COORDINATION MODELS FOR INTERNET OF THINGS

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By

JUAN PABLO SANCHEZ GUERRERO

PERMISSION TO USE

In presenting this thesis/dissertation in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis/dissertation in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis/dissertation work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis/dissertation or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis/dissertation.

Requests for permission to copy or to make other uses of materials in this thesis/dissertation in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada S7N 5C9

ABSTRACT

In constrained environments, there is a variety of devices like sensors and actuators with limited computation power or energy that form an Internet of Things (IoT) system. When processing complex tasks is required, those devices send the data to the cloud and obtain the result later. However, the IoT system could process complex task if more devices work together, sharing computational resources and cooperating. This cooperation can be achieved using a coordination model that distributes the load among the different devices based on a set of parameters, laws and defined entities. This research implements and evaluates a data-oriented coordination model with three variations for Internet of Things (IoT). It also presents, implements and evaluates a new process-oriented coordination model that can make constrained environments much more effective and allow the processing of more complex tasks closer to the network. The development of all the coordination models was focused on using the system's computational resources effectively. As IoT is a heterogeneous field, devices with more power can process more complex tasks, creating an uneven but adequate load distribution. Various experiments were conducted to evaluate the performance of each model using one and two workers. The results showed that every coordination model works effectively when distributing the load among more workers. For the process-oriented model, implementing some CoAP features allowed the system to perform better when repetitive tasks are required.

ACKNOWLEDGMENTS

I want to thank my supervisor, Dr. Ralph Deters. In the last two years his guidance, supervision, and feedback were crucial for my research and the development of the present thesis. Thank you for trusting in me and helping me to learn and become a better computer scientist.

I also want to thank my wife Diana for helping me with everything she could during this journey, your support, trust, and love gave me the motivation to complete this achievement. I admire you for everything you do, and this achievement is also yours.

Thanks to my family, my mom, sister, and niece who always believed in this project and supported me. You were always a source of inspiration and help in the hardest moments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
COAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
CPU	Central Processing Unit
CRUD	Create-Read-Update-Delete
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
IoT	Internet of Things
IWIM	Ideal Worker Ideal Manager
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
M2M	Machine-to-Machine
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
TPS	Transactions per Second
UC	Ubiquitous Computing
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WWW	World Wide Web
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

IoT (Internet of Things) is a modern paradigm that connects heterogeneous physical devices that have direct contact with the real world [1]. These devices can be small smart components, sensors or actuators. Although these devices are very diverse, they all tend to be smaller, cheaper and use less energy. Therefore, they have fewer resources (CPU power, ram, storage) and sometimes limited energy available. The lack of resources limits the complexity of tasks they can handle and demands them to use their available resources efficiently.

Figure 1-1. IoT search interests[2]

According to the IoT trend, more constrained devices are going to be deployed in the near future [3] (Figure 1-2). Moreover, more complex systems will be developed, and their functions will be crucial for different fields like health, industry, smart homes, smart retail and more. Because of that, in this environment, optimizing the use of resources is necessary; using resources inefficiently could cause the system to fail at some point. For instance, if a constrained device is given a too complex task it cannot handle, it could return an unexpected result or one that is not in the defined quality of service times.

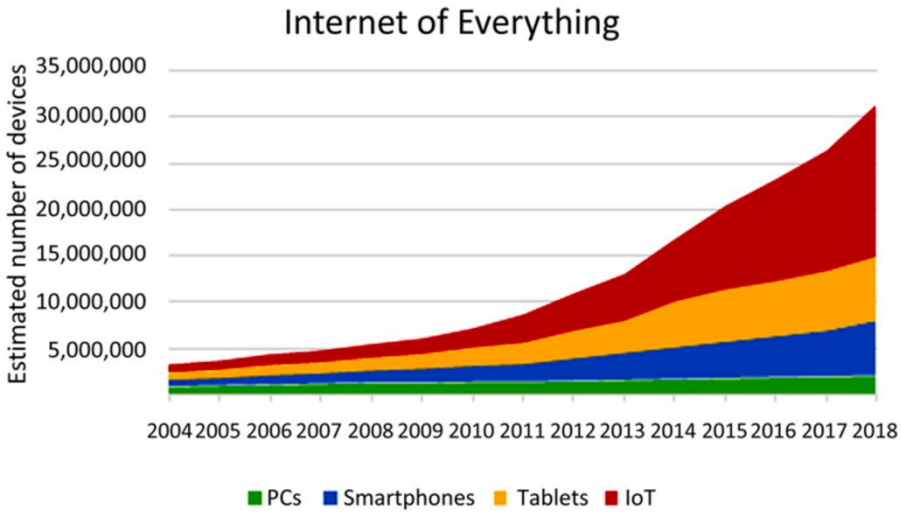

Figure 1-2. IoT trend forecast[4]

Within an IoT system, significant ammounts of data are being harvested, stored and processed, so a device malfunction could cause the entire system to break because of data losses or connectivity issues. That is why QoS (Quality of Service) times are defined. To address this problem, many IoT systems are designed as sensors collecting data and sending it to the cloud to be processed. However, sending all the information to the cloud is not always the most efficient solution. When a device in the system can handle some task, sending it to the cloud to get the result later is wasteful. The closer the data is to the system, the faster the device is going to function because of communication overhead. Having a heterogeneous structure with a diversity of devices that have different capabilities and free resources creates another scenario: the one in which a component is not able to process a task, but another one within the system is. That can happen because the device has more resources or is less busy at a certain moment. The ideal scenario would be one in which tasks are created for each device according to its characteristics, and such tasks would be assigned depending on the available resources of every device. This system could be understood as **fog computing**, in which, with a coordination model, devices can work together and use their resources most efficiently. This cooperation would improve the response times of the system and reduce the probability of having overloads.

Implementing a coordination model would allow devices to "borrow" or "lend" resources from and to other devices. However, due to the IoT characteristics, it is a challenge to design a coordination model that does not generate excessive work. Devices should be able to communicate the tasks that need to be done without spending more resources than performing the actual task would take.

Although much research has been done in the area of coordination models [5], and technologies have been developed specifically for coordination and mobility issues within mobile and constrained environments like Lime (Linda in a Mobile Environment) [6]. More research can be done in the IoT environment where devices have different characteristics and limitations. This research introduces the idea of using coordination models within IoT and evaluates process and data oriented approaches.

The remaining parts of the present document are organized as follows:

- **Chapter 2 – Problem definition:** Explains some restrictions the IoT environment have and describes the problem the current work intends to address.
- **Chapter 3 – Literature review:** Reviews the relevant technologies and paradigms for solving the problem exposed in chapter two. Explores the coordination models already developed, how they are divided and their objective.
- **Chapter 4 - Architecture:** Describes the design of the implemented and developed models and gives an introduction to the way they work.
- **Chapter 5 - Implementation:** Explains more in detail how the coordination models developed work, their characteristics, restrictions, and advantages.
- **Chapter 6 - Evaluation:** Shows the results of the performance testing done to the coordination models and compares them.
- **Chapter 7 – Conclusions and Future Work:** Concludes the present work according to the obtained results and describes the next steps of this research.

CHAPTER 2

PROBLEM DEFINITION

Due to the features the IoT environment has, ways for allowing constrained devices to cooperate in a stable and scalable manner are required. A typical IoT system (Figure 2-1) is comprised of a constrained environment and other more robust components that are usually on the edge of the network or the cloud [7]. Within the constrained environment, most devices have limited resources and power. When these systems require computation to solve complex tasks, they usually send the data to the cloud and wait for the result. Nevertheless, the further the processing is from the network; the more time is going to take to obtain results.

As these devices tend to be smaller and cheaper, the system can easily have a significant number of them. It becomes vital for those devices to have ways to coordinate and solve complex tasks that should not be sent to the cloud.

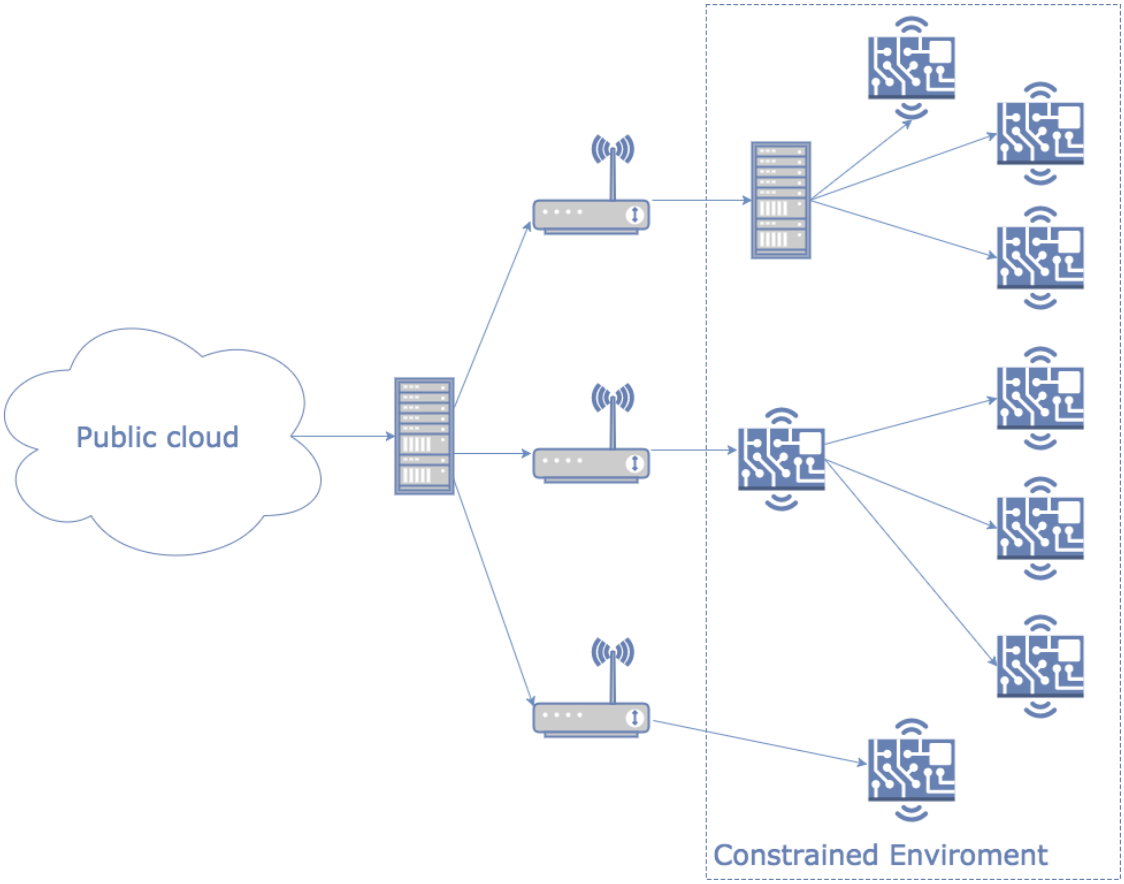

Figure 2-3. IoT and fog architecture

For constrained environments to be able to process more and more complex tasks, a stable coordination that allows devices to share their resources efficiently is required. As the communication with the cloud is not always reliable, bringing computation to the edge of the network that allows constrained devices to coordinate is a necessity.

This research aims to design, develop and evaluate different coordination models specifically for constrained, low-bandwidth environments. Implementing coordination models for the IoT has some challenges:

- Reducing the data transfer costs is a key factor in the coordination model. One of the objectives is to reduce response times by using resources more efficiently. This means information should be transferred to powerful devices that can

process it. However, this transmission should be fast enough, so the response time of sending the data and getting the result is less than processing it locally. The coordination model only represents an improvement if the communication costs are kept at a minimum.

- Designing the coordination model has to take into account the IoT characteristics to make it affordable and efficient. One of the advantages of implementing a coordination model is to avoid sending data to the cloud. The model has to ensure the response times are within the defined QoS, so it can run on constrained devices and allow devices to engage and disengage easily.
- The model needs to implement a high level of fault-tolerance that allows the entire system to work stably even when errors occur, or devices stop responding. Guaranteeing availability and data integrity is a key point the model should address by being flexible and dynamic. Nevertheless, it is important that this dynamism does not generate more work within the system that affects the QoS mentioned above.
- As heterogeneous devices comprise the IoT system, to use the system resources efficiently, the model should distribute the tasks evenly depending on the characteristics of every component. Some devices will have more power than others, which means they can handle more complex tasks.

According to the aforesaid description and challenges, the central question of this research is:

How can constrained devices efficiently coordinate with each other to share computational resources and create a fault-tolerant IoT environment?

CHAPTER 3

LITERATURE REVIEW

The following are the concepts related to designing a coordination model for IoT:

- Internet of Things and M2M: It is important to understand very good what this environment is, its characteristics, challenges, and opportunities. This understanding allows this research to find the specific points of contribution and key factors when designing and implementing a coordination model.
- COAP (The Constrained Application Protocol): The communication protocol designed for IoT that aims to reduce data transfer costs based on simple paradigms like HTTP or REST. As the coordination model proposed in this research aims to keep communication costs at its minimum, we review this protocol.
- Coordination Models: This area has been widely researched in the past, and it is very important how coordination models are designed and their characteristics form a suitable and complete model for IoT.
- RESTful Services: The proposed communication model of this research uses a RESTFul architecture for many reasons explained later. Understanding how this technology works and how it is designed allows the model to use it appropriately.
- Fog Computing: This paradigm is defined to extend the services offered in the cloud to the edge of the system to reduce latency. In a fog computing environment, there are various distributed devices (not necessarily constrained). This is an architecture very similar to a primary IoT environment, which is why exploring this concept is important to design a coordination model.
- Blockchain: A new technology that is still being developed and offers some attractive characteristics and enables new kinds of distributed software architectures, for the IoT environment, it can be used as a software connector which creates a shared data space that allows devices to communicate.

3.1. Internet of Things and M2M

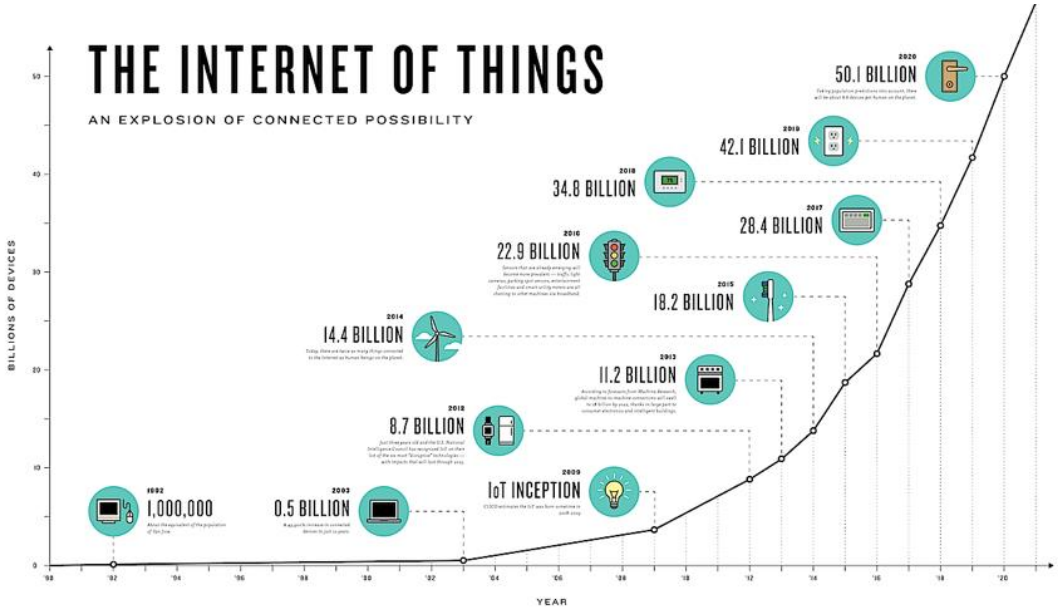

Figure 3-1. Internet of Things growth[1]

In 1999, Kevin Ashton formulated the term "Internet Of Things" [8] with a simple idea in mind: if we gave ordinary objects sensing and communication capabilities and the ability to process the data they generate, we would have the possibility to count and track everything. It was a simple idea but still very robust; objects could get information from the environment through sensors, process it and take actions based on results. Something obvious comes to mind when thinking of giving communication capabilities to objects: they should be capable of communicating with each other. Here is where the concept of M2M appears, which is a system composed of many interconnected heterogeneous devices that enables new applications in different domains like home automation, health and medical treatment, city planning and greenhouse automation[9].

[1] http://www.i-scoop.eu/internet-of-things-guide/

M2M systems are highly automated, meaning they work without, or with limited human intervention. One of the objectives of this automation is for the systems to generate data, exchange information and make decisions on their own, and, in many cases, act on the environment. For example, one device can measure the moisture of a plant, and another can use that measurement to decide if it should water the plant. M2M systems usually are composed of constrained devices. This limitation is due to many reasons:

Price: As in M2M systems and IoT environments many devices are deployed, their prices should be low enough to make the system affordable. Besides, these devices are usually deployed in unusual conditions that put them at risk of damaging (e.g. extreme weather), so keeping them within an affordable price range allows companies to replace them easily.

Energy Consumption: IoT devices are usually sensors or actuators, which results in them being deployed in specific areas that do not always offer stable power sources. Due to that, devices usually have limited energy sources like batteries. Because of that, energy should be used efficiently. Therefore, computational resources are limited as CPU, I/O, and RAM operations consume energy. The more powerful the device, the more energy it consumes.

Size: In many instances, IoT devices have to be deployed in specific areas that sometimes do not offer much space. This added to the number of devices to be deployed means it is desirable they are small. This means less available resources as some characteristics like CPU coolers are not available.

A perfect example of a constrained, cheap and small IoT device is the Intel Edison [10] (see Fig 3-2). It is a super small computing platform powered by an Intel Atom SoC dual-core CPU with WiFi, Bluetooth LE, and a specific connector to add to different boards that increase its functionality. The base module dimensions are 35.5 x 25 x 3.9 mm, making it one of the smallest and most powerful devices in the market.

Due to its good qualities, characteristics, flexibility, and popularity, the experiments for the current research are conducnet using the Intel Edison.

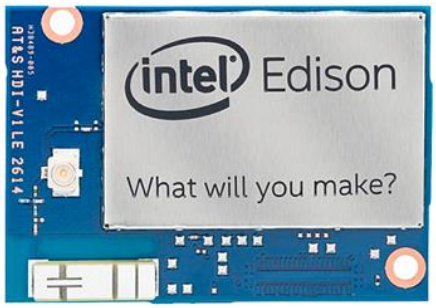

Figure 3-2. Intel Edison[2]

M2M is going to keep growing at a rapid pace. In the future billions of devices will be connected to each other and the internet. Those devices will be changing everything and creating a huge impact in our lives. Currently, the market is so full of devices that they are more numerous than people on earth, automating everything from parts of our lives to industrial environments. In the future, that trend will continue to grow. Moreover, according to Cisco [11] [12], by 2020 there will be over 50 billion devices, over 6.58 per person. These interesting numbers show how the market is growing incredibly fast. That is why, cooperation between constrained devices is so important. Managing that big number of devices will not be a straightforward process and it would be even harder if human interaction is needed to coordinate them.

There are three key components of the Internet of Things vision [13]:

- **A growth of low-cost/constrained and powerful devices:** They will require small price points and low power consumption. Therefore, the majority of devices that will compose the Internet of Things environment will be constrained. Nevertheless, more powerful devices like gateways, management stations or centric servers are also expected to grow to support the variety of applications.

[2] https://software.intel.com/en-us/iot/hardware/edison

- **High scalability level of connectivity:** For the M2M vision, this could be the most critical component. Devices have to be connected to be managed and to work properly. Even though devices can operate on their own, they should be manageable when required. Moreover, the Machine-To-Machine definition itself requires devices to communicate with each other to work together. This is a critical challenge as enabling stable communication channels for low-power heterogeneous devices in different environments is not a straightforward process. For coordination purposes, the communications costs are key to keep latency times within defined QoS and ensure a safe and stable system's performance.
- **Cloud-based management:** The M2M vision implies that all devices work together. It is no longer an independent vision in which devices worked on their own. However, managing billions of devices is going to be achieved from the cloud with a centric perspective. Besides, all the generated data has to be aggregated and analyzed in more powerful systems in the cloud.

3.2. REST

Introduced by Fielding[14] and also known as RESTful web services, REST is an architectural style designed as a model for how the World Wide Web (WWW) should operate. REST specifies a way of how computer systems (heterogeneous or not) should communicate.

REST was designed with a clear objective: Try to reduce latency and network communication and maximize the separation and scalability of components[15]. Therefore, it reduces payload sizes to the minimum, using in most cases, JSON, which is a lightweight format for data interchange.

REST web services grant the possibility to requesters to manipulate web resources through a logical path (URI). Requesters have the opportunity to use several operations for CRUD (Create, Read, Update and Delete) actions. As REST was designed to be an extension to HTTP/1.1, the operations available can be executed using the standard HTTP verbs providing the uniform interface[16]:

GET is used to read or "retrieve" a resource from the server represented by a specific path. The result of this operation usually responds with an HTTP code 200 (OK) and returns the representation of that resource using XML or JSON. It is important to understand that GET should be used only to read and never to change data. This means the method is safe as it cannot affect anything on the server, which means that calling it once or many times should generate the same result if no other method is being called. The GET response is cacheable if it follows the caching requirements defined in [16]. In the IoT environment, for example, this method could be used to retrieve the value of a sensor like humidity or temperature.

POST is employed to create a new resource in the specified path. Using usually either XML or JSON, the body of the request contains all the information representing the resource. When the operation is successful, the server responds with an HTTP code 201 (created) and the resource can now be identified by a specific URI. Otherwise, it will respond with a 400 code like 404 (not found). POST is not a safe method as it affects the information in the server creating the desired resource. The entity that is being posted is subordinate to that URI like a record is subordinate to a database. Making numerous identical POST requests will result in the creation of multiple entities with the same attributes. A POST response is non-cacheable as it usually does not have any content worthy of being cached. However, if the Cache-Control or Expires headers are present, it can be cached. For instance, in the IoT environment, this operation can be used to create a new feed used to keep track of sensor values.

PUT represents the update in the previously mentioned CRUD operations. It represents putting some information contained in the body of the request to a known and existent URI resource. The request body of this operation contains the updated representation of the entity in JSON or XML representations. Although PUT can also be used for resource creation, in cases where the resource identifier is decided by the client instead of by the server. This means PUT will create a resource if the defined URI contains no resources and, if it does, it will update that resource. The server should respond with the corresponding HTTP code depending on the result: 201 (Created) for the creation of a new resource and 204 (No Content) or 200 (OK) for an update. If neither the creation nor update was successful, the corresponding error code should be returned. In contrast to POST, sending multiple identical PUT requests will not create

multiple resources as they would always be updated. This operation is not safe as it modifies the state on the server. PUT can also be used in the IoT context altering the value of a resource, for example turning on or off a LED

DELETE requests the server to eliminate the entity identified by the requested URI. The server will respond with the corresponding HTTP code, 200 (OK) or 204 (No content). For a successful deletion, the code 200 is used when the server returns the deleted entity in the response body. Calling this operation multiple times will have the same response after the first one when the entity is removed (if it existed). DELETE is a non-safe operation as it removes information from the server. In the IoT context, this operation could be used to shut down a device.

According to the vision of REST, entities handled by client-server application logic are modeled as resources with five key concepts [17][18]:

- **URIs as resource identifiers:** Servers expose resources through specific URIs which are part of a global addressing space.
- **Uniform interface:** The interaction with resources exposed by the server is expressed by the four operations (CRUD) defined previously.
- **Self-descriptive messages:** Messages between REST components have to be self-descriptive with the information that allows their management.
- **Stateless Interactions:** Every request sent from the client to the server must contain all the information for the server to process it and produce a response. No previous requests are required, each one is completely independent.
- **Hypermedia as the engine of application state:** Hypermedia is very simple, and that is why it was chosen for REST; participants transfer resource representations that contain links allowing unlimited structuring.

3.3. COAP

CoAP, or *Constrained Application Protocol*, is a web software protocol in the application layer. It was designed specifically for environments with constrained devices. CoAP is foreseen to become the future standard for all application protocols as every day, more and more vendors are adopting it for light-weight devices that consume low energy [19].

M2M communication is key for enabling smooth integration of virtual and physical devices no matter what their location is and without requiring human intervention. Nevertheless, this is a feature that is difficult to achieve. Because of that, the collaboration among all communication layers must be enabled[20]. In the application layer, CoAP supports a request/response model with a pulling mechanism for getting messages from the queue.

As the usage of APIs on the Internet has grown over time, a substantial number of applications rely on Representational State Transfer (REST) models. However, even though Restful HTTP is a proven and efficient protocol, for IoT environments where many of the devices communicating have limited RAM, ROM, processor capacity and even power, a more suitable protocol must be used. CoAP intends to be the standardized protocol for this kind of constrained environments as its characteristics aim to make it simple and very efficient [21].

As the features of an IoT environment are so explicitly defined, there are some common features suggested for all new IoT protocols [19]:

- **Communication with low power consumption:** One of the key advantages of IoT devices is their size and how they can be deployed almost everywhere. However, regarding energy, this can be counterproductive. In many occasions, a constant and stable power source is not available, so sources like batteries need to be used. Usually, they must be small enough to be easily carried. An IoT protocol should aim to preserve energy to avoid communication disruption.
- **Highly reliable communication:** The Internet and M2M connectivity should be fast and very stable. Nevertheless, sometimes for unexpected reasons like

environment interference, the connectivity is lost causing devices to re-transmit, incurring in more power costs.

- **Internet-Enabled:** Internet connection enabling M2M communication is a must.

CoAP intends to have all these characteristics and enable them for constrained devices without incurring unnecessary costs.

This protocol was developed by the IETF Constrained RESTful Environments (CoRE) working group, it has been standardized under the RFC 7552. The objective of CoAP is to make the REST model suitable for devices with limited resources, scenarios where using the original HTTP approaches is not feasible.[22]. CoAP does not aim to use a limited version of HTTP but rather to implement a subset of it enhanced for M2M applications.

The following are some of the most important characteristics of CoAP:

- It supports M2M requirements for communication of constrained devices.
- UDP binding with optional support for unicast and multicast requests.
- Low header overhead and parsing complexity.
- It supports URI.
- Proxying and caching

Even though there are other communication protocols for constrained environments like MQTT [23], CoAP is a solid way for constrained devices to communicate. Furthermore, it is easy to understand and develop, which is why it is widely used nowadays. The way CoAP interacts is very similar to the client/server model of HTTP. However, in M2M communication, a device will usually act as a client and server at the same time, requesting actions of other devices (acting as a client) on specific resources exposed by other devices (acting as servers).

CoAP uses IPv6 as well and 6LoWPAN in its network layer to manage nodes identification. Some efforts are still being made to combine these protocols and create a unified standard, besides supporting publish/subscribe and request/response models [24],[25].

CoAP has defined four types of messages: Confirmable, Non-confirmable, Acknowledgement and Reset. Requests are usually of the types Confirmable or Non-confirmable whereas responses are typically Acknowledgment or Reset.

As shown on Figure 3-3, CoAP uses a two-layer structure: The bottom layer is the Message layer that works over UDP and the Requests/Responses layer that communicates with applications using specific codes and methods.

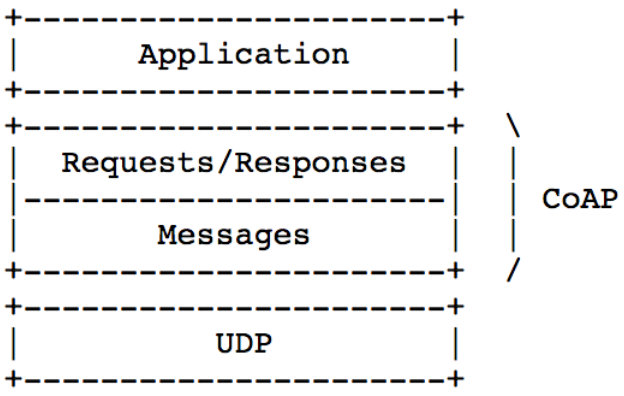

Figure 3-3. Abstract layering of CoAP [26]

Requests are sent over a Confirmable or Non-confirmable message, and, if possible, the response is sent in the following message in the resulting Acknowledgement message. This is called piggybacked response. Two examples of this communication model can be seen in the Figure 3-4 where a client makes a GET request to the /temperature resource with a Confirmable message. In the first example, the server returns the value of that resource ("22.5 C") in an Acknowledgment message. In the second example, the server has no value for the requested resource, so it responds with "Not Found." As can be seen in this example, responses also have response codes. In the first example, the response is 2.05 (Content) and in the second example, the response is 4.04 (Not Found).

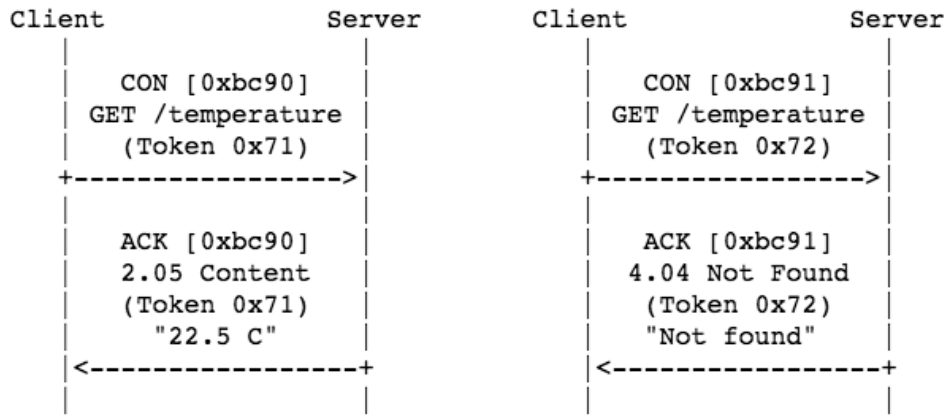

Figure 3-4. Example of two GET requests[26]

In a typical IoT environment, sensors are servers that expose the value gathered from the environment in a URI-identified resource. That way, the values can be pulled from them only when required by other devices (clients).

One key characteristic of CoAP is that the messages exchanged are compact and transported over UDP (can be implemented over TCP as well). They are encoded in a simple binary format in which the header occupies the first 4 bytes. It is followed by a variable-length token that goes between 0 and 8 bytes long. After the Token, it comes a variable number of CoAP Options, followed by the payload when present as the last part of the message.

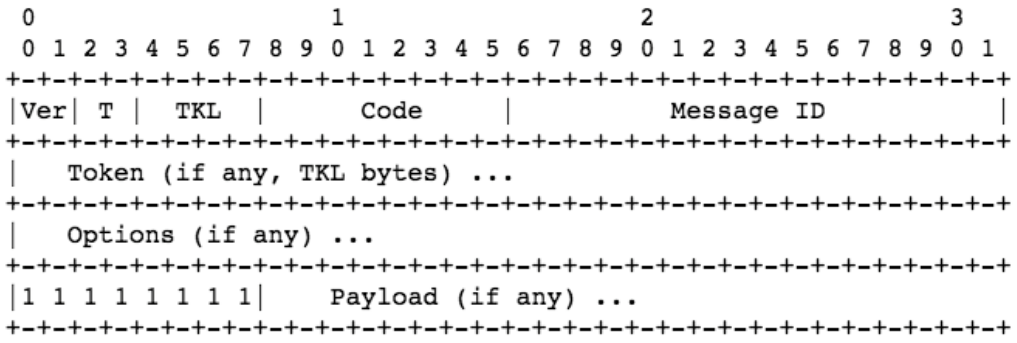

Figure 3-5. CoAP message format[26]

CoAP is a suitable protocol for IoT that allows different operations to the connected devices while minimizing the communication costs using a simple request/response model. However, when devices are waiting for a value to change, it might not be efficient to make

several requests and evaluate each response. For that reason, CoAP implements an attractive feature called *Observe Resources* [27]. This feature allows clients to register for a specific URI-identified resource of a server. When that resource changes, all registered clients are notified. This reduces the number of calls that need to be made and makes communication more efficient. This is one of the key features of the protocol that will play a major role in the future for monitoring resources [28]

3.3.1. Caching

Endpoints in CoAP have the possibility of caching responses to reduce processing time and therefore, make the communication faster by reducing bandwidth. Sometimes, and depending on the IoT environment, prior responses can be used to satisfy an ongoing request. Usually, a response is stored with the request that generated it. When the same request is received again, the stored response is sent, avoiding unnecessary processing or another network request. This further reduces response times and saves energy. Each response can be cached for a period of time, using the "freshness model", which using the Max-Age option, determines for how long a response can be considered fresh and cacheable. Once the Max-Age has been exceeded, the response is no longer valid within the cache. Depending on how each device works, this option must be set.

Whether CoAP responses can be cached or not, does not depend on the request method but on the response code. Some response codes cannot be cached while others can.

3.3.2. Proxying

Endpoints can play the role of proxies with CoAP; they have the capacity of acting on behalf of other clients to perform requests. This is a nice feature when requests cannot be made directly by clients because of system limitations. Besides, when using a CoAP proxy with cache, the response times can be improved greatly, thus, reducing bandwidth.

Proxies have the possibility of being directly selected by clients, and they can have one of two main functions:

- CoAP-to-CoAP proxy: Maps from a CoAP request to a CoAP response.

- Cross-proxy: Translates for a different protocol.

CoAP-to-CoAP proxies are usually used within IoT networks to enable communication between several devices. Sometimes, they work as hubs for communication, and because of that, they can be more robust than the rest of the devices. On the other hand, Cross-Proxies function as gateways to enable communication with other networks that may not support CoAP.

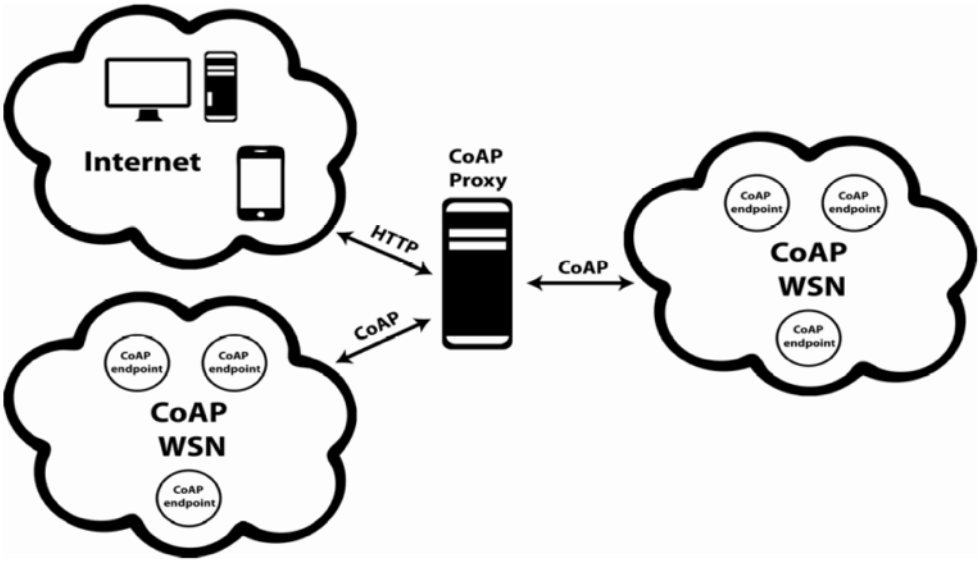

Figure 3-6. Common CoAP architecture with cross-proxy[29]

3.4. Coordination Models

Various types of systems are becoming more and more complex every day, and they are now being assembled by more and more processes (entities) that are usually residing in different devices. It has been recognized that interaction between those components is a key factor for any complex system [30]. Defining, configuring and managing those systems require another level of abstraction: A framework or group of tools that are intended to model interactions between the components, while taking advantage of distributed systems. That is what coordination models aim to do [31].

Coordination models are the different ways systems have to combine various activities into a whole. A coordination model keeps together a group of activities or agents providing a framework within which, the interaction between agents can be expressed [32], [33]. For any

cooperation system, eventually, a coordination model must be used, keeping in mind that its design and implementation can affect the system's performance [5].

There are some aspects a coordination model should define and cover: communication between agents, their creation, destruction and distribution of their actions, among others. Defining the communication model between the entities is very important. It needs to be clear that for changing environments, a reliable and well-defined communication is key. Besides, as the environment can change over time, the coordination model should handle two events: when the agents are created and can start forming part of the coordinated group, and when they are destroyed. In both scenarios, the model has to handle them correctly in the least disruptive way.

When designing a concurrent application, the main concern is its model of cooperation. Defining how every entity is going to cooperate with the other to achieve the task in the best and more efficient way is very important [5]. Though there are very well-defined coordination models that can be put in place, there is no key formula that can be applied to every system. Depending on its characteristics, a different and specified model must be established.

A coordination model can be defined by identifying its principal components [33]:

- **Coordination entities:** The items that will be coordinated; these are the activities or processes that will cooperate with each other.
- **Coordination media:** The media over which the communication takes place.
- **Coordination laws:** These are the rules the coordination model dictates. They determine how the cooperation will be achieved. The coordination laws must take into account the different scenarios the system can have, and they must be followed by the entities using the communication channels defined.

A coordination model can be enclosed in an architecture (software) or a language. Some examples of coordination architectures are the software pipeline and the blackboard. Different components are organized in a specific way to achieve cooperation protocols [34]. On the other hand, a coordination language is the linguistic representation of a coordination model [32]. In a coordination language, two different languages have to be combined: one for coordination, and one for computation. These two aspects are fundamental as they define how the model works.

For some models, the coordination language is completely separated from the computation language, in others, the two of them are tightly combined.

One important characteristic every coordination model must have is that agents should be able to join and leave dynamically. It is key that the system can self-organize and not cause disruption when agents come or go. The blackboard model is a good example of a coordination architecture. It defines a group of "experts" or agents that are trying to solve a complex problem, and they share a blackboard. When an agent wants to process some information, it takes it from the blackboard and puts the result back so other agents can continue working on the solution. No agent has to know about the others and new agents joining the system or leaving it will not affect the others. This is the key concept of the most famous coordination model: Linda Tuple Space, which will be explained in detail in the following sections.

Coordination models, languages, and architectures are closely associated with the concept of heterogeneity. As the computation component is independent of the coordination one, which sees the processes in the system as black boxes, the coordination component has no concern on how the computational one works, is conceived or modeled [31]

The coordination models can be classified into different categories based on the entities being coordinated, the communication medium, the architecture, issues of scalability or more [35]. However, it has been proposed[31] to divide coordination models into two categories: data-oriented and process-oriented. What differentiates the two is the division between the computation and coordination components.

In the data-oriented group, coordinated processes oversee examining and processing data, and the coordination task is done by themselves or other processes using the coordination mechanisms provided by each language. Usually, within this group, the coordination language offers coordination primitives which are blended with the computational tasks. That means that every process (coordinator or coordinated) cannot be easily identified as a computation or coordination process as it is the developer's responsibility to model the program to develop the coordination and computation tasks.

In the process-oriented category, the coordination aspects of the program are separated from the computational ones. That means that the data being manipulated at any moment in

time has nothing to do with how the coordination is being done. The way to accomplish that is to define a coordination language in which processes appear as black boxes with specific input and output. The model requires that the computation module and the coordination module are separate.

3.4.1. Data-Oriented

The data-oriented or data-driven coordination models do not show a visible division between computation and communication processes. Most of the models of this group employ the concept of a *shared data space*, which is a common, content-addressable data structure [36]. Every coordinated process accesses the shared data space to communicate with others; it is the only channel available for that purpose. Moreover, they can put information into the data space and retrieve other data based on specific criteria (similar to the blackboard example explained above). The content of the data space at any specific moment is independent of the status of the processes. With this model processes are decoupled in time and space. Using this medium allows coordinated entities to communicate with each other without knowing who the consumer or the producer is. As, in the blackboard example, the agents take and post information from the shared space regardless of who might use it.

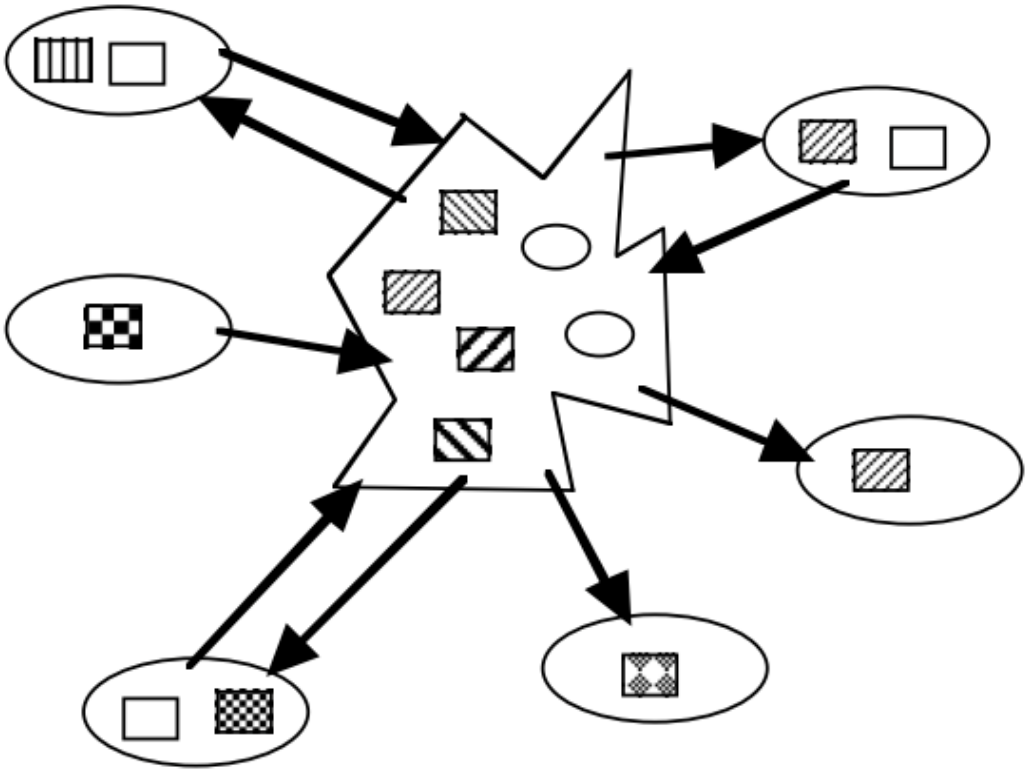

Figure 3-7. Shared data space model [31]

The Figure 3-7 shows how the shared data space model is seen. The round objects are the coordinated entities or the processes that are posting and retrieving data from the data space (middle). Those data structures in the data space can be of different kinds. When a process retrieves data from the shared space, it can copy it or remove it, which means that no one else can access it. This is a very interesting concept for concurrent operations as removing information from the data space might help passively coordinate the entities. As processes can do two different actions (consume and produce data), it is up to the system designers to decide what is going to be the role of each component, keeping in mind that they can be either consumer, producer or both. Usually, in these models, the data structures are based in tuples. They can be simple plain ones or more complex, permitting searches based on specific parameters.

Linda

There are several different coordination models, but perhaps the most important one and certainly the first one to be considered a coordination model is Linda [37], [38], which was designed to support the creation of parallel programs. One of its objective is to avoid the problem of coupling between parallel processes and make them not have to deal with each other directly, creating a decoupled way of parallel programs communication

Linda is based on the common data space concept using tuples, creating a shared tuple space, in which, if two entities want to communicate with each other, the producer must generate a new tuple containing that data and put it in the tuple space [39]. Then, the consumer can retrieve said tuple by copying it or removing it [31]. This makes the two entities decoupled in two different dimensions: time and space; time because there is no need for the two entities to be alive and working at the same time, and the information can be placed in tuple space and be consumed later. And space because no process needs to know the identity of the others or their location. This decoupling level provides a required level of flexibility these kind of systems should have [30]. For instance, when the producer works much faster than the consumer, or there are more producers than consumers, the tuple space works very well in allowing producers to push data even when the consumer is not ready for it, making them available much faster and creating an ideal parallel scenario.

Linda can be used within many programming languages; it offers some simple operations that allow processes to interact with the shared tuple space. When these operations are added to a programming language, a parallel programming dialect is involved [38]. For interacting with the tuple space, Linda defines four fundamental operations: out(), in(), read() and eval(). The out(*t*) operation is used to add a specific tuple *t* to the tuple space and continue the process normally. in(*s*) is used to retrieve or withdraw a tuple *t* that matches the search criteria and template *s.* If a tuple is found, it is retrieved and the process continues; if no tuple is found, the process is suspended until a tuple is available and retrieved. The read(*s)* operation is used in the same way as in(*s)* but the tuple is read and copied, only this copy is retrieved and the actual tuple is left in the tuple space. The eval() operation works very similar to out() but it

does not add simple plain tuples to the tuple space but rather a real process that can be retrieved and started by other entities.

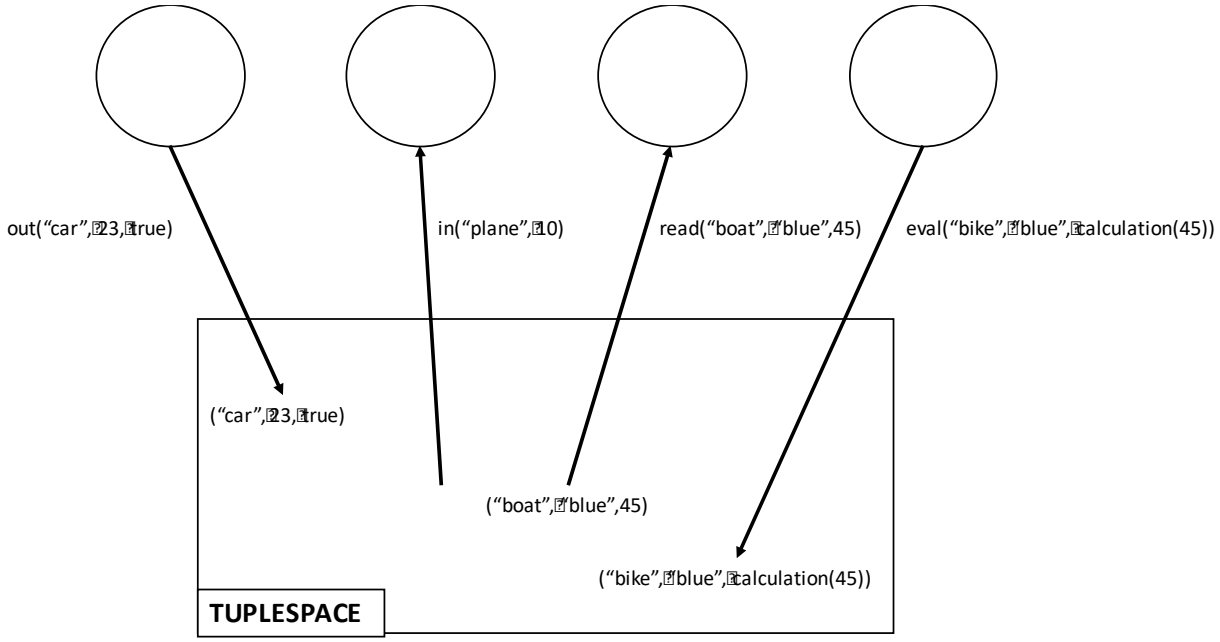

Figure 3-8. Fundamental Linda operations

Figure 3-8 shows an example of a tuple space and four agents performing the fundamental Linda operations. Every agent can perform any operation to a specified tuple, and tuples can be of any type. For example, the tuple ("boat", "blue", 45) has a logical name, a string value and an integer value. The logical name is how tuples can be identified and searched, and the rest of values are the actual tuple data.[37].

Because of Linda's advantages, it has been an active research topic. Many researchers have worked on it and created several variations with specific changes. Depending on the objective, some of those variations are Bauhaus Linda [40], Bonita [41], Law-Governed Linda [42] LAURA [43], Objective Linda [44] and Lime (Linda in a Mobile Environment) [6].

Lime [45] is a framework based on the Linda operations that aims to support the development of distributed applications that are in mobile hosts. The idea behind Lime is to have the Linda tuple space but in a transient and shared way. Individual mobile agents share

identically-named tuple spaces. In Lime, mobile agents are hosted in the coordinated entities. These agents can move logically of physically and the system adapts to continue working.

Lime takes the global concept of the tuple space and distributes its content among the mobile units. When these units are in range or can communicate, the contents of their tuple spaces are shared creating a virtual shared space.

3.4.2. Process-Oriented

The process-oriented or control-driven coordination models have been developed to address some of the aspects data-oriented models lack [46]: A complete and visible division between computation and coordination components of the processes, and to avoid the need for some processes to know more about consumers or producers. This type of coordination model aims to make processes aware of state changes of the other processes by observing them and allows the broadcasting of events [31].

In contrast to data-oriented models where coordinators handle the data directly, here, processes are just black boxes with defined input and output interfaces (also called ports). The relationships between consumers and producers are established by creating a channel or stream that connects the output interface of the producer to the input port of the consumer. Besides the point-to-point communication processes have through ports, they can also broadcast messages to the environment to let other interested entities know their current state.

When talking about connecting agents and coordinators through their input/output ports using channels, we are also talking about the Ideal Worker Ideal Manager (IWIM) model [5] of communication. This is a family a communication models rather than only one and defines processes as black boxes and enforces complete separation between the computation and communication tasks.

Figure 3-9 shows an example of a process-oriented coordination model architecture. There is one producer *Prod* with two output interfaces, and two consumers: *Cons1* with only one input interface and one output interface, *Cons2*, with two input interfaces and one output interface. A channel has been created between the output interfaces of the producer and the

input ones of the consumer. It is possible to have more than one channel connected to an output or input interface.

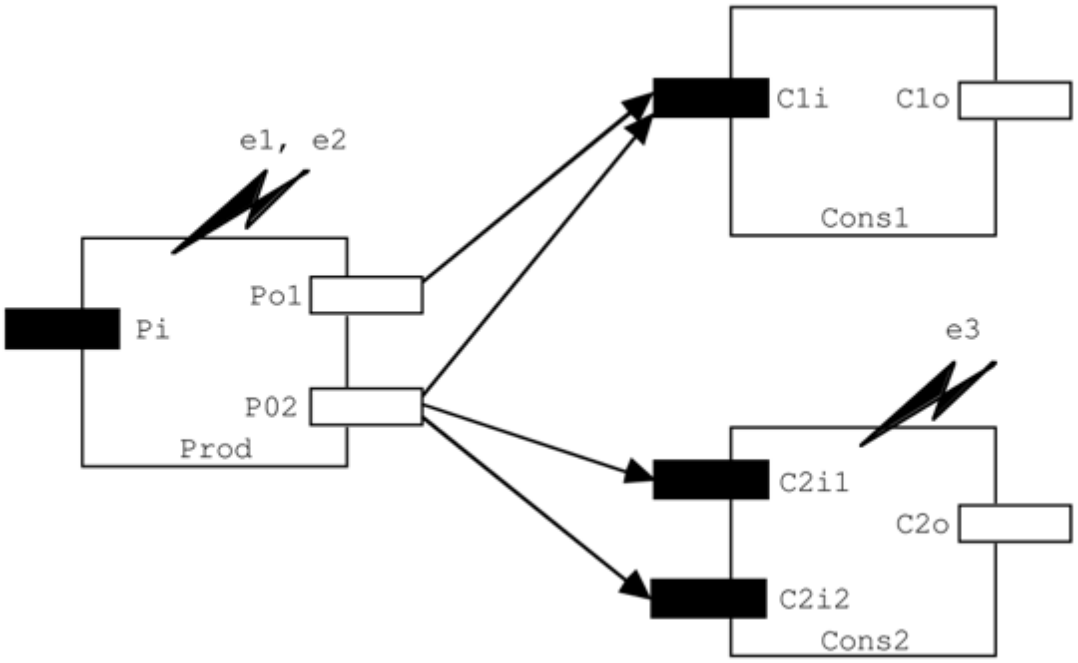

Figure 3-9. Process oriented coordination model [31]

There is a variety of process oriented coordination languages that have been developed with different objectives in mind. Some of them are the following:

- PCL (Proteus Configuration Language) [47]: A language developed to design architectures of different versions of computer systems
- Conic [48]: In Conic, coordination is seen as configuration, is comprised of a programming language and a coordination language, which is in charge of creating the links or channels among the entities.
- Durra [49], [50]: An architecture configuration language in which the applications are composed of a set of components and a set of configurations that specify how the components are related.

Even though the process oriented list of coordination models is large, MANIFOLD [51] stands out because it is one of the latest developments in the evolution of the process oriented group. In MANIFOLD, the coordination processes are clearly separated from the computational ones which can be developed in any programming language. Coordination entities are called Manifolds and they use input/output ports to communicate with each other over streams. The model is event-driven based on state transitions, meaning that Manifolds are always at a specific state. When events are detected, the coordinators evolve into another predefined state. Events cannot be used to carry data, only to trigger state changes in the entities.

Typically, when a Manifold is in a certain state, it has set up a network of processes. When an event is received, the communication with those processes is finished and with the new state, a new network of processes is set up. This gives MANIFOLD the characteristic of dynamic reconfiguration and system consistency.

In terms of coordination models, the research done is broad and is very advanced. Many models, frameworks and types of middleware have been developed for different environments. Moreover, for mobile environments, specific solutions have been developed to address the specific characteristics of this environment. However, to the best of our knowledge, the IoT environment still lacks specific coordination models that allow devices to cooperate effectively.

3.5. Fog Computing

Fog computing is a new computing paradigm that has one specific objective: bring some of the services offered by the cloud closer to the network, to the edge of it. Fog computing is located in the layer between the cloud and the edge devices and is actually another network that enables the modeling and creation of better applications or services [52], [53].

Fog computing makes it easier to manage, configure and program different kinds of devices located between the cloud and the end devices. This is accomplished by locating components of the applications in the cloud and in the edge. Edge devices with this purpose can be routers, smart gateways or dedicated fog devices. When using routers or networking

components, they should be more robust than regular devices in order to handle the additional load [53].

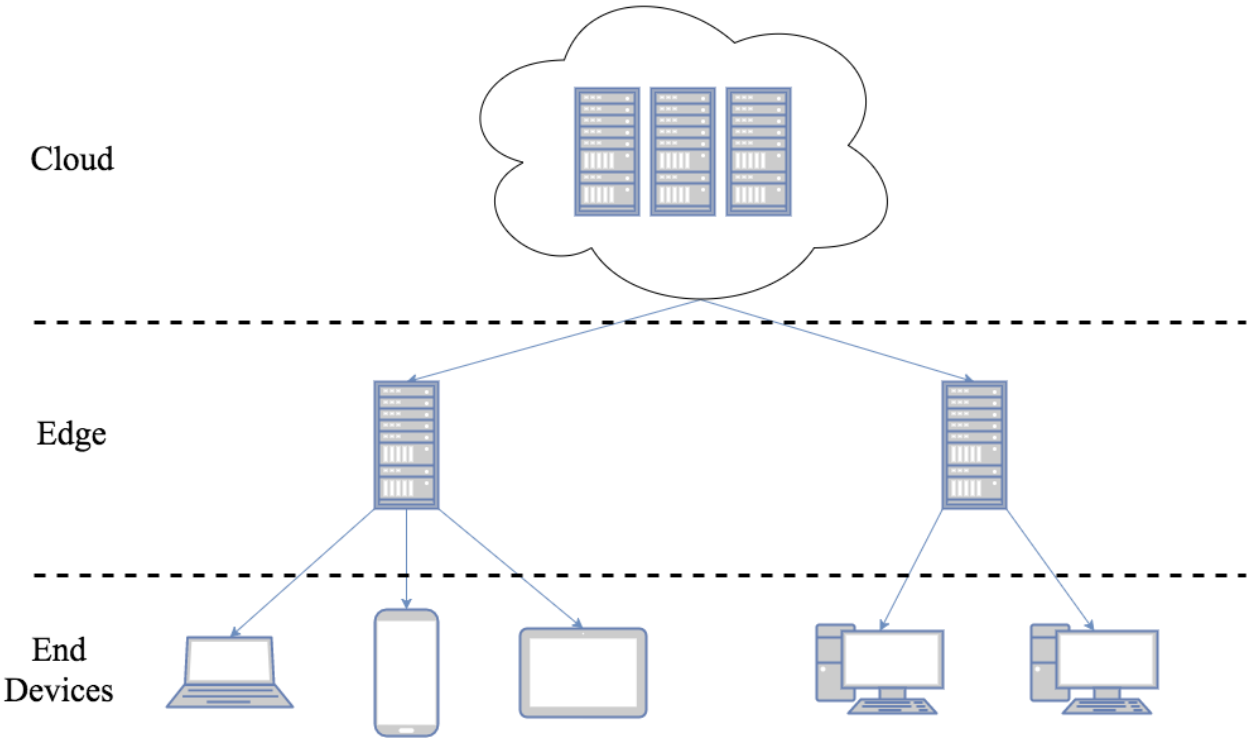

Figure 3-10. Fog Overview

The layer that fog computing creates between devices and the cloud enables many different approaches for the Internet of Things and Ubiquitous Computing (UC), extending cloud services to include more powerful sensors capable of handling more computation and making decisions without going to the cloud [54].

The main advantages offered by the fog computing model are the following:

- Network traffic reduction: According to Cisco [12], there are over 25 billion devices connected around the world. By 2020 that number could go to 50 billion. All those devices are generating, retrieving and sending data all the time. It does not seem very effective to send all that data to the cloud to be processed. Also, because many times that data is not completely useful. Getting computational

capabilities closer to these devices makes much sense in order to analyze and filter data before sending it to the cloud.

- Suitable for IoT tasks: As the number of smart devices increases every day, every device is going to be surrounded by others. In many situations one device can affect the others for collaboration purposes. In those scenarios, there is no need to send data to the cloud if the device is closer and reachable.
- Low-latency: Many complex applications require real-time data for making decisions. For example, the brakes of the car need to know the sensors information to make decisions in very short time. It might take too long for the sensors to send data to the cloud and for the breaks to retrieve it. By bringing computation closer to the devices, the transfer time for data is reduced.
- Scalability: With so many devices interacting with the cloud and more coming in the foreseeable future, the cloud could become the bottleneck for data processing. With fog computing, the burden of that processing in the cloud is reduced and the scalability issues of it are addressed.

Fog networks are usually, but not exclusively, located at the edge of the network. Even though that is the most common approach, fog computing aims to work on applications and services widely distributed, as opposed to the cloud where there is a centralized point of integration. This allows the fog to communicate with mobile nodes with the possibility of creating a smart, dynamic network in which a device like a smart gateway can be used [53].

As the IoT environment is so dynamic and the network traffic and latency must be kept at their minimum, the fog offers important advantages that can solve many of the issues the IoT environment has. If the nodes do not have to communicate with the cloud at all times and have computation closer to them, response times will be better and the overall system's performance will be improved.

Fog computing as a model has many advantages as explained above. However, it also allows the implementation of other characteristics that can improve a system's performance. Caching and preprocessing are two of those, in which the middle layer processes the data and

can either send it or not to the cloud or other devices depending on the result of that processing and previous requests. For example, Zhu et al. [55] explore how to use edge devices to improve the performance of websites. End users connect to the internet through fog devices that capture all requests and optimize the requests to reduce the time the user must wait until the page loads. There are several kinds of processing these devices can make. In IoT, they could, for instance, keep track of the sensor values that are being sent to the cloud and only send values with significant changes. That way, a sensor will not be sending the same values again and again. Furthermore, there are many applications for fog computing within IoT. Data preprocessing and trimming is another, where the data is transformed, reduced or optimized before sending it to the cloud, which is a big necessity as sensors are generating large volumes of data.

Fog computing can also be used to manage resources in an effective way. As many times devices require more resources to perform specific tasks, models for creating elastic edge-based applications have been developed. One of the most explored has been computation offloading, where usually workloads are offloaded to the cloud [56], [57], [58]. As offloading computation to the cloud is not always possible, offloading computation to edge devices has also been explored [59] to facilitate the development of elastic, edge-based mobile applications.

3.6. Blockchain

Blockchain is the core of Bitcoin [60] and it is defined as "a global decentralized ledger which stores the full history of all bitcoin transactions"[61]. Bitcoin transactions are stored in the blockchain in blocks and every block is linked to the others. More specifically, in addition to the timestamp, each block contains a hash of the previous block. This creates a sequence of linked blocks that form a chain in which each block can confirm the integrity of the previous one, allowing the members to verify transactions. In a blockchain network, nodes can join or leave at any moment without affecting the overall functionality since the blockchain is stored in every node of the network, which, for June 2017, contained over 7400 nodes [62].

As there is no centralized authority to validate or confirm the integrity of transactions, these are validated with a consensus mechanism. Any node can create transactions that are

propagated within the network. Other nodes can take those transactions to create a new block containing them along with the link to the previous block. This process is called “mining”. In order to prevent minority control, miners have to perform a computationally expensive task and add the block to the blockchain. This technique is called proof of work [63].

One of the most popular blockchain networks is Bitcoin [60], which is public access. Anyone can access the blocks and the transactions but no participant is trusted; they all must provide their proof of work for validating their transactions. Every user is provided with a public and private key. The private one is used for signing transactions, therefore, no real-world identity is required, the public key system creates a form of pseudo-identity [63].

Private blockchains have also been developed. One of the most popular is called Multichain [61] and has been designed with three main differences from Bitcoin: It ensures that blocks, transactions and overall activity is only visible to selected members, it introduces controls to permit specific transactions, and it enables mining without proof of work and associated costs.

Even though one of the main usages for the blockchain is for cryptocurrencies, it has also been explored in other areas where using blockchain is useful [63]. One of its main usages beyond cryptocurrencies is data storage, taking advantage of its characteristics like scalability, high fault tolerance and tamper resistance. For distributed systems and IoT environments, and from the architectural point of view, it has been proposed to use blockchain as a software connector which can be an alternative for centralized shared data storage [64].

For the present work, the private blockchain Multichain is used as a software connector to share data across different devices.

3.7. Summary

Table 3-1. Literature Review Summary

Section	Description
Internet of Things and M2M	Introduces the environment in which the research is developed. States the current importance of IoT. Explains the basic concepts about constrained environments and how the M2M communication works. Describes the characteristics of the IoT environment and the challenges it has
REST	Explains the ideas behind the REST communication protocol. Describes how it works, the operations it uses, its characteristics and main advantages.
COAP	Describes the way CoAP works and its value for constrained environments. Explains how the M2M communication works using CoAP. Illustrates how the messages exchange is done and how it affects constrained environments. This section also describes the characteristics CoAP has, including its caching and proxying features, which are important for developing the coordination models.
Coordination Models	Explores the origin and types of the coordination models. Reviews the main objective when implementing coordination models, how they were conceived and how they work. Lists and illustrates some of the most important coordination models developed and the difference between them.
Fog Computing	Describes the fog computing paradigm. Explains and illustrates what a fog environment is. Analyzes the advantages this paradigm has and how they can be applied to the IoT environment.
Blockchain	Explains the origin and objective of the blockchain, some use cases and how it works. Describes how the blockchain can be used as a software connector to share data across devices.

CHAPTER 4

SYSTEM ARCHITECTURE

Internet of Things is a relevant technology in which constrained devices and physical sensors are interconnected and create M2M resource-constrained networks.

This research explores different coordination models and their application to IoT. The literature review shows that even though there is a variety of coordination models and languages developed, there is still space for research about coordination models specially designed for constrained devices in the IoT environment. Moreover, the work that has been done in the area has mostly been focused on data-oriented coordination models.

IoT networks usually consist of a variety of devices with very limited computation power and resources. Those resources are not always being used, and some of them are not being used effectively. Moreover, devices often waste resources sending repeated information or with processing tasks that have already been processed.

This research explores how the data-oriented coordination models can be used within IoT taking into account its limits and characteristics. Also, one of the most important contributions of our work is the development of a new way of implementing a process-oriented coordination model for IoT.

The main objective of the present work is to allow constrained devices to communicate with each other in a coordinated way to cooperate to solve more complicated tasks, using computational resources and energy effectively.

In IoT, devices can suddenly disconnect or connect for many reasons like lack of energy, programmed maintenance, environment changes or just because new devices are being introduced to the system. The coordination models proposed in this work are fault-tolerant, and for that reason, they might be partly centralized as well.

There were four coordination models implemented, three were variations of data-oriented models based on shared data spaces, and one was a new process-oriented model

developed to connect devices over specific communication channels and coordinators. The details of the architecture of all the models are explained in the following sections.

For homogeneity and to make all the implementations comparable, this work uses the Intel Edison[10] as constrained device, which is a small, cheap yet powerful system on a chip (SoC) widely used nowadays. This device includes a dual-core CPU, Bluetooth, and WiFi connectivity, making it resourceful and very useful for IoT tasks. The Intel Edison was used on top of an Arduino board, called the Intel Edison Arduino Kit [65], these way the Arduino acts as the interface with the exterior world, and the Edison provides computation power and an environment where programs written in different languages can be run.

As in a typical IoT environment where the communication is over WiFi, the connectivity between every part of the system was done over WiFi as well. Nevertheless, this is a channel in which connection can sometimes fail, so testing the fault-tolerant characteristic is important. Also, to have an efficient system and to reduce the communication costs to the minimum, CoAP was used as the communication protocol.

Figure 4-1. Intel Edison Arduino boards used

4.1. Data Oriented models

In this group, three different models were implemented. All of them were based on the concept of having a shared tuple space: one or more tuple spaces that could be located anywhere within the network. The system implementation needs to know the location of them and the devices need to know what kind of information to share.

For each of the data oriented models, two variations were implemented:

- One producer and one consumer or worker
- One producer and two consumers or workers

The idea behind implementing these modifications was to measure how different the model behaved with the various numbers of coordinated entities. Besides, we could say that in a well-defined coordination model, depending on the defined tasks, the more workers are available, the faster the tasks will be performed. Consequently, in this case, the two consumers variation was expected to be more efficient than the one worker variation.

4.1.1. Tuple space in a dedicated server

The first implemented model was using a dedicated and more robust server as a tuple space. It ran a COAP server and kept the data space in RAM, making it very efficient to look up tuples and add new ones.

Figure 4-2 shows the first implementation. Here, only two constrained devices were coordinating, the device number one was the producer, only sending data to the tuple space server. The device three was the consumer or worker, searching for tuples in the tuple space and writing the results back. The device number two was a robust server in charge of writing tuples and searching for them when required. The hardware characteristics of this server are: 3.2 GHz Intel Core i5 processor, 16GB of RAM and 1TB hard disk drive.

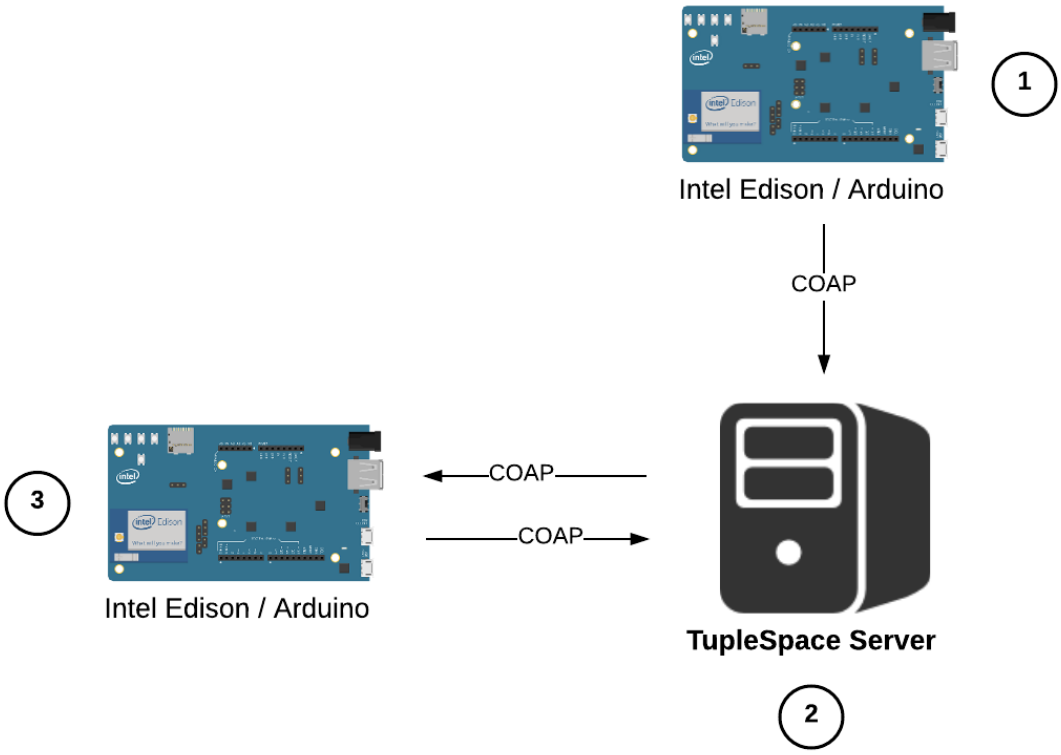

Figure 4-2. Tuple space model in RAM with one worker

The next experiment was very similar than the previous one, with the difference that three constrained devices were coordinating (Figure 4-3). The device number one was the producer, and the devices three and four were the consumers or workers. The device number two is the same server used in the previous implementation.

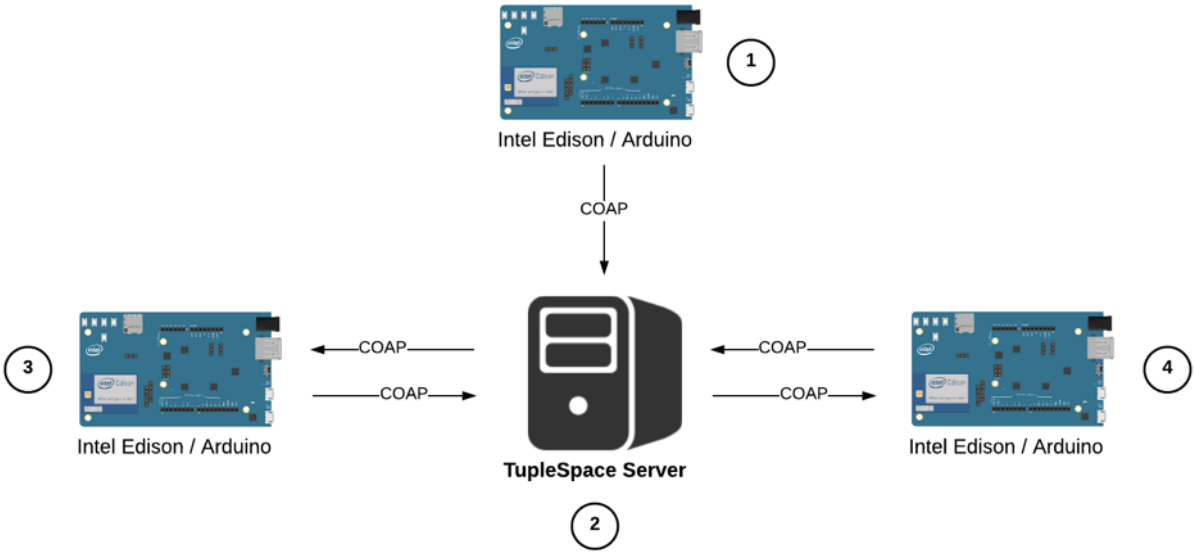

Figure 4-3. Tuple space model in RAM with two workers

4.1.2. Tuple space in a relational database

Another approach was to keep the tuples in the hard disk, saving them to a relational database, which in this case was MySQL 5.7 [66]. To make this deployment very similar to the others, the Edison devices continued to send the tuples to the tuple space server, which would communicate with the database in another server. That would allow the original tuple space server to expose the same CoAP operations but change the storage in the background.

Figure 4-4 shows how the first experiment was deployed, with only two devices coordinating. The first device was the producer, in charge of generating data and sending the tuples to the tuple space. The device number three was the consumer, in charge of getting the tuples, processing them and sending the results back. Device number three was the robust server which exposed the CoAP operations to write and read tuple. The hardware characteristics of this server are the same as the device number two of the previous experiment. In addition to those components, another component was introduced: The MySQL server in charge of storing the tuples. For communication purposes, the MySQL server exposed a set of REST services developed in Java to allow interaction with the tuples. The hardware

characteristics of this server are: 3.4 GHz Intel Core i7 processor, 32GB of RAM and 2TB hard disk drive.

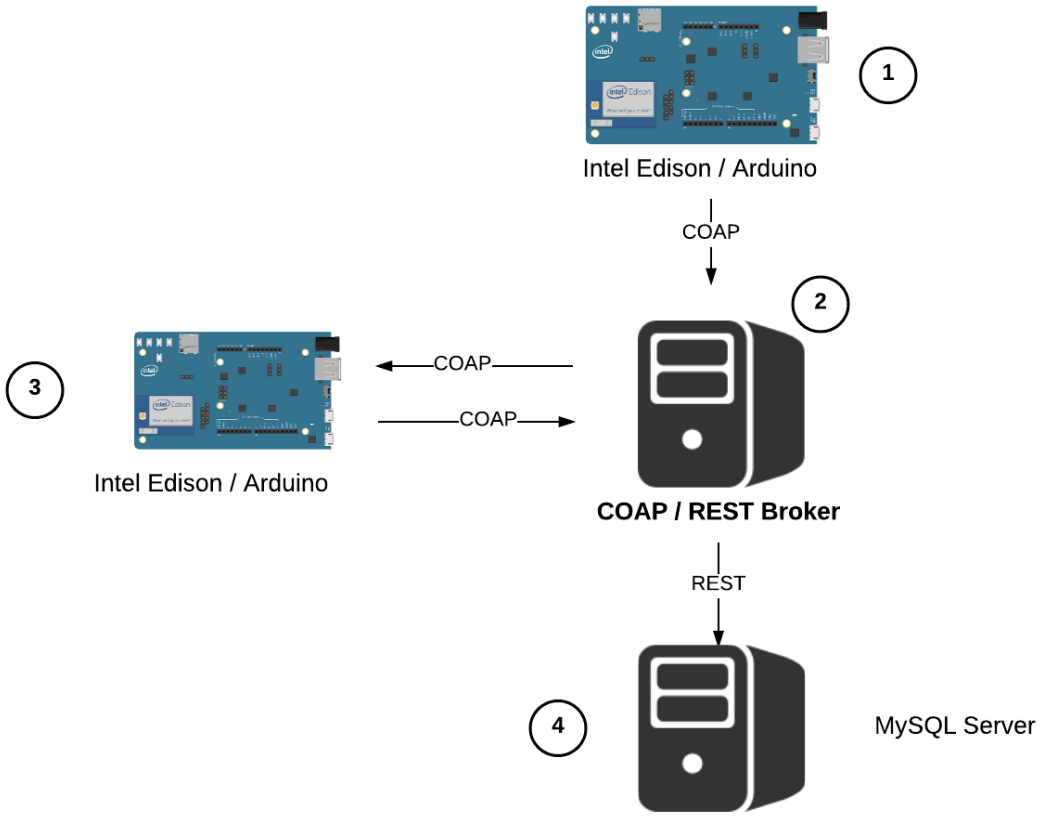

Figure 4-4. Tuple space model in a relational database with one worker

The next experiment was very similar; the only difference is that another worker was introduced to communicate with the tuple space. For both this and the previous test, the communication between the constrained devices and the tuple space server was done using CoAP, and this server would communicate with the database server using REST. As these two servers are more robust, using REST is entirely appropriate.

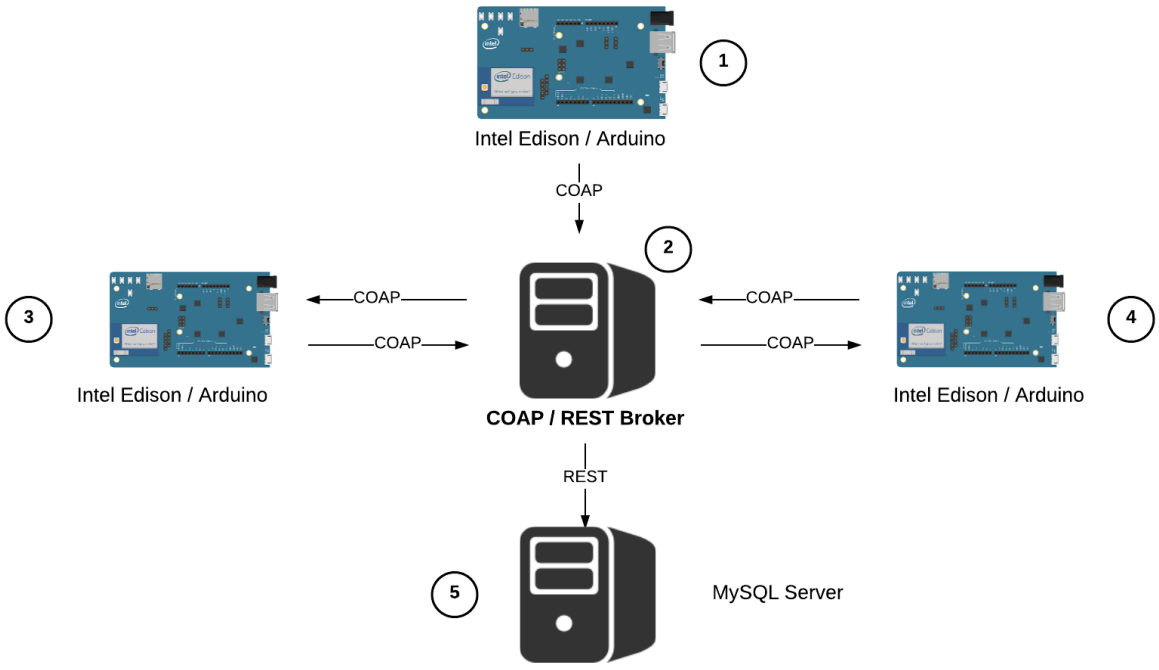

Figure 4-5. Tuple space model in a relational database with two workers

4.1.3. Tuple space in the blockchain

The last data-oriented model that was implemented was using blockchain. In this case, the constrained devices would continue communicating with the tuple space server which stores the tuples in a private blockchain called Multichain[61], which works as a software connector [64]. In this case, the relational database is being replaced by Multichain, which can do everything the database does if robustness and trust are not an issue. Nevertheless, some differences need to be taken into account[67]

Using blockchain has many advantages [68], of which some of the most important are:

- High quality data
- Trustless exchange
- Fault-tolerance

For the IoT environment, perhaps one of the most attractive advantages of Blockchain is its distributed model, which allows every device to have the blockchain and create a fault-tolerant environment.

Figure 4-6 shows how the approach to use Multichain was done. In this case, two constrained devices were collaborating. Device one was the worker, and device three was the consumer. As in the previous experiments, they would send and retrieve tuples from a tuple space server. As an interface to Multichain, the device four was used, which exposed REST services to interact with the Multichain.

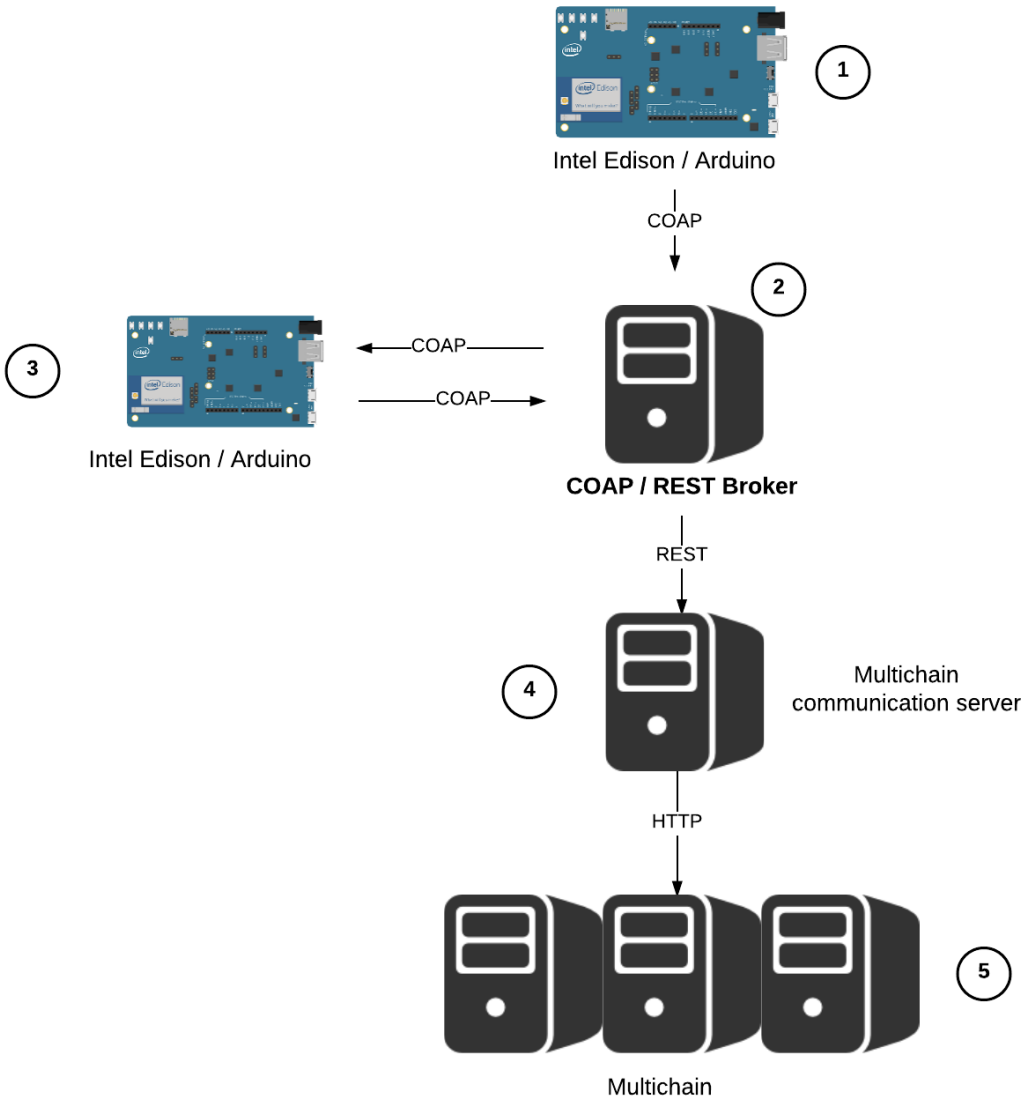

Figure 4-6. Tuple space model in Multichain with one worker

The variation of this experiment, as for the other explored models, was performed using one more device which functioned as a worker. In both experiments, communication between constrained devices and the tuple space server was done using CoAP, and communication between that server and the Multichain was done using REST.

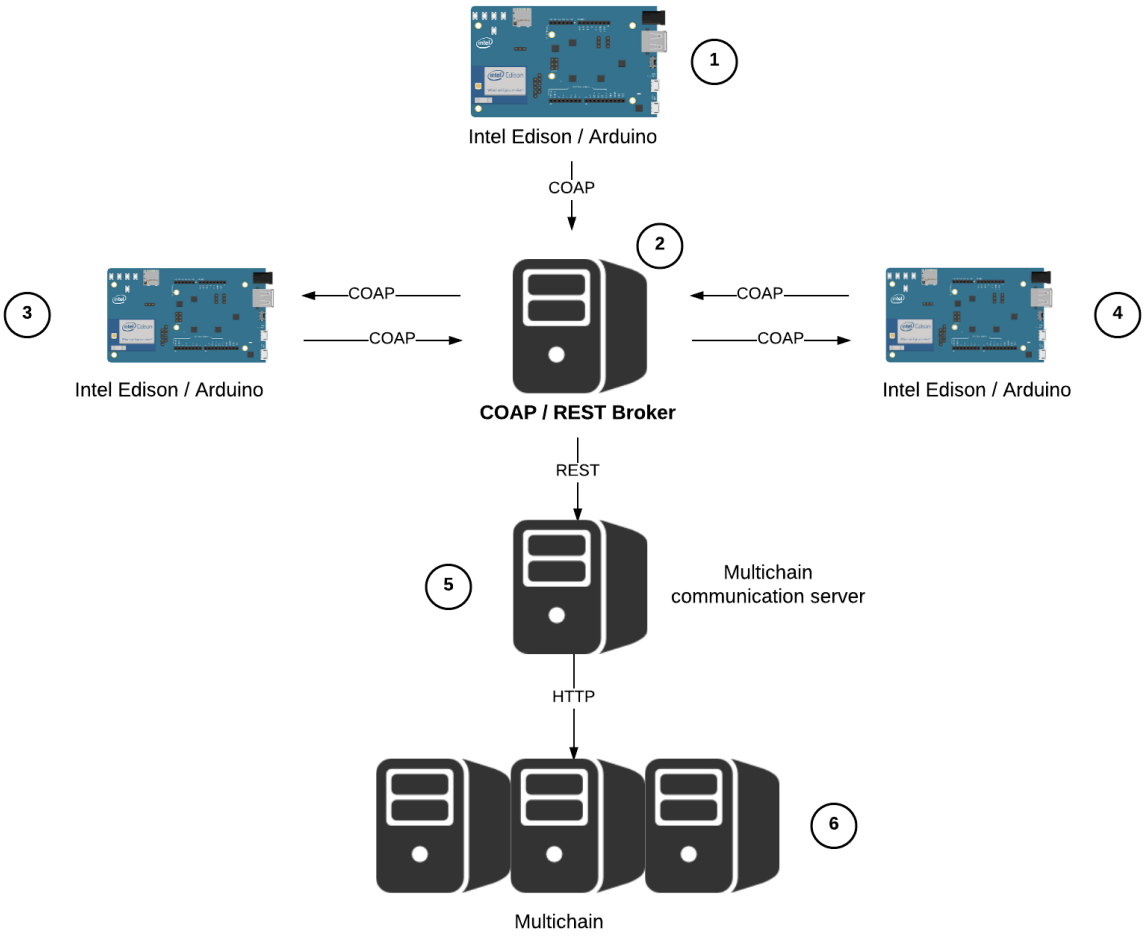

Figure 4-7. Tuple space model in Multichain with two workers

4.2. Process-Oriented model

One of the key contributions of this work is the development of a new process-oriented coordination model for IoT. This model aims to reduce the components involved to the minimum and reduce communication costs by using CoAP with elementary operations. The new model is fault-tolerant and is based on the concept of proxying. Using proxies to take

advantage of some of the benefits offered by CoAP, the communication can be handled very efficiently. Proxies are in charge of connecting two or more devices and they have the ability to cache responses of previous requests. This is one of the most attractive characteristics that give the model part of its value and contribution.

The concept behind this model is to deploy proxy servers within the system. These proxies can be other constrained devices or more robust servers depending on the system's size and workload. For the experiments conducted in this study, robust servers were used. Each IoT device would communicate only to the proxy server. When doing so, the device, using CoAP, tells the proxy which service it intends to consume. Then the proxy can communicate with another device, obtain the result and respond to the original requester. Figure 4-8 shows the basic operation of the system, in which the devices do not have to know about each other, they only need to know about the proxy who connects their ports and creates the channel for them to communicate. In this example, the device number one is the consumer and the device number three is the producer.

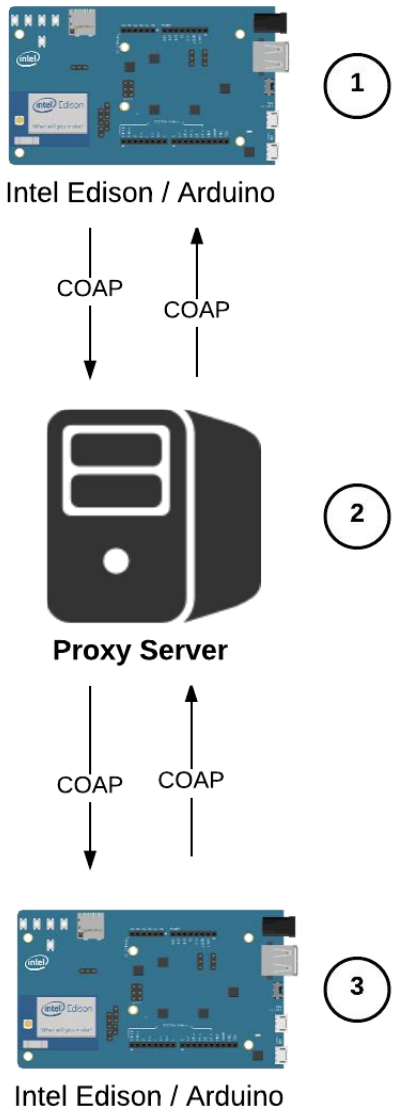

Figure 4-8. Process-Oriented model with one worker

Figure 4-9 shows the same model but using another device as a worker or producer. The proxy only needs to know about the producers, it does not have to know about the consumer, and none of them has to be aware of the other devices.

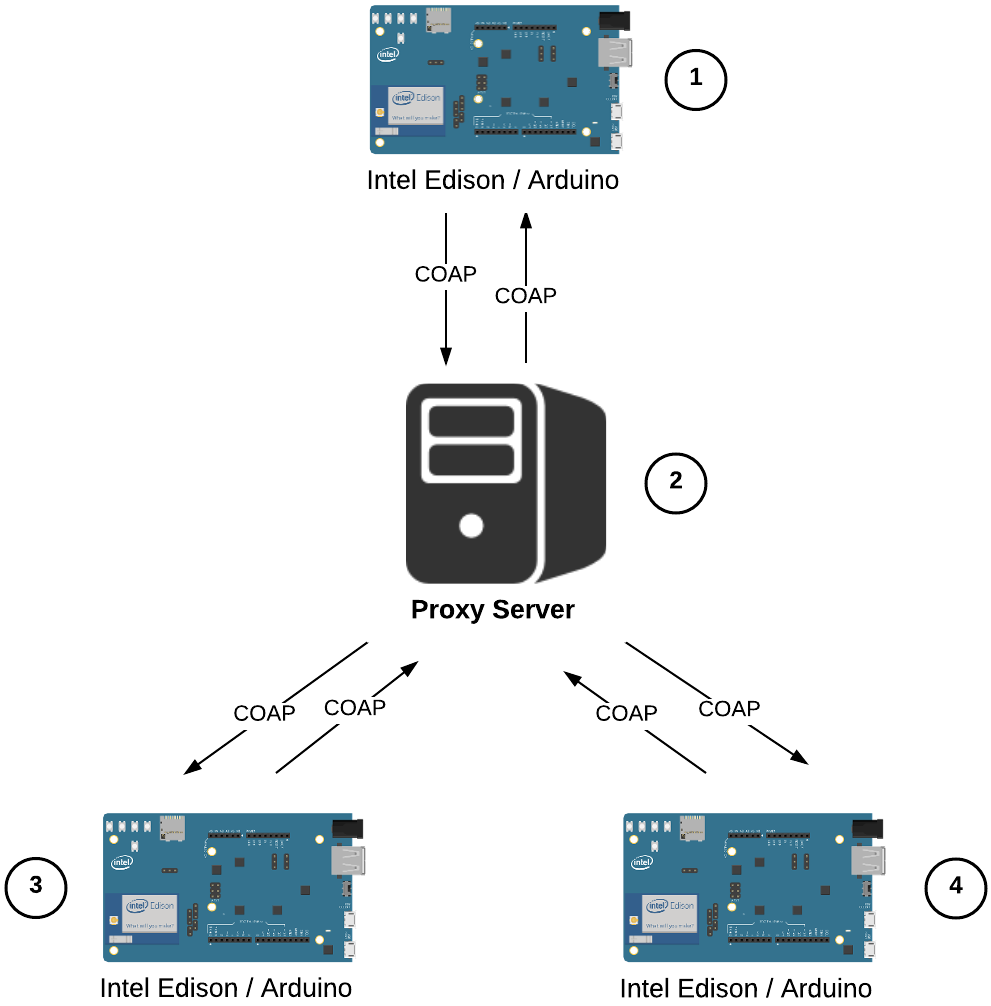

Figure 4-9. Process-Oriented model with two workers

This new process oriented model treats devices as black boxes. Proxies know what services are exposed by each device so they can call them when another device requests it. When the system is too big for one proxy, several can be deployed; these would allow more devices to coordinate. Furthermore, this model allows any device to be a proxy, which means that constrained devices could do computation and work as coordinators. However, for the present work, simple systems with only one dedicated proxy were developed. A key difference between this model and the data-oriented ones is that the device through which all communication passes can make decisions based on that, allowing it, for example, to prevent

multiple calls that might generate the same result and incur a waste of resources by processing repetitive tasks.

CHAPTER 5

IMPLEMENTATION

As seen in the previous chapter, the different models proposed were comprised of various components. Each of them had a different configuration to adjust to the implementation characteristics. The applications running on the constrained devices and the components they communicate with were developed in Golang [69], which was used here because it was designed to run on multiple platforms and it has native support for concurrent activities. As CoAP is used for communication in every implemented model, the library go-coap [70]was employed.

Four models were designed and implemented, three data-oriented and one process-oriented. This chapter explains how those models are implemented.

5.1. Data Oriented models

5.1.1. Tuples

Tuples can have any number of parameters, and they must be transported in the payload section of the CoAP packet. Values are comma separated to reduce the amount of transferred data. Tuples that require processing are generated by the producers and sent to the tuple space. Consumers would get those tuples, process them and form a new tuple with the result that is sent back to the tuple space. From there, any entity that requires that result can access it.

5.1.2. Tuple Space

The key part of this group of models is the tuple space server. Whether it saves the tuples itself or somewhere else, the tuple space server is the most important component and exposes the two basic Linda operations *in* and *out,* which are identified by two different URIs. The tuple space was designed to be efficient, and allow multiple devices to interact with it almost simultaneously. Figure 5-1 shows the function that creates the tuple space and starts responding to requests in the resources identified by the URIs */in* and */out*. When an entity

requests a tuple, if it exists the server would respond with a CoAP message with the code 2.05 and the tuple in the payload. If it does not exist, the payload will be empty and the message will have the code 4.05. These codes allow clients to make fast validations over the responses from the server.

```
func StartServer() {
    space = tupleSpace.NewSpace()
    log.Fatal(coap.ListenAndServeMulticast("udp", "224.0.1.187:5683",
        coap.FuncHandler(func(l *net.UDPConn, a *net.UDPAddr, m *coap.Message) *coap.Message {
            if len(m.Path()) > 0 {
                switch m.Path()[0] {
                case "in":
                    res := inTuple(m)
                    return res

                case "out":
                    res := outTuple(m)
                    return res

                default:
                    res := notFoundHandler(m)
                    return res
                }
            } else {
                res := notFoundHandler(m)
                return res
            }
            return nil
        }),"en1"))
}
```

Figure 5-1.Function to create the tuple space and start serving CoAP requests

```
func rdTuple(m *coap.Message) *coap.Message
{
    tuple := space.Read(m.Payload)
    payload := []byte(tuple)
    res := &coap.Message{
        Type:      coap.Acknowledgement,
        Code:      coap.Content,
        MessageID: m.MessageID,
        Token:     m.Token,
        Payload:   payload,
    }
    res.SetOption(coap.ContentFormat, coap.TextPlain)
    return res
}
```

Figure 5-2.Example of a function returning a tuple in a CoAP message

Figure 5-3 depicts a basic example of how the communication is done with the tuple space in the three models. Even though the tuple space is shown as a separate component, for one of the implementations the tuple space is in the same server as the tuple space server, for the others it is located in the Multichain or the relational database. In this example only two

devices are coordinating, one producer generating tasks that need to be processed, and a consumer or worker. This is the description of each of the steps:

- 1, 2 and 3: The producer generates a work tuple, sends it to the tuple space server in a CoAP request. When the tuple is added to the tuple space, the tuple space server responds with the CoAP request confirming the result.
- 4, 5 and 6: The consumer sends a request to the tuple space server requesting a tuple. The tuple space server searches the tuple in the tuple space and responds to the CoAP request with the appropriate information.
- 7, 8 and 9: After processing the tuple, the consumer sends the tuple back to the tuple space server. When the tuple is added, the server responds.
- 10, 11 and 12: The producer sends a request querying for the result tuple to the tuple space server. If the tuple space server finds it, it sends it back to the producer, otherwise, a not found message is sent.

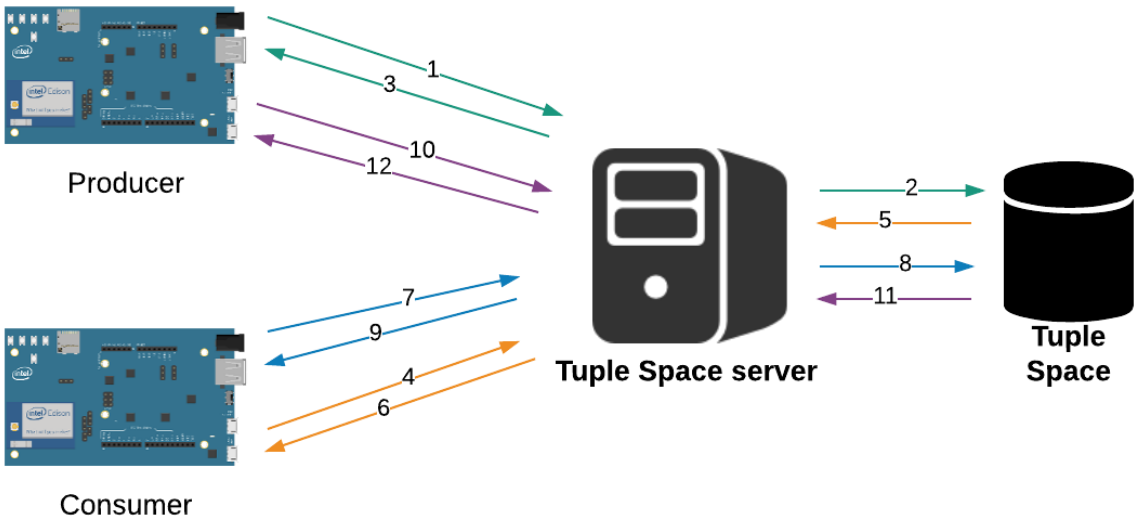

Figure 5-3.Communication flow with the tuple space

The previous flow shows a simple example of how the communication would be done in this system. However, there are some points to keep in mind when implementing this coordination model:

- As these kinds of models have to be aware of the data being sent, the tuples used must be defined according to the objective of the system. A suitable way for identifying work and result tuples must be implemented.
- Depending on the system requirements and characteristics, the periodicity for the worker to query for work tuples and the producer to query for results needs to be adjusted. More powerful devices might query more often for queries than more constrained devices. In this system, the only way for components to discover new tuples is by querying the tuple space server.
- For evaluating the system's performance, the time between steps one and twelve should be measured. The complete process is from the first work tuple is generated until the last result tuple is generated.

For the tuple space, it is important to store the tuples in a correct way, it order to be able to be searched and accessed efficiently. Three ways of storing tuples were developed in the current work:

- RAM: Tuples were stored in a Go list. For adding tuples, they are added on top of the list. For searching them, the whole list is iterated until the specified value is found.
- Relational database: The communication between the tuple space and the database was done through a centralized point, a web server exposing REST operations. That web server used a JDBC for connecting to the database, which oversaw performing the CRUD operations to search, retrieve and add tuples.
- Blockchain: To maintain the implementations very similar, this model was implemented using a central REST web server as well. That server was in charge of performing the CRUD operations on the blockchain. However, as the data in the blockchain cannot be altered or deleted, the tuple space server would

identify each tuple before sending it back to the clients. When a tuple is taken, the tuple space server marks it as deleted and does not return it anymore.

-

5.1.3. Coordinated entities

The coordinated entities for this model are the consumers and producers. Depending on the system being implemented, they must generate tasks and results. It is up to the system designers and developers to decide how the coordination is required and how it will be performed. As mentioned before, these models have the computation and coordination parts tightly coupled, which is why every system needs to be adjusted to use the model. For example, a device with a stable source of energy could query tuples continuously. On the other hand, for a device with limited power, that would be a waste of resources.

5.2. Process-Oriented model

The process oriented model developed in this work is based on the concept of connecting interfaces of black boxes to allow them to coordinate. To achieve this, coordinators were conceived as forward-proxies, components that receive requests, find the most suitable worker and forward the work. Contrary to the previous implementations, the definition of the format of the data being transferred is not relevant. The worker is developed as a CoAP server, and the requester sends a CoAP request. The proxy server sees this exchange of CoAP messages and takes decisions based on that, not on the content of the message.

Figure 5-4 shows a basic communication flow example between devices and the proxy server. Fewer communication steps are required for a requester to obtain the result of an operation than in the previous model. To maintain the communication effectively and stably, the model was implemented asynchronously: the responses for the CoAP requests do not contain the result of the operation, that is sent in another operation. For this example, only one requester and worker are described. The following is the description of each of the steps required for a coordinated operation:

- 1 and 2: The device requiring the result of the task sends the CoAP request to the proxy server. The proxy server responds to the request immediately

acknowledging the original request. As the proxy server implements a cache functionality, when the result of the request number one is cached, the result will be sent in the response number two and no more operations will be made, reducing communication and processing costs.

- 3 and 4: The proxy server sends the request to the worker, who immediately responds to acknowledge.
- 4 and 6: After the worker is finished processing the task, it sends the result in another request to the proxy server which acknowledges immediately.
- 7 and 8: The result is sent to the requester who acknowledges.

Both worker and requester act as a server and client at the same time. The requester must expose a resource to receive the result of the worker, creating an asynchronous operation. The proxy server knows about all the resources both components expose and oversees deciding where to send each request. The proxy server has to be previously loaded with the list of resources exposed by the devices. Each resource on the list is identified by its URI. That way when requests come, the proxy knows where to send them as they also have a URI. In this context, devices do not need to know about other devices, but they do need to know the URI of the exposed service.

To correctly send the requests to the forward-proxy, the model uses the proxy-uri CoAP option, which is used to set the final URI destination.

Figure 5-4.Communication flow with the proxy server

Figure 5-5 shows an example of how the function of the proxy server is called. The URI consumed in the proxy is always **/fwd**, and the actual URI that is required is sent in the proxy-URI option. This example also shows how every CoAP call is made within the system.

```
func sendCoapMsg() *coap.Message{
    req := coap.Message{
        Type:      coap.PUT,
        Code:      coap.Confirmable,
        MessageID: 123456,
        Payload:   []byte("test"),
    }


    req.SetOption(coap.MaxAge, 3)
    req.SetOption(coap.ProxyURI, proxyUri)

    req.SetPathString("/fwd")

    c, err := coap.Dial("udp", host)
    if err != nil {
        log.Fatalf("Error dialing: %v", err)
    }

    rv, err := c.Send(req)
    if err != nil {
        log.Fatalf("Error sending request: %v", err)
    }

    return rv
}
```

Figure 5-5.Example of a function sending a CoAP request to a proxy server

The asynchronous characteristic of the model is key for ensuring a stable communication. When the CoAP request is sent to the proxy the first time, many actions must take place, one of them being the processing of the task, which might take a long time. Consequently, it is not feasible to send the result of the work in the following CoAP response. That restriction is because of network constraints and because it is not desirable to keep the client waiting for a response. To address this situation, the communication is asynchronous, meaning that when a client requires processing a task, it exposes a resource to receive the result.

5.3. Cache

One of the major advantages of this model is the usage of a cache system. Following the guidelines for handling a CoAP cache [26], requests of processing tasks and their responses can be cached by the proxy. This allows the proxy not to make unnecessary requests to the entities, saving communication costs, resources and improving response times.

When a result request (asynchronous response) is received by the proxy, it matches it with the original request and caches it if possible. Responses follow the freshness model, which defines whether a response is fresh or not. If it is fresh, the proxy can return it in the first response, avoiding any further calls or operations. The way to determine the freshness of a response is for the origin server to set an expiration time. This is the time for which the response is valid, which is done using the CoAP option max-age.

5.4. Concurrency

As the proxy server is designed for responding to every request in the least possible time, it is not efficient to have only one thread acting as a server (to serve the forward requests) and a client (to forward the requests). With only one thread, the system would stop responding while waiting for the process to make a particular call to another device. This issue was addressed using the concurrency characteristic of Golang [71]. Figure 5-6 shows how the parallel system was implemented; the proxy server is comprised of two threads: One to act as a server and respond to every request, and one to serve as a client and call the required resources. They communicate over a channel and are always running. The requests are always received by the same thread (routine in Go) that pushes them into the channel. The client thread reads the channel and makes the necessary call. As the two processes are handled independently, every request is going to be responded to almost instantaneously, allowing the clients to continue their work while waiting for the response. Figure 5-7 shows an example of how routines are created in Go and how the channel is established between them.

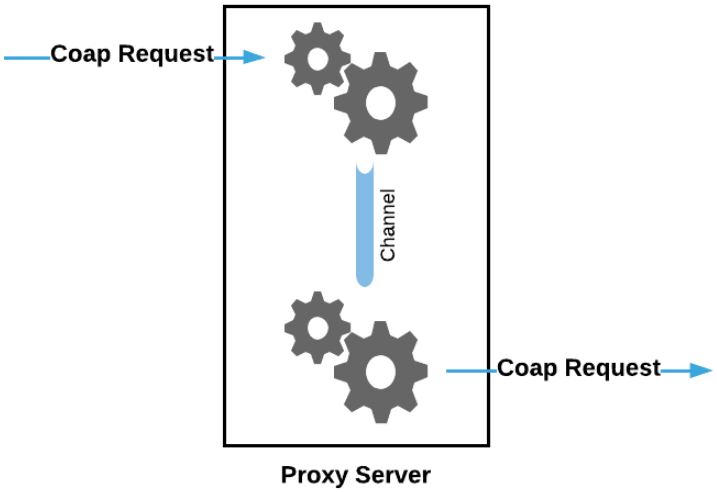

Figure 5-6.Concurrency in the proxy server

```
func main() {
    c := make(chan *coap.Message, 100000)
    go startClient(c)
    startServer(c)
}
```

Figure 5-7.Example of go routines and channels.

CHAPTER 5

EVALUATION

As the Intel Edison was the device used for the current experiments, a basic evaluation of these devices using CoAP was conducted. The idea of this evaluation was to determine how these devices behave with high loads and with different message sizes. The objective of this performance evaluation was to determine what loads these devices can handle depending on the message sizes being exchanged.

A set of experiments were put in place: The Edison device would expose a CoAP server that would only read the payload and respond immediately. Using JMeter [72] a high number of requests per second was generated for 60 seconds with different payload sizes: 1, 20, 100 and 500 bytes. The average time the Intel Edison took to respond every request was measured.

Figure 6-1 shows the result of this evaluation. The x-axis of the graph is the average response times for the defined transactions per second (TPS) that are represented in the y-axis. The graph shows the results of all the experiments with the different message sizes. Although they all have the same trend, it is noticeable how smaller message sizes tend to have a lower response times. However, big message sizes don't always have higher response times. For 200 TPS the difference between response times is not too big. Furthermore, in the graph, for every defined TPS, the distance between the different payload sizes stayed very similar; the growth of the graph is not exponential.

In conclusion, the behavior of the Intel Edison was as expected; as more TPS were set, the response times increased as well. Although the message sizes increased the response times, they did not drastically affect the device's efficiency. It can also be appreciated that the Intel Edison running a CoAP server is capable of handling 200 TPS with payloads of 500 bytes and respond in an average of 120ms.

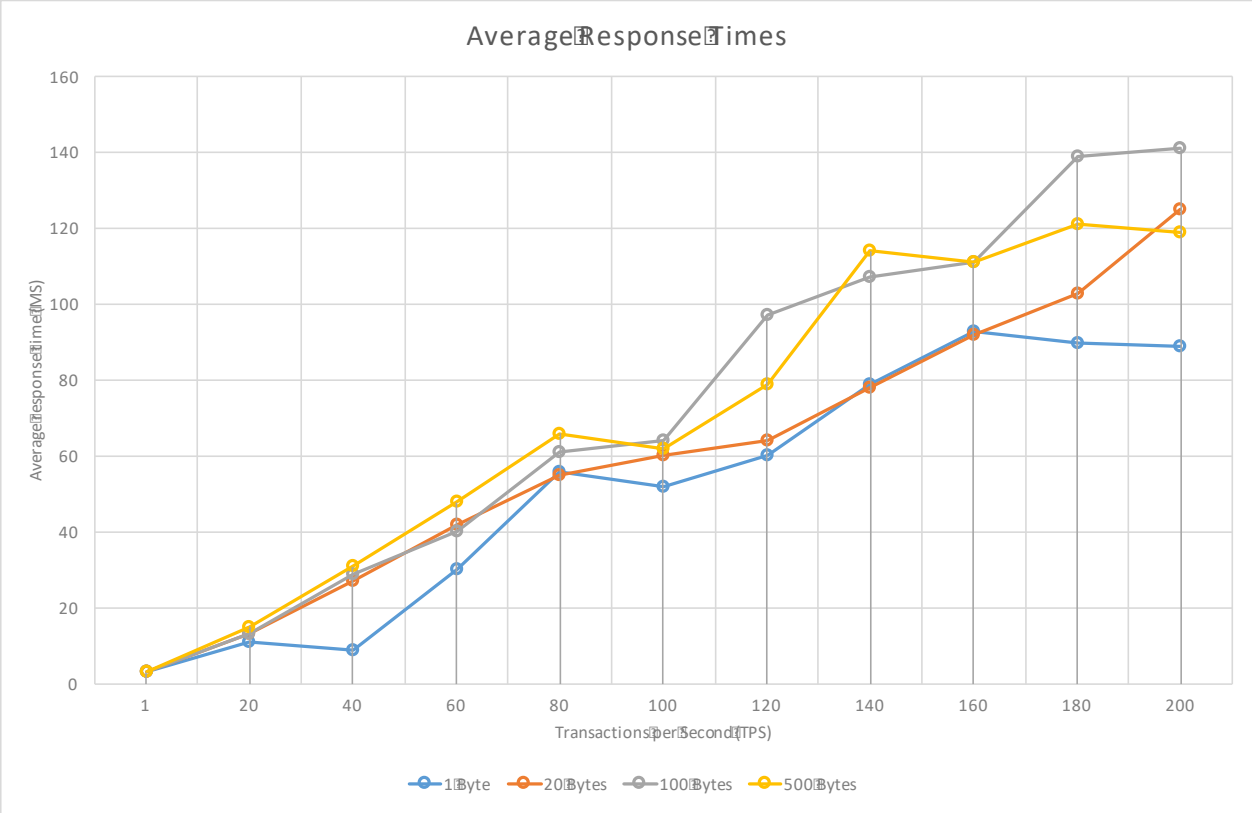

Figure 6-1.Average response times for 1, 20, 100 and 500 bytes payload messages

The main objective of the development of the previously explained coordination models is to allow devices to coordinate with each other, creating an IoT system where the resources of every device can be used most efficiently. That is why the purpose of the experiments conducted was focused on evaluating the effectiveness of each of the described models. This evaluation was made by measuring the time it took for the whole system to complete a series of complex tasks. This chapter explains how that measurement was made and the results for each model.

To evaluate the performance of the systems, a suitable complex task was implemented: The calculation of prime numbers was used. This job consists of calculating the highest prime number smaller than a given maximum value max. The algorithm was implemented in a basic way, iterating through every number from 1 to max and validating if the number was prime or not. To increase the complexity, that task was run several times. For a given a number n, the

complete process would be to calculate the highest prime number for every number from 2 to n. For example, given n=10, the process would have to calculate the highest prime number smaller than 10, the highest prime number smaller than 9, and so on. This would result in computing the highest n-2 prime numbers.

```
func calcPrimeNumber(max int) int {
    var result int

    for i := 2; i < max; i++ {
        for j := 2; j < i; j++ {
            if i%j == 0 {
                break
            } else if i == j+1 {
                result = i
            }
        }
    }
    return result
}
```

Figure 6-2.Algorithm to calculate prime numbers

According to that task, the complexity is given by the *n* value. The higher it is, the more prime numbers must be calculated, the more processing the system needs, and the more time the complete process takes. All the models used this same task with different values of *n* within the range of 50 to 5000. For evaluation purposes and to have more consistency, each run was performed three times. All the experiments were run using one and two workers; it was expected to have a better performance with two workers as there are more resources available in the system. Table 6-1 shows the summary of all the experiments performed and the following sections show the results obtained for every model implemented.

Table 6-1. Experiments summary

Type of coordination model	Coordination model	Experiments
Data-Oriented	Tuple space in a dedicated server (RAM)	Three equal experiments
	Tuple space in a relational database	Three equal experiments
	Tuple space in the blockchain	Three equal experiments
Process-Oriented	Proxy-based	Three equal experiments

6.1. Data Oriented models

For all the data-oriented models implemented, only two operations to manage tuples were defined: the basic Linda functions: in and out. For the experiments designed in the current work, tuples were comprised of two values: one value that means if the tuple needs to be processed or is a result, and the other is the actual content. To identify the tuple as result, the first value is the letter R; if the tuple needs to be processed, the letter W is used. The idea is that at the beginning many more tuples are W but as workers process them, they would reduce while the number of R tuples grow.

For this specific task, the producer is in charge of generating work tuples, each of them containing the values from 1 to n. So, for example, if n = 50, the producer has to generate 50 tuples, and the worker must obtain them and generate one result tuple for each of those 50 work tuples. The time of the task was counted since the moment the producer generated the first work tuple until the last result tuple was generated by any worker.

6.1.1. Tuple space in a dedicated server

Three equal experiments were performed for this model, each using one and two workers. Figures 6-3, 6-4 and 6-5 show the results of the tuple space model storing the tuples in RAM in a dedicated server. The X-axis of the graph is the value of *n*, or the number of iterations for the previously explained task. The Y-axis of the graph is the time in milliseconds that the system took to complete the defined iterations. In overall, the graphs show that the time it took the system to finish the processing was much lower with two workers than one. For 5000 iterations, the average time the system with one worker took was 358,935 ms, whereas with two workers, that time is decreased to 77,855, which is 20% of the original time. This percentage changes for the other iterations, in which there was also a reduction in time. For smaller iterations, the difference between one and two workers is barely noticeable. However, in this experiment, the system with two workers always had less time than the system with one.

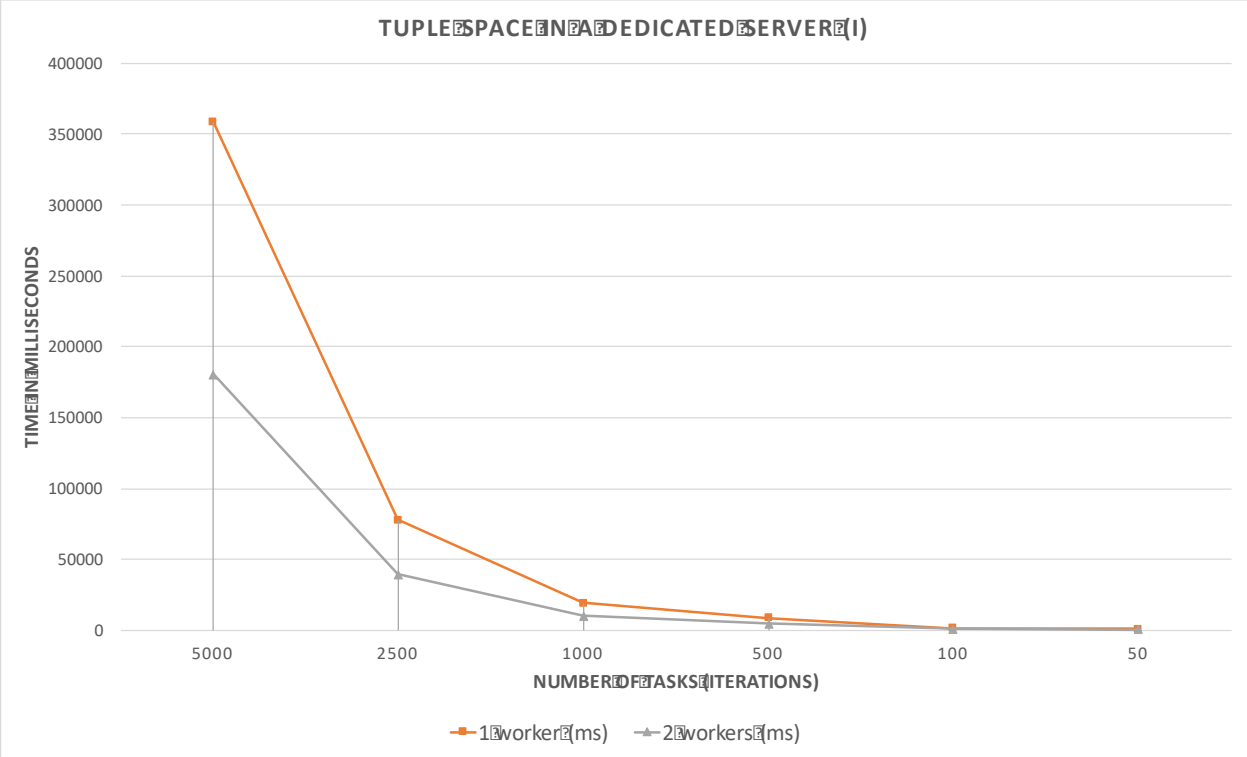

Figure 6-3.Tuple space in RAM model results (I)

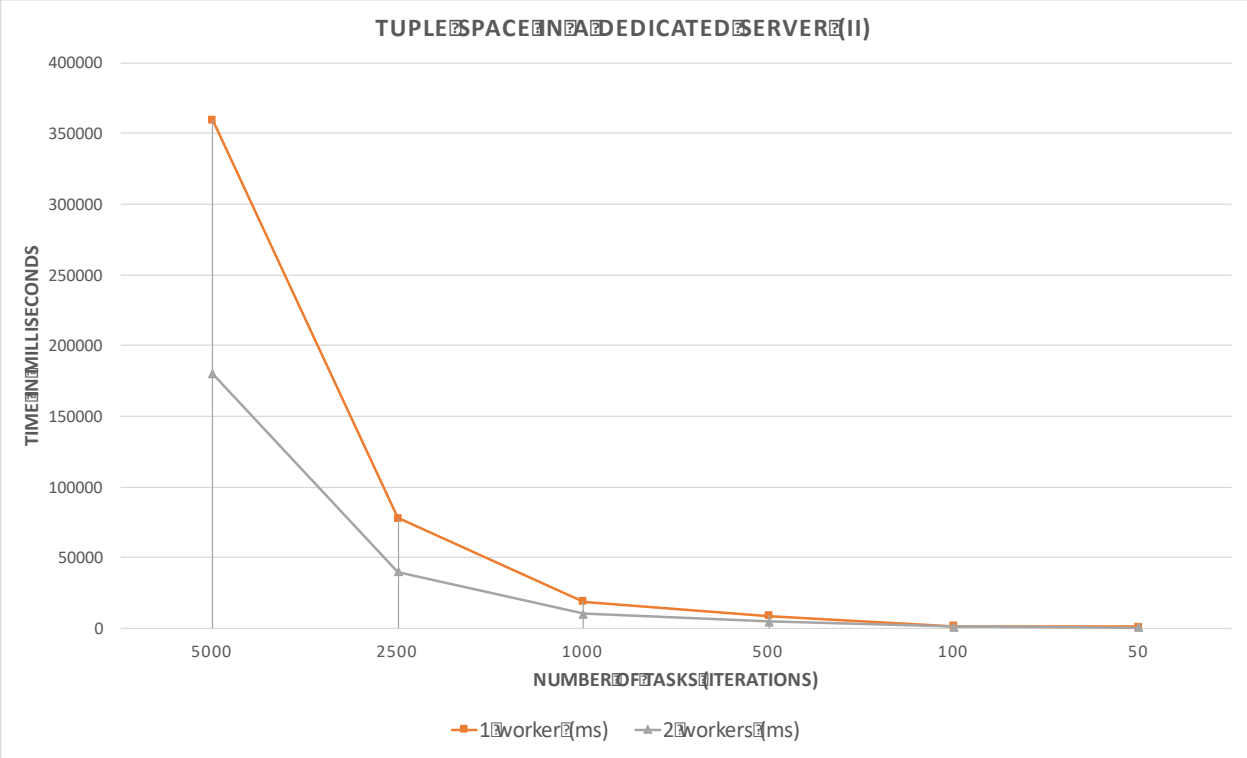

Figure 6-4.Tuple space in RAM model results (II)

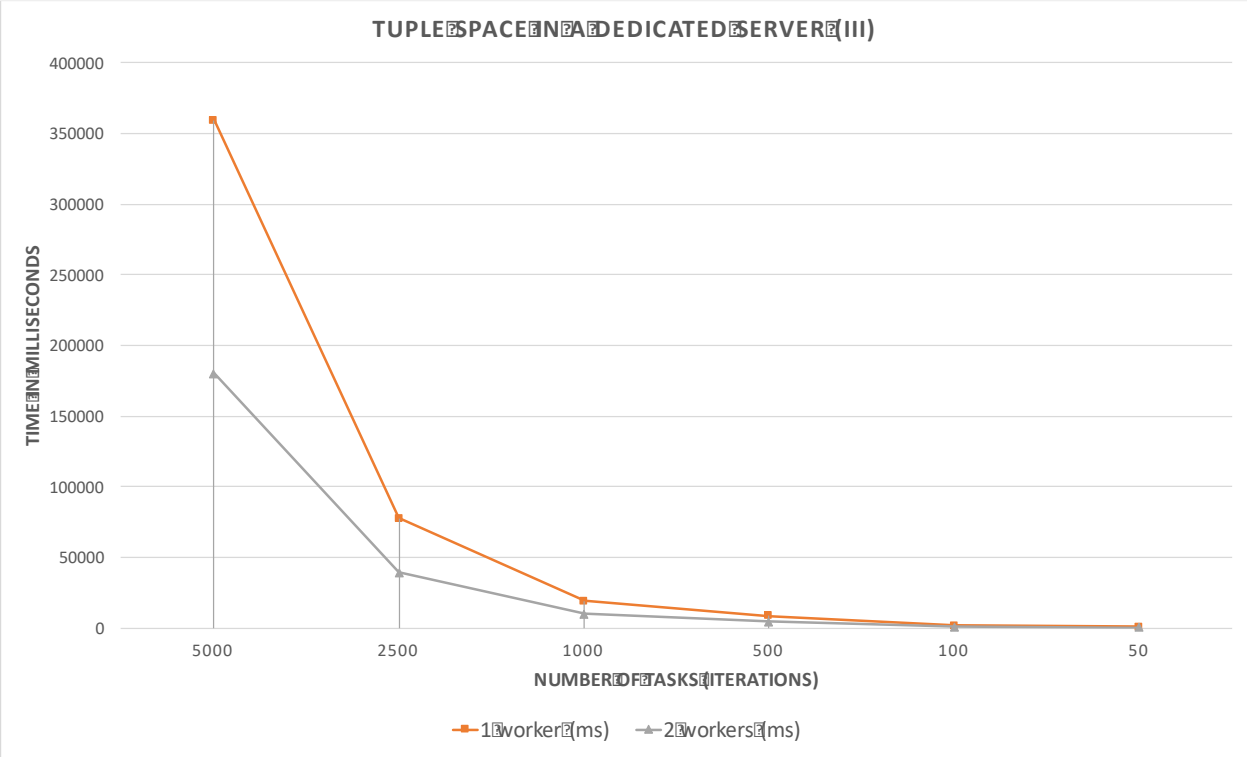

Figure 6-5.Tuple space in RAM model results (III)

6.1.2. Tuple space in a relational database

In these experiments, there was an additional component: the relational database. This component could constitute a bottleneck point and make the system incur more communication costs. For figures 6-6, 6-7 and 6-8 the X and Y axis are the same as in the previous experiments.

It is noticeable that the difference between using one and two workers, for the largest number of iterations (5000), the system with one worker took an average of 812,613 ms to complete, whereas with two workers, that time was reduced by almost 50% to 450,049 ms. For the smallest number of iterations (50), the two workers model was also faster; the reduction in time was about 66%.

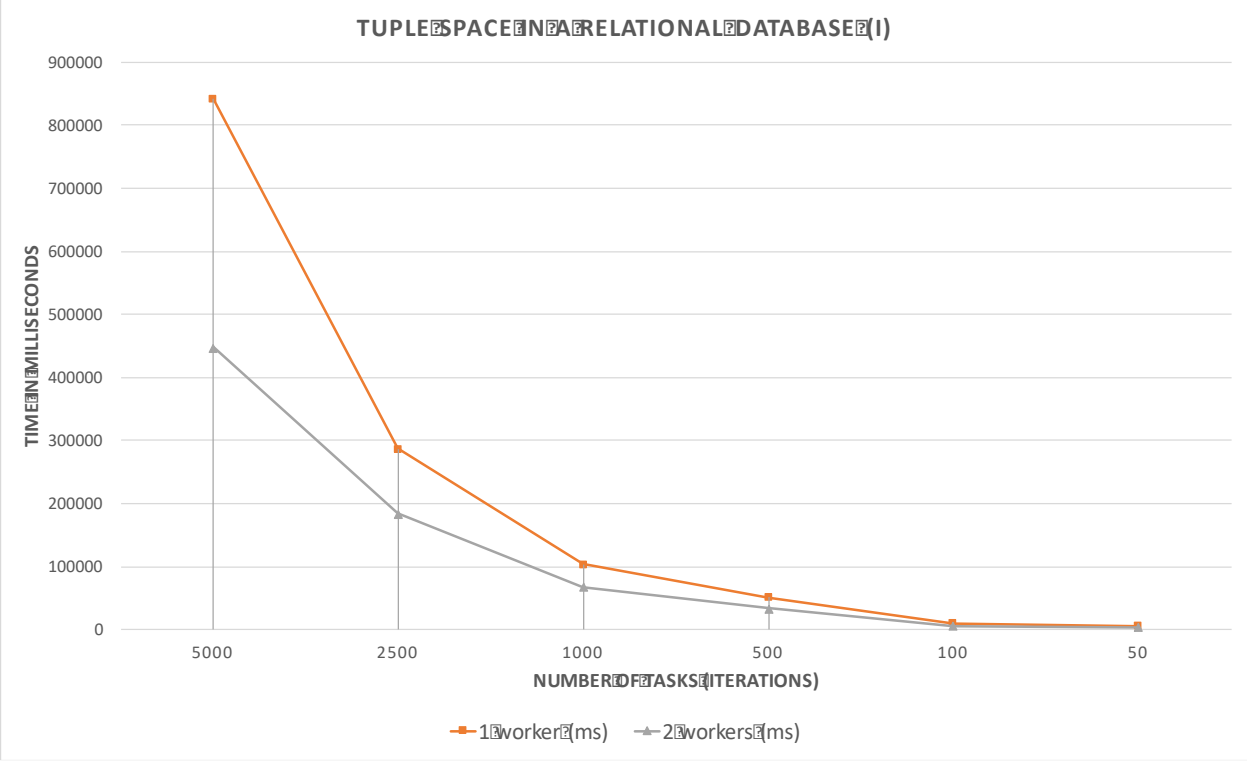

Figure 6-6.Tuple space in a relational database model results (I)

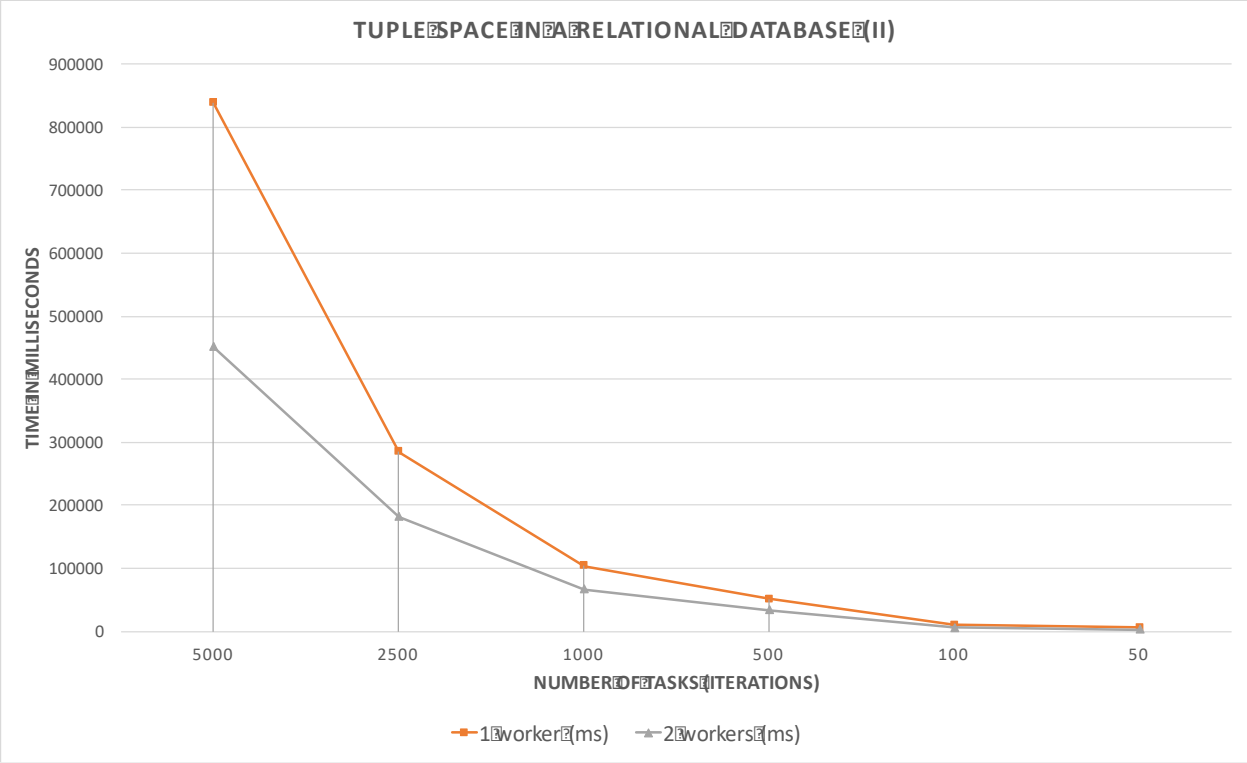

Figure 6-7.Tuple space in a relational database model results (II)

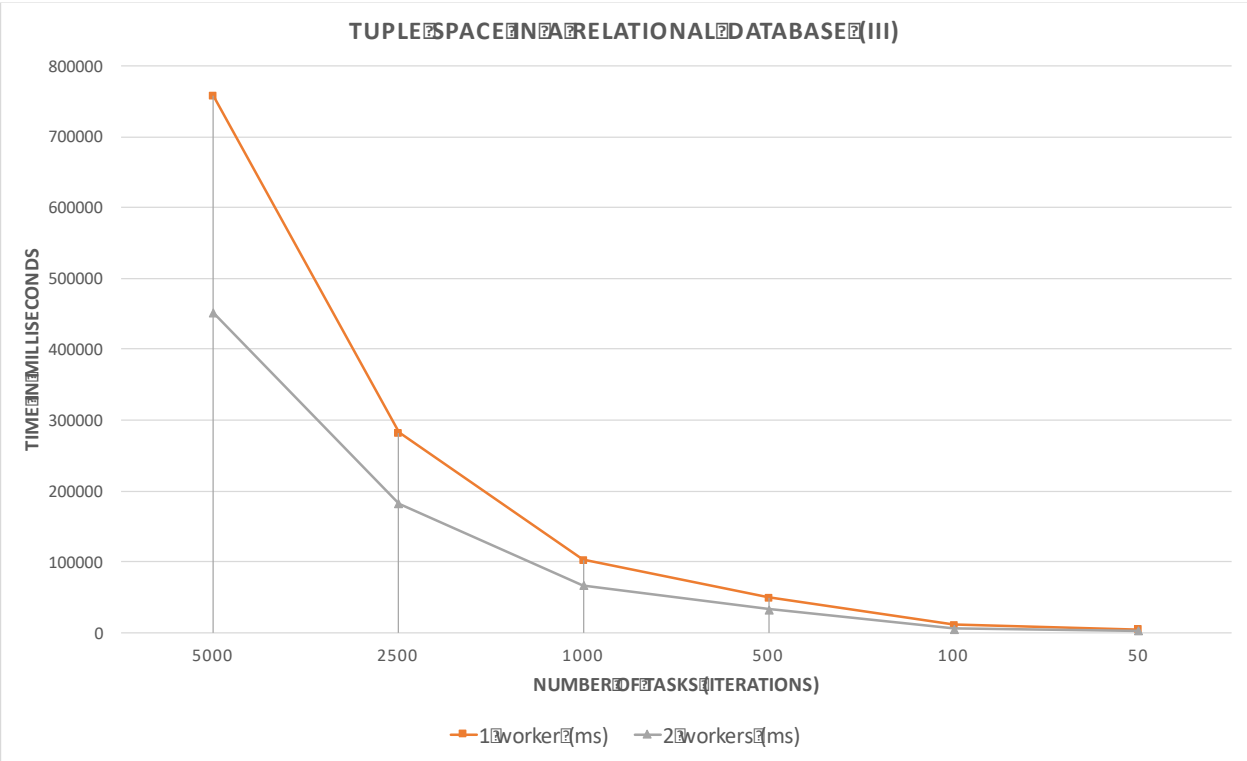

Figure 6-8.Tuple space in a relational database model results (III)

6.1.3. Tuple space in the blockchain

Using Multichain to store the tuples had a similar result as storing them in the relational database. As in the last two experiments, the time was shorter when two workers were in the system instead of one. The most noticeable difference, as in the previous experiments, was obtained with the most iterations (5000), on which in average, having two workers reduced the total time by 45%.

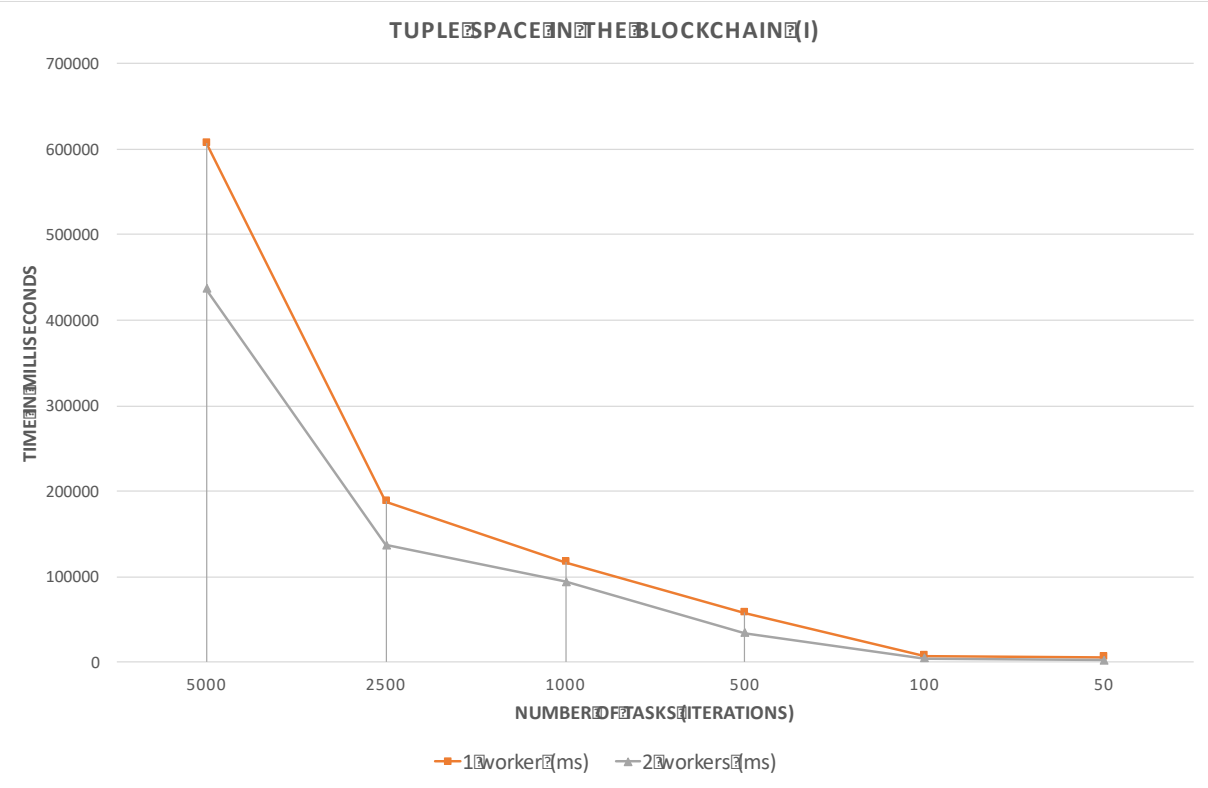

Figure 6-9.Tuple space in the blockchain model results (I)

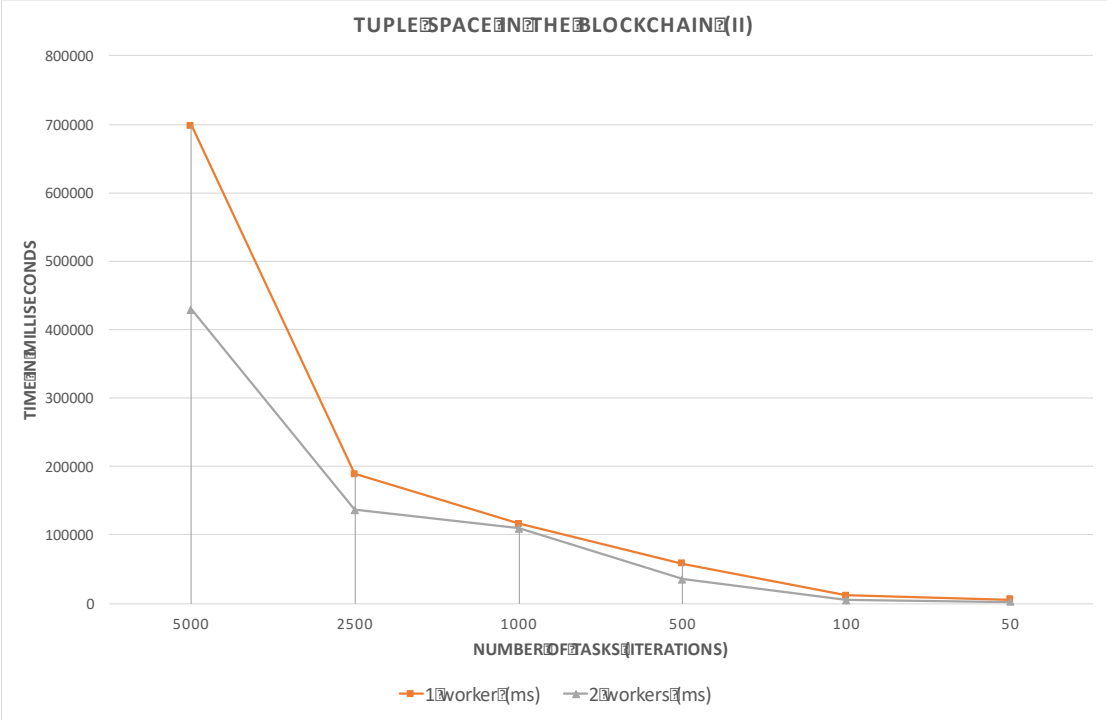

Figure 6-10.Tuple space in the blockchain model results (II)

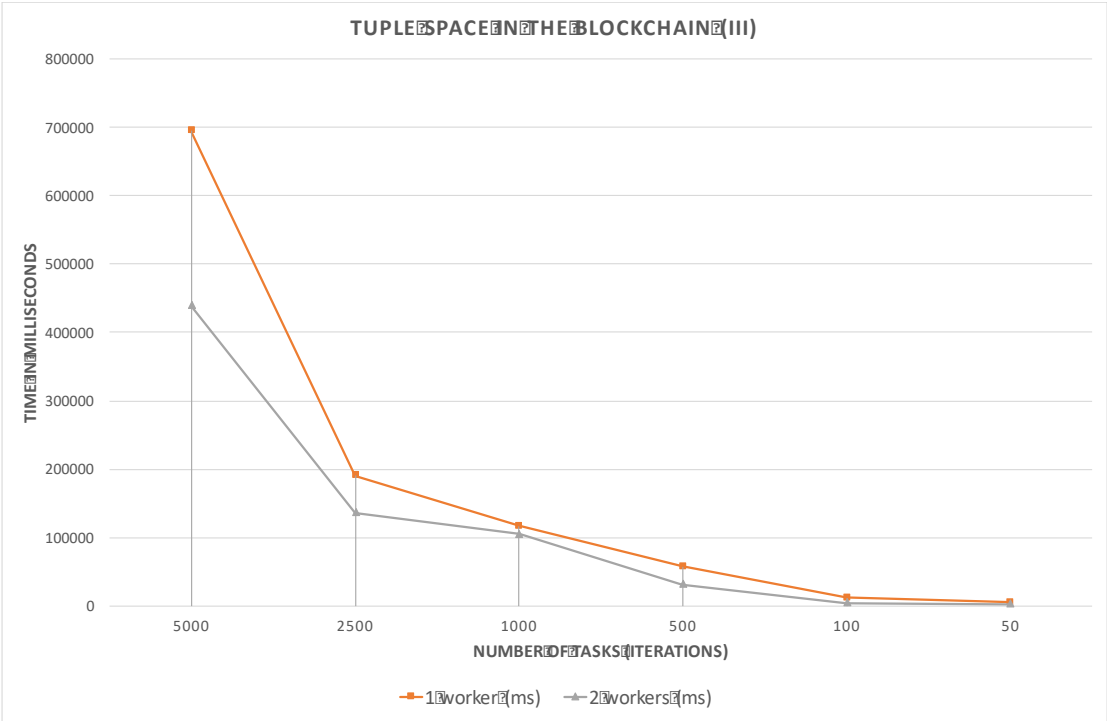

Figure 6-11.Tuple space in the blockchain model results (III)

6.1.4. Data-Oriented models summary

The three models had an expected result in that they all performed better with more workers. As seen in the Figure 6-12, the relational database and blockchain storage had a similar performance with one worker, with the relational database being the model that had the smallest times. The model where the tuples were stored in RAM performed much better as it kept the times lower in every case. This behavior suggests that the communication costs to transfer the tuples to another component affected the overall times. Furthermore, it also demonstrates that writing, searching and reading information from RAM is faster than from a relational database or blockchain.

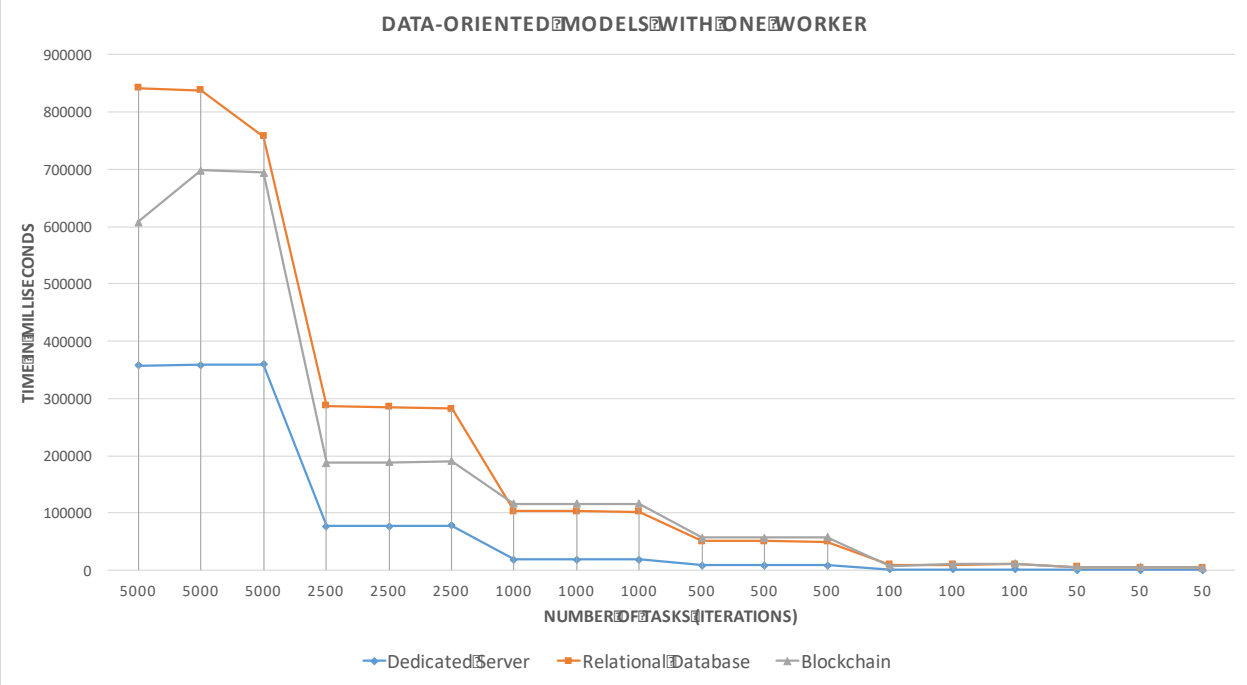

Figure 6-12.Data-Oriented models with one worker

With two workers, the three models had a better performance and their behavior was very similar to the previous scenario. The relational database and blockchain models performed worse than the dedicated server. Although the relational database performed better for larger number of tasks, the blockchain model had better times in some cases. In average, storing the tuples in RAM was 60% faster for the system than in the relational database and blockchain.

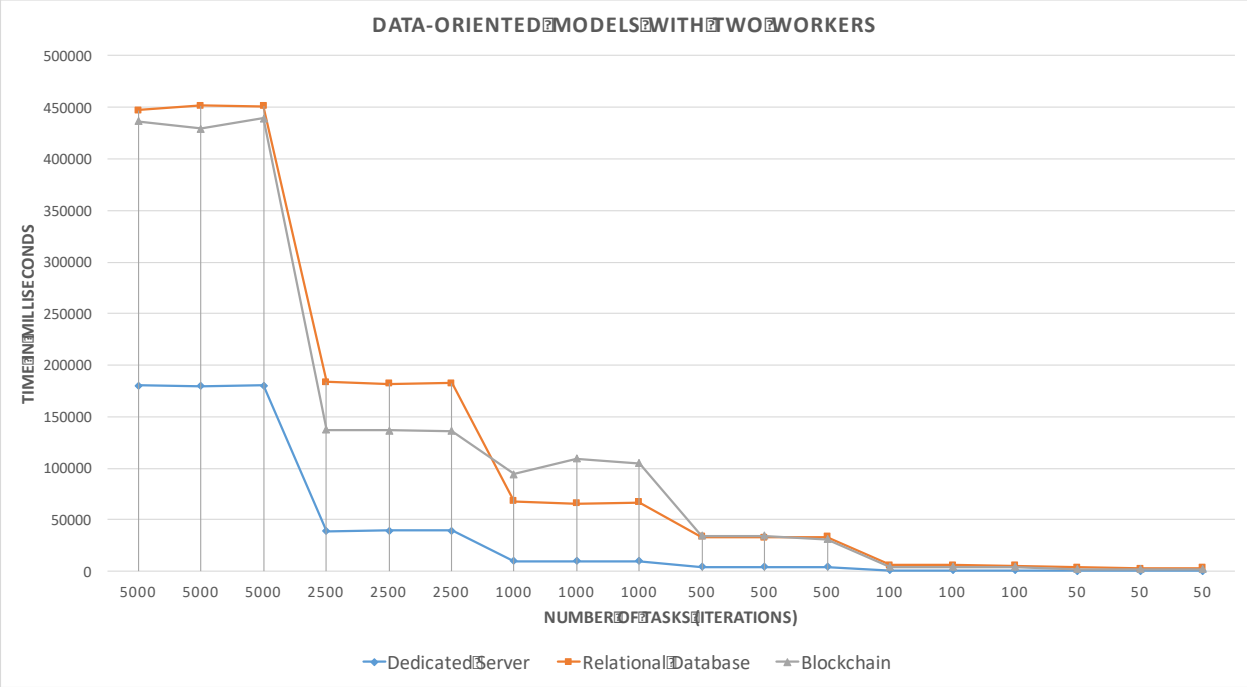

Figure 6-13.Data-Oriented models with two workers

6.2. Process-Oriented model

For this model, the same task for calculating prime numbers was used. However, it had some changes as the way the model works is different. In this model, the workers appear as servers. They expose an operation to calculate the prime numbers, when that operation is called, the algorithm exposed in the figure 6-2 is executed and the result is returned. As mentioned before, the result is sent back in another operation, which is why the producers act as servers as well as clients. Given a value of *n*, the producer will make *n* calls to the proxy, each of them with the numbers from 1 to *n*, the proxy will forward them to the workers and they will send back the response to the producer through the proxy. The bigger the value of *n*, the more tasks must be performed and the more resources are required.

Besides the already mentioned differences between process-oriented and data-oriented models, what characterizes this model is the usage of the cache. As the proxy has both request and response, it can keep them in the cache and avoid future calls. The experiments were run three times for each value of *n*. In the first of those times, the proxy had no cache stored. In the

second and third time, the proxy already had an entirely fresh cache, which drastically improved the time the system took to finish.

To evaluate the performance of the model under a more secure environment, two variations were implemented: One with the payload in clear text and another one encrypting the payload using a data encryption specification called AES [73] with a key of 32 bytes. For the second variation, the producer encrypted the value of *n* and sent it to the proxy. When the workers received it, they decrypted it with the previously shared secret key. When the workers had the result, they would follow the same procedure using the same key.

The purpose of implementing AES was to recreate a secure IoT environment. As proxies do not require knowing about the data being transferred, they should not necessarily see it in plain text. Implementing this approach generated two drawbacks in the overall system's performance:

- Encrypting and decrypting the payload for each CoAP message requires more processing and therefore, resource consumption in the constrained devices. This would increase the time the system takes to complete the work.
- As all the payloads are encrypted and therefore, different, there are no identical requests, which means that, even though the system is caching responses, they are not going to be returned to any request.

In both variations, the time was counted since the moment the producer sent the first request until it received the last response. The following section shows the results for the two variations of the model implemented.

Using AES

Figures 6-14, 6-15 and 6-16 show the results of the system with one and two workers using AES to encrypt the payloads. The X-axis represents the value of *n* used for each of the experiments. The Y-axis represents the time in milliseconds the system took to complete the task. When two workers were used, the times were lower. The biggest difference can be seen for 5000 iterations: the average time the system with one worker took to finish was 330,088 ms,

whereas with two workers the time reduced to 173,674 ms, a reduction of around 50%. For the rest of iterations, clearly the system performed better with two workers and the difference is noticeable in the graph.

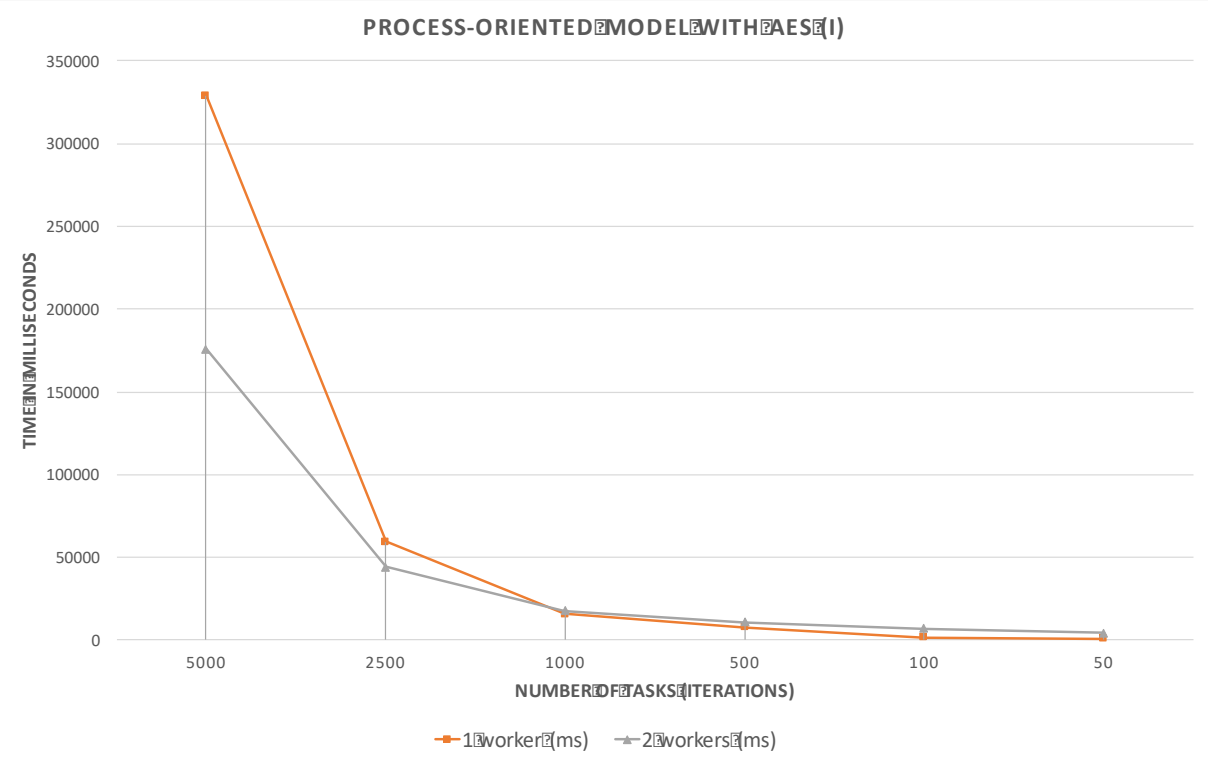

Figure 6-14.Process-Oriented model using AES (I)

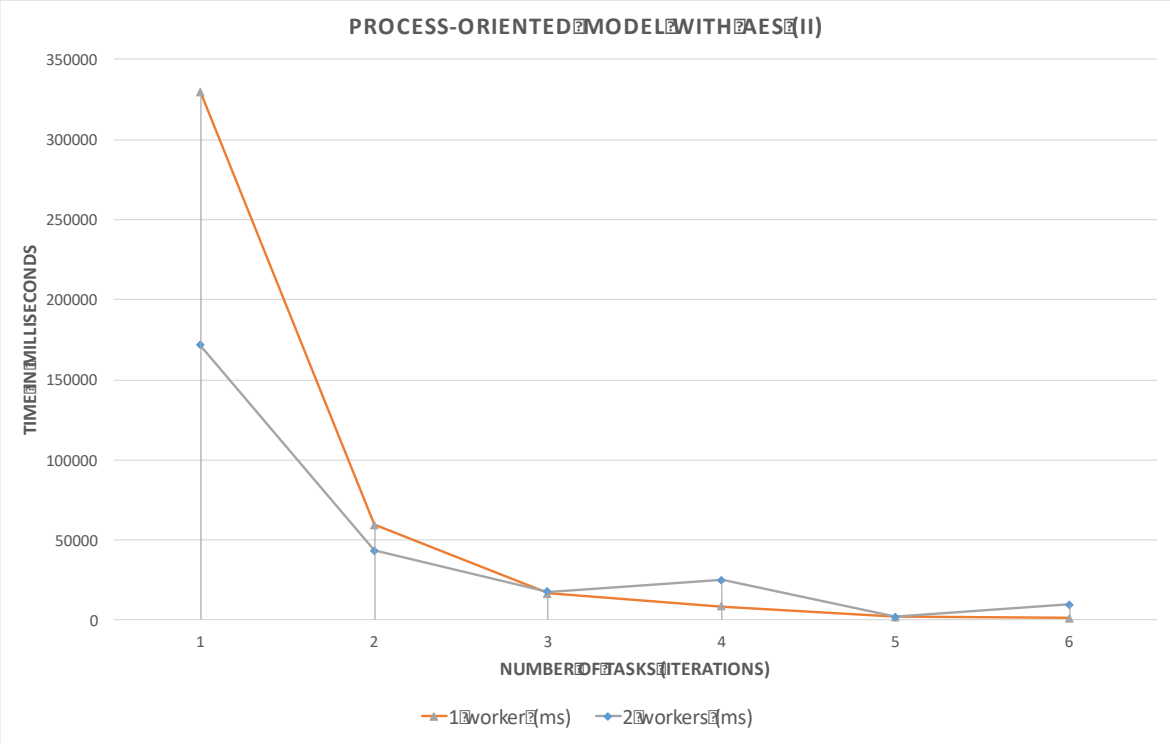

Figure 6-15.Process-Oriented model using AES (II)

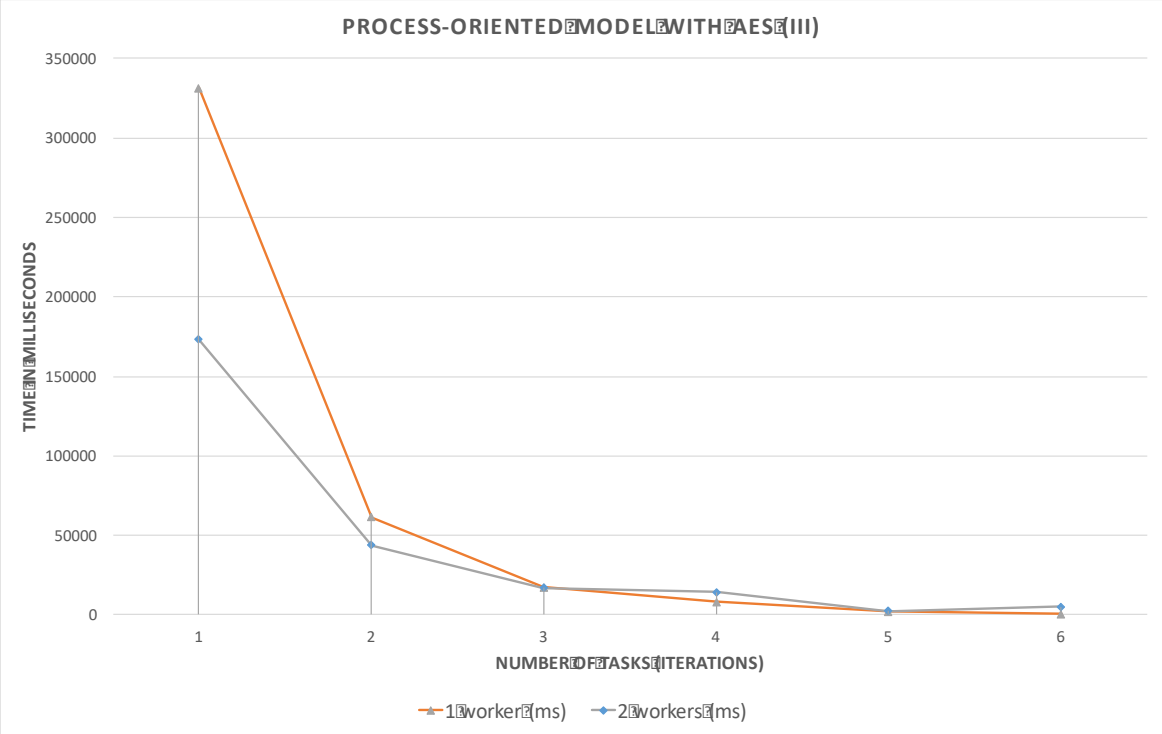

Figure 6-16.Process-Oriented model using AES (III)

Not using AES

When not using AES, the results were very different, and the trend changed. The figure 6-17 shows the time it took the system to process all the tasks the first time, with no cache. Figures 6-18 and 6-19 show the time it took the system to process all the tasks the second and third time, with an existing cache. It is appreciable that for each of the three runs, the times were not so similar. This is due to the cache. The first time the system runs, it has no cache, which means that the workers must receive the task, process it and return the result. When the proxy already has a cache of previous requests, the times are highly reduced. Moreover, the first time the system runs for each iteration, the time is reduced remarkably when two workers are used instead of one. However, the second and third time the system runs, the times stay very similar for one or two workers. This is due to the cache, as the proxy does not need to make any calls to the workers.

For the highest number of iterations (5000), the first time the system ran, it took 326,901 ms with one worker, whereas with two workers, the time was reduced to 168,534 ms. This is a reduction of around 50%. In some cases, the system with two workers took longer to complete, which was not expected. This is likely due to the caching process. The proxy could have incurred in an additional load for caching the responses of two workers, affecting the times negatively.

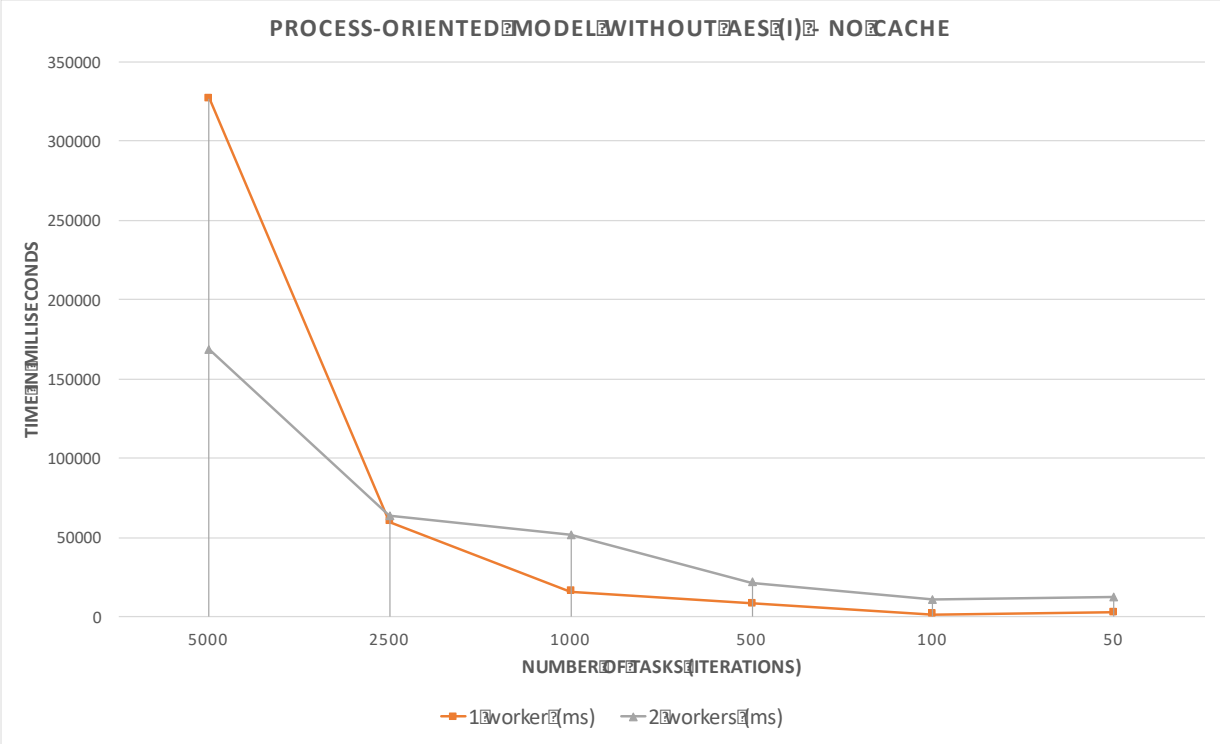

Figure 6-17.Process-Oriented model not using AES

For the second and third run, the cache was already generated, causing the times to reduce drastically as can be seen in the figures 6-18 and 6-19. For 5000 iterations, the time with cache was 27,377 ms on average, which is a reduction of 91% against the one worker system and 84% against the two workers' system. This remarkable reduction of time is what gives the model its value as it was the time it took the producer to generate the calls, the proxy to search the cache and respond to each request. When having a cache in place, the number of workers available is not so important as they are never called.

To summarize, the result of this model was good, as it performed as expected and the cache proved to be very effective in reducing the amount of calls. This resulted in better times for processing complex tasks. It needs to be noted that for these experiments the same task was run several times, which is an "ideal" scenario for the cache functionality. In IoT environments this can happen. Devices might need to run similar or exact tasks, which makes the model applicable. The model can be adjusted depending on its characteristics and the required tasks. For example, less repetitive tasks could have longer freshness times for responses.

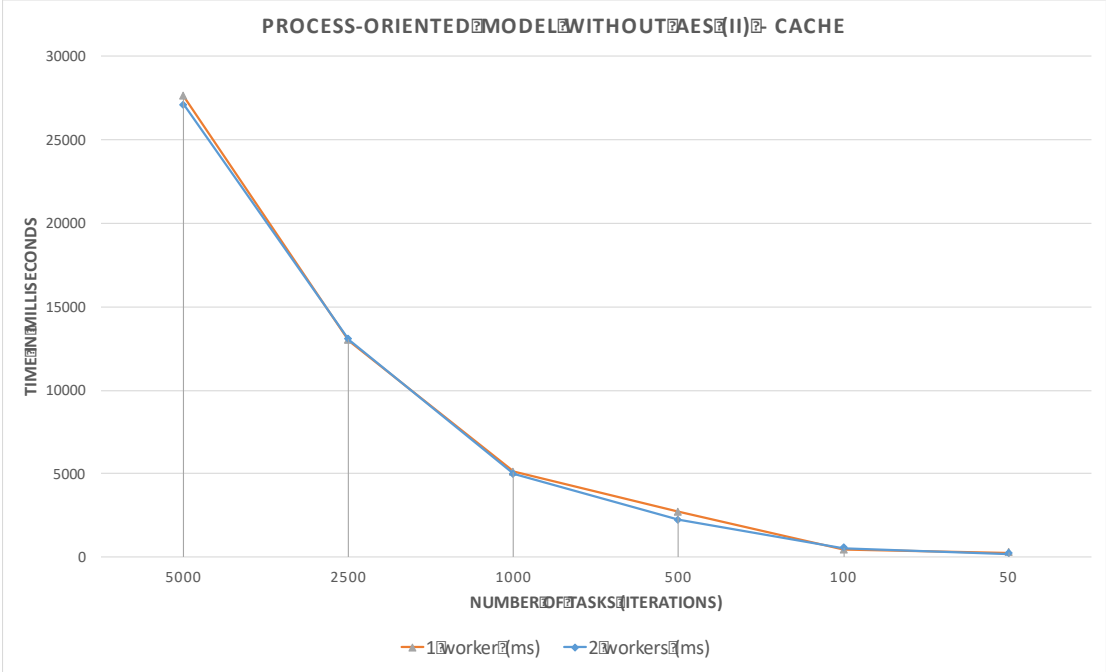

Figure 6-18.Process-Oriented model not using AES

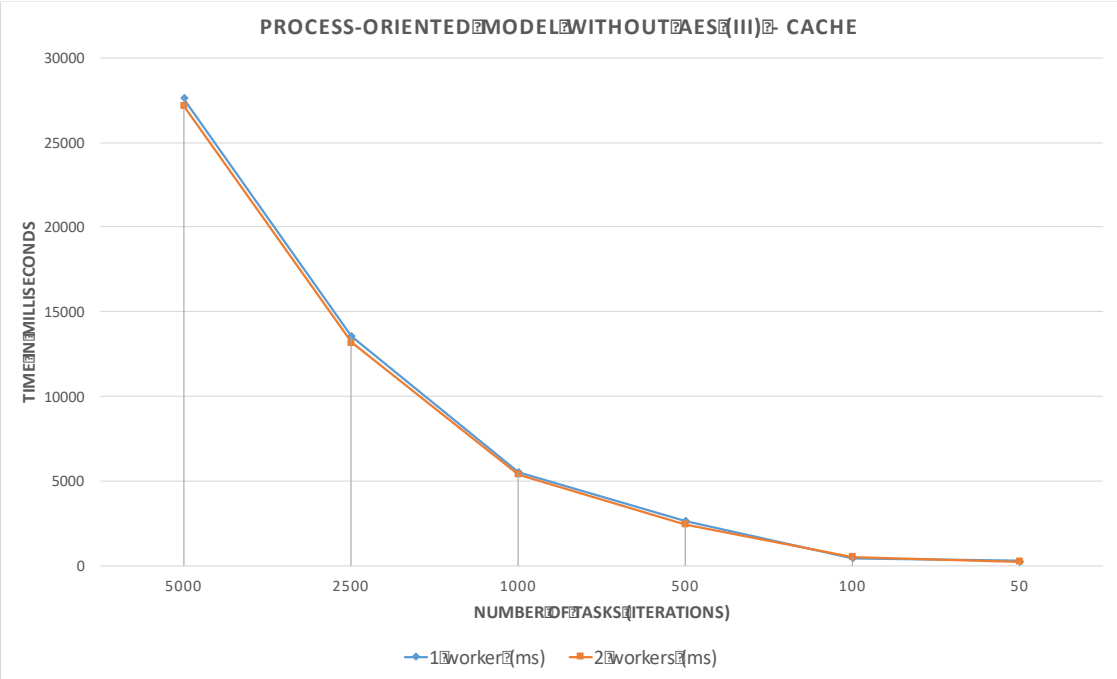

Figure 6-19.Process-Oriented model not using AES

6.3. Comparison

The figure 6-20 shows a summary of the time each of the models took to finish each task with one worker. For the data-oriented models and the process-oriented model with AES, the average of the three runs was taken. For the process oriented model without AES, the value of the second run (with cache) was taken. The models that took the most were the ones with the relational database and blockchain, as they were the only ones where another component was used. It is likely that the times were increased by the communication to this new component. However, the relational database shows a slower performance for higher number of iterations, whereas the blockchain performs very similarly for lower number of iterations. Storing the tuples in a dedicated server in RAM and the process-oriented model without cache performed very similar. However, it must be considered that for the process-oriented model, there was an additional load of encrypting and decrypting the values.

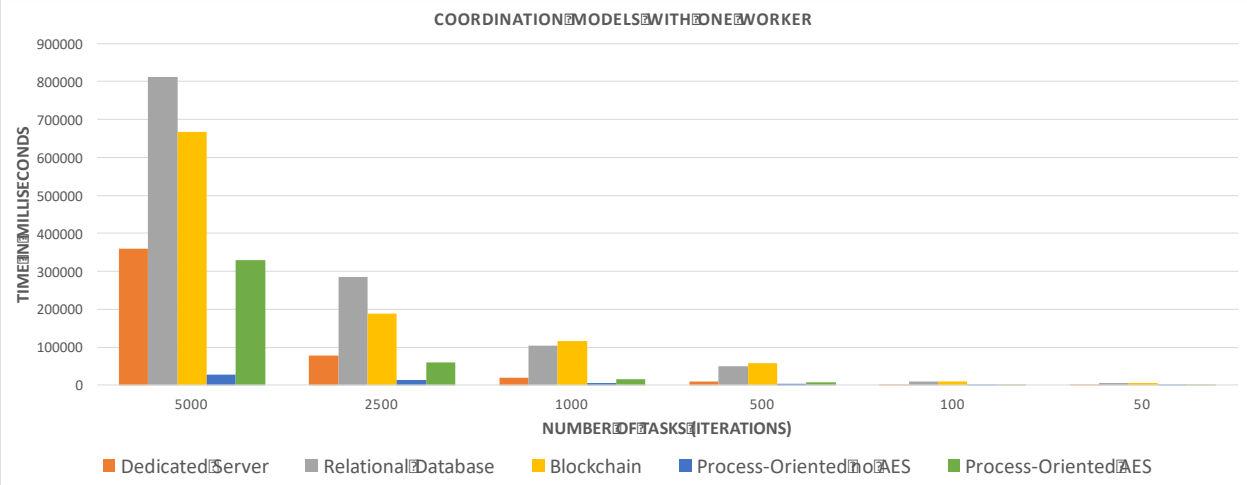

Figure 6-20.Coordination models performance summary with one worker

With two workers, the times for every model were reduced. This means that the resources of the two workers were being used efficiently and the performance of the tuple space and the proxy did not affect the overall performance of the models. Figure 6-21 shows the results. Again, the process-oriented model with cache had the lowest times as no extra requests

were required. Same as for one worker, both the blockchain and relational database models performed poorly and had the higher times of all the models.

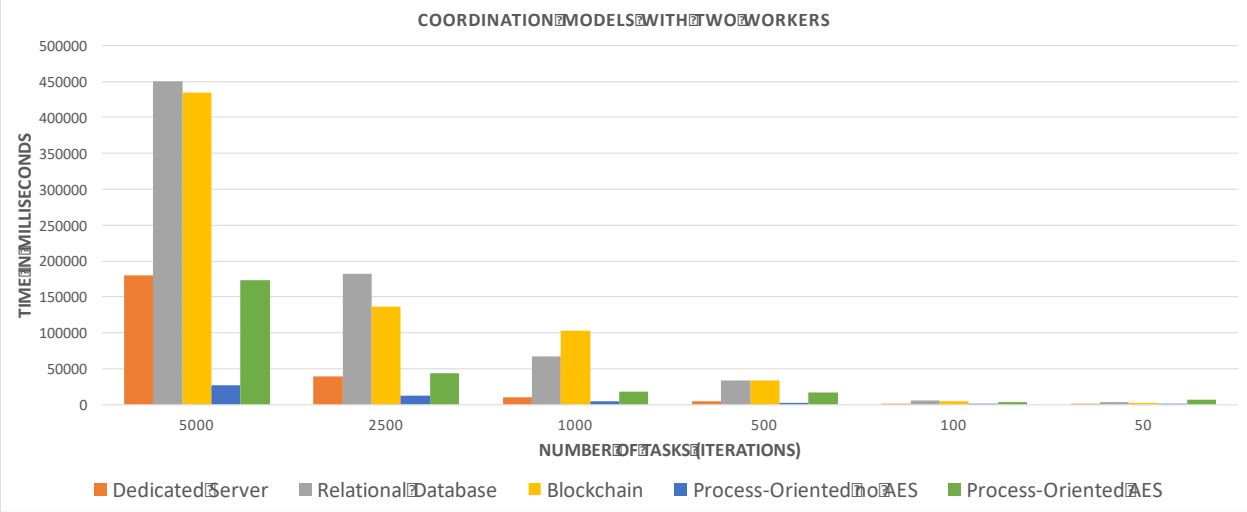

Figure 6-21.Coordination models performance summary with two workers

6.4. Summary

Four models were implemented and evaluated with a simple resource-consuming task. The objective of the experiments was to evaluate the performance of each of those models and compare them. For the data-oriented models, it was seen that the storage of the tuples affects how the system behaves directly. Keeping them in another device generates communication costs that might be unnecessary. Besides, keeping tuples in a way they are easy to search for seems very important when a large number are stored.

The process-oriented model had a very good overall performance. Even when the model did not employ the cache and the devices were incurring in an additional work load, it had low transaction times, that in some cases were lower than every data-oriented model. Perhaps its times would have been much better if the devices did not have to encrypt and decrypt the data. When the model employed cache and no encryption was used, the model had an outstanding performance. It had the lowest times by a big margin. Clearly, the usage of the cache reduced the communication costs substantially.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

7.1. Summary

Internet of Things is a growing model with many applications. It has become easier, cheaper and faster to deploy complex IoT systems. Those systems are comprised of heterogeneous constrained devices that are in charge of generating data and usually sending it to the cloud for processing. That processing can be very complex depending on the system and will generate communication costs. These constrained environments have different types of devices. The ones with fewer resources are forced to gather data only. The more robust could process the more complex tasks, allowing the system to share resources. To allow devices to cooperate with each other in an effective way, a coordination model must be put in place, allowing devices to distribute tasks among the system depending on their capabilities.

The present research evaluates different coordination models for IoT, including the most common data-oriented model, Linda. Three variations of this model were adjusted for IoT and implemented using current technologies and protocols for constrained environments. The core of the data-oriented coordination models is the data space. To evaluate the performance in different scenarios, in this research three ways of storing the tuple space were implemented: RAM, relational database and blockchain. In addition, a new process-oriented model was implemented and evaluated for constrained environments using the caching and proxying features of CoAP. This model defined the communication specification for devices to share resources and how multiple calls can be avoided.

The evaluations of the four coordination models showed that devices can share resources and improve processing times when more devices are available with usable computation power. The evaluations also showed that the way the data space is stored affects the performance of the entire system. Also, it was proven that the process-oriented model was significantly more effective when multiple repetitive tasks are required.

7.2. Contribution

The main contributions of the present research are the following:

- **Design, implementation and evaluation of three data-oriented coordination models for IoT:** The present work proposes three variations of data-oriented coordination models. The three were designed specifically for constrained environments ensuring that the coordination expenses were kept to a minimum. This research also showed that these kinds of models are applicable for IoT environments. Those models can be very effective when the tasks are worth candidates for coordinating and there are workers available.
- **Proposal of a fault-tolerant process-oriented coordination model for IoT:** This research designs a new coordination model based on process-oriented approaches. The model sees devices as black boxes with output and input ports that are connected over channels. The evaluation of this new model showed that it is feasible to have a high-performance and effective system in which devices can cooperate and where computation and communication directives are separated.

7.3. Future Work

Coordination models for IoT are expected to evolve in the following aspects:

- **Complete decentralized approaches:** IoT systems are comprised of a variety of devices. To create a complete fault-tolerant and dynamic environment, all those devices should be able to coordinate without the intervention of other components that might be present in the edge of the network. For the data-oriented models, a completely decentralized approach can be created using the blockchain. In the present work only one blockchain node was accessed. However, distributing the blockchain correctly and accessing all nodes can create a more dynamic and decentralized approach.

- **Coordination decisions based on states:** The coordination can be more effective if the tasks are distributed taking into account the characteristics of the devices and their current state. If a device is busy with a certain task, that task should be processed by another device. As IoT is a heterogeneous system, the tasks will not be distributed evenly. More powerful devices would process more and more complex tasks. Furthermore, the system could decide which device must process a task based on previous tasks and results.
- **Runtime evolution:** The coordination models proposed in this research rely on the availability of workers. However, those workers could fail and become unable to process tasks. The system could notice that unavailability and reconfigure itself. For example, when the tuples are sent to the tuple space and no worker is available, the producer could process them and not send more tuples until workers become available. Also, more complex systems can be created in which devices can be workers or managers depending on the activity they are performing.
- **Other communication protocols:** The current work is based on the features CoAP offers. However, other communication protocols like MQTT can be reviewed and evaluated to validate their performance.

REFERENCES

[1] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." *Computer networks* 54, no. 15 (2010): 2787-2805.

[2] “Internet Of Things - Explore - Google Trends.” [Online]. Available: https://www.google.com/trends/explore?date=all&q=Internet Of Things. [Accessed: 12-Jan-2017].

[3] Bröring, Arne, Soumya Kanti Datta, and Christian Bonnet. "A Categorization of Discovery Technologies for the Internet of Things." In *Proceedings of the 6th International Conference on the Internet of Things*, pp. 131-139. ACM, 2016.

[4] “Research for the digital age.” [Online]. Available: intelligence.businessinsider.com. [Accessed: 30-Jun-2017].

[5] Arbab, Farhad. "The IWIM model for coordination of concurrent activities." *Coordination languages and models* (1996): 34-56.

[6] Murphy, Amy L., Gian Pietro Picco, and G-C. Roman. "Lime: A middleware for physical and logical mobility." In *Distributed Computing Systems, 2001. 21st International Conference on.*, pp. 524-533. IEEE, 2001.

[7] Gubbi, Jayavardhana, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions." *Future generation computer systems* 29, no. 7 (2013): 1645-1660.

[8] Lopez Research, “An Introduction to the Internet of Things (IoT),” *Lopez Res. Llc*, vol. Part 1. of, no. November, pp. 1–6, 2013.

[9] Cao, Yang, Tao Jiang, and Zhu Han. "A Survey of Emerging M2M Systems: Context, Task, and Objective." *IEEE Internet of Things Journal* 3, no. 6 (2016): 1246-1258.

[10] “The Intel® Edison Module | IoT | Intel® Software.” [Online]. Available: https://software.intel.com/en-us/iot/hardware/edison. [Accessed: 13-Jan-2017].

[11] Evans, Dave. "The internet of things: How the next evolution of the internet is changing everything." CISCO white paper 1, no. 2011 (2011): 1-11.

[12] Cisco Systems, “Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are,” 2015.

[13] Wu, Geng, Shilpa Talwar, Kerstin Johnsson, Nageen Himayat, and Kevin D. Johnson. "M2M: From mobile to embedded internet." *IEEE Communications Magazine* 49, no. 4 (2011).

[14] Fielding, Roy T., and Richard N. Taylor. *Architectural styles and the design of network-based software architectures*. Doctoral dissertation: University of California, Irvine, 2000.

[15] Fielding, Roy T., and Richard N. Taylor. "Principled design of the modern Web

architecture." *ACM Transactions on Internet Technology (TOIT)* 2, no. 2 (2002): 115-150.

[16] Fielding, Roy, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol--HTTP/1.1. No. RFC 2616. 1999.

[17] Paganelli, Federica, Stefano Turchi, and Dino Giuli. "A web of things framework for restful applications and its experimentation in a smart city." *IEEE Systems Journal* 10, no. 4 (2016): 1412-1423.

[18] Guinard, Dominique, Vlad Trifa, and Erik Wilde. "A resource oriented architecture for the web of things." In *Internet of Things (IOT), 2010*, pp. 1-8. IEEE, 2010.

[19] Rahman, Reem Abdul, and Babar Shah. "Security analysis of IoT protocols: A focus in CoAP." In *Big Data and Smart City (ICBDSC), 2016 3rd MEC International Conference on*, pp. 1-7. IEEE, 2016.

[20] Elmangoush, Asma, Ronald Steinke, Thomas Magedanz, Andreea Ancuta Corici, Alex Bourreau, and Adel Al-Hezmi. "Application-derived communication protocol selection in M2M platforms for smart cities." In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pp. 76-82. IEEE, 2015.

[21] Bormann, Carsten, Angelo P. Castellani, and Zach Shelby. "Coap: An application protocol for billions of tiny internet nodes." *IEEE Internet Computing* 16, no. 2 (2012): 62-67.

[22] Shelby, Zach, Klaus Hartke, and Carsten Bormann. "The constrained application protocol (CoAP)." (2014).

[23] Andrew Banks and Rahul Gupta, “MQTT Version 3.1.1,” 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.doc. [Accessed: 02-Jun-2017].

[24] Kovatsch, Matthias, Martin Lanter, and Zach Shelby. "Californium: Scalable cloud services for the internet of things with coap." In *Internet of Things (IOT), 2014 International Conference on the*, pp. 1-6. IEEE, 2014.

[25] Teklemariam, Girum Ketema, Jeroen Hoebeke, Ingrid Moerman, and Piet Demeester. "Facilitating the creation of IoT applications through conditional observations in CoAP." *EURASIP Journal on Wireless Communications and Networking* 2013, no. 1 (2013): 177.

[26] “RFC 7252 - The Constrained Application Protocol (CoAP).” [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 09-Jan-2017].

[27] Hartke, Klaus. "Observing resources in the constrained application protocol (CoAP)." (2015).

[28] Tanganelli, Giacomo, Carlo Vallati, Enzo Mingozzi, and Matthias Kovatsch. "Efficient proxying of CoAP observe with quality of service support." In *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*, pp. 401-406. IEEE, 2016.

[29] Ludovici, Alessandro, Pol Moreno, and Anna Calveras. "TinyCoAP: A novel constrained application protocol (CoAP) implementation for embedding RESTful web services in wireless sensor networks based on TinyOS." *Journal of Sensor and Actuator Networks* 2, no. 2 (2013): 288-315.

[30] Viroli, Mirko, and Alessandro Ricci. "Tuple-based coordination models in event-based scenarios." In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pp. 595-601. IEEE, 2002.

[31] Papadopoulos, George A., and Farhad Arbab. "Coordination models and languages." *Advances in computers* 46 (1998): 329-400.

[32] Gelernter, David, and Nicholas Carriero. "Coordination languages and their significance." *Communications of the ACM* 35, no. 2 (1992): 96.

[33] Ciancarini, Paolo. "Coordination models and languages as software integrators." *ACM Computing Surveys (CSUR)* 28, no. 2 (1996): 300-302.

[34] Shaw, Mary, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, and Gregory Zelesnik. "Abstractions for software architecture and tools to support them." *IEEE transactions on software engineering* 21, no. 4 (1995): 314-335.

[35] Wegner, Peter. "Coordination as constrained interaction." *Coordination Languages and Models* (1996): 28-33.

[36] Roman, G-C., and H. Conrad Cunningham. "Mixed programming metaphors in a shared dataspace model of concurrency." *IEEE Transactions on Software Engineering* 16, no. 12 (1990): 1361-1373.

[37] Ahuja, Sudhir, N. Curriero, and David Gelernter. "Linda and friends." *Computer;(United States)* 19, no. 8 (1986)..

[38] Carriero, Nicholas, and David Gelernter. "Linda in context." *Communications of the ACM* 32, no. 4 (1989): 444-458.

[39] Ciancarini, Paolo. "Distributed programming with logic tuple spaces." *New Generation Computing* 12, no. 3 (1994): 251-283.

[40] Ciancarini, Paolo, Oscar Nierstrasz, and Akinori Yonezawa. *Object-Based Models and Languages for Concurrent Systems: ECOOP'94 Workshop on Models and Languages for Coordination of Parallelism and Distribution, Bologna, Italy, July 5, 1994. Selected Papers*. Vol. 924. Springer Science & Business Media, 1995.

[41] Rowstron, Antony IT, and Alan M. Wood. "Bonita: A set of tuple space primitives for distributed coordination." In *System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on*, vol. 1, pp. 379-388. IEEE, 1997.

[42] Minsky, Naftaly H., and Jerrold Leichter. "Law-governed Linda as a coordination model." In *European Conference on Object-Oriented Programming*, pp. 125-146. Springer, Berlin, Heidelberg, 1994.

[43] Tolksdorf, Robert. "Laura: A coordination language for open distributed systems." In *Distributed Computing Systems, 1993., Proceedings the 13th International Conference on*, pp. 39-46. IEEE, 1993.

[44] Kielmann, Thilo. "Designing a coordination model for open systems." In *International Conference on Coordination Languages and Models*, pp. 267-284. Springer, Berlin, Heidelberg, 1996.

[45] Picco, Gian Pietro, Amy L. Murphy, and G-C. Roman. "LIME: Linda meets mobility." In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pp. 368-377. IEEE, 1999.

[46] Papadopoulos, George A., and Farhad Arbab. "Control-driven coordination programming in shared dataspace." In *International Conference on Parallel Computing Technologies*, pp. 247-261. Springer, Berlin, Heidelberg, 1997.

[47] Sommerville, Ian, and Graham Dean. "PCL: a language for modelling evolving system architectures." *Software Engineering Journal* 11, no. 2 (1996): 111-121.

[48] Kramer, Jeff, Jeff Magee, and Anthony Finkelstein. "A constructive approach to the design of distributed systems." In *Building Distributed Systems, IEE Colloquium on*, pp. 3-1. IET, 1990.

[49] Barbacci, Mario R., Charles B. Weinstock, Dennis L. Doubleday, Michael J. Gardner, and Randall W. Lichota. "Durra: a structure description language for developing distributed applications." *Software Engineering Journal* 8, no. 2 (1993): 83-94.

[50] Barbacci, Mario R., and Jeannette M. Wing. "A language for distributed applications." In *Computer Languages, 1990., International Conference on*, pp. 59-68. IEEE, 1990.

[51] Arbab, Farhad, Ivan Herman, and Pål Spilling. "An overview of Manifold and its implementation." *Concurrency and Computation: Practice and Experience*5, no. 1 (1993): 23-70.

[52] Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things." In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13-16. ACM, 2012.

[53] Aazam, Mohammad, and Eui-Nam Huh. "Fog computing and smart gateway based communication for cloud of things." In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pp. 464-470. IEEE, 2014.

[54] Madsen, Henrik, Bernard Burtschy, G. Albeanu, and F. L. Popentiu-Vladicescu. "Reliability in the utility computing era: Towards reliable fog computing." In *Systems, Signals and Image Processing (IWSSIP), 2013 20th International Conference on*, pp. 43-46. IEEE, 2013.

[55] Zhu, Jiang, Douglas S. Chan, Mythili Suryanarayana Prabhu, Preethi Natarajan, Hao Hu, and Flavio Bonomi. "Improving web sites performance using edge servers in fog computing architecture." In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th*

International Symposium on, pp. 320-323. IEEE, 2013.

[56] Banerjee, Arijit, Xu Chen, Jeffrey Erman, Vijay Gopalakrishnan, Seungjoon Lee, and Jacobus Van Der Merwe. "MOCA: a lightweight mobile cloud offloading architecture." In *Proceedings of the eighth ACM international workshop on Mobility in the evolving internet architecture*, pp. 11-16. ACM, 2013.

[57] Chun, Byung-Gon, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. "Clonecloud: elastic execution between mobile device and cloud." In *Proceedings of the sixth conference on Computer systems*, pp. 301-314. ACM, 2011.

[58] Kosta, Sokol, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading." In *Infocom, 2012 Proceedings IEEE*, pp. 945-953. IEEE, 2012.

[59] Orsini, Gabriel, Dirk Bade, and Winfried Lamersdorf. "Computing at the mobile edge: designing elastic android applications for computation offloading." In *IFIP Wireless and Mobile Networking Conference (WMNC), 2015 8th*, pp. 112-119. IEEE, 2015..

[60] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." (2008): 28.

[61] Greenspan, G. "MultiChain Private Blockchain—White Paper." (2015).

[62] “Global Bitcoin Nodes Distribution - Bitnodes.” [Online]. Available: https://bitnodes.21.co/. [Accessed: 14-Jun-2017].

[63] Conoscenti, Marco, Antonio Vetrò, and Juan Carlos De Martin. "Blockchain for the Internet of Things: a Systematic Literature Review." (2016): 1-6.

[64] Xu, Xiwei, Cesare Pautasso, Liming Zhu, Vincent Gramoli, Alexander Ponomarev, An Binh Tran, and Shiping Chen. "The blockchain as a software connector." In *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*, pp. 182-191. IEEE, 2016.

[65] “Arduino - IntelEdison.” [Online]. Available: https://www.arduino.cc/en/ArduinoCertified/IntelEdison#toc3. [Accessed: 16-Jun-2017].

[66] “MySQL.” [Online]. Available: https://www.mysql.com/. [Accessed: 20-Jun-2017].

[67] “Blockchains vs centralized databases | MultiChain.” [Online]. Available: http://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases/. [Accessed: 20-Jun-2017].

[68] “Blockchain technology: 9 benefits & 7 challenges | Deloitte.” [Online]. Available: https://www2.deloitte.com/nl/nl/pages/innovatie/artikelen/blockchain-technology-9-benefits-and-7-challenges.html. [Accessed: 20-Jun-2017].

[69] “The Go Programming Language.” [Online]. Available: https://golang.org/. [Accessed: 21-Jun-2017].

[70] “Github - dustin/go-coap: Implementation of CoAP in Go.” [Online]. Available:

https://github.com/dustin/go-coap. [Accessed: 21-Jun-2017].

[71] “Concurrency — An Introduction to Programming in Go | Go Resources.” [Online]. Available: https://www.golang-book.com/books/intro/10. [Accessed: 26-Jun-2017].

[72] “Apache JMeter - Apache JMeter™.” [Online]. Available: https://jmeter.apache.org/. [Accessed: 27-Jul-2017].

[73] “AES encryption.” [Online]. Available: http://aesencryption.net/. [Accessed: 29-Jun-2017].