Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 130 (2018) 488–495

Procedia Computer Science

www.elsevier.com/locate/procedia

The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)

Cloud vs Fog: assessment of alternative deployments for a latency-sensitive IoT application

Marcus Gomes, Miguel L. Pardal*

INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa, Portugal

Abstract

Internet of Things (IoT) smart places are systems composed of sensors, actuators and computing infrastructure that acquires data about the surrounding environment and uses that data to improve the user experience of the smart place. For instance, RFID readers can detect a tag approaching and, after the event is processed in a dedicated server, open a door automatically. Many IoT applications are latency-sensitive because actions need to be done in a timely manner. To meet this requirement these applications are usually provisioned close to the physical place, which represents an infrastructure burden because it is not always practical to deploy a physical server at a location. *Utility computing* in the *Cloud* can solve this issue but the latency requirements must be carefully assessed. *Fog* computing is a concept that brings the cloud close to devices at the edge of the network, aiming to provide low latency communication for applications and services. The present work implemented a provisioning mechanism to deploy a "smart warehouse" IoT application according in utility computing platforms: Cloud and Fog. We compared the event latency performance of both approaches and the results show that a fog deployment is more adequate for the considered IoT application.

© 2018 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the Conference Program Chairs.

Keywords: Utility computing, Cloud computing, Fog computing, Provisioning, Internet of Things, RFID

1. Introduction

Internet of Things (IoT) applications are composed of physical objects that are continuously connected to the virtual world and can act as physical access points to Internet services[1]. Many times these systems require *low-latency interactions* with users and environments, which traditionally implies that at least part of an IoT application needs to be tightly bound to the local infrastructure of the interacting environment.

The objective of this work is to determine if the dedicated, local infrastructure traditionally required for supporting the application, can be replaced by a form of *utility computing*[2], but still fulfill the low-latency requirements of IoT smart places. Our efforts will use a "smart warehouse" that relies on the RFID technology[3] where products are tagged

* Corresponding author. Tel.: +351.213100300; fax: +351.213145843.
E-mail address: miguel.pardal@tecnico.ulisboa.pt

with Radio-Frequency IDentification (RFID) tags that can be identified by RFID readers. Through the data collected by these readers is possible to gather information for the business operation of the smart place.

2. Related Work

Until recently, *utility computing* and *cloud computing* were synonymous. The traditional delivery models allow use of shared computing resources at different abstraction levels[4]: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). An example of PaaS offering for RFID-based IoT applications is Fosstrak (Free and Open Source Software for Track and Trace – `http://fosstrak.github.io/`), an EPCglobal Network compliant RFID software platform that was developed by Floerkemeier et. al[5]. Fosstrak is composed by several modules: *Filtering & Collection Server* (FCServer) – called Application Level Events (ALE) in the EPCglobal standards – that collects raw data, *Capturing Application* that performs application-specific event interpretation, and *EPCIS Repository* that stores persistent events.

The Fog Computing[6] is a recent paradigm that aims to bring the cloud close to the "edge of the network", bringing the cloud close to the ground - hence the *fog* meteorological analogy. The Fog should be able to meet the requirements of several applications that the traditional clouds are not able to accomplish.

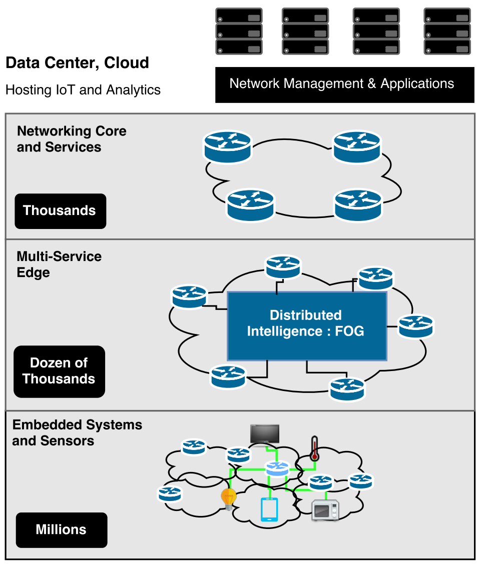

Data Center, Cloud
Hosting IoT and Analytics
Network Management & Applications
Networking Core and Services
Thousands
Multi-Service Edge
Distributed Intelligence : FOG
Dozen of Thousands
Embedded Systems and Sensors
Millions

Fig. 1. The Internet of Things and Fog Computing (adapted from Bonomi et. al (2012)).

Bonomi et. al[6] presents the architecture of a Fog Computing platform, illustrated in Figure 1. The distributed infrastructure of the Fog comprises data centers, core of the network, edge of the network and end devices. *Multi-Service Edge* is the lowest tier of the Fog and it is responsible for performing machine-to-machine (M2M) interactions. It collects and process the data from the *Embedded Systems and Sensors* tier, issues commands to the actuators and also filters the data that is locally consumed and sent to the higher tiers. *Core Networking and Services* tier is responsible for providing network services that are used to exchange data between sub-networks. This tier also provides security services as well Quality-of-Service and multicast services for the applications. Since the interaction time between the different tiers can range from seconds - e.g. "real-time" analytics - to days - "data warehouse" analytics - the Fog must support several types of storage, from ephemeral storage at the lowest tiers to semi-permanent at the highest tier. At the high tiers, the geographical coverage and the latency both increase[7]. The global coverage is given by central repositories for persistent data that can be used to perform business analytics.

Configuration Management (CM) tools are software management tools that allows to automate and specify the deployment of applications in utility computing platforms. The system resources and their desired state are described and the CM tool is responsible for enforcing the desired state. For instance, CM tools allows the automation of the provisioning of physical and virtual machines, perform dependency management of software components and automate management tasks. Currently, there are several solutions to perform configuration management of software, where the most relevant are *Chef* (`https://www.chef.io/`), *Puppet* (`https://puppetlabs.com/`), *Ansible* (`http://www.ansible.com/`) and *Salt* (`http://saltstack.com/`).

3. Solution

A provisioning solution, shown in Figure 2, was developed for the "smart warehouse" application using Configuration Management (CM) tools. After the provisioning policies are defined and configured, the Orchestrator uploads

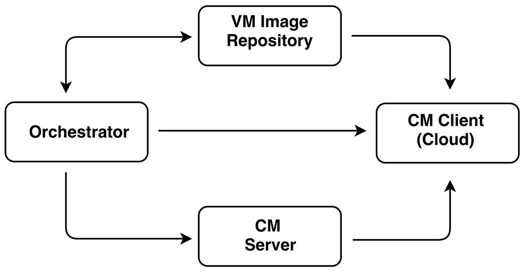


Fig. 2. Provisioning mechanism conceptual architecture.

them to its respective remote repositories (CM Server and VM Image Repository). When the provisioning request is performed - through a configuration management interface provided by the Orchestrator - the configuration management client (CM Client) pulls the polices from the configuration management server (CM Server) that is responsible for maintaining a consistent state of the provisioned nodes. In order to enforce the polices, the CM Client pulls the VM images or Docker containers from a central repository containing the required software components to deploy the application, and then performs its configuration. After provisioning the infrastructure, the CM client periodically polls the CM server in order to determine if its current state is consistent with the most recent policy.

The implementation of the provisioning mechanism relies on the *Chef* tool. Chef has *recipes* that describe infrastructure as code. The *knife* command-line tool executes the necessary actions and provides an interface between the local Chef repository and the Chef server. We chose Chef instead of its competitors - i.e. Puppet and Ansible - mainly because of this *knife* tool as it is very powerful and allows us to interact with our entire infrastructure. For example, with *knife ssh* it is possible to execute a command on a certain number of nodes in our environment. Also, if we change the role configuration that is assigned to a set of nodes, knife is able to update all of them to the most recent policy with a single command.

3.1. Smart Warehouse Deployment

The main goal of the "smart warehouse" application deployment is to guarantee that the latency requirements of these applications are fulfilled.

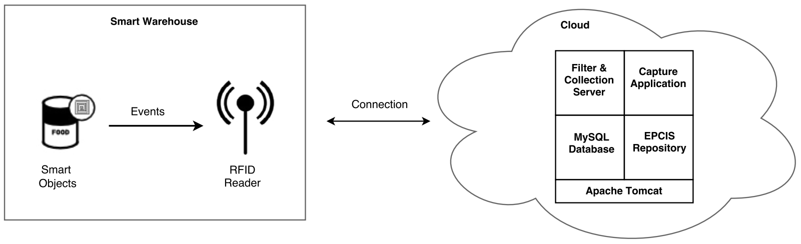


Fig. 3. Cloud deployment: smart warehouse technological architecture.

For the *cloud deployment*, illustrated in Figure 3, all the Fosstrak components are provisioned in a single virtual machine. The Fosstrak FCServer, EPCIS repository and Capture application require an Apache server to run. The EPCIS repository is connected to a MySQL database that stores the event data. The "smart warehouse" can be connected to the cloud through a wired (e.g. ADSL or Fiber-optic) or a wireless network connection (e.g. Wi-Fi or 3G).

For the *fog deployment*, illustrated in Figure 4, part of the Fosstrak components are provisioned closer to the "smart warehouse". This is in contrast with the cloud deployment, where all the provisioning is done at a long distance from the location, e.g. hundreds of kilometers away. The Fosstrak components are provisioned across the Core and the

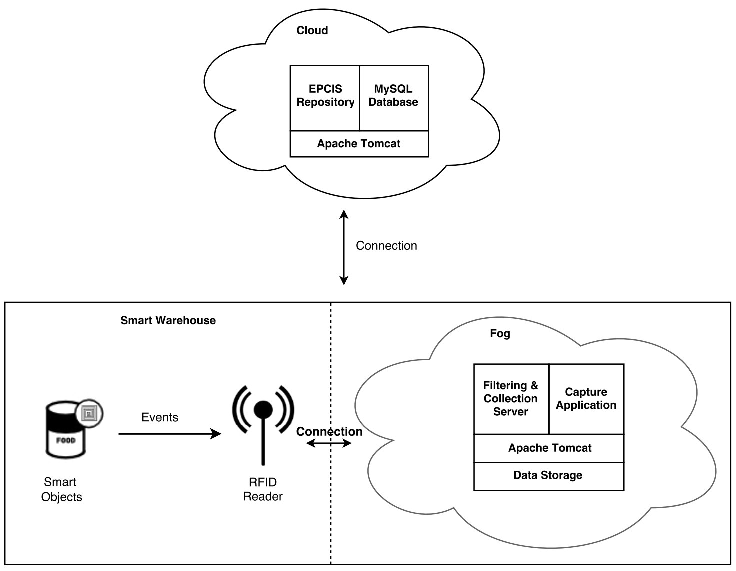


Fig. 4. Fog deployment: smart warehouse technological architecture.

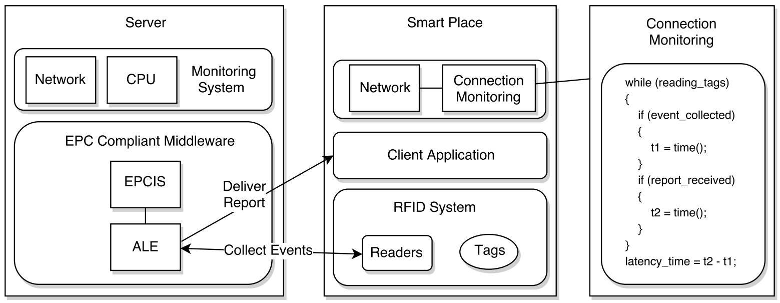
Edge (see architecture depicted in Figure 1). At the Core, the Electronic Product Code Information System (EPCIS) repository is deployed and running on top of an Apache Tomcat server. The repository is connected to a MySQL database, which stores the event data. At the Edge, the Filtering & Collection Server (FCServer) and the Capture application are deployed on a Tomcat server. The Capture application sends the events collected by the FCServer to the EPCIS repository through the *EPCIS Capture Interface* - via HTTP requests. Again the network connections can be wired or wireless.

4. Evaluation

The main goal of the evaluation is to compare the cloud and fog deployments and measure which one is more suitable to fulfill the latency requirements of the "smart warehouse" application. The *response time* between an event that occurs in the "smart warehouse" and the corresponding action that is triggered in the physical space is measured. The detailed instrumentation required for collecting the measurements is shown in Figure 5.

The FCServer module is responsible for collecting and processing the reader events. The collection and processing of reader events is performed according to an *Event Cycle* specification. The *Event Cycle* is a set of periodical cycles where the FCServer collects events from the RFID readers. The data about the *Event Cycle* is delivered to the client application through a report. The information in the report is used to trigger an action, in the warehouse such as "open" or "close" a door.

The network connections were monitored with the *tcpdump* (`http://www.tcpdump.org/`), a command-line tool that allows the monitoring of the packets that are being transmitted or received over a network. Through the logs produced by this tool, we are able to determine how the connection time was spent.



```
while (reading_tags)
{
    if (event_collected)
    {
        t1 = time();
    }
    if (report_received)
    {
        t2 = time();
    }
}
latency_time = t2 - t1;
```

Fig. 5. Latency measurement approach.

4.1. Setup

To perform the evaluation experiments we chose Amazon Web Service (AWS) Elastic Cloud Computing (EC2) instances running the Amazon Linux Amazon Machine Image (AMI) operating system. The Virtual Machines (VMs) presents a configuration with a 2.5 *GHz* single-core processor with 1 *GB* of RAM.

In the fog deployment configuration, the experiments were conducted in a VM with a 2.6 *Gigahertz (GHz)* dual-core processor with 2 *Gigabyte (GB)* of Random-access memory (RAM) and running the *Linux Ubuntu 14.04.2 LTS* operating system. The smart warehouse was connected to the cloud and fog through a Asymmetric Digital Subscriber Line (ADSL) connection with a bandwidth of 10 Mbps.

Regarding the software components, the application stack is composed of *Apache Tomcat 7.0.52.0* with *Java* version *1.7.0* update *79*. The Fosstrak versions were: *FCServer* version *1.2.0*; *Capture Application* version *0.1.1*; and *EPCIS Repository* version *0.5.0*. Furthermore, the EPCIS Repository was connected to a *MySQL database server* version *5.5*. The Rifidi Emulator used to emulate the RFID readers was in version *1.6.0*.

The experiments performed in our evaluation were based on the scenario and data from RFIDToys[8], that provides low level RFID readings recorded from a warehouse demonstrator where a robot transported products identified by RFID tags through a physical space.

To evaluate the latency we consider the moment in the RFIDToys dataset when the robot stops during 5 seconds in front of the door and then continues on its way. The door must be opened before the robot starts to move again. To perform the simulation we defined two different specifications (*ECspec*) for the *Event Cycles* of the FCServer module, *Full Event Cycle* with the default value of 10 seconds and a *Half-period Event Cycle* with a 5 second value, aligned with the robot waiting time.

The evaluation of the event latency for the proposed approaches was performed in two steps. First, we determine during a *Event Cycle* how much time the FCServer is processing an event and how much time the module is in an idle state. Furthermore, we determine how much time each stage of the event processing takes: (i) *Upload Time*; (ii) *Tag Processing Time*; (iii) *Filtering & Aggregation Time*; (iv) *Report Creation Time*; and finally (v) *Response Time* (Download).

4.1.1. Cloud-based warehouse latency

The behavior expected with a shorter *Event Cycle* specification is that the event latency presents a better overall performance. According the obtained results it is possible to observe that the event latency decrease from 8.244s to 4.266s. The values for the network latency improved when the FCServer is configured with the faster *ECspec*, close to ≈ 65% of improvement for the *Upload Time* metric - from 0.294s to 0.103s - and ≈ 40% for the *Response Time* metric - from 0.228s to 0.149s. The values for the time where the FCServer remains in an idle state also presented a significant improvement, from 7.346s to 2.569s. The value for the *Tag Processing Time* increased ≈ 1000% when the FCServer is configured with *Half-period ECspec* - from 0.002s to 0.024s. The value for the *Filtering & Aggregation Time* metric increased ≈ 300% - from 0.370s to 1.490s.

a) Overall latency breakdown.

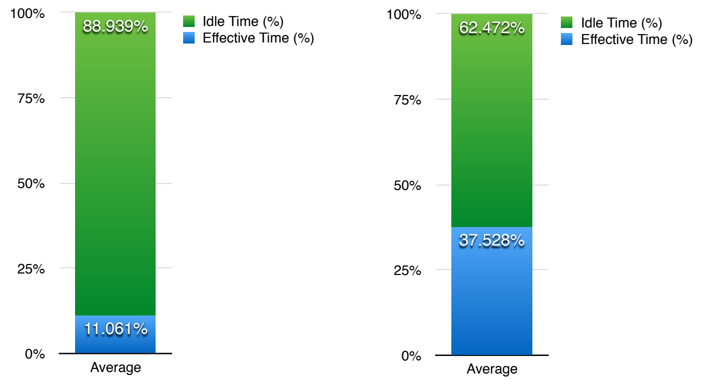


Fig. 6. Full and Half-period Event Cycle for Cloud deployment: overall latency breakdown.

In both *Full ECspec* and *Half-period ECspec*, shown in Figure 6, the FCServer module is in an idle state during most of the time. Also, it is possible to observe that the *ECspecs* affected the percentage of time where FCServer is processing the events (*Effective Time*) and where is in an idle state (*Idle Time*). With the *Full ECspec* the FCServer remains ≈ 89% of the *Event Cycle* period in an idle state while when configured with the *Half-period ECspec* this value decreases to ≈ 62%. This means that during the *Event Cycle* period the FCServer module can be in an idle state during 9 seconds when configured with the *Full ECspec* and for 3 seconds with the *Half-period ECspec*.

b) Effective time breakdown. Figure 7 presents the how much time is spent in each phase of the pipeline when the FCServer is configured with *Full ECspec* and with the *Half-period ECspec*.

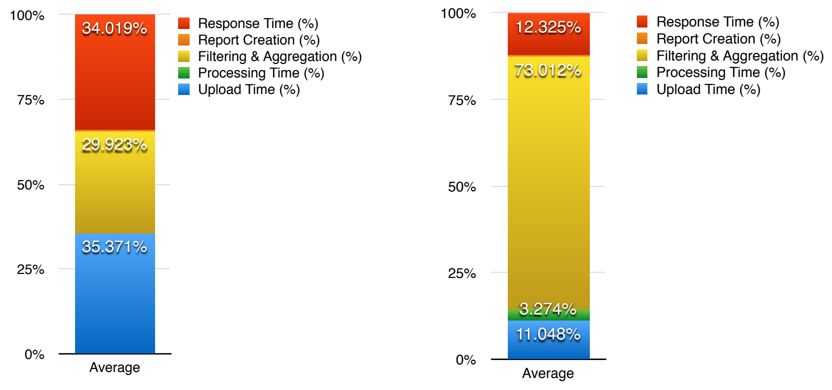


Fig. 7. Full and Half-period Event Cycle for Cloud deployment: effective time breakdown.

Comparing the obtained results, it is possible to observe that time breakdown is evenly distributed between the *Upload* (≈ 35%), *Filtering & Aggregation* (≈ 30%) and *Response* (≈ 34%) stages when the FCServer module is configured with the *Full ECspec*, while the *Tag Processing* and *Report Creation* stages represents a small percentage of the total time, less than ≈ 1%. When the FCServer is configured with the *Half-period ECspec*, the *Filtering & Aggregation* stage is the most time consuming, representing close to ≈ 73% of the total time. As when configured with the *Full ECspec*, the *Upload* and *Response* stages presents similar results, respectively close to ≈ 11% and ≈ 12%. Regarding the *Processing* stage, the time required to process the event data increased from less than ≈ 0.3% to ≈ 3%. The *Report Creation* stage presented the same values as the *Full ECspec* configuration (≈ 0.3%).

4.1.2. Fog-based warehouse latency

As in the previous experiment the event latency presented a better overall performance for the shorter *ECspec*. According the obtained values we observed that the event latency improves in a significant way - from 7.450s to 4.250s. This result is achieved thanks to the improvement in the latency of at the *Filtering & Aggregation Time* by ≈ 52% - from 2.530s to 1.230s - and the amount of time that FCServer is in an idle state - from 4.944s to 2.747s. Regarding the network latency, the values for the *Upload Time* and *Response Time* improved 1ms for both metrics. However, when configured with a faster *Event Cycle* specification the tag processing time presented an inferior performance, where time to process the event data increases ≈ 470% - from 0.049s to 0.279s. Also the report creation time increased 300% - from 0.001s to 0.003s.

a) Overall latency breakdown. Figure 8 shows the latency breakdown for an event when the FCServer module is configured with the *Full ECspec* and when it is configured with the *Half-period ECspec*.

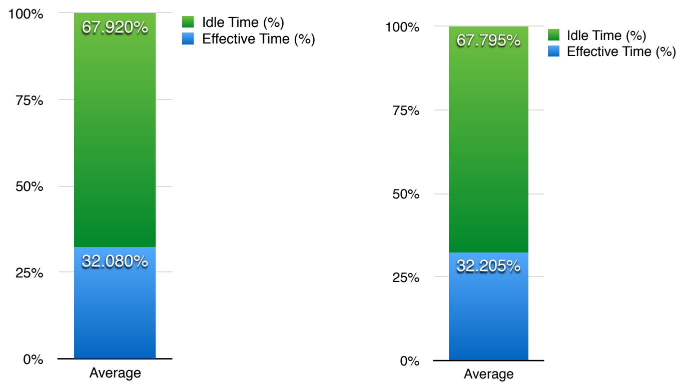


Fig. 8. Full and Half-period Event Cycle for Fog deployment: overall latency breakdown.

Comparing the graphs for both *ECspecs* is possible to conclude that during most of the time of an *Event Cycle* the FCServer module is in an idle state (*Idle Time*) - close to ≈ 68% in both configurations - while in the remaining time the FCServer is processing the event that was collected (*Effective Time*). Considering the duration of the *ECspecs* it means that in average when the FCServer is configured with the *Full ECspec* the module can be in an idle state during 7s while with the *Half-period ECspec* this idle state can last for 3 seconds.

b) Effective time breakdown. Figure 9 presents the time breakdown for each stage of the pipeline when the FCServer is configured with *Full ECspec* and with the *Half-period ECspec*.

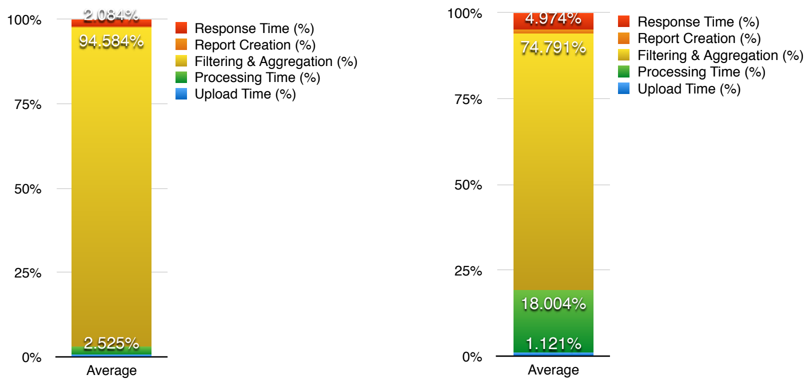


Fig. 9. Full and Half-period Event Cycle for Fog deployment: effective time breakdown.

It is possible to observe that the *Filtering & Aggregation* stage is the most time consuming for both *Event Cycle* specifications. With the *Full ECspec* this stage occupies close to ≈ 95% of the total time, while with the *Half-period ECspec* this value is close to ≈ 75%. The reason for this difference is in the *Tag Processing* stage. With the *Full ECspec* the time for processing the event data represents close to ≈ 2.5% of the total time while with the *Half-period ECspec* this value is close to ≈ 18%. The *Upload* and *Response* stages together represents a small percentage of the time spent to process the event - close to ≈ 5% for both specifications - while the percentage of time to create the reports represents less than ≈ 1% of the total time.

4.2. Discussion

Regarding the *latency* metric, the fog-based deployment presented the best results, but with a small difference when compared with the cloud deployment. According to the results, both deployments meet the requirement of having a latency time lower than the robot wait time, i.e. the warehouse door only will be able to open in time for the robot to pass. But only when the FCServer is configured with the Half-period ECspec (5 seconds). The default configuration value of 10 seconds does not allow meeting the latency requirement. The experimental methodology was able to assert this configuration inadequacy in a clear way.

The values for the *Upload* and *Response* (Download) metrics presented a substantial difference, where the fog deployment is the best one. This can be explained by the proximity to the physical location. However, for the time scale considered in the RFIDToys dataset, where the robot stops for 5 seconds at the door, the difference between cloud and fog deployments is irrelevant in this case.

5. Conclusion and Future Work

The present work explored alternative deployments of an IoT application for a "smart warehouse" using RFID technology. A fully automated provisioning solution using *Chef* tools was built for this purpose. One approach deployed the application in a typical *cloud*. The other approach deployed the application in a *fog*-like infrastructure, where part of the components are at the Edge tier and the rest are at the Core tier. The goal of the exercise was to determine if a cloud-based approach is able to meet the latency requirements of the applications, given that low-latency is an often considered an essential requirement of many IoT applications. With the followed methodology we were able to compare the event latency performance for both cloud and fog deployments. The obtained results show that the event latency performance presented better results when the application was deployed according the fog-based approach. However, for the specific case of the considered "smart warehouse" application with the RFIDToys dataset, both deployments were able to meet the requirement.

Regarding the system evaluation, it was performed only in AWS EC2 instances in the closest geographical location. For the future is important to evaluate our solution in other regions and with other cloud providers to compare performance. Also, in the performed experiments we considered only a particular situation – a robot approaching a door. In future work, we want to evaluate the latency performance in more varied latency-sensitive situations. For the specific case considered, we saw that cloud deployment was able to satisfy the requirements of the IoT application, but it remains to be seen if it is able to cope with more demanding scenarios, such industrial settings in the IIoT (Industrial Internet of Things). Using the tools developed for this work, we will have a head start to perform latency-sensitive tests for more applications in diverse scenarios.

Acknowledgements This work was supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID)

References

1. Mattern, F., Floerkemeier, C.. From the internet of computers to the internet of things. In: *From active data management to event-based systems and more*. Springer; 2010, p. 242–259.
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., et al. A view of cloud computing. *Communications of the ACM* 2010;**53**(4):50–58.
3. Want, R.. An introduction to RFID technology. *Pervasive Computing, IEEE* 2006;**5**(1):25–33.
4. Zhang, Q., Cheng, L., Boutaba, R.. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications* 2010;**1**(1):7–18.
5. Floerkemeier, C., Roduner, C., Lampe, M.. RFID application development with the Accada middleware platform. *Systems Journal, IEEE* 2007;**1**(2):82–94.
6. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.. 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, p. 13–16.
7. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.. Fog computing: A platform for internet of things and analytics. In: *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer; 2014, p. 169–186.
8. Correia, N.. *RFIDToys: A Flexible Testbed Framework for RFID Systems*. Master's thesis; Instituto Superior Técnico; Portugal; 2014.