UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Isaac Agaba

Adaptive Process Distribution at the Edge of IoT using the Integration of BPMS and Containerization

Master's Thesis (30 ECTS)

Supervisor(s):
Dr. Chii Chang

Tartu 2017

Adaptive Process Distribution at the Edge of IoT using the Integration of BPMS and Containerization

Abstract:

Emerging cloud-centric Internet of Things (IoT) system relies on distant data centers to manage the entire processes, which raises the issue of latency. To address the issue, researchers have introduced the Edge computing methodologies that carry out computation closer to the edge network of IoT system. Among the numerous Edge computing approaches, Mist computing paradigm emphasises the mechanism that moves the computation further to the front-end IoT devices. Although the architecture of Mist computing is promising, it raises a new challenge in how the Business Process Management System for IoT (BPMS4IoT) distributes the business process workflow to the heterogeneous IoT devices? In general, executing business process workflows relies on the common platform for executing customized tasks. For example, if the management server defines a Python script task in a workflow, which has been allocated to an IoT device, the workflow engine of the IoT device must have the compatible execution method. Such a requirement is less flexible when one considers the heterogeneity of the IoT devices. Therefore, in this thesis, the author proposes a framework to decouple the workflow task execution method from the workflow engines using the containerization technology. A proof-of-concept prototype has been developed and has been tested on several single-board computers-based IoT devices. Further, a case study has been performed to demonstrate the performance of the proposed framework comparing to the cloud-centric system.

Keywords:

Internet of Things, Edge Computing, Containerization, Mist Computing, Business Process Systems

CERCS: P170 Computer science, numerical analysis, systems, control

Konteinertehnoloogia ning protsessihaldussüsteemide integratsioonil põhinev adaptiivne protsessijaotus värkvõrgu serval

Lühikokkuvõte:

Täna levivad pilvepõhised värkvõrgu (asjade interneti) süsteemid tuginevad protsesside halduseks kaugel asuvatel andmekeskustel, mis toob endaga kaasa latentsusprobleeme. Vastusena sellele probleemile on varem välja pakutud servaarvutuse lähenemine, kus arvutused viiakse läbi asjade interneti süsteemi võrgule füüsiliselt lähemal. Mitmete servaarvutuse metoodikate seas on uduarvutus lähenemine, kus rõhk on arvutuste liigutamisel värkvõrgu seadmetele endile. Ehkki uduarvutusel põhinev arhitektuur on paljutõotav, tõstatab see küsimuse – kuidas värkvõrgu protsessihaldussüsteemid (BPMS4IoT-süsteemid) äriprotsesse heterogeensetele värkvõrgu seadmetele jaotama peaksid? Levinud on lähenemine, kus protsesside töövooülesannete käituseks tuginetakse ühisele platvormile. Näiteks, kui haldusserver defineerib teatud töövoo ülesandena Pythoni skripti ja määrab selle seadmele, siis peab seadme töövookäitusmootor toetama vastavat mehhanismi skriptide jooksutamiseks. Selline nõue ei ole paindlik, arvestades värkvõrgu seadmete heterogeensust. Käesolevas magistritöös pakub autor välja raamistiku, mis eraldab töövoo ülesannete käitusmeetodi käitusmootorist kasutades selleks konteinertehnoloogiat. Töö käigus arendati välja raamistiku prototüüp ning viidi läbi katseid mikroarvutitel põhinevail seadmetel. Lisaks võrreldi väljapakutud uduarvutuse raamistiku jõudlust pilvearvutusel põhineva süsteemiga.

Märksõnad:

Asjade internet, Värkvõrk, Servarvutus, Konteinertehnoloogia, Uduarvutus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1 Introduction

1.1 Preamble

Internet of things (IoT) [Ash09] refers to a global interconnection of objects (food, home appliance, and vehicles) with unique identifiers such as Internet Protocol (IP) address with the ability to communicate, interact or react to given changes with each other [Soma15].

As the number of IoT devices increases with time, it is predicted that by 2020, there will be almost 50 billons physical devices being connected to the Internet [Rose04].

Various computing methods have been proposed in last decade to improve the IoT devices related problems such as low performance and high resource usage.

Cloud computing method allows IoT devices to carry executions remotely with the Internet accessible computer (Cloud). This gives IoT devices virtually unlimited capabilities in terms of storage and processing power [Bpp14].

As the use of IoT devices increases in most critical environments such as homes, hospitals, military, Cloud computing paradigm can hardly satisfy the requirements of high mobility support, location awareness and low latency [Sw14]. To address some of these issues, Edge computing was proposed.

Edge computing methodology shifted computation from remote cloud to the computational devices that are closer to the front-end IoT devices within edge networks [Phmsl16]. The closeness of edge devices has improved the efficiency IoT devices as it enables them to do real-time operations with less latency limitation [Mb16].

Mist Computing an immerging methodology goes further beyond Edge computing as it pushes the computation to the sensors and actuators. Hence, this even saves more power since communication from a node to Edge nodes takes more power than computation at the nodes [Ptjrc15].

Business Process Management Systems (BPMs) utilize workflow engines that provide the management capabilities to the overall IoT system without getting involved in the low-level complex programming languages [Drmr13].

This thesis address the problems associated with Mist Computing, such that it will enable smooth process execution onto this constrained IoT devices which do differ in operating system and platform wise [Cnb16].

1.2 Motivation & Challenges

1.2.1 Motivation

Let's us take smart environments such as smart parks with several wireless sensors and actuators devices deployed by researchers or companies. Actuators being single board computers (SBCs) with minimal capacity compared to personal computers which do carry out some computation on the data collected before it is submitted back to cloud data centres.

SBCs have limited capacity of storage and processing power thus if they have heavy tasks executed they do consume resources such as memory and CPU usage.

1.2.1.1 Scenario

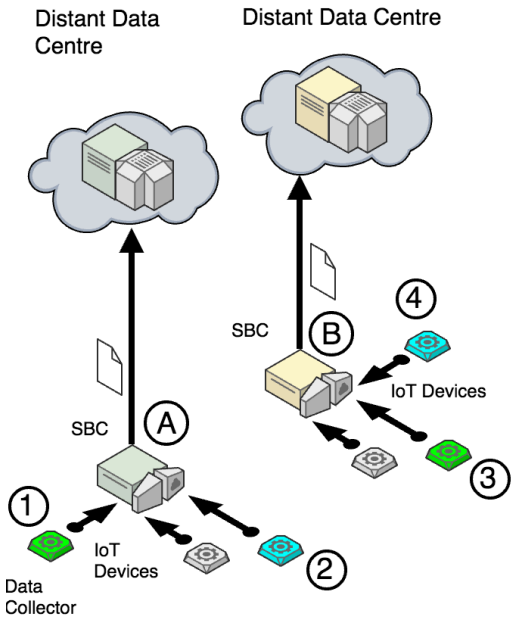


Figure 1: Smart Park with SBCs.

In our scenario, we do consider to have a smart environment park that contains different SBCs, which belong to different companies, collecting data such as air quality, temperature, and humidity (see Figure 1).

At a given point of time, SBCA may be overloaded with continuous carrying out heavy computations such as data filtering, sorting, and transformation before it's submission to the Cloud.

In the proximity of SBCA there exists SBCB which may be idle or even with more processing capacity. It would be a good option for SBCA to take this advantage by offloading some of its executions to SBCB.

There is a need of smooth collaboration computing model that could enable them manage the extending of a workflow process from one edge device to another.

1.2.2 Challenges

- **Heterogeneity**. In our scenario, as these SBCs may be from different vendors. These devices can differ in hardware and operating system, so it may not feasible to deploy straight away and run one workflow from one device to another as they may be a need to have some dependency fixation.

Therefore, there is a need to address this heterogeneity issue by implementing a common standard-based execution strategy which will enable cross-platform execution across all edge devices.

- **Lightweight**. The deployed processes need to be light in size so that it can easily be transferred between two devices, but the fact none of the devices needs to have prior knowledge of resources required execute the deployed process. Packaging the entire implementation would be a better option but this makes the deployed system size heavier.

Therefore, our approach needs to find a way of making deployable workflow process's size light but at the same time containing all the required implementation.

1.3 Research Objectives and Contribution

The goals of this thesis are:

1. Develop Mist Computing Resource Planning Framework (MRF) that will be used to validate whether the workflow business process can be lightly transferred and executed on edge devices without worries about the heterogeneity of the devices.
2. To evaluate what benefits does MRF add to the devices compared to the existing computing methods.

1.4 Thesis Outline

The rest of the thesis is divided into sections: Section 2 which contains literature review, background of the technologies being used and related works. Section 3 consists of the proposed system architecture description. Section 4 describes system implementation, testing, and discussion. This thesis is concluded in Section 5 together with future research directions.

2 Review of the state of the art

2.1 Internet of Things

Internet of Things (IoT) introduced by AutoID labs [Ash09] was initially used for radio frequency identification (RFID) tags system. IoT is the global composition of things or objects which are active participants such as (food, vehicles, Fitbit) in processes [Gbmp13], having unique identifiers that enable them to be discovered and to interact with other objects using existing communication protocols [Zwclq10].

IoT is categorized into four major application domains namely Personal and Home, Enterprise, Utilities, and Mobile. These domains scale respective to homes, community, national or regional scale, and mobile which spreads across other domains because of its connectivity and scale nature [Gbmp13] (see in Figure. 2).

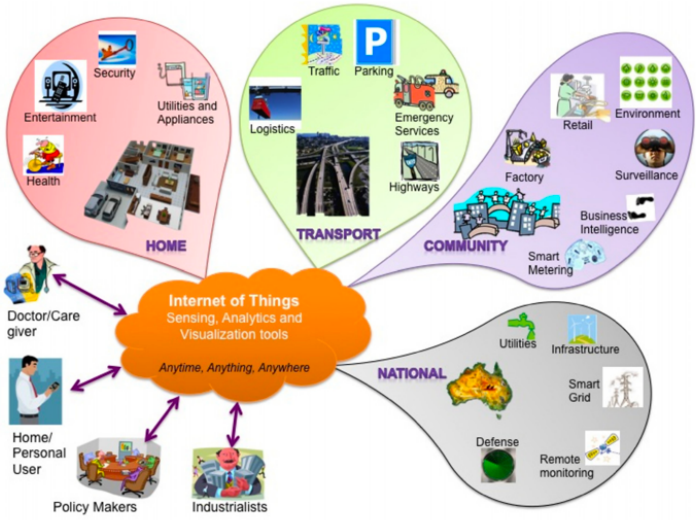


Figure 2: IoT Domain Adopted From [Gbmp13].

In [Wllsd10], Wu et al categorized IoT architecture is into five layers (see in Figure. 3):

- Physical layer is the first and lowest layer that deals with hardware.
- Data layer acts as protocol layer which transfers data between adjacent network nodes.
- Network layer deals with logical device addressing, data packaging, manipulation and delivery, and routing. It handles communication between two devices.
- Transport manages communication of the two end-to-end applications that run on the two devices that are on the internetwork.
- Application layer provides services that are required for the application programs that are in communication.

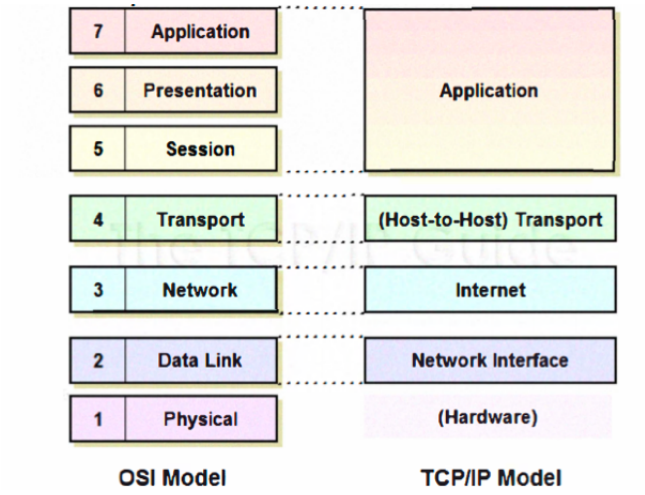


Figure 3: IoT architecture Adopted From [Wllsd10].

- **IOT Single Board Computer Devices**

Single board computer (SBC) are tiny, low specification computers with single circuit boards, microprocessor(s), little memory, capable of using input and output devices and that fully operate just as a regular personal computer [Vm15, Cmlp14] (see in Figure 4).

Their limited consumption of power has increased their usage in IoT environments. They are easily deployed anywhere as actuators in Wireless Sensor and Actuator Networks (WSAN) a composition of sensors, that collect specified data in the surrounding, and actuators that perform specific action [Mpga05].

Some of the widely-used SBCs, include Raspberry[1], Odroid[2], Cubie board[3], and Arduino[4].

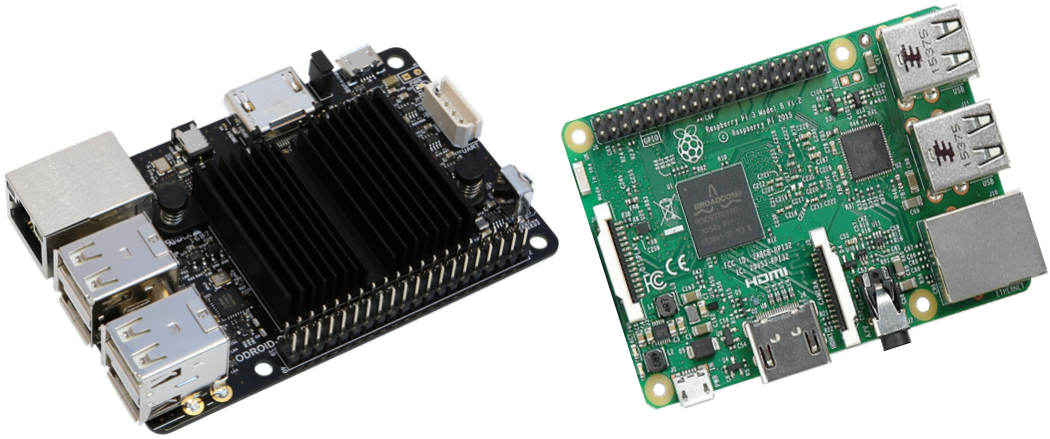


Figure 4: SBC Devices Adopted From [Ras16, Har16].

[1] https://www.raspberrypi.org

[2] http://www.hardkernel.com/main/main.php

[3] http://cubieboard.org

[4] https://www.arduino.cc

2.2 IoT Computing Methods

Since most IoT devices are constrained in hardware, storage and processing power. As the workload becomes heavier this may lead to low efficiency and performance.

Their increased usage in critical and real-time processes, that demand faster computation has brought forward a need for them to optimize the resources and improved, device performance and efficiency.

Below are some of the profound IoT computing methods, being proposed to help these IoT networks with these constrained IoT devices.

2.2.1 Cloud Computing

Cloud computing methodology usage in past decade in IoT networks has provided on-demand access to shared computing resources pool (storage, applications, services, and software) that are hosted in the cloud.

These are easily provisioned when needed by any authorised device in need of them with minimal vendor interaction [Rsms12, Nist09]. See figure 5, the architecture of cloud computing model.

The threat of insecurity of data transmitted between devices, service instability, and latency are major drawbacks of Cloud computing[Wb10]. As Cloud computing participant's machines, may be many hops away from each other, some data packets can be lost or man in middle attacks can be done on the transmitted data.

To reduce on the drawbacks of far way cloud, usage of cloudlets was proposed as it brought, a limited local Cloud nearby [Sbcd09].

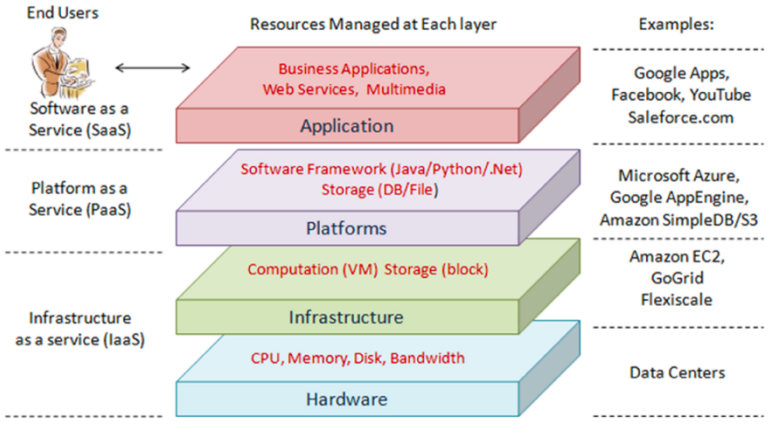


Figure 5: Cloud Computing Adopted From [Zcb10].

2.2.2 Fog and Edge Computing

Fog computing pushes closer Cloud computing paradigm down to the edge network by processing data at fog nodes or IoT gateway. This has solved some of Cloud computing challenges such as high latency and failure ensure total location awareness [Frpj14, SW14]. These fog nodes can be deployed at factories, parks, health care units, transport stations [Cis15].

Edge computing brings, even more, closer the intelligence and application logic past the fog nodes, as it directly does these computations at devices programmable automation controllers that are in the edge networks [Pt04]. This increases the infrastructure efficiency as it provides intermediate layers of computation, networking, and storage closer to IoT devices [MB16].

However, in most cases, Fog computing and Edge computing terms are being interchangeably used. This is incorrect as they are completely different. Fog computing works hand in hand with Cloud computing but Edge can work without Cloud [Ope17].

Figure 6 shows comparison of attributes from Cloud and Fog computing.

Requirements	Cloud Computing	Fog Computing
Latency	High	Low
Delay Jitter	High	Very low
Location of Service	Within the Internet	At the edge of the local network
Distance between client and server	Multiple hops	One hope
Security	Undefined	Can be defined
Attack on data enroute	High probability	Very low probability
Location awareness	No	Yes
Geo-distribution	Centralized	Distributed
No. of server nodes	Few	Very large
Support for Mobility	Limited	Supported
Real time interactions	Supported	Supported
Type of last mile connectivity	Leased Line	Wireless

Figure 6: Computing Comparison Adopted From [Cis17].

2.2.3 Mobile Cloud Computing (MCC)

In [Flr13], Niroshinie et al describe Mobile Cloud computing as:

1. MCC gives applications ability to be run on remote machines in the cloud so that they can be accessed by client mobile devices that use resources being served over an internet connection.
2. MCC clusters resources in a peer network among mobile devices. This forms a local cloud of mobile devices in the vicinity that provides different services to each other.
3. Mobile cloud computing enables mobile devices to use cloudlet computers with in the proximity, to carry out executions that would have been carried out in the cloud.

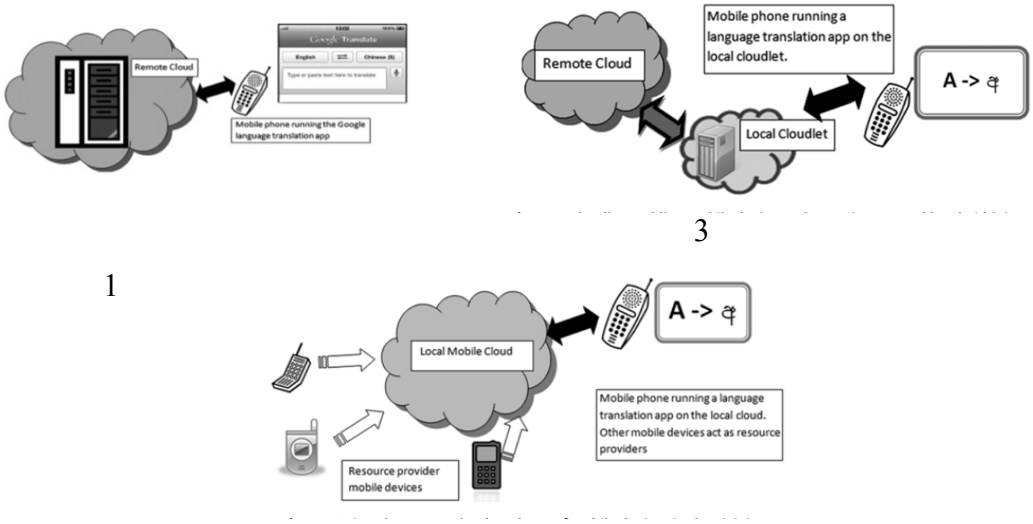


Figure 7: Adopted From [Flr13].

Even though Mobile cloud computing reduces high latency and bandwidth usage when compared to Cloud computing though it self also has some drawbacks such as low reliability and privacy related issues [KI10].

2.2.4 Mobile Edge Computing (MEC)

This brings Cloud computing services at the edge of the cellular network. MEC runs a cloud server at the edge of a mobile network and performs specific tasks that could not be accomplished with traditional network infrastructure.

"*Operators can open the radio network edge to third-party partners, allowing them to rapidly deploy innovative applications and services towards mobile subscribers, enterprises, and other vertical segments*" [Mec16].

2.3 Virtualization

Virtual machines are machines that are being fooled [Rose04], to think that they are being run on a real hardware device. Therefore, on one device severally virtual machines can be run all operating independently as if they are only one using the device hardware.

Below is the summarization of some of the common forms of virtualizations:

2.3.1 Full Virtualization

This type of virtualizations enables complete simulation of computer hardware parts. This makes it easier to run different operating systems on a given device as it can virtualize memory, processors, and I/O devices [Uhi05].

Most full virtualized machines use hypervisors which is a layer of software that can implement instructions set on hardware as it can run directly on the hardware [Mlo97].

Hypervisors are classified into: Type 1 hypervisors which are placed directly on top of the system hardware such as Microsoft Hyper-V[5], Citrix XenServer[6], and Type 2 hypervisors which are hosted on top a host operating system, for example VMware Player [7], Parallels[8] [Dk13] (see in Figure .8).

[5] https://www.microsoft.com/en-us/cloud-platform/server-virtualization

[6] https://xenserver.org/

[7] http://www.vmware.com/

[8] http://www.parallels.com/

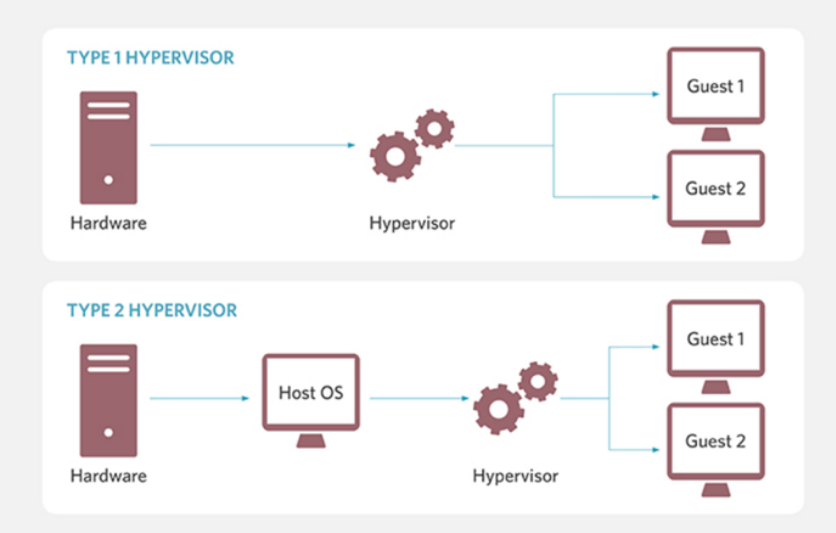


Figure 8: Hypervisor Types Adopted From [Hyp17].

2.3.2 Container Virtualization

Containerization is an operating system virtualization method which replaces the tradition hypervisor virtualization methods as it allows the use of virtualized machines that share the same kernel as the host operating system [Cmfvp16, Car15].

Linux kernel virtualization is classified into: namespaces which isolates process groups so that each process could only see processes resources that belong to the same group and Control groups (Cgroups) which does limit how much of the resources a given process can use for example reservation of memory, central processing unit (CPU) usage that is being assigned to a given process [Pahl15].

Container Images are lightweight independent bundled and software with all the dependencies needed to be executed regardless of the computer platform (Linux, Windows) [Doc, Phmsl16]. When container images are being executed the running instance of a container image is called a container [Car15].

Runnable containers do add minimal overhead on the device being used compared to hypervisors [RN16] because they do share the same kernel as the host machine [MB16].

Containers use namespaces for process isolation of processes. Different containers can be interlinked through network interfaces [Pahl15]. This reduces creation or multiple guest operating system which reduces overhead due to virtualization of both hardware and drivers (figure 9).

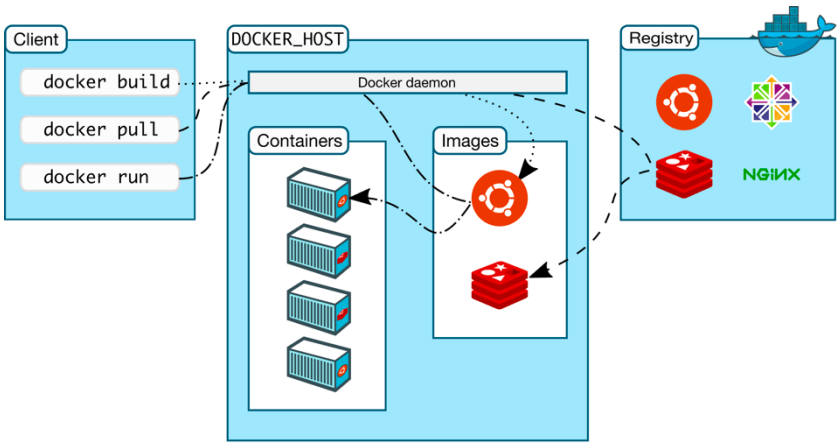


Figure 9: Docker Architecture Adopted From [Dar17].

Containers do use cluster managements techniques such as Mesos and Kubernetes for management, scaling, and deployment of containers.

Kubernetes developed by Google sets several nodes made up containers with services that can be accessed by other containers in other hosts by automatically scheduling jobs to ensure that the applications run in the desired state, through its auto starting, self-healing and rescheduling techniques [Amj15].

2.4 Peer to peer Communication

Most SBC's IoT devices come pre-installed with the support of Bluetooth and WIFI technologies, which do enable these devices to interact locally and globally with other devices in the network. Other technologies such as CoAP[9], ZigBee[10], MQTT[11] that can be used for device to device communications. The following subsections contain the review of some of these devices to device communication technologies.

[9] http://coap.technology

[10] http://www.zigbee.org

[11] http://mqtt.org

2.4.1 Bluetooth

Bluetooth technology ordinary usage was in audio and stereo communications [Cha14], however, in the past decades it has expanded its usage to many short-range wireless communication markets such as the IoT and machine-to-machine (M2M) communications.

Bluetooth technologies include regular Bluetooth, Bluetooth EDR, Bluetooth HS and Bluetooth low energy. Bluetooth LE devices do consume less energy consumption, memory footprint. The ability that they can be used in end-to-end IP connectivity makes them suitable to be used in critical areas [Cha14].

2.4.2 Wi-Fi Direct

Wi-Fi Direct[12] technology by Wi-Fi alliance takes a different approach to enhance device to device connectivity as it builds upon the successful IEEE 802.11 *infrastructure* mode and lets devices negotiate who will take over the access point-like functionalities dynamically [Css13].

Wi-Fi Direct dynamically enables devices to act as a peer-to-peer group owner (P2P GO) or a peer to peer client (P2P Client).

2.5 Business Process Workflow

Business Process Management (BPM) it is an art and science of how a workflow in an organization or systems are executed to ensure consistent outputs [Drmr13]. Business Process Model and Notation (BPMN) expresses all the information in an IoT system process (see in Figure 10).

In [Sac13], Sonja et al categorised the major components in the IoT Domain Model of the IoT-Architecture as:

- IoT service: These interfaces allow access to other heterogeneous components at native interfaces of the devices hence exposing devices functionality as a single unit business process.

[12] http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

- Physical entity: This refers to a unique element within the proximity in which is of central interest for the IoT.
- IoT device: This act as a mediator between the BPM process and the physical world from which data is being collected from.
- Native service: These are hosted onto IoT devices collect information about entities or perform actions on entities.

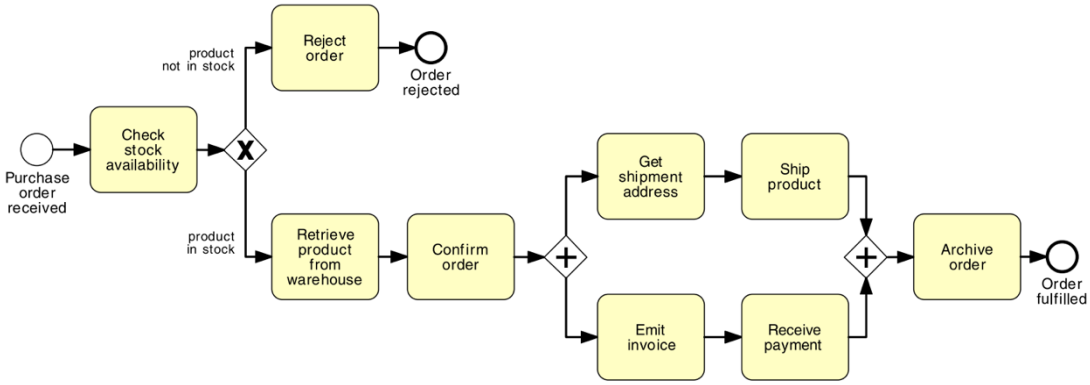


Figure 10: Adopted From [Drmr13].

2.6 Related Works

In [Phal16], Pahl et al review about the impact container virtualization on edge devices when being placed into clusters. As their study focused on how Edge clouds could move heavy-weight computations to distributed lightweight resources close to users.

They used containerization technology to build clusters that consisted of customized platforms of SBCs nodes, running different containers with orchestration services that enabled the communication of these SBCs nodes in the clusters.

In [Pmlm15], Riccardo et al proposes the designing of gateways used in Cloud of Things which distributes a collection of resources, enabled in a horizontal integration with various IoT platforms and applications.

These gateways would oversee, manage data from IoT devices and act as endpoint for the communication between cloud data-centers and local devices. The proposed gateways in their study used container based virtualization which gave an improvement of 2.67%, 6.04% and 10% in CPU, memory performance and Disk I/O.

In [Rn16], Ramalho et al study evaluates the performance difference between containerized based and the hypervisor-based virtualization at the network edge. The use of hypervisor-based virtualization had good results in regards of isolation in the last decade but containerization abilities such fast to boot up, fast migration and easy to maintain have taken virtualization to next level.

From their study, the performance tests were run on CubieBoard2 with container based vs hypervisor-based virtualization. Both NBENCH and SysBENCH tests showed that container virtualization outperforms KVM in every situation when compared to the Native execution.

3 System Overview

This section introduces our proposed framework and in the subsections, we will discuss the architecture and overview of different components of our proposed system.

"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" [Lcs16].

Mist Computing Resource Planning Framework (MRF), an open standard-based service-oriented context-aware computing model that uses Mist Computing, virtualization, and workflow management technologies.

MRF implementation address challenges of the proposed "Adaptive Process Distribution at the Edge of IoT using the Integration of BPMS and Containerization". It's application on SBCs devices will widen the device workflow execution scope. As it will enable collaborative distribution of sub process of a complex workflow to several Edge devices with in the proximity.

3.1 Scenario

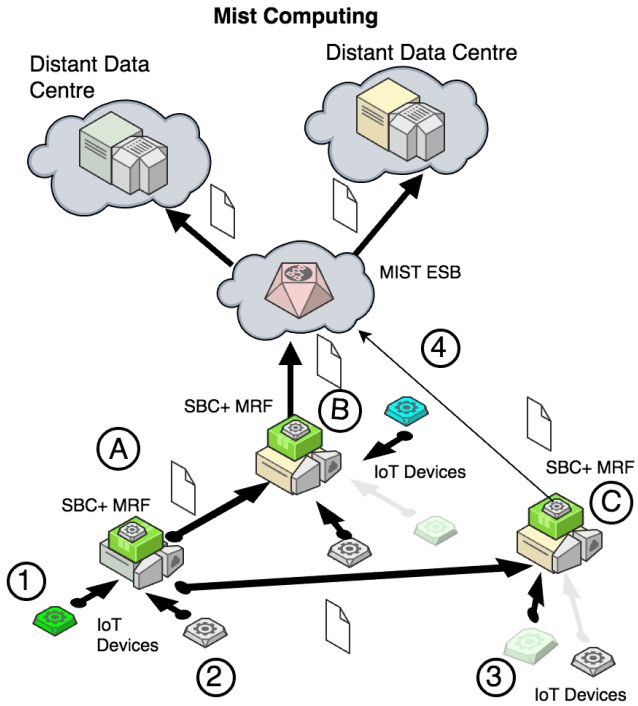


Figure 11: Edge With MRF.

The requirement at hand of a collaborative, cross-platform Edge computing model in Section 1.2.1.1 that would address challenges described in Section 1.2.2.

Let's take a scenario (Figure 11) that consists three SBCs identified as SBCA, SBCB and SBCC that are within proximity of each other.

As mentioned before, that these devices may belong to different organizations. There is low possibility that these devices to have similar or compatible hardware device platforms and operating system.

Installation of MRF on these devices will enable them to use each other's resources such as (computation power, memory). They will be to offload sub processes to each other freely without the dependencies limitation. Therefore, the offloaded processes will be dynamically executed out of the box, regardless of the device specifications in hardware and software.

In our scenario, MRF will dynamically enable these SBCs to interact, deploy, execute the given processes and sending of the response call-back to the seeker SBC which did deploy the sub process.

Therefore, SBCA will distribute its sub process to SBCB and SBCC.These would process these executions and after each execution, a desired response will be sent back to SBCA. This would reduce SBCA from being overloaded with heavy tasks or from being fully reliable to some far away cloud.

3.2 System Architecture

MRF should be lightweight so that it can be easily run on all SBC devices with minimal overhead on the existing resources to enable SBCs to dynamically communicate, deploy, execute and manage business process tasks regardless of the heterogeneity of devices.

In the following subsections, we are going to discuss in detail about the various components of the system in depicted in Figure 12.

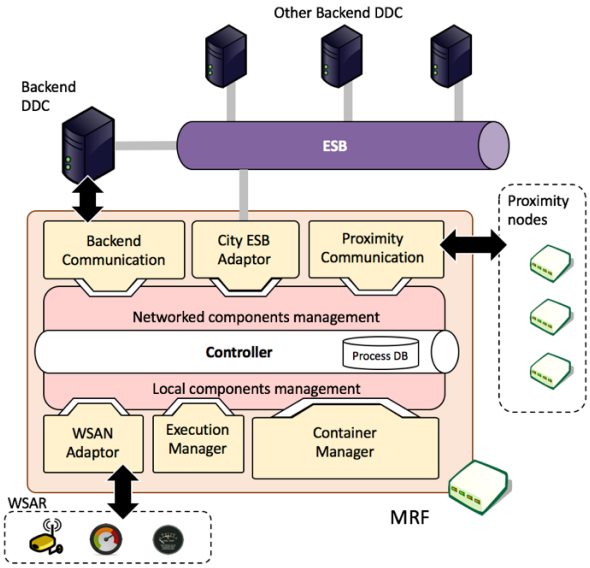


Figure 12: System Architecture

3.3 Container Manager

The system container manager section will use container virtualization techniques to substituting native implementation some parts of the workflow processes with virtualization containers. This will make workflows lighter in size.

Therefore, the responsibility of the Container Manager is to ensure successful operation of containers that belong to workflows are executed.

Below are some of the functionality that could be done by Container Manager:

- Since an SBC can run different workflow processes that may belong to different processes. The Container Manager should ensure that there is a complete isolation of processes to avoid conflict of resources between processes.
- Container Manager is responsible for fetching and starting of required containers at runtime to carry out the desired implementation at given part a workflow.
- The Container Manager ability to remove those containers that are not being used, thus freeing system resources of an SBC.
- The Container Manager also allows communication between containers. This enables us to re-use resources available in some other containers or from the host machine.

3.4 Proximity Communication

Proximity Communication component takes the role of the discovery and communication of SBC's that are within the proximity.

Discovery and connection establishment should be automated. These connections can only be active for a specific time during the collaboration between devices and are taking down once workflow execution between SBCs is completed.

3.5 Execution Manager

The execution manager receives, sends, runs deployable workflow processes between SBC devices. This component exposes resources of a given SBC to others through resource end-points.

3.5.1 Execution Server

Sending, receiving and extracting encapsulated process out of their deployable form into it a form that can be executed by workflow manager is done by the execution server.

The execution server will create all the necessary Representational state transfer (REST) endpoints that can be accessed by other SBCs when they are deploying workflows or when they are sending back responses after the executions completed.

It should run and manage simultaneously identified workflows that whose parts can be accessed by other SBCs.

3.5.2 Workflow Manager

This component executes the deployable workflow received from the execution server. It carries out the given business logic basing on the conditions that were being determined in the workflow modelling description.

The Workflow Manager ensures the atomicity of the workflow as it caters that all re-quired operations are executed in a controlled manner thus making the system more con-sistent.

Since some parts of the workflow are to be substituted with containers. The Workflow Manager needs to work in hand with the Container Manager so that it could dynamically find out a way of executing these implementations that correspond to the given part of the workflow.

3.6 WSAN Adaptor

This component interacts with constrained Wireless Sensor and Actuator Network (WSAN) devices. This component enables SBCs to retrieve data from WSAN devices such as sensory data.

There exist a couple of WSAN adaptors available in the IoT industry such as OpenHab [Ope] that easily enables SBCs to access, read values and change the state of these IoT WSAN devices.

3.7 Backend Communication

SBC devices communicate with their respective backend data centres through this component. Hypertext Transfer Protocol (HTTP)[13] connections can be established between SBC and the backend data centre via Internet connections using 3G, Local area Network (LAN) or WIFI.

3.8 ESB Adaptor

Because of the heterogeneity of SBC devices, there is a high possibility of difference resource request and communication format. For example, one SBC may be using Extensible Markup Language (XML) and the other using JavaScript Object Notation (JSON) format. Therefore, it's through enterprise service bus (ESB) adapter that each SBC could be able to translate or transform data sent or received in a form that it can use or the other SBC [Ibm17].

[13] https://www.w3.org/Protocols/

4 System Implementation & Testing

4.1 System Implementation

This section describes the implementation of Mist Computing Resource Planning Framework built on existing open source technologies. The implementation can be found on GitHub[14]. The following subsections describe how different components of the system were implemented.

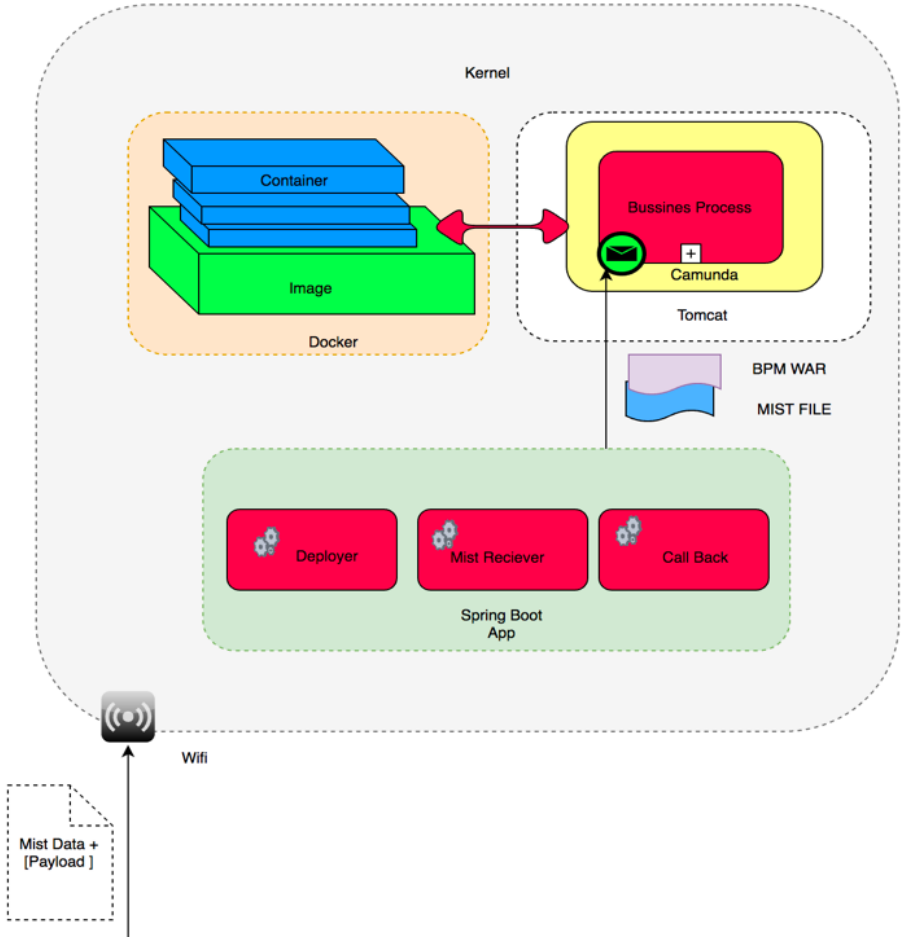


Figure 13: System Implementation

[14] https://github.com/akaiz/mist-framework

4.1.1 Container Manager Implementation

Docker [Doc] was chosen to be our Container Manager. With Docker, we can package implementation of some components of our workflow into Docker images so that, while wrapping the deployable workflow we would substitute the actual implementation of some parts of the workflow with a Docker image that would carry out the same functionality.

Runnable instances of Docker images, called containers can be run right away on any device that has Docker installed. Therefore, this gives us assurance that our implementation can be run on any of these devices without heterogeneity worries.

Docker uses namespace methodology containers isolation, which is done through process-id, networking, mount, and through Control groups (cgroups) [Iigkts14] methodology which uses UnionFS to limit hardware resources assigned to containers.

We did create a Docker image "akaiz/mist-image-procesor" that contained an image processing spring boot application that would take an image as input and extracts out the most dominant colours through iterating throughout all the pixels of the image.

The Docker image was being pushed to Dockerhub[15], so that it could be accessed by any SBC with Docker installed. Therefore, whenever the workflow manager could request the container manager to execute some execution on a given Docker image. It could fetch this image from Docker hub, if it didn't exist locally, then it starts running this Docker image.

Docker is highly rich in commands that can be applied onto the containers when started, such mounting, networking, security, management commands, for example, docker ([start, stop, kill, ps, port, images, build]) ([-v, -a, -q, --link]).

[15] https://hub.docker.com

```
12  public class DockerCommands {
13      Logger LOGGER = Logger.getLogger("Mist tomcat app");
14      Timestamp timestamp = new Timestamp(System.currentTimeMillis());
15      public String stopContainers(String imageName) throws IOException, InterruptedException {
16          LOGGER.info(timestamp+" remove and kill containers that are already  started");
17          String command = "docker ps -a -q --filter=ancestor="+imageName;
18          Process proc = Runtime.getRuntime().exec(command);
19          TimeUnit.SECONDS.sleep(2);
20
21          BufferedReader reader =
22                  new BufferedReader(new InputStreamReader(proc.getInputStream()));
23          String line;
24          while((line = reader.readLine()) != null) {
25              LOGGER.info(timestamp+" Result from stopping container started "+line+" \n");
26              TimeUnit.SECONDS.sleep(2);
27
28              Runtime.getRuntime().exec("docker kill "+line);
29
30              TimeUnit.SECONDS.sleep(2);
31
32              Runtime.getRuntime().exec("docker rm "+line);
33              LOGGER.info(timestamp+" Result from stopping containers finished"+line+" \n");
34          }
35          return "success";
36
37      }
38      public  Process startContainer(String containerCommand){
39          LOGGER.info(timestamp+" Container Intialization \n");
40
41          Process proc = null;
42          try {
43              proc = Runtime.getRuntime().exec(containerCommand);
44          } catch (IOException e) {
45              e.printStackTrace();
46          }
47
48          return proc;
49
50      }
51  }
52
```

Figure 14: Docker Command Manager.

In our case (figure 14) the Docker commands file has two functions stopContainers which stops all containers belonging to a given Docker image and startContainers which starts the execution of container with the provided container commands.

Instances of a given container can be run simultaneously, (see figure 15), running “docker ps –a” results into two running instances of “akaiz/mist-image-procesor:latest”.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f2ff1af05451	akaiz/mist-image-procesor:latest	/bin/sh -c 'exec doc	2 weeks ago	Up 1 minutes	0.0.0.0:5000->5000/tcp	docker-registry
ere2e3405450	akaiz/mist-image-processor:latest	/bin/sh -c 'exec doc	2 weeks ago	Up 12 minutes	0.0.0.0:5000->5001/tcp	docker-registry

Figure 15: Docker Containers

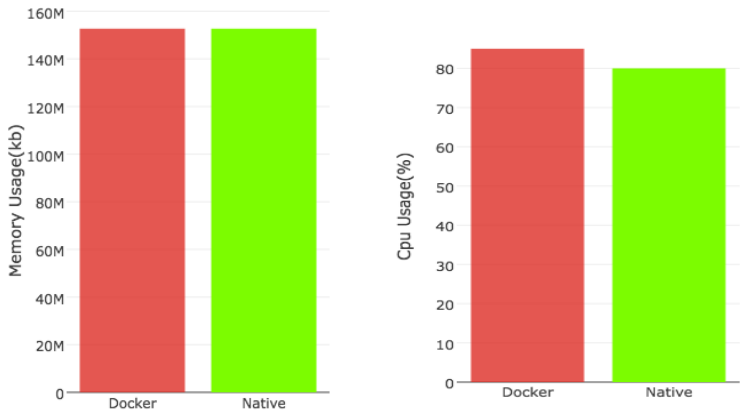


Figure 16: Docker Effect SBC.

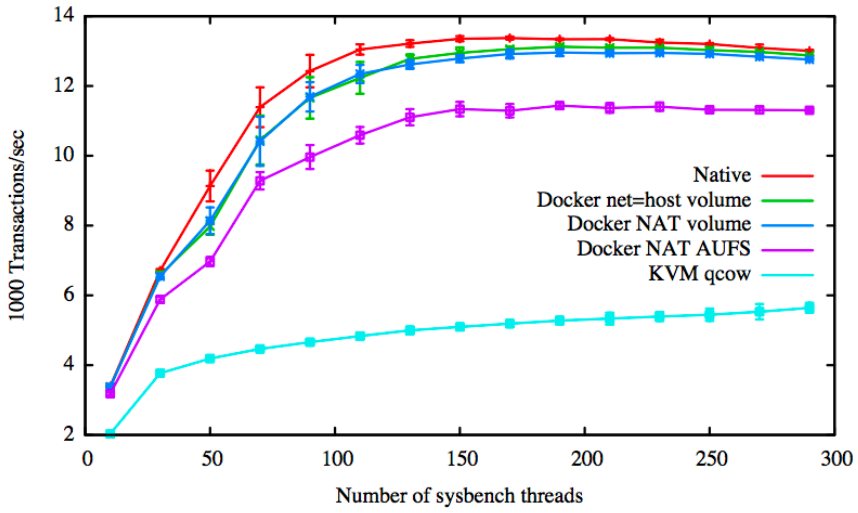


Figure 17: Virtualization Overhead Comparison Adopted from [Ffrr15].

Since Docker images share the same kernel with the host machine, they do add a negligible effect towards the memory and CPU usage (see Figure 16). Usage of Docker added an overhead 2% effect on performance as compared to KVM which gave a high overhead of 40% (see in Figure 17).

4.2 Execution Manager Implementation

Spring boot[16] application was chosen to be the system execution manager because of its dependency management and auto-configuration ability that simplifies the application development process.

```
373  @RequestMapping(value = "/deploy/final",method = RequestMethod.POST)
374  @ResponseBody
375  public ResponseEntity<?> uploadFilenew(@RequestParam("war") MultipartFile uploadfile ,
376                                         @RequestParam("mist") MultipartFile mistfile,
377                                         @RequestParam(name = "payload", required=false) MultipartFile payload,
378                                         @RequestParam("baseFolder")  String folder,
379                                         @RequestParam("callback") String callback,
380                                         @RequestParam("processId") String processId)
381          throws IOException {
382      credsProvider.setCredentials(AuthScope.ANY,new UsernamePasswordCredentials("tomcat", "tomcat"));
383      baseFolder= folder;
384      try {
385          Timestamp timestamp = new Timestamp(System.currentTimeMillis());
386          CsvFile.write(processId,"Recieved deployment ",baseFolder);
387
388          if(!uploadfile.isEmpty() && !mistfile.isEmpty()){
389
390              // Storing payload image
391
392              String directory = savePayload(uploadfile, payload);
393              BufferedOutputStream stream;
394
395          //        Getting data from the mist file
396
397              extractMistInstractions(mistfile, processId, directory);
398              // deploying to tomcat and  start
399
400              return new ResponseEntity<>(deployToCamunda(processId),HttpStatus.OK);
401          }
402
403          return new ResponseEntity<>("PLEASE PROVIDE CORRECT INPUT",HttpStatus.OK);
404      }
405      catch (Exception e) {
406          System.out.println(e.getMessage());
407          return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
408      }
409  }
```

Figure 18: Spring Boot.

Tiny spring boot applications could be run as micro services which would communicate with each other to form complex business applications [Spg]. These are accessible by end points that could be accessed locally and publically by other SBCs within the collaboration.

For example, end-point "[SBC-IP-ADDRESS]/deploy/final" does receive a war (deployable workflow), mist (contains execution pre-defined commands), payload and other parameters. This end-point will be called by any SBC that wishes to make collaboration with this device (see in Figure. 18 line 375).

[16] https://spring.io/guides/gs/spring-boot/

4.3 Workflow Manager Implementation

We choose to use BPMN because it supports orchestration as well as choreography and it because it has a larger set of workflow patterns and events [Dtbeg15].

Camunda[17] (Figure 20) an open source platform workflow and business process management system which runs on top of Tomcat[18] (Figure 19) was chosen to be the Workflow Manager. Therefore, it will manage the deployed business processes each identified by a unique business process id, this will enable simultaneous workflow execution.

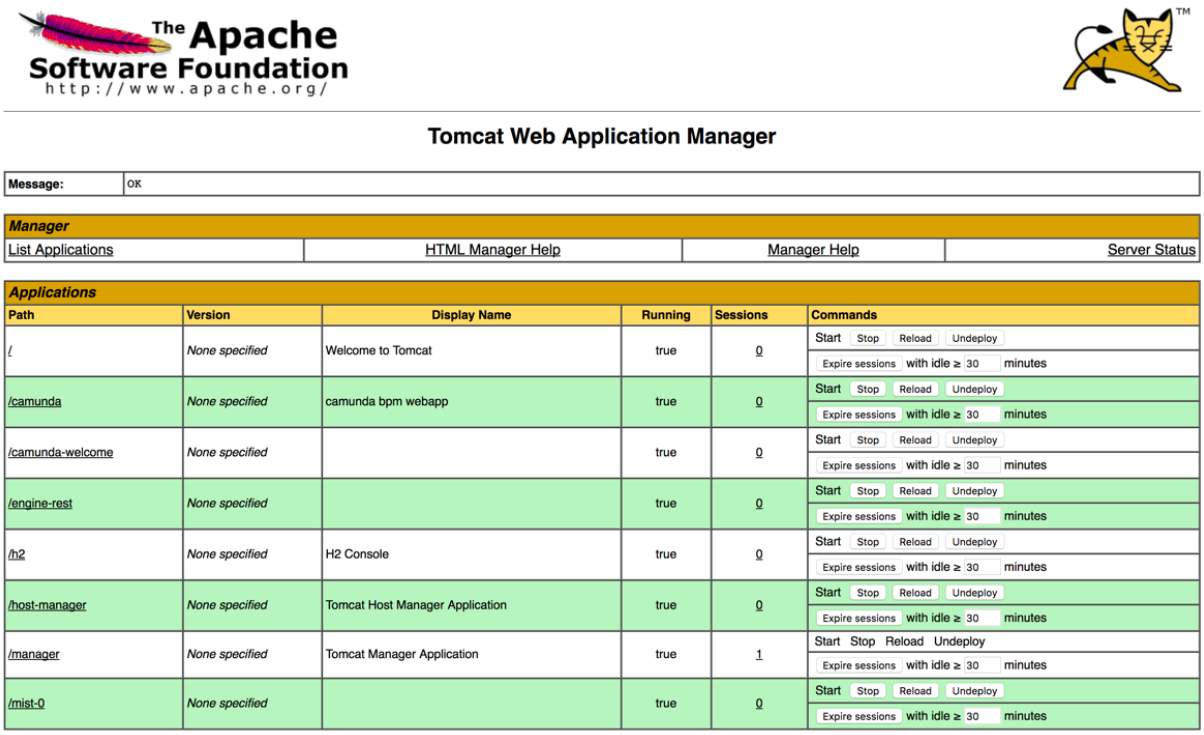


Figure 19: Tomcat.

[17] https://camunda.org

[18] http://tomcat.apache.org

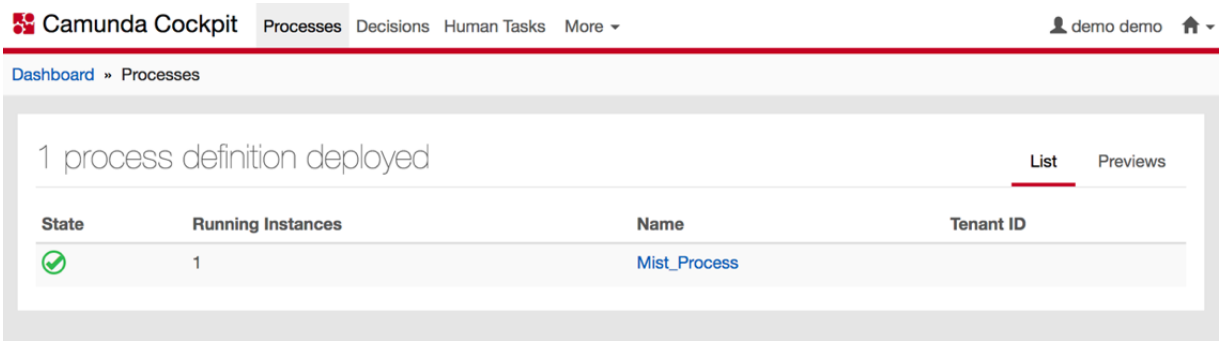


Figure 20: Camunda Cockpit.

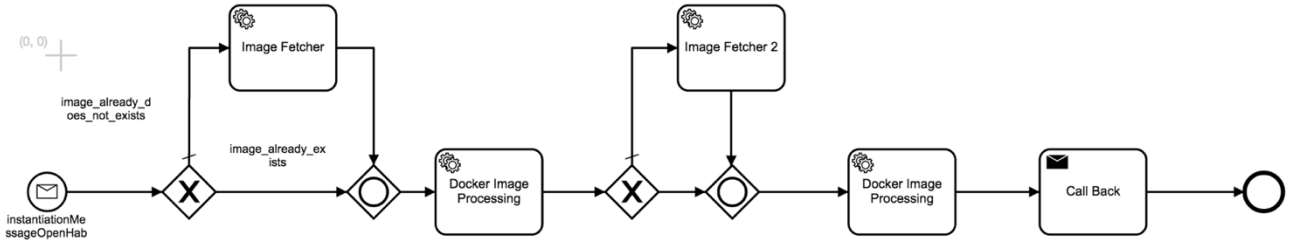


Figure 21: BPMN Full.

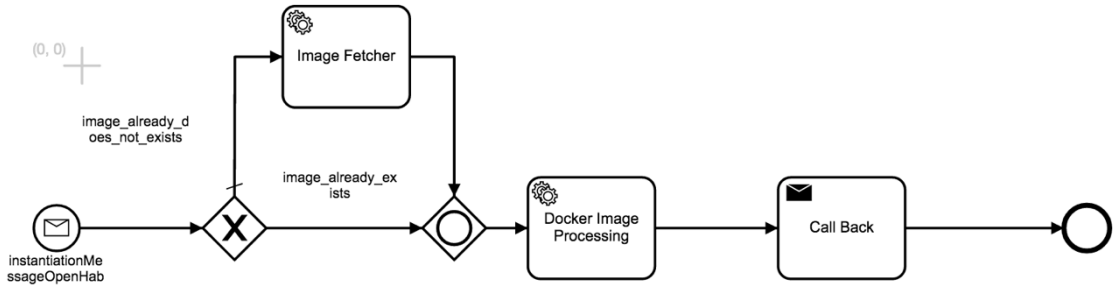
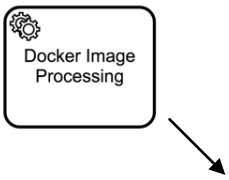


Figure 22: BPMN Partial.

The table below describes some of the components of our BPMN illustrated in Table 1:

Id	Element	Use
1	Message start event	The Execution server starts the workflow by making a request to Camunda with the identifier of this message event.
3	XOR Split	This element holds the logical condition that checks if the image payload was not being sent and if it is true it directs the flow to image fetch component.
5	OR Join	This element directs the workflow to the Docker Image Process Task regardless of what was the previous decision at element.
4	Image Fetch Service Task	This component fetches the image from a given URL in cases were the image payload is not being sent.
6	Docker Image Process Task	This component does image processing but this implementation is being done by a Docker container
7	Call Back Service Task	This component sends back response to the respective sender SBC once the processing has been done.
8	End event	This is the ends of the workflow process.

Table 1: BPMN Elements.




```
108     <bpmn:serviceTask id="Task_1p0zmxc" name="Docker Image Process" camunda:class="ut.DockerImageProcessing">
109       <bpmn:extensionElements>
110         <camunda:field name="dockerImage"> <camunda:string>akaiz/mist-image-processor</camunda:string>
111 </camunda:field>
112         <camunda:field name="command"> <camunda:string>findDorminantColour</camunda:string>
113 </camunda:field>
114         <camunda:field name="imagePath"><camunda:string>/home/pi/mist/mist_img.jpg</camunda:string>
115 </camunda:field>
116       </bpmn:extensionElements>
117       <bpmn:incoming>SequenceFlow_0bbflhi</bpmn:incoming>
118       <bpmn:outgoing>SequenceFlow_07o7ol4</bpmn:outgoing>
119     </bpmn:serviceTask>
```

Figure 23: Camunda Service Task.

Camunda adds more functionality to the regular BPMN as it adds the use of custom extensions properties to service tasks. These extensions are used to pass input parameters to the service tasks.

In our case, we passed the desired Docker image name and defined what commands needed to be executed (figure 23). This is what makes our workflow light weight and free heterogonous issues as Docker images can be run across all platforms.

This also makes modelling more flexible as business process modellers, would careless of how to do the implementation. In figure 23, they could just add the service Task they need for example image-processing by just adding “akaiz/mist-image-processing” and the command they want in this case it was finding dominant color. These values would be picked by the respective custom Docker service task (figure 24 line no 24 and 25).

Once this Docker image is started it would launch a light spring boot service with all the required dependencies, that could carry out this computation (figure 25) to find out the desired command onto the image.

```
10  public class DockerImageProcessing extends DockerCommands implements JavaDelegate {
11      private static final Logger LOGGER = Logger.getLogger("CallBack");
12      private Expression dockerImage;
13      private Expression command;
14      private Expression imagePath;
15       String localhost=null;
16      public DockerImageProcessing() {
17          super();
18      }
19      public void execute(DelegateExecution execution) throws Exception {
20          String baseFolder = (String) execution.getVariable("baseFolder");
21          String camundaHost = (String) execution.getVariable("camundaHost");
22          localhost=camundaHost;
23          String dockerImageValue = (String) dockerImage.getValue(execution);
24          String commandValue = (String) command.getValue(execution);
25          String imageUrlValue = (String) imagePath.getValue(execution);
26          File file = new File(imageUrlValue);
27          String folder = baseFolder+"/mist-framework/mist-deployer-app/mist-files/";
28          String command = "docker run -p 8090:8080 -v "+folder+":/mist"+ "+dockerImageValue;
29          LOGGER.info("Final docker command"+command);
30          String line = " ";
31          // starting container
32          startingContainer(execution, baseFolder, commandValue, command);
33          CsvFile.write(execution.getVariable("log_id").toString(),"Mist-docker completed",baseFolder);
34         super.stopContainers(dockerImageValue);
35
36      }
37
```

Figure 24: Java Delegate.

```
269     private Map getMapFromImage(ImageReader imageReader) {
270         BufferedImage image = imageReader.read(0);
271
272
273         int height = image.getHeight();
274         int width = image.getWidth();
275
276         Map m = new HashMap();
277         for(int i=0; i < width ; i++)
278         {
279             for(int j=0; j < height ; j++)
280             {
281                 int rgb = image.getRGB(i, j);
282                 int[] rgbArr = getRGBArr(rgb);
283                 if (!isGray(rgbArr)) {
284                     Integer counter = (Integer) m.get(rgb);
285                     if (counter == null)
286                         counter = 0;
287                     counter++;
288                     m.put(rgb, counter);
289                 }
290             }
291         }
292         return m;
293     }
294
295     public static String getMostCommonColour(Map map) {
296         List list = new LinkedList(map.entrySet());
297         Collections.sort(list, new Comparator() {
298             public int compare(Object o1, Object o2) {
299                 return ((Comparable) ((Map.Entry) (o1)).getValue())
300                         .compareTo(((Map.Entry) (o2)).getValue());
301             }
302         });
303         Map.Entry me = (Map.Entry )list.get(list.size()-1);
304         int[] rgb= getRGBArr((Integer)me.getKey());
305         return Integer.toHexString(rgb[0])+""+Integer.toHexString(rgb[1])+""+
306                 Integer.toHexString(rgb[2]);
307     }
308
309     public static int[] getRGBArr(int pixel) {
310         int alpha = (pixel >> 24) & 0xff;
311         int red = (pixel >> 16) & 0xff;
312         int green = (pixel >> 8) & 0xff;
313         int blue = (pixel) & 0xff;
314         return new int[]{red,green,blue};
315
316     }
```

Figure 25: Most Common Colour.

4.3.1.1 REST API

Representational State Transfer (REST) services help in machines can communicate with each other. In our implementation who choose to use JavaScript Object Notation (JSON)[19] requests for all internal and external communication requests of SBCs.

Figure 26 depicts contents of the mist file JSON data that is sent to the Workflow Manager.

```
{"messageName" : "instantiationMessageOpenHab",
"businessKey" : "aBusinessKey",
"processVariables" : {"call_back_url" :
 {"value" : "http://192.168.17.168:8098/callback","type": "String"},
 "image_already_exists":{"value" : "yes","type": "String"},
 "log_id":{"value" :"5RLP3,mist_file_heavy_0.txt","type": "String"}
}
}
```

Figure 26: Mist File.

[19] http://www.json.org/

4.4 Aim of Testing

Our major aim is to confirm whether the use of MRF which uses Mist computing had a positive impact on execution time and system resources such as CPU usage, memory usage of the edge SBCs compared to use of Sole computing or Cloud computing.

Sole computing involves executing everything locally on the SBC device. Cloud computing involves offloading parts of the process to the cloud. Mist computing will offload sub processes to other SBC devices within the edge network.

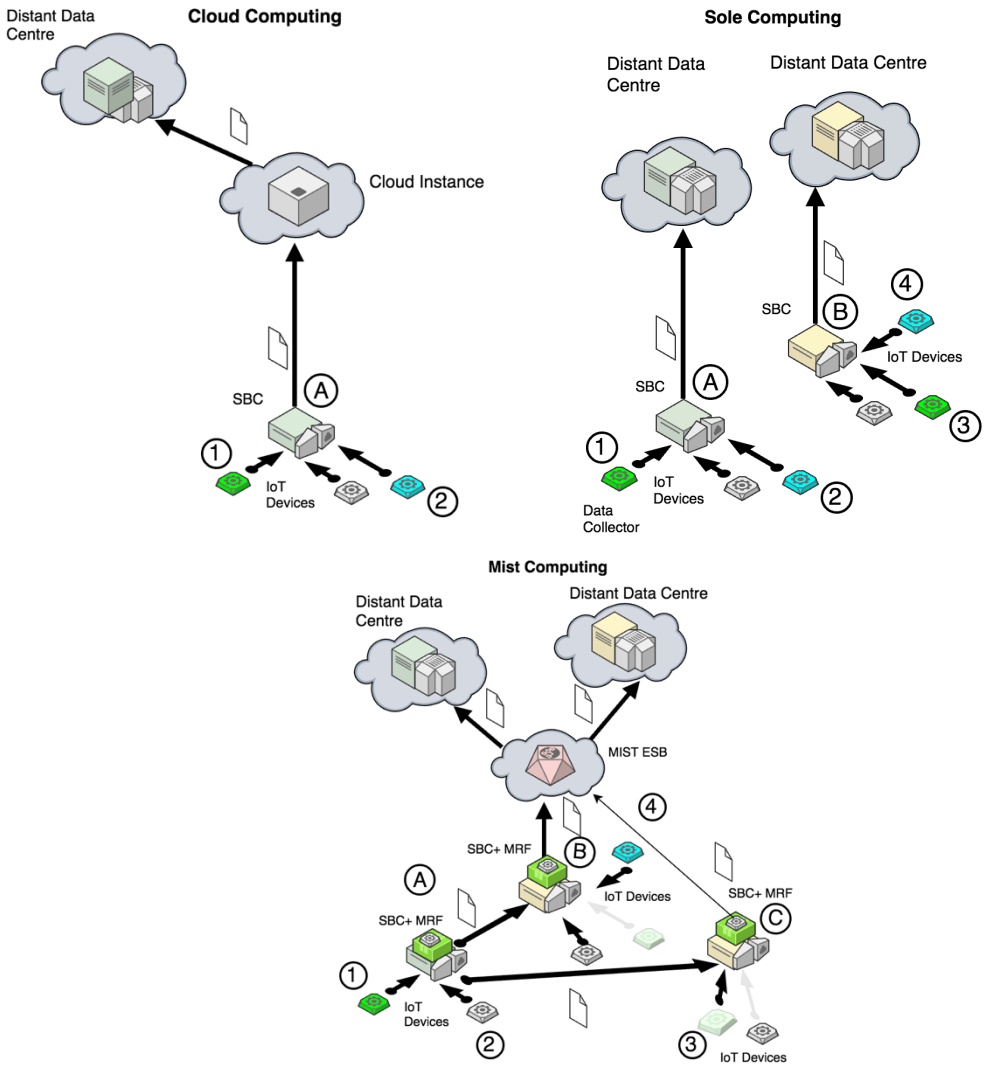


Figure 27: Computing Methods.

4.5 Test Experiments

Two tests experiments for each of the computing methods in section 4.4 (see in Table 2).

Devices/methods	Seeker SBC Node	2 Remote SBC Nodes	Cloud
Sole computing	✓	×	×
Mist computing	✓	✓	×
Cloud computing	✓	×	✓

Table 2: Method and Devices.

During each of the test experiments in Table 2, the following tasks were being executed and the results that were being collected:

- File Deploy

This part of the system involves:

1. Transferring of the listed files to the respective tomcat servlet running with Camunda.
2. Unwrapping of the deployed files.
3. Deployment and auto starting of the Camunda application at the tomcat servlet.

Table 3 describes the description of files being sent

- Docker Image Processing

It's through this part of the system where image processing computation of finding out the most common colour patterns in an image.

- Call Back

When cloud or mist nodes complete execution of the deployed workflow they do send back their response call back that contained processed result to the endpoint provided in the Mist File.

One important hint to be noted is that for tests that used Mist computing, since these tasks were being run parallel at the same time at different participant devices. The mean of values attained from the devices was taken.

File Name	Meta Data
Mist text	1. Description: This file contains all the execution commands. 2. Format: Json. 3. Size: 500b (apx).
Mist war	1. Description: This file contains all the execution workflow engine. 2. Format: war. 3. Size: 3.7mb (apx).
Image Payload	4. Description: This is the image that is being sent in tests that deploy with payload. 5. Format: jpeg. 6. Size: 15mb (to each of the two mist nodes) or 30mb (sole and cloud tests)

Table 3: Files Description.

4.6 Devices and their specifications

Our tests were being carried onto three SBCs and one cloud device. Below are the specifications of all the devices used.

Device	Specification
SBC	CPU: A 1.2GHz 64-bit quad-core ARMv8 CP RAM:1GB Bluetooth: Bluetooth 4.1 Classic, Bluetooth Operating system: Raspbian Jessie Storage: 8 GB
Cloud	Provider: Digital Ocean CPU: 1.7GHz, 1 core processor Droplet: Ubuntu 16.10 x64 Memory: 1 GB Hard disk: 30GB

Table 4: Devices Specifications.

4.7 Test Analysis

In this section, we will explain with help graphical representation generated from test experiments data collected while running the test cases discussed in Section 4.4.

The following observation was made on the data collected when three computing methods namely Sole, Mist and Cloud computing were being used by the sender node device.

The first stage was for the sender node to deploy these three files mist.war, mist.txt and image-payload which are described in Table 3. This deployment was made to the respective receiver node that could dynamically unwrap out the intended execution instructions from the mist.txt and it could immediately deploy the mist.war which did contain the BPMN workflow it to its Camunda engine running on top of tomcat.

As Mist computing, could require the division of the workflow into parts that could be parallel executed at receiver nodes. In our case, the BPMN workflow in figure 22 which was a half of the complete workflow in figure 21 was being executed at each of the Mist receiver nodes.

However, for Sole and Cloud computing since they didn't require parallelism the complete workflow was being used.

BPMN workflow contained the intended logic of execution to be carried out by the receiver and as it was a requirement for us to keep our system light weight and free from heterogeneity. Our workflow did use custom service tasks that used Docker.

Therefore, when these service tasks could be run they would dynamically run the Docker image required, provided from the service task properties. For proof of concept, akaiz/mist-image-processor Docker image was node being executed at the Docker Image processing task which extracted different most common colours from the payload image.

Once the executions were being finished the results were being returned to the respective nodes.

The following sub sections of this thesis contains an overview and interpretation of the data collected while using the three computing methods.

4.7.1 Overall Time comparison

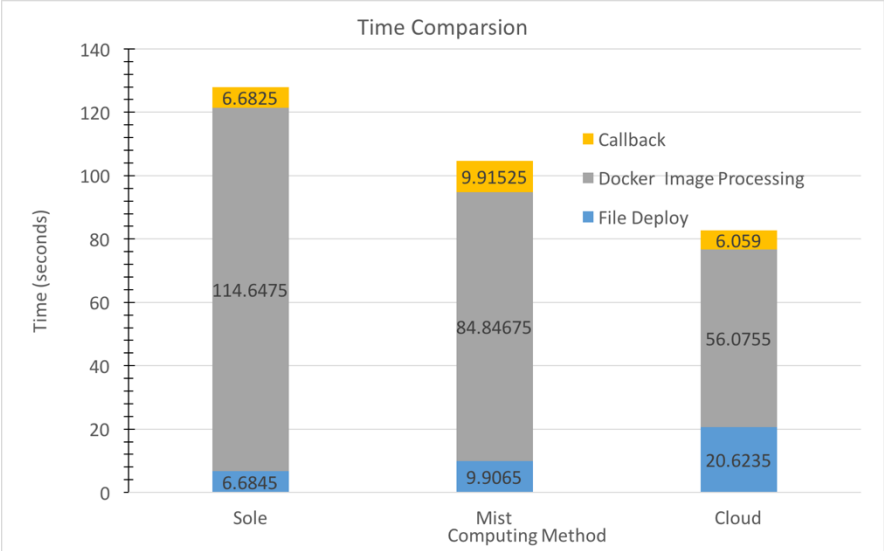


Figure 28: Time Comparison.

Cloud computing had the highest File Deployment time of 20.62 seconds when compare it to Mist and Sole computing which had low values of 9.9 seconds and 6.68 seconds respectively.

This was anticipated as the sender node had to send this heavy payload of 30MB to a remote cloud node in one post request. Faster file deploy to the cloud, would require the node to have good system specifications and high internet bandwidth. In our case, the sender node had limited specifications and with a 3G internet connection, these contributed to the high time taken.

In Mist computing as the workflow and payload were partially redistributed distributed to two Mist nodes with in proximity. Because in this case instead of transmitting the 30MB payload at once the payload and the workflow were broken into two parts. These were deployed parallel to the respective receiver nodes. From our experiments, the average time of 9.9 seconds was recorded.

Image processing was our intended task and once the Camunda workflow was being processed we could see that the high specification of the Cloud enabled it to have less time of 56 seconds compared to Mist nodes which had 84.4 seconds. As Sole computing does all the execution at the sender node it had the highest value of 114.6 seconds.

Comparing the total time taken while using the three computing options. Time of 128 seconds, 104 seconds, 82 seconds for Sole, Mist and Cloud computing. This shows that choosing either Cloud or Mist the IoT edge node device would reduce the total execution time.

4.7.2 Overall CPU usage comparison.

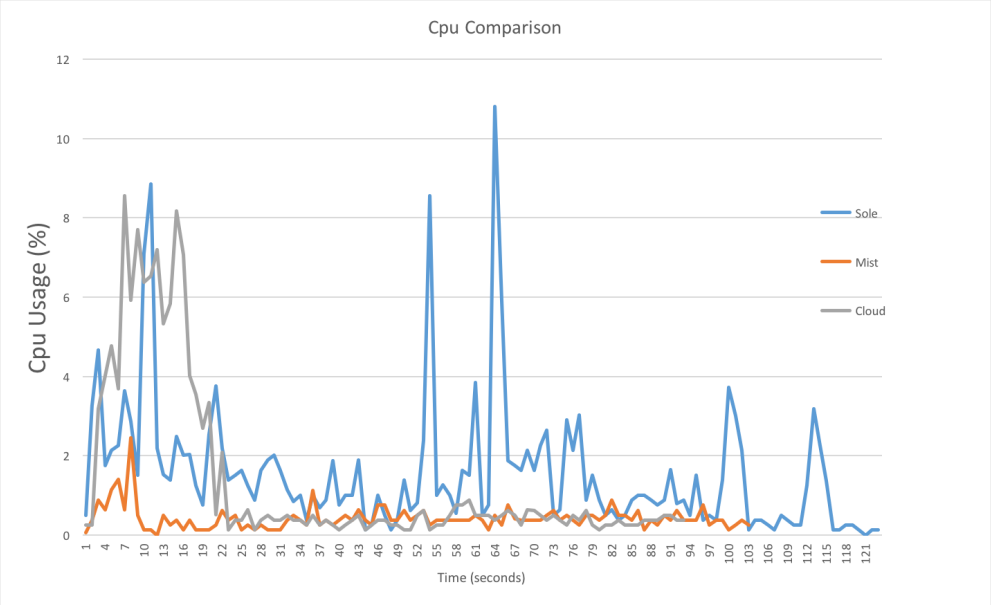


Figure 29: Cpu usage comparison.

The file deployment of a 30MB payload to the cloud receiver, executed with the early 20 seconds, did consume up to 9% of Cpu usage. This is value could rise further as more the value of payload size is increased.

However, the usage of parallel distribution of the workflow and payload, mentioned in section 4.7.1 by Mist computing added less Cpu usage was around 2% during the file deployment stage.

When using Mist or Cloud after the deployment of the workflow to the respective receiver node, the sender node does attain very low Cpu usage values as the entire process would be executed remotely as compared to Sole.

Therefore, among the available options, the usage of Mist computing enables the edge node device to have the lowest usage of Cpu hence this can increase the overall performance of the devices.

4.7.3 Overall RAM usage comparison

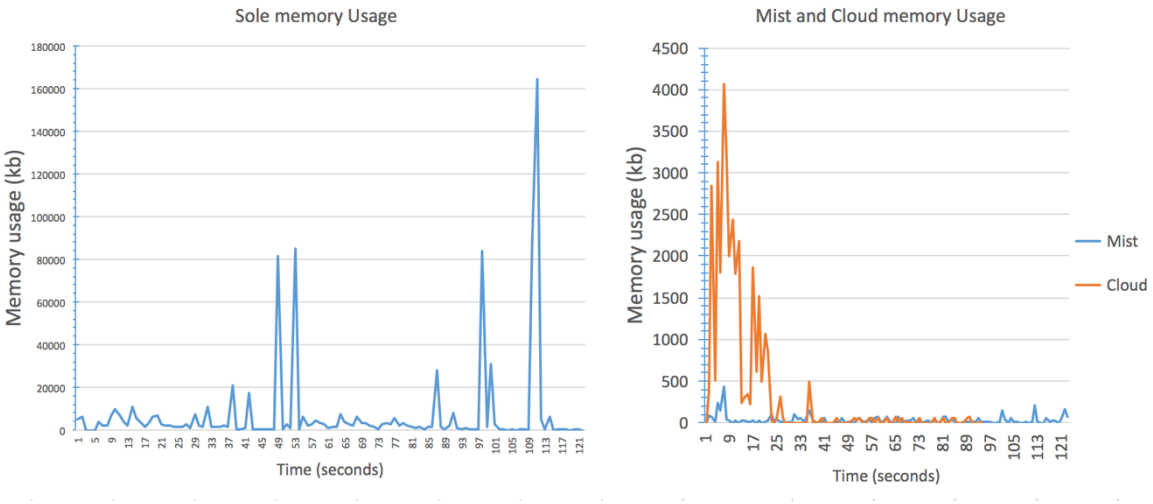


Figure 30: Memory usage comparison.

In figure 30 we can see that Mist and Cloud are far much better than Sole computing when we consider the memory usage of the sender edge node device.

When using Sole computing the node is active carrying complex computation throughout the entire time therefore this resulted to high memory usage increase. It's values sometimes went above 200000KB but for Mist and Cloud computing, there is a negligible memory usage increase once the executions were deployed to their respective receiver nodes.

Using Mist computing at the stage of file deployment gave better results as it used a range of 0 to 500KB memory compared to Cloud which used 0 to 4000KB.

4.8 Discussion

Considering the comparison of the different computing methods in section 4.4, Mist and Cloud computing are good options for edge SBC devices compared to Sole as they do have better total time, memory and CPU usage results.

Total time comparison see section 4.7.1 does favors Cloud computing as it does have less time spent compared to Mist computing this is because two minimal SBC devices were being used as the Mist receivers compared to a high powerful Cloud machine instance (see Table 4).

However, if the number of Mist receivers would be increased the total time for different parts of the execution would reduce thus this will make Mist to have less total time than Cloud.

Therefore, basing on our analysis based on time, Cpu, and Memory usage comparison in section 4.7. IoT edge node devices would highly benefit from using MRF as it not only enables them to quickly carry out executions faster but it also enables them to have low system overhead in terms of memory, Cpu usage.

MRF could also benefit to already existing IoT environments that use Cloud as it would reduce on the payload of data being sent to the Cloud see figure 31. Therefore, when the Cloud data centres do receive this preprocessed data, this will reduce them from being over processing. As it assumed that each Cloud instances at a given point of time may be carrying out multiple executions from IoT edge nodes.

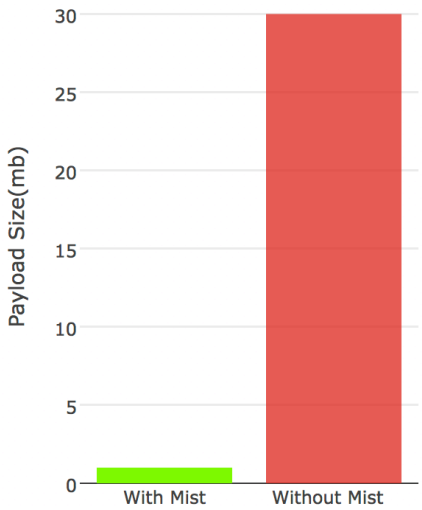


Figure 31: Payload Sent to Cloud.

5 Conclusions

This thesis targeted minimizing interaction and transmission of data between SBC and Cloud data centres, the author approaches this requirement by suggesting that collaborative distribution of workflow processes to nearby edge devices would be a better option because this brings computation to devices that are few hops counts away.

Smooth execution of one workflow process across all devices in the Edge network currently has been greatly hindered by heterogeneity as these devices differ from each other in terms of hardware and software.

The usage virtual containerization technologies (Docker) and business process management technologies (Camunda). Enabled us to have a frictionless collaboration of execution of workflows across edge devices as these deployed light weight workflows would dynamically fetch containers that could carry out a specified implementation need at runtime.

Our containers contained implementation and required dependencies thus this made them to a ready to be executed at any Mist node receiver device thus the heterogeneity factor was being solved as our deployed workflows that use containers could be executed at any device that supports Docker, irrespective of its system architecture and operating system.

From our test experiments, the usage MRF distribution would reduce latency levels and bandwidth usage of the IoT edge devices as most of the heavier tasks would be preprocessed at the proximal Mist nodes. Thus, using this option would even free them from over usage of system resources.

Lastly, this adoption of MRF in IoT networks will enable IoT edge devices to send preprocessed data to Cloud data centres. This will reduce the number continuous heavy system resource consuming executions currently carried out at Cloud data centres on raw data.

5.1 Future Works

1. **Mist Enterprise Service Bus**

   We aim to extend the implementation of Enterprise service bus of the Mist nodes which would be a central control unit of the Mist nodes.

2. **Mist Computing Peer to Peer Service Discovery and Communications**

   We aim to fully implement total service WSAN discovery technologies such as use of WIFI Direct and Bluetooth Low energy.

References

[Amj15] Ann Mary Joy. Performance comparison between Linux containers and virtual machines 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, 2015, pp. 342-346.

[Am13] Arash Asadi, Vincenzo Mancuso.WiFi Direct and LTE D2D in action. 2013 IFIP Wireless Days (WD), Valencia, 2013, pp. 1-8.

[Ash09] Kevin Ashton. That 'internet of things' thing. RFID Journal, 2009. http://www.rfidjournal.com/articles/view?4986.

[Bpp14] Alessio Botta, Walter de Donato, Valerio Persico, Antonio Pescapé. On the Integration of Cloud Computing and Internet of Things. International Conference on Future Internet of Things and Cloud, Barcelona, (2014), pp. 23-30.

[Car15] Carl Boettiger. 2015. An introduction to Docker for reproducible research. SIGOPS Oper. Syst. Rev. 49, 1 (January 2015), 71-79.

[Cccgm07] J. C. Cano, J. M. Cano, C. Calafate, E. Gonzalez and P. Manzoni. Evaluation of the Trade-Off between Power Consumption and Performance in Bluetooth Based Systems. 2007 International Conference on Sensor Technologies and Applications (SENSORCOMM 2007), Valencia, 2007, pp. 313-318.

[Cha14] Kuor-Hsin Chang. Bluetooth: a viable solution for IoT? [Industry Perspectives]. in IEEE, Wireless Communications, vol. 21, no. 6, pp. 6-7, December 2014.

[Cis15] Cisco. Paper, White. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are (2015) [accessed 05 Feb. 2017].

[Cis17] Cisco. IoT http://cisco.com [accessed 15 Feb. 2017]

[Cmfvp16] Antonio Celesti, Davide Mulfari, Maria Fazio, Massimo Villari, Antonio Puliafito. Exploring Container Virtualization in IoT Clouds.2016 IEEE International Conference on Smart Computing (SMARTCOMP), St. Louis, MO, (2016), pp. 1-6.

[Cmlp14] James J. Cusick, William Miller, Nicholas Laurita, Tasha Pitt. Design, Construction, and Use of a Single Board Computer Beowulf Cluster: Application of the Small-Footprint, Low-Cost, InSignal 5420 Octa Board, in arXiv.org, (2014)

[Cnb16] Chii Chang, Satish Narayana Srirama, Rajkumar Buyya. 2016. Mobile Cloud Business Process Management System for the Internet of Things: A Survey. ACM Comput. Surv. 49, 4, Article 70 (December 2016), 42 pages.

[Css13] Daniel Camps-Mur, Andres Garcia-Saavedra, Pablo Serrano. Device-to-device communications with Wi-Fi Direct: overview and experimentation. in IEEE, Wireless Communications, vol. 20, no. 3, pp. 96-104, June 2013.
[Dk13] Fred Douglis, Orran Krieger.Virtualization .in IEEE Internet Computing, vol. 17, no. 2, pp. 6-9, March-April 2013.
[Doc] Docker. https://docs.docker.com. [accessed 23 April. 2017].
[Drmr13] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. Fundamentals of business process management. Springer, 2013.
[Dtbeg15] Kashif Dar, Amir Taherkordi, Harun Baraki, Frank Eliassen, Kurt Geihs, A resource oriented integration architecture for the Internet of Things: A business process perspective, Pervasive and Mobile Computing, Volume 20, July 2015, Pages145-159.
[Ffrr15] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio. An updated performance comparison of virtual machines and Linux containers, 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, pp. 171-172.
[Flr13] Niroshinie Fernando, Seng W. Loke, Wenny Rahayu, Mobile Cloud computing: A survey, Future Generation Computer Systems, Volume 29, Issue 1, January 2013, Pages 84-106, ISSN 0167-739X.
[Frpj14] Bonomi Flavio, Milito Rodolfo, Natarajan Preethi, Zhu Jiang. Fog Computing: A Platform for Internet of Things and Analytics. Big Data and Internet of Things: A Roadmap for Smart Environments in Springer International Publishing Springer International Publishing (2014).
[Gbmp13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems, Volume 29, Issue 7, September 2013, Pages 1645-1660.
[Har16] Hardkernel. Odroid. http://www.hardkernel.com [accessed 17 May 2017]
[Hyp17] What Is Hypervisor? From Whatis.Com (2017) [accessed 4 Feb. 2017].
[Ibm17] IBM, Enterprise Service Bus.https://www.ibm.com/developerworks/library/ar-esbpat1/ar-esbpat1-pdf.pdf. [04 April. 2017].

[Iigkts14] Ismail, Bukhary Ikhwan, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing Yuan Luke, and Ong Hong Hoe. Evaluation of Docker as Edge computing platform. In Open Systems (ICOS), 2015 IEEE Confernece on, pp. 130-135. IEEE, 2015.

[KI10] Karthik Kumar,Yung-Hsiang Lu. Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? in Computer, vol. 43, no. 4, pp. 51-56, April 2010.

[Lcs16] Liyanage M, Chang C, Srirama SN (2016).mePaaS: mobile-embedded platform as a service for distributing fog computing to edge nodes. In: 17th international conference on parallel and distributed computing, applications and technologies (PDCAT-16).

[Mcs16] Jakob Mass, Chii Chang, Satish N. Srirama. Workflow Model Distribution or Code Distribution? Ideal Approach for Service Composition of the Internet of Things, 2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA, 2016, pp. 649-656.

[Mec16] Mobile-Edge Computing. White Paper Mobile and Wireless Networks (2016): 283-306. Portal.etsi.org. Web. 9 May 2017.

[Mlo97] T. Mitchem, R. Lu, R. O'Brien. Using kernel hypervisors to secure applications. Proceedings 13th Annual Computer Security Applications Conference, San Diego, CA, 1997, pp. 175-181.

[Mb16] Roberto Morabito, Nicklas Beijar. Enabling Data Processing at the Network Edge through Lightweight Virtualization Technologies. IEEE International Conference on Sensing, Communication and Networking (SECON Workshops). London, United Kingdom, (2016), pp. 1-6.

[Mpga05] Tommaso Melodia, Dario Pompili, Vehbi C. Gungor, and Ian F. Akyildiz. 2005. A distributed coordination framework for wireless sensor and actor networks. In Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '05).

[Mpga07] Tommaso Melodia, Dario Pompili,Vehbi C. Gungor, Ian F. Akyildiz. Communication and Coordination in Wireless Sensor and Actor Networks in IEEE Transactions on Mobile Computing, vol. 6, no. 10, pp. 1116-1129, (Oct. 2007).

[Nist09] NIST, Definition of Cloud computing v15 (2009), National Institute of Standards and Technology: Gaithersburg Editor. 2009.

[Nmps15] Preeth E N, Fr. Jaison Paul Mulerickal, Biju Paul, Yedhu Sastri. Evaluation of Docker containers based on hardware utilization,2015 International Conference on Control Communication & Computing India (ICCC), Trivandrum, 2015, pp. 697-700.
[Ope] Openhab. Empowering the smart home. http://www.openhab.org. [accessed 04 April. 2017].
[Ope17] OpenFog, (2017). White paper OpenFog Reference Architecture for Fog Computing, http://www.openfogconsortium.org.
[Pahl15] Claus Pahl.Containerization and the PaaS Cloud in IEEE Cloud Computing, vol. 2, no. 3, pp. 24-31, May-June 2015.
[Phmsl16] Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin, Brian Lee. A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters. IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), Vienna, (2016), pp. 117-124.
[Ptjrc15] Jürgo S. Preden,Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid,Emine Calis. The Benefits of Self-Awareness and Attention in Fog and Mist Computing. in Computer, vol. 48, no. 7, pp. 37-45, (July 2015).
[Pt04] H. H. Pang, K. L. Tan. Authenticating query results in edge computing. Proceedings. 20th International Conference on Data Engineering, 2004, pp. 560-571.
[Pmlm15] Petrolo, Riccardo, Roberto Morabito, Valeria Loscrì, and Nathalie Mitton. "The design of the gateway for the Cloud of Things." Annals of Telecommunications (2015): 1-10.
[Ras16] Raspberry Pi. https://www.raspberrypi.org [accessed 17 May 2017].
[Rn16] Flávio Ramalho, Augusto Neto. Virtualization at the network edge: A performance comparison, 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Coimbra, 2016, pp. 1-6.
[Rose04] Mendel Rosenblum.2004. The Reincarnation of Virtual Machines. Queue2,5(July2004),34-40.
[Rsms12] B. B Prahlada Rao, Paval Saluia, Neetu Sharma, Ankit Mittal, Shivay Veer Sharma. Cloud computing for Internet of Things & sensing based applications, 2012 Sixth International Conference on Sensing Technology (ICST), Kolkata, 2012, pp. 374-380.

[Sac13] Meyer Sonja, Ruppen Andreas, Magerkurth Carsten, Internet of Things-Aware Process Modeling: Integrating IoT Devices as Business Process Resources: Advanced Information Systems Engineering: 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013.

[Spg] Spring Boot. https://spring.io/guides/gs/spring-boot. [accessed 25 Apr 2017].

[Sbcd09] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, Nigel Davies. The Case for VM-Based Cloudlets in Mobile Computing in IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, (Oct.-Dec. 2009).

[Soma15] Madakam, Somayya. International Journal of Future Computer and Communication; Singapore4.4 (Aug 2015): 250-253.

[Stan08] John A. Stankovic, When Sensor and Actuator Networks Cover the World.ETRI Journal, vol. 30, no. 5, (Oct. 2008), pp.) 627-633.

[Sw14] Ivan Stojmenovic, Sheng Wen. The Fog computing paradigm: Scenarios and security issues. Federated Conference on Computer Science and Information Systems, Warsaw, (2014), pp. 1-8.

[Uhi05] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, L. Smith. Intel virtualization technology. in Computer, vol. 38, no. 5, pp. 48-56, May 2005.

[Vm15] Vladimir Vujović, Mirjana Maksimović, Raspberry Pi as a Sensor Web node for home automation, Computers & Electrical Engineering, Volume 44, May 2015, Pages 153-171.

[Wb10] Yi Wei, M. Brian Blake. Service-Oriented Computing and Cloud Computing: Challenges and Opportunities. In IEEE Internet Computing, vol. 14, no. 6, pp. 72-75, Nov.-Dec. 2010.

[Wllsd10] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun and Hui-Ying Du. Research on the architecture of Internet of Things. 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), Chengdu, (2010), pp. V5-484-V5-487.

[Zcb10] Qi Zhang, Lu Cheng, Raouf Boutaba. Cloud computing: state-of-the-art and research challenges, Journal of Internet Services and Applications, 1 (2010), pp.7–8, Springer, 2013

[Zwclq10] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, Weijun Qin. IOT Gateway: Bridging Wireless Sensor Networks into Internet of Thing. 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Hong Kong, 2010, pp. 347-352.

Appendix

Test Results

Test Case ID:	1
Test Case Name:	Sole payload available
Description:	We are to collect values of CPU, total time taken and memory at one SBC locally
Preconditions:	Payload already available
Environment	Raspberry pi 3
Status	Success

Test Case ID:	2
Test Case Name:	Cloud payload sent
Description:	We are to collect values of CPU, total time taken and memory when it offloads the workflow to Cloud
Preconditions:	Payload sent with the Workflow process
Environment	Raspberry pi 3 + Digital ocean droplet (Ubuntu)
Status	Success

Test Case ID:	3
Test Case Name:	Mist payload sent
Description:	We are to collect values of CPU, total time taken and memory when it offloads the workflow to SBC nodes when payload was sent
Preconditions:	Payload sent with the Workflow process
Environment	Edge network (3 Raspberry pi 3)
Status	Success

I. Search Structure

Source:	Search parameter	Link
Google Scholar	Internet of Things	Scholar.google.com
Google Scholar	Containerization	Scholar.google.com
Google Scholar	IoT Computing	Scholar.google.com
Google Scholar	IoT Business Process Systems	Scholar.google.com
Springer Link	IoT communication	link.springer.com
ACM Digital	Business Process Management	dl.acm.org

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Isaac Agaba**,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Adaptive Process Distribution at the Edge of IoT using the Integration of BPMS and Containerization,

(title of thesis)

supervised by Chii Chang,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **15.08.2017**