UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

OTÁVIO MORAES DE CARVALHO

A model for distributed data aggregation in edge and cloud environments

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Philippe O. A. Navaux

Porto Alegre
January 2019

CIP — CATALOGING-IN-PUBLICATION

Carvalho, Otávio Moraes de

A model for distributed data aggregation in edge and cloud environments / Otávio Moraes de Carvalho. – Porto Alegre: PPGC da UFRGS, 2019.

83 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Advisor: Philippe O. A. Navaux.

1. Edge Computing. 2. Fog Computing. 3. Cloud Computing. 4. Internet of Things. 5. Distributed Computing. I. Navaux, Philippe O. A.. II. Título.

“The one who has conquered himself
is a far greater hero
than he who has defeated
a thousand times a thousand men.”
— Siddhārtha Gautama

ACKNOWLEDGMENTS

First of all, I would like to give a special thanks to my parents, Jorge and Nadia, for all the love and support. You are my life lessons on the importance of kindness and wholeheartedness. I would also like to wholeheartedly thank my uncle Marcelo and my aunt Ana — without whom I would never be here — not only for their extensive support throughout my life but also for sharing their family values with me and for making me part of their family.

Second, I would like to thank my advisor and co-advisor, Prof. Dr. Philippe Navaux and Prof. MSc. Eduardo Roloff, respectively, for all of the support, openness to new ideas and confidence into my work. Thanks also to everyone in the Informatics Institute at the Federal University of Rio Grande do Sul (UFRGS), by the opportunities and high-quality education provided to me and my colleagues.

Furthermore, I would also like to thank all of my colleagues at Parallel and Distributed Research Group (GPPD) and ThoughtWorks. Without them, my education certainly would not have been as complete and, my path would not have been as enjoyable.

Finally, I would like to thank all of my friends (many of them also colleagues and co-workers). You all live in my heart and are a considerable part of what has made possible this accomplishment.

This study was financed in part FAPERGS in the context of the GreenCloud Project. This research also received partial funding from CYTED under the context of the RICAP Project. It also received partial funding from the EU H2020 Programme and from MCTI/RNP Brazil under the HPC4E project. Microsoft has provided cloud computing instances on Microsoft Azure to the execution of the experiments.

ABSTRACT

The Internet of Things generates each time larger amounts of data through devices that range from personal smartphones sensors to large scale smart cities sensor deployments. Edge Computing and Fog Computing hardware resources nowadays have processing power that in the past was only found on Cloud Computing resources. These capabilities enable the possibility of aggregating data directly on the edge of the network, avoiding network traffic that was previously necessary to transfer data to the cloud.

However, the ability to leverage each time more powerful edge nodes to decentralize data processing and aggregation is still a significant challenge for both industry and academia. Although large scale sensor deployments and powerful edge nodes are now available, the proper frameworks and processing patterns for these profiles of data are still under active research.

In this work, it is analyzed the impact of a model for data aggregation in a large scale smart grid application dataset. The results obtained show that the implemented testbed application, through the usage of edge node aggregation and messaging windows, was able to achieve data aggregation rates of above 400 million measurements per second. These results were obtained using machines on 15 distinct geographic regions on the Microsoft Azure platform, for a total of 1366 machines in the largest evaluation scenario.

Keywords: Edge Computing. Fog Computing. Cloud Computing. Internet of Things. Distributed Computing.

Um modelo para agregação de dados distribuída em ambientes de edge e cloud

RESUMO

A internet das coisas é responsável pela geração de quantidades de dados cada vez maiores, oriundas de dispositivos que vão de sensores de smartphones até implementações de sensores em larga escala para cidades inteligentes.

Os recursos de hardware dos dispositivos de edge computing e fog computing apresentam poder de processamento que no passado só era encontrado em recursos de computação em nuvem. O aumento da capacidade de processamento habilita a possibilidade de agregar dados na borda da rede, evitando tráfego de rede que anteriormente era necessário para transmitir os dados para a nuvem.

Por outro lado, a capacidade de alavancar nodos de borda cada vez mais poderosos, para descentralizar o processamento e agregação de dados, ainda é um desafio significante tanto para a indústria quanto para a academia. Ainda que nodos de borda poderosos e grandes instalações de sensores estejam disponíveis hoje em dia, as frameworks e padrões de processamento necessários para esses perfis de dados ainda são material de ativa pesquisa.

Nesse trabalho, é analisado o impacto de um modelo para agregação de dados em um conjunto de dados para redes de energia inteligentes de larga escala. Os resultados obtidos mostram que, para a aplicação de testes desenvolvida, através da utilização de agragação na borda da rede e janelas de mensagens, foi possível atingir taxas de agregação acima de 400 milhões de medidas por segundo. Estes resultados foram obtidos utilizando máquinas em 15 regiões geográficas distintas na plataforma Microsoft Azure, totalizando 1366 máquinas no maior cenário de avaliação.

Palavras-chave: Internet das Coisas, Computação em névoa, Computação na Nuvem, Computação Distribuída.

LIST OF ABBREVIATIONS AND ACRONYMS

AP	Access Point
AMI	Advanced Metering Infrastructure
API	Advanced Programming Interface
AMQP	Advanced Message Queuing Protocol
ARM	Advanced RISC Machine
ARIMA	Autoregressive Integrated Moving Average
AWS	Amazon Web Services
CDN	Content Delivery Network
CEP	Complex Event Processing
CLI	Command Line Interface
DEBS	Distributed Event-Based Systems
DSMS	Data Stream Management Systems
DBMS	Database Management Systems
DSM	Demand Side Management
ETSI	European Telecommunications Standards Institute
GC	Garbage Collection
GFS	Google File System
GRPC	gRPC Remote Procedure Calls
HFC	Hybrid Fiber-Coaxial
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
IaaS	Infrastructure as a Service
ISG	Industry Specification Group
LTLF	Long Term Load Forecast

LTE	Long-Term Evolution
MEC	Multi-access Edge Computing
MTLF	Medium Term Load Forecast
MLP	Multilayer Perceptron
NFV	Network Function Virtualization
NIST	National Institute of Standards and Technology
OEC	Open Edge Computing
OS	Operating System
PaaS	Platform as a Service
QoS	Quality of Service
QPS	Queries Per Second
RFID	Radio-Frequency Identification
SDN	Software-Defined Networking
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
STLF	Short Term Load Forecast
SSM	State-Space Model
TCP	Transmission Control Protocol
VANET	Vehicular Ad-hoc Network
VM	Virtual Machine
XaaS	Everything as a Service

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The ubiquity of the Internet of Things (IoT) unleashes the potential of using innovations based on sensor data to improve society's overall quality of life. These profiles of data can be used to enable technologies such as large scale smart cities deployments, smart home monitoring, smart industries and smart energy grids. Devices that provide these kinds of capabilities are now widespread through cities and homes on devices such as smartphones, medical devices, home appliances and street signals. By 2025, researchers estimate that the IoT will have a potential economic impact of $11 trillion per year – which would be equivalent to about 11% of the world economy. They also expect that one trillion IoT devices will be deployed by 2025 (BUYYA; DASTJERDI, 2016).

IoT is now a reality, and we are connecting each time more devices – such as personal consumer electronic devices, home appliances, cameras, medical devices, and all types of sensors – to the Internet environment. This ubiquity unlocks the potential to innovations that can use the data generated by those devices to enable smart cities, smart infrastructures and smart services that can improve quality of life.

By 2025, researchers estimate that the IoT will have a potential economic impact of 11 trillion per year – which would be equivalent to about 11% of the world economy. They also expect that one trillion IoT devices will be deployed by 2025. In majority of the IoT domains such as infrastructure management and healthcare, the major role of IoT is the delivery of highly complex knowledge-based and action-oriented applications in real-time (BUYYA; DASTJERDI, 2016).

Novel technologies for mobile computing and IoT are shifting the focus of its research and development to computing toward dispersion. In this context, edge computing is one of the most prominent areas, it is a new paradigm in which a high volume of computing and storage resources – generally referred to as cloudlets, micro datacenters or fog nodes – are placed at the Internet's edge in close proximity to mobile devices or sensors (SATYANARAYANAN, 2017).

The ability to scale these solutions to a large number of nodes is an important research topic, due to the fact that millions of users will be part of these information flows. Cloud computing services are now widespread and have proven their value by providing reliable processing at global scale. However, there is a large set of applications that cannot tolerate the latency penalties of sending data to be aggregated on the cloud and then sending it back to edge nodes. Also, the act of sending a potentially large number of

small packets to the cloud for data processing can saturate the network and decrease the scalability of the applications (DASTJERDI; BUYYA, 2016).

Since millions of end-users will be taking part into processes and information flows of smart grids, high scalability of these methods turns into an important issue. To solve these issues, cloud computing services present themselves as a viable solution, by providing reliable, distributed and redundant capabilities at global scale (BUYYA et al., 2009). However, there is a large set of applications which cannot accept the delay caused by transferring data to the cloud and back, being bounded by latency. However, it is not efficient to send a large number of small packages of data to the cloud for processing, as it would saturate network bandwidth and decrease scalability of the applications (DASTJERDI; BUYYA, 2016). Therefore, one of the most important points explored by this work is geographical distribution of processing elements, and how it impacts the processing ability in global scale scenarios. The largest evaluation is executed in 1366 nodes, distributed globally across 15 distinct geographical regions. This way, it was possible to evaluate the real impact of latencies and how data aggregation at nearby regions can help to improve processing. Which applies not exclusively to smart grids, but also to many other scenarios where offloading all data to the cloud is not a viable option.

Furthermore, finding ways to achieve a more sustainable lifestyle plays a large hole on the path of our society growth, and poses some challenges to a society that depends each time more on a increasing number of electrical devices. to achieve this, the smart grid incentives pretend to improve the legacy systems for energy production and consumption, based on research, to advance technologies in the energy field.

Smart grids will allow consumers to receive near real-time feedback about household energy consumption and price, allowing consumers to make informed decisions about their spending. For energy producers, it will be possible to leverage home consumption data to produce energy forecasts, enabling near real-time actions and better scheduling of energy generation and distribution (BROWN, 2008). This way, smart grids will save billions of dollars in the long run, for consumers and the generators, as well as to reduce the impact on the environment, according to recent forecasts (REUTERS, 2011). In this work, it is utilized a smart grid short-term load forecast algorithm to evaluate a testbed implementation of the model.

In order to provide a new model for aggregate data in fog computing and edge computing environments, this thesis provides a model that combines existing cloud computing resources to edge devices, extending the research on cloud computing applied to

IoT data profiles (CARVALHO; ROLOFF; NAVAUX, 2017). Using this model, it is possible to provide a high performance aggregation model that leverages the existing research in distributed stream processing systems and applies it to the context of fog computing. The results obtained show that the implemented testbed application, through the usage of edge node aggregation and messaging windows, was able to achieve data aggregation rates of above 400 million measurements per second.

The rest of this thesis is organized as follows. Chapter 2 introduces the state of the art in distributed processing, explains mandatory concepts to understand this work and shows the most recent and relevant research publications on the field. Chapter 3 describes the first generation of the model, its main architectural concepts and the algorithm used to evaluate the platform. Chapter 4 evaluates the performance of the first generation of the model and discusses the results obtained. Chapter 5 describes the second generation of the aggregation model, which introduces a new processing layer to better explore locality and decrease the processing latency. Chapter 6 evaluates the performance of the second generation of the model in a global scale deployment. Chapter 7 analyses what was possible to achieve with this model in terms of design and performance and discusses the future works.

2 STATE OF THE ART

In this chapter, we introduce several background concepts necessary to the proper understanding of this work. Distributed systems are the basis of this work, since it is known from the state of the art that centralized applications have certain limits in terms of amount of users and Quality Of Service (QoS) that they are able to provide.

In Section 2.1, there is a brief overview of the evolution of the distributed computing platforms, starting from distributed computing research on the late 60s, up to the recent research advances in Fog Computing and Edge Computing.

Smart grids are an important research topic, since the traditional design of energy grids has not being improved during the last century, and are not able to consider the metrics collected by households to optimize energy generation and distribution. In this work, it is utilized a smart grid short-term load forecast algorithm to evaluate a testbed implementation of the model. In Section 2.2, we introduce some important concepts regarding smart grids and how they are different from traditional approaches to design energy grids.

Finally, on Section 2.3, we review the related works that represent the state of the art on applications and models for edge processing. We also explain what are the main challenges of the research field and how this work relates to those publications.

2.1 Evolution of distributed computing

Distributed computing refers to the study of decentralized models of systems and its ways to divide computation between multiple network devices. The beginning of this research field trace back to the 60s, when the first studies on concurrent processes communicated via message passing were developed (LEOPOLD, 2001). After that, many other computation paradigms were created. Starting from the concept of utility computing over a distributed framework, the computing domain has gradually moved towards the concept of Cloud Computing and, more recently, efforts on improving computation at the edge of the network where expressed via new concepts like internet of things, fog computing and edge computing.

In this Section, we introduce the background required to understand the foundation of this work. The background review starts with established research areas like utility computing and cluster computing. In the following sections, grid computing and cloud

computing are introduced, explaining the evolution of these ideas to broader network areas and the main challenges in this area regarding performance and network latencies.

Finally, more recent research topics are introduced, in areas like IoT, fog computing and edge computing. Their challenges on how to aggregate data in a timely manner with low power computation and high latencies are also briefly discussed. It is also introduced some important concepts on smart grids research, since it is the basis of the case study used to evaluate the proposed platform.

2.1.1 Utility computing

A major requirement from the end-users is that they need to get computing and storage services within a short period of time. This strict requirement of deadline driven services has created the demand for getting services without the botheration about deployment and operationalization of custom hardware. The traditional mainframe based systems lack these features of providing deadline driven services, as the users need to purchase hardware, customize them, and install tools and software to make them operational. That is why, the end users have shifted towards getting real-time services from the vendors. These services are provided to the end users as an utility. The users need not worry about the underlying hardware infrastructure. The services are provided to the end users whenever they require that.

The provision of providing computations and services to the end users based on their need has created the distributed computation model named as utility computing. In this early computing model over distributed systems, users have to pay for the services whenever they use it. The concept was first presented by John McCarthy in 1961 (GARFINKEL, 1999).

Though the utility computing was not very popular in those times, it was again introduced in late 90s as the cost for computation hardware gradually dropped and miniaturization of servers become practical. The excessive demands for services have generated the need for this utility based service provisioning by the service providers. Previously, there was no proper access to resources in several systems. Further, there was no provision of supporting a fixed and predefined deadline for response time over these systems. However, the utility computing has given the user a proper valuation of their services. Utility computing systems can be considered as a marketplace where the users compete for getting their service by the service providers. The advantages of such utility comput-

ing systems are much more in comparison with the single time-sharing system. Utility computing supports the users by giving higher throughput than a single time-sharing system as multiple servers are placed in utility computing (PADALA et al., 2007).

2.1.2 Cluster computing

In cluster computing (BUYYA et al., 1999), many connected computers work together in order to behave like a single system. These computers do the same work, which is controlled by a scheduling software. In the advent of low cost microprocessors, the cluster computers have emerged as a new computing platform.

The different personal computers along with communication software and network interfaces are connected to the high speed network or switch. The cluster middleware is the software environment that interconnects the cluster computing nodes with different applications.

The advantages of the cluster computing system are the reliability and the availability. The end users can be given more computing power and storage facilities by allowing several cluster computers to work. The system failure rate is decreased in case of cluster computing as we have redundancy within the clusters. The dedicated and high speed network connects these cluster nodes in order to provide more reliability in case of system failure. These are the driving forces behind the development of cluster computing concepts. There are several types of applications of cluster computing, such as load balancing (WERSTEIN; SITU; HUANG, 2006), high availability clusters (AGBARIA; FRIEDMAN, 1999), etc. In load balancing, a single task can be divided between several cluster nodes in order to provide the particular service. The high availability clusters provide the users with the required service in case of system failure. The cluster computing systems have data redundancy among the cluster nodes. That is why, cluster computing systems have greater reliability than utility computing systems.

2.1.3 Grid computing

Grid computing (BERMAN et al., 2003) (CHERVENAK et al., 2000) was originated in early 90s as an effect of extending the distributed computation models beyond the cluster computing framework. Cluster computing environment faced several limitations

during its implementation. One of the major problems faced over the cluster architecture is that, there can be node failures in clusters; however as the cluster size increases, the complexity of finding the location of failures also increases. However, the grid computing systems are much modular and have very less number of points of failure. The grid software is responsible for policy management.

A generic architecture of a grid computing framework consists of several grid clients that are connected by an underlying computer network to the grid server, and the grid servers are in turn connected to the end users.

In grid computing, several computers, termed as computing grids, work together to provide a high performance distributed environment. The computation is divided and distributed among several nodes. The computing nodes are loosely coupled in the sense that they make use of little or no knowledge of the definitions of other separate nodes. On the contrary, in tightly coupled systems, the computing nodes are not only linked together but also dependent upon each other. The disadvantages of tightly coupled system is that the entire system becomes down in case of even a single node failure. However, the resources are generally heterogeneous and largely distributed (KRAUTER; BUYYA; MAHESWARAN, 2002).

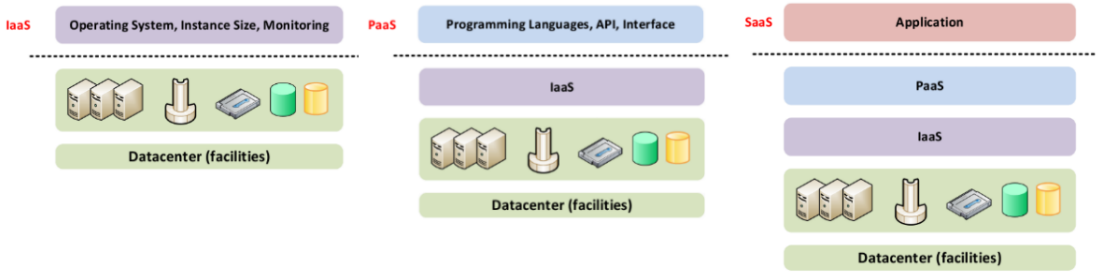
Nowadays, grid computing is mainly used in commercial organizations for its advantage of workload distribution. Grid computing are great in the sense that they provide fault tolerance which helps to provide better QoS requirements.

2.1.4 Cloud computing

Cloud computing is a new paradigm of computing. It was developed with the combination and evolution of distributed computing and virtualization, with strong contributions from grid and parallel computing (BUYYA et al., 2009). There are many efforts to provide a definition of cloud computing, such as the work of Grid Computing and Distributed Systems Laboratory (BUYYA et al., 2009) and the initiative from Berkeley University (BUYYA et al., 2009). In 2011, NIST (National Institute of Standards and Technology) has published its definition that consolidates several studies and became widely adopted.

Cloud computing has two main actors that are defined as the user and the provider. The user is defined as the consumer and can be a single person or an entire organization. The provider is an organization that provides the services to the user. According to NIST

Figure 2.1: Cloud computing service models stack and their relationships



definition (MELL; GRANCE et al., 2011), cloud computing is a model that conveniently provides on-demand network access to a shared pool of configurable computing resources that can be provisioned and released quickly without large management efforts and interaction with the service provider. This model definition is composed of five essential characteristics, three service models and four implementation models, which will be discussed in this section.

A cloud computing service needs to present the following characteristics to be considered adherent to the NIST definition: On-demand service; Broad network access; Resource Pooling; Rapid Elasticity and Measured Service.

As it is shown on Figure 2.1.4, the services provided by a cloud provider are categorized into three service models (BADGER et al., 2011): Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Some providers denominates other service models such as Database as a Service and Framework as a Service. Those models are commonly defined as Everything as a Service (XaaS), but it is possible to classify these models into one of the three defined by NIST.

The NIST cloud definition lists four implementation models: private, community, public and hybrid. Each one of these models has its particularities regarding to aspects such as network dependency, security, quantity of resources, among others. In the public model the user rents the resources from a provider, the private and community models can be used in two configurations, outsourced, where the user rents exclusive resources from a provider, and on-site, where the resources are owned by the user. The hybrid model is a combination between any of the other three models.

Generally we can say that a cloud service is controlled by a cloud platform, which is responsible for all procedures related to the service. A cloud platform is a very abstract term. In a practical approach it is made up of several components that are responsible for its operation.

The base of any cloud service is the hardware. By hardware we mean servers, storage and networking equipment. In a public cloud provider this hardware is maintained in a datacenter facility. The hardware is normally grouped into a container, that holds thousands of physical machines and storages interconnected with a high-speed network. This strategy is used by providers to optimize energy consumption. The containers are switched on and off according to user demand, each container has its own cooling system that is only used when the container is turned on.

In a cloud service normally the machines and storages offered to the users are a virtualization of real hardware. The component that performs this virtualization is the hypervisor. Basically the hypervisor controls the underlying hardware and provides VMs to the upper layers of the cloud platform. The hardware is a group of machines composed by different sizes and configurations and also a different type of processor architectures and operating systems. The purposes of the use of a hypervisor are to take care of all this heterogeneity and provide a standard interface to the cloud platform. Example of hypervisors that can be used are: Xen [1], KVM[2], Virtualbox[3], Hyper-V[4] and VMware[5]. The hypervisor layer delivers basically virtual machines to the other layers. The Virtual Machines (VMs) are abstractions of real hardware and can be used for general purposes.

The resource manager is responsible for providing the interface between the resources and the cloud platform. It controls the VM allocation and deallocation, also the VM migration between different servers is controlled in this layer. The security of resources is defined in this layer too. For example, the policies of VMs interconnection are defined by the resource manager. This is the main component regarding the energy consumption. Because this is the layer that decides when to power up a new machine, or an entire container, according to the demand. The resource manager has the responsibility to perform the VM consolidation to be possible turn off a server.

The main part of a cloud service is the cloud manager. This component performs the entire administrative tasks of the cloud platform. The user authentication is performed by the manager, that has a complete user record system controlling each user rights. The consumption of resources that each user makes of the system and the pricing mode are also controlled by this component. This control is used for billing purposes, in this way it can be stated that the pay-per-use charging model is implemented here. The instance

[1] <http://www.openstack.org/>
[2] <http://www.linux-kvm.org/>
[3] <https://www.virtualbox.org/>
[4] <http://www.microsoft.com/hyper-v-server>
[5] <http://www.vmware.com/>

sizes of VMs and the standard operating systems are also defined in the cloud manager. All the capabilities of customization of the images, size changing, multiple creation, user access security, among others are controlled in this layer too. The user reports are generated and provided here. Examples of cloud managers are: OpenStack [6], Eucalyptus [7], OpenNebula [8] and Nimbus [9]. Several providers implement their own proprietary cloud managers (ROLOFF et al., 2017).

The user interface is the front-end layer of the cloud platform. All the user-provider interaction is made through this layer. The user interface is normally a web page or a smart phone application, from the point of view of the user this layer is the entire cloud platform. Commonly the cloud managers provide a standard user interface, but each provider customizes it.

Microsoft started its initiative in cloud computing with the release of Microsoft Azure[10] in 2008, which initially was a PaaS to develop and run applications written in the programming languages supported by the ṄET framework. At these days, the company owns products that covers all types of service models.

Microsoft Azure PaaS is a platform developed to provide to the user, the capability to develop and deploy a complete application into Microsoft's infrastructure. To have access to this services, the user needs to develop his application following the provided framework. The Azure framework has support to a wide range of programming languages, including all .NET languages, Python, Java and PHP. A generic framework is provided in which the user can develop in any programming language that is supported by Windows Operating System (OS).

Microsoft Azure IaaS is a service developed to provide to the user access to VMs running in Microsoft's infrastructure. The user has a set of base images of Windows and Linux OS, but other images can be created using Hyper-V. The user can also configure an image directly into the Azure and capture it to use locally or to deploy to another provider that supports Hyper-V.

In this work, we use Microsoft Azure IaaS extensively, as the basis for our deployment and evaluation. It completely fulfills our needs for a stable and flexible large

[6] <http://www.openstack.org/>
[7] <http://www.eucalyptus.com/>
[8] <http://opennebula.org/>
[9] <http://www.nimbusproject.org/>
[10] <http://azure.microsoft.com/>

scale platform to deploy VMs, as well as provides several tools to partially automate our deployments of VMs using the Azure Command Line Interface (CLI) [11].

2.1.5 Distributed event stream processing systems

Applications that require real-time or near real-time processing functionalities are changing the way that traditional data processing systems infrastructures operate. They are pushing the limits of current processing systems forcing them to provide better throughputs with the lowest possible latencies.

The main problems to be solved nowadays are not primarily focused on raw data, but rather in the high-level intelligence that can be extracted from it. As a response, systems were developed to filter, aggregate and correlate data, and notify interested parties about its results, abnormalities, or interesting facts.

However, the distributed processing ecosystem today is mostly focused on Hadoop (WHITE, 2012), which itself is the result of the Google's Inc. research effort into large scale processing, that ended up producing several tools — such as MapReduce (DEAN; GHEMAWAT, 2004) and Google File System (GFS) (GHEMAWAT; GOBIOFF; LEUNG, 2003) — that were rewritten for the open source community through the Apache Foundation.

Hadoop has proved that the development of large scale distributed processing systems on the cloud is achievable. After understanding that it was possible to develop such systems, better approaches were proposed using Hadoop's infrastructure, but focusing on improving the performance of these kinds of systems. The aim was not to limit them only to batch processing, but to evolve them into systems of near real-time processing (CARVALHO; ROLOFF et al., 2013).

During the development of applications that aim to achieve better throughputs using Hadoop, its bottlenecks were exposed, proving that it is not the best platform for certain kinds of intensive data processing systems, being better for workloads that are more batch processing oriented. Aiming improvements in the fields in which Hadoop failed, new approaches to distributed processing were proposed, focusing each time on more reliable processing systems that are not heavily bounded by intensive processing workloads (PAVLO et al., 2009).

[11] <https://azure.microsoft.com/pt-br/documentation/articles/xplat-cli/>

These efforts generated a convergence between event processing systems and distributed processing systems, in direction for a merge between those fields. Nowadays event processing systems have been developed focusing on ways for distribute data processing. On the other hand, the most recent distributed processing systems also include complex built-in tools and specific Advanced Programming Interface (API) for easier the process of analysing data (CARVALHO; ROLOFF; NAVAUX, 2013).

We can trace the development of the event stream processing area back to Data Stream Management Systems (DSMSs), such as TelegraphCQ (CHANDRASEKARAN et al., 2003) and Aurora/Borealis (ABADI et al., 2003) (ABADI et al., 2005), which are similar to Database Management Systems (DBMSs), but focused on managing continuous data streams. In contrast to DBMSs, they execute a continuous query that is not only performed once, but is permanently executed until it is explicitly stopped. The development of the area could also be traced back to the origins of Complex Event Processing (CEP) systems, which are event processing systems that combine data from multiple sources to infer events or patterns that suggest more complicated situations. These systems are represented broadly by traditional content-based publish-subscribe systems like Rapide (LUCKHAM et al., 1995) and TESLA/T-Rex (CUGOLA; MARGARA, 2010) (CUGOLA; MARGARA, 2012).

In the evolution of both kinds of systems, a process of convergence between DSMSs systems and CEP systems had generated intersections between those fields, complicating more the characterization of them in distinct and clear groups.

Aiming to solve this naming problems, efforts were done to group all those kinds of systems into a common terminology. The term Information Flow Processing (MARGARA; CUGOLA, 2011) was created to refer to an application domain in which users need to collect information produced by multiple sources, to process it in a timely way, in order to extract new knowledge as soon as the relevant information is collected.

As well as the information processing systems had aggregated characteristics of distributed processing systems, many distributed processing system are aggregating information flow processing capabilities into their platforms. These changes are making it harder to explain the differences between them, because they are merging into tools that offer characteristics of both of them (CARVALHO; ROLOFF; NAVAUX, 2013).

These new systems have their designs strongly driven by the trend towards cloud computing, which requires the data stream processing engines to be highly scalable and robust towards faults. This trend can be seen through well-known systems of this gen-

eration, such as Flink (CARBONE et al., 2015) and Spark Streaming (ZAHARIA et al., 2013).

More specifically, this generation of streaming systems present a pattern towards a set of common requirements: 1) the scenarios typically involve input streams with high up to very high data rates (> 10000 event/s); 2) they have relaxed latency constraints (up to a few seconds); 3) the use cases require the correlation among historical and live data; 4) they require systems to elastically scale and to support diverse workloads and; 5) they need low overhead fault tolerance, supporting out-of-order events and exactly once semantic (HEINZE et al., 2014).

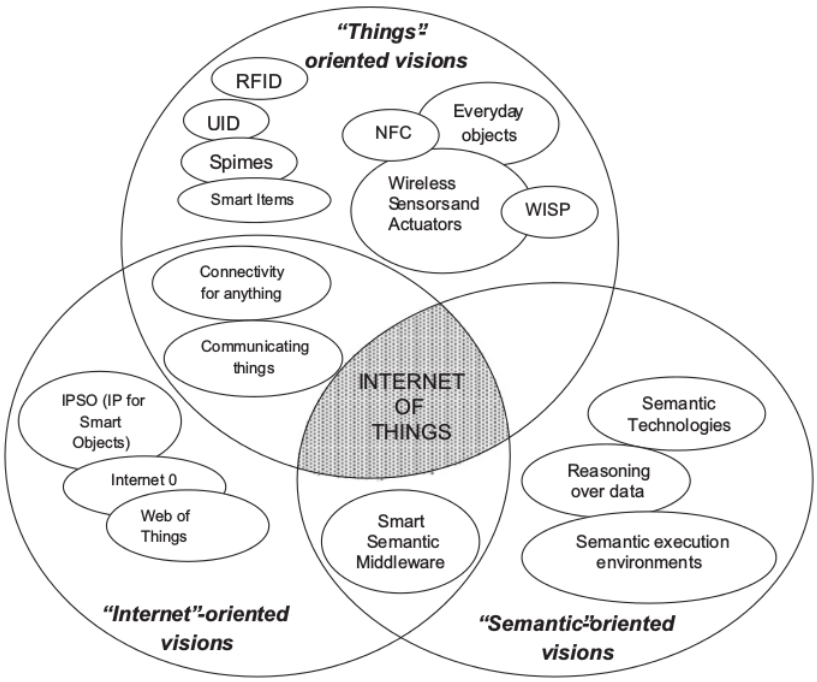
2.1.6 Internet of things

IoT is a novel paradigm that is rapidly gaining ground in the scenario of modern wireless telecommunications. The IoT builds on the pervasive presence around us of a variety of things or objects – such as Radio-Frequency Identification (RFID) tags, sensors, actuators, mobile phones, etc. – which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals (ATZORI et al., 2010).

However, for the IoT vision to successfully emerge, the computing criterion will need to go beyond traditional mobile computing scenarios that use smartphones and portables, and evolve into connecting everyday existing objects and embedding intelligence into our environment. For technology to disappear from the consciousness of the user, the IoT demands: (1) a shared understanding of the situation of its users and their appliances, (2) software architectures and pervasive communication networks to process and convey the contextual information to where it is relevant, and (3) the computational artifacts in the IoT that aim for autonomous and smart behavior. With these three fundamental grounds in place, smart connectivity and context-aware computation via anything, anywhere, and anytime can be accomplished (YAN et al., 2008).

Gartner, Inc. forecasts that the IoT will reach 26 billion units by 2020, up from 0.9 billion in 2009, and will impact the information available to supply chain partners and how the supply chain operates. From production line and warehousing to retail delivery and store shelving, the IoT is transforming business processes by providing more accurate and real-time visibility into the flow of materials and products. Firms will invest in the IoT to redesign factory workflows, improve tracking of materials, and optimize distribution

Figure 2.2: The internet of things paradigm as the convergence of different visions (ATZORI et al., 2010)



costs. For example, large enterprises such as John Deere and UPS are already using IoT-enabled fleet tracking technologies to cut costs and improve supply efficiency (LEE et al., 2015).

In Figure 2.2, the main concepts, technologies and standards are highlighted and classified with reference to the IoT visions they contribute to characterize best. From such an illustration, it clearly appears that the paradigm shall be the result of the convergence of three main visions. The "Internet oriented" together with the "Things oriented" perspective imply the huge number of interconnected devices connected through unique addressing protocols. The "Semantic oriented" perspective, on the other hand, represents the challenges in data representation for storage and information exchange (ATZORI et al., 2010).

2.1.7 Fog computing

The concept of fog computing (STOJMENOVIC, 2014) emerged from the concept that part of the computing can be brought back near the edge devices. The term fog computing has been proposed by Cisco in 2012.

The fog computing architecture extend the cloud computing paradigm to the edge of the network. The edge devices (i.e. routers, gateways etc.) can be used as the computing nodes along with the existing cloud data centers.

Fog computing has been envisioned to provide computation from the network edge, through the network core and to the cloud data centers.

The different services are hosted in the fog nodes, which are using its resources through the hypervisor, the management software for virtualizing the computing environment. Fog computing does the proper interplay of the services with the cloud.

The applications which require real time response and context aware computing rely on the fog computing framework. Furthermore, there are situations where there are need for supporting huge amounts of data generated from the IoT devices. Cloud computing alone is not sufficient in these situations as there is a requirement of real-time service provisioning. The typical applications of fog computing paradigm can be in real time health-care monitoring systems, smart cities, smart grids, vehicular ad-hoc network (VANET) etc. Being loosely coupled and highly distributed, QoS management and dynamic adaptability are the key challenges faced by the fog computing domain which need to be solved.

The OpenFog Consortium was founded to drive industry and academic leadership in fog computing architecture, testbed development, and a variety of inter-operability and composability deliverables that seamlessly leverage cloud and edge architectures to enable end-to-end IoT scenarios. OpenFog Consortium published a white paper on fog computing in February 2016, in which the OpenFog Consortium's approach to an open fog computing architecture (OpenFog architecture) has been outlined (GROUP et al., 2016).

Fog computing different from edge computing provides tools for distributing, orchestrating, managing and securing resources and services across networks and between devices that reside at the edge. Edge architecture places servers, applications, and small clouds at the edge. Fog jointly works with the cloud, while edge is defined by the exclusion of cloud (AI; PENG; ZHANG, 2018).

2.1.8 Edge computing

Edge computing is a new paradigm in which substantial computing and storage resources – variously referred to as cloudlets, micro datacenters, or fog nodes – are placed at the Internet's edge in close proximity to mobile devices or sensors.

The roots of edge computing reach back to the late 1990s, when Akamai [12] introduced content delivery networks (CDNs) to accelerate web performance.

A CDN uses nodes at the edge close to users to prefetch and cache web content. These edge nodes can also perform some content customization, such as adding location-relevant advertising. CDNs are especially valuable for video content, because the bandwidth savings from caching can be substantial (WEIN et al., 2007).

Edge computing generalizes and extends the CDN concept by leveraging cloud computing infrastructure. As with CDNs, the proximity of cloudlets to end users is crucial. However, instead of being limited to caching web content, a cloudlet can run arbitrary code just as in cloud computing. This code is typically encapsulated in a VM or a lighter-weight recognition could be implemented with acceptable performance on a resource-limited mobile device by offloading computation to a nearby server.

Cloud computing's emergence in the mid-2000s led to the cloud becoming the most obvious infrastructure to leverage from a mobile device. Today, Apple's Siri and Google's speech-recognition services both offload computation to the cloud. Unfortunately, consolidation implies large average separation between a mobile device and its optimal cloud datacenter.

Observations about end-to-end latency and cloud computing were first articulated in a 2009 article that laid the conceptual foundation for edge computing (SATYANARAYANAN et al., 2009). In this article authors advocated a two-level architecture: the first level is today's unmodified cloud infrastructure; the second level consists of dispersed elements called cloudlets with state cached from the first level. Using persistent caching instead of hard state simplifies the management of cloudlets despite their physical dispersal at the Internet edge.

In order to accelerate the development of the ecosystem based on cloudlets, the Open Edge Computing (OEC) initiative has been launched in June 2015 by Vodafone, Intel, and Huawei companies in partnership with Carnegie Mellon University. Similarly, Nokia Networks company introduced a computing platform in 2013, which is integrated with the base station. The initial concept that applications and services are executed at the edge of the network has been formed gradually. In September 2014, a new industry specification group (ISG) was proposed to be set up in European Telecommunications Standards Institute (ETSI) to allow the creation of industry specifications for multi-access edge computing (MEC), which has been supported by Huawei, IBM, Intel, Nokia Net-

[12] <https://www.akamai.com>

works, NTT DoCoMo, Vodafone, and etc. In MEC World Congress 2016, the MEC ISG has renamed Mobile Edge Computing as Multi-access Edge Computing in order to reflect the growing interests from non-cellular operators (AI; PENG; ZHANG, 2018).

Edge computing clearly offers many benefits. At the same time, it also faces many technical and nontechnical challenges. On the technical side, there are many unknowns pertaining to the software mechanisms and algorithms needed for the collective control and sharing of cloudlets in distributed computing. There are also substantial hurdles in managing dispersed cloudlet infrastructure.

One of cloud computing's driving forces is the lower management cost of centralized infrastructure. The dispersion inherent in edge computing raises the complexity of management considerably. Developing innovative technical solutions to reduce this complexity is a research priority for edge computing. Another important area of study will be the development of mechanisms to compensate for the weaker perimeter security of cloudlets, relative to cloud datacenters.

Since the 60s, computing has alternated between centralization and decentralization. The centralized approaches of batch processing and timesharing prevailed in the 60s and 70s. During the 80s and 90s saw decentralization through the rise of personal computing. By the mid-00s, the centralized approach of cloud computing began its ascent to the preeminent position that it holds today. Edge computing represents the latest phase of this ongoing dialectic (SATYANARAYANAN, 2017).

2.2 Smart grid

For 100 years, there has been no change in the basic structure of the electrical power grid. Experiences have shown that the hierarchical, centrally controlled grid of the 20th Century is ill-suited to the needs of the 21st Century. To address the challenges of the existing power grid, the new concept of smart grid has emerged. The smart grid can be considered as a modern electric power grid infrastructure for enhanced efficiency and reliability through automated control, high-power converters, modern communications infrastructure, sensing and metering technologies, and modern energy management techniques based on the optimization of demand, energy and network availability, and so on. While current power systems are based on a solid information and communication infrastructure, the new smart grid needs a different and much more complex one, as its dimension is much larger (GÜNGÖR et al., 2011).

According to the U.S. Department of Energy report, the demand and consumption for electricity in the U.S. have increased by 2.5% annually over the last 20 years (GUNGOR; LU; HANCKE, 2010). Today's electric power distribution network is very complex and ill-suited to the needs of the 21st Century. Among the deficiencies are a lack of automated analysis, poor visibility, mechanical switches causing slow response times, lack of situational awareness, etc (ENERGY, 2015). These have contributed to the blackouts happening over the past 40 years. Some additional inhibiting factors are the growing population and demand for energy, the global climate change, equipment failures, energy storage problems, the capacity limitations of electricity generation, one-way communication, decrease in fossil fuels, and resilience problems (EROL-KANTARCI; MOUFTAH, 2011). Also, the greenhouse gas emissions on Earth have been a significant threat that is caused by the electricity and transportation industries (SABER; VENAYAGAMOORTHY, 2011). Consequently, a new grid infrastructure is urgently needed to address these challenges.

Due to the emergence of newer technologies to energy grids and their direct applicability to cloud and edge processing, smart grids were selected to be the case study used to evaluate the model implementation.

2.2.1 Advanced metering infrastructure

The Advanced Metering Infrastructure (AMI) is regarded as the most fundamental and crucial part of smart grid. It is designed to read, measure, and analyse the energy consumption data of consumers through smart meters in order to allow for dynamic and automatic electricity pricing.

AMI requires a two way communication and spans through all the network components of smart grid from the private networks and Field Area Networks to Wide Area Networks. AMI goes beyond automatic meter reading scenarios which according to IEC 61968-9 — a series of standards under development that will define standards for information exchanges between electrical distribution systems — only have to do with meter reading, meter events, grid events and alarms. AMI will include customer price signals, load management information, power support for prepaid services, Home Energy Management Systems and Demand Response. It can also be used to monitor power quality, electricity produced or stored by distributed energy resources units as well as interconnected intelligent electronic devices (ANCILLOTTI; BRUNO; CONTI, 2013).

In addition, AMI is also expected to support customer switch between suppliers and help in detection and reducing electricity theft. Electricity theft has plagued many utilities companies especially in developing countries. To address these issues, authors in (ANAS et al., 2012) have reviewed electricity theft and reduction issues using security and efficient AMIs (TSADO; LUND; GAMAGE, 2015).

2.2.2 Demand side management

Demand Side Management (DSM) is the action that influences the quantity or pattern of energy consumption by end users. These actions may include targeting reduction of peak demand by end users during periods when energy supply systems are constrained. Energy peak management does not necessarily decrease the amount of total energy consumption, but it will reduce the need for investments on power generation sources or spinning reserves at peak periods (WANG; XU; KHANNA, 2011) (DAVITO; TAI; UHLANER, 2010). DSM includes the following:

- Demand Response enabling the utility operator to optimally balance power generation and consumption either by offering dynamic pricing programs or by implementing various load control programs.
- Load Management through dynamic pricing which helps to reduce energy consumption during peak hours by encouraging customers to limit energy usage or shifting demand to other periods. Existing dynamic pricing programs include: Time-of-use, Real-Time Pricing, Critical Peak timing and Peak time Rebates.
- Conservation of energy through load control program which involve performing remote load control programs where communicating networks are used to control usage of appliances remotely to use less energy across many hours (TSADO; LUND; GAMAGE, 2015).

2.2.3 Consumption forecasting

The term forecasting is frequently confused with the terms prediction and predictive analytics. A prediction in the general sense involves an imagination of an oracle which can reason about the past based on some experience and which on this basis is able to look into future to predict a certain event.

Prediction and predictive analytics in the scientific sense means predicting a behavior of someone or a trend characterized with a probability and based on statistical data analysis and the current evolution. In contrast, forecasting refers to predicting an (aggregated) value, or an occurrence of an event at certain time point, based on historical data analysis, the current state and sometimes on predictive analytics (ANALYTICSWORLD, 2015).

Electricity load forecasts provide a prediction of an amount of electricity consumed at a certain point of time. The purpose of electricity load forecasting is in most cases an efficient economic and quality planning. Good forecasts ensure economic profitability of the service and safety of the network.

Energy consumption forecasts can be performed on different levels of time interval resolution. The range of the forecasts generally depends on the available reliable data and the goal of the forecast. Usually, the following three terms for forecasting interval are used: short term, medium term and long term forecasts (ALFARES; NAZEERUDDIN, 2002).

- Short Term Load Forecasting (STLF) means to give forecasts for the next minutes up to one day on minutes or hourly basis. Such forecasts are required for the scheduling, capacity planning and control of power systems.
- Medium Term Load Forecasting (MTLF) are required for planning and operation of power systems. Such forecasts can be provided from one day to months ahead on hourly or days basis.
- Long Term Load Forecasting (LTLF), in contrast to short and medium term forecasting which support operational decisions, has the aim to support strategic decisions, more than a year ahead.

Metrics to measure the quality of load forecasting can be subdivided into two main categories: Measuring the forecast accuracy and measuring the processing delay (latency).

2.3 Related work and discussion

The main works on the state of the art of edge computing focus on platforms and frameworks aiming to provide scalable processing as close as possible to the network border. These approaches are primarily focused on providing the lowest possible latency

results and better utilize the resources available on the network. MECs and Cloudlets, which can be described as cloud-like deployments at the network edge, are currently the predominant approaches to address these challenges.

The most prominent approaches and how they relate to this work are briefly described below. At the end of this section there are detailed explanations of the relationship between the works on the state of the art and the model proposed in this work.

The state of the art works include FemtoClouds (HABAK et al., 2015), REPLISOM (ABDELWAHAB et al., 2016), Cumulus (GEDAWY et al., 2016), CloudAware (ORSINI et al., 2016), ParaDrop (LIU et al., 2016), HomeCloud (PAN et al., 2016), ENORM (WANG et al., 2017), RT-SANE (SINGH et al., 2017), EdgeIoT (SUN; ANSARI, 2016) and cloud provider based implementations (TÄRNEBERG; CHANDRASEKARAN; HUMPHREY, 2016), which are examples of applications that explore computational offloading to nearby devices.

One important thing to perceive is that the majority of the related works either rely on offloading computation to edges that are underutilized or offload processing to nearby network centralizers (modified wireless network access points or specialized mobile network base station hardware). EdgeIoT (SUN; ANSARI, 2016) stands apart on this aspect by exploring computation to considerably more performance nodes, by relying on virtual machines in a nearby mobile base station.

Several works on the state of the art either rely on hardware specific tools or significant modifications on their underlining communication protocols. This work, on the other hand, relies on standard tools and protocols that add flexibility and turn easier to port it to other platforms.

Examples of applications which require significant changes in the underlining communication protocols or specific hardware are ParaDrop, a specific edge computing framework implemented on WiFi Access Points (APs) or other wireless gateways (such as set-top boxes). It uses a lightweight virtualization framework through which third-party developers can create, deploy, and revoke their services in different APs.

HomeCloud (PAN et al., 2016) focus on being an open and efficient new application delivery in edge cloud, by integrating two complementary technologies: Network Function Virtualization (NFV) and Software-Defined Networking (SDN).

EdgeIoT, in its turn, is an architecture to handle data stream at the mobile edge. The central idea consists of fog nodes communicating with a VM positioned at a nearby

Table 2.1: Research scope comparison of the state-of-the-art with the proposed work

Name	Cloud	Edge	Mobility	Large Scale	Hardware Agnostic
GaruaGeo (this work)	•	•		•	•
ENORM	•	•			•
RT-SANE	•	•			•
Tarneberg et al.	•	•			•
HomeCloud	•	•			•
CloudAware	•	•	•		
FemtoClouds		•	•		
REPLISOM		•	•		
Cumulus		•	•		•
ParaDrop		•			•
EdgeIoT		•	•		

base station. On the top of the fog nodes, the SDN based cellular core is designed to facilitate the package forwarding among fog nodes.

Multiple works, like Cumulus (GEDAWY et al., 2016), FemtoClouds (HABAK et al., 2015), CloudAware (ORSINI et al., 2016) and RT-SANE (SINGH et al., 2017) have a greater focus on task placement, providing dynamic scheduling capabilities for operators and tasks (such as tasks migration functionalities) which are beyond the scope of this work. Cumulus (GEDAWY et al., 2016) provides a complete framework that controls task distribution under a heterogeneous set of devices on its cloudlet. FemtoClouds (HABAK et al., 2015) and CloudAware (ORSINI et al., 2016) monitor device usage on its cloudlets in order to improve device usage as part of its scheduling algorithms. CloudAware (ORSINI et al., 2016) also provides a specific API to improve the user experience of software developers. RT-SANE (SINGH et al., 2017) evaluates several scheduling heuristics in comparison to a cloud-only scenario.

Although the related works present multiple initiatives on Edge Computing towards computational offloading, only the work of Tärneberg et al. (TÄRNEBERG; CHANDRASEKARAN; HUMPHREY, 2016), ENORM (WANG et al., 2017) and RT-SANE (SINGH et al., 2017) explore the potential of combining low latency edge nodes processing with scalable and more powerful sets of commodity machines on public clouds. However, ENORM (WANG et al., 2017) evaluation only considers a small set of edge nodes communicating with a nearby Amazon Web Services (AWS) cloud node in Dublin. RT-SANE (SINGH et al., 2017), in its turn, relies on a specific fog simulator to obtain

its results, which limits the scope in comparison to a real-world evaluation. Finally, the work of Tärneberg et al. (TÄRNEBERG; CHANDRASEKARAN; HUMPHREY, 2016) directly offload from the edges devices to the cloud infrastructure, which limits the scope of the work in relation to data aggregation.

These types of systems, which represent the convergence between edge and cloud computing, are described in a recent survey paper as 4th generation distributed stream processing systems (ASSUNCAO; VEITH; BUYYA, 2018).

Apart from that, most works presented in this section either rely on generated datasets or small datasets (tenths of mobile devices) for its evaluations. On the other hand, the present work uses a realistic dataset, based on a real-world dataset from household energy consumption in Germany (ZIEKOW; JERZAK, 2014).

In Table 2.1, we present a comprehensive description of the coverage of the state of the art in comparison with this work. In comparison to the state of the art, this work is the only one that combines the given characteristics: (1) Focus on hybrid processing (edge nodes working in collaboration with cloud computing resources); (2) Does not focus on mobility issues; (3) Does not rely on specific hardware extensions to its processing mechanism (uses commodity hardware); (4) Provides a large scale evaluation (thousands of nodes in comparison to tenths of nodes on the similar works).

The taxonomy used follows the ideas found on recent surveys on fourth-generation distributed stream processing systems, fog computing and edge computing (ASSUNCAO; VEITH; BUYYA, 2018) (ATZORI; IERA; MORABITO, 2010) (MAO et al., 2017) (MAHMUD; KOTAGIRI; BUYYA, 2018).

3 GARUA: ARCHITECTURE AND IMPLEMENTATION

The Garua architecture was designed to provide an innovative model to aggregate data in edge and cloud computing environments, combining the existing cloud computing resources and research knowledge to the recent advances in fog and edge computing.

This work builds on top of the authors previous research on cloud computing applied to IoT data profiles (CARVALHO; ROLOFF; NAVAUX, 2017).

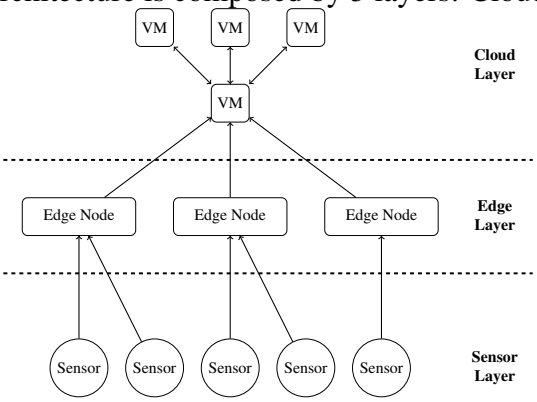
The main idea is that Garua should provide a high performance aggregation model that leverages the existing research in distributed stream processing systems and applies it to the context of edge computing.

In this Chapter, we introduce the Garua architecture, their layers and the reasoning behind the model design and implementation. Section 3.1 describes which layers compose the model and explains their high level responsibilities. Section 3.2 describes the cloud layer, its design and details about the implementation. Section 3.3 describes the edge layer, its design and details about the implementation using physical hardware. Section 3.4 describes the sensor layer, its design and details about their simulation using a realistic dataset. Section 3.5 describes the reasoning behind the selection of the communication protocol. Section 3.6 describes the algorithm used to evaluate the platform on Chapter 4.

3.1 Architectural overview

The architectural infrastructure of the testbed application deployment can be described as a composition of three layers, as it is represented on Figure 3.1: (1) Cloud layer, which executes long running scalable jobs that can provide more performant processing with the trade-off of greater latencies; (2) Edge layer, which is composed by edge nodes that are used to pre-process and aggregating data before sending to the Cloud; and the (3) Sensor layer, which is composed by the sensors that communicate directly with Edge layer nodes to receive actuation requests and provide measurements to the network.

Figure 3.1: The architecture is composed by 3 layers: Cloud, Edge and Sensor



3.2 Cloud layer

This layer is composed by virtual machines that execute the application in order to aggregate data to be received from edge layer nodes. The cloud layer should be composed by elements that can be able to process data as it arrives. It can be instantiated by implementing the same application logic from the layer below, but instead it should be configured to receive data from multiple edge nodes.

This layer receives data from queues and exchanges through a message hub, so that the inputs can be parallelized through multiple consumers. It can also be configured to support clusters of machines to execute transformations as distributed stream processing jobs over these queues and exchanges.

Table 3.1: Cloud layer configuration: Virtual machine type and toolset description

Parameter	**Description**
Instance Type	Basic_A3 (4 cores, 7 GB RAM)
Operating System	Ubuntu 16.04 LTS
Location	Brazil South
Golang version	1.8
GRPC version	1.3.0-dev
Protocol Buffers version	3.2.0
Network Interconnection	5Mbps HFC (Edge to Cloud)

In the evaluation work, the model is implemented as an application running in a single node inside a Linux VM at Microsoft Azure, which was chosen due to its perceived benefits over other cloud platforms (ROLOFF et al., 2012) (ROLOFF et al., 2017). This application was written in Go programming language and receives processing request from the layer below through GRPC Remote Procedure Calls (GRPC) communication framework. The VM instance utilized was configured as it is described in Table 3.1.

3.3 Edge layer

The edge layer is composed by a set of nodes with transformation operators to apply over sensor measurements one-at-time. Operators can be either transformations over results, combinations with sets of measurements received or mappings to machines on the cloud layer above.

On this layer, the application code will be expressed to define which computation will be done inside of edge nodes and which computation will be managed by VMs on the cloud computing environment. The degree of control provided by this level makes it possible to decrease the number of messages sent to the cloud. In this way, it is possible to decrease the amount of data that is sent to the cloud. Also, by processing certain amounts of data directly on the edge nodes, the latency experienced by actuator sensors is in the order of tenths of milliseconds instead of a couple of seconds of cloud processing latencies.

Actuator sensor logic can also be implemented on this layer, in such a way that when a given condition is matched by an edge node, it can trigger actuators on the Sensor layer in order to act on external applications. For example, a consumer can configure its smart grid energy meter to maintain the energy consumption below a certain level during peak cost energy hours. In this way, the smart grid meter can turn off certain machines when the average consumption reaches a certain threshold.

In the evaluation testbed, this layer was composed by a set of Raspberry Pi edge nodes connected to the internet through wireless connection. Each edge node is a complete Linux machine running an application in an Advanced RISC Machine (ARM) architecture. They were configured to communicate with the underlining sensors directly, as well as the Linux VM on the Azure cloud service. The configuration of the edge nodes is described in details on Table 3.2.

3.4 Sensor layer

The sensor layer is represented by a given set of sensors that communicate with the Edge nodes. Ideally, sensors should communicate with edge nodes through their available input/output hardware interconnections or lightweight wireless connection such as Bluetooth or Long-Term Evolution (LTE) networks. However, in order to limit the

Table 3.2: Edge layer configuration: Architecture and software description

Parameter	**Description**
Number of Edge Nodes	4
Hardware	Raspberry Pi Zero W
Hardware CPU	1 GHz Single Core CPU
Hardware RAM	512 MB
Operating System	Raspbian Jessie Lite 4.4
Golang version	1.8
GRPC version	1.3.0-dev
Protocol Buffers version	3.2.0
Wireless Router	HUMAX HG100R

analysis scope of this work, the sensor network dataset is previously loaded into edge nodes prior to the execution of tests.

Smart grid environments rely on specific meters and plugs on households to collect data, which are provided by the energy grid provider or standardized to support only a set of accepted and verified plugs and meters types. The data types generated by these environments also need to respect a certain schema to be shared, aggregated and analyzed by the energy provider companies.

The dataset used to the evaluation of this work is based on the dataset provided by by the 8th ACM International Conference on Distributed Event-Based Systems (DEBS). This conference provides competitions with problems which are relevant for the industry. In the year of 2014, the conference challenge focus was on the ability of CEP systems to apply on real-time predictions over a large amount of sensor data. For this purpose, household energy consumption measurements where generated, based on simulations driven by real-world energy consumption profiles, originating from smart plugs deployed in households (ZIEKOW; JERZAK, 2014). For the purpose of this challenge, a large number of smart plugs has been deployed in households with data being collected roughly every second for each sensor in each smart plug.

3.5 Communication protocol

Although multiple protocols for communication in IoT systems have been proposed in the recent years, the protocols in use today are still being evaluated and are subject of discussion and standardization initiatives, mainly due to advancements of internet protocols to support mobile and IoT applications. The most widely adopted protocols in

use today are, respectively, MQTT (BANKS; GUPTA, 2014) and CoAP (BORMANN et al., 2012).

One of the most prominent proposals on this area is the HTTP/2 protocol. The standard was finished in 2005 and provides several improvements over previous protocols, mainly due to the capability of multiplexing data, avoiding handshake overhead and their data compression capabilities (BELSHE et al., 2015) (RUELLAN; PEON, 2015).

As an alternative to broker centric communication protocols and synchronization costly protocols such as Representational State Transfer (REST) (RICHARDSON; RUBY, 2008), Google Inc. has adopted a Remote Procedure Call (RPC) protocol and service discovery framework Stubby/Chubby (BURROWS, 2006). The open source version of its tool is called GRPC (Google, 2015), which relies on HTTP/2 in order to avoid handshake overhead, and Protocol Buffers (GLIGORIĆ et al., 2011) to communicate using a binary method, which provides better data compaction by reducing the message size.

Due to the performance benefits reported from the usage of HTTP/2 protocols over standard HyperText Transfer Protocol (HTTP), and their ease of usage for flexible prototyping of distributed applications, GRPC was used to build a reliable and fast communication channel for all of the communication layers implemented on this work.

GRPC as a communication platform presents several advantages over TCP only connections and communication protocols such as REST. Its has a simple interface which hides configuration complexity but is still able to provide high level features, such as long lived connections (to avoid unnecessary communication handshakes) and communication multiplexing inside a small number of channels (reducing the number of required connections open at a given point in time). However, their usage is still subject of evaluation, mainly on networks with high package loss percentages, which are a limiting factor not only for HTTP/2 but also for Advanced Message Queuing Protocol (AMQP) based applications (GOEL et al., 2016) (LEE et al., 2013) (CHOWDHURY et al., 2015) (THANGAVEL et al., 2014).

3.6 Measurement algorithm

In order to extract metrics from the data collected by sensors, we have selected a widely used approach to aggregate and generate STLF for smart grids data. The algorithm was chosen not only due to the fact it is well-known by the community by the research

community but also because it provides the potential to be applied at multiple aggregation layers.

Smart grids promise to provide better control and balance of energy supply and demand through near real-time, continuous visibility into detailed energy generation and consumption patterns. Methods to extract knowledge from near real-time and accumulated observations are hence critical to the extraction of value from the infrastructure investment.

In this context, STLF refers to the prediction of power consumption levels in the next hour, next day, or up to a week ahead. Methods for STLF consider variables such as date (e.g., day of week and hour of the day), temperature (including weather forecasts), humidity, temperature-humidity index, wind-chill index and most importantly, historical load. Residential versus commercial or industrial uses are rarely specified.

Time series modeling for STLF has been widely used over the last 30 years and a myriad of approaches have been developed. These methods (KYRIAKIDES; POLYCARPOU, 2007) can be summarized as follows:

- Regression models that represent electricity load as a linear combination of variables related to weather factors, day type, and customer class.
- Linear time series-based methods including the Autoregressive Integrated Moving Average (ARIMA) model, auto regressive moving average with external inputs model, generalized auto-regressive conditional heteroscedastic model and State-Space Models (SSMs).
- SSMs typically relying on a filtering-based (e.g., Kalman) technique and a characterization of dynamical systems.
- Nonlinear time series modeling through machine learning methods such as nonlinear regression.

Shawkat Ali (ALI, 2013) argues that the three most accurate models for load prediction are, respectively, Multilayer Perceptron (MLP), Support Vector Machine and Least Mean Squares. Due to the model fit in relation to the distributed architecture, it was decided to implement an approach similar to the suggested by the DEBS 2014 conference committee (ZIEKOW; JERZAK, 2014), that is schematically described in Equation (3.1). This approach could be interpreted as a mixed approach between MLP and ARIMA. It brings together characteristics from both Linear time series-based methods and SSMs (BYLANDER; ROSEN, 1997).

More specifically, the set of queries provide a forecast of the load for: (1) each house, i.e., house-based and (2) for each individual plug, i.e., plug-based. The forecast for each house and plug is made based on the current load of the connected plugs and a plug specific prediction model.

$$L(s_{i+2}) = \frac{avgL(s_i) + median(avgL(s_j))}{2} \tag{3.1}$$

In the Equation (3.1), $avgL(s_i)$ represents the current average load for the slice s_i. The value of $avgL(s_i)$, in case of plug-based prediction, is calculated as the average of all load values reported by the given plug with timestamps $\in s_i$. In case of a house-based prediction the $avgL(s_i)$ is calculated as a sum of average values for each plug within the house. $avgL(s_j)$ is a set of average load value for all slices s_j such that:

$$s_j = s_{i+2-n*k} \tag{3.2}$$

In the Equation (3.2), k is the number of slices in a 24 hour period and n is a natural number with values between 1 and $floor(\frac{i+2}{k})$. The value of $avgL(s_j)$ is calculated analogously to $avgL(s_i)$ in case of plug-based and house-based (sum of averages) variants.

4 GARUA: EVALUATION

In this Chapter, the performance of the Garua architecture (CARVALHO et al., 2017b) is evaluated under three main aspects: Communication, application and latency impact. Section 4.1 analyses the network communication between edge processors and cloud nodes and compare the raw network traffic results to the application performance with distinct message sizes. Section 4.2 discusses about experiments executed to test potential hypothesis to improve the model implementation performance and their drawbacks. The model implementation is tested using multiple green threads, multiple edge nodes and message windowing at the edge. Section 4.3 evaluates the platform using a latency simulation mechanism, in order to further understand the latency impact for windowed and non-windowed requests.

4.1 Communication evaluation

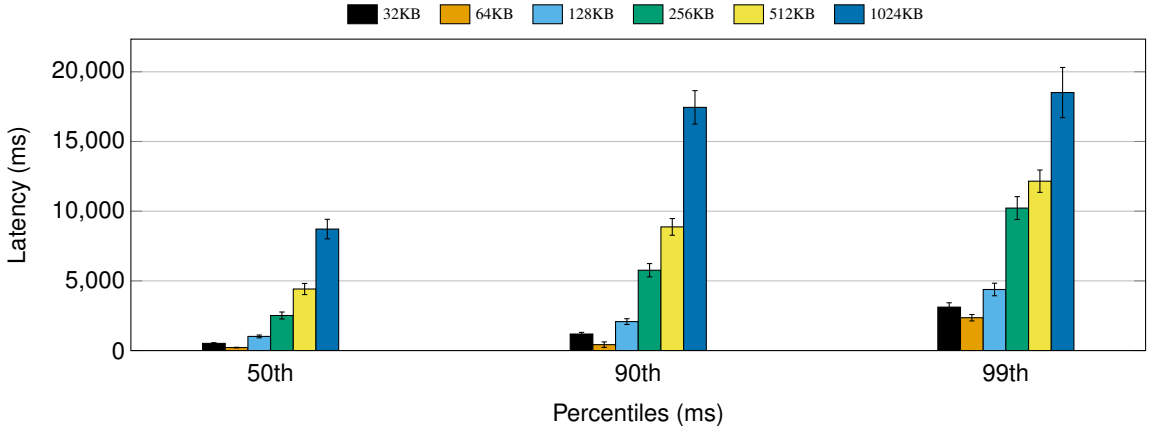
In order to successfully evaluate the testbed middleware implementation, the first step is to evaluate the underlining network connection. The network evaluation started by measuring the maximum amount of data that could be sent from edge nodes to the cloud provider, using the specified network router and the given network connection described on Chapter 3.

Table 4.1: Network measurements with Iperf

Parameter	**Edge to Edge**	**Edge to Cloud**
TCP window size	43.8 KByte	43.8 KByte
Interval	60 seconds	60 seconds
Total transfered	388MBytes	7.76 MBytes
Bandwidth	54.1Mbits/sec	1.03 Mbits/sec

The measurements started with the throughput analysis through the Iperf tool (TIRUMALA et al., 2005). Those experiments have shown that the average throughput was slightly below the network bandwidth described by the provider, which is common on internet providers of cable connections and was expected on the measurement results (DISCHINGER et al., 2007). The results obtained of this first measurement step are described on Table 4.1.

Figure 4.1: PingPong: Latency Percentiles by Message Sizes (32KB to 1MB).



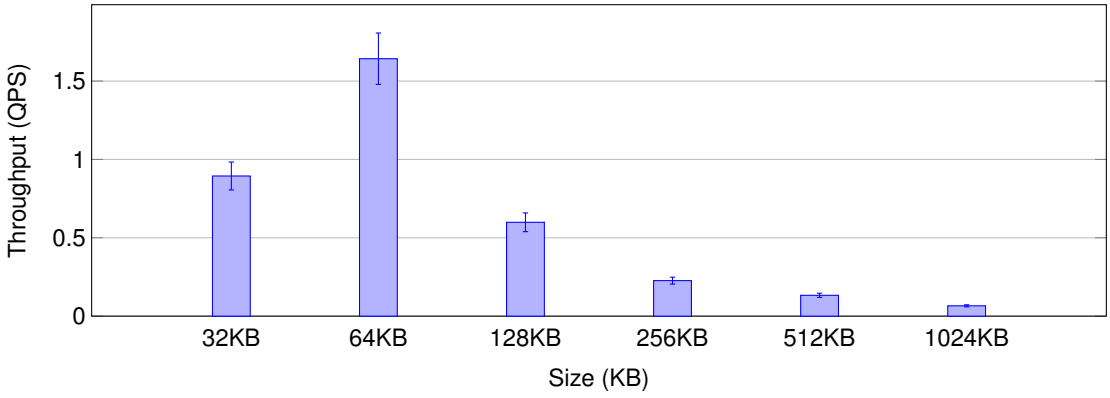
After observing the real throughput of the underlining network, it was designed a simple application in order to evaluate the performance or the GRPC middleware and the HTTP/2 protocol with varying message sizes.

The simple application designed for this task was called PingPong, which is responsible to execute the following steps: (1) sends a message from the edge node to the cloud node; (2) the cloud node receives the message and sends it back to the edge node, completing a round-trip.

Distributed applications are prone to performance penalties due to effect of fat tails on their latency distribution percentiles, which can be several times greater than the expected average latency. On applications with multiple users that send thousands of messages per second, it is a known fact that these fat tail latencies might be experienced by several users of these systems (MAAS et al., 2015) (BAILIS; KINGSBURY, 2014).

In the Figure 4.1 it is possible to analyze the impact of message sizes, from 32KB up to 1MB, in the latency of the messages being sent over the network. As shown on this figure, the 50th percentile (the median), is as low as a couple of milliseconds for small

Figure 4.2: PingPong: Maximum Throughput by Message Size (32KB to 1MB)



messages sizes, but it increases highly for tail latencies. The impact of messages that are delayed by Garbage Collection (GC) pauses, package losses or other network failures is a limiting factor depending on the application. These measurements also serve to as a guideline to build message windows, given that it is possible to expect, for example, messages latencies up to 2.5 seconds for 64KB messages at the 99th percentile.

The PingPong method was used to evaluate the maximum achievable network throughput of the communication middleware, as it can be visualized on Figure 4.2. This evaluation was important to understand that the usage of 64KB messages increases the overall application throughput, which is probably due to a better fit on Transmission Control Protocol (TCP) windows used by the GRPC communication framework.

4.2 Application evaluation

The application evaluation was done by distributing the aggregation step of the application processing between edge and cloud nodes. The main objective for this evaluation was to understand how latency impacts the processing throughput, as well as to analyze the performance gains obtained by aggregating data on edge nodes before sending data to the cloud.

In order to execute the evaluation, the data was preloaded on the edge nodes prior to the execution of the tests. The schema is quite similar to the original dataset and is composed by a timestamp, the value of the energy measurement (in Watts) and an identifier id of the house/plug that is been measured. As it is compressed by the Protocol Buffers binary protocol, the final message payload size is 32 bytes.

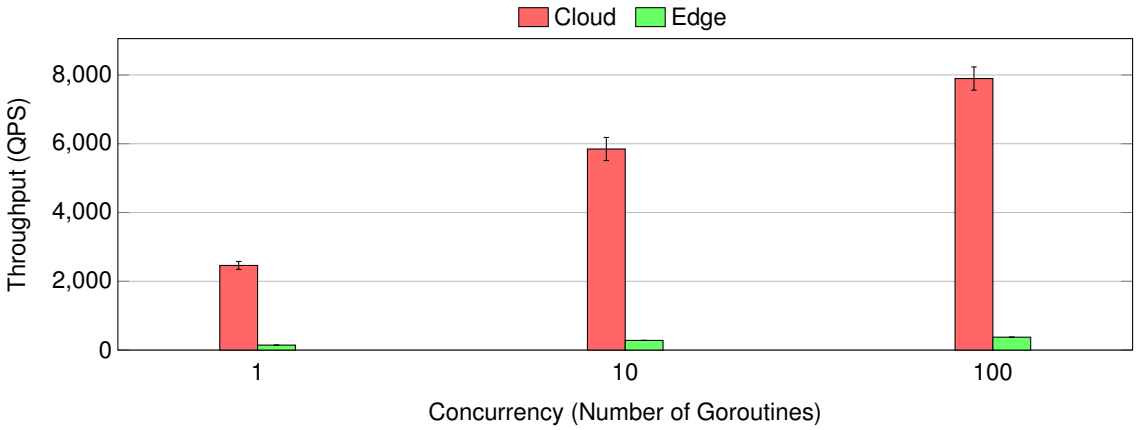
The evaluation can be better described in three phases: (1) Evaluation of the impact of the concurrency degree on the throughput; (2) Scalability evaluation to understand how the number of edge nodes impacts into the cloud node throughput; (3) Windowing and strategies to aggregate data on the edge node before sending data to the cloud node.

4.2.1 Concurrency evaluation

Given that edge nodes execute multiple requests per second to its respective cloud nodes, it is important to explore concurrency strategies to obtain performance gains by executing multiple concurrent requests to remote services.

In the Go programming language, the concurrency execution is done not directly through the creation of threads, but through Go's green threads model which are called Goroutines (TOGASHI; KLYUEV, 2014).

Figure 4.3: Concurrency Analysis: Impact of Goroutines usage on throughput (Edge and Cloud nodes).



The experiment on Figure 4.3 explores the interplay between the number of Goroutines used to process data and the overall application throughput obtained. The outcome of this experiment suggests that both edge and cloud nodes are able to benefit from concurrency. The experiments show that, in comparison with the sequential approach, it is possible to achieve a 4 times speedup on the testbed application by using 100 Goroutines.

4.2.2 Scalability evaluation

One important aspect of the model is the ability to aggregate messages from multiple edge nodes. In order to understand the limits of cloud nodes to receive messages from edge nodes, it was done a set of experiments to evaluate which was the perceived impact on throughput as it was increased the number of edge nodes. In this experiment, it was used a single cloud node communicating with one to four edge nodes.

As it is shown on Figure 4.4, a single cloud node was able to scale linearly up to four edge nodes, each one of them maintaining an average of approximately 500 requests per second.

4.2.3 Impact of message windowing

Finally, in order to explore the limitations of communication in terms of bandwidth and latency, it was decided to explore distinct possibilities to aggregate multiple energy measurements into edge nodes before sending data to cloud nodes. Prior to that, for each new measurement, it was necessary to receive data at the edge node, send it to the cloud to be processed and finally receiving an updated forecast.

In these experiments, as it is presented on Figure 4.5, it was analyzed the behavior of the testbed application when processing locally grouped sets of messages before sending it to the cloud. In this way, it was possible to validate the assumption that processing more messages at the edge improves the overall throughput and increases scalability (by decreasing the number of measurements being processed on the cloud node).

The results show that the overall application (the combination of cloud and edge nodes) was able to process almost 800k messages per second by using 4 edge nodes and window sizes of 1000 combined messages.

The approach that was used to build windows of processing consists in aggregating the local measurements into a unified representation of the set. Using this approach, it was possible to send the grouped view with the same payload size of a single message of the window.

Figure 4.4: Scalability Analysis: Throughput with multiple consumers (1 to 4 edge nodes).

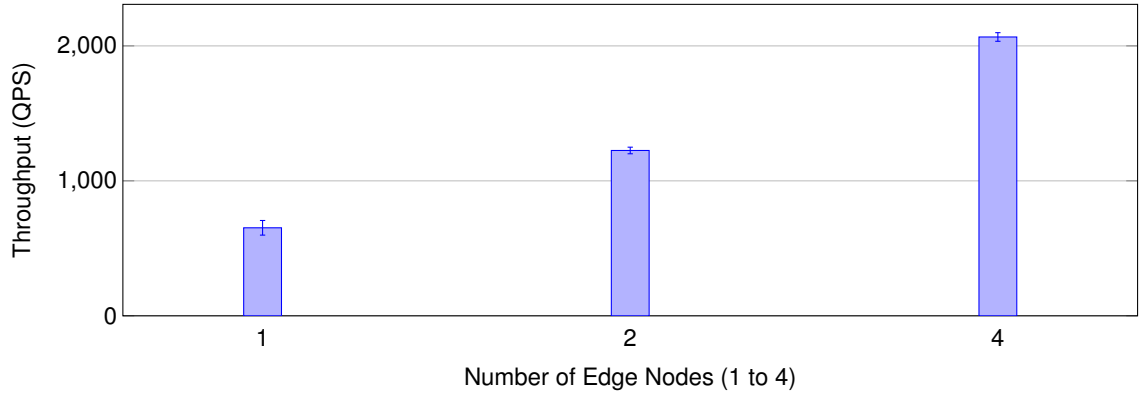
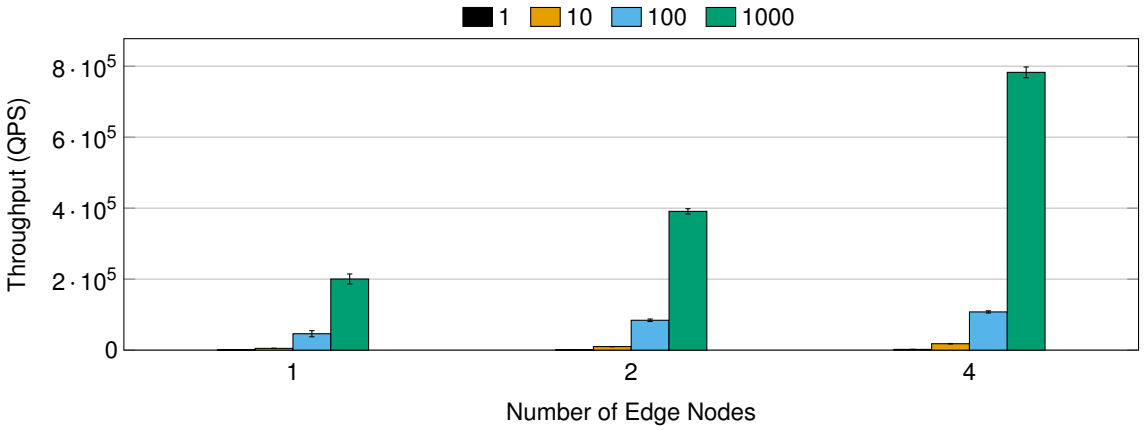


Figure 4.5: Windowing Analysis: Windowing impact on throughput (1 to 1000 messages per request).



4.3 Simulated latencies

In this section, it was made use of simulated latencies on a cloud computing environment to evaluate the platform's behavior on a large scale edge computing scenario.

All of the experiments described on this section were ran at least 20 times. Each one of the nodes used on the following tests has sent groups of 10000 or 100000 messages per execution between edge and cloud nodes. These experiments were executed using instances with concurrency degree of 100 goroutines, which was selected due to the results obtained on the previous experiments on Section 4.2.1.

In order to simulate the latency behavior, this simulation was first validated on Section 4.3.1. After this step, it was done the evaluation of the throughput obtained under distinct latency constrained scenarios on Section 4.3.2. Finally, on Section 4.3.3 follows the analysis of the behavior of the message windowing strategy under distinct latency profiles.

4.3.1 Validation

It was necessary to design a requests throttler for the application in order to simulate latencies on the cloud. The job of its requests throttler was to limit the number of data packets flowing between the simulated edge nodes – that were in this case VMs running on Microsoft Azure – and the cloud node.

The configuration used to run these experiments is shown on Table 4.2. It consists mainly of a single cloud node – with the same configuration described previously –

and sets of up to 90 simulated edge nodes, which are single core machines deployed on the same cloud region as the main cloud node. Virtual machines instances of type Standard_DS1_v2 were selected on Azure due to their similarity (single core with low amount of RAM) with the real edge node machines used on the previous analysis.

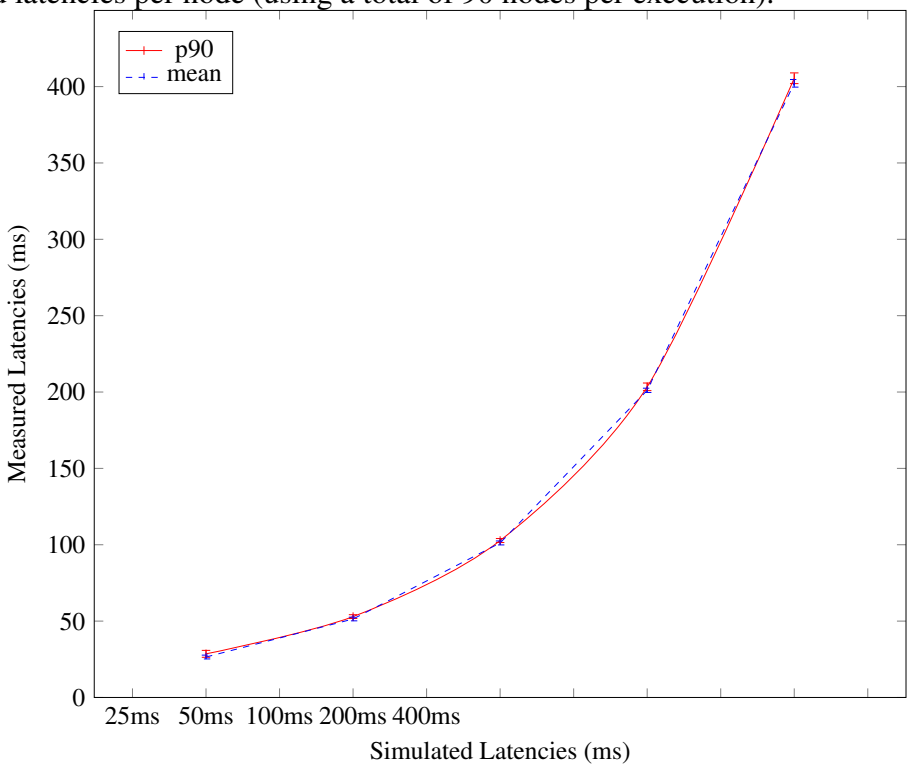
Table 4.2: Machines configuration: Virtual machine types and toolset description

Parameter	**Description**
Instance Type - Cloud Node	Basic_A3 (4 cores, 7 GB RAM)
Instance Type - Edge Nodes	Standard_DS1_v2 (1 core, 3.5 GB RAM)
Operating System	Ubuntu 16.04 LTS
Location	Brazil South
Golang version	1.8
GRPC version	1.3.0-dev
Protocol Buffers version	3.2.0

The accuracy of the throttling mechanism was analyzed by comparing the latency values of the requests between the simulated edge and cloud nodes.

In this experiment, it was done a comparison of real request latencies with simulated latencies, where values closer to equality on x and y axis mean that the simulated latencies are equal to the measured latencies, as it can be visualized on Figure 4.6.

Figure 4.6: Latency simulation validation: Simulated windowing latencies versus real obtained latencies per node (using a total of 90 nodes per execution).



These results show that the requests throttler implementation was able to simulate real latencies for at least 90 percent of the requests. These tests were ran more than 20 times with 90 nodes, each of them running 10000 requests per execution.

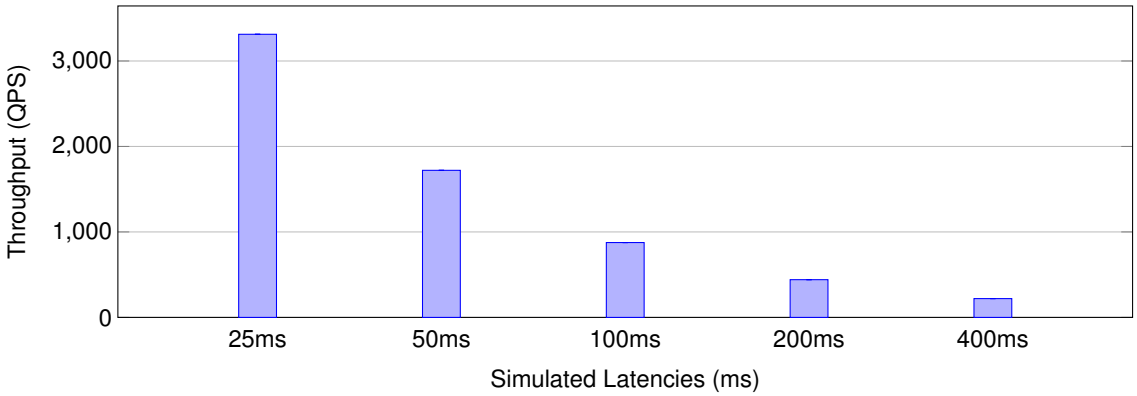
4.3.2 Throughput evaluation

In order to evaluate throughput under latency constraints, it was designed of two sets of experiments. The first experiment is the throughput evaluation based on the results found on the latency verification, as it can be seen on Figure 4.6. The second experiment expands this analysis to multiple latency behaviours under sets of 30, 60 and 90 edge nodes.

On the first experiment, the throughput was evaluated under multiple latency constraints. The same amount of nodes and simulated latencies values used on this experiment were also used to the latency verification on 4.3.1, but this time to understand the rate in which throughput increases as simulated latencies are decreased.

The relationship between these factors is shown on Figure 4.7, demonstrating that the latency directly impacts throughput when running with 90 edge nodes. For the given scenario, our results show that throughput increases almost in the same proportion as latency decreases. For example, when latency goes from 100ms to 50ms, throughput for 90 nodes almost doubles and goes from approximately 75k Queries per Second (QPS) to approximately 150k QPS.

Figure 4.7: Latency simulation validation: Throughput obtained per node with 90 nodes.



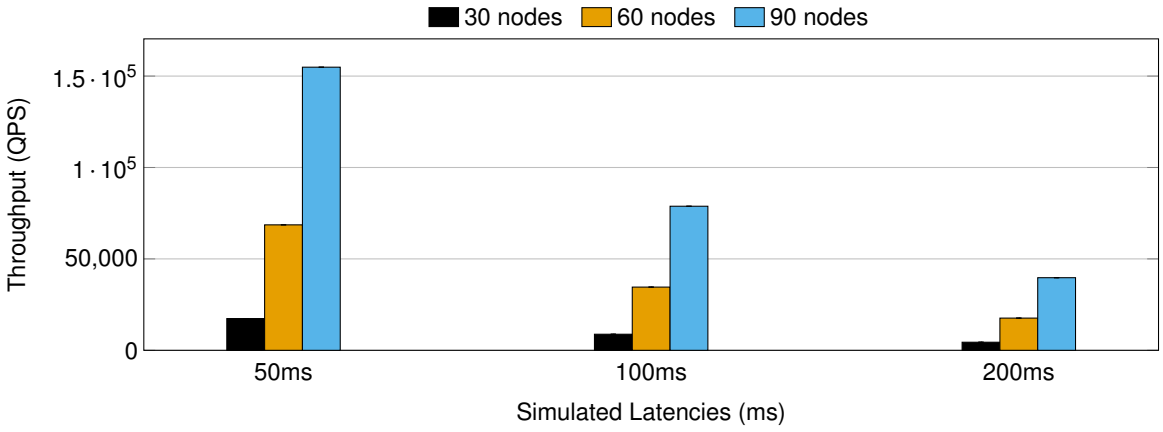
Given that it was found that throughput per node is directly proportional to latency decreases with 90 nodes, the next step was to explore how the number of nodes could impact throughput rates.

By comparing latency behaviours under the same set of nodes – sets of 30 nodes under latency constraints of 50ms, 100ms and 200ms – it was perceived that the same behaviour exists and throughput is still a factor of how many nodes are added to the analysis, as it can be seen on Figure 4.8.

However, when compared the throughput increase under distinct sets of nodes – sets of 30, 60 and 90 nodes under 50ms latency constraints – the experiments have shown that the speedup relationship is not linear, which could represent the saturation of our cloud node.

Using the analysis with 50ms as an example, the obtained speedup was 3.95, which represents the case when the number of nodes was increased from 30 to 60 nodes. On the other hand, when the number of nodes was increased from 60 nodes to 90 nodes, the speedup obtained was 2.26. From this analysis it is possible to infer that the speedup is decreasing as the number of nodes doubles. The obtained speedups for 100ms and 200ms follow the same pattern with similar speedup values.

Figure 4.8: Aggregated throughput evaluation with multiple sets of edge nodes (30 to 90 nodes).



4.3.3 Windowing evaluation

In this experiment, it is evaluated the windowing capabilities to aggregate message on edge nodes, but this time using larger sets of nodes and distinct profiles of simulated latencies. During the design of this experiment it was decided to evaluate it with the largest amount of edge nodes available (90 nodes) and the same profiles of latencies used on the previous tests, in order to facilitate the comparison. The message windows analyzed were

composed by batches of 100 to 800 messages and latency profiles of 50ms, 100ms and 200ms.

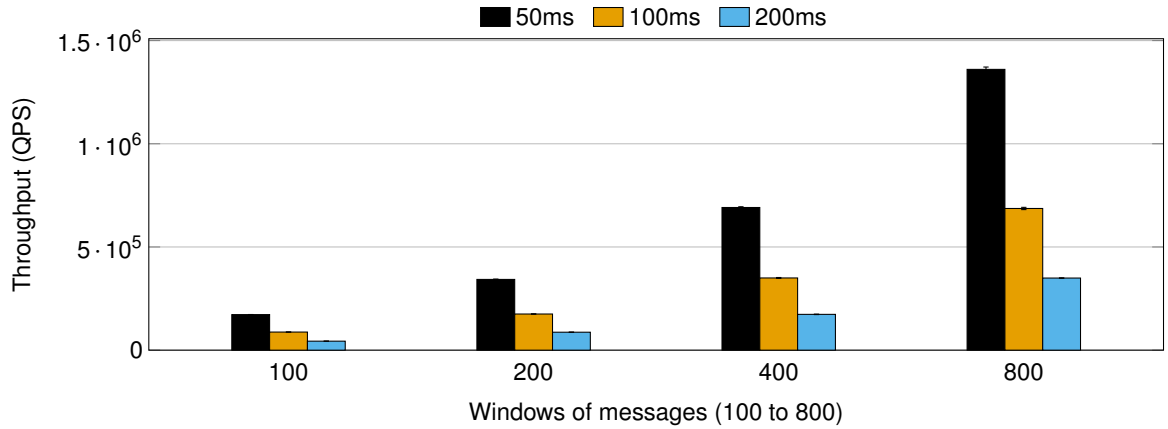
On the result graph of Figure 4.9, it can be perceived that message windowing adds another linearly increasing dimensionality to the application, which can be used in order to better tune the message throughput on latency constrained environments.

As it can seen on Figure 4.9 the tests, which were run using 90 nodes, shows that message windowing adds another linearly increasing dimensionality to the application, being able to be used in order to better tune the message throughput on latency constrained environments. This way, it is not only possible to increase throughput by adding more nodes and partitioning message pre-processing on edge nodes, but also by grouping data before sending it to the cloud nodes.

An important tuning factor, not only in relation to throughput but also to monetary costs, is the relationship between the number of nodes and the message windowing. As the throughput increases by processing messages at edge node level, it is not only possible to process more messages from sensors with a smaller number of edge nodes, but it is also possible to spend less on edge nodes hardware by better leveraging edge nodes processing capabilities.

Finally, it is important to perceive that the latency impact is similar to previous scenarios, where throughput linearly decreases as latency increases.

Figure 4.9: Windowing evaluation with simulated latencies running on 90 nodes



5 GARUAGEO: ARCHITECTURE AND IMPLEMENTATION

GaruaGeo is an extension of the Garua architecture which was the previous generation of the architecture, as it is described on Chapter 3. In the previous architecture, it was possible to evaluate physical edge nodes interconnected to cloud nodes would behave in a variety of scenarios.

However, in our previous experiments with simulated latencies, it was clear that latency penalties were severely penalizing the overall performance of the architecture. In order to properly evaluate the scalability of the platform in a real-world large scale scenarios, and improve the performance in high latency scenarios, it was decided to improve the predecessor architecture by including aggregation nodes, which are nodes placed nearby the geographical location of edge nodes.

As a way to avoid moving physical hardware across the globe to execute the evaluation procedures, we have decided to instantiate edge nodes as less powerful VMs into distinct cloud datacenters across the globe. Therefore, in this Chapter we introduce the GaruaGeo architecture, discuss its design, implementation and compare it with the Garua architecture.

The rest of this Chapter is described as follows. Section 5.1 does an overview of the architecture and its improvements over the previous generation. Section 5.2 describes the specific implementation details of the cloud layer in this generation of the platform. Section 5.3 describes the reasoning behind adding a new layer to the platform, and how it could potentially improve the architecture. Section 5.4 describes the specific implementation details of the edge layer in this generation of the platform. Section 5.5 describes the specific implementation details of the sensor layer in this generation of the platform.

The testbed implementation of the platform described in this Chapter is available online and can be found in a public git repository [1].

5.1 Architectural overview

The architectural infrastructure of the testbed application deployment can be described as a composition of four layers, as it is represented on Figure 5.1: (1) Cloud layer, which executes long-running scalable jobs that can provide more powerful machines for processing with the trade-off of greater latencies; (2) Aggregator layer, which aggregates

[1] <https://github.com/otaviocarvalho/garua>

Table 5.1: Cloud layer configuration: Virtual machine type and toolset description

Parameter	**Description**
Instance Type	Basic_A3 (4 cores, 7 GB RAM)
Operating System	Ubuntu 16.04 LTS
Golang version	1.8
GRPC version	1.3.0-dev
Protocol Buffers version	3.2.0

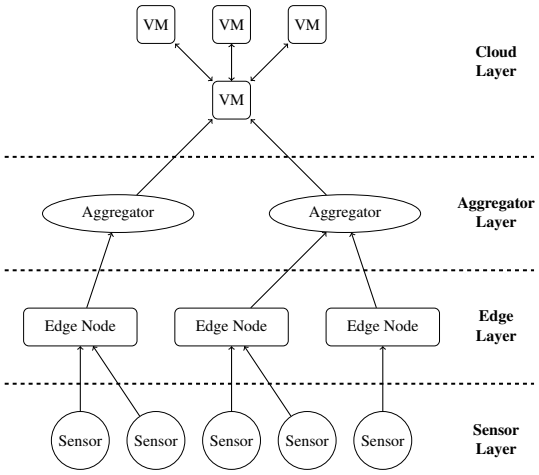
from multiple edge nodes, usually placed on the same geographic region as edge nodes, in order to aggregate all data for a given geographic region before sending data to the cloud layer, which is potentially placed in a distant geographic region; (3) Edge layer, which is composed of edge nodes that are used to pre-process and aggregating sensor data before sending to the Aggregator nodes; and the (4) Sensor layer, which is composed of the sensors that communicate directly with Edge layer nodes to receive actuation requests and provide measurements to the network.

GaruaGeo extends the ideas applied to Garua by including a new layer nearby each geographic region where edge nodes are deployed. This way, edges nodes are capable of offloading data to nearby aggregator nodes, potentially decreasing the latency penalties experienced by edge nodes. The architectural implementation used for GaruaGeo differs from Garua in several aspects. In order to distribute nodes across the globe, it was necessary to implement edge nodes as less powerful VM on Azure, instead of physical ARM processors. Furthermore, deploying the testbed architecture in global scale scenarios posed distinct challenges to build, execute and collect data from experiments.

5.2 Cloud layer

The cloud layer is composed of a set of virtual machines that execute a given algorithm in order to aggregate data received from edge nodes. Similar to the description of Garua's cloud layer on Section 3.2 of Chapter 3, GaruaGeo's cloud layer is implemented as an application running in a single node inside of a Linux VM running on Microsoft Azure. The hardware and software configuration of these VMs can be visualized on Figure 5.1. The communication pattern is still via GRPC communication framework, but it communicates with the local aggregator nodes instead of directly receiving messages from the edge nodes.

Figure 5.1: The architecture is composed by 4 layers: Cloud, Aggregator, Edge and Sensor



Furthermore, the network interconnection (between the cloud layer and the other layers) used on the evaluation on Chapter 6 is not limited by the Hybrid Fiber-Coaxial (HFC) network interconnection as it was on Figure 3.1 of Chapter 4 on Garua's architecture. In the evaluation procedure of GaruaGeo's architecture the network is limited instead by Microsoft Azure's data center interconnections across the globe (since both aggregator and edge nodes are instantiated as VMs for evaluation purposes).

5.3 Aggregator layer

The aggregator layer represents one or multiple intermediate layers of aggregation that could be potentially used to mitigate the impact of latency between data collection and the collection of global metrics into the cloud layer.

In previous evaluations, it was possible to observe that is possible to obtain significant throughput gains by aggregating data from sensors on edge nodes.

However, it was perceived that there was room for improvement if it was possible to have an operator in the architecture responsible for aggregate data for specific regions before communicating with the cloud layer. An example of this scenario is when multiple edge nodes in Japan are transferring data to the USA, each one of them paying the latency penalty of communicating with another continent. Instead, aggregators could be placed in this location and aggregate data from multiple edge nodes in Japan, before transferring it to the USA.

In our evaluation, this layer was represented as a single core machine, in order to potentially represent less powerful machines, such as Raspberry Pi's and similar ARM

processors which are well-suited for the given scenario (and that were used for the edge layer on previous evaluations).

5.4 Edge layer

The edge layer is composed of a set of nodes with transformation operators to apply over sensor measurements one-at-time. These operators can be transformations to apply over results, combinations with sets of measurements received or just simple mappings to machines on the cloud layer above in a publish-subscribe pattern implementation. The GaruaGeo's edge layer is similar from Garua's edge layer (described previously on Section 3.3 of Chapter 3 in its processing functionality and overall role in the architecture. However, GaruaGeo's edge layer is different from Garua's edge layer by being implemented as an application running in a Linux VM on Microsoft Azure. Garua's edge layer, on the other hand, was implemented and evaluated as a set of Raspberry Pi nodes based on ARM processors. The configuration used for this layer is displayed on Figure 5.3.

Furthermore, the network interconnection (between the edge layer and the other layers) used on the evaluation on Chapter 6 is not limited by the WiFi router interconnection as it was on Figure 3.2 of Chapter 4 on Garua's architecture. In the evaluation procedure of GaruaGeo's architecture the network is limited instead by Microsoft Azure's data center interconnections across the globe (since nodes on all layers are instantiated as VMs for evaluation purposes).

Table 5.2: Aggregator layer configuration: Virtual machine type and toolset description

Parameter	**Description**
Instance Type	Standard_DS2_v2 (2 cores, 7 GB RAM)
Operating System	Ubuntu 16.04 LTS
Golang version	1.8
GRPC version	1.3.0-dev
Protocol Buffers version	3.2.0

Table 5.3: Edge layer configuration: Virtual machine type and toolset description

Parameter	**Description**
Instance Type	Standard_DS1_v2 (1 cores, 3.5 GB RAM)
Operating System	Ubuntu 16.04 LTS
Golang version	1.8
GRPC version	1.3.0-dev
Protocol Buffers version	3.2.0

In this implementation the edge layer communicates directly with aggregator nodes, which are going to be placed in a (physical) nearby region. The main hypothesis being tested was that offloading to nearby aggregator nodes would improve the overall performance (in terms of number of elements processed per second) since edge nodes would pay smaller latency penalties to communicate with the layer above. Apart from that, an important hypothesis to be tested was the potential performance upside of processing messages on event windows, in order to only experience overseas network latencies (to communicate with machines geographically distant) for large groups of pre-processed messages.

The usage of a cloud platform has turned easier to scale the evaluation. However, there were several challenges in implementing and deploying large scale experiments. For example, the template format used to automate the deployment of the experiments (Azure Resource Manager templates) has a fixed limit on the number of resources which can be described in a single deployment (e.g. virtual machine instances, network interconnections, naming services, etc.). Apart from that, it was necessary to automate data collection since in the largest evaluation scenario the application was deployed in 1350 edge nodes.

5.5 Sensor layer

The GaruaGeo's sensor layer is similar from Garua's sensor layer (described previously on Section 3.4 of Chapter 3 both in terms of implementation and overall role in the architecture. From a model perspective, the hole of this layer is also to be represented by sensors that should communicate with edge nodes via hardware interconnections or lightweight wireless connection such as Bluetooth or LTE network. The main challenges of this layer on this new architecture were driven by the scale of the experiments instead of additional features.

Due to amount of potential nodes to be deployed in a single experiment, there was a large effort in terms of deployment automation and data collection. As it was described on the previous section, the larger evaluation scenarios had up to 1350 edge nodes, and it was necessary to split the sensor dataset to in such a way that a given edge node only had measurement which were from a given set of households (in order to respect the data locality aspect of the algorithm processing).

Apart from the effort of splitting sensor data across their respective edge nodes, the simulation of the edge layer on GaruaGeo also had to consider that the execution of

the experiments could be executed at the same time across edge nodes. This way, there was an additional effort to provide the right tools to not only distribute the workload, but also to trigger the execution and collect data after the execution was finished.

6 GARUAGEO: EVALUATION

The main hypothesis on the contribution of adding aggregators to the platform was that, by including a layer near to the edge nodes, the latency experienced by the edge nodes would decrease and it would be possible to decide, on aggregator nodes, when it was necessary to suffer the latency penalties to communicate with potentially distant cloud nodes.

After adding the new layer, some scenarios would be required to be validated in order to verify that our platform was able to provide its expected benefits in comparison to the previous version without aggregators.

Aggregators have the ability to receive, buffer, pre-process and group messages before sending data to the cloud layer nodes. This additional layer gives the platform the ability to answer edge layer node requests faster than the cloud layer nodes, which could potentially be placed at other countries and continents, providing slower responses due to the greater latencies implied in communication between machines in distant geographical regions.

This Chapter is organized as follows. Section 6.1, discusses the global position of the operators used in this evaluations and their latency profiles. Section 6.2, explores the additional performance impact of adding a new layer of communication into the platform. Section 6.3, evaluates the performance of a fixed number of edge nodes when distributed between groups of aggregators. Section 6.4, relates the number of aggregators per region and message windowing to the overall performance of the system. Finally, on Section 6.5, there is a comprehensive analysis of the system's edge nodes performance in a multi-region experiment.

The scripts used to automate the evaluation process of experiments described in this Chapter is available online and can be found in a public git repository [1].

6.1 Infrastructure setup and operators placement

Before the execution of the tests, it was essential to setup a minimal set of tools to help on the automation of the deployment process in multiple regions around the globe.

Since we are using Microsoft Azure, it was possible to rely on Azure Resource Manager templates to describe the infrastructure to setup. By using templates, it was pos-

[1] <https://github.com/otaviocarvalho/garua-azure-deployment>

Table 6.1: Region profiles: Description of the latency profiles between Azure regions selected for evaluation

Region	Low latency	Medium latency	High latency
westus2	•		
brazilsouth		•	
ukwest	•		
southeastasia		•	
eastasia			•
japaneast		•	
westeurope	•		
australiasoutheast			•
northeurope		•	
centralus	•		
southindia			•
canadacentral	•		
centralindia		•	
koreasouth			•
francecentral		•	

sible to describe the number of nodes, size of virtual machines, operating system version and default tools that were important to have pre-installed on all machines.

The decision of which regions to use in our analysis scenarios came from a previous latency measurement evaluation done on previous works.

In this evaluation, it was analyzed the communication between machines placed in multiple regions and one node placed on US West, which in this evaluation was the region where the master node of the cloud layer was placed. After this analysis was made, these regions were classified regions into three categories: Low latency, medium latency, and high latency. The results with regions selected and their latency profiles are displayed on Table 6.1.

6.2 Exploring the impact of adding aggregators into the infrastructure

In this testbed evaluation, aggregators nodes are placed in the same regions (datacenters) as the simulated edge nodes. Using this approach messages are buffered into a

nearby aggregator, where they can be aggregated or pre-processed and sent to a centralized node which is potentially in a distant region.

The first experiment we have made was to evaluate the impact of adding another moving piece to our infrastructure. The rational behind it was to evaluate how much it would impact the performance in a stress scenario. In Figure 6.1 we evaluate a stress scenario with a single master, before and after adding an aggregator.

From this experiment, it was possible to perceive that the performance improves for smaller groups of messages, but in general, including an aggregator does not impact the performance negatively in comparison to the previous architecture. In comparison to the previous architecture, it is a worst-case evaluation scenario, since with an aggregator there is an extra layer of communication overhead. Apart from that, in this scenario the aggregator could not benefit from batching of multiple messages since batching, in this case, is done at edge node level. The potential performance benefits of using one or multiple aggregators should be perceived only in the communication between aggregator and master nodes (where it is possible to wait or batch multiple edge node messages before communicating with nodes on the cloud layer).

6.3 Multiple aggregators in a given global region

In our previous version of the architecture, which was composed by a master node on the cloud layer and edge nodes that received sensor data, it was possible to validate that it the architecture was able to scale linearly up to 90 nodes (regarding throughput).

Figure 6.1: Aggregator stress comparison analysis: 90 nodes maximum throughput with and without an aggregation layer

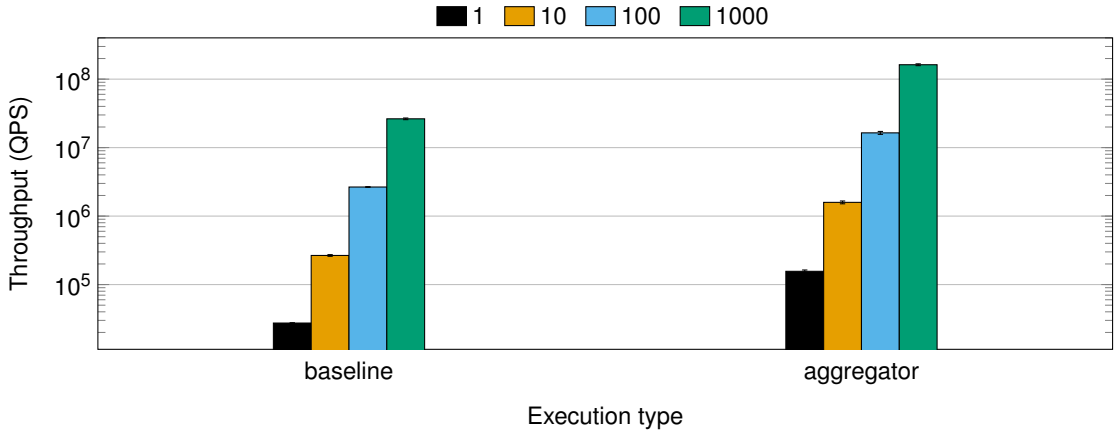
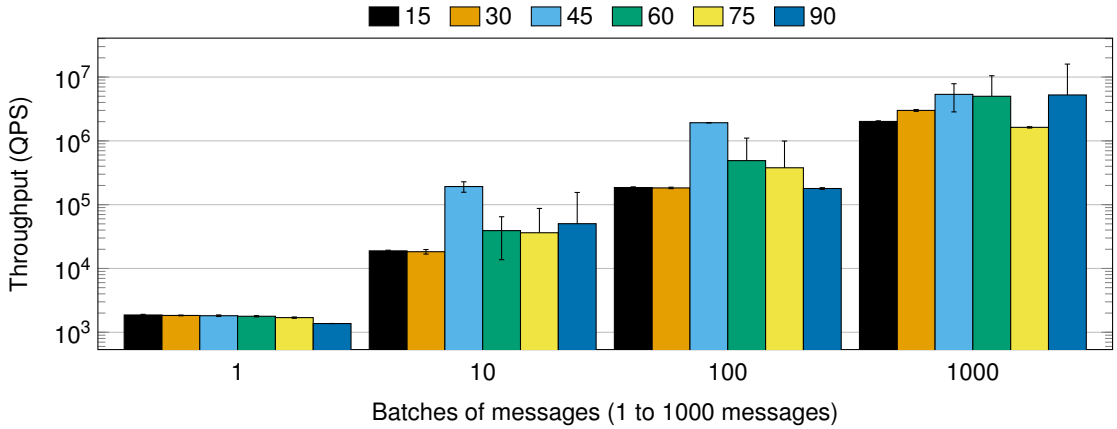


Figure 6.2: Aggregator stress evaluation: Throughtput analysis from 15 to 90 nodes



In this experiment, after the addition of our new processing layer, it was collected data to understand if this behavior has not changed.

As it can be seen on Figure 6.2, regarding batches of messages, it is still possible to perceive that the architecture can scale linearly. However, regarding the number of nodes, it is not possible to perceive any performance gains when we increase the number of nodes from 15 to 90. This behavior could be explained by the fact that our aggregators are now in the same geographic region as our edge nodes.

In our previous analysis (which considered cloud nodes and edges nodes only), cloud nodes were placed in a region and edge nodes were placed in another. In this analysis, from the standpoint of the master node a stress scenario was never achieved (the upper bound of the system regarding throughput). However, in the current scenario, the only piece of the architecture which is not in the same region is the master node. Hence, the latency experienced by our edge nodes is much lower, and the upper bound of the system is experienced by all scenarios from 15 to 90 nodes.

This analysis has shown three crucial facts: 1) The throughput from the standpoint of the edge nodes has significantly improved since aggregator nodes now handle requests in the same geographic region with lower latencies; 2) By adding aggregator nodes in the same geographic regions as the edge nodes which collect sensor data, we are now bounded by the communication between the aggregator nodes and cloud nodes (which will always be slower than the communication between edge nodes and aggregator nodes); 3) In order to evaluate the overall performance (regarding throughput and latency) of the system, it is not possible anymore to aggregate the number of requests made from edge nodes to their counterparts (cloud nodes in the previous architecture or aggregator nodes in the current),

but it is required to evaluate the performance of the communication between aggregator nodes and the cloud node.

6.4 Groups of aggregators into a single region

In this experiment, it was evaluated the impact of adding multiple aggregators into the same local region. The analysis was made using VMs on the uswest region of Azure with a fixed number of edge nodes (40 edge nodes) and a variable number of aggregators (1 to 8 aggregators).

In order to execute this analysis, the number of existent edge nodes was spread evenly between aggregators. In Figure 6.3, it was possible to verify that by adding 8 aggregators it was possible to achieve exponential increase on throughput up to 8 aggregators, where the aggregator has achieved a throughput of around 100k messages per second received from the edge nodes.

Multiple aggregators perform better probably due to the number of concurrent requests that one aggregator can answer at the same time. Given a certain amount of edge nodes, spreading their requests between multiple aggregator nodes decreases contention level in each aggregator. This way, they can answer more requests per second since the number of nodes for each aggregator is more balanced. However, introducing a more significant amount of aggregators will require that more elements pay the extra latency needed to communicate with cloud layer nodes, but these trade-offs were not explored in this scenario.

Figure 6.3: Aggregator groups evaluation: 40 edge nodes distributed between distinct groups of aggregators

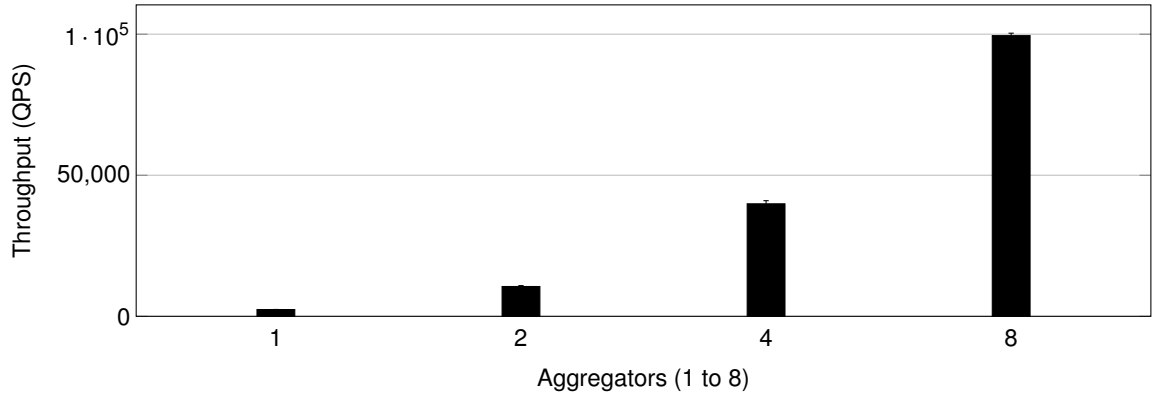
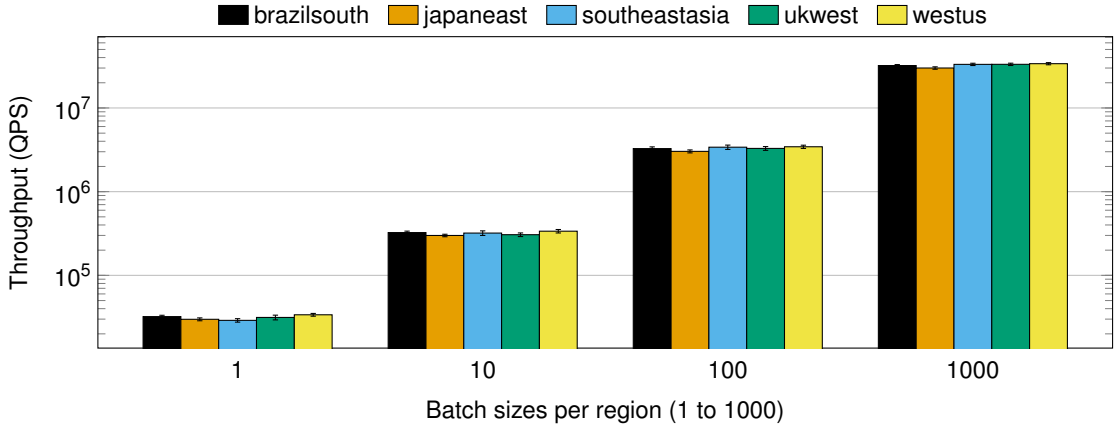


Figure 6.4: Global performance evaluation: 90 nodes per region, 5 regions and variable batch sizes (1 to 1000 messages)



6.5 Multiple region edge analysis

In this evaluation scenario, it was explored the maximum achievable performance from the standpoint of data collection on edge nodes. In order to evaluate it, we have deployed a large number of machines, into multiple geographic regions, and collected data about the aggregated data collection throughput of its edge nodes.

Each scenario evaluated on these tests relies on regions (datacenters) on Microsoft Azure across the globe. In each region, there were placed 90 Edge nodes and a single aggregator node. The analysis was made based on data collected from at least 20 executions for 5, 10 and 15 regions. In each execution scenario, each edge node sends at least 100k energy measurements to its respective aggregator nodes.

In Figure 6.6 it is possible to visualize the extension of the largest execution. For this analysis it was used: 15 regions on Azure; one Global aggregator node on the cloud

Figure 6.5: Global performance evaluation: 90 nodes per regions, 1000 messages batches and variable number of regions (5 to 15)

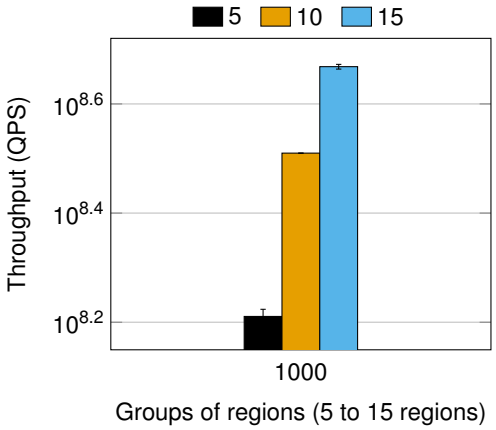
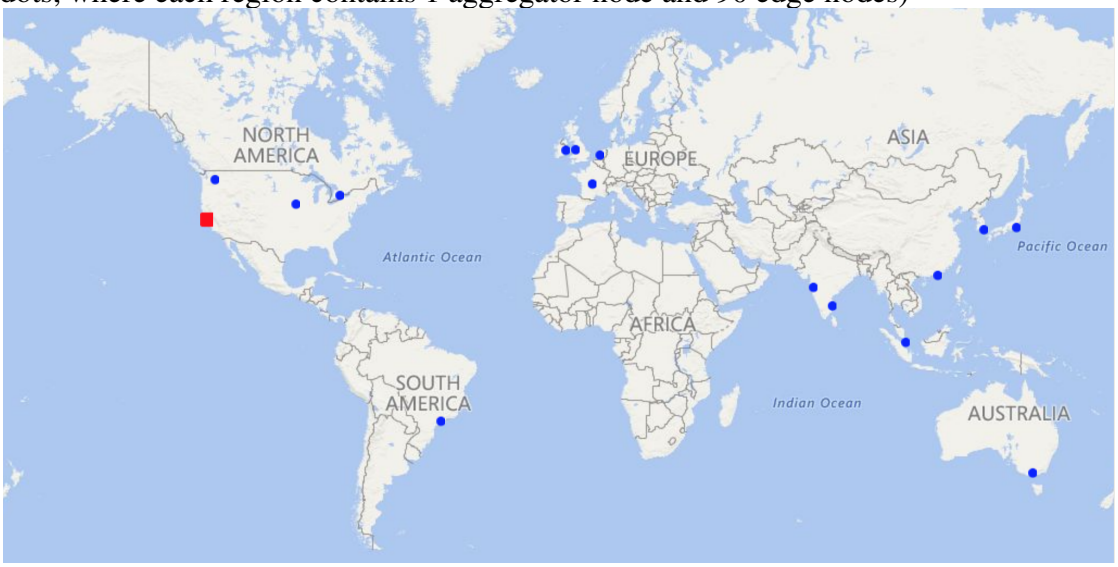


Figure 6.6: Global scale deployment: One global master (red square) and 15 regions (blue dots, where each region contains 1 aggregator node and 90 edge nodes)



layer; 15 aggregator nodes on the aggregator layer; 1350 Edge nodes, for a grand total of 1366 machines aggregating data across the globe.

Another important aspect captured by this analysis is the discrepancy in performance between distinct regions. From the standpoint of the Edge nodes, it was not possible to perceive significant performance discrepancies between regions for each one of the analyzed regions, as it is displayed in Figure 6.4.

Linear scalability is also obtained as we increase the batch factor on the number of messages which are aggregated before being sent to the aggregator nodes. Hence, it was possible to validate in this scenario that the pattern analyzed in the scenario with a single region in Figure 6.2 holds significant for multiple regions.

Finally, it is shown in Figure 6.5 a summary of the highest throughput scenarios observed, which are those that display the largest amount of messages per batch evaluated. The results show that the performance does not only increases linearly with batch sizes, but also data collection rates, which increase linearly as more regions are added globally. From these tests, up to 15 regions (which use 90 nodes and one aggregator per region), it was not possible to achieve an inflection point where data collection rates start to decrease.

7 CONCLUSION AND FUTURE WORKS

In this work, it was analyzed a model for workload distribution and data aggregation using a large scale smart grid application dataset. This work improves and extends our research on cloud computing applied to IoT data profiles (CARVALHO; ROLOFF; NAVAUX, 2017). This application was able to achieve a higher throughput by leveraging processing on edge nodes and data aggregation to reduce communication with the cloud environment.

The results show that, using the Garua architecture, it is possible to achieve throughputs of above 1 million records being processed per second with latencies up to 50ms (CARVALHO et al., 2017b) (CARVALHO et al., 2017a), using a single cloud node on Microsoft Azure and 90 Raspberry Pi based edge nodes.

After the introduction of the aggregator layer, on the GaruaGeo architecture, it was possible to improve the results by leveraging the geographical locality on the aggregation processing. The results obtained show that the implemented testbed application was able to achieve data aggregation rates of above 400 million measurements per second. These results were obtained using machines on 15 distinct geographic regions on the Microsoft Azure platform, for a total of 1366 machines in the largest evaluation scenario.

The experiments show that the impact of windowing and aggregation on edge nodes is not negligible and needs further investigation by the research community. Although it has similarities to data stream processing research, the topic is still being initially explored by researchers on the fields of the Internet of Things, Fog Computing and Edge Computing.

The future works should focus on the exploration of other scheduling, windowing and aggregation techniques for edge processing. Apart from that, one important line of research would be to explore how to evolve this testbed application and its middleware into a generic framework for applications that need to distribute processing through edge and cloud nodes. Finally, it would be important to explore other communication protocols to understand their suitability to multiple scenarios based on hybrid computations on cloud and edge environments.

REFERENCES

ABADI, D. J. et al. Aurora: A New Model and Architecture for Data Stream Management. **The VLDB Journal - The Int. Journal on Very Large Data Bases**, Springer-Verlag New York, Inc., v. 12, n. 2, p. 120–139, 2003.

ABADI, D. J. et al. The Design of the Borealis Stream Processing Engine. In: **2nd Biennial Conference on Innovative Data Systems Research (CIDR)**. [S.l.: s.n.], 2005. v. 5, n. 2005, p. 277–289.

ABDELWAHAB, S. et al. Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 3, p. 327–338, 2016.

AGBARIA, A. M.; FRIEDMAN, R. Starfish: Fault-tolerant dynamic mpi programs on clusters of workstations. In: IEEE. **High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on**. [S.l.], 1999. p. 167–176.

AI, Y.; PENG, M.; ZHANG, K. Edge computing technologies for internet of things: a primer. **Digital Communications and Networks**, Elsevier, v. 4, n. 2, p. 77–86, 2018.

ALFARES, H. K.; NAZEERUDDIN, M. Electric Load Forecasting: Literature Survey and Classification of Methods. **International Journal of Systems Science**, Taylor & Francis, v. 33, n. 1, p. 23–34, 2002.

ALI, A. B. M. S. **Smart Grids: Opportunities, Developments, and Trends**. [S.l.]: Springer, 2013.

ANALYTICSWORLD, P. **How is Predictive Analytics Different from Forecasting?** 2015. Disponível em: <http://www.predictiveanalyticsworld.com/faq.php#q3-2>.

ANAS, M. et al. Minimizing Electricity Theft Using Smart Meters in AMI. In: IEEE. **P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Seventh International Conference on**. [S.l.], 2012. p. 176–182.

ANCILLOTTI, E.; BRUNO, R.; CONTI, M. The Role of Communication Systems in Smart Grids: Architectures, Technical Solutions and Research Challenges. **Computer Communications**, Elsevier, v. 36, n. 17, p. 1665–1697, 2013.

ASSUNCAO, M. D. de; VEITH, A. da S.; BUYYA, R. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. In: . [S.l.]: Elsevier, 2018. v. 103, p. 1–17.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

ATZORI, L. et al. The Internet of Things: A Survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

BADGER, L. et al. Cloud Computing Synopsis and Recommendations. **NIST Special Publications**, v. 800, p. 146, 2011.

BAILIS, P.; KINGSBURY, K. The network is reliable. **Queue**, ACM, v. 12, n. 7, p. 20, 2014.

BANKS, A.; GUPTA, R. Mqtt version 3.1. 1. **OASIS standard**, v. 29, 2014.

BELSHE, M. et al. Hypertext transfer protocol version 2 (HTTP/2). **Internet Engineering Task Force (IETF) - RFC-7540**, 2015.

BERMAN, F. et al. **Grid computing: making the global infrastructure a reality**. [S.l.]: John Wiley and sons, 2003. v. 2.

BORMANN, C. et al. CoAP: An application protocol for billions of tiny internet nodes. **Internet Computing**, IEEE, v. 16, n. 2, p. 62–67, 2012.

BROWN, R. E. Impact of Smart Grid on Distribution System Design. In: IEEE. **Power and Energy Society General Meeting-Conversion**. [S.l.], 2008. p. 1–4.

BURROWS, M. The Chubby lock service for loosely-coupled distributed systems. In: USENIX. **Proceedings of the 7th symposium on Operating systems design and implementation**. [S.l.], 2006. p. 335–350.

BUYYA, R.; DASTJERDI, A. V. **Internet of Things: Principles and paradigms**. [S.l.]: Elsevier, 2016.

BUYYA, R. et al. High performance cluster computing: Architectures and systems (volume 1). **Prentice Hall, Upper SaddleRiver, NJ, USA**, v. 1, p. 999, 1999.

BUYYA, R. et al. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. **Future Generation computer systems**, Elsevier, v. 25, n. 6, p. 599–616, 2009.

BYLANDER, T.; ROSEN, B. A Perceptron-like Online Algorithm for Tracking the Median. In: IEEE. **Neural Networks, 1997., International Conference on**. [S.l.], 1997. v. 4, p. 2219–2224.

CARBONE, P. et al. Apache flink: Stream and batch processing in a single engine. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, IEEE Computer Society, v. 36, n. 4, 2015.

CARVALHO, O. et al. Exploring the Impact of Workload Distribution in a Hybrid Edge and Cloud Application for Smart Grids. In: **15th Workshop on Parallel and Distributed Processing (WSPPD)**. [S.l.: s.n.], 2017.

CARVALHO, O. et al. IoT Workload Distribution Impact Between Edge and Cloud Computing in a Smart Grid Application. In: SPRINGER. **Latin American High Performance Computing Conference (CARLA)**. [S.l.], 2017. p. 203–217. ISBN 978-3-319-73353-1.

CARVALHO, O.; ROLOFF, E.; NAVAUX, P. A Survey of the State-of-the-art in Event Processing. In: **11th Workshop on Parallel and Distributed Processing (WSPPD)**. [S.l.: s.n.], 2013.

CARVALHO, O.; ROLOFF, E.; NAVAUX, P. O. A distributed stream processing based architecture for iot smart grids monitoring. In: **Companion Proceedings of the 10th International Conference on Utility and Cloud Computing**. [S.l.]: ACM, 2017. (UCC '17 Companion), p. 9–14.

CARVALHO, O.; ROLOFF, E. et al. Beyond Hadoop: An Analysis of The Evolution of New Technologies for Cloud Computing. In: **Anais do, Workshop de Iniciacao Cientifica, XXV Simposio em Sistemas Computacionais, WSCAD-WIC**. [S.l.: s.n.], 2013.

CHANDRASEKARAN, S. et al. TelegraphCQ: Continuous Dataflow Processing. In: ACM. **Proc. of the 2003 ACM SIGMOD Int. Conference on Management of Data**. [S.l.], 2003. p. 668–668.

CHERVENAK, A. et al. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. **Journal of network and computer applications**, Elsevier, v. 23, n. 3, p. 187–200, 2000.

CHOWDHURY, S. A. et al. Is HTTP/2 more energy efficient than HTTP/1.1 for mobile users? **PeerJ PrePrints**, PeerJ Inc. San Francisco, USA, v. 3, p. e1280v1, 2015.

CUGOLA, G.; MARGARA, A. TESLA: A Formally Defined Event Specification Language. In: ACM. **Proc. of the Fourth ACM Int. Conf. on Distributed Event-Based Systems**. [S.l.], 2010. p. 50–61.

CUGOLA, G.; MARGARA, A. Complex Event Processing with T-REX. **Journal of Systems and Software**, Elsevier, v. 85, n. 8, p. 1709–1728, 2012.

DASTJERDI, A. V.; BUYYA, R. Fog Computing: Helping the Internet of Things Realize Its Potential. **Computer**, IEEE, v. 49, n. 8, p. 112–116, 2016.

DAVITO, B.; TAI, H.; UHLANER, R. The Smart Grid and the Promise of Demand-side Management. **McKinsey on Smart Grid**, v. 3, p. 8–44, 2010.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. In: **Symposium on Operating System Design and Implementation (OSDI)**. [S.l.: s.n.], 2004. p. 137–150.

DISCHINGER, M. et al. Characterizing residential broadband networks. In: **Internet Measurement Conference**. [S.l.: s.n.], 2007. p. 43–56.

ENERGY Department of. **U.S. Department of Energy**. 2015. Disponível em: <http://www.oe.energy.gov>.

EROL-KANTARCI, M.; MOUFTAH, H. T. Wireless Multimedia Sensor and Actor Networks for the Next Generation Power Grid. **Ad Hoc Networks**, Elsevier, v. 9, n. 4, p. 542–551, 2011.

GARFINKEL, S. **Architects of the information society: 35 years of the Laboratory for Computer Science at MIT**. [S.l.]: MIT press, 1999.

GEDAWY, H. et al. Cumulus: A distributed and flexible computing testbed for edge cloud computational offloading. In: IEEE. **Cloudification of the Internet of Things (CIoT)**. [S.l.], 2016. p. 1–6.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The Google File System. In: ACM. **Proc. of ACM Int. Conf. SIGOPS Operating Systems Review**. [S.l.], 2003. v. 37, p. 29–43.

GLIGORIĆ, N. et al. Performance evaluation of compact binary XML representation for constrained devices. In: IEEE. **Distributed Computing in Sensor Systems and Workshops, International Conference on**. [S.l.], 2011.

GOEL, U. et al. HTTP/2 Performance in Cellular Networks. In: **ACM MobiCom**. [S.l.: s.n.], 2016.

Google. **gRPC Motivation and Design Principles**. 2015. Disponível em: <http://www.grpc.io/blog/principles>.

GROUP, O. C. A. W. et al. Openfog architecture overview. **White Paper OPFWP001**, v. 216, p. 35, 2016.

GUNGOR, V. C.; LU, B.; HANCKE, G. P. Opportunities and Challenges of Wireless Sensor Networks in Smart Grid. **Industrial Electronics, IEEE Transactions on**, IEEE, v. 57, n. 10, p. 3557–3564, 2010.

GÜNGÖR, V. C. et al. Smart Grid Technologies: Communication Technologies and Standards. **Industrial informatics, IEEE transactions on**, IEEE, v. 7, n. 4, p. 529–539, 2011.

HABAK, K. et al. Femto clouds: Leveraging mobile devices to provide cloud service at the edge. In: IEEE. **Cloud Computing (CLOUD), IEEE 8th International Conference on**. [S.l.], 2015. p. 9–16.

HEINZE, T. et al. Cloud-based data stream processing. In: **Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems**. ACM, 2014. p. 238–245. Disponível em: <http://doi.acm.org/10.1145/2611286.2611309>.

KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Software: Practice and Experience**, Wiley Online Library, v. 32, n. 2, p. 135–164, 2002.

KYRIAKIDES, E.; POLYCARPOU, M. Short Term Electric Load Forecasting: A Tutorial. In: **Trends in Neural Computation**. [S.l.]: Springer, 2007. p. 391–418.

LEE, I. et al. The Internet of Things (IoT): Applications, Investments and Challenges for Enterprises. **Business Horizons**, Elsevier, v. 58, n. 4, p. 431–440, 2015.

LEE, S. et al. Correlation analysis of MQTT loss and delay according to QoS level. In: IEEE. **Information Networking (ICOIN), International Conference on**. [S.l.], 2013. p. 714–717.

LEOPOLD, C. **Parallel and Distributed Computing: A survey of Models, Paradigms and approaches**. [S.l.]: John Wiley & Sons, Inc., 2001. ISBN 0471358312.

LIU, P. et al. ParaDrop: Enabling Lightweight Multi-tenancy at the Network's Extreme Edge. In: IEEE. **Edge Computing (SEC), IEEE/ACM Symposium on**. [S.l.], 2016. p. 1–13.

LUCKHAM, D. C. et al. Specification and Analysis of System Architecture Using Rapide. **Software Engineering, IEEE Transactions on**, IEEE, v. 21, n. 4, p. 336–354, 1995.

MAAS, M. et al. Trash day: Coordinating garbage collection in distributed systems. In: **15th Workshop on Hot Topics in Operating Systems (HotOS XV)**. Kartause Ittingen, Switzerland: USENIX Association, 2015. Disponível em: <https://www.usenix.org/conference/hotos15/workshop-program/presentation/maas>.

MAHMUD, R.; KOTAGIRI, R.; BUYYA, R. Fog computing: A taxonomy, survey and future directions. In: **Internet of everything**. [S.l.]: Springer, 2018. p. 103–130.

MAO, Y. et al. A survey on mobile edge computing: The communication perspective. **IEEE Communications Surveys & Tutorials**, IEEE, v. 19, n. 4, p. 2322–2358, 2017.

MARGARA, A.; CUGOLA, G. Processing Flows of Information: From Data Stream to Complex Event Processing. In: ACM. **Proc. of the 5th ACM Int. Conf. on Distributed Event-based Systems**. [S.l.], 2011. p. 359–360.

MELL, P.; GRANCE, T. et al. The NIST definition of Cloud Computing. **National institute of standards and technology**, v. 53, n. 6, p. 50, 2011.

ORSINI, G. et al. CloudAware: A Context-Adaptive Middleware for Mobile Edge and Cloud Computing Applications. In: IEEE. **Foundations and Applications of Self* Systems, IEEE International Workshops on**. [S.l.], 2016. p. 216–221.

PADALA, P. et al. Adaptive control of virtualized resources in utility computing environments. In: ACM. **ACM SIGOPS Operating Systems Review**. [S.l.], 2007. v. 41, n. 3, p. 289–302.

PAN, J. et al. HomeCloud: An edge cloud framework and testbed for new application delivery. In: IEEE. **Telecommunications (ICT), 23rd International Conference on**. [S.l.], 2016. p. 1–6.

PAVLO, A. et al. A Comparison of Approaches to Large-Scale Data Analysis. In: ACM. **Proc. ACM SIGMOD Int. Conf. on Management of Data**. [S.l.], 2009. p. 165–178.

REUTERS. **U.S. Smart Grid to Cost Billions, Save Trillions**. 2011. Disponível em: <http://reut.rs/iorv4f>.

RICHARDSON, L.; RUBY, S. **RESTful web services**. [S.l.]: O'Reilly Media, Inc., 2008.

ROLOFF, E. et al. High Performance Computing in the Cloud: Deployment, performance and cost efficiency. In: IEEE. **4th International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.], 2012. p. 371–378.

ROLOFF, E. et al. HPC Application Performance and Cost Efficiency in the Cloud. In: **25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)**. [S.l.: s.n.], 2017. p. 473–477.

RUELLAN, H.; PEON, R. HPACK: Header Compression for HTTP/2. **Internet Engineering Task Force (IETF) - RFC-7541**, 2015.

SABER, A. Y.; VENAYAGAMOORTHY, G. K. Plug-in Vehicles and Renewable Energy Sources for Cost and Emission Reductions. **Industrial Electronics, IEEE Transactions on**, IEEE, v. 58, n. 4, p. 1229–1238, 2011.

SATYANARAYANAN, M. The Emergence of Edge Computing. **Computer**, IEEE, v. 50, n. 1, p. 30–39, 2017.

SATYANARAYANAN, M. et al. The case for vm-based cloudlets in mobile computing. **IEEE pervasive Computing**, IEEE, v. 8, n. 4, p. 14–23, 2009.

SINGH, A. et al. Rt-sane: Real time security aware scheduling on the network edge. In: ACM. **Proceedings of the10th International Conference on Utility and Cloud Computing**. [S.l.], 2017. p. 131–140.

STOJMENOVIC, I. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In: IEEE. **Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian**. [S.l.], 2014. p. 117–122.

SUN, X.; ANSARI, N. EdgeIoT: Mobile Edge Computing for the Internet of Things. **IEEE Communications Magazine**, IEEE, v. 54, n. 12, p. 22–29, 2016.

TÄRNEBERG, W.; CHANDRASEKARAN, V.; HUMPHREY, M. Experiences creating a framework for smart traffic control using aws iot. In: ACM. **Proceedings of the 9th International Conference on Utility and Cloud Computing**. [S.l.], 2016. p. 63–69.

THANGAVEL, D. et al. Performance evaluation of MQTT and CoAP via a common middleware. In: IEEE. **9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)**. [S.l.], 2014. p. 1–6.

TIRUMALA, A. et al. Iperf: The TCP/UDP bandwidth measurement tool. **http://iperf.fr**, 2005.

TOGASHI, N.; KLYUEV, V. Concurrency in Go and Java: Performance analysis. In: IEEE. **4th International Conference on Information Science and Technology (ICIST)**. [S.l.], 2014. p. 213–216.

TSADO, Y.; LUND, D.; GAMAGE, K. A. Resilient Communication for Smart Grid Ubiquitous Sensor Network: State of the Art and Prospects for Next Generation. **Computer Communications**, Elsevier, v. 71, p. 34–49, 2015.

WANG, N. et al. Enorm: A framework for edge node resource management. **IEEE Transactions on Services Computing**, IEEE, 2017.

WANG, W.; XU, Y.; KHANNA, M. A Survey on the Communication Architectures in Smart Grid. **Computer Networks**, Elsevier, v. 55, n. 15, p. 3604–3629, 2011.

WEIN, J. M. et al. **Content delivery network (CDN) content server request handling mechanism with metadata framework support**. [S.l.]: Google Patents, 2007. US Patent 7,240,100.

WERSTEIN, P.; SITU, H.; HUANG, Z. Load balancing in a cluster computer. In: IEEE. **Parallel and Distributed Computing, Applications and Technologies, 2006. PDCAT'06. Seventh International Conference on**. [S.l.], 2006. p. 569–577.

WHITE, T. **Hadoop: The Definitive Guide**. [S.l.]: O'Reilly Media, Inc., 2012.

YAN, L. et al. **The Internet of Things: from RFID to the Next-generation Pervasive Networked Systems**. [S.l.]: CRC Press, 2008.

ZAHARIA, M. et al. Discretized streams: Fault-tolerant streaming computation at scale. In: ACM. **Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles**. [S.l.], 2013. p. 423–438.

ZIEKOW, H.; JERZAK, Z. The DEBS 2014 Grand Challenge. In: **Proceedings of the 8th ACM DEBS Conference**. [S.l.: s.n.], 2014. v. 14, p. 266–269.