Item, then for example, hardware parameters being considered are CPU, RAM, Bandwidth, then mark CPU = 2, RAM = 2, Bandwidth = 1, corresponding to CPU and RAM both plus one. Hence, the weight of CPU will be 2/5, RAM will be 2/5, Bandwidth will be 1/5.

STEP 3.3. 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 (1).

$$score_x = \sum_{j \in |\mathcal{A}|} \left(\frac{value_j^x}{\sum_{i \in |\mathcal{M}|} value_j^i} \times w'^x_j\right) \quad (1)$$

where w'^x_j denotes the normalised weight of the hardware usage α_j at μ_x, which is derived from (2).

$$w'^x_j = \frac{w_j^x}{\sum_{k \in |H^x|} w_k} \quad (2)$$

where w_j^x 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 $\mu_x \in \mathcal{M}$.

III. 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, Service Provisioning**. These three components are the core elements of mePaaS. They have been implemented as one local service of Android OS.
- **Service modules**. The service modules were implemented as independent local applications. They are managed by *LSMM* component in which *LSMM* can launch them as plugin-based services dynamically for fulfilling the requests and they are automatically terminated when they are no longer needed. Since the service modules are plugin-like software component, it is easy to be extended by installing more modules (Android application package). Current version of prototype has five service modules: GPS, Temperature, HTTP, MQTT and CoAP.
- **Program Execution Engine**. In the current version of prototype, the program execution engine is an extension of the Android-ported Activiti BPM engine (http://activiti.org) derived from [28]. It can execute the program that has been modelled as BPMN with scrip language support. The details and performance testing of the process engine can be found in [28]. Although the ported Activiti 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. We have also implemented a workflow execution engine prototype for Raspberry Pi with two different BPMN model execution approaches (full version of Camunda (https://camunda.com) BPM and a lightweight *workflow code execution engine* that executes the program code translated from BPMN). The work has been published in [29]. We are considering to integrate them in our future work.

A. Self-configured Service Description

We have tested the *self-configured service provisioning scheme* by executing a resource intensive application on the device while mePaaS is operating in the background. Since the resource intensive application makes the hardware resource required for certain service modules insufficient, SP has re-configured the SDM automatically.

Below shows an example of the generated SDM (simplified) from SP component :

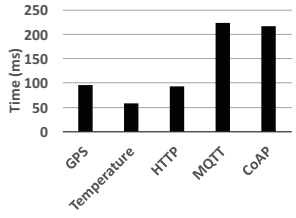
```
{"@context":{ "geo": "http://schema.org/geo",
"latitude":{"@id":"http://schema.org/latitude",
"@type": "xsd:float" }, "longitude": { "@id":
"http://schema.org/2001/XMLSchema#" }, "name":
"Current Location of the Server", "url":
"http://172.19.28.237:8765/location", "@context":{
"geo": "http://schema.org/dataupload","data":
{"@id": "http://schema.org/***","@type":
"xsd:bytes" }, "xsd":
"http://www.w3.org/2001/XMLSchema#" }, "name":
"DataUploading Service", "url":
"http://172.19.28.237:8765/dataForwader"}
```

B. Dynamic Service Module Execution

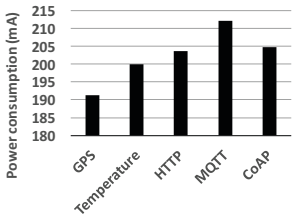
As described previously, service modules are launched on-demand. So, we were concerned that the bootstrapping process of service modules can influence the overall performance and also add extra cost (e.g. energy, which is important if the Mist node is running in uncharged mode). Hence, we performed the bootstrapping test of each implemented service module.

First, we observed the latency caused by bootstrapping the service module. Figure 2a shows the average time consumption of each service module. As the figure shows, MQTT and CoAP service modules can cause explicit latency.

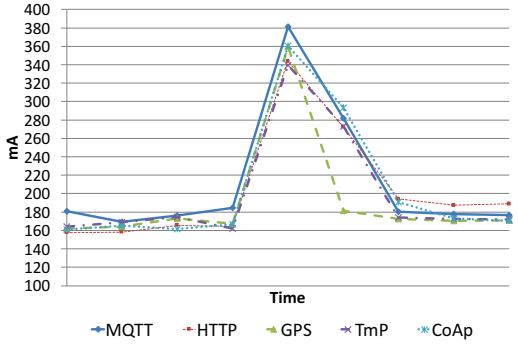
Commonly, the bootstrapping of Android software can consume extra power. The corresponding test results are shown



(a) Time consumption for bootstrapping.



(b) Average power consumption for bootstrapping.



(c) Power consumption for module bootstrapping.

Figure 2: Service module bootstrapping cost and performance.

in Figure 2b and Figure 2c, where Figure 2b shows the average power consumption (measured by PeakTeck 3430 Digital Multimeter) of each service module's bootstrapping, and Figure 2c illustrates the power consumption at the bootstrapping time.

The test results indicate the need for improving the bootstrapping process in mePaaS, especially for reducing the bootstrapping time.

C. Process Substitution Performance

Section II-C described the use of other Mist nodes for process substitution based on selecting the best node for offloading the process. We concern that the efficiency of the approach is influenced by two subjects.

- First, the decision making algorithm, which is based on comparing the performance and service availability of each candidate node, which is influenced by the number of potential candidates in the area. The more candidates exist, the longer latency of the decision making can take.
- Second, since the decision making requires the SDM of each candidate, the SDM retrieval time will highly influence the time of the process.

The second concern may be purely related to the fundamental wireless network protocol speed. Since the upcoming IEEE 802.11ax reaches 10 Gbps speed, this concern may be solved by the underline hardware. Hence, the time for the overall process can be measured based on:

$$\mathcal{T} = max\{\mathcal{T}_m^{getSDM}\} + \sum_{m \in |\mathcal{D}|} \mathcal{T}_m^{readSDM} + \mathcal{T}_m^{runAlg} \quad (3)$$

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.

The first concern is influenced by performing the matchmaking algorithm, which may be influenced a lot while the number of candidates are increasing. Figure 3 shows the testing result for the first concern. As the result shows, when the number of candidates increased, the more SDMs need to be processed, which involves both reading SDM and applying the parameters from SDM to the matchmaking algorithm, and hence the latency is increased. However, the processing time does not explicitly cause much latency, which indicates that today's smartphones are quite capable of performing computational tasks.

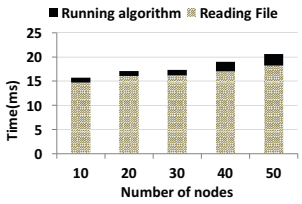


Figure 3: Average time consumption for identifying offloading node

D. Cost and Performance Testing

Here, we summarise the process distribution performance using the comparison of three cases below:

Case 1: the Mist node owner is not using the device while the device is executing the program from requester. This case is denoted by 'Normal' in Figure 4.

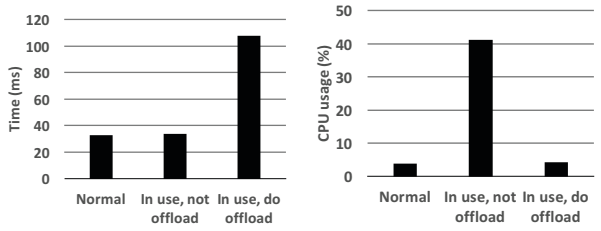
Case 2: the Mist node owner is using the device while the device is executing the program from requester. This case is denoted by 'In use, not offload' in Figure 4.

Case 3: the Mist node owner is using the device and mePaaS has distributed the process to another Mist node. This case is denoted by 'In use, do offload' in Figure 4.

The setting of this testing is as below:

- The process is a program that reads 50 SDM and performs the semantic matchmaking algorithm.
- The application being run by the device owner is an Android game which consumes around 35%—40% CPU usage.
- The offloading node is also a LG G4C smartphone.

Figure 4a shows the time comparison of the three cases. As the figure shows, offloading consumed the longest time. It is because the Wi-Fi speed between the two devices was not quite fast. Figure 4b shows the CPU consumption comparison among the three cases. As the figure shows, without offloading the process, the CPU usage can go over 40%. This only happens in the case of local computational process. Consider that if the process involves MQTT-based streaming, the device



(a) Average time consumption comparison. (b) Average CPU usage comparison.

Figure 4: Process offloading testing results.

may not be able to handle both Mist service provisioning and user's activity. Consequently, the device's operating system may crash.

E. Mist vs. Fog vs. Distant Data Centre

Finally, we performed a performance comparison between three cases: process offloading to *Mist*, *Fog* and *Cloud* (see Figure 5). The Fog node in this testing is a computer in 1-hop Wi-Fi range of the requester. The process used here is the same as the previous test case (i.e. semantic matchmaking process). The Cloud is a remote Web server, and the request was sent to the Cloud via Tele2 (Estonia) 4G mobile Internet with average 50Mbps for download speed and 33.87 Mbps for upload speed. The transmission shown in the Figure is the sum of request and response including server-side delay, and the size of the request package is around 1KB.

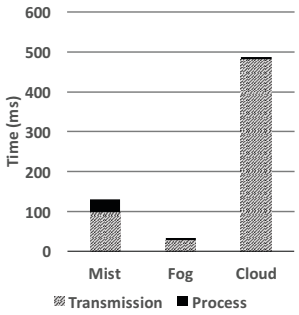


Figure 5: Average time consumption of three approaches

As commonly expected, distributing the process to Fog node results the best performance. Followed by the Mist node. The computational process at Cloud took very little time. However, the transmission latency is the major drawback of the Cloud. Note that the main objective of Mist is to reduce the burden of Fog. It is not meant to be a replacement of Fog.

IV. Related Work

A. Proximity-based Mobile Grid Computing

MapCloud [30] and MUSIC [31] are 2-tier cloud computing middlewares that manage the computational tasks, which are modelled in workflows, and allocate the work between the low latency local cloud computing nodes and the public cloud such as Amazon Web Services (AWS; https://aws.amazon.com) or Google App Engine (GAE; https://cloud.google.com/appengine). The contribution focuses on the scheme for efficient work allocation between computational nodes. The similar concept and model that utilises the proximity-based computational resources for distributing the process have been proposed in Mobile Edge Computing (MEC) [3], Fog Computing [2]. These proposals were focusing on introducing the architecture at abstract level. Cloudlet [4] is a framework that can be utilised to realise MEC and Fog Computing. The main concept is to host virtual machine on the computers of local business and provide the VM as a service to proximity-based users. Hyrax [15] 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 heavyweight. Honeybee [32] is a framework that aims the same purpose as Hyrax. However, instead of utilising the heavyweight MapReduce, Honeybee applied the active work stealing scheme in which participants in mobile grid computing environment will actively be taking the work (e.g. processing image files) from the other nodes who have more works to do. Other similar works can be found in the comparison from [33].

Different to 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 sharable 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.

B. Mobile Service Provisioning

The provisioning of Web-service oriented server on mobile devices has already been studied from the early 2000s. For examples, micro Web server that hosts SOAP Web server on Windows CE [9] and SOAP Web server from Symbian OS [10]. Today, the trend of mobile Web servers and provisioning has shifted from SOAP to RESTful service provisioning [11], [34]. The service description, which originally followed W3C's WSDL, WADL, is slowly moving to RESTful service description such as JSON-LD, which combines service/resource description and also semantic description.

Today, various embedded servers already exist and are being used in small devices for Wireless Sensor & Actuators and IoT. They follow standards such as CoAP and MQTT for constrained devices to provision services. Service provisioning from constrained devices is no longer a challenge. Moreover, as modern smartphones can outperform the low-end IaaS Cloud instances [19], [20] 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 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 was not the focus of MCrC prototype-Honeybee [32]. mePaaS framework can be seen as the complement of MCrC.

V. Conclusion and Future Work

In this paper, we proposed the **m**obile-**e**mbedded **P**latform **a**s **a** **S**ervice (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, GPS etc.) into service. With the combination of the 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 real world mobile device together with performance evaluation. The evaluation results have shown the promising nature of mePaaS. However, it still needs optimisation to improve the quality of service provisioning.

In future, we plan to address the following aspects:

- Optimise the bootstrapping performance of service modules. As the evaluation result have shown, certain service modules took significant time at the bootstrapping phase, which requires a lot of improvement.
- The dynamic service module management scheme described in this paper may not be sufficient to overcome the more complex situations. We plan to integrate the scheduling scheme to improve the dynamic service module management mechanism.

Acknowledgment

This research is supported by Estonian Research Council grant PUT360.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Elsevier FGCS Journal*, vol. 29, no. 7, pp. 1645 – 1660, 2013.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[3] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," 2014.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[5] J. S. Preden, K. Tammemae, A. Jantsch, M. Leier, A. Riid, and E. Calis, "The Benefits of Self-Awareness and Attention in Fog and Mist Computing," *Computer*, vol. 48, no. 7, pp. 37–45, 2015.

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

[7] M. J. Martin, "Cloud, Fog, and now, Mist Computing," December 2015, [Online] https://www.linkedin.com/pulse/cloud-computing-fog-now-mist-martin-ma-mba-med-gdm-scpm-pmp; [accessed 6 June 2016].

[8] P. Levis, S. Madden, J. Polastre *et al.*, "Tinyos: An operating system for sensor networks," in *Ambient intelligence*. Springer, 2005, pp. 115–148.

[9] N. Nicoloudis and D. Pratistha, ".net compact framework mobile web server architecture," https://msdn.microsoft.com/en-us/library/aa446537.aspx, July 2003, [last accessed] 20 May, 2016.

[10] S. N. Srirama, M. Jarke, and W. Prinz, "Mobile web service provisioning," in *ICIW '06*. IEEE, 2006, p. 120.

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

[12] J. Rivera and R. van der Meulen, "Gartner says the internet of things installed base will grow to 26 billion units by 2020," 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2636073

[13] I. Lunden, "6.1 b smartphone users globally by 2020, overtaking basic fixed phone subscriptions," 2015.

[14] D. C. Chu and M. Humphrey, "Mobile ogsi. net: Grid computing on mobile devices," in *5th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, 2004, pp. 182–191.

[15] E. E. Marinelli, "Hyrax: cloud computing on mobile devices using mapreduce," DTIC Document, Tech. Rep., 2009.

[16] D. G. Murray, E. Yoneki, J. Crowcroft, and S. Hand, "The case for crowd computing," in *2nd ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*. ACM, 2010, pp. 39–44.

[17] S. W. Loke, K. Napier, A. Alali, N. Fernando, and W. Rahayu, "Mobile computations with surrounding devices: Proximity sensing and multilayered work stealing," *ACM TECS*, vol. 14, no. 2, p. 22, 2015.

[18] L. Wang, D. Zhang, and H. Xiong, "effsense: energy-efficient and cost-effective data uploading in mobile crowdsensing," in *UbiComp '13*. ACM, 2013, pp. 1075–1086.

[19] H. Flores, S. N. Srirama, and R. Buyya, "Computational offloading or data binding? bridging the cloud infrastructure to the proximity of the mobile user," in *Mobile Cloud, 2nd IEEE Int. Conf. on*, 2014, pp. 10–18.

[20] C. Chang, S. N. Srirama, and J. Mass, "A Middleware for Discovering Proximity-Based Service-Oriented Industrial Internet of Things," in *Services Computing, 2015 IEEE Int. Conf. on*, 2015, pp. 130–137.

[21] R. Robinson, "Understand enterprise service bus scenarios and solutions in service-oriented architecture, part 1: The role of the enterprise service bus," http://www.ibm.com/developerworks/webservices/library/ws-esbscen/, 2004.

[22] X. Lin, J. Andrews, A. Ghosh, and R. Ratasuk, "An overview of 3gpp device-to-device proximity services," *Communications Magazine, IEEE*, vol. 52, no. 4, pp. 40–48, 2014.

[23] C. Chang, S. Ling, and S. Srirama, "Trustworthy service discovery for mobile social network in proximity," in *PerCom '14 Workshops*. IEEE, 2014, pp. 478–483.

[24] W. M. Van der Aalst, "The application of petri nets to workflow management," *Journal of circuits, systems, and computers*, vol. 8, no. 01, pp. 21–66, 1998.

[25] L. Baresi, A. Maurino, and S. Modafferi, "Workflow partitioning in mobile information systems," in *Mobile information systems*. Springer, 2005, pp. 93–106.

[26] X. Qin, H. Jiang, Y. Zhu, and D. R. Swanson, "Dynamic load balancing for i/o-intensive tasks on heterogeneous clusters," in *High Performance Computing-HiPC 2003*. Springer, 2003, pp. 300–309.

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

[28] K. Dar, A. Taherkordi, H. Baraki, F. Eliassen, and K. Geihs, "A resource oriented integration architecture for the internet of things: A business process perspective," *Elsevier PMC Journal*, vol. 20, pp. 145–159, 2015.

[29] J. Mass, C. Chang, and S. N. Srirama, "Workflow model distribution or code distribution? ideal approach for service composition of the internet of things," in *IEEE SCC '16*, 2016, pp. 649–656.

[30] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos, "MAPCloud: mobile applications on an elastic and scalable 2-tier cloud architecture," in *UCC '12*. IEEE, 2012, pp. 83–90.

[31] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "MuSIC: Mobility-Aware Optimal Service Allocation in Mobile Cloud Computing," in *CLOUD '13*. IEEE, 2013, pp. 75–82.

[32] N. Fernando, S. W. Loke, and W. Rahayu, "Honeybee: A programming framework for mobile crowd computing," in *MobiQuitous '12*. Springer, 2012, pp. 224–236.

[33] ——, "Computing with Nearby Mobile Devices: a Work Sharing Algorithm for Mobile Edge-Clouds," *Transactions on Cloud Computing*, vol. 00, no. 99, pp. 01–14, 2016.

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