T.C.

ISTANBUL ALTINBAS UNIVERSITESI

INFORMATION TECHNOLOGY

INTEGRATION OF THE QUALITY OF SERVICE (QOS) IN NETWORKED CLOUD COMPUTING SYSTEM

Haitham Ahmed Yaseen

Master Thesis

Supervisor Prof. Dr. Osman Nuri UCAN

Istanbul, (2019)

INTEGRATION OF THE QUALITY OF SERVICE (QOS) IN NETWORKED CLOUD COMPUTING SYSTEM

By

Haitham Ahmed Yaseen

Information Technology

Submitted to the Graduate School of Science and Engineering

In partial fulfillment of the requirements for the degree of

Master of Science

ALTINBAS UNIVERSITY

2019

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Osman Nuri UÇAN

Supervisor

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to supervisor)

Prof. Dr. Osman Nuri UÇAN	School of Engineering and Natural Sciences, Altinbas University	_____
Prof. Dr. Oğuz BAYAT	School of Engineering and Natural Sciences, Altinbas University	_____
Asst. Prof. Dr. Adil Deniz DURU	Physical Education and Sport, Marmara University	_____

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Asst. Prof. Dr. Oğuz ATA

Head of Department

_____

Approval Date of Graduate School of Science and Engineering: ____/____/____

Prof. Dr. Oğuz BAYAT

Director

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Haitham Ahmed Yaseen

DEDICATION

I dedicate this thesis to my family, classmates and all my friends for the support and encouragement throughout my education and life. Special dedication goes to my supervisor Prof. Dr. Osman Nuri Ucan, my sisters and my mother for their support and prayers during my research work.

ACKNOWLEDGEMENTS

I might want to offer my thanks to every one of the individuals who have upheld me all through the regularly extended periods of this voyage. I might want to thank my advisor, Prof. Dr. Osman Nuri Ucan for being my compass notwithstanding when I believed I was lost and being in extraordinary part in charge of the zenith of this work. I might likewise want to thank my supervisors for their supportive exhortation, which incredibly enhanced the nature of this work. Finally, I thank this institution for hosting me during these years, securely earning its place as my home. I am very thankful to my mom and dad, whose values and education motivate me to keep asking questions; to my siblings and family for their infinitely appreciated love and to my country which, although inanimate, keeps me anchored and offers me an example of resiliency.

ABSTRACT

INTEGRATION OF THE QUALITY OF SERVICE (QOS) IN NETWORKED CLOUD COMPUTING SYSTEM

Yaseen, Haitham Ahmed
M.S., Information technology, Altınbaş University
Supervisor: Prof. Dr. Osman Nuri Ucan
Date: May /2019
Pages: 67

Cloud computing and quality of service in cloud computing is a competitive activity attracting millions of researches and practitioners worldwide. Cloud Providers allow new ideas for business to grow up easier, breaking the past barrier of having to own a big datacenter and facing its capital expenses. They are companies that sell computer resources such as storage, CPU time, network traffic, applications and other services of their datacenters as an on-demand service. A Cloud is a net-worked pool of datacenter hardware and software that is shared among many users. Cloud computing has opened doors to a new era of enterprises that harness the new Cloud enabled business. The introduction of cloud computing has revolutionized business and technology. Cloud computing has merged technology and business creating an almost indistinguishable framework. Cloud computing has utilized various techniques that have been vital in reshaping the way computers are used in business, IT, and education. Cloud computing has replaced the distributed system of using computing resources to a centralized system where resources are easily shared between user and organizations located in different geographical locations. Traditionally the resources are usually stored and managed by a third-party, but the process is usually transparent to the user. The new technology led to the introduction of various user needs such as to search the cloud and associated databases. The development of a selection system used to search the cloud such as in the case of ELECTRE IS and Skyline; this research will develop a system that will be used to manage and determine the quality of service constraints of these new systems with regards to networked cloud computing. The method applied will mimic the various selection system in JAVA and evaluate the Quality of service for multiple cloud services. The FogTorch search tool will be used for quality service management of three cloud services.

Keywords: Cloud computing, Quality of Service, Cloud services, FogTorch, MapReduce, Data-Store, Java

TABLE OF CONTENT

LIST OF TABLE

Pages

LIST OF FIGURES

Pages

LIST OF ABBREVIATIONS

SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
AWS	Amazon Web Services
GCP	Google Cloud Platform
GAE	Google App Engine
DBMS	Database Management System
RDBMS	Relational Database Management System
HDFS	Hadoop Distributed File System
CAP	Consistency, Availability and Partition Tolerance
BASE	Basically, Available, Soft state, eventually consistent
WAL	Write-Ahead Log
YCSB	Yahoo! Cloud Serving Benchmark
GFS	Google File System
QOS	Quality of Service

1. INTRODUCTION

Cloud Computing can be defined as a new style of computing that has transformed a large part of the IT industry by allowing companies to build applications delivered as services over the Internet without making big capital investments. Thus these companies do not need to have their own datacenters. The initial financial requirements of running new IT services have been significantly reduced, besides the fact that companies do not need to be concerned about usage statistics, amount of resources, peak usage or about over/under provisioning. All of these features are possible thanks to Cloud Providers, companies that sell computing resources on demand, with you-go billing system. Such an approach to selling computing as a utility similar to how water or electricity is being sold is known as [9].

Quality of service (QoS) is the allocation of resources to any application to guarantee a service level that will ensure performance, reliability, and availability of the network. In earlier cloud designs there were issues with the attainment of QoS especially in the networked cloud computing technologies. This thesis will evaluate the study will evaluate the applicability of quality of service into networked cloud computing and the effect on the performance on resource management. The quality of service in networked (distributed) cloud computing has been researched for a while; therefore, various methods have been used to improve the quality of service of this technology. First, virtualization techniques for cloud computing have been developed so that it can optimize the utilization of resources and the data access method. These techniques have also ensured maximum availability and restoration especially in case of disaster management and recovery [2]. Further, this thesis will provide a direction for the quality of service management while managing the security of data. The current concern for the quality of service has been fueled by the increase in the number of the organization using the cloud for storage, backup, and distribution of computing resources.

Cloud Computing is defined by National Institute of Standards and Technology (NIST) [57] as a" model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics:

1. On-demand self-service
2. Broad network access
3. Resource pooling

4. Rapid elasticity
5. Measured service

It is offered in three different service models:

1. Software as a Service (SaaS)
2. Platform as a Service (PaaS)
3. Infrastructure as a Service (IaaS)

And it has four deployment models:

1. Private cloud
2. Community cloud
3. Public cloud
4. Hybrid cloud

We explain these terms below.

Typically, conventional (private) datacenters have a really low CPU load compared with their total resources, they are underutilized (average server utilization varies from 5% to 20% [18]). This is because companies do not just deal with their average work load, but also need to have capacity to also handle peak loads. Hence, new Internet companies managing their own datacenters have to pay beyond the resources they are going to need. Therefore, Cloud Providers can become highly competitive in this scenario. They can offer what the company needs, neither more nor less, with the big advantage of costumers only paying for the resources they use, regardless of how much peak demand they will have to deal with.

This type of rapid provisioning of resources for scale out and rapid releasing them for scale in is called elasticity and it is possible because of the huge amount of resources Cloud Providers have. All of these resources are available through standard protocols for all kinds of different client platforms. In terms of who owns and manages the cloud, we find four types of clouds (Deployment models) defined by Jin, Hai, et al. in the" Handbook of Cloud
Computing Cloud types and services" [16]:

1. Public Cloud: This is the most common type of cloud deployment, in which services and infrastructure are available to the general public on the basis of a pay-as-you-go policy or even free of charge. Both individual users and enterprises use these services over the Internet from a third-party provider sharing computing resources with the rest of its customers. Main public cloud vendors are Amazon AWS, Google and Microsoft.

2. Private Cloud: Type of cloud where cloud infrastructure is operated solely for a single

organization. Private Clouds are used internally, which means within the organization bounds. Private clouds are the choice for these large organizations or government departments who must have total control over the data they manage, achieving a more secure environment.

3. Hybrid Cloud: It is the composition of two or more previous cloud deployment types: Private and Public Clouds. The private cloud is able to scale out resorting to external resources provided by the public cloud to handle hardware failures, peaks usage or any other kind of temporal needs. Hybrid clouds allows organizations to keep their vital data and applications within organization bounds and use public clouds to host less vital data or applications.

4. Community Cloud: This idea is derivated from the Grid Computing and Volunteer Computer paradigms. The idea is to share infrastructures between several organizations with similar requirements, allowing them to scale if needed while spreading the cost over the organizations [25].

Cloud Providers offer resources of three different types: In SaaS, the Provider offers users applications ready to use, running on their cloud infrastructure. Users do not manage anything related to the infrastructure they are using (network, servers, storage, etc). In PaaS, users are able to deploy onto the cloud or use applications created using tools provided by the Cloud Provider (programming languages, services, libraries). Once more, users have not control over the used infrastructure. Finally, in IaaS, users control a big part of the cloud infrastructure that has been given to them, being able to deploy and run arbitrary software applications, different operating systems, and to control storage, host firewalls and related networking components.

Figure 1.1: AWS, GCP and Microsoft's Azure Services: The Cloud provider's leaders [2].

1.1. CLOUD PROVIDERS

There are many cloud computing companies. Amazon and Google are two major players in this field, in which they offer Amazon Web Services (AWS) and Google Cloud Platform (GCP) cloud platforms, respectively as shown in Figure 1.1. In order to have a clearer view of the actual Cloud landscape, we proceed to explain these two big cloud solutions, among some others in the following paragraphs.

1.1.1. Amazon

Since 2002, Amazon has been providing Cloud resources under the name Amazon Web Services [5]. AWS is a Cloud Computing Platform that offers scalable computing resources over the Internet. Amazon states that their users can have these resources up and running within minutes. AWS's offerings are accessed over HTTP, using REST and SOAP protocols. The communication with AWS is done through various web tools, browser plugins and standalone applications that provide an interface to AWS. Also this can be done using the Application Programming Interface (API) Amazon provides and integrating it into users' applications.

List of Amazon Web Services (related to this Thesis) [5]

1.1.1.1. Compute

(a) Amazon Elastic Compute Cloud (EC2) is the Amazon IaaS solution. It provides scalable virtual machine based computation environments using the Xen Hypervisor [17] to manage their Amazon Machine Image (AMI) instances. This server instances can be set up and booted within minutes, scaling their capacity according to computing requirements changes through a simple web interface.

Amazon EC2 provides users with total control of their computing resources.

(b) Amazon Elastic MapReduce (EMR) allows businesses, researchers, data analysts, and developers to easily and cheaply process vast amounts of data. It uses a hosted Hadoop framework running on the web-scale infrastructure of EC2 and Amazon S3.

1.1.1.2. Storage & content delivery:

(c) Simple Storage Service (S3) provides Web Service based storage.

(d) Amazon Glacier, provides a very low cost long-term storage option (when compared to its S3 service). High redundancy and availability, but very long access latency. Ideal for archiving data.

(e) AWS Storage Gateway is an iSCSI block storage virtual appliance with cloud-based backup.

(f) Amazon Elastic Block Store (EBS) provides persistent block-level storage volumes for EC2.

1.1.1.3. Deployment

AWS Elastic Beanstalk provides quick and easy deployment and management of applications in AWS cloud. It is the main Amazon's PaaS solution. Elastic Beanstalk harnesses AWS services to complete its task successfully. Elastic Beanstalk takes care of the application's deployment: capacity provisioning, load balancing request, automatic scaling, etc. It supports Node.js, PHP, Python, Ruby, .NET and Java.

Amazon Web Services is used by many cloud companies to provide new cloud services, including RightScale providing IaaS, Heroku providing PaaS or Dropbox providing SaaS.

1.1.2. Google

Google infrastructure has been built and continues being built to work on datacenters of commodity hardware as opposed to high-end hardware due to what is called the economy of scale: costs are reduced and overall computing power is maximized. Google has managed to develop a fault-tolerant elastically scalable system to work on several datacenters of commodity hardware, letting them to offer cheap Cloud Services.With that premise, Google entered the Cloud Computing business in 2008 with the Google App Engine offering. Whilst cloud provisioning is not their core business, they have given to the world numerous and important contributions in this field. Google App Engine (GAE) is Google's PaaS solution. GAE allows users to host and run web applications and store data in Google-managed datacenters distributed around the world. It supports Java, Python, PHP and Google's Go programming languages. Users develop their applications on their local machines before uploading the applications to GAE. Then, it is GAE who takes care of the provisioning and deployment of the application uploaded on the infrastructure. Also automatic scaling is done during the life of the deployed application. Users do not need to keep an eye on the servers as this is GAE infrastructure's job. GAE software development toolkit (SDK) is provided by Google in order to allow users to develop their solutions in a simulated GAE's environment. Google also supplies APIs that can be used to integrate Google services with developer applications.

On June 2012, Google announced Google Compute Engine (GCE) at Google IO. GCE is Google's IaaS solution. GCE allows users to run large-scale CPU works on Linux virtual machines hosted on Google's infrastructure. GCE provides all resources through the Google APIs Console, a collection of Google's APIs. GCE is very similar to Amazon's EC2 solution, both provides scalable CPU capacity. After that, Google unified its cloud solutions under the name of Google Cloud Platform (GCP) [37].

Google Apps [36] is the Google's SaaS implementation launched on 2006 and entirely based on Google's own infrastructure. It is a set of several Web applications which offer an online alternative to traditional offline office software. But Google Apps is not just an online office suite, but also a solution that allows users to communicate and collaborate between their projects easily. With Google Mail users can communicate with emails, online messaging and voice or video calls. Thanks to Google Docs users can create, edit, delete their documents, spreadsheets or slides. Google Calendar is a powerful online calendar application. Google Web Pages allows for publication Web pages. Google Drive offers file storage and synchronization to users. Recently, in the Google IO 2013 meeting, Google added to their Google Apps collection Google Hangouts. It creates online video meetings with a click, allowing users to work with clients or partners in real time.

The main advantage of Google Apps is that everything is online, users do not need to install any software locally, they just need a computer with Internet connection and a Web browser to interact with Google Apps.

1.1.3. Microsoft

When analyzing this field, Microsoft has always something to offer. Windows Azure [59] is Microsoft's platform for Cloud Computing. It was announced at the Professional Developers Conference in October 2008, and commercially available since February 2010. In the beginning, Windows Azure supported only .NET development. However, now it supports many different programming languages, tools and systems, not only Microsoft-specific ones, but also third-party tools, such as Java, Ruby, PHP, Node.js and C++. Windows Azure is hosted in Microsoft-managed datacenters distributed around the world. Microsoft states that Windows Azure enables users to deploy their applications within minutes and scale in/out them to any size in a fully automated way. Windows Azure provides an API built on REST, HTTP, SOAP and JAVA that allows users to communicate with Windows Azure services easily. Microsoft also provides open-source Windows Azure client libraries for multiple programming languages. These SDKs

help users build, deploy and manage their Windows Azure applications.

Nowadays, Windows Azure Platform offers Infrastructure as a Service (IaaS) features that complement their initial offering of PaaS features. The Azure platform provides three distinct computing service models:

(a) Windows Azure Web Sites, which is their PaaS service for Web hosting. Users can create web sites in PHP, .NET, Python and Node.js and deploy them using git, FTP or TFS. Web Sites supports horizontal scalability of web sites, from shared single instances to dedicated large instances.

(b) Windows Azure Cloud Services, the traditional Microsoft PaaS service offering. Cloud Services are containers of hosted applications. Applications execute in virtual machines, also called instances, running Windows Server OS. Windows Azure itself manages the instances. Cloud Services allows to create scalable and reliable applications.

(c) Windows Azure Virtual Machines, which comprises the Microsoft IaaS solution for their public cloud. Users create Virtual machines on demand. Unlike Windows Azure Cloud Services, users have total control of their created virtual machines. The Virtual machines offering includes Windows Server images as well as Linux distributions images provided by Microsoft partners.

Windows cloud offerings are not just PaaS and IaaS solutions, but also SaaS tools. Windows Live is a Windows's SaaS which integrates search, email and a social network system. Sky-drive is the cloud hosting Microsoft SaaS model. And Microsoft SharePoint is Microsoft collaborative cloud system, which allows multiple users to work together in real time.

1.1.4. Rack Space

Behind these three big players, RackSpace [65] is one of the strongest cloud provider's companies. Powered by an offering mostly based on the OpenStack Cloud Computing platform, Rackspace is a mostly open source based alternative to Amazon, Google and Windows Azure. One of its strengths is its ability to roll out the latest Openstack features, thus continuosly improving its functionality. Rackspace offers three different Cloud Services: Cloud Servers, their IaaS solution, Cloud Sites, which is their PaaS solution, and Cloud Files for storing files in the cloud, which is their SaaS solution.As Eric Savitz states in Forbes article 1, in the last quarter of 2012, Rackspace total server amount has increased from 89.051 to 90.524, along with

an increase of their total customers from 197.635 to 205.538.

1.2. BEHIND THE BIG CLOUD PROVIDERS

Nowadays Amazon, Google, Microsoft are the biggest Cloud providers over the world, but this does not mean there are no other competitors offering excelent products in the Cloud field. Here we show the most promising cloud providers categorized by their offerings of resources: IaaS, PaaS and SaaS.

1.2.1. IaaS

Infrastructure as a Service cloud segment accounted for the majority of total market revenue in 2012 with more than half of the total public cloud market share [98]. The main responsibles for this impressive market quota are Google with its Google Cloud Platform, Amazon and Microsoft Azure, all of them deeply studied in the previous section 1.1.

1.2.2. PaaS

Among the subcategories of Cloud Computing, the Paas layer is experiencing a fast growth. According to the Market Monitor research report [14], PaaS accounted for the 24% of the total public cloud revenue in 2012 and it is expected to grow between 2012 and 2016 at a 41% compound annual growth rate (CAGR). Many companies are behind this success; in the following lines we describe some of them.

Heroku [41] is a cloud Platform as a Service (PaaS). It has been in development since 2007, what makes Heroku one of the original PaaS offerings in the world. Nowadays it is owned by Salesforce.com since 2008. At the beginning, Heroku supported only the Ruby programming language, but since then, Heroku has been adding support for more programming languages: Java, Scala, Python, Node.js, Clojure, Grails, Gradle, Play and PHP. Heroku platform is entirely based on the AWS EC2 and S3, giving it the ability to scale in/out to satisfy customers' demands. According to former Heroku CEO Byron Sebastian, Heroku was hosting more than 1.5 million applications by November 2010.

Openshift [29] is another interesting and new cloud PaaS product offered and developed by Red Hat. OpenShift is free and open-source, although now it has a paid version which adds extra support and features. The software that runs the service is called OpenShift Origin 3 and can be downloaded from Github, allowing users to change it according to their needs.OpenShift is aimed at Java, Python, PHP, Node.js, Perl and Ruby developers and, as many others PaaS,

OpenShift is based on AWS, this one specifically in Amazon EC2.

CloudBees [25] offers a Java-based PaaS to host, run and manage Java applications. It is one of the first PaaS aimed mainly at the Java developer. CloudBees supports any JVM-based programming language or framework. Jenkins Continous Integration (CI) tool is included in the Cloud-Bees PaaS.

It supports developers through the whole application life cycle directly in the cloud from Github. The Cloud-Bees service provides middleware on top of some public cloud, such as Amazon Web Services, OpenStack and VMware vSphere though customers can also run the service on private cloud infrastructure.

1.2.3. SaaS

Software as a Service layer also strikes strongly in the Cloud race. SaaS represented 25% of total cloud revenue in 2012 and it is expected to follow growing in the following years [22]. In the next paragraphs, we highlight three successful SaaS companies without forgetting Google SaaS products Google Apps and Windows SaaS offerings previously described.

Talking about successful SaaS solutions, we always find Dropbox [27]. Dropbox is a SaaS solution developed by Dropbox Inc., and launched on September 2008, that offers cloud storage and file synchronization to users. Dropbox uses Amazon S3 to store all files. In the released fact sheet of March 2012 4, Dropbox stated that they had over 50 million users, and nine months later, in November 2012, Dropbox announced that they had over 100 million users 5.

Another notable SaaS is Salesforce.com [28], which is one of the most popular SaaS Customer Relationship Management (CRM) platform. It offers on-demand CRM services for all kind of organizations. Salesforce.com presents two major products. Sales Cloud is a set of applications to manage sales, customers and other business activities more easily and efficiently; and Service Cloud, which provides organizations with a community help-desk.

Unlike Dropbox or many other SaaS solutions, Salesforce.com is built on its own infrastructure: Force.com [34], which is the Salesforce.com's PaaS product. Talking about successful SaaS solutions, we have to talk about photo Cloud-storage. Nowadays, photo storage has become one of the most consumed SaaS solutions around the world. In this field, Flickr is one of the leading solutions. Flickr is an image and video hosting developed by Ludicorp in 2004 and adquired by Yahoo in 2005.

1.3. BIG DATA

Until now, we have showed the three most popular cloud paradigms: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) and how Google, Amazon, Microsoft and Rackspace among other companies offer their Cloud Services. Now it is time to move to one problem that has born with the outbreak of the Cloud Computing.

Thanks to the features that Cloud Computing has brought with itself (pay-on-demand, elasticity, etc.), lot of new applications have seen the light. Thanks to the new cloud business, they are now economically viable and before not. More and more novel applications harness the Cloud Computing paradigm every day, which means a never seen increase in the amount of generated as well as consumed data, called Big Data as shown in Figure 1.2. Thus, scalable Database Management Systems (DBMS) have become a fundamental and critical part of cloud infrastructures [3].

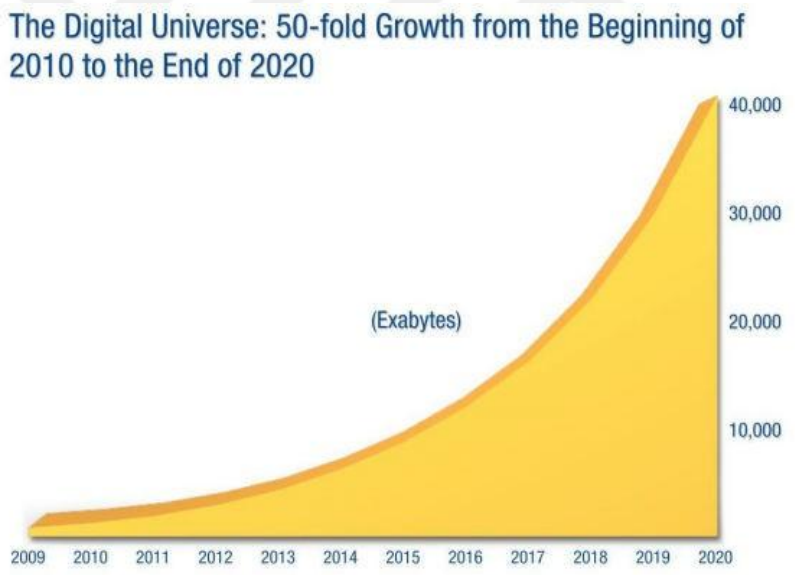

Figure 1.2: Growth of data from the beginning of 2010 to 2020 [11].

Google has been one of the first companies that has addressed the Big Data problem, which is handling large amounts of data. Google has designed its own MapReduce programming paradigm [26], allowing them to do scalable distributed batch processing of large amounts of data: Web request logs, crawled documents, etc. Related to this field, Google designed and

implemented their own distributed file system to be used in combination with their MapReduce framework, which was called Google File System (GFS) [32]. These are systems designed to run on a large cluster of commodity machines and are highly scalable, but both implementations remain private to Google.

After the publication of MapReduce and the GFS, Apache Hadoop [47] entered the scene as the open-source alternative to Google MapReduce and Google File System. Apache Hadoop is an Apache project that includes implementations of a distributed file system, namely Hadoop Distributed File System (HDFS) [23, 24], and Hadoop MapReduce [37], both inspired by Google projects. Hadoop was initially created in 2004 by Doug Cutting (and named after his son's toy elephant). In January 2008, Hadoop became a top-level Apache Software Foundation project. Nowadays it has many contributors, both academic and commercial (Yahoo being the largest commercial contributor), and continues growing.

Table 1.1: The resource utilization and assessment for data acquisition.

Heuristic Rar	Consumed RA	Consumed HI
70.106225	0.2	0.0625
80.1214	0.2	0.0625
50.075875	0.2	0.0625
40.0607	0.2	0.0625
60.09105	0.316666667	0.09375
30.045525	0.316666667	0.09375
95.163	0.316666667	0.09375
85.6467	0.316666667	0.09375
20.87635	0.483333333	0.15625
10.438175	0.483333333	0.15625

The field of distributed systems for Cloud Computing continued expanding, and as a logical next step, researchers needed a database for the applications to store the massive amount of data they were generating during the process of heuristic rank evaluation and consumed RAM with HDD as shown in Table 1.1. Until then, the traditional and massive used Relational Databases Management Systems (RDBMS) have offered a simple and good solution according to the needs of the moment, but with the arrival of Web applications with massive numbers of users, the requirements of storage database systems for this new generation of applications have

changed. Traditional databases did not offer a suitable and feasible storage solution to the new massive amount of data. In 2006, Google published a paper talking about Big-Table [17]. Big-Table is a distributed storage system built on GFS for managing petabytes of structured data across thousands of commodity servers. Between its goals, we can find wide applicability, high performance, high scalability and high availability. To achieve such goals, Big-Table uses a simple data model that supports dynamic control over data layout and format. Developers do not have to define a schema to store structured data, giving them a high flexibility when building applications. Albeit, such a simple data store brings with it a lack of characteristics. Lacking in particular are ACID transactions, Join operations and accessing through a natural query language like SQL that RDBMS's offer.

As a response to Big-Table we find Apache QOS [36], which is an open source project, modeled after Google's Big-Table and written in Java. Developed as part of Apache Software Foundation's Apache Hadoop project, it became an Apache top-level project on 2010. QOS uses HDFS as the underlying storage system for the created tables and the Apache ZooKeeper as a distributed coordination service, similar to the use of Chubby [15] in Big-Table. QOS features are similar to Big-Table features; its implementation is very close to Big-Table implementation with same properties. Main differences lie in implementation details. For example, how the memory is mapped or how the Garbage Collector works [42].

QOS, Big-Table and many others Cloud data stores are included in the group of so-called No-SQL databases. All of them are distributed storage systems mainly designed to offer a really good performance when dealing with Big Data, contrary to what traditional SQL databases do.

2. BACKGROUND

In the following section we focus on the Column-oriented datastores to increase the quality of service for cloud based research, explained before, and in particular in the QOS solution, one of the most popular and open-source Column-oriented datastores

2.1. COLUMN-ORIENTED DATA STORES

When referring to Column-Oriented datastores, also called Extensible Record Stores, where Big-Table is the pioneer. Big-Table and many other datastores of this type present a simple and flexible data model that can be extended at any moment. Some famous Column-Oriented datastores are Apache QOS [16], Hyper-Table [44], Apache Accumulo [19], and Apache Cassandra [18] in addition to many others.

Among all these solutions, QOS [31] is likely the most popular open-source Column-Oriented data-store and is the one that we are going to use for our experiments.

In the next section, we will describe QOS and how it works in order to fully understand this Thesis.

2.2. QOS (QUALITY OF SERVICE)

QOS in clouds for instance is an important Apache Hadoop-based sequencing of clouds, which, as we stated before, is modeled on Google's Big-Table database. QOS can be characterized as a distributed, fault tolerant scalable database built on top of the HDFS file system. It belongs to the group of column-oriented datastores and uses Apache Zoo-Keeper for management of partial failures.

Below, all must-known aspects of QOS are presented: QOS data model, storage, architecture, write/read/delete paths and the client API.

The architecture of cloud computing entails the system design of the various software components that are contained within the delivery of the cloud computing such as the cloud computing components that collaborate to accomplish task or messages over a queue. The architecture of cloud defines the components and the specific subcomponents that are required for the operation of the cloud network [8] [6]. Technically the cloud components are made of the front-end which contain the application either desktop or mobile and the backend, i.e., the storage, service and the server [4]. Additional, the cloud system and network such as the intranet, internet and inter-cloud are vital in supporting the access of the computing resources.

2.2.1. Data Model

QOS stores data items, those are key/value pairs. The keys are multidimensional. Each single value is indexed by a row key, column key and a timestamp 1. Row keys are unique and allow the user to address all columns in one logical row. The column key is the combination of a column family and a column qualifier. Column families are the main unit of separation within a table. The last part is the timestamp which is used for versioning the data. Timestamp is usually automatically generated by the corresponding Region Server but it can be specified by the user.

Summarizing, a key is commonly represented as the tuple (row, column, timestamp), which addresses a specified value:(Heuristic Rank, Consumed RAM, Consumed HDD) -value where column equals (column family, column qualifier), timestamp is a 64-bit integer and row, column family, column qualifier and value are un-interpreted array strings, since in QOS everything is store as bytes as shown in Table 2.1.

Table 2.1: Example QOS table given by

	Heuristic Rank	**Consumed RAM**	**Consumed HDD**
1	54.75732778	0.2	0.0625
2	59.73526667	0.2	0.0625
3	44.80145	0.2	0.0625
4	39.82351111	0.2	0.0625
5	94.14406111	0.316666667	0.09375
6	49.77938889	0.316666667	0.09375
7	99.122	0.316666667	0.09375
8	34.84557222	0.316666667	0.09375
9	76.1528	0.316666667	0.09375
10	71.39325	0.316666667	0.09375
11	29.86763333	0.366666667	0.125
12	24.88969444	0.366666667	0.125
13	20.35075556	0.483333333	0.15625
14	70.34964444	0.483333333	0.15625
15	65.26195556	0.483333333	0.15625
16	15.26306667	0.483333333	0.15625
17	10.17537778	0.483333333	0.15625
18	5.087688889	0.483333333	0.15625
19	90.04412222	0.483333333	0.15625
20	84.95643333	0.483333333	0.15625

A cell is a set of data items with a common row and column key (remember that column key stands for (column family: column qualifier)), being the cell key = (row, column). Each data

item in a cell is called a version of that cell. QOS supports multiple versions of cells. Each version of a cell is stored as a separated cell, next to other versions of that cell. Versions of a cell are sorted descending by the timestamp so that users will see the newest value first while reading it. Another QOS table feature is that it does not store NULL values as RDBMSs do. Files storing the data only contain data explicitly set.

A table is organized by grouping cells into rows. These cells are sorted lexicographically by row key first and then by column key, so row keys lexicographically close will be stored near to each other. The sorting allows the table to be partitioned into the denominated regions, which hold exclusive ranges of row keys. The regions of a table are distributed between different nodes.

2.2.2. Storage

The real view of tables differs from the conceptual view explained before. Physically, tables are stored on a per-column family basis, which means that all column family members are stored together in files called QOS-Files/StoreFiles. Such an approach brings advantages, one of which is that they can be compressed together. Column families must be declared while creating the table, whereas column qualifiers can be added to column families at any time.

As written before, the QOS storage files are called QOS-Files. They are based on Hadoop's TFile class and mimic the QoS-Table format used in Google's Big-Table system. What is stored inside them is called Key-Value instances. They are the physical view of the conceptual cells. The whole cell, with its structured data (row length, key type, etc.) is what a Key-Value object is. Figure 2.1 shows a conceptual depiction of the Key-Value format.

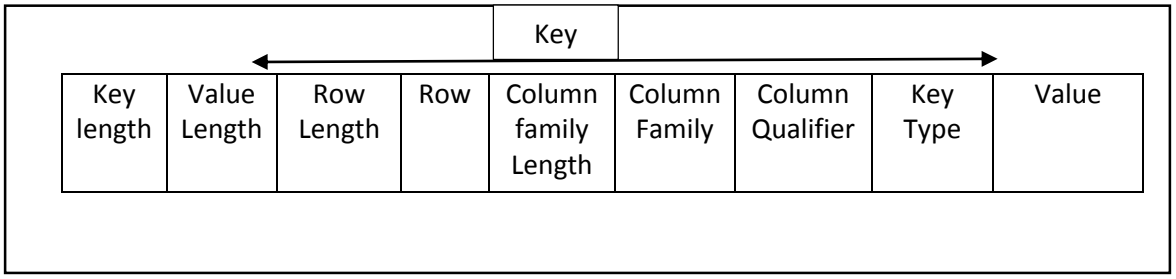

Figure 2.1: The Key-Value format, extracted from QOS: The definitive guide [31]

2.2.3. Architecture

QOS consists of three layers: the client, the server and the storage layers. The server layer consists of a master server and many region servers; the client has the library to communicate

with the existing QOS installation; the storage layer is composed of a file system and a coordination service. Hadoop Distributed File System (HDFS) is the most used and tested file system to work with QOS. As the coordination service, Cloud is the one QOS uses for its distributed coordination service. In the following section each component is explained as well as some QOS's features as shown in Figure 2.2.

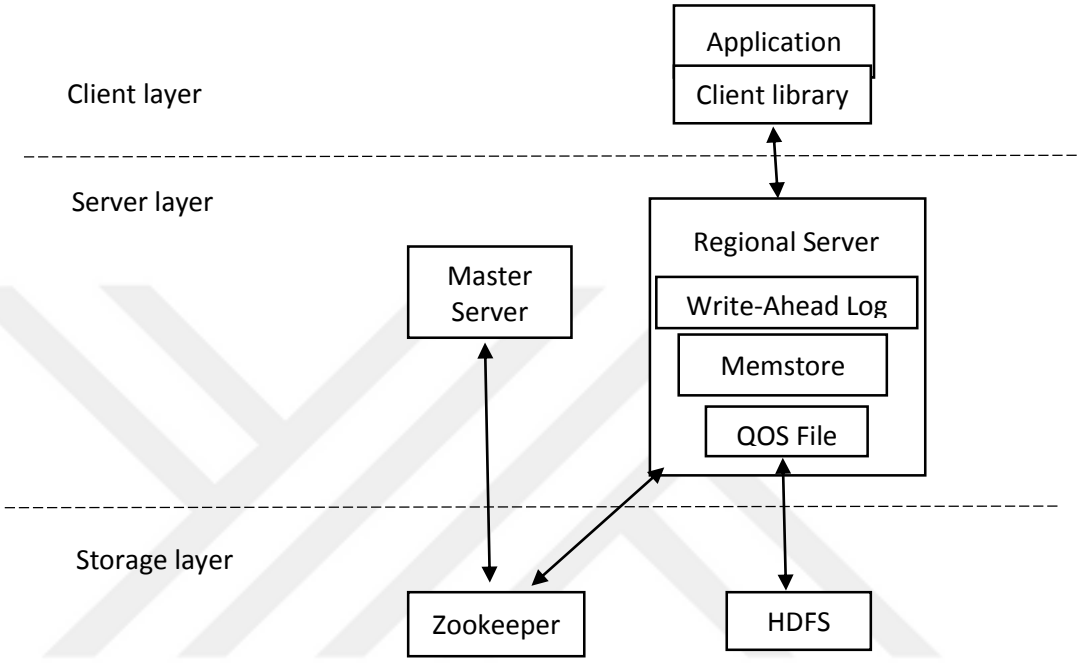

Figure 2.2: QOS architecture overview pre-defined by researchers [39]

2.2.3.1. Storage layer

The storage layer is composed of a chosen file system for the QOS cluster and a coordination service:

- File system: HDFS [12, 57] is the default file system when deploying an QOS cluster, but optionally it can be replaced by any other file system. For QOS, HDFS is the primary option as it is scalable, fail safe, has automatic replication and is built to run on commodity hardware. It fits with the needs of a distributed system.

- Coordination service: Apache Zookeeper [43, 48, and 57] is an open source project and a part of the Apache Software Foundation. Zookeeper is a highly-reliable distributed coordination service, comparable to Chubby [15], owned by Google and used for Big-Table. Its aim is to offer a file-system-like access to clients with directories and files (nodes) that are used to store data, register services or watch for updates in a simple

interface. In QOS, every region server creates a node in Zookeeper, which will be used by the master to discover them. QOS uses Zookeeper to choose the unique master and to store"-ROOT"'s address too. Master server and region servers communicate with ZooKeeper to keep track of the current situation of the regions and region servers.

2.2.3.2. Server layer

Server layer is compound of two parts:

Master: The master server does not store any actual data and is not part of the retrieval path. It is responsible for assigning regions to region servers, handling load balancing of regions across region servers, unloading busy servers and moving regions to servers that are freer, and performing garbage collection of files. Since it never stores or provides data to clients or region servers, is usually slightly loaded. Furthermore, it takes care of all administrative operations such as schema changes or creation of tables or column families.

If the master server goes down, the cluster can still work as the QOS client does not talk with it directly. Nevertheless, master should be restarted as soon as possible.

Region: As Lars George states in his book QOS: The definitive guide [31], a region is the basic unit of scalability and load balancing in QOS and is responsible for storing the actual data. Inside it, we can see contiguous ranges of rows stored together. Clients communicate directly with region servers to run all retrieve / write data operations. Each region is served by only one region server, although each region server can store multiple regions. The region servers also split regions into two pieces at the row key which is in the middle of the whole region once they have exceeded the maximum allowed region size.

2.2.3.3 Client layer

Client needs to be able to find the region server which has a specific key. In order to achieve that, client communicates with the Zookeeper server and then retrieves the location of a table called"-ROOT-" table from there. This table stores information about all regions in the". META." table, which is another table storing where regions are and its row key ranges. So client reads the. "META" table and receives the exact address of the correct region handling a determined key or key range. Thanks to this three level lookup, the client is able to find the correct region server and perform its operations. For optimizing the process, client caches region locations because once the user table region is known, it can be accessed straightforwardly without the three level lookup.

Only if the region server informs the client that it is no longer serving a region, does the client a new three level lookup.

2.2.4. Write

This subsection clarifies how writes and related stuff are done in as such a complex system as QOS is.Write requests are served by Region Servers. These components have three main elements: Write-Ahead Log (WAL), Memstore and QOS-Files. The WAL acts as a log for all modifications done to data in that exact region, guaranteeing atomicity and durability. The WAL allows regions to recover from server failures. It contains all previous modifications and can be used to recover from and to replay the data, achieving the last known stable stage. Memstore is an in-memory buffer that contains recently updated data items sorted by key. It works similar to how write buffers work on microprocessors, MemStore buffers writings, thus reducing write latencies. There is one MemStore for each column family of a table in the region. QOS-Files stores actual QOS's data.

QOS stores data to disk in a way similar to how a Log-Structured Merge Tree works [61]. First, when new data arrives at the region server it is put on the Write-Ahead Log (WAL). If the write to the WAL is achieved, the region server writes the data into the memory store. Region continues writing data to the Memstore until a configured maximum size. Once that threshold is reached, Memstore flushes the data to disk. When flushing, multiple QOS-Files are created, one per column family, which can affect negatively to QOS performance. Massive amounts of tiny files stand for more seeks and thus, higher latencies while reading. Due to that fact, QOS monitors the number and size of these files and does compactions from time to time. A compaction is the process of merging multiple QOS-Files together, and there are two types:

- Minor compaction: It is responsible for rewriting the last few files into a larger one. It is triggered when certain properties and ratios are reached.
- Major compaction: It merges all files of a region into one single file. It is usually triggered every twenty-four hours or even more due to heavy load. Sometimes, it is recommended to disable major compactions and manage them manually as they incur a high performance penalty due to rewriting all of the database contents.

QOS-File files store an index at the end of the file to locate blocks within the QOS-File. The index is loaded into memory when the QOS-File is opened allowing look-ups to be performed with a single disk seek. For a more complete overview of how these files are designed refer to [44].

2.2.5. Read for QOS

This paragraph explains reads in QOS. First of all, when a Region Server receives a Get request, it checks whether the desired row is in the MemStore or not. If not, it starts to search through the QOS-Files starting from the newest working towards older QOS-Files, which means to take a look through the disk contents because the data could be spread over multiple files.

2.2.6. Delete for QOS

Here we explain how data is deleted from a QOS cluster. We must be clear that rows are never directly erased from QOS. When a Region Server receives a Delete request, it looks for the row and writes a delete marker to it. Whenever a Get request tries to access a row that has been previously deleted, it will find the delete marker and data will not be returned. During the next major compaction, rows with the delete marker will finally be deleted.

2.2.7. QOS API

QOS provides a powerful client API written in Java as QOS does. It provides from basic operations to expensive ones. The initial set of basic operations are called CRUD operations which stands for “Create, Read, Update and Delete”.

- Put Operation for QOS of Cloud

There are two groups of Put operations, first one works on single rows and the other works on a list of rows, both allow users to store data into QOS’s tables in a transparently manner. Defined as as in equation 2.1:

$$\text{put([List]Put put[s])} \tag{2.1}$$

The user needs to supply a Put object or a list of them. These Put objects are created with the Put constructor as in equation 2.2:

Put (byte [] row) (2.2)

A row key is supplied in order to create the Put instance, and once the user has created it, he/she can start to add values to the specified Put instance with the Put.add(value) method. The user can also supply a version number for a given key/value pair (timestamp), but if it is not specified, QOS gives to it a version number created from the current time of the Region Server responsible for that given row.

When providing the value for the Put.add() method, as opposed to what happens with column qualifiers that can be whatever the user needs, an existing column family need to be given. Column families are usually defined when creating tables, but the user can always add new families calling to an expensive operation. It is because of that, QOS heavily recommends users to use fixed column families, although they can be altered. Unlike column families, new column qualifiers, version numbers and row keys can be provided on-the-fly within Put operations with no extra cost.

The QOS API allows to use a built-in client-side write buffer that collects sets of Put operations before sending them as a unique RPC connection to the corresponding server. Hence, less RPC connections are needed resulting in an increase of the overall performance. QOS is smart enough to group and sort Puts by Region Server. If write buffer is not used, any time a Put operation is completed, QOS's API will submit it to the right Region Server.

- Atomic Compare-and-Set Operation for QOS of Cloud

As a variation to the Put calls, the user can use Check and Put operation, defined as:

checkAndPut(byte[] row, byte[] family, byte[] qualifier,byte[] value, Put put) (2.3)

This method allows users to issue Puts with a checking point. Only if the check step is successfully completed, the put operation is performed, everything as an atomic operation. It is really useful when dealing with data that needs previous values or similar stuff as showed in equation 2.3.

All rows inside the Put object must be equal to the given row. The user cannot use this operation to check different row keys. Otherwise, Check and Put operation will fail.

- Get Operation for QOS of Cloud

Like in QOS Put method, there are two groups of Get operations as shown in equation 2.4, the ones that work on a single row and the others that operate with multiple rows. Both allows the user to retrieve data stored in QOS's tables.

Get operation is defined as:

$$get([List]Get\ \ get[s]) \tag{2.4}$$

The user needs to supply a Get object. These objects are created with the Get constructor like in Put operations as in equation 2.5:

$$Get(byte[]row) \tag{2.5}$$

In analogy with Put constructor, the user provides a row key to the Get method in order to get a Get instance. Get operation is bounded to a specified row, but can retrieve any data stored in it, from one value to all columns with its values.

The user can add parameters to the Get object in order to narrow down the search. They will act as filters. If the user wants everything of a row, no filters are used, but if the user only wants a column family, the Get.addFamily() method must be used. Same thing happens if the user only wants a column qualifier from a column family (Get.addColumn()), or the row with a known timestamp (Get.setTimeStamp()). Lastly, there are methods, acting as filters, that allow users to specify how many versions want to be retrieved (Get.setMaxVersions()) and many others.

Like in Put method, the user can retrieve a list of Gets, instead of only one row. The main difference is that the user issues a list of Gets, instead of one Get object. The result will be an array of Results, one for each Get instance.

- Delete Operation for QOS of Cloud

QOS client API provides a method to delete data from its tables. Delete method is defined as:

$$delete([List]Delete\ delete[s]) \tag{2.6}$$

Once more, the user is able to delete one row by one row or a list of them, the difference is the type of parameter: a Delete instance or a list of them as shown in equation 2.6.

As with Get and Put calls, the user has to create a Delete instance and then adds details (filters) about the data he/she wants to remove, the constructor is:

Delete(byte[] row) (2.7)

A row is provided. Subsequently, what user wants to be removed is added to it using different methods. Most important ones are Delete.deleteFamily() method as defined in equation 2.7, used to remove an entire column family, including all its columns, and Delete.deleteColumns() method, which operates on one column of a given column family, deleting all versions contained or just the cells matching the timestamp if it is provided. There are other types of Delete methods, but less used.

- Atomic Compare-and-delete Operation for QOS of Cloud

As a variation to the Delete call, the user can use Check and Delete operation. It is defined as:

checkAndDelete(byte[] row, byte[] family, byte[] qualifier, byte[] value, Del del) (2.8)

It works as a Delete operation but adding a previous step in which a specify row key, column family, column qualifier and value are checked before deleting the desired row.Users can only check and delete on the same row. If the row key differs from the one pointed by the Delete instance, the Check and Delete operation will fail as defined in equation 2.8.

Useful hint. Row Locks for Row mutations:

Previous operations: Put, Delete and Check and Delete are executed in such a way that they guarantee row level atomicity. They are executed entirely. A row lock is provided by the corresponding Region Server, protecting the row from other users trying to access it. Not from users trying to read it, only for those submitting row mutations operations.

- Scan Operation for QOS of Cloud

Scan operation allows the user to scan a range of data, from one row to a determinate stop row, taking advantage of the underlying sequential storage layout QOS has. The scan returns every row between the chosen range of rows. This operation is not executed atomically; it can be partially executed. Hence, returned data can be outdated if there has been a write operation during the scan.

Scan operation is really similar to Get method and it works as a iterator, it means that user has a Scan instance and he/she must iterate over it to get all the results.

Scan method is defined as in equation 2.9:

getScanner(Scan scan) getScanner(byte[]Z family) getScanner(byte[] family, byte[] qualifier) (2.9)

Each of them narrow the read data, from a general Scan to one that only returns values from a Column Family and inside it, from a Column Qualifier. As in Put or Get methods, Scan object is created with its constructor,
Which is:

$$\text{Scan(byte[] startRow, byte[] stopRow)} \tag{2.10}$$

It returns a Scan instance. Start row is mandatory and always inclusive, while stop row is not and is exclusive ([startRow, stopRow)). The user can submit filters as well.
Once the user has created the Scan instance, the user can narrow the scanned data using Scan.addFamily() method or even a more restricted one which is, Scan.addColumn(). There are more mechanisms that allows users to set timestamps or time ranges defined in equation 2.10.
It is important to notice that the Scan operation do not cover all results within a single RPC connection, but instead it returns row by row since they can be very large. In order to iterate over the results, Scan presents next() and next(int nbRows) methods. Using them, each call will be traduce to a RPC for every row.
Summarizing, QOS Scan method can be seen as a lot of Gets operations, and indeed, it is what it does.

2.2.8. QOS properties

In this section we discuss the CAP theorem and the BASE and the ACID model for QOS.QOS is called a CP type system since it supports Consistency and Partition Tolerance facets of the CAP model. QOS offers Partition Tolerance as it survives message loss due to server failures, network problems, etc. If a region server collapses, other nodes take over the tasks which comprised the region by replaying its commit log and memStores. Regarding consistency, QOS does trade some availability to achieve a stronger level of consistency. After a write completes, the next read will see the lastest value because at any given time only one region server is responsible for that key. Availability is given up because if a region server dies, its data will be unavailable until another region server comes up and picks up died regions.
QOS is not an ACID compliant database, but it guarantees some ACID properties, such as atomicity within a single row but not across multiple rows, or strong consistency thanks to the design of regions only being hosted in one region server at any one time, which creates only one

responsible for serving a given data and also thanks to the Multi-Version Concurrency Control (MVCC) used by QOS to manage concurrent access to the database (no locks are used as in old ACID SQL databases, instead timestamps offers all QOS needs to get all of ACID).

2.3. HDFS

HDFS is the distributed file system of the Apache Hadoop open-source framework that provides high-throughput access to the stored data for QOS of Cloud. It is scalable, fail safe, offers automatic replication and is built to run on a set of commodity hardware.

HDFS consists of a master node called Name-Node, and slave nodes called Data-Nodes. HDFS splits the data into equal size blocks and spread them across all available Data-Nodes in the cluster. Each block is replicated three times by default with at least one replica within the same node. The Name-Node as shown in Figure 2.3. The Name-Node retains all metadata about blocks and replicas.

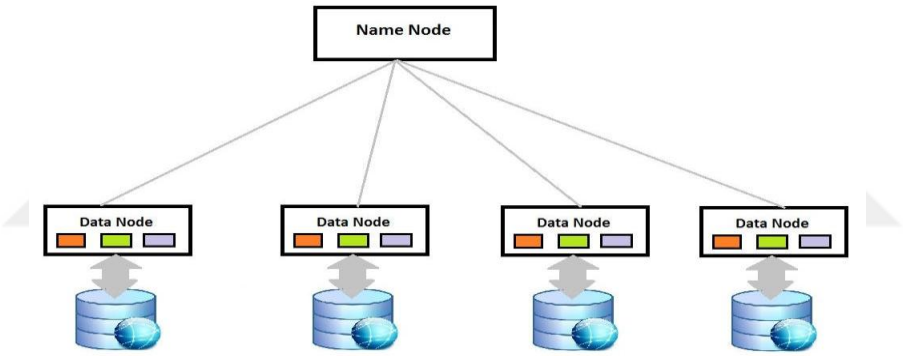

Figure 2.3: HDFS overview for increased Quality of Service in Cloud [26].

2.4. MAP REDUCE

Map Reduce programming model allows to abstract from the complexity of writing concurrent programs. It is based on two functions: map() and reduce(). The framework takes care of invoking them in the correct order on the input data and scheduling parallel execution of these two functions across any number of computation nodes. The user just needs to write the map and the reduce functions.

Map takes an input as key/value pair (<k1, v1>), and emits a number of intermediate key/value pairs as its output (<k2, v2>). The MapReduce job group together all the values which have the same intermediate key and passes them to the Reduce function (<k2, list(v2)>).

Reduce accepts intermediate key and a set of values for that key as input(<k2, list(v2)>). For

each pair the reduce function outputs a final key/value pair (<k2, v3>). Map and reduce functions can be summarized in the following equations:

$$map(<k1, v1>) \rightarrow list(<k2, v2>)$$

$$reduce(<k2, list(v2)>) \rightarrow <k2, v3>$$

The MapReduce model fits with many large-scale data problems and can be efficiently implemented to support problems which input data is hundreds or thousands of megabytes. Due to the large size of the data, is more efficient and easier to move the computation instead the data. Therefore, in a MapReduce framework, data is split into blocks and distributed across many nodes in a cluster. Then, it is the MapReduce framework who takes advantage of data locality by moving computation to data rather than sending data to the nodes. That is, MapReduce schedules Map tasks close to the data block on which they will work, so it can be read and processed very fast for each node in parallel. This is the principal factor in MapReduce's performance.

2.4.1. Hadoop MapReduce For QOS Operations

Hadoop MapReduce [57] [61] is the popular Apache open-source Java implementation of the MapReduce model. It is usually used with HDFS as the underlying file system. Hadoop MapReduce consists of a single master node called JobTracker and worker nodes called Task-Trackers. When using HDFS with Hadoop MapReduce, Task-Trackers live on the same nodes where HDFS Data-Nodes live.

When a Hadoop MapReduce job is submitted, it is divided into map tasks, also known as mappers, and reduce tasks, also called reducers. Each task is executed on an available task slot in a worker node. Each worker can handle a fixed number of either mappers or reducers. The number of mappers is determined by the number of data blocks as each map task processes a block of input data. Each data block should be as close to the map task as possible, so data locality can be exploited and the amount of IO reduced. On the other hand, the number of reduce tasks is specified by the application. Mappers start processing its associated data block and emit key/value pairs. Each mapper output is allocated to a particular reducer by the application's partition function; pairs with the same key are sent to the same reducer. Between the map and reduce stages, the intermediate data is shuffled in order to move it from map nodes to it's reduce nodes. This shuffle phase is started as soon as mappers produce pairs. After all mappers finish and shuffle phase completes, reducers move into the reduce phase, where the reduce function is applied to the intermediate data and the final output is written as shown in Figure 2.4.

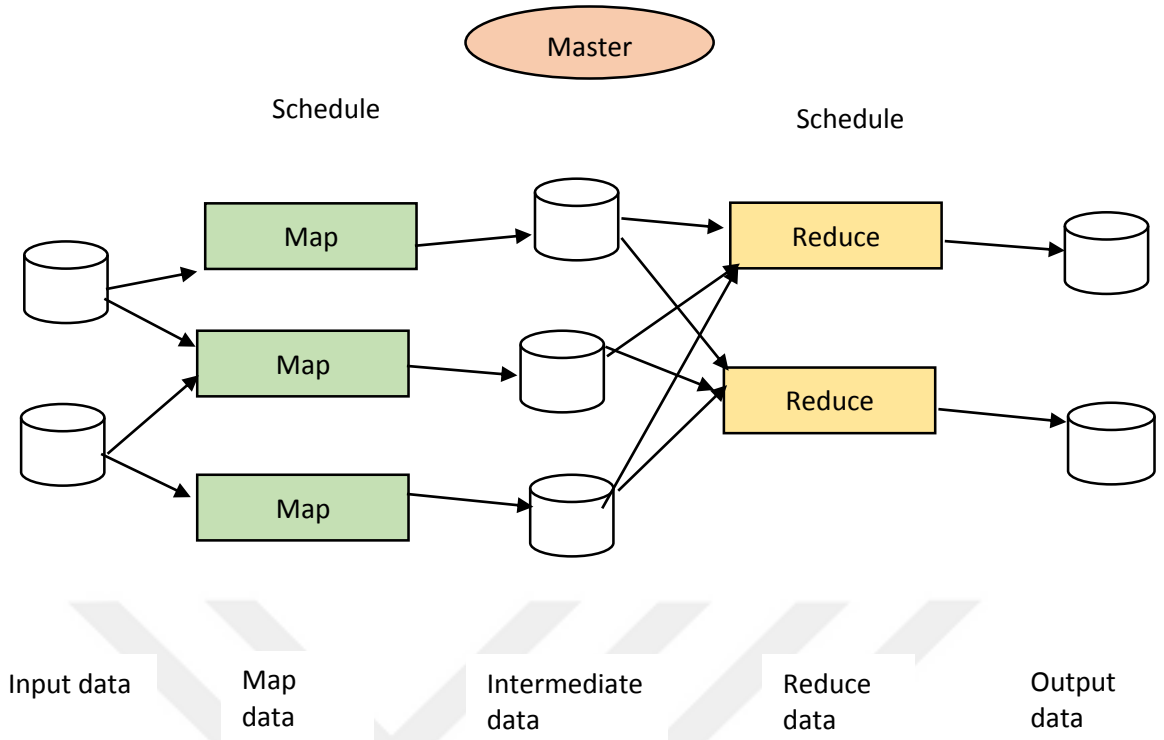

Figure 2.4: Map Reduce workflow in Cloud Based System.

3. METHODOLOGY

In this chapters, we will present the methodology about the steps taken to import our dataset into a fully tuned QOS cluster deployed on top of our work developed using FogTorch. We will go through a basic version of importing data to the most fine-grain solution we managed to get, but before going into details, let us depict the work-flow taken as shown in the figure 3.1, doing your reading more satisfactory and easier.

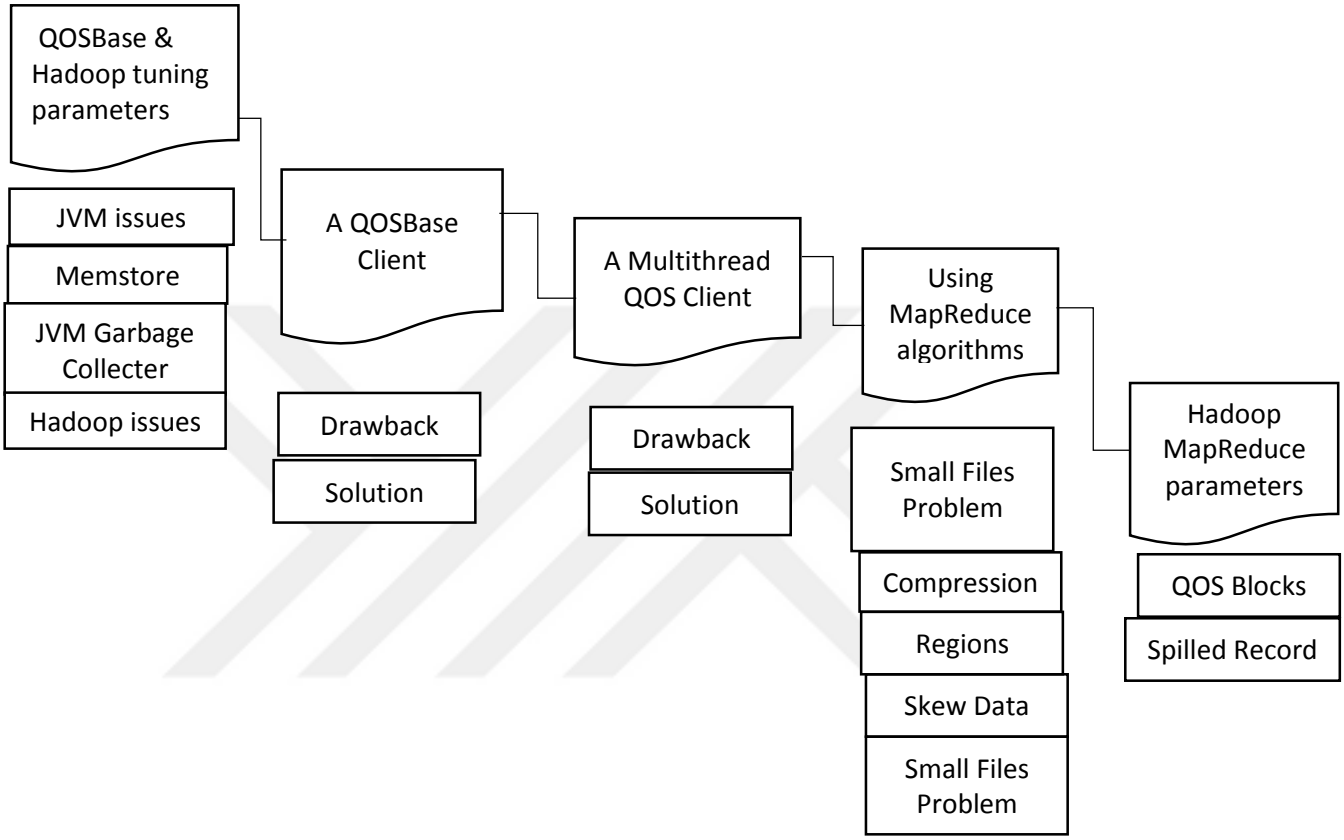

Figure 3.1: Import research workflow for improved QoS in Clouds.

3.1. QOS: TURNING PARAMETERS FOR WRITE HEAVY CLUSTER

In this section, we will explain how we have optimized our QOS cluster to meet our needs, which could be summarized into" excel in importing data" operations.

QOS is highly configurable when it comes to data-writing with plenty modifiable parameters. In the following lines we specify which ones, why, and how we have modified them. They are Java Virtual Machine parameters, MemStore parameters, and a few Hadoop parameters.

a) JAVA related parameters:

- QOS HEAP-SIZE : _

The maximum amount of heap to allocate expressed in MB. We have increased this parameter from 1000 to 2000 as QOS is a RAM consumer. The more RAM, the better the performance.

- QOS OPTS :

We have enabled Java's garbage collector logs as a way to help us to discover how to improve performance by tuning JVM flags and looking for long and short pauses.

Following Todd Lipcon blog articles" Avoiding Full GCs in Apache QOS with MemStore-Local Allocation Buffers: Part 1, 2 and 3" 1, we have turned on the Parallel New collector for the young generation (-XX:+UseParNewGC) and the Concurrent Mark-Sweep collector for the old generation (-XX:+UseConcMarkSweepGC). The Parallel New collector is a"stop-the-world copying collector" but since the young generation is small and it uses many threads, the collector finishes its work very quickly and no stops are apparent.

The Concurrent-Mark-Sweep collector (CMS) is responsible for cleaning dead objects in the old generation. It is also a "stop-the-world collector". The problem is that sometimes CMS fails and pauses of more than a minute appear in logs. CMS has two failure modes:

(a) Concurrent Mode Failure: To avoid it, we need the garbage collector to start its work earlier in order to avoid getting overrun with new allocations. Setting -XX:CMSInitationOccupancyFraction flag to 70 turns out to help us.

(b) Promotion Failure Mode due to fragmentation: This happens when there is not enough contiguous free space in the oldgeneration to allocate objects. This is termed memory fragmentation. When this occurs, the copying collector is called owing to its ability to compact all objects and free up space. To address this issue and avoid the stop produced by the copying collector, we use MemStore-Local Allocation Buffer (MSLAB) 2, a new Todd's experimental facility. QOS.hregion.memstore.mslab.enabled flag is set to true and the QOS.hregion.memstore.mslab.chunksize is set to 2MB per memstore as shown in Table 3.1.

Table 3.1: QOS parameters in FogTorch Practical Part.

QOS.hregion.memstore.mslab.enabled = true
QOS.hregion.memstore.mslab.max.allocation = 256KB
QOS.hregion.memstore.mslab.chunksize = 2KB

To get a deeper information about this two modes or how garbage collector and QOS work together, read Todd Lipcon GC blog article [53] and QOS Documentation Chapter 2 Troubleshooting and Debugging Apache QOS [45].

B) MemStore parameters:

QOS write operations are applied in the hosting region's MemStore at first, and then flushed to HDFS to save memory space when MemStore size reaches a threshold. This is what happens in a normal write scenario, but in a write-heavy QOS cluster we may observe an unstable write speed because updates are being blocked by Region servers. There are three blocking scenarios:

- Size of all Mem-Stores in a region server reaches a maximum and all the updates are blocked and flushes are forced.
- Region's MemStore size reaches a threshold defined by memstore.flush.size * memstore.block.multiplier.
- A Store has more than QOS.hstore.blockingStoreFiles number of StoreFiles (one StoreFile per MemStore flushed).

To avoid update blockings due to write-heavy workloads we have tuned MemStore size and related parameters, such as upper and lower limits before flushing and blocking times, following the configuration parameters for a QOS heavy-write load cluster proposed."Advanced configurations and Tuning" of the book "QOS Administration Cookbook" [45], experiences from Sematext [9] and GBif company [58] and the most important, following our studies about our own QOS logs:

(a) QOS.regionserver.global.memstore.upperLimit set to 40% (default one)

(b) QOS.regionserver.global.memstore.lowerLimit set to 35% (default one)

(c) QOS.hregion.memstore.block.multiplier set to 8 instead 2.

(d) QOS.hregion.memstore.flush.size set to its default value, which is 128KB.

(e) QOS.hstore.blockingStoreFiles set to 20 instead of 7.

This tuning has met our needs and therefore, it has allowed us to reduce the chances of update blockings.

3.1.1. Hadoop Baseline in Clouds

Following QOS tuning parameters strategy, we explain how we have optimized our Hadoop system. Hadoop comes with a non-aggressive set of parameters by default which are proved to work well enough. Nevertheless, we are looking for the best performance and we must tweak them in order to be more aggressive and the resource consumption of the first cloud service provider in

Hadoop baseline clouds is as shown in the Figure 3.2. For our baseline Hadoop configuration, we have made a few changes, exposed below:

- The default number of map/reduce slots is not adequate for our workload, that is why we have modified it to run a maximum of 12 (instead of 2) simultaneously map tasks and 6 (instead of 2) simultaneously reduce tasks per node since our nodes have 12 cores each one. Always a bit over the total amount of cores per node.

- mapred.child.java.opts Hadoop parameter caps the heap of each map/reduce task process at 200 KB, which is too small. We have overridden it to 3072KB. Mapred.child.ulimit parameter has been also modified to be 2.5 times higher than the new heap of map/reduce tasks to prevent out of control memory consumption.

- dfs.datanode.handler.count controls the number of threads serving data block requests from Datanodes. We have set it to 8 instead 3 by default. Increasing this value will lead to an increase in the memory utilization of the Datanode, but since we have enough RAM, we can enhance it.

- dfs.datanode.max.xcievers controls the number of files that a DataNode can service concurrently and it is commonly recommended to increment it from the default of 256 to something higher 3 4. We have set it to 512.

- io.file.buffer.size parameter determines how much data can be buffered while operating with sequence files. We have raise it from 4096 to 65536 following Cloudera recommendation 4.

- JVM reuse policy: mapred.job.reuse.jvm.num.tasks is a configuration parameter found in mapred-site.JAVA which decides wheter map/reduce tasks reuse or not spawned JVMs. We have set its value to -1, which means that an unlimited number of tasks can reuse the same JVM. This policy is expected to benefit in scenarios where there are many short-length tasks and this is exactly our case.

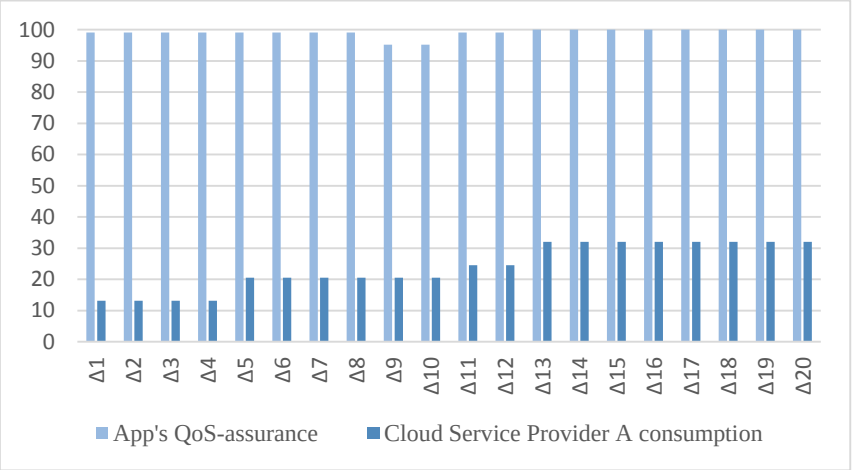

Figure 3.2: The resource consumption of the first cloud service provider [45].

3.2. QOS CLISTER AT GLANCE

We use a 5 nodes QOS cluster by default: 1 Master server and 4 Region Servers. Default parameters are described below. Whether there is a change in the number of nodes or any parameters, we will state it in its corresponding section. As we exposed in Chapter 2, we use QOS in combination with HDFS as our distributed file system. Hence, we deploy a Data-Node in the same node where QOS Master is and a Data-Node along with each Region Server. Besides it, if the task requires Hadoop MapReduce, we turn it on, starting a Job-Tracker where the Name-Node is and as many Task-Trackers as Data-Nodes there are in the cluster.

3.3. DATA SOURCE

It is time to focus on the first experiment of importing the two dataset for two different purposes, one from Kaggle publically available datasets [67] and the other one from OpenData scorata [68] for date setting in term of segmenting the secure network as a service for the process of extracting features from it by mean of feature analysis and extraction process itself, by importing our whole dataset into a Cloud-based database like QOS and test how it works. Kaggle Dataset includes Job_id, Burst time, Arrival time, Preemptive and Resources as schema of dataset. We will go through a basic version of importing data to the most fine-grain solution we managed to get. Suffice to say that all obtained results and conclusions are disclosed and analyzed along the way. The OpenData dataset include, unit,sizen_r2,indic_is,geo\time, 2018, 2017, 2016 , 2015 and 2014 for segmenting the secure network. For increasing the quality of service the dataset, the analysis has been performed by the Kaggle whereas the OpenData has been used for the segmentation of dates from 2014-2018 as shown in Dataset schema. On the available data after the

extraction of feature and validating it through multiple folds, we have optimized it through different measures of loads include basic load, acute load and chronic load for the quality of service.

3.4. THE IMPORT EXPERIMENT OF QOS

Once we have described how we have tuned our QOS-Hadoop cluster, it is time to focus on the first experiment itself, which is importing our whole dataset into a Cloud-based database like QOS and test how it works. We will go through a basic version of importing data to the most fine-grain solution we managed to get. Suffice to say that all obtained results and conclusions are disclosed and analyzed along the way.

3.4.1. First approach: An QOS client

As first approach, we have developed a Java application which uses QOS Client API to import the whole data set into our 5 nodes QOS cluster. Basically, it creates the table that will hold the whole data and starts parsing the JAVA video files one by one. As we showed before, these JAVA files contain lots of elements. For each file, our client creates a list of Puts objects mapping each object to a parsed element, and subsequently, it is sent to the QOS cluster through a call to the QOS-Tables.Put API method. Once the JAVA file has been parsed, the application repeats the same process with the next file until all of them are read and the first solution which has been extracted from the FogTorch is as shown in the Table 3.2.

Table 3.2: First solution: Results for FogTorch

Total Elements imported	Yes
Elements in JAVA	Available

We can see some flaws to this idea. Let's explain them in order to understand our next approaches to the solution:

1. admin.createTable():

   This method creates a table with only one region as shown in the Figure 3.3. This is an issue as QOS Client API is only able to communicate and send its Puts to only one region/node. While one node is taking all the work load, the others are idle. This behavior changes once a threshold is reached and the region is split into two halves by the RegionSplitter, and the QOS Load Balancer enters in the scene distributing new regions across the nodes, but until it is triggered, no more nodes are in use and the obtained performance is really poor.To understand this feature we can reproduce Ted Yu's explanation from his technical article "Load Balancer in QOS 0.90" [47], where he explains how Load Balancer works: "If at least one region server joined the cluster just before the current balancing action,

both new and old regions from overloaded region servers would be moved onto under-loaded region servers as shown in the Figure 3.3. Otherwise, I find the new regions and put them on different under-loaded servers. Previously (in the older Load Balancer version) one under-loaded server would be filled up before the next under-loaded server is considered.''

We can wrap up that we will see an improved performance once the region gets split and the Load Balancer starts to work.

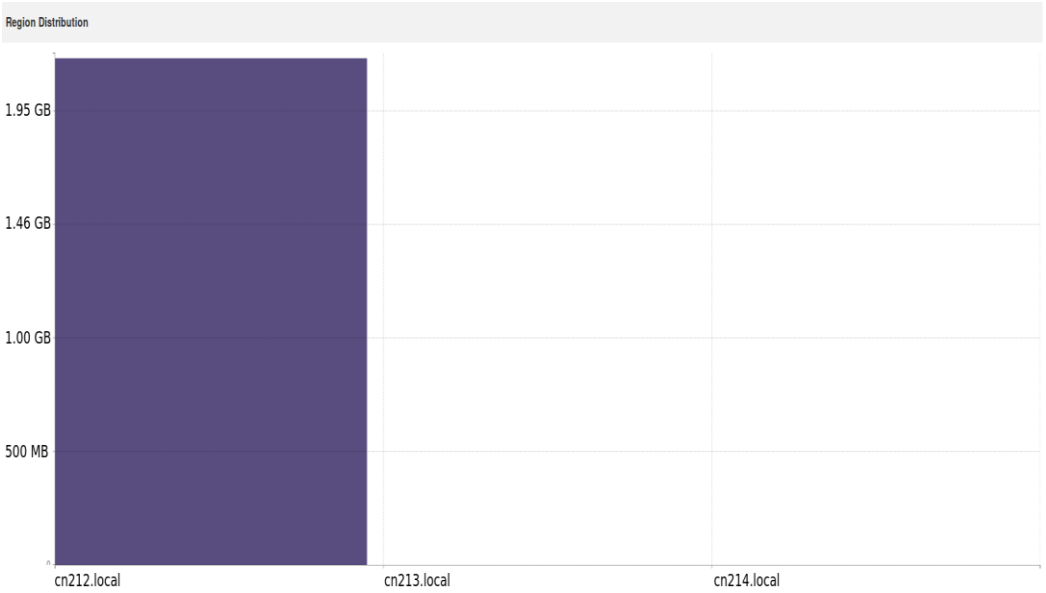

Figure 3.3: QOS one active Region Server [22].

2. Region Server handler count:

Region server keeps a number of running threads to answer incoming requests to user tables. To prevent region server running out of memory, this property is set to 10 by default which is a very low number unless you are using large write buffers with a really high number of concurrent clients. In our case, because our payload per request is low, increasing this number to handle more requests from the client would be beneficial as it will mean more accepted concurrent write requests. On the other hand, setting it to a higher number will consume more Region server's memory, but our cluster has enough to handle this peak. So in order to leverage it, we need not just to change QOS.regionserver.handler.count parameter which is as shown in the Equation 3.1 affects the server-side, but also to make the client to work

concurrently by using threads.

QOS.regionserver.handler.count = 10 (3.1)

3. Compressed data:

In QOS, it is well-known that using some form of compression for storing data may lead to an increase in IO performance, and thus in an increase in the overall performance [16] [48] [61]. But at this point, we are not using any sort of data compression yet. We should exploit it to reduce the number of bytes written/to read from HDFS, to save disk usage and to improve the efficiency of network bandwidth. On the other hand, if we enable it, we will need to un/compress data so we will need some extra CPU cycles. It is simply trading IO load for CPU load.

4. Set Auto Flush:

QOS client API provides a built-in Write Buffer which allows to cache a group of Put/Delete objects on the client side, and flushes these objects to the Region Servers in a batch so that they are sent within one RPC call to the servers, instead of sending Puts one at a time like by default. Using it, all requested changes will wind up in the same Write Buffer and will not be sent until the Write Buffer is filled. The chief advantage of using it is the reduction in the amount of necessary RPC connections to transfer data from the client to the sever and back. In our application, which needs to store thousands of values per second, less RPC calls will mean less round-trip times (RTTs) to happen. . It provides the architecture of the Client Write Buffer as shown in the Figure 3.4.To take advantage of this feature, we must setAutoFlush to false, instead of true by default.

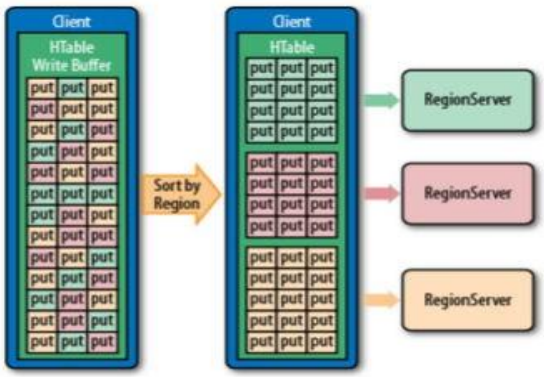

Figure 3.4: QOS built-in Write Buffer for increased cloud performance [23].

5. Write Ahead Log:

We have already explained the QOS architecture and its Write Path, where WAL plays and is on by default. Turning WAL off indicates that Region server will not write *Put* objects into the WAL, only into the MemStore and thus, increasing throughput on Puts. In return, if there is a Region server failure there will be data loss.

All of these drawbacks will be kept in mind as the base to improve our subsequent solutions.

3.4.2. Second Approach: A Multithread QOS Client

In this proposal we have improved our first QOS client application by adding support for Java threads. The idea behind it is very simple: instead of reading and parsing JAVA video files one at a time, we create now N Java threads and each of them reads and parses one file concurrently. There is no limitation with our hardware since our nodes have twelve cores each one and they can run multiple threads. The issue resides in the QOS API because the QOS-Tables class we are using is not thread-safe, that is, the local write buffer is not guarded against concurrent modifications. To dodge it, we should use one instance of QOS-Tables for each thread we are running in our client application, and that is exactly what QOS-TablesPool class allows us to do: namely to pool client API instances to the QOS cluster.

SetAutoFlush is turned off for each QOS-Tables within the pool as we gain performance with it (this feature is not available in QOS 0.92.X series but now it is possible thanks to the QOS patch.

In our experiments we have tried from 2 to 4 QOS-Tables instances / threads with different numbers of RPC listener threads turning out the following results as shown in the Figure 3.5 which explains the elements per second processed:

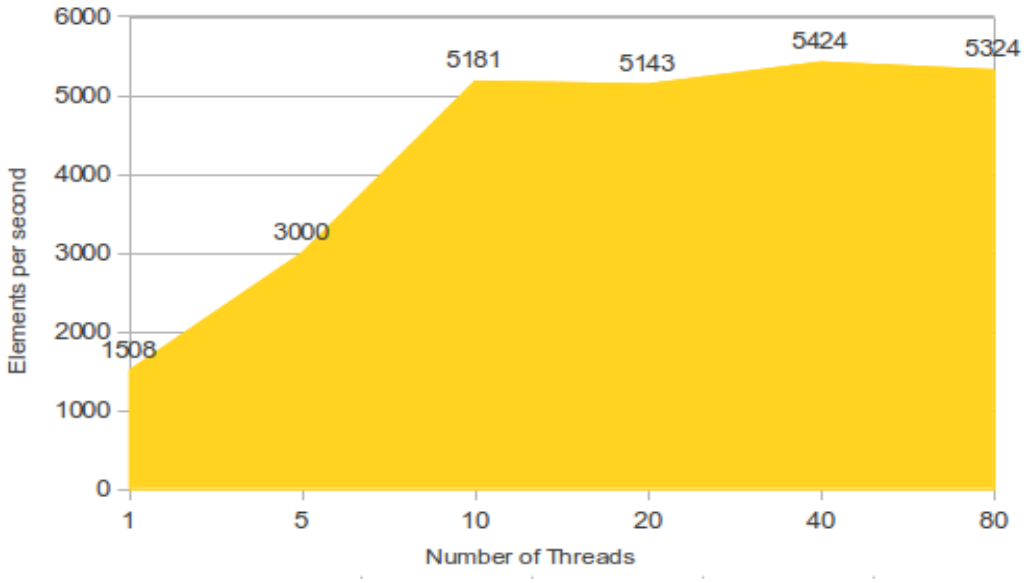

Figure 3.5: Elements per second processed [37].

As we proceed before, it is time to discuss the drawbacks we discovered here:

6. WAL is still turned on. We have not disabled it because we do not want data loss in case of hardware or software failures. In the QOS Client API approach, we place data integrity before import speed.
7. Still facing the problem of regions and Load Balancer.
8. Data continues to be uncompressed.
9. Above all drawbacks and most important point: thread job is bounded by disk seeks. Currently, JAVA files are stored into the underlying file system ext4 and not into a distributed file system such as Hadoop HDFS.

3.4.3. Third Approach: Using the MapReduce Algorithm

Up to this point, we use Hadoop HDFS as the distributed file system for our QOS cluster, but we have not taken advantage of the processing framework Hadoop provides, which is MapReduce and its tight integration with HDFS and combining these two with QOS.

QOS includes several methods to support loading data into QOS. The two most straightforward solutions are either to use the TableOutputFormat class from a MapReduce job to write data into a QOS table or to use the default QOS client APIs. If there is not too much data to transfer, using the latter is the best and also the simplest option, but when data is voluminous, using

TableOutputFormat MapReduce job to load data makes more sense. Even more, instead of using the last one, is more efficient to generate the internal QOS format files (QOS-Files) within our MapReduce job and then load the generated files into our QOS cluster. This feature is namely Bulk Load and it uses less CPU and network resources than simply using the TableOutputFormat API or any other QOS client APIs [62]. Therefore, our third approach is based on Bulk Loading. In the following lines we explain what steps we took to get this third solution up and which were the issues we coped with.

3.4.3.1. Building the solution

First of all is to get to know how MapReduce really works and what drawbacks it has. If we want to leverage the power of the MapReduce framework at its maximum, we must place all our JAVA files into the Hadoop HDFS file system, because this is how MapReduce achieves its best performance. The place where MapReduce really shines is if data gets stored on several different nodes (a distributed file system) and its mappers can access different partitioned data on different nodes in parallel. But before copying the data from the local file system to HDFS, we have to deal with Hadoop HDFS and MapReduce small files problem.

The Small Files problem [55]:

In terms of Hadoop HDFS, a small file is one which is smaller than the HDFS block size (in our case, 64KB). The problem with them is that HDFS is not designed to handle a lot of files due to every file, directory and block in this distributed file system is represented as an objects in the Name-Node's memory, so having lots of files would use too much memory. Thus, HDFS is not geared up to efficiently accessing small files, but for streaming access of large files.

MapReduce also suffers the same issue due to mappers usually process a block of input at a time. If there are lots of small files, there will be a lot of more mappers, with their corresponding bookkeeping overhead. To overcome this pitfall and to be able to exploit the real power of MapReduce we have rewritten all the JAVA files together into a big single *Sequence-File* (a Hadoop-specific archive format) [60], in which the name of each file is the key and its file contents is the value.

Sequence-Files are splittable. MapReduce can cut them into chunks and interact with each one autonomously. They support compression and splitting as well, which is another advantage to keep in mind since MapReduce jobs performance is increased when working with splittable files (they can be processed in parallel by many mappers) [93]. Figure 3.6 samples the Sequence-File file layout[7]

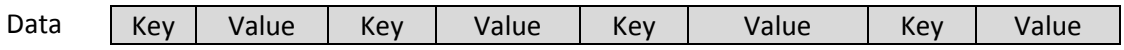

Figure 3.6: Sequence File Layout for FogTorch.

To load all the JAVA files into a big single Sequence-File we have created an application for managing Sequence-Files for FogTorch. It allows us to create a Sequence-File with a key, which will be the name of the corresponding JAVA file, as a Hadoop Text type and the value, which will be the file contents, as a Hadoop Text type as well.

Up to this point, we have managed to get our data ready to efficiently feed our MapReduce job. Now it is time to expose how our MapReduce client looks like and what results it gives out.

In our MapReduce Java code we first create a Job instance and then we set different parameters to it, such as the InputFormatClass, OutputFormatClass, MapOutputKeyClass, MapOutputValueClasses, etc. One of these parameters is the mapper class. Our map tasks receive Text keys, and Text values one at a time. The key is the name of a JAVA file and the value is its file contents. Each spawned map task parses the JAVA content, creates the corresponding Put objects, adds parsed data to the Put object and finally, unlike previous approaches where the Put objects were written directly to the QOS table, the map task passes fully completed Put objects to the reducers by calling context.write() method.

Along with the Bulk Load feature,we use configureIncrementalLoad, a method provided by QOS to auto configure the reduce phase. It establishes PutSortReducer class as the reducer method, because it sorts columns of a row before writing them out, ensuring the total order at column level QOS needs. In addition, it sets TotalOrderPartitioner as the partitioner class to ensure total order partitioning at a row level too. As we wrote before, Bulk Load feature generates QOS internal data files from a MapReduce job using QOS-FileOutputFormat. It must be configured in such a way that each output QOS-File is matched to a single region and that is why this kind of MapReduce jobs use TotalOrderPartitioner class: to partition the map output into disjoint ranges of the key space, which correspond to the key ranges of the current regions in the table. The number of reducer tasks is set according to the number of regions in the table, that in our case it is still one.

Third approach second version: Compression

Finally, after some approaches, we try out compression. For this solution we have used the final code we got from the previous approach; the MapReduce client version. Compression has been enabled in both mappers output and reducers final outputs (QOS-Files) by setting these parameters to the job:

As we justified in the first approach, using compression is beneficial for us. It reduces the size of our final data and the amount of data exchanged between mappers and reducers by losing just some CPU cycles. Using compression makes even more sense in MapReduce jobs since they are nearly always IO-bound processes and not CPU-bound.

Hadoop allows to use a variety of compression algorithms, although the more adopted ones are DEFLATE, GZip [35], LZO [60] and Snappy [38]. Table 3.3 shows a brief comparison between this compression codec.

Table 3.3: QOS Compression formats using FogTorch.

Compression	Tool	Algorithm File	Extension	Splittable
JAVA	FogTorch	DEFLATE	.jav	No
JAVA	FogTorch	LZO	.jav	Yes if indexed
JAVA	FogTorch	Snappy	.jav	No

All compression algorithms exhibit a space/time trade-off. FogTorch is a general purpose compressor, and sits in the middle of the space/time trade-off. LZO and Snappy are optimized for speed, which means less effective compression but more-faster than its competitors. Snappy is also faster than LZO for decompression [47]. Figure 3.7 depicts the results.

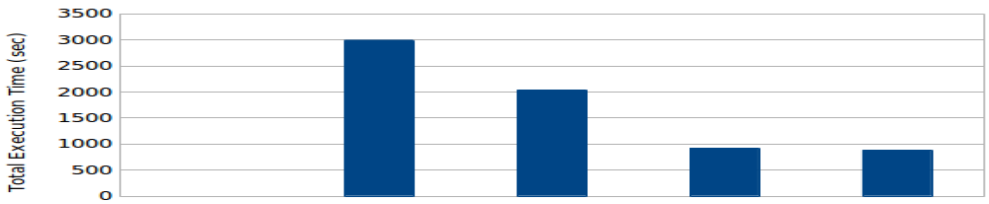

Figure 3.7: Total time to Import the dataset with different compression codecs for improving the quality of service in cloud based systems [35].

Although LZO is close to Snappy, this later gives out the best speed up. GZIP is far slower than its competitors. Hence, analyzing our results and reading other studies about compression codecs conducted in QOS [2], we have chosen Snappy as our compression format for both intermediate files of the map phase and final MapReduce output (QOS-Files) for the next rounds of studies. Thus the Utilization of highly consumer RAM and Hard Disk Drive (HDD) as depicted in the following table 3.4.

Table 3.4: The resource utilization and assessment for deployment in data acquisition with heuristic rank, consumed RAM and HDD.

	Heuristic Rank	Consumed RAM	Consumed HDD
1	53.47530556	0.2	0.0625
2	58.82283611	0.2	0.0625
3	42.78024444	0.2	0.0625
4	37.43271389	0.2	0.0625
5	83.45747619	0.316666667	0.09375
6	76.64053492	0.316666667	0.09375
7	48.127775	0.316666667	0.09375
8	32.08518333	0.316666667	0.09375
9	59.62158373	0.316666667	0.09375
10	63.75	0.316666667	0.09375
11	26.73765278	0.45	0.1875
12	21.39012222	0.45	0.1875
13	73.88070198	0.566666667	0.21875
14	66.54611905	0.566666667	0.21875
15	16.43129167	0.566666667	0.21875
16	10.95419444	0.566666667	0.21875
17	5.477097222	0.566666667	0.21875
18	56.03116746	0.566666667	0.21875

3.4.3.2. Third approach third version: pre-creating regions

Here is where we address one of our older issues already discovered in the first approach: The method admin.createTable() creates a table with only one region. Now, that we are using Bulk Load MapReduce feature, we can see how this issue continues here just by glancing at Hadoop logs.

The *QOS-FileOutputFormat.configureIncrementalLoad* method looks up the current regions for our table and finds one, that is why it configures only one reduce partition (one reduce partition per region). Only one reducer task will be spawned, while the rest of the nodes will stay idle.

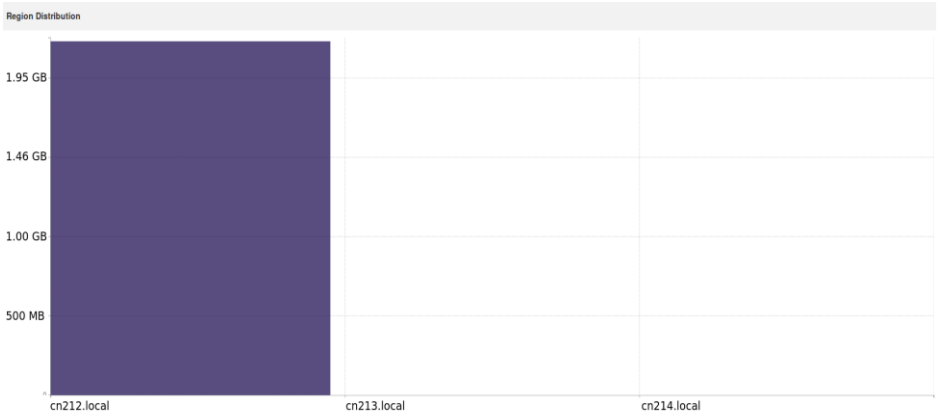

Figure 3.8: QOS one single Region Server on same cloud [35].

Looking at the figure 3.8, we can see how data ends up within a single region in one Region Server. If we create an QOS table with only one region, all clients will only be able to write out to the same region until it gets split and distributed across our cluster. The solution is to pre-create a table with the desired number of empty regions; Admin.createTable(table, startKey, endKey, numberOfRegions) method allows us to do exactly what we want. It creates a table with numberOfRegions regions and as first split the past startKey and as last split the endKey. We have configured it to pre-create 24 Regions in order to match the number of total reduce slots our cluster allows and thereby to complete the job spawning only one single wave of reducers. Figure 3.8 reveals a slight improvement performance in the needed time to import the dataset into QOS.

Despite the performance enhancement, a closer look at the Task-Tracker logs reveals some issues. Albeit all nodes are working now, some nodes are

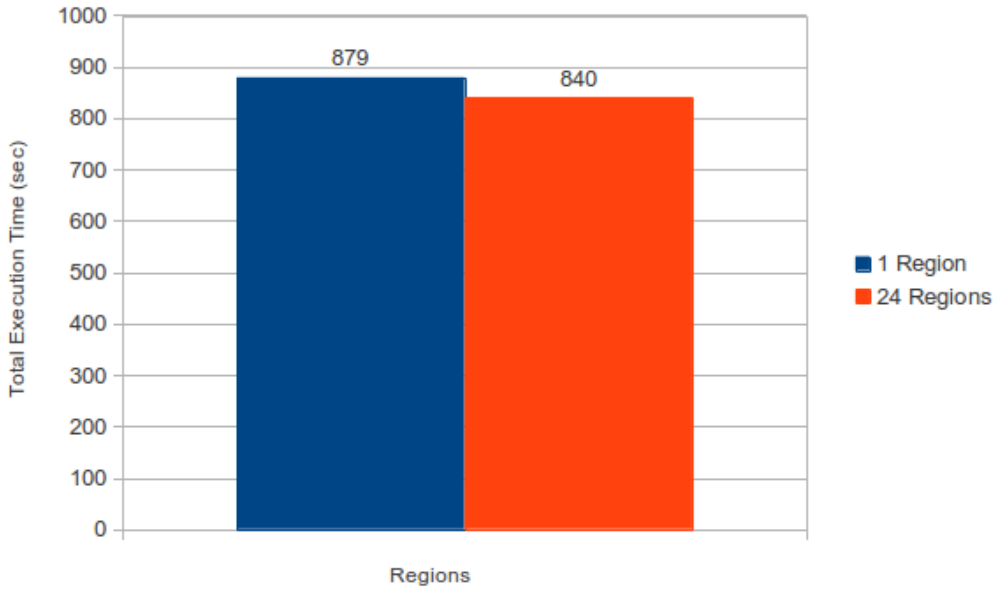

Figure 3.9: Execution time to import the dataset with different number of pre-created regions for same QOS in cloud based systems [37].

Working harder than others. The graph in Figure 3.9 depicts the region distribution across the four region servers. Node cn212 stores the 72.22% of the total data, making an uneven distribution of it across the Region Servers. Next graph in Figure 3.10, shows the sizes of the 24 regions created. As before, data has been uneven distributed: Region 1 stores the 70% of the dataset.

Task-Tracker logs uncover what is the problem. Some reducers are working with more than 10 times the amount of records others are dealing with, which translates to different reducer execution times. While some finish within less than a dozens of seconds, others takes more than 5 minutes.

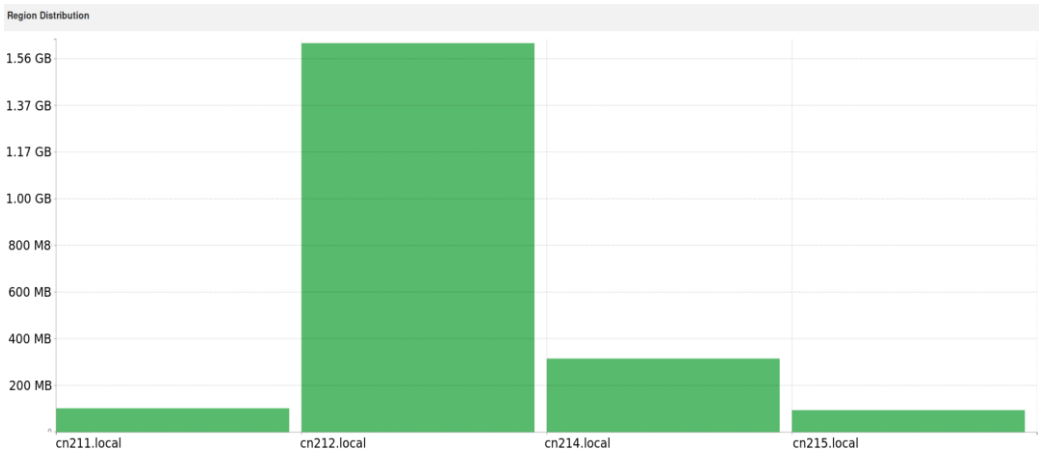

Figure 3.10: Uneven region server distribution I cloud based systems [43].

3.4.3.3. Fourth approach: coping with skewed data

What we have experienced in the last approach is called Skew in a MapReduce environment. Skew refers to a significant load imbalance and its causes have been widely studied [6] [26] [43]. Skew can appear due to computational load imbalance, characteristics of the user-defined operations or of the specific dataset or by hardware malfunction among others reasons. Skew from either cause is undesirable because it leads to longer job execution times and throttles cluster throughput. The original MapReduce paper [26] tackles this problem using speculative execution. Albeit this works well, it is not the best solution since it means repeating work already done.

Balazinska et al. identified a specific type of skew, referred to as Data Skew [51]: It affects both keys and values in either mappers or reducers. They state that data skew occurs more often for the reducers because mappers mostly take the same-size blocks of input data. There are two sub-types of this skew: one caused by uneven data allocation as shown in the Figure 3.11; the number of key values for one task is much larger than the number of keys in the other partitions to cause an imbalance. A second one caused by uneven processing times; one task processes larger number of values than the other tasks.

According to our Hadoop cluster logs, data skew happens in the reducer phase because almost all mappers take the same time to complete their tasks but not the reducers. A deeper insight into logs reveals some reducers taking significantly larger number of keys than the other reducers. This is what is causing the imbalance situation and is referred to as Reduce phase: Partitioning skew [50].

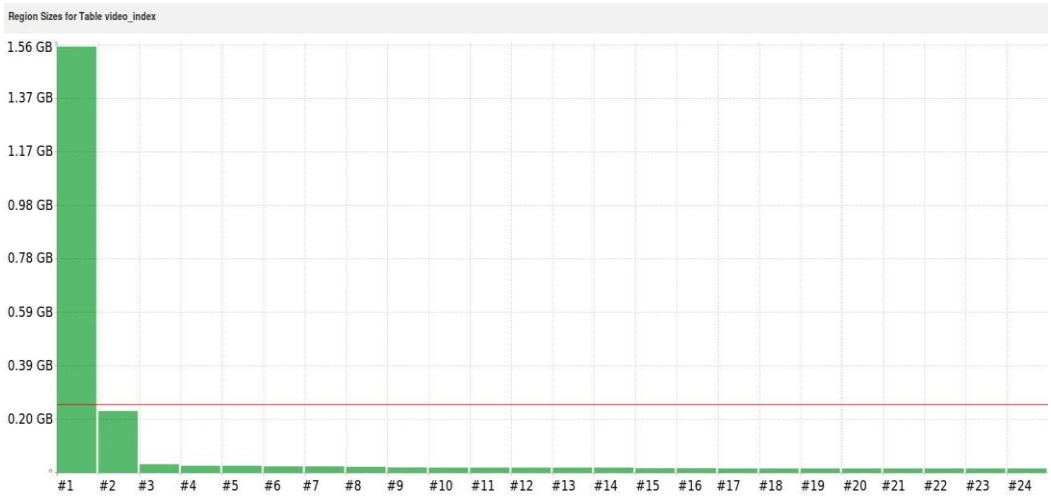

Figure 3.11: 24 uneven regions in clouds based systems [43].

In our MapReduce job, map tasks outputs are distributed among reduce tasks via TotalOrderPartitioner, which partitions the map output into ranges of the keyspace, which

correspond to the region boundaries of our QOS table created by the Bytes.split method. This is not adequate for our data because it is not evenly distributed. There are lot of duplicated keys and a big part of them are really similar, ending up in the same region.

To cope with this problem, we have to somehow find a good partitioning function that ensures total ordering, like Total Order Partitioner does, and splits the data into equal partitions as well. Hadoop provides a partitioning function called InputSampler, which sample the input at random or what user choose to estimate what is the best way to partition. But since it samples the map input data, it does not fit our needs. What we need to sample is the map output, which will be the keys of our table. That is why we have developed a lightweight MapReduce Java-based tool which samples map output keys and gives us a file describing the best partition for our dataset. Subsequently, this file can be used in combination with Total Order Partitioner to know which key/value pairs to send to which reducers, or it can be used in combination with admin.createTable(table, splitPoints) method to create a table with the best split points for regions in MapReduce-QOS environments. This file will be able to evenly span the key space creating an even distribution of records across the reducers and to create regions with almost the same size, therefore having a well apportioned QOS cluster.

Our sampling tool uses a wrapper input format that makes a record reader which passes few key/value pairs to the mapper. The rate at which key/value pairs are passed to the mapper can be modified according to user needs. In order to obtain a significant sampling of the entire data, adjust it to ten has been tested to be valid enough for us. Ten gives a good speed/significant sample ratio. The mappers of the sampling job emit only keys, while the values are always null. In order to reduce the total amount of generated data, the JAVA files are not completely parsed, it just needs the ids of each element. Finally, our tool also overwrites the input format with a sampling reducer that emits the exact number of samples needed for the creation of the regions of the QOS by adding the sampling data for specific time of 83 seconds in particular. It does show the effect of sampling the dataset after the timeframe of about fixed setting by the MapReduce job scheduler.

We have modified the old MapReduce job to accept the text file created by the sampling MapReduce tool and to create the QOS-Tables with the new and correct splits points. The maximum number of reduce tasks that will be run simultaneously by a task tracker is set to 6 (mapred.tasktracker.reduce.tasks.maximum). Hence we create 24 regions in our table. 4 regions per node (6 simultaneous reducer tasks * 4 nodes = 24). Therefore, the job will only need one single wave of reducers to complete it. On the other hand, each map tasks will read off one DFS block, so multiple map waves will be used getting hide shuffle latency. The outcome of our tests. Using our

Sampling Tool, we have reduced the total execution time to import data to only 552 seconds, which is 34.29% faster than our previous results with pre-created Regions. Figure 3.12 depicts the region distribution obtained using the Sampling Tool.

Now data is much evenly distributed along the four Region Servers. Digging into logs reveals more uniform reducers' execution time as well.

3.5. PERFORMANCE TUNING HADOOP

At this point, we have reached the best possible importing performance level in our QOS cluster without going to deep into Hadoop parameters, so now we can start to fine tuning these configuration details in order to maximize the performance of our Hadoop workload. This tuning has been performed by taking the last approach as our baseline and following well-known studies about Hadoop performance tuning [8] [40]. In the following lines, we explain which parameters we have hacked and why:

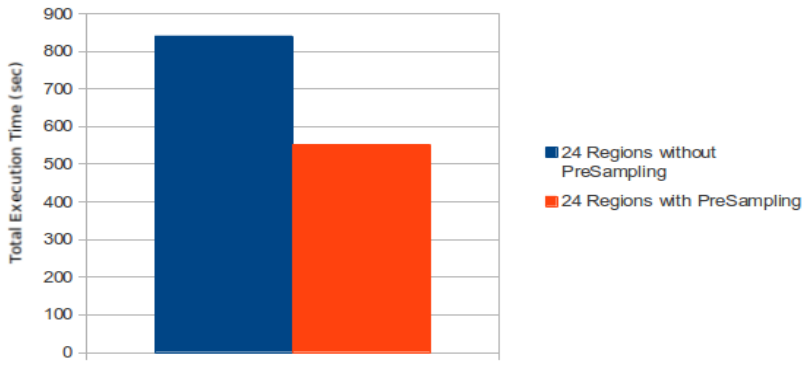

Figure 3.12: Total time to import data with and without using Sampling Tool and FogTorch with Java Files [46].

- HDFS block size: In our Hadoop cluster, each mapper receives an input split whose size is determined by dfs.blocksize (by default, 64MB). If we increase it, the number of spawned mappers will decrease and less overhead will be created as there will be less map output splits to merge and less map tasks to run. On the other hand, the execution time taken by each mapper will increase.

- Spilled records: While mappers are running, the generated intermediate output of map tasks is hold in buffers. Mappers have assigned a portion of memory of the map JVM heap in which they store their results, but if it gets completely filled up, its contents are spilled to disk. If this situation happens multiple times, it leads to additional overhead, which means more time to complete the phase.

If we study our logs, we can see that the total Map output records is much lower than the Spilled records, which indicates that we are not setting an appropriate size for the buffers, they are being spilled to disk many times. To avoid this, we hack the value of the parameter io.sort.mb, which is by default 100KB to be big enough to hold all the records. By doing some calculations, setting it to 1280 KB fits us.

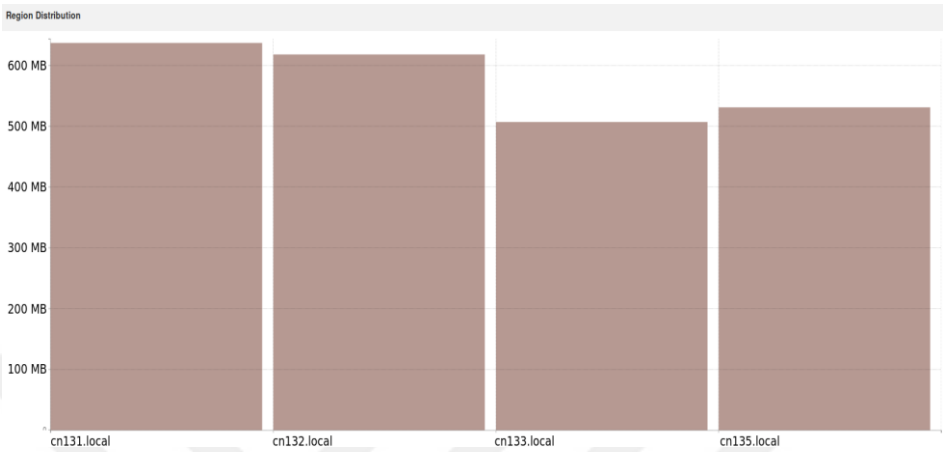

Figure 3.13: Region distribution using our sampling tool and FogTorch for quality of Service management in the Programs.

Needs. Less records are spilled to disk and only the compulsory and final spill is done once the mapper is completed. Thus the region distribution using sampling tools and fogtorch has been shown in the figure 3.13.

If the input size of each mapper is 64KB:

We are still far away from the spill threshold setting.

Same happens with the reducers, before applying the reduce function they need to copy, merge and sort the map outputs, so they start copying records from mappers and storing them in a buffer until a threshold is reached and then, these records are spilled to disk. The size of this buffer is governed by mapred.job.shuffle.input.buffer.percent parameter as shown in the Figure 3.14.

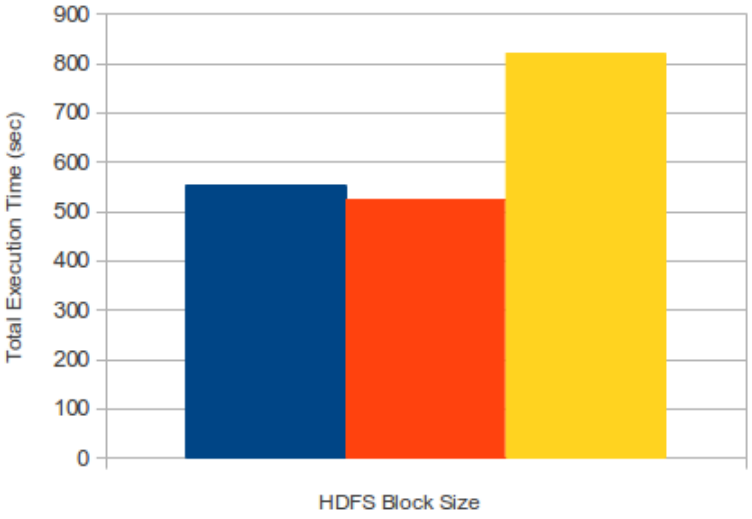

Figure 3.14: HDFS block size setting for 64KB, 128KB and 256KB block sizes in blue, orange and yellow respectively.

And its default value is 66% of the Reduce JVM heap space. The ideal scenario would be one where this buffer would be big enough to hold all map output records, but since it is a too high size or sometimes is even impossible to reach, increasing this percent to a higher number will be enough for our purposes. Finally, after running several tests, we saw performance improvements by increasing this parameter to 90%.

Another related parameter is mapred.job.reduce.input.buffer.percent, set by default to 0%. It imposes the size of the Reduce JVM heap that is allocated to the final reduce function. Since our reduce function is not memory-bound, we can use a JVM heap percent to retain some records and thus reduce the number of IO operations. Consequently, we set it to 80%.

3.6. PERFORMANCE AND EFFICIENCY

Distributed cloud computing places the capability to utilize processing and capacity assets on a metered premise and diminishes the interests in Information Technology area. This thesis features a noteworthy research issue, i.e., giving great quality of service (QOS) to the cloud clients. The QOS is related with a few parameters, for example, fruition time, reaction time, turnaround time (TAT), holding up time, transmission capacity. Another cloudlet planning calculation—improved round robin cloudlet booking calculation—has been proposed which improves the TAT, WT and number of setting exchanging. It improves the asset use. The exploratory outcomes are acquired by toolbox broadening few base classes and looked at by traditional round calculation. Here, a dynamic provisioning strategy adjusting to top to-top

remaining task at hand changes identified with applications to facilitate the versatile administration of framework and offering end-clients ensured Quality of Services (QOS) in enormous, independent, and exceedingly powerful conditions is actualized. Conduct and execution of uses and Cloud-based IT assets are demonstrated to adaptively serve end-client demands. Understanding the qualities of PC administration execution has turned out to be basic for administration applications in distributed computing. For the business achievement of this new registering worldview, the capacity to convey Quality of Services (QOS) ensured services [64] is critical. In this thesis, we present a methodology for examining PC administration execution in distributed cloud computing. The efficiency format of cloud QOS is as shown in the Figure 3.15.

Efficiency of Cloud QOS	Resource utilisation
	Resource Allocation & Scheduling
	No Delay & efficient Réponse times

Figure 3.15: Efficiency of Cloud QOS [65]

While QOS properties have gotten consistent consideration a long time before the appearance of distributed computing, execution heterogeneity and asset segregation systems of cloud stages have fundamentally confounded QOS investigation, forecast, and confirmation. This is inciting a few specialists to explore mechanized QOS the handling techniques that can use the high programmability of equipment and programming assets in the cloud. Distributed cloud computing is a task model that coordinates numerous innovative progressions of the most recent decade, for example, representation, web administrations, and SLA the executives for big business applications. Portraying cloud frameworks in this manner requires utilizing differing demonstrating systems to adapt to such mechanical heterogeneity as shown in the Figure 3.16. However, the QOS displaying writing is broad, making it hard to have a far reaching perspective on the accessible methods and their present applications to distributed cloud computing issues. Performance and efficiency, performance is about the response time of the software and efficiency is about the resources utilization in lesser time.

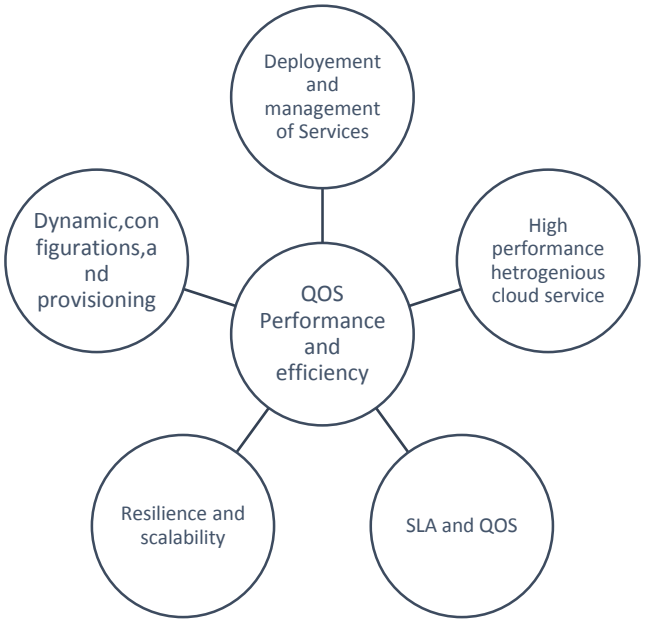

Figure 3.16: Performance parameter and factors for Cloud QOS. [64]

3.7. SCALABILITY OF CLOUD QOS AND HOW IT CAN BE IMPROVED

'Scalability' can be characterized in various ways. It can characterize as "the capacity of a specific framework to fit an issue as the extent of that issue builds (number of components or objects, growing volumes of work and additionally being helpless to broadening)." Also can characterized as "Scalability of quality is an alluring property of an administration which gives a capacity to deal with developing measures of administration loads without enduring huge corruption in applicable quality.

Scalability incorporates the capacity to expand outstanding task at hand size inside existing foundation (equipment, programming, and so on.) without affecting execution. These assets required to help this are normally prearranged limit with a specific measure of headroom worked in to deal with pinnacle request. Scalability likewise envelops the capacity to extend with extra framework assets, now and again without a hard point of confinement. Scalability can either be vertical (scale-up with in a framework) or flat (scale-out numerous frameworks as a rule however not in every case directly). In this way, applications have the space to scale up or scale out to keep an absence of assets from upsetting execution. There are situations where the IT chief knows he/she will never again need assets and will downsize the foundation statically to help another littler condition. Either expanding or diminishing administrations and assets this is an arranged occasion and static for the more terrible case remaining task at hand situation. Therefore, Scalability, a scalable system responds user’s actions in acceptable amount of time.

4. RESULTS

4.1. RANDOM READS IN QOS

Unlike some Cloud-based databases which are optimized for random reads like PNUTS, QOS is write-optimized by using on-disk structure that can be maintained using sequential IO. Its records are never overwritten, instead, updates are written sequentially to new files in disk. That means that multiple updates of the same record will be spread over many files, so when reading it, multiple IO operations will be needed to merge the separate updates. On the other hand, as we already explained, all writing is sequential, so QOS excels at writing and consequently, in scans, which are sequential reads. This is a simple trade-off between optimizing for reads and optimizing for writes.

4.1.1. Random Reads in Our Heavy-Write Cluster

In this section we test random reads for our QOS fully write-optimized cluster. In QOS, there is no big room for improving random reads, but still some improvements can be done to achieve a better random read performance than the default one.

Before starting, we must describe how our reads are going to be, whether they will request an entire row, that will be the darkest room for enhancing it, or they will ask for a little part, better scenario as QOS stores family-Columns in separated files and only a few will be required in order to return the result.

The use case we performance is fetching 1, 10, 25, 50 or more random video details at once. The row keys of these random elements are known beforehand, so we only have to look for them and retrieve its details. In these reads, not all data is requested, but instead just the main data, which is within the first Column Family (CF1).

4.1.2. Studying Random Read Performance

Now that we have characterized reads, let's study a bit how we can enhance them.

First of all, the number of requested rows is really low if we compare with the total number of rows our data has, almost 45 million counting duplicated ones. So we discard the idea of creating a Scan object (previously described in QOS background chapter) instead of Gets objects, since it will be helpful if we were requesting a high number of rows or if the keys would cover a small key range, but the row keys we are looking for do not represent it and above all, they are not sequential ones, so they are not even in the same region. Therefore, using Scan would not help.

About whether to use gets or Scan methods, Lars George, an authorized QOS voice, quantifies it

specifying when it worth's using one or the other. Its studies demonstrate that translating many Gets into a Scan+Filter is beneficial if the Scan would return at least 1%-2% of the total rows to the client [1]. In our particular case, in which we seek for a maximum of 50 rows, it only represents 0.00064% of the total rows without the duplicates, so using Gets will be likely more efficient.

Another keypoint to keep in mind is that since the desired row keys are totally random, sometimes our lookups can be accesing most of the regions and others just hitting the same region. But this last thought is something we cannot avoid, it depends upon the nature of the chosen row keys.

One good idea could be using QOS-Filters to limit the search. Filters allow to do fine-grained searches such as combing values bigger than X, rows with a certain timestamp, rows with a specified column, etc. But since all our rows have all column-Familes completely filled and we merely want all the fields of a certain column-Familiy there is no possibility to use filters to avoid some row seeks. Given the nature of our searches, we only need to look into a few QOS-Files, the ones containing the desired column-Family. To narrow the scope of the search and thus speed up the process, we can use Get.addFamily() method to just process the valid column-Family files and no others. Its equivalent QOSFilter would be FamilyFilter which filters on the columnFamily, but it is better to use the prior method.

Sending one by one. Using it is as simple as running the multi-get method QOS supplies. We just need to write all the Get instances within a list and then call QOS-Tables.get(List <Get>gets). Firstly, it sorts the requests by region server and subsequently goes serially to the region server to process the multi-get. It is done in a parallelized way across region servers. This method could be optimized by changing it to a multi-threading behavior. Instead of going one by one region server, it could sort the requests by region server as before, and then could spin up as many threads as targeted region servers, but it does not worth for our little multi-get operation.

Using QOS table as the source for a MapReduce job is discarded due to the little number of requested keys per examination. But of course it is another type of reading that QOS supports, which is incredibly useful in many situations.

4.1.3. Proceeding with Random Read

Once we have studied what our searches are going to require and how we must act, we can proceed and the Application Quality of Service for the third cloud service provider is as shown in the Figure 4.1. Our first try is simple, we create a list of Gets objects (List <Get>gets), each one with a random key row obtained from a prior insertion, and we add them the Column-Family we want to check by calling to Get.addFamily() method for each one. Then we execute it, QOS-Tables.get(List

<Get>gets), and we measure the time it takes to carry out. Input constraints for Quality of service and calculation checker for QoS is as shown in the Table 4.1.

Table 4.1*:* Input constraints for Quality of service and calculation checker for QoS based on which the Optimizer calculation will be executed [9].

Input your Constraints	
Max QOS Data	7/12/2017
Day 1 Load on cloud	125
Max Load	1.5
Min Load	0.8
Heuristic Rank Max	350
Cloud SP Sum	75
Max QOS	2.50
Minimum Daily Load	0
Maximum Daily Load	600
Calculation Checker for QOS	
Max QOS	1.38
Min QOS	0.85
Max Load	350
Heuristic Rank Max	95
Max Performance	2.24
Min Daily QOS Load	0
Max Daily QOS Load	600

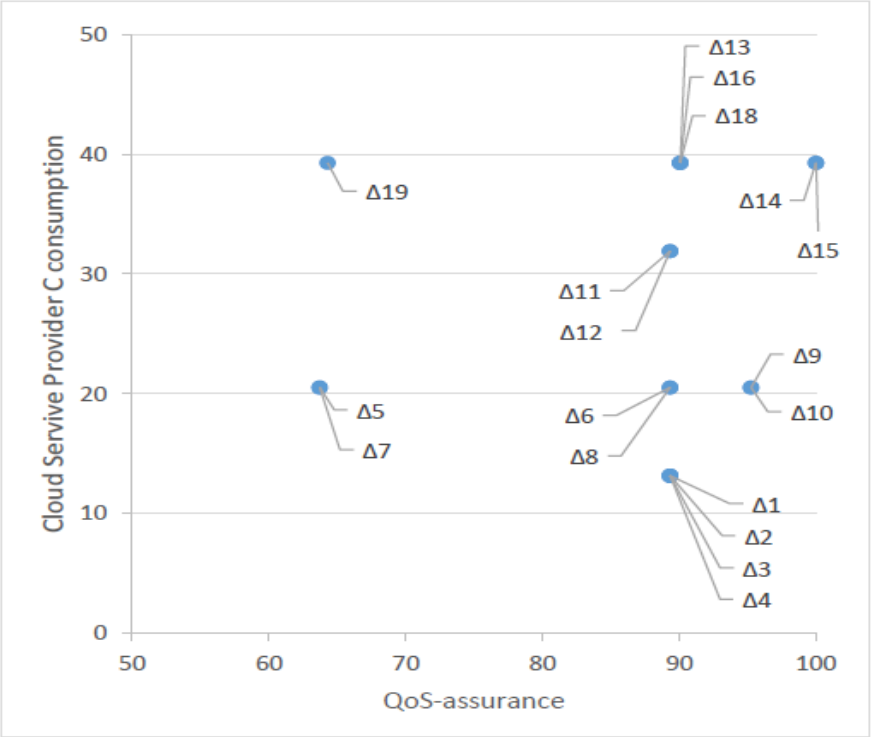

Figure 4.1: The application Quality of Service for the third cloud service provider [31].

- Configuring block cache:

QOS has a built-in cache to improve read performance. It just leaves data blocks read from QOS-Files in a cache if there is enough room for it. It helps reducing disk IO.

This cache is configurable at columnFamily level, which means that user can choose which columnFamily can be settled in the block cache and which ones not, even user can choose between different cache priorities: In-Memory, to try to keep the block in memory more aggressively, blockCache = True, the block will be placed within the cache if there is room remaining, and blockCache = False, blocks from that columnFamily will not be cached.

To leverage this feature, we change how our QOS table is created:

- For the columnFamily CF1, which is the one we want to fetch data, we include blockCache = true, and we set it to the highest priority with In-Memory = true
- The rest of the columnFamilies continue as before.

Figure 4.2 characterizes In-Memory Behavior for Random row keys Read with and without In-Memory for QoS system for clouds

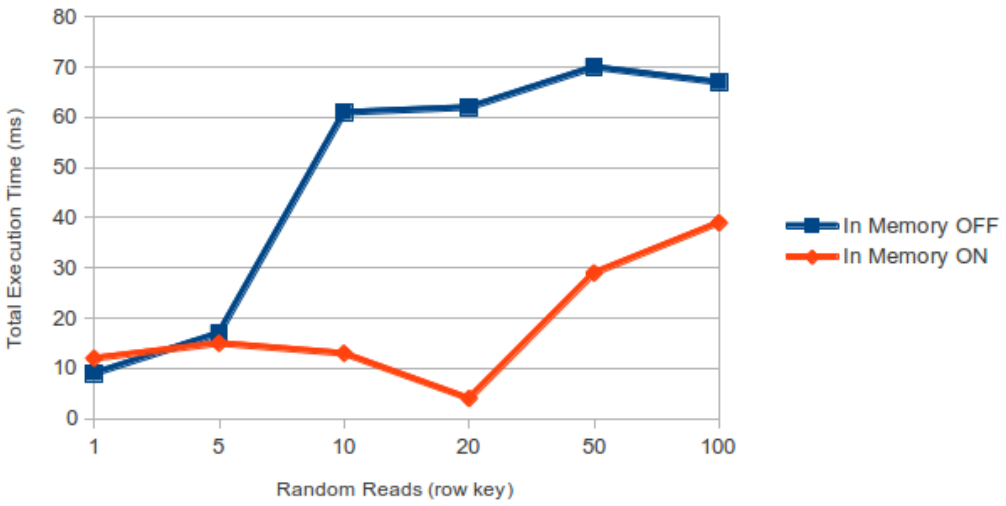

Figure 4.2: Random row keys Read with and without In-Memory for QoS system for clouds in hypothetical approach [57].

Figure 4.3 characterizes In-Memory behavior. The total execution time gets reduced if more than 5 rows are retrieved. If we just read once and our requested data is not within the same block, it will be difficult to see how block cache helps, but in a scenario were we would be continuously retrieving 25/50 or more row keys per time this feature would be helpful as there will be data within caches and they will not be empty.

- Tuning QOS-File's block size:

QOS-File are the actual QOS storage files. Each one is composed of blocks which are the smallest unit of data QOS reads and places in the block cache. These blocks store key/value pairs and have a minimum size, which is by default 64KB. To achieve better performance, we can be modified its minimum block size. If we want to improve random reads, we should decrease this value to avoid too many key/value pairs within each block, because the read operation always loads entire blocks and subsequently it looks inside the block for the key/value. Setting it to a lower value will decrease the amount of data fetched by each seek operation, thus decreasing IO and time needed for de-compression. On the other hand, it will require more memory to hold the block index, now bigger due to the raise in the number of blocks. The resource consumption patterns of the first cloud service provider is depicted in the Figure 4.3.

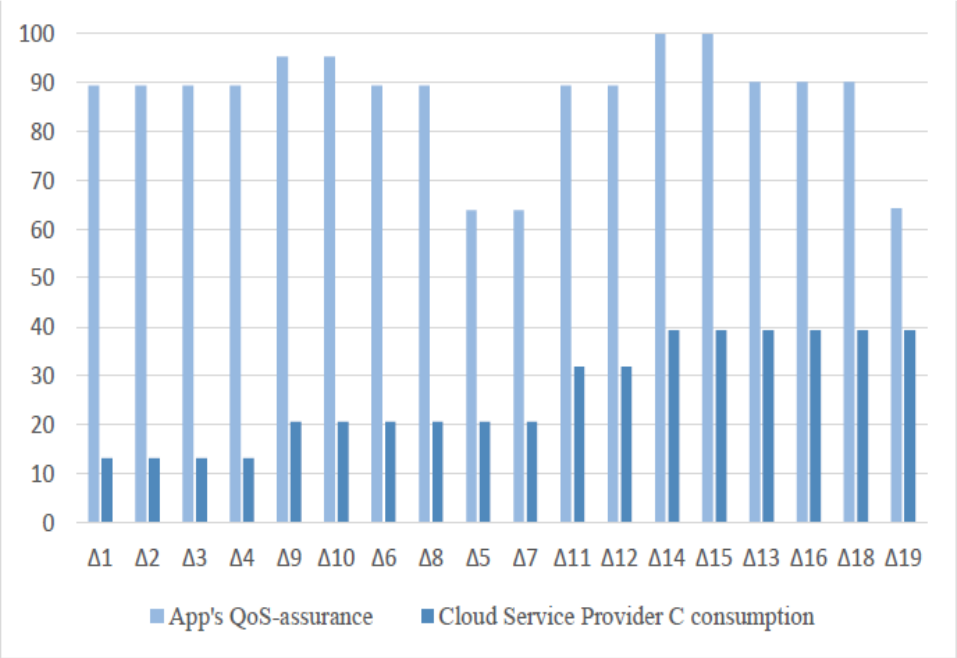

Figure 4.3: The resource consumption of the first cloud service provider [45].

64KB turned out to be good enough when dealing with our predefined random reads. Testing it with bigger values (128KB, 256KB) was increasing the latency as expected. In contrast, testing with smaller blocks (32KB, 16KB or 8KB) gave out same or even worse results as with the 64KB value due to QOS is ultimately bound by disk read latency causing bottlenecks when fetching data from disks.

- Bloom Filters:

QOS supports Bloom Filter [11]. Bloom Filters are a space-efficient and constant-time mechanism to figure out whether an QOS-File/store-File stores a specific row, row-Col cell or not without loading the entire file and scanning the block. They avoid the process of going through each store-File's block index, which has the start row key of each block inside it, to check whether the row can be there or not, and if it does, then QOS needs to load the block and start scanning it in order to confirm if the row is there or not. The drawback of using Bloom Filter is that it needs to be stored within the QOS-File and consequently, QOS-File's size will be boosted.

4.2. LOAD, ACUTE LOAD AND CHRONIC LOAD IN CLOUD QOS.

Types of Load in which the Quality of Service are explained here below:

4.2.1. Load

The amount of framework resources utilized by applications, typically communicated as the proportion of the utilized part to the all-out asset. Productive use/use implies that applications utilize the assets dispensed to them as completely as possible. Quality of service allude to a specific dimension of execution and accessibility of an administration. It likewise covers different viewpoints which are outside the extent of this paper, for example, security and reliability. Service level agreement (SLA), a progression of objectives got through consulting between specialist co-ops and clients. Its motivation is to accomplish and keep up a particular QOS. Commonplace parameters of distributed computing SLA incorporate [67] CPU and memory limit, asset extension speed and authorizations, asset accessibility and dependability, application reaction time, correspondence postponement, and security and protection. It likewise characterizes the punishments that ought to be forced when somebody abuses the pertinent terms as shown in the Figure 4.4.

Load just suggests doing stack adjusting, for example disseminating remaining burden over numerous servers or server farms, organize joins, focal preparing units, plate drives, on the cloud, for example inside the Internet itself. This technique, as other burden adjusting strategies, accomplishes ideal asset usage, expand throughput, limit reaction time, and maintain a strategic distance from over-burden of assets.

Load Balancers enable you to convey the outstanding load and parity it between at least two Cloud Servers. You can thusly shape your framework to enable it to meet action spikes, improve the assignment of assets and guarantee a negligible reaction time. Some features are:

- Ensured Service progression

- Handle high traffic
- Be set up for abrupt solicitation spikes

4.2.2. Acute Load

The infrastructure supplier automatically limits portion issue emerging at the supplier side includes choosing the ideal arrangement of running applications on an appropriate number of VMs, which thusly must be executed on fitting physical servers [67].

With the multiplication of private and open cloud server farms, today is very regular to rent virtual machines to have applications rather than physical machines. Cloud clients regularly pay for a statically designed VM measure, regardless of the genuine assets utilization of the application (e.g., Amazon EC2). This charging mode is clearly absurd particularly for applications with variable burden. It is generally hard for cloud clients to make sense of which size of VM is appropriate for their applications as their heaps are infrequently steady. Solicitation load adjusting is an inexorably upheld include of cloud contributions. A heap balancer dispatches demand from clients to servers as per a heap dispatching strategy. This process is for framework supplier load adjusting [67]. In the foundation client load adjusting, the heap balancer is introduced and oversaw straightforwardly by the cloud supplier. A cloud client can likewise choose to introduce its own heap balancer for cloud application. This might be useful in mutually following limit designation and burden adjusting.

4.2.3. Chronic Load

The chronic load balancing is versatile and dynamic load adjusting is actualized notwithstanding assess in a two hub disseminated framework. The exhibition of the proposed dynamic load adjusting arrangement is contrasted with that of static approaches just as existing unique load adjusting strategies by taking into account the normal consummation time per task and the framework preparing rate in the presence of arbitrary landings of the outside loads. So Here by we reason that with the expansion in number of clients there is issue of time defer that emerges. It would be exceptionally attractive for cloud suppliers to give progressively better grained online adaptable administrations that assign assets as per application's interest that could urge the client to pay a more expensive rate for better administration contrasted with paying a level charge dependent on the measure of their VMs.

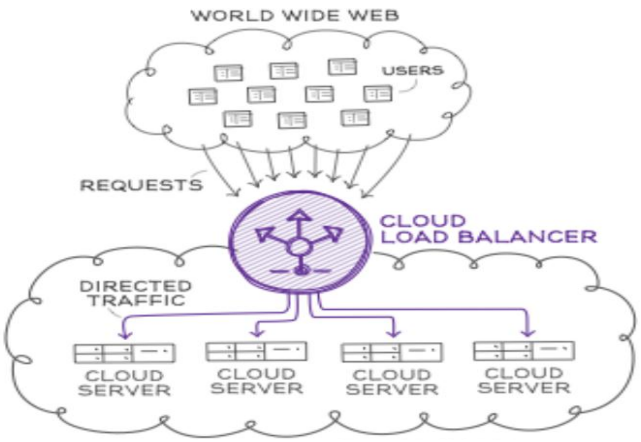

Figure 4.4: Load & auto load Infrastructure in the cloud provisioning setup. [66]

This Graph below shows that how load, acute load and chronic load utilize their adaptability to progressively enhance the assets portion, improve assets usage, and accomplish most extreme advantage. Significantly the represents the load they handle to outperform the cloud resources and improve the consumption of users and they data flow. The Paying order improve also with the types depend what we user. They surely don't care to pay for the assets they hold however not utilize when load is light. This Load concept is related to the cloud servers where the client access to its machine online sees its resources. What's more, so as to productively adapt to the extreme variety of the fame of the substance, we actualized dynamic video substance the framework which handles the heap adjusting by checking of the conditions of the servers continuously in real time cloud servers as shown in figure 4.5.

Nevertheless, Bloom filters reduce the number of unnecessary block loads, which translate into an improvement in the overall throughput of the entire cluster and that is the reason why we use them and Quality of Service assurance between specific Time of period in which the clouds were evolved as a service as shown in figure 4.6.

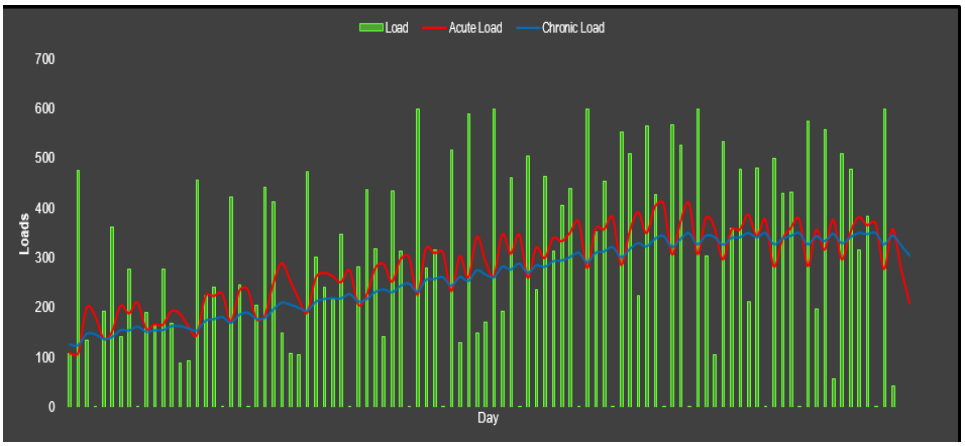

Figure 4.5: Measuring the Types of Load in which the Quality of Service operates.

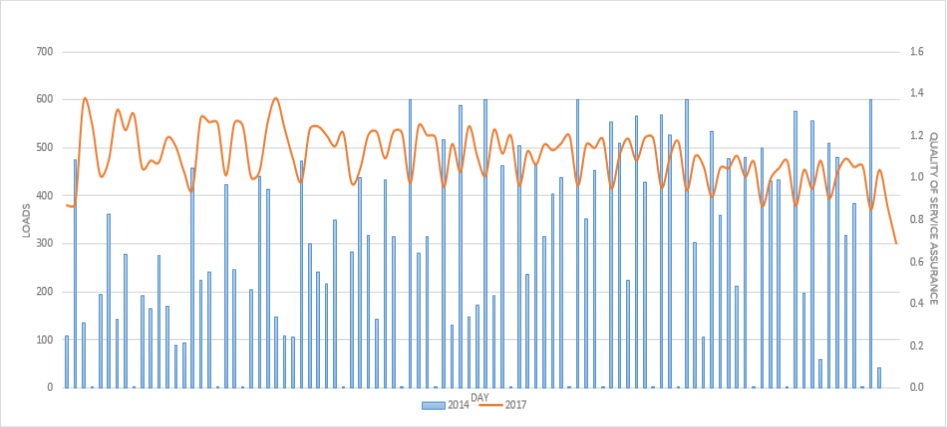

Figure 4.6: Quality of Service assurance between specific Time of period in which the clouds were evolved as a service.

5. DISCUSSION

In this final project we present methods related with scaling-out the data of a commercial company. In order to improve the performance, we implement QOS cluster for ultimate performance of networked cloud, a Cloud-based datastore, along with solutions based on Big Data algorithms, such as MapReduce.

This project evaluates the obtained performance from three different points of view: Firstly, the main problem of importing a really big dataset to a new Cloud-based datastore. Several approaches have been developed and carefully tested, uncovering its benefits and drawbacks in order to improve the obtained approach. Secondly, the performance of reading random data in a write-optimized database like QOS. Once more, conceptual ideas have been developed and tested and the results have been exposed. Finally, the tuned QOS cluster has been benchmarked against a QoS cluster similar to the one the company where the data comes from uses.

We have been able to improve the default performance in every area. Importing data, we have passed from an API client whose execution time is more than five hours to a MapReduce-based client which enables us to reduce the processing time to only nine minutes. There are lot of improvements behind these simple numbers, such as HDFS issues, skew data, compression, JVM issues, etc. As of retrieving random data, we have also improved default results by using concepts such as Bloom filters, QOS-File's block sizes or block caches. All obtained results have been studied and improved when possible. Setting aside the results, one main tool has been developed not only to fit the uneven data issue our dataset has, but also every MapReduce/QOS job suffering from skew data in the mapper outputs. In a brief way, it samples the whole dataset in a light-weightly way with a confident level defined by the user and returns the best split points which removes the uneven distribution of mappers output.

The conducted benchmarks show how our tuned QOS cluster performs against a QoS cluster for Cloud based System. Three main scenarios are developed and the outcomes are discussed. QOS outperforms in writes and is really close to MySQL in random reads. It is worth to state we have improved the Yahoo! Cloud Benchmark Tool by developing some tools to overcome pitfalls already presented in its solution and thus letting us enhance QOS results (not hack them, but get closer results to real scenarios).

We can conclude that we have achieved enough good results as to change the company datastore backend system to QOS.

The quality of service parameter for cloud computing will be evaluated in terms of performance,

response, reliability, cost, and security [63]. The results of the evaluation have provided resource consumption, performance characteristics and application quality behavior. Cloud service provider B performed very well compared to the other two with the CSP C performing poorly especially in the heuristic ranking. In terms of application QoS assurance, the CSP has a 100% guarantee to the quality in comparison to the less than 50% resource consumption by the application that was determined. The CSP C has the largest resource consumption while CSP B had the lowest resource consumption. The implementation of the cloud deployment scheme has shown that most cloud service provider guarantees QoS and they can support some applications in their platforms.

In terms of performance, the CSPs offered different solutions that are vital for the IT need of a business such as the IaaS, PaaS and SaaS. The solution offered has accuracy in response time, stability, suitability to the business needs and interoperability. The provision of QoS I these technologies ensure the satisfaction of the users. It is also evident from the investigation as there are security and privacy of data. Each application was segregated from the rest to allow the access of data and resource via their channel. The hosting requirement led to the application of stringent security policies to alleviate breaches and access from unauthorized personnel. The cloud service providers evaluated above have also shown the usability of their cloud services which is significant in ensuring change-on-demand. The usability of a cloud service allows an organization to utilize its cloud resource while maintaining accessibility, operability, and installability.

6. CONCLUSION

In this chapter we discuss opportunities for improvement in Section 6.1 and review the work done for the final thesis project.

The virtualization of computing resources has been associated with an increase in business performance and fast availability of these resources to an organization via the cloud services. This paper presents a quantitative evaluation of QoS of three cloud providers evaluating the availability of resources, performance, and usability of the service. The initial review illustrated that researches had been done to ensure the improvement of the quality of service in cloud service. The parameters suggested in these researches were resource consumption, power requirements, data accessibility, and the improvement of response time. The evaluation in this paper thus answers some of these questions but little has been done to determine the performance of the CSP during peak hours. This research will encourage future work to be undertaken to determine QoS during increased traffic in the cloud service provider.

6.1. FUTURE WORK

Dealing with data, no matter whether it is import or retrieval operation, has been carefully studied and discussed. Nonetheless, some improvements not tested arise here:

- Followed scheme design for our QOS table has proven to work well. However, we may consider a redesign of it. A schema with only one column-Family would be beneficial as we would have a better control over the QOS behavior; easier manage of store-Files, reads/block caches and similar opportunities derivated from having only one column-Family.

- Dataset has lot of duplicates elements. By now, we just import them, it does not matter whether they are already stored or not. Nonetheless, we could create a combiner class to get rid of duplicates in the mapper side. It would reduce the amount of IO operations between mappers and reducers and would reduce the total execution time of the job. The drawback would be that only one version of each element would be stored in the database.
- In QOS, it is possible for a client to read directly from disk instead of going through the DataNode. This action is called a short-circuit read. Region servers read directly off the local node data disks instead of asking the DataNode for the data. This feature has been tested to work well, with little or no drawbacks, hence we could use it instead the default QOS built-in read behavior.

- Relationship between data disks and Hadoop / QOS ecosystems has been and continuous being the focus of a lot of research activity [49] [28] [7]. Researcher Shrinivas B. Joshi points out the advantages of using more than one data disk in Hadoop workloads (achieved more than 50 % performance improvement) [47]. It is well-known Hadoop performance scales with the number of available data disks, however, we were not able to check it owing to our hardware boundaries, but it may be worth trying it out.

- It is well-known that QOS communication stack does not work correctly when using high performance networks like InfiniBand because of its implementation based on Java Sockets Interface that provides non-optimal performance due to the created overhead [52].

  Although we are using Gigabit Ethernet for our experiments, Triton cluster provides InfiniBand network communication. Therefore, we could harness it by using the novel desing of QOS that Jian et al. have done for their research, an QOS fully InfiniBand compatible [42]. They claim to have achieved a factor of 3.5 improvement over 10Gigabit Ethernet network latency when retrieving data (Get operations).

- HOYA[1]: It is a YARN application[2] that provisions Region Servers based on an QOS cluster configuration, it can be used to spin up temporary QOS clusters during MapReduce or other jobs. HOYA could be useful for our interests due to it would help us to spin up more QOS resources during heavy batch workloads such as night import of new data. It will allow us to create on-demand QOS resources and thanks to it, we will be able to utilize cluster resources better.

REFERENCES

[1] Abadi, D. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. Computer 45, 2 (2012), 37–42.

[2] Agaoglu, E. LZO vs Snappy vs LZF vs ZLIB, A comparison of compression algorithms for fat cells in QOS, 2013.

[3] Agrawal, D., Das, S., and El Abbadi, A. Big data and cloud computing: New wine or just new bottles? Proceedings of the VLDB Endowment 3, 1-2 (2010), 1647–1648.

[4] Aiyer, A. S., Bautin, M., Chen, G. J., Damania, P., Khemani, P., Muthukkaruppan, K., Ranganathan, K., Spiegelberg, N., Tang, L., and Vaidya, M. Storage Infrastructure Behind Facebook Messages: Using QOS at Scale. IEEE Data Eng. Bull. 35, 2 (2012), 4–13.

[5] Amazon.com Inc. Amazon Web Services web page, 2013

[6] Ananthanarayanan, G., Kandula, S., Greenberg, A. G., Stoica, I., Lu, Y., Saha, B., and Harris, E. Reining in the Outliers in Map-Reduce Clusters using Mantri. In Symposium on Operating Systems Design and Implementation (OSDI) (2010), vol. 10, p. 24.

[7] Awasthi, A., Nandini, A., Bhattacharya, A., and Sehgal, P. Hybrid QOS: Leveraging Flash SSDs to improve cost per throughput of QOS.

[8] Babu, S. Towards automatic optimization of mapreduce programs. In Proceedings of the 1st ACM symposium on Cloud computing (2010), ACM, pp. 137–142.

[9] Baranau, A. Configuring QOS Memstore: What You Should Know, July 2012.

[10] Bernstein, P. A., and Goodman, N. Multiversion concurrency control theory and algorithms. ACM Transactions on Database Systems (TODS) 8, 4 (1983), 465–483.

[11] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13, 7 (1970), 422–426.

[12] Borthakur, D. HDFS Architecture, June 2012.

[13] Brewer, E. CAP twelve years later: How the ”rules” have changed.Computer (2012), 23–29.

[14] Brewer, E. A. Towards robust distributed systems. In Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (2000), ACM, p. 7.

[15] Burrows, M. The Chubby lock service for loosely-coupled distributed systems. In Proceedings of the 7th symposium on Operating systems design and implementation (2006), USENIX Association, pp. 335–350.

[16] Cattell, R. Scalable SQL and NoSQL data stores. ACM SIGMOD Record 39, 4 (2011), 12–27.

[17] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach,D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber,R. E. Big-Table: A distributed storage system for structured data. ACM Transactions

on Computer Systems (TOCS) 26, 2 (2008), 4.

[18] Cheng, P., and An, J. The Key as Dictionary Compression Method of Inverted Index Table under the QOS database. Journal of Software 8, 5 (2013), 1086–1093.

[19] Cloudera, Inc. CDH web page, 2013. http://www.cloudera.com/ content/cloudera/en/products/cdh.html. Accessed 1.4.2013.

[20] Cloudera, Inc. Cloudera web page, 2013. http://www.cloudera.com/.Accessed 1.4.2013.

[21] Codd, E. F. A relational model of data for large shared data banks. In Pioneers and Their Contributions to Software Engineering. Springer, 2001, pp. 61–98.

[22] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., and Yerneni, R. PNUTS: Yahoo!'s hosted data serving platform. Proceedings of the VLDB Endowment 1, 2 (2008), 1277–1288.

[23] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (2010), ACM, pp. 143–154.

[24] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost,C., Furman, J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., et al. Spanner: Google's globally-distributed database.

[25] CoudBees Inc. CloudBees web page, 2013. http://www.cloudbees. com/. Accessed 11.6.2013.

[26] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. Communications of the ACM 51, 1 (2008), 107–113.

[27] Dropbox Inc. Dropbox web page, 2008. http://www.dropbox.com.Accessed 11.6.2013.

[28] Fan, B., Tantisiriroj, W., Xiao, L., and Gibson, G. DiskReduce: RAID for Data-Intensive Scalable Computing. In Proceedings of the 4th Annual Workshop on Petascale Data Storage (2009), ACM, pp. 6–10.

[29] Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica. Above the clouds: A Berkeley view of cloud computing. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS 28 (2009).

[30] Gantz, J., and Reinsel, D. The digital universe in 2020: Big Data, bigger digital shadows, and biggest growth in the far east. IDC iView: IDC Analyze the Future (2012).

[31] George, L. QOS: The Definitive Guide. O'Reilly Media, Inc., 2011.

[32] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google file system. In ACM SIGOPS Operating Systems Review (2003), vol. 37, ACM, pp. 29–43.

[33] Gilbert, S., and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News 33, 2 (2002), 51–59.

[34] Gilbert, S., and Lynch, Perspectives on the CAP theorem.Computer 45, 2 (2012), 30–36.

[35] GNU project. GZip web page, 2013. http://www.gnu.org/software/ gzip/. Accessed 25.7.2013.

[36] Google Inc. Google Apps web page, 2008. http://www.google.com/ enterprise/apps/business/. Accessed 11.6.2013.

[37] Google Inc. Google Cloud Platform web page, 2013. https://cloud. google.com/. Accessed 11.6.2013.

[38] Google Inc. Snappy web page, 2013. https://code.google.com/p/ snappy/. Accessed 25.7.2013.

[39] Gray, J., and Reuter, A. Transaction processing. Kaufmann, 1993. [40] Heger, D. Hadoop performance tuning-A pragmatic & iterative ap-proach. CMG Journal (2013).

[40] Heroku Inc. Heroku web page, 2013. http://www.heroku.com/. Accessed 11.6.2013.

[41] Huang, J., Ouyang, X., Jose, J., Wasi-ur Rahman, M., Wang,H., Luo, M., Subramoni, H., Murthy, C., and Panda, D. K. High-performance design of QOS with RDMA over InfiniBand. In Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International (2012), IEEE, pp. 774–785.

[42] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. Zookeeper: Wait-free coordination for internet-scale systems. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (2010), vol. 8, pp. 11–11.

[43] HyperTable Inc. HyperTable web page, 2012. http://hypertable. org/. Accessed 25.5.2013.

[44] Jiang, Y. QOS Administration Cookbook. Packt Publishing, 2012.

[45] Jin, H., Ibrahim, S., Bell, T., Gao, W., Huang, D., and Wu, S.Cloud types and services. In Handbook of Cloud Computing. Springer, 2010, pp. 335–355.

[46] Joshi, S. B. Apache Hadoop performance-tuning methodologies and best practices. In Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering (2012), ACM, pp. 241– 242.

[47] Junqueira, F. P., Reed, B. C., and Serafini, M. Zab: Highperformance broadcast for primary-backup systems. In Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on (2011), IEEE, pp. 245–256.

[48] Kang, S.-H., Koo, D.-H., Kang, W.-H., and Lee, S.-W. A Case for Flash Memory SSD in Hadoop Applications. International Journal of Control and Automation 6, 1 (2013), 201–210.

[49] Kwon, Y., Balazinska, M., Howe, B., and Rolia, J. Skewtune: Mitigating skew in MapReduce applications. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (2012), ACM, pp. 25–36.

[50] Kwon, Y., Ren, K., Balazinska, M., Howe, B., and Rolia, J.Managing Skew in Hadoop. IEEE Data

Eng. Bull 36, 1 (2013), 24–33.

[51] Lai, E. Computerworld: No to SQL Anti-database movement gains steam, 2009. http://www.computerworld.com/s/article/9135086/. Accessed 4.8.2013.

[52] Lipcon,T.Avoiding full GCs in Apache QOS with MemStore-Local Allocation Buffers,April 2012

[53] Lipcon, T. Avoiding full GCs with MemStore-Local Allocation Buffers Cloudera Presentation, February 2012.

[54] Liu, X., Han, J., Zhong, Y., Han, C., and He, X. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS. In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on (2009), IEEE, pp. 1–8.

[55] Medeiros, A. HAcid: A lightweight transaction for QOS. Master's thesis, Aalto University, Dept. of ICS, Finland, 2012.

[56] Mell, P., and Grance, T. The NIST definition of cloud computing.NIST special publication 800 (2011), 145.

[57] Meyn,O.Optimizing,Writes-QOS,July2012.http://gbif.blogspot.fi/2012/07/optimizingwrites-in-QOS.html. Accessed 11.7.2013.

[58] MicrosoftInc.Windows Azure web page, 2013.

[59] Oberhumer, M. LZO real-time data compression library. User manual for LZO version 0.28, URL: http://www. infosys. tuwien. ac. at/Staff/lux/marco/lzo. html (February 1997) (2005).

[60] O'Neil, P., Cheng, E., Gawlick, D., and O'Neil, E. The logstructured merge-tree (LSM-tree). Acta Informatica 33, 4 (1996), 351–385.

[61] Oracle Inc. MySQL web page, 2008. http://www.mysql.com. Accessed 11.6.2013.

[62] Padhy, R. P., Patra, M. R., and Satapathy, S. C. RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's. International Journal of Advanced Engineering Science and Technologies 11, 1 (2011), 15–30.

[63] Peter Mell, Timothy Grance "The NIST Definition of Cloud Computing", NIST Special Publication 800-145, 2011

[64] Sukhpal Singh, Inderveer Chana, "Q-aware: Quality of service based cloud resource provisioning", Computers and Electrical Engineering, vol. 47, pp. 138–16, 2015

[65] https://www.globaldots.com/knowledge-base/cloud-load-balancing/cloud_balancer/

[66] Kumar, Pawan & Kumar, Rakesh. (2019). Issues and Challenges of Load Balancing Techniques in Cloud Computing: A Survey. ACM Computing Surveys. 51. 1-35. 10.1145/3281010.

[67] Nayak, Suvendu Chandan, and Chitaranjan Tripathy. "Deadline sensitive lease scheduling in cloud computing environment using AHP." Journal of King Saud University-Computer and Information Sciences (2016). Online Available: https://www.kaggle.com/pitasr/scheduling-in-cloud-computing/downloads/scheduling-in-cloud-computing.zip/2

[68] Kwon, Y., Balazinska, M., Howe, B., and Rolia, J. Skewtune: "Get IT Support and Enjoy Secure Cloud Computing Services applications for increase quality of service" Online available: http://newyorkcloudcomputing.wordpress.com/network-security/dataset/

[69] H. Yaseen , O. UCAN , O. BAYAT , " Integration of the Quality of Service in Networked Cloud Computing System" in AURUM-Mühendislik Sistemleri ve Mimarlık , pp1-12, 2019, Submitted .