ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

DNS BIG DATA PROCESSING FOR DETECTING CUSTOMERS BEHAVIOUR OF ISP USING AN OPTIMIZED APACHE SPARK CLUSTER

M.Sc. THESIS

Yousef ALKHANAFSEH

Department of Electrical and Electronics

Electrical Engineering Programme

JANUARY 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

DNS BIG DATA PROCESSING FOR DETECTING CUSTOMERS BEHAVIOUR OF ISP USING AN OPTIMIZED APACHE SPARK CLUSTER

M.Sc. THESIS

Yousef ALKHANAFSEH
(504191100)

Department of Electrical and Electronics Engineering

Electrical Engineering Programme

Thesis Advisor: Assoc. Dr. T. Çetin AKINCI

JANUARY 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

İSP MÜŞTERİLERİN DAVRANIŞLARINI TESPİTİ İÇİN OPTİMİZE EDİLMİŞ BİR APACHE SPARK KÜMESİ KULLANARAK DNS BÜYÜK VERİ İŞLEME

YÜKSEK LİSANS TEZİ

YOUSEF ALKHANAFSEH
(504191100)

Elektrik-Elektronik Mühendisliği AnaBilim Dalı

Elektrik Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Tahir Çetin AKINCI

OCAK 2022

YOUSEF ALKHANAFSEH, a **M.Sc.** student of ITU **Graduate School** student ID 504191100, successfully defended the the-sis entitled **"DNS BIG DATA PROCESSING FOR DETECTING CUSTOMERS BEHAVIOUR OF ISP USING AN OPTIMIZED APACHE SPARK CLUSTER"**, which he prepared after fulfilling the requirements specified in the associated legisla-tions, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Tahir Çetin AKINCI** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Salih Barış ÖZTÜRK** Istanbul Technical University	.....
	Assist. Prof. Dr. Hüseyin YÜCE Marmara University	.....

Date of Submission : 18 January 2022
Date of Defense : 03 February 2022

To my dear family,

FOREWORD

The trigger of conducting such a research is primarily stemmed from my passion to enhance my expertise and to try to develop better methods for analyzing big data such as writing my own software modules. As the size of data is getting bigger with time and the world moves further into complex technologies, my research gives me an advantage to keep up with the world progress. The demand for data increases day by day and there will be a greater demand for analyzing these huge amounts of data. How will we process this amount of data in a short time and get the maximum benefit from it with high accuracy? I have an impetuous passion not only to learn, but also to improve tools and modules that can help future generations.

In truth, I could say that my initial goals of my plan to be professional in this area are achieved. However, there is still a lot to learn . First of all, I would to thank my parents, May God have mercy on them, who were a substantial source of encouragement and financial support during my master level, and my teacher T. Çetin AKINCI who was an understanding teacher and gave me a lot of valuable information during my undergraduate and graduate levels. I would also to thank Turknet telecommunication company whom provided the data of this project. Lastly, I would like to thank my committee members for their patient advice and guidance. Thank you all for your steady support.

JANUARY 2022 **YOUSEF ALKHANAFSEH**
(Electrical Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

DNS	: Domain Name Server
RDD	: Resilient distributed datasets
RAM	: Random Access Memory
vCores	: Virtual Cores
IoTs	: Internet of Things
EB	: ExaByte
YB	: YottaByte
TB	: TeraByte
GB	: Gigabyte
MB	: Megabyte
AWS	: Amazon Web Services
EMR	: Elastic map reduce
HDFS	: Hadoop Distributed File System
ELK	: Elasticsearch, Logstash, and Kibana
YARN	: Yet another resource negotiator
CDR	: Call detail record
CGNAT	: Carrier-grade network address translation
CRM	: Customer relationship management
MAPE	: Mean absolute percentage error
AR	: Autoregression
SARIMA	: Seasonal autoregressive integrated moving-average
VAR	: Vector autoregression
HWES	: Holt winter's exponential Smoothing
MCMC	: Markov chain monte carlo
SQL	: Structured query language
HTML	: Hypertext markup language
URL	: Uniform resource locator
OS	: Operating system
B	: Billion

SYMBOLS

g(t)	:	Growth function
s(t)	:	Seasonality function
h(t)	:	Holiday function
ϵ	:	Error value
a_n, b_n	:	Seasonality function parameters
N	:	Number of observations
n	:	Number of samples
y	:	Actual Values
y'	:	Predicted Values

LIST OF TABLES

Page

LIST OF FIGURES

Page

DNS BIG DATA PROCESSING IN TERMS OF DETECTING CUSTOMERS BEHAVIOUR OF ISP USING AN OPTIMIZED APACHE SPARK CLUSTER

SUMMERY

During the past few decades, technology fields, especially Internet of Things (IoTs), have surpassingly evolved which in turn have contributed to great proliferation of data sources. Unfortunately, at that time, the available data processing tools in terms of variety and advancement were insufficient to analyze that huge data in a reasonable time. They suffered from several problems such as slowness, lack of comprehensiveness, limit size of clusters, high expense. These problems have constituted major obstacles for the progress and achievement in Big data field. Therefore, data has been unemployed for a while. However, when its enormous benefits such as making smart decisions, saving time and cost, monitoring servers, improving performance, minimizing hidden correlations, and providing high quality reports have been closely realized, processing big data started to be prevalent. When dealing with big data, the most famous question that can be asked is "how can big data analysis make the enterprise jobs and business better?". Currently, huge amounts of structured and unstructured data-sets, called as big data, have started to be processed by different types of companies such as telecommunications, software and hardware, marketplaces, social media and so on. The current advanced services, hardware, and software have played an important role in promoting big data processing by making its analysis faster, easier and inexpensive. It is important to know the difference between big data and traditional data sources. The main difference between them can be clearly noticed in data size, types, frequency, capturing speed, and used processing tools. Despite the current advanced technologies, processing ExaByte (EB) or even YottaByte (YB) of data in an efficient way that includes the optimal usage of used system by completely utilizing its precise features is still a challenge and need an expert who has a good mathematical background, knowledge of statistics, and superior experience in this field. Based on that, this thesis aims to provide a comprehensive approach of setting up a system that consists of three different stages which are collecting, processing, and visualizing huge amount of DNS data, daily of 1.3 TB, using an optimized YARN-based Apache Spark cluster. The process is achieved in two different clusters in terms of their place of establishment. The first one was established on cloud by using Amazon Web Services Elastic MapReduce (AWS EMR) and the other one was established on local machines using Apache Ambari. Nevertheless, in this project, just the cloud cluster was discussed and reported in detail. The main goal of the one who was on cloud is to determine the features of needed machines for local cluster. Moreover, it adequately made the understanding of Apache Spark various configurations easier by trying each one of them with different values. Additionally, different structures of Python codes, especially related to Pyspark, were tried in different ways in order to specify the most efficient one. Initially, the thesis starts by stating an extensive introduction that takes into consideration different subjects such as big data concepts, properties, sources, importance, future, limitations, challenges, and processing tools. Moreover, the architecture of the used DNS servers

was thoroughly explained by stating their general purpose and their working principle. Similarly, under the title of data collecting, the project's main big data, DNS, and the other used data-sets, which are Call Detail Record (CDR), Customer Relationship Management (CRM), Carrier-grade Network Address Translation (CGNAT), and IP-Blocks, were distinctly clarified by representing a sample of each one in separate tables. All these data-sets are encrypted and only the concerned authorities can understand its content. Then, an additional data-set that was captured from internet websites was introduced by representing a sample of it. A web scraping method has been talked about as well. There were more than one thousand URLs which can be classified in almost 31 categories including education, games, VPNs, Services, banks, economy, etc. After that, several services that are utilized to process the data such as Apache Spark, Yet Another Resource Negotiator (YARN), Hadoop Distributed File System (HDFS), ZooKeeper, and Hive were briefly investigated by interpreting their importance, working principle, architecture, and main configurations. Meticulously, Apache Spark is the data processing engine in this project. On the other hand, HDFS and Hive were used as general storages to save processed data-sets and metadata, respectively. Zookeeper is a service that is utilized in order to maintain centralized configuration information and provide distributed synchronization. Other services such as AWS EMR and AWS s3 were also used in this project. AWS EMR is a platform that Apache Spark clusters can be built on. AWS s3 is a cloud storage that was temporarily used for saving processed data-sets. Next, based on different factors, the differences between Apache Spark APIs, which are Resilient Distributed Data-set (RDD), Dataframe, and Dataset, were concisely illustrated. Subsequently, a procedure of optimizing a YARN-based Apache Spark cluster was proposed by interpreting the used mathematical equations and giving a detailed example of how to start the object of Apache spark in an optimal way. Both Apache Spark and YARN configurations that are related to application properties, run-time environment and networking, shuffle behavior, compression and serialization, memory management, and execution behavior were extremely elaborated. Next, various experiments of processing data were done by using different cluster sizes that started from small number of machines with a small amount of resources of RAM and vCores to huge ones with high number of machines and large amounts of RAM and vCores. These clusters were optimized based on the previously stated configurations and the values that can be found on both Resourcemanager and Spark admin interface were exactly the same as the calculated ones that are related to the amount of RAM, number of vCores, number of containers, and parallel tasks which in turn confirms the efficient use of the available resources. As a result, about %95 of RAM and CPUs of the clusters were successfully utilized. On the other side, the results of the experiments which contain input data size, number of operations, execution time, and output data size were efficiently reported. Based on these results, a local cluster that has the same features of the most appropriate cluster that was obtained in the experiments, is locally established. After that, the output DNS data was grouped based on specific schema and saved in a compressed format which is Parquet that reduces the size of the data approximately four times. Then, it was transferred to an optimized Elasticsearch cluster which is established in order to make fast queries to the output data and visualize it by using an interactive Kibana dashboard. The Elasticsearch cluster includes one master node and two slave nodes. The indices of Elasticsearch were properly configured and split into small indices. Also, they were defined in a way that only uses needed features which in turn leads to enhance and tune the work of disks. Captured visualizations

have played a major role in determining useful information such as the situation of DNS servers, customers segmentations, distribution of DNS traffic across Turkey neighborhoods, types of customers, most visited categories, most used URLs, and suitable places for advertising. Eventually an application that is based on time siers forcasting was made. A sample of the output data was prepared to be used in a time series forecasting using Facebook Prophet model which were selected after trying several models such as autoregression (AR), Seasonal Autoregressive Integrated Moving-Average (SARIMA) and Vector Autoregression (VAR). However, only the results of Fbprophet model were discussed in this project. The main target of this prediction is defining the density of the used DNS servers, giving information about missed data, and providing approximate information about the future of servers. The models were evaluated by comparing the test data-set with prediction one and calculating its mean absolute error. It was almost %2.49 for Fbprophet. In short, some of this thesis achievements can be concluded as providing solid knowledge about cloud computing systems and big data different processing tools, performing various experiments on different clusters with different sizes and resources, establishing local cluster based on these experiments, transforming daily of 1.3 TB of raw data into meaningful information, and making a system for processing new data continuously. Furthermore, these processed informative DNS data is used in a wide range of fields such as congestion prediction for DNS servers, classifying customers, enhancing content delivery network of some specific websites, running successful market advertising campaigns.

İSP MÜŞTERİLERİN DAVRANIŞLARINI TESPİTİ İÇİN OPTİMİZE EDİLMİŞ BİR APACHE SPARK KÜMESİ KULLANARAK DNS BÜYÜK VERİ İŞLEME

ÖZET

Geçtiğimiz birkaç yıl boyunca, teknoloji alanları, özellikle Nesnelerin İnterneti (IoT), aşırı derecede gelişti ve bu nedenle veri kaynaklarının büyük ölçüde çoğalmasına katkıda bulundu. Fakat, o zamanlarda büyük verinin analiz edilmesi mantıklı değildi çünkü o dönemde mevcut veri işleme araçları çeşitlilik ve gelişme açısından yetersizdi. Bu araçlarda yavaşlık, kapsamlılık eksikliği, kümelerin sınırlı boyutu, ve yüksek maliyet gibi çeşitli sorunlar bulundu. Bu sorunlar, Büyük veri alanında ilerleme ve başarının önünde büyük engeller oluşturdu. Böylece, büyük veriler bir süredir ham ve önemsiz kaldı. Ancak, büyük verilerin taşıdığı avantajlarını; akıllı kararlar oluşturma, zaman kazanma, maliyet düşürme, sunucuların monitor edilmesi, performans artırma, ve yüksek kaliteli raporlar hazırlama gibi avantajlar iyi anlayınca büyük veri işleme yaygınlaşmaya başladı. Büyük veri ile ilgilenirekn sorulabilecek en ünlü soru "Büyük veri analizi kurumsal işlerini nasıl daha iyi hale getirebilir?". Şimdiki zamanlarda, telekomünikasyon, yazılım ve donanım, pazaryerleri, sosyal medya vb. farklı türdeki şirketler tarafından çok büyük miktarlarda yapılandırılmış ve yapılandırılmamış verileri işlemeye başlamıştır. Mevcut gelişmiş hizmetler, donanım ve yazılımlar büyük veri analizini daha hızlı, daha kolay ve ucuz hale getirerek büyük verinin işlemesi teşvik edilmesinde önemli bir rol oynamıştır. Büyük veri ile geleneksel veri kaynakları arasındaki fark ta bilinmelidir. Genel olarak, veri boyutları, türleri, akış frekansları, yakalama hızları, işlemesinde kullanılan araçlardan bu iki verinin aralarındaki temel fark belirlenebilir. Fakat, var olan gelişmiş teknolojilere ragmen, mevcut gelişmiş teknolojilere rağmen, ExaByte (EB) yada YottaByte (YB) verilerinin işlemesinde kullanılan sistemin optimum bir şekilde çalışabilmesinde yani hassas özelliklerini tamamen kullanabilmesinde bir zorluk bulunur ve system üzreinde hem üstün deneyime ve iyi bir matematik,ve istatistik bilgisine sahip olan bir uzman bulunmasıdır. Buna dayanarak, bu tez, optimize edilmiş bir YARN tabanlı Apache Spark kümesi kullanarak 1.3 TB (bir günlük verisi) DNS verisinin toplama, işleme ve görselleştirme olmak üzere üç farklı aşamadan oluşan bir sistemin yaklaşımı sağlamayı amaçlamaktadır. Veri işlemsi kuruluş yeri açısından iki farklı kümede gerçekleştirildi. İlk küme Amazon Web Services Elastic MapReduce (AWS EMR) kullanılarak bulut üzerinde kullanıldı ve diğer küme ise Apache Ambari kullanılarak yerel makinelerde kuruldu. Yine de bu projede sadece bulut kümesi tartışıldı ve detaylı bir şekilde raporlandı. Bulutta olanın asıl amacı, yerel küme için ihtiyaç duyulan makinelerin özelliklerini belirlemektir. Üstelik, Apache Spark ve diğer servislerin çeşitli konfigürasyonların anlaşılmasını her birini farklı değerlerle deneyerek yeterince kolaylaştırdı. Ayrıca, Python kodlarının özellikle Pyspark ile ilgili farklı yapıları farklı şekillerde denenerek en verimli olanın belirlenmeyi fırsatı verdi. İlk olarak tez, büyük veri kavramları, özellikleri, kaynakları, önemi, geleceği, sınırlamaları, zorlukları ve işleme araçları gibi farklı konuları dikkate alan kapsamlı bir giriş ile başlamaktadır. Buna ek olarak, kullanılan DNS sunucu-

larının mimarisi, genel amaçları ve çalışma prensipleri belirtilerek detaylı bir şekilde anlatıldı. Benzer şekilde, veri toplama başlığı altında, projenin ana büyük verisi olan DNS ve kullanılan diğer veri setleri olan Call Detail Record (CDR), Customer Relationship Management (CRM), Carrier-grade Network Address Translation (CGNAT), ve IP Blokları her birinin bir örneği ayrı tablolarda temsil edilerek ve belirgin bir şekilde açıklayarak kavuşturulmuştur. Tüm veriler şifrelenerek kullanıldı ve içeriğini yalnızca ilgili makamlar anlayabilir. Python üzerinde bir web kazıma yöntemi de konuşuldu. Eğitim, oyunlar, VPN'ler, Hizmetler, bankalar, ekonomi vb. dahil olmak üzere neredeyse 31 kategoride üç yüz binden fazla URL sınıflandırıldı. Ardından, AWS EMR, AWS S3, Apache Spark, Yet Another Resource Negotiator (YARN), Hadoop Distributed File System (HDFS), ZooKeeper, Hive gibi verilerin işlemsinde kullanılan birçok servisin önemi, çalışma prensibi, mimari ve ana konfigürasyonlar yorumlanarak kısaca incelenmiştir. Özenle, Apache Spark bu projedeki veri işleme motorudur. Öte yandan, HDFS ve Hive, sırasıyla işlenmiş veri setlerini ve meta verileri kaydetmek için genel depolar olarak kullanıldı. Zookeeper, merkezi yapılandırma bilgilerini korumak ve dağıtılmış senkronizasyon sağlamak için kullanılan bir hizmettir. AWS EMR ve AWS s3 gibi başka servisler de bu projede kullanıldı. AWS EMR, Apache Spark kümelerinin oluşturulabileceği bir platformdur. Bu projecde, AWS s3, işlenmiş veri kümelerini kaydetmek için geçici olarak kullanılan bir bulut depolamadır. Daha sonra, farklı faktörlere dayalı olarak, Resilient Distributed Dataset (RDD), Dataframe ve Dataset olan Apache Spark API'leri arasındaki farklar kısa ve öz bir şekilde gösterilmiştir. Ardından, kullanılan matematiksel denklemler yorumlanarak ve Apache Spark nesnesinin optimal bir şekilde nasıl başlatılacağına dair ayrıntılı bir örnek verilerek, YARN tabanlı bir Apache Spark kümesini optimize etme prosedürü önerildi. Uygulama özellikleri, çalışma zamanı ortamı ve ağ iletişimi, karıştırma davranışı, sıkıştırma ve serileştirme, bellek yönetimi ve yürütme davranışı ile ilgili hem Apache Spark hem de YARN konfigürasyonları son derece ayrıntılı bir şekilde anlatıldı. Daha sonra, az miktarda RAM ve sanal çekirdek kaynağına sahip az sayıda makineden, çok sayıda makineye ve büyük miktarda RAM ve sanal çekirdek kaynağına sahip büyük makinelere kadar farklı küme boyutları kullanılarak çeşitli veri işleme deneyleri yapıldı. Bu kümeler daha önce belirtilen konfigürasyonlara göre optimize edildi ve hem Resourcemanager hem de Spark yönetici arayüzünde bulunabilen değerler, RAM miktarı, sanal çekirdek sayısı, kapsayıcı sayısı formuller ile e hesaplananlarla değerler ile tamamen aynıydı. Sonuç olarak, kümelerin RAM ve CPU'larının yaklaşık %95'i başarıyla kullanıldı. Bu deneylerin sonuçlarında bulunan giriş veri boyutunu, işlem sayısını, işlem süresini ve çıktı veri boyutunu verimli bir şekilde raporlandı. Bu sonuçlara dayalı olarak, deneylerde elde edilen en uygun küme ile aynı özelliklere sahip yerel bir küme yerel olarak kuruldu. Daha sonra, çıktı DNS verileri belirli bir şemaya göre gruplandırıldı ve verilerin boyutunu yaklaşık dört kat azaltan Parquet formatında kaydedildi. Ondan sonra, bu çıktı veriler üzerinde hızlı sorgulamalar yapabilmek için optimize edilmiş bir Elasticsearch kümesine aktarıldı. Elasticsearh kümesi bir ana düğüm ve iki bağımlı düğüm içerir. Elasticsearch'ün endeksleri hazırlarken uygun şekilde yapılandırılmış ve küçük endekslere bölünmüştür ve bu disklerin çalışmasını iyileştirmeye yol açtı. Son olarak, verileri daha anlamlı bir şekilde göstermek için interaktif Kibana gösterge paneli kullanıldı. Hazırlanan görselleştirmeler, DNS sunucularının durumu, müşteri segmentasyonları, Türkiye mahalleleri üzerinde DNS trafiğinin dağılımı, müşteri türleri, en çok ziyaret edilen kategoriler, en çok kullanılan URL'ler ve reklam için uygun yerler gibi faydalı bilgilerin belirlenmesinde büyük rol oynamıştır. Sonunda çıktı ver-

ilerinden bir örnek üzerinden zaman siers tahminine dayalı bir uygulama yapılmıştır. autoregression (AR), Seasonal Autoregressive Integrated Moving-Average (SARIMA) ve Vector Autoregression (VAR) gibi çeşitli modeller denendikten sonra Facebook Prophet (Fbprohet) modelini zaman siers tahmini yapmak için seçildi. Ancak, bu projede sadece Fbprophet modelinin sonuçları tartışıldı. Bu tahminin ana hedefi, kullanılan DNS sunucularının yoğunluğunu tanımlamak, kayıp veriler hakkında bilgi vermek ve sunucuların geleceği hakkında yaklaşık bilgi vermektir. Modellerin mean absolute error değerleri karşılaştırılarak değerlendirildi, Fbprohet için neredeyse %2.49'du. Kısacası, bu tez başarılarından bazıları, bulut bilişim sistemleri ve büyük veri farklı işleme araçları hakkında bilgi sağlamak, farklı boyut ve kaynaklara sahip kümeler üzerinde çeşitli deneyler yapmak, bu deneylere dayalı olarak yerel küme oluşturmak, günlük 1.3 TB veriyi işlemek ve ondan anlamlı bilgiler çıkartmak, ve sürekli olarak yeni verileri işleyebilen bir sistem yapmak olarak ifade edilebilir. Ayrıca, işlenen DNS verileri, DNS sunucuları için tıkanıklık tahmini, müşterilerin sınıflandırılması, belirli bazı web sitelerinin içerik dağıtım ağının geliştirilmesi, başarılı pazar reklam kampanyalarının yürütülmesi gibi çok çeşitli alanlarda kullanıldı.

1. INTRODUCTION

1.1 Background

The idea of using data is not a recent approach as it is common. Pharaohs, approximately 3100 B.C, used specific buildings, called Nilometer, in the Nile river for the purpose of measuring the level of its water in order to benefit from flooding disasters. When the flood of the Nile is over, lands that have been exposed to floods become fertile. Using the information of Nilometer, they were able to use it as a predict for the future of arable lands, so that, their crops were planted in a productive ways. However, during the past few decades, the unprecedented progress in technology fields leads to a huge data proliferation, both structured and unstructured, which refreshes the idea of using data. Consequently, interests, almost from all kinds of companies, in extracting useful and meaningful information from these data-sets started growing exponentially in order to enhance their services and jobs. Moreover, they started using it for predicting their future plans. At the same time, there are other several factors that had significant impact on increasing the interest on big data such as the low price of RAMs, CPUs, storages, and available platforms on the cloud. The amount of data does not increase steadily, as the days passes the amount of data increases faster than before. For example, within just five years, comparing to the total amount of available data before five years, the amount of data is quintupled [1]. The growing amount of data does not follow a specific pattern and it is expected to double every two years [2]. However, at the beginning since the technology of processing data was new and underdeveloped, processing exabyte or even yottabyte of data requires advanced storages, hardware devices, and software programs which create many challenges. These challenges almost disappeared with the advancement of technology over time. Even though there is available advanced devices, processing big data can only be applied by an expert person. Advances in many fields, such as invention of computers, mobile devices digital sensors, communications, internet, etc., have led to the need for collecting data[3]. Recently, the term of Big Data was firstly appeared when companies such as Google and Facebook started utilizing

their own huge amount of data. [4].

When dealing with big data, the most famous question is "how can big data analysis makes the enterprise jobs and business better?". Big data, generally, is used to specify new solutions and opportunities for the enterprise itself. Therefore, the company's decisions will be more efficient and smarter, which in turn makes those people and another companies who deal with it more comfortable. By benefiting from 50 businesses experiences, based on [5], the usage of big data can be summarized in reducing the cost of operations, making better decisions, and creating new products and services. In some other sources another purposes of big data were stated such as time saving, marketing insights, social media listing, providing immediate reports, analyzing business trends, and monitoring digital devices. In addition, the importance of analyzing big data can be explicitly found in creating transparency, identify possible performance improvements, and minimizing hidden correlations and hidden risks of the company's current jobs. It enables companies to understand the behaviour of their customers and to detect their problems instantly as well [6]. It also plays an important role in enhancing the job of government companies by combating crimes, forecasting weather situation, preventing diseases [7].

Nevertheless, it is important to know the difference between big data and traditional data sources. The main difference between them can be clearly noticed from the first word of Big Data concept, Big. Big data size is roughly bigger than the size of traditional data. Also, the data is distributed in big data and is centralized in traditional data. Moreover, traditional data is almost structured, where Big data can be structured, unstructured, or even semi-structured. Big data frequency and its capture speed is higher than traditional one [8]. The two types of data are wholly dissimilar in the types of systems and tools that are used to process them. Big data requires spacial tools and advanced system must be used to process big data, Unlike traditional data which requires basic tools normal systems. Finally, They differ in the type of used functions to manipulate the data [9].

In the past few years, the available data processing tools were insufficient, in terms of variety and advancement, to analyze data in a reasonable time [10]. The available tools were suffering from several problems such as slowness, lack of comprehensiveness, limit size of clusters, high expense, which in turn were a major obstacle for the progress

and achievement in Big data filed. Now, there are many and various purposeful tools that were introduced for not only processing big data but also for storing, managing, analyzing and visualizing it. However, the current functional big data tools and software programs must be configured carefully in order to achieve the highest efficiency ratio. Optimizing the configurations of Big data tools relies on several factors such as size of the cluster, amount of data, type of used software and tool, and written code functions.

Researches are still conducting over several aspects of processing big data such as enhancing the execution performance, reducing process execution time, minimizing hardware and software costs, implementing the ideal configuration for the used tools, and developing existing tools. This project is conducted on processing almost daily of 1.3 TB of DNS data, one month 2020 September. The procedures of the project are interpreted in such a way that allows anyone who cares about big data to understand and start processing data with a solid knowledge of all big data substantial concepts. The used data storing tools can be stated as Hive, and AWS S3. Hive depends on Hadoop distributed File system (HDFS), and AWS s3 is Amazon's Simple Storage. On the other hand, the main utilized processing and analyzing data tool was AWS Elastic MapReduce (EMR). Several sizes and types of machines were utilized in different sizes of clusters to adopt the most effective ones with high processing speed and low cost. Eventually, a cluster of Elasticsearch, Logstash, and kibana (ELK) have been employed as a visualizing tool. Before using these programs, they were configured in a way that grants the full potential of the used tool.

1.2 Previous Studies and Limitations

A literature study related to big data was closely archived so as to grasp the essence of previous studies and get their approaches. Based on that, it can be concluded that the most researches related to Big Data were established on defining general information about big data and its management and processing tools. However, in recent years, a quite few researches are conducted on how to optimize, analyze, process, and visualize big data efficiently. So, from this point of view the project aims to clarify a big data comprehensive processing model by stating every paramount details. Almost the whole previous studies of Big Data can be split into two main groups. A group that always takes into consideration the main architecture art of big data such as characters,

challenges, and the other group interests in big data processing tools by analyzing and comparing their execution time, enhancements, and limitations.

The first group principally focused on carrying out researches to compare the execution time of Spark and Hadoop. A wordcount experiment was performed on MpaReduce, Pig, and Hive to compare and measure the efficiency of these frameworks [11]. A study, similar to the previous study, was driven by implementing a wordcount test over different data sizes [12]. Another study, using a cluster of one master and seven slaves, covered a brief comparison between Hadoop and Spark system performances by testing them based on PageRank algorithm [13]. Likewise, a search about Spark and Hadoop was done by contrasting their execution time of k-means algorithm with introducing the advantages of in-memory process [14] [15]. A comparison between Hive and Apache Spark was considered in respect of calculating execution time of analyzing household electric power consumption data-set of 2 million records [16].

The fundamental limitation of the previously stated studies is that applying just a word-count test or making a simple measurement of Spark or Hadoop execution time may not give an evident conclusion about the real capacity of these frameworks due to the simplicity of experiments. For instance, making just a count operation using Spark over a data-set will not give the actual ability of the used machine or cluster. Furthermore, surprisingly, they did not discuss a very substantial topic which is optimizing the appropriate configurations of used clusters. Their researchers were limited to just measuring the execution time without mention anything about their prepared configurations.

Meanwhile, the other group often talked about the evolution and the general knowledge of big data architecture, basic structure of a system with its elements. A research called that they were able to make a that is easy to understand by inspecting and rearranging the current big data architectures and by adding their own architecture notations which included data collection, processing, model specification and visualization [17]. One more study, kind of similar to the previous study, also presented the architecture of big data and the architecture of Amazon analytic services [17, 18]. Another research went briefly into describing the relationship between big data and business intelligent management and reviewed a way to manage RBAC ID Management Cloud-Services Orchestration. It tacitly debated a big data ingestion strategy (Sqoop) [19].

Undoubtedly, Researches related to the architecture of big data are explicitly useful. They give subtle meanings and make big data architecture easy to understand and even sometimes to use their stated strategies. even though it is useful sometimes, in most cases, the architecture of big data differs from company to company based on their documentation, type and size of data, financial strength, and goals. So, when a company is willing to apply a big data project, using the available big data architectures may carry various difficulties and challenges, at the same time it could lead to awful results.

1.3 Study Objectives and Contributions

This project is mainly prepared to give an overall idea about Big data, which includes analyzing, performance optimization, processing, and visualizing steps using real data, based on accurate researches and personal experiments. This is specifically achieved by analyzing one month of DNS data which has been taken from ISP DNS servers. The general flow chart of the project is basiclly described in Figure 1.1. As it is well known that practically ever project starts with a question that explicate the outcomes that can be handled by conducting the project. In order to specify the benefits that can be gained from conducting the project, a profound study, based on previous studies with taking another missed information into consideration, must be completed.

After that, when the green light is given for the project to be foreword, firstly the preparation stage comes. Commonly, hardware, software, and costs are allocated in this stage which includes storages, machines, and licensed programs, if required. At this stage, Team size is specified as well.

Then, the process of gathering data from the companys' different sources such as its database and its servers, which is called data collection, must be cautiously accomplished. The contents of DNS data and the other required data-sets are fairly discussed in this stage. DNS data-set in itself is useless unless the other following data-sets are used and joined with it. The first data-set is Call Detail Record (CDR), its original significance, its role in DNS analyzing, and the meaning of its every single column are distinctly explained by representing a sample of it. Equally, another additional data-sets such as Customer relationship management (CRM), Carrier-Grade Network Address Translation (CGNAT), and Internet Protocol Blocks (IPB) are highly evaluated. The

earlier stated data-sets should be gatherd from the companys' database itself, can not be found anywhere else. However, another data-set, which consists of name of sites and their categories, must be scrapped from the available sites on the internet.

The projects' data analyzing part consists of specifying the target schema of the output data. Useless features/columns and observations/rows will be dropped since they will not be used in the future. Actually, another influential steps are explicitly included in this step. For example, determination of data-sets that will be joined together are assigned in this stage. Moreover, the idealistic join order which represents join of small data-sets with each other followed by a join with the big data-set is planned in this stage.

Subsequently, before getting deep into the processing step, the performance of the used clusters should be optimized which helps in reducing the execution time, diminishing the output data size, minimizing the costs, and increasing the potential of machines. Performance optimization highly depends on determining the configurations of the used hardware and software. As a consequence, the most adequate cluster in terms of machine features and machine sizes will be adopted as the identical cluster for this operation. Nevertheless, it should be carefully understand that any misunderstand in configurations may lead to overload which in turn might cause damage to machines.

Data processing part is entirely done by utilizing YARN-based Apache spark clusters that are established on the cloud. Amazon Web Services (AWS) was the cloud of this project. Specifically, AWS EMR, a big data platform, was employed to establish clusters with different sizes and different machines. The features and the sizes of the used clusters next to their execution time are displayed in distinct tables. Besides, the metadata of the output data including number of rows and size of input and output data are exhibited in tables.

On the other hand, the results were initially visualized using Kibana tool, based on local Elasticsearch one master and two slaves cluster. Kibana side can be realized as a user friendly visualization tool that includes many useful charts. A dashboard, with the ability of applying filtering, that includes maps, pie, line, and bar charts that exhibit the meaning of DNS in an obvious way is prepared to get a rapid information from processed DNS data. Another remarkable Python visualization libraries such as Plotly and Matplotlib were utilized as well.

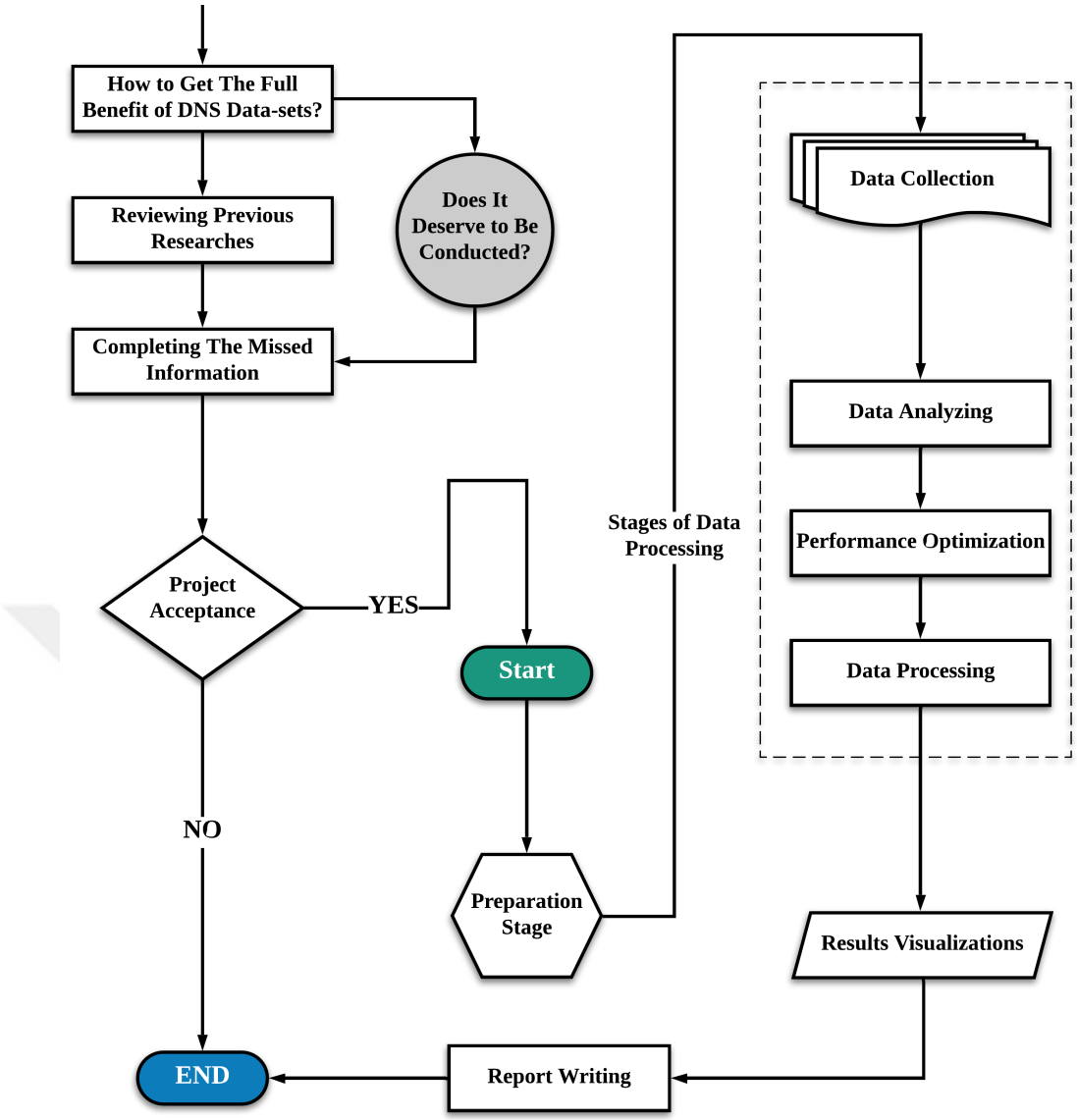


Figure 1.1: Project's General Flowchart.

1.4 Thesis Outline

The flow of thesis chapters, starting from the introduction until the conclusion, can be obviously found in Figure 1.2. The prime contents of each chapter can be stated as following:

Chapter 1 contains a general introduction about big data. In addition, three major subjects which are previous studies and limitations, study objectives and contributions, and thesis outline (This section itself) are plainly formulated.

Cahpter 2, with name of Big Data. introduces Big data prime concepts such as its sources and characters. Moreover, some of Big data multiple challenges and its are

challenges are slightly introduced.

Chapter 3 explains the main server, DNS, that main data captured from. Also, the current architecture of the used DNS is blankly apprised.

Chapter 4 mentions the used data-sets. This chapter does not only represent DNS data-set but also all the other used data-sets which are CDR, CGNAT, CRM, and IP-Blocks. They are briefly explained by showing a three row example for each of them

Chapter 5 purposes a method to get data from open sources websites, which is Beautiful Soup with Python. As it is previously claimed that data from websites should be fetched so that the category of each website can be known.

Chapter 6 shows the most important software paltforms that are used along side with Apache spark. This includes the structure and the work principle of HDFS, YARN, MapReduce, TEZ, Hive, and ZooKeeper platforms. Further, their configurations which promote their potential are characterized in a perspicuous way.

Chapter 7 investigates the used big data process engine, Apache Spark. Its corestone concepts, architecture, and deployment modes can be found in this section. Furthermore, the most functional configurations of Apache Spark that controls the number of cores, the amount of memory, and another network operations are exceedingly described. Finally, our optimized cluster with its parameters are recognized and considered in this chapter.

Chapter 8 clarifies the used cloud, AWS. Particularly, two services of AWS are used in this project which are EMR and S3. Both of them are widely interpreted based on their working principle, structure and costs. Also, their important role in processing and storing the data is smartly elucidated.

Chapter 9 illustrates the obtained results by displaying them in a way that is convenient. Visualizations, which are maps, lines, word cloud, pies, and vertical bar charts, are made by using Kibana visualization tool based on Easticsearch cluster. Other tools such as Plotly and Matplotlib also had a prominent role in visualizing the results.

Chapter 10 provides a method for time series data forecasting. It involves a full description of FBProphet model. A sample of the gained output DNS data is trained and fitted in the model in order to detect the future pattern of DNS devices in terms

of capturing number of observations. At the end, the model is evaluated using Mean Absolute Percentage Error (MAPE).

Chapter 11 ends the thesis with a comprehensive conclusion and recommendations.

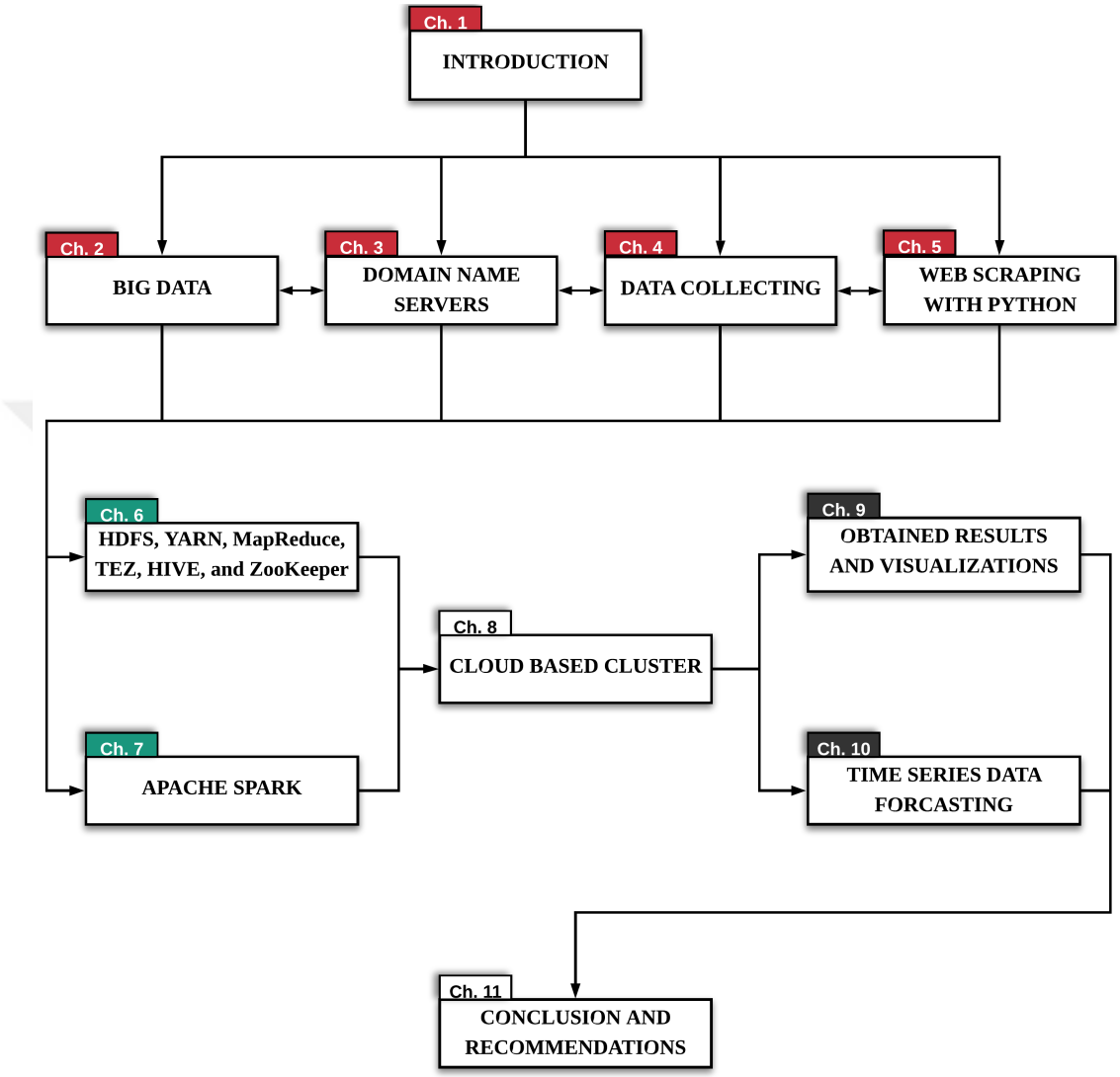


Figure 1.2: Thesis Outline Flowchart.

2. BIG DATA

As early as the appearance of technology, the growth of size of digital data has been staggeringly risen. Since data is tangible, its amount can not be notably imagined. If existing data, until 2011, are collected and printed on pages, these pages can snugly cover US land, about 9.834 million km², 52 times [20]. Over and above what current data size is, approximately 50% of it will be increased per year [21]. Even though the increase of data is tremendous, a lot of companies have been starting processing it. A leading aim of using big data in industries is forming a management revolution [22]. Big data, at first glance, can be thought as a huge amount of data. This definition is relatively true, but the data features, called ten V's of big data which are volume, velocity, variety, veracity, value, Variability, and Visualization must be rigidly stated with the definition of big data. Big data can be captured from different sources. These sources can be summarized as Internet of Things (IoT), Self-quantification, Multimedia, and Social media.

2.1 Sources of Big Data

From the concept of big data it can be explicitly indicated that there are massive and diverse devices that are able to produce data. Four principal sources of big data, which are Internet of Things (IoT), self-quantified, multimedia, and social media data [23], are to be merely stated.

IoT is the first big data source to be discussed. IoT includes technological devices that are embedded with another, small in size, smart devices which are able to make them accessible for other devices on internet. Based on that, sensors, GPS devices, Computers, Mobile phones, washing machines, etc., can be recognized as IoT devices. The project main target is to process a data coming from Domain Name Servers (DNS) which is also an example of IoT. The second source of big data can be considered as Self-quantification data. Generally, this data is captured from medical and body monitoring devices such as blood pressure gauge, and sport Watches. Another source

of big data is Multimedia source. It is contemplated as a great source of data due to its popular usage between people. Images, videos, audios, files, etc., can be found under multimedia notion. Eventually, data of Social media sites such as the data of YouTube, Google, Apple, Samsung, Facebook, and so on is realized a extremely huge source of data.

2.2 Big Data Characteristics

Previously, the important of big data characters on its definition and their effects had been shortly stated. Big data generators/sources characters must meet certain specifications which are called as ten V's of big data (see Figure). These ten Vs can be summarized in one sentence: Big data sources must generate a meaningful (Value) and valid (Validity) data that has a plausible complexity (Viscosity) with reasonable size (Volume) of different types (Variety) within a desired speed (Velocity) that has a feature of activeness (Viability) and can be uploaded to any database with consistent speed (Variability) and are able to be visualized in a brilliant way (Visualization) without affecting the accuracy of the raw data (Veracity).

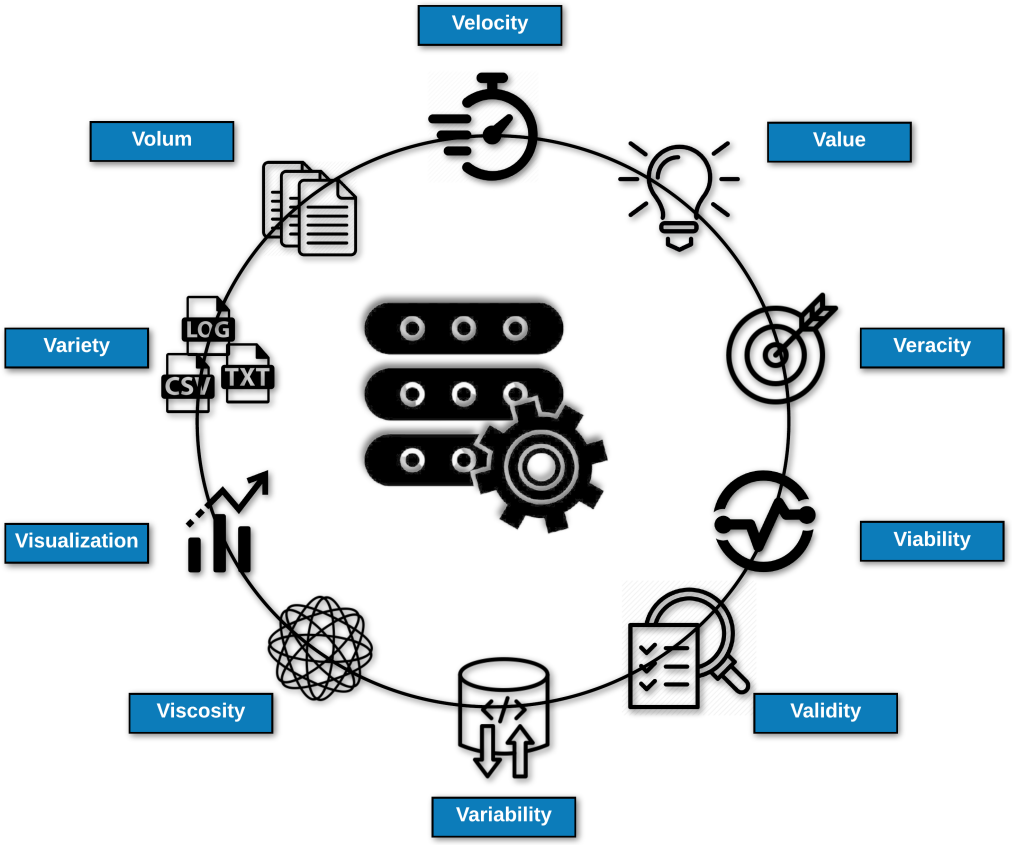


Figure 2.1: Big Data Ten V's

2.3 Big Data Applications and Limitations

Recently, several factors such as available data sources, suitable set-up prices, and developed software technologies have brought interests in big data, which in turn highly promoted big data applications. Its application can almost help in developing the company's accuracy, productivity, efficiency, revenue, and even profitability [24]. By analyzing data coming from customers platforms which contain some specific features such as customers favourite items, opinions about services, reports, difficulties, and so on, can play an important role in guiding the company to enhance its services. Moreover, based on the captured data, effective predictions can be made, thereby enabling company to anticipate the ratio of customers who will leave their services or event the number of people who will join the company services. In addition, data from servers themselves can nearly explicate the performance, current situation, and every detail related to the servers. Therefore, advance steps can be taken before any failure that could strike the servers. The models based on big data, since they are fed with a huge amount of data, are more rigorous that the ones that depend on small data-sets. Moreover, big data can be useful in another sides such as enhancing the job of government companies by combating crimes, forecasting weather situation, analyzing precision small devices, and preventing diseases.

Though, processing Big data actually necessitates a very advanced system. Unfortunately, when a traditional system is used to deal with big data, several limitations that could obstruct the processing deal exceedingly emerge, leading to unpleasant results. The titles of big data limitations can be declared as big data characteristics, needed storage, transport method, management, processing, privacy and security, data access and sharing, and technical challenges. So that, before getting deep into processing big data, its limitations must be closely perceived. Issues related to the characteristics can be summarized in deficiency of handling huge volume of data, inability to analyze movable data, mismanage of different types of data, and small size of the current traditional systems. On the other hand, because of the huge amount of data that are being daily produced by devices, a problems in storing them has been arose. Moreover, transporting, accessing, processing, and managing these amounts of data became a real challenge since they need vigorous networks, sophisticated hardware, advanced software tools, and a lot of additional helpful devices. In other respects, privacy and

security constitutes realistic issues for big data. For instance, the contents of data must be privately protected from any unlawful attack and must not include sensitive and illegal information that might be related to a specific people or to any other private party. Ultimately, while treating big data, some other technical data issues, which can be presented in fault tolerance, scalability, quality of Data, and heterogeneous Data [25], could affect the process in a bad manner.

3. DNS SERVERS

3.1 Background

The acronym DNS can be obviously seen in the title of the thesis. Therefor, it should be clearly explained in order to understand the main source of projects' data. DNS or Domain Name System, in general, can be considered as an intellectual link between humans and computer devices. As it is well known that computers understand the language of numbers, however, it is difficult for humans to memorize numbers, quite the opposite for words. So that, DNS comes here to play a big role in translating words that are written by humans into numbers that are understandable by computers. Specifically, DNS main purpose is that it translates domains to IP addresses. For example, when a domain such as www.something.com is written on the search bar of a browser, then, this domain will be matched with its current IP address, after that, DNS will convert that domain into its private IP address which is in the parttern of ###.###.###.###, eghit bits or one byte are represented in ### . If a human is able to memorize the IP address of sites, he/she can directly enter the desired IP address into the search bar of a browser, it will work. principle Steps of DNS server while resolving a domain is clearly described in Figure 3.1:

1. A client enters a specific domain, considered as www.something.com in this case, on a web browser. The web browser then tries to resolve this domain into its IP address on its own. If the IP address can not be found on the web browser cash memory,

2. it sends a query to the Resolver, or your internet service provider ISP, server which contains a request asking about that domains' IP address. If the Resolver is able to find the IP address of that domain, then the process will be ended successfully. If not,

3. Resolver sends the same query that received from web browser to Root server, which in turn helps it to find the Top Level Domain (TLD) server of that domain, which is .com in this case.

4. Resolver again sends the same query for TLD server asking about the IP address of

that domain. On the other hand, TLD server directs Resolver to Authoritative Name Server.

5. Finally, the query of asking about the IP address of www.something.com comes to its final destination Authoritative Name server, which contains every important detail about domains. That means it will provide the IP address of www.something.com to the Resolver which keep it on its own cash memory and transport it to the web browser.

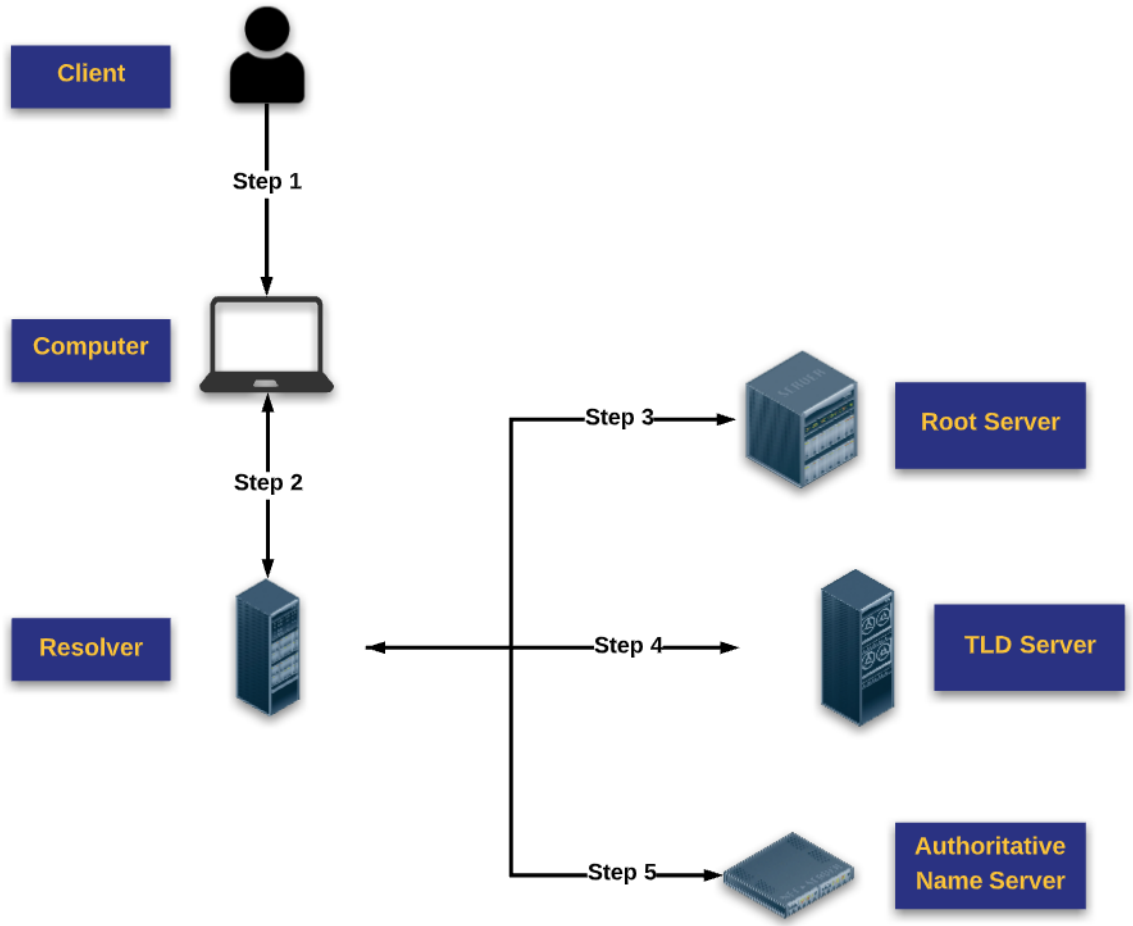


Figure 3.1: Principle Working Steps of DNS Server.

3.2 Used DNS Architecture

Same as the main DNS architecture is used in the ISP that provided DNS data-sets (see Figure 3.2). In particular, this ISP company, uses two DNS servers which are called as DNS1 and DNS2, respectively. Due to the customers requests density, every DNS server is split into smaller servers. For instance, DNS1 is split into DNS-G2, DNS-G3, DNS-G11, and DNS-G12, on the other hand, DNS2 is divided into DNS-E1 and DNS-E2. This division improves health and performance of the servers as well. Modem, in

Figure 3.2, represents the Resolver and Root server is symbolized in the last part of the figure as Root DNS. Movements of clients over internet are taken from these servers and saved in special servers. Consequently, the Big Data of this project is fetched from these servers. The contents of the captured DNS data-sets is comprehensively discussed in the following section.

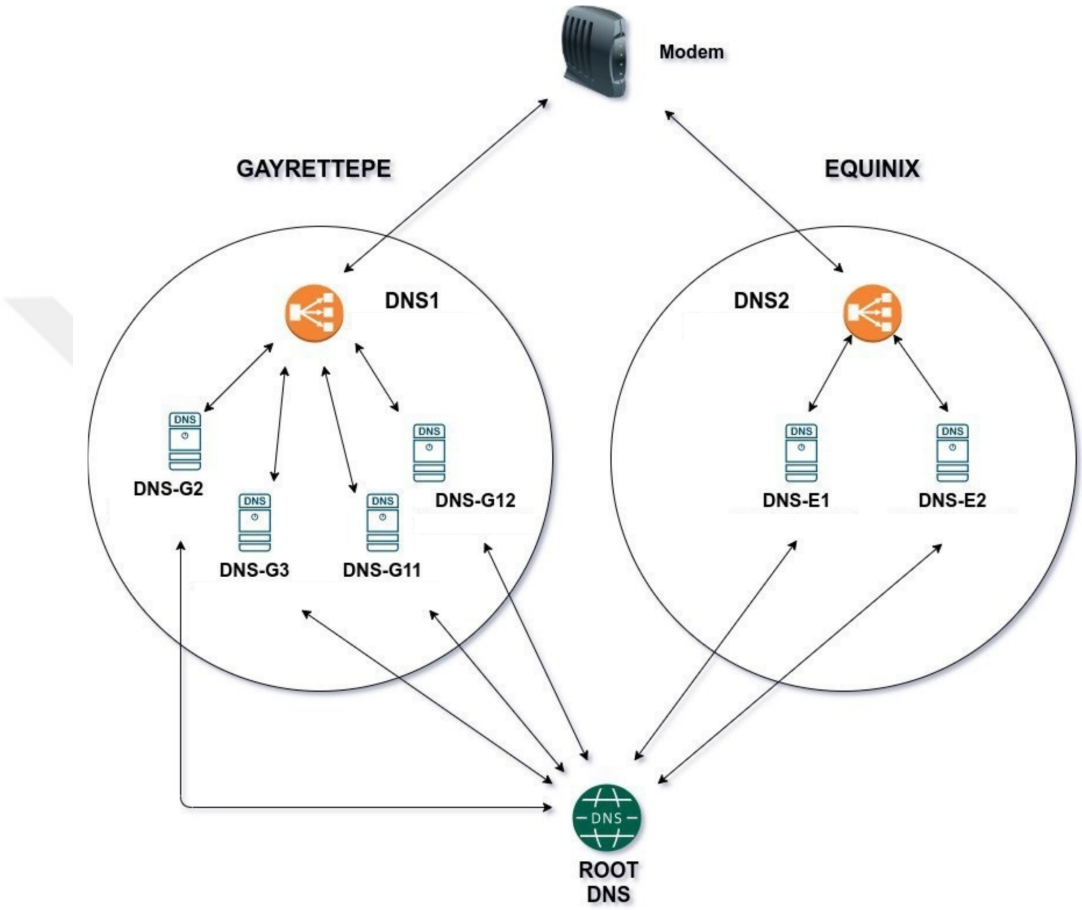


Figure 3.2: DNS Architecture.

4. DATA COLLECTING

Data collection can be succinctly defined as a prime step in data processing that involves gathering essential data-sets from different sources with considering their quality, quantity, and validity. Data ingestion, moving data from its source to the destination storage, is another term that lies under data collection. Moreover, the concept of data aggregation refers to the process of grouping similar observations based on one or several features which in turn reduces the size of data. Data collection step needs to be meticulously done which in turn requires a lot of hard work, patience, and accuracy. The main goal of this step is to ensure that all the required data-sets that are needed to start processing step are gathered in such a way that guarantees a clear instructions and no errors [26].

As mentioned previously that DNS data-set is the big data of this project. A sample of DNS data-set can be found in Table 4.1 below. It is obvious that the initial state of DNS data-set does not make sense which needs to be particularly treated. During this project, it comes in txt format. The meaning of its first row can be stated as follow: Name of month that data captured through, Sep, time of capturing data, 00:07:12, Name of server that captured the data, dns2, the IP and port of the query of the client, queries client 31.223.41.69#4753, and finally the URL of the visited site, query: www.youtube.com.

4.1 Domain Name Server's Data

Table 4.1: Sample of Raw DNS data-set

All Data
Sep 23 00:07:12 dns2 queries client 31.223.41.69#47530 query: www.youtube.com
Sep 23 00:07:12 dns2 queries client 95.70.133.59#39603 query: google.com

4.2 Call Detail Record's Data

In this project, Call Detail Record (CDR) can be defined as data that provides a detailed record of the connection duration of users to the internet or any other telecommuni-

cations equipment. It plays an important role in the field of revenue generation, law enforcement, and Voice over Internet Protocol (VOIP). Table 4.2 represents a sample of three observations of CDR data-set. First feature, which is CustomerID, displays the ID of the client. Second and forth columns illustrate the start and the end of the connection, respectively. Finally, the IP of that customer is entered in the IP column. However, this data must be manipulated since it contains personal observations. Based on that, both CustomerID and IP features were encrypted in a way that just a few workers can understand them.

Table 4.2: Sample of CDR data-set

CustomerID	IP	Start Datetime	End Datetime
PSSONJSK	100.x.x.x	1600775874	1600861674
BNLXKSGJ	100.x.x.x	1600762412	1600766240

4.3 Carrier-grade Network Address Translation's Data

Carrier-grade Network Address Translation (CGNAT) or large-scale NAT (LSN) is an IPv4 network design that enables ISP to maintain its own public IPv4 addresses. In addition, it prevents the ISP customers from using port forwarding, redirecting a request from one address and port number to another address. In this project, a small part of the utilized CGNAT data-set can be obviously noticed in Table 4.3

Table 4.3: Sample of CGNAT data-set

StartDate	EndDate	IPNAT	IPReal	SPort	EPort
1600775874	1600861674	100.x.x.x	88.19.3.97	1xxx	4xxx
1600762412	1600766240	100.x.x.x	18.11.32.47	3xxx	3xxx

4.4 Customer Relationship Management's Data

Customer Relationship Management (CRM) is a software that enables enterprises to be more efficient with their customers. It includes all required strategies, techniques and tools which make certain that the relationship between enterprise and its customers is perfectly going on. In general, CRM data-set contains features that show the personal information of the customers such as their name and surname, address, gender, purchase

history, internet type, activation time, and so on. A sample of CRM data-set is summarized in Table 4.4.

Table 4.4: Sample of CRM data-set

CustomerID	TranType	City	District	Neighborhood
PSSONJSK	FTTH	Istanbul	Avcılar	Cihangir
BNLXKSGJ	YAPA	Sakarya	Akyazi	Pazarköy

4.5 Internet Protocol Address Block's Data

IPBlock data-set can be called as the the final collected data-set in this project. Particularly, it contains the company's' privet blocks of IPs. Table 4.5 offers a small portion of the used CRM data-sets.

Table 4.5: Sample of IP-Blocks data-set

IP Block	IP Type	IP ID
100.73.0.0	CGNAT	58
212.154.2.0	XDSL Static Ip Blokları	21

5. WEB SCRAPING WITH PYTHON

Recently, the proliferation of websites has increased in an unbelievable way. This enormous number of websites constitutes a valuable source of data that can be freely extracted. So, it is implausible to leave these websites without benefiting from their advantages. Also, handily copying and pasting from websites is a choose that should be avoided as it consumes a lot of time. However, The definition of web scraping is based on taking data quickly out from websites. Subsequently, several web scraper methods have been developed so that the access to websites data-sets became simply done. Despite the help of web-scraper methods, some websites such as Twitter prohibits these methods and makes its own API available for programmers to get their data-sets [27]. In this project, conversely, websites data-sets that can be retrieved from scraper methods are scraped. A fresh web-scraper method which is Beautiful Soup Module was primarily utilized in this project.

5.1 Beautiful Soup Module

Python has diverse and simple built-in libraries that encourage programmers to learn web scraping. The most used one is Beautiful Soup library. The first word of the library, beautiful, refers to arrangement of unstructured data. Soup is a shortcut of software of unknown that interprets a discontent of web pages that are structurally equipped with incorrect HTML lines. There are another definitions of Beautiful soup that state that it is a python-based database which depends on the foundation of HTML/XML engines which can help in extracting and analyzing web pages [28].

However, to effectively receive the advantage of Beautiful soup some basic knowledge of HTML must be known. In short, HTML is an acronym of HyperText Markup Language. It is a programming language that web pages are created using it. The structure of this language is built based on elements called tags. The most important tags of HTML that can be used in this model are <a> which refers to rendering a link and <href> which is used in determining where the link goes.

5.2 Extracted Data-sets

Benefiting from web scraping, a data with three features are fetched using Python Beautiful soup model. More than two thousand websites, most of them are Turkish websites, are included in this data. As it is apparent in table 5.1 that the first column includes the name of the category. Next to it is the number of websites that are comprised by that category. The final column consists of two sub columns, which are Appname and URL, that represent a sample of a name of an app and its URL, respectively.

Table 5.1: Categories and Their Scraped Apps Number

Category	Example	# Included sites
Advertising	outbrain	30
Antivirus	kaspersky	12
Bank	Akbank	52
Cargo Delivery	dhl	10
Content Server	gcloudcs	19
Crypto Investing	Coinbase	38
Dating	Meetme	17
Delivery	Getir	24
Economy	Uzmanpara	26
Education	ITU	201
Game	minecraft	33
Government	E-Devlet	35
Health	112Acil	16
HouseholdAppliances	arcelik	6
Jobsearch	Kariyer	19
Mail	hotmail	14
Meeting	Zoom	8
News	TRThaber	109
OnlineCourse	Coursera	16
Onlinemarket	Trendyol	23
Porngraphy	Pornhub	311
RemoteControl	Teamviewer	22
Search/WebBrowser	Google	64
Service Provider	globalsign	44
Socialnetowrk	Facebook	44
Sport	Nike	42
Supermarket	BIM	13
Technology	Siemens	129
VPN	Hotspot Shield	20
gsm	TurkNet	12
internetTV	Netflix	44

6. HDFS, YARN, MAPREDUCE, TEZ, HIVE, AND ZOOKEEPER

So as to construct a suitable cluster for processing big data with the usage of Apache spark, some other prior applications must be firstly built. This is completed by installing HDFS, YARN, MAP REDUCE, TEZ, HIVE, and ZooKeeper applications. Yet, there are another optional monitoring applications such as Infra Solar and Metrics which provide interfaces that display charts related to the performance of the cluster. Moreover, to make it easy to write live codes on the cluster some open-source web application notebooks such as Zeppelin or Jupyter notebook can also be built in the cluster.

6.1 HDFS

HDFS is an abbreviation formed from the initial letters of Hadoop Distributed File System words. In computing, a distributed file system (DFS), generally, is a concept that is called on any file system that gives the thumbs up to multiple hosts on the same network to access different files. As a result, these multiple hosts can plainly share files and storage between them found on that file system. However, HDFS is considered as a sub-project in Hadoop. It is a type of distributed file system which is designed to run on commodity hardware[29]. This means HDFS has the ability, without requiring special drivers, to run on Windows or Linux or other OSes[30]. HDFS has superior properties over the other traditional DFSs such as high fault-tolerant, deploy-ability on low-cost hardware, and suitable storage for large data sets.

6.1.1 HDFS architecture

As it is stated before that HDFS is a distributed file system . So, in order to effectively administer it, it is built on the logic of clusters. This contributes HDFS to obtain a master-slave architecture as it can be clearly seen in Figure 6.1. HDFS, mainly, consists of two major nodes which are NameNode and DataNode. Additionally, another NameNode, called Secondary NameNode, can be built next to NameNode.

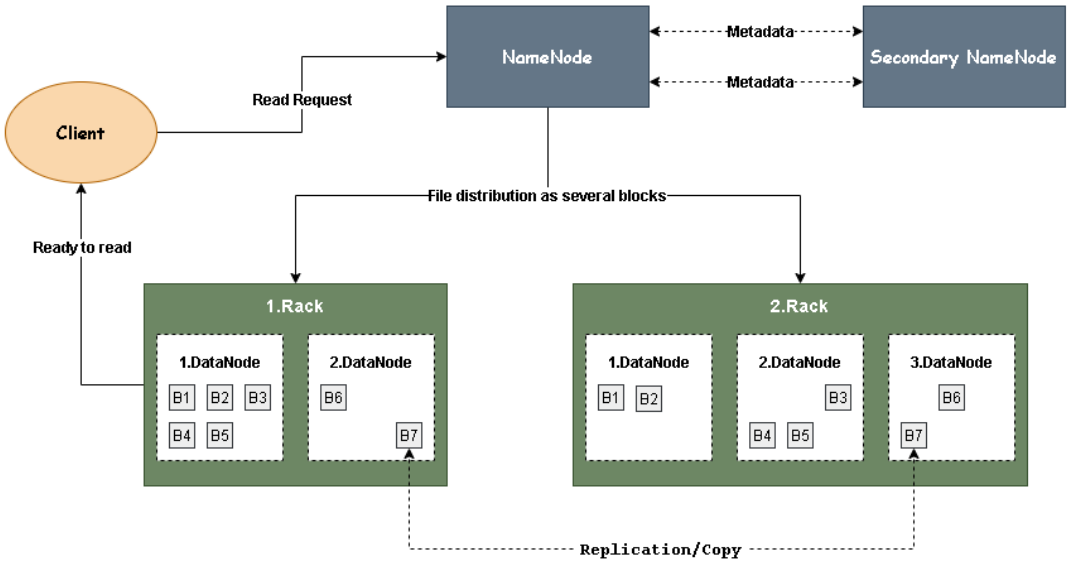


Figure 6.1: HDFS Architecture.

NameNode is always run on the master node of the cluster. It works in favour of storing and managing file systems' metadata. Metadata can be defined as basic information about data. For instance, the metadata of a document includes information such as its author name, size, and creating date. However, the metadata that Namenode manages is almost saved in a file called fsimage. To make the access to the metadata easier for clients, NameNode enables it to be cached in the main memory of the cluster. Which leads to faster read and write operations. Anyway, Namenode is responsible for another important jobs such as controlling DataNodes, splitting files into blocks, and distributing them over DataNodes. On the other hand, Secondary NameNode can not be contemplated as backup for NameNode[31], it fundamentally helps in reading file system, and applying changes to metadata.

DataNode is another magnificent component of HDFS which can be regarded as a primary storage of HDFS. As a rule, It has to be found on every slave node of the cluster Unless that slave node will not be able to access HDFS. DataNode's intrinsic job can be interpreted as achieving the read/write requests on the files stored on HDFS. The files' blocks of HDFS are copied on every DataNode to enable the property of fault tolerance. As a result, HDFS attains high reliability, availability, and swift computation.

6.2 Apache Hadoop YARN

The concept of Yet Another Resource Negotiator, which is shortened as YARN, could be specified as a cluster management tool. One of its functional roles is that it permits the diverse data processing engines (e.g., batch, stream, and interactive engines) to reach and process the data that is accumulated and stored in HDFS. In addition, it is responsible for allocating the clusters' available resources for processes that are just about to be simultaneously executed [32]. It also helps spark to be built on it without the need for any required pre-installation and the root access permissions [33].

6.2.1 Spark-based Hadoop YARN architecture

In this paragraph, the working principle of YARN in Apache spark is coherently discussed without any complexity. The architecture of YARN can be recognized in Figure 6.2. First of all, the concepts in the Figure 6.2 should be grasped. The green block shows the client node which can be considered as the machine, mostly the master machine, that programmer uses for running spark-submit process. It must include Spark driver which is the program or the script of the process that includes transformations and actions, that should be applied over data. It converts the user program into tasks as well. Spark Context displays the situation and connectivity of spark. The application name and application master type of the spark job is determined in Spark Context as well. Zookepers' main purpose is providing high reach availability for Spark to HDFS. Red blocks represent the master nodes of the cluster. One master node sometimes can not be sufficient for huge applications. Because of that one more master node can be utilized to manage the cluster. The need of MapReduce 2.0 (MRv2) or YARN, can primarily be attributed to the need of splitting JobTracker, resource management and job scheduling/monitoring, into separate daemons [34]. Its principle depends on establishing one Resource Manager and a number of Application Masters, the same number of running applications. Application Master takes the charge of deliberating resources from the Resource Manager. Then, it proceeds these resources amount to the NodeManager(s) so that tasks can be correctly executed. YARN Resource Manager Service is one of the main components of YARN. It is built on the master node of the cluster. However, it can be built on slave nodes that have network connection with Spark and YARN cluster. Its vital goal can be explained in making allocation decisions

for the amount of both CPU and RAM of the worker nodes. On the same node, master node, HDFS NameNode must be built and a connection between it and YARN Resource Manager must be available. HDFS Name Node has two principal purposes which are completing file system namespace operations, such as opening, closing, and renaming files and directories, and defining the mapping of blocks to DataNodes that are found on worker nodes. The light blue rectangular present the worker nodes of the cluster which are three nodes in this figure. The first square in the worker node is YARN Node Manager. Every worker node must include it since it plays the role of managing YARN Executors, Containers, by taking the amount of RAM and vCPU, given by YARN Resource Manager, and provide them to its own worker node. Besides, it monitors resource usage of its worker node. At the same time, it is responsible for reporting the results to the YARN Resource Manager. YARN Executor, or YARN container, carries out spark application. The executor, as it can be manifestly viewed from Figure 6.2, contains the tasks of the spark job that will be processed at. These tasks are concurrently, in parallel, mainpualted in the container. The RAM and vCPU amounts must be properly divided between containers. Cache block manifests the space in worker node in both its hard drive and in its RAM where the tasks are temporarily saved. The final block of the worker node contains HDFS DataNode. Read and write requests that come from file system's clients are performed in DataNode.

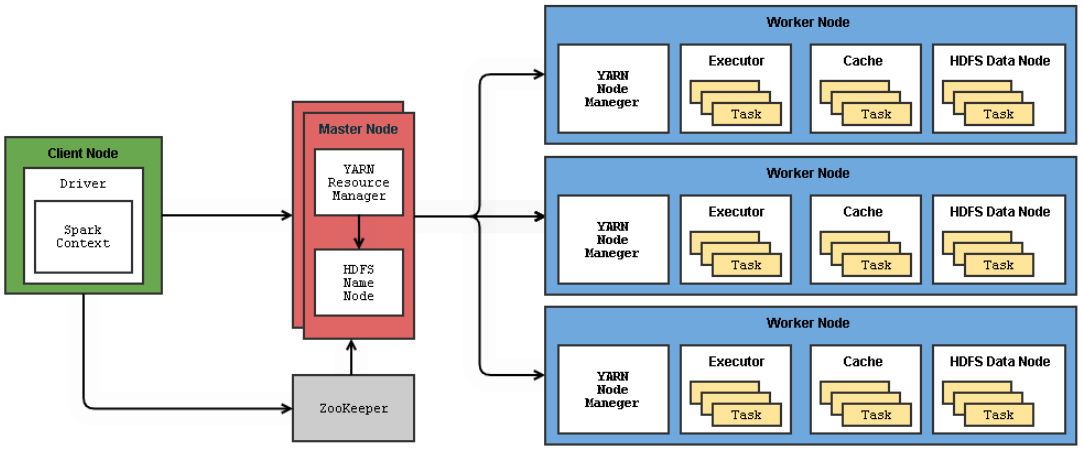


Figure 6.2: Apache Spark-Based Yarn Architecture.

6.3 Hadoop Yarn Deployment Mode: Client and Cluster

When master node is sharing the same host or location of worker nodes, it is recommended to use the client mode. For example, lunching master and worker nodes, the

cluster, on Amazon Web services which means all nodes share the same host, then, it should be used client mode. In this case, master node is considered as a client to the cluster since it is directly run with spark-submit process without the need to do additional steps. However, when worker nodes are established on a specific host or location and the master node is on another host, it is better to use the cluster mode in order to reduce the latency of the network. It is clear that both modes are using the technique of cluster of distributing tasks across nodes. But, they differ in the location of master or of slaves.

It can be called that client mode may have a better performance than cluster mode when the application of Apache Spark is performed on spark-shell or idle. However, cluster mode is preferred when the master has several jobs. Such a situation, cluster mode can archive a better balance of job distribution than client mode. Another differences betwen the two modes can be seen in Table 6.1 below.

Table 6.1: Differences between Client and Cluster modes.

	Cluster	Client
Sparkcontext on	Application Master	Client
Requesting resources by	Application Master	Application Master
Starting Executor Processes by	YARN NodeManager	YARN NodeManager
Presistent Services by	Yarn RessourceManager NodeManager	Yarn RessourceManager NodeManager
Supporting Spark Sehll?	No	Yes

6.3.1 Yarn configuration properties

YARN node manager amounts of RAM and vCPU can be specified by declaring yarn.nodemanager.resource.memory-mb and yarn.nodemanager.resource.cpu-vcores configuration properties, respectively. However, it is important to note that a (%10-20) of RAM and a kind of similar amount of vCPU should be left for underlying operating system. Otherwise, the cluster might not work due to operating system memory problems or because of a scanty of vCPUs. spark.yarn.executor.memoryOverhead property is responsible for allocating this amount of RAM. The last two properties of YARN are yarn.scheduler.minimum-allocation-mb yarn.scheduler.maximum-allocation-mb. They are in charge of specifying the minimum and maximum amounts of memory that YARN can not exceed. vCPU has the same last two properties of

RAM as well. At YARN resource manager UI, just memory information for allocated containers is revealed. Therefore, the adoptation of CPU scheduling information can be fulfiiled by modifying and setting yarn.scheduler.capacity.resource-calculator property as org.apache.hadoop.yarn.util.resource.DominantResourceCalculator. Finally, to determine the number of YARN containers, referring to [35] is suggested.

Table 6.2: YARN Configurations

YARN Configuration Property	Usage in YARN Container
yarn.nodemanager.resource.memory-mb	Allocates amount of physical memory for YARN node manager
yarn.scheduler.minimum-allocation-mb	States the minimum value of memory for YARN container
yarn.scheduler.maximum-allocation-mb	States the maximum value of memory for YARN container
yarn.nodemanager.resource.cpu-vcores	Allocates amount of virtual cores for YARN node manager
yarn.scheduler.minimum-allocation-vcores	States the minimum value of memory for YARN container
yarn.scheduler.maximum-allocation-vcores	States the maximum value of memory for YARN container
yarn.scheduler.capacity.resource-calculator	Calculates the usage of RAM and vCPU for cluster

6.4 Apache TEZ

Apache TEZ can be defined as a frameowrk that is built on top of YARN. Its main purpose is enabling a complex directed acyclic graph (DAG) for Apache spark tasks [36]. Apache Spark uses DAG as an alternative method for MapReduce. The DAG concept in Apache spark refers to graph of several vertices and edges. In DAG, RDDs or splitted data sets are considered as vertices and the operations that will be executed over these vertices are displayed as edges. The sequence or order of edges is based on the order of actions that will be applied on RDDs. In short, the first edge starts from the first action and so on. When an action is applied, a task will be lunched. The stages of that task is divided into several stages by DAG Scheduler. DAG in spark has another useful approaches such as fault tolerance (recovering lost RDDs) and executing several SQL queries.

6.5 HIVE

Hive can be considered as a system that has the ability of data warehouse in making fast data analysis and reporting. Hive, in general, is built on top of Hadoop. The orginal target of Hive is enabling smart analysis for the data that is stored in HDFS by using HiveQL, SQL-like language. Its feature of Use Defined Functions (UDF) paves the way for python and java scripts to be used in filtering data-sets found in HDFS without the need to apply MapReduce or any other methods [37]. Hive has the ability to get files from HDFS with different formats such as ORC, Parquet, CSV, and Txt. It is important to understand that Hive's Metastore, which is a file that contains metadata of HDFS data-sets, must be initially created before starting qureis to data found in HDFS.

6.6 ZooKeeper

When applications are distributed in Hadoop, the need for repository, called as Zookeeper, that provides a centralized place for distributed applications so that they can put data and take it out of it. In addition, ZooKeeper helps distributed applications by achieving their synchronization, serialization, coordination, and configuration.

7. APACHE SPARK

7.1 Overview

Apache Spark is a data engine that benefits from clusters mechanism to distribute data over several nodes, so that, data is processed faster. At the beginning, Apache spark was written by using Scala [38], then Java and Python software languages were became available to execute Apache spark. In general, Apache spark can be considered as an option to overcome the slowness of disk based operations [39] as it provides the ability to compute and process data on memory by catching it, disk option is also valid in Apaches Spark [40]. Processing data in memory helps in getting rid of disk overhead speed limitations. Compared to Hadoop MapReduce, on memory, Apache spark is 100x faster, and it is 10x faster on disk [41]. It can deal with both structured and unstructured data-sets [42]. DataFrames, Datasets, and Spark SQL can be thought as examples of structured data-sets that are provided by Apache spark, on the other hand, Resilient Distributed Data-sets (RDDs), Accumulators, and Broadcast variables can be summarized as examples of unstructured data-sets. Apache spark has several built-in libraries that can contribute to make the deal computer complex algorithms in big data much more easy. For instance, MLlib and GraphX libraries do the jobs of machine learning and graph processing, respectively

7.1.1 Cornerstone concepts

Before diving in Apache Spark, there are several important concepts such as MapReduce, fault tolerance, cluster, driver, executor, core, shuffling, data buffer, and data eviction that must be cautiously understand.

First of all, MapReduce divides one job into several small pieces called chunks which will be treated in parallel by map tasks. After that, these chunks will be combined together with reduce tasks. MapReduce process differs from Apache spark in processing data, since the first one process data on disk and the other one process data on memory.

As Apache spark has the property of fault tolerance which acquired since it keeps

running even there is a failure of some of its components. On the other hand, Cluster can be defined as a group of machines that have number of master and slave nodes. Master can be thought as the main machine that sends jobs as queries to slaves machines. Slave machine can be considered as a worker machine that does make the job sent by master, then, gives feedback to the master. In Apache Spark, Master and slave nodes are called as Driver and Executor nodes, respectively. In addition, the number of jobs that can be done by one slave in parallel is called core.

Eventually, data shuffling refers to the rearrangement of data between portions. While moving data from one place to another, a special size of memory, for the moment, is allocated as a storage for that data which is called data buffer. Next, data eviction refers to getting rid of the old and unused data-sets that are allocated in the cache memory.

7.2 Apache Spark Architecture

Apache Spark architecture, see Figure 7.1, consists of three main parts which are program driver, cluster manager, and worker nodes. Moreover, executors are found inside Worker nodes and can be defined as processes that utilize some of the cluster's CPU and RAM in order to run tasks in parallel. Starting from the left side, Driver program or Master node is responsible to run SparkContext or SparkSession which are entry points for Apache Spark application. However, SparkContext is used to initiate Apach Spark RDD API and SparkContext is used to start Apache Spark Dataframe API. On the other hand, Cluster manager is in charge of allocating the cluster resources orderly.

When an Apache Spark application is executed, first the SparkContext or SparkSessipn will be initialized by driver program. Then, executors will be established inside worker nodes. Driver program also runs users' code and divide the task into independent small tasks and send them to executors to be processed. It is important to note that the entry point of Apach Spark application should be connected to cluster manager in order to allocate the resources between worker nodes appropriately. This can be achieved by using different types of management tools such as Hadoop YARN, Mesos, Kubernetes, or even Apache spark Standalone. In this project the manager of Apache Spark applications is decided to be YARN.Finally, executors start doing their definite job and when results are obtained, they send them back to the driver.

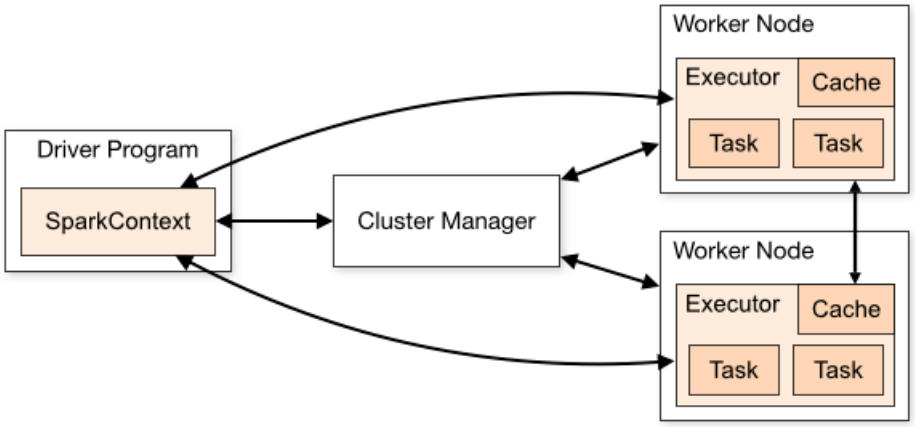


Figure 7.1: Apache Spark Architecture [43]

7.3 Apache Spark Configurations

When talking about Apache Spark, the subject of Apache Spark configurations definitely must be accurately studied. Apache spark has several configurations that have diverse functions. However, since some of these configurations are trivial and can be skipped, only the major configurations will be discussed in this research. Configurations can be classified in six different groups which are application properties, runtime environment, shuffle behavior, compression and serialization, memory management, and execution behaviour.

7.3.1 Application properties

Primarily, there are several configurations lay under the title of application Properties. The most important ones can be summarized as: **spark.app.name** is responsible for defining the name of the being executed spark application. **spark.driver.cores** and **spark.executor.cores** determine the number of driver and executor cores to be used during the application. **spark.driver.memory** and **spark.executor.memory** are in charge of controlling the amount of driver and executor memories. **spark.driver.memoryOverhead** and **spark.executor.memoryOverhead** specifies the amount of driver and executor non-heap memories, respectively. Non-heap memory involves VM overheads, interned strings, other native overheads, off-heap memory, and memory of driver and non-driver processes.

Finally, **spark.submit.deployMode** which identifies the driver deployment mode which can be either client or cluster modes. **spark.executor.instances** allocates the number

of containers that will be established on one node.

spark.dynamicAllocation.enabled it regulates the number of executors automatically based on the workload. If it is set as true, then both **spark.shuffle.service.enabled** and **spark.dynamicAllocation.shuffleTrack-**
ing.enabled must be also set as true.

As it is popular that data in Apache spark is processed on memory, so, problems of out of memory are common while dealing with Apache spark. Thus, the maximum memory size that can be allocated for each container must be well known. For driver, it equals to the sum of spark.driver.memory and spark.driver.memoryOverhead. Similarly, for executor, it is equal to the sum of spark.executor.memory and spark.executor.memoryOverhead. If the configuration property of off-heap memory is enabled, then, its value must be added to the maximum memory sizes of both driver and executor. If the memory configuration is left without writing its unit, it will be autmatically set as megabyte. However, its unit can be named as k for kilbyte, m for megabyte, g for gigabyte, and t for terabyte.

7.3.2 Runtime environment and networking

spark.driver.defaultJavaOptions and **spark.executor.defaultJavaOptions** can be considered as configurations that set the default Java virtual machine (JVM) to the driver and to the executors. It must be checked by administrators. On the other hand, **spark.driver.extraJavaOptions** and **spark.executor.extraJavaOptions** are in control of determining the extra java option for the driver and the executors of the cluster. It is important to call that spark.driver.defaultJavaOptions must be stated before spark.driver.extraJavaOptions. **spark.network.timeout** represents the timeout of network transactions.

7.3.3 Shuffle behavior, compression and serialization

The configurations of shuffle behavior and compression and serialization are mostly Boolean variables which means their values can be either true or false. Generally, compression operations use spark.io.compression.codec to achieve their jobs. **spark.shuffle.compress** is used to give the permission for compressing output files from map operation. Compression, by default, is in Snappy format which diminishes

the errors of memory since it works by employing the mentality of using 32KB of buffer. It plays an important role in increasing execution time of operations by reducing the seeking for disks and system calls. The size of buffer for each compressed shuffle can be changed by using **spark.shuffle.file.buffer** configuration. **spark.broadcast.compress** is used to enable sending broadcast variables as a compressed files. Eventually, **spark.rdd.compress** gives the right to compress serialized RDD partitions. These RDD partitions are kept on CPU, Memory, or Disk by using StorageLevel option. While enabling spark.rdd.compress, a huge space can be relieved on CPU which promotes the execution time. **spark.shuffle.spill.compress** enables the compression of data spilled.

7.3.4 Memory management

spark.memory.fraction locates the fraction of memory that can be used by other internal operations that are related to metadata, data structures, and data sparse. The value of this configuration should not be dramatically low, since it has an inverse relationship with spills and data eviction. **spark.memory.storageFraction** defines the fraction of storage memory in terms of spark.memory.fraction. Moreover, It has an inverse relationship with memory as the higher the storage fraction memory is the lower available memory. **spark.memory.offHeap.enabled** enables Apache spark to utilize off-heap memory. **spark.memory.offHeap.size** assigns the usable amount of off-heap memory. While considering this configuration, the fact that it has no impact on heap memory usage should be known.

7.3.5 Execution behavior

spark.broadcast.blockSize sets the size of broadcast blocks that will be transmitted to TorrentBroadcastFactory. It is important to notice that higher value of this configuration may reduce the number of parallelism of broadcast partitions. **spark.broadcast.checksum** main purpose is to ensure there is no corrupted blocks during broadcast, if so, it tries repairing the corrupted blocks by giving information to the resourcemanager. **spark.default.parallelism** states the number of data partitions returned by transformations. The maximum size of a single partitions can be limited by using **spark.files.maxPartitionBytes** configuration. **spark.executor.heartbeatInterval** de-

termines the time interval between the same executor heartbeats that tells the driver about the current situation of that executor. This configuration must be seriously adjusted with **spark.network.timeout** since there should be enough time between them

7.4 Optimizing Apache Spark Cluster

As it is mentioned before that the number of tasks which can be done at the same time by one executor is set by using spark.executors.cores, so, when the virtual cores of one slave is bigger than or equal to 8 cores, both spark.executors.cores and spark.driver.cores should be set as 5 which enhances HDFS throughput. Although this choose is based on several researches and historical data [44], It is sometimes inefficient. Increasing the number of tasks per executor reduces the number of executors themselves, thereby, reducing the number of parallelism, However, for slaves that have cores less than 8, it is not plausible to set spark.executors.cores as 5, it should be lower. The number of total executors that can be located in one node can be calculated by using Equation 7.1. The minus one in this equation is used in order to keep one core to be used by the system and other services. If it is already extracted from the total cores, minus one will be deprecated.

$$\textbf{Total Executors Per Node} = \frac{Number of Cores per Node - 1}{spark.executors.cores} \tag{7.1}$$

After calculating the total number of executors per node, the total executor memory per node can be found, see Equation 7.2, by dividing RAM per instance over the result of Equation 7.1. It is clear from the equation that 1 GB of memory is reduced from the RAM of that instance in order to allocate that 1 GB for the Operation System different processes.

$$\textbf{Total Executor Memory} = \frac{Total RAM per Instance - 1}{Number of Executors per Instance} \tag{7.2}$$

Then, from Equation 7.3 it is obvious that spark.executors.memory and spark.driver.memory configurations are equal to %90 of the total executor memory as %10 is allocated for YARN overhead memory which can be determined by using spark.yarn.executor.memoryOverhead, see Equation 7.5.

$$\textbf{spark.executor.memory} = TotalExecutorMemory * 0.90 \quad (7.3)$$

$$\textbf{spark.driver.memory} = spark.executors.memory \quad (7.4)$$

$$\textbf{spark.yarn.executor.memoryOverhead} = TotalExecutorMemory * 0.10 \quad (7.5)$$

The property of deployment of Apache Spark of spark.executor.instances can be considered as a configuration that defines the number of containers of Apache spark application. In YARN-based Apache Spark cluster, the request for containers is sent to YARN ResourceManager. Container numbers can be calculated by using Equation 7.6 which states that it equals to the number of executors per node multiplied by the number of cores per node minus one for other operations.

$$\textbf{spark.executor.instances} = (\#ExecutorsperNode * \#WorkerNodes) - 1 \quad (7.6)$$

Another important configuration that should be carefully utilized is spark.default.parallelism. It determines the number of partitions of distributed data-sets used by different operations, especially transformations such as join and reduceByKey. The value of this configuration can be specified by using the Equation 7.7, below. In the case of using Dataframe, another configuration which is spark.sql.shuffle.partitions must be added along with the previous one. As it is obvious from Equation 7.8 that both configurations have the same value. However, it is encouraged to use df.repartition(numOfPartitions) or df.coalesce(numOfPartitions) rather than spark.default.parallelism in specifying the number of parallisim as they can be changed whenever it is required.

$$\textbf{spark.default.parallelism} = (spark.executor.instances * spark.executors.cores) * 2 \quad (7.7)$$

$$\textbf{spark.sql.shuffle.partitions} = spark.default.parallelism \quad (7.8)$$

The total amount of memory that can be allocated by YARN NodeManager for containers on one node can be adjusted by controlling yarn.nodemanager.resource.memory-mb.

So as to avoid out of memory problems, Equation 7.9 must be guardedly calculated.

$$\textbf{yarn.nodemanager.resource.memory-mb} >= (spark.executor.memory + spark.yarn.executor.memoryOverhead) * TotalExecutorsPerNode \quad (7.9)$$

The maximum number of tasks that a cluster can execute in parallel is determined by multiplying spark.executor.instances and spark.executor.cores, see Equation 7.10.

$$\textbf{Maximum number of tasks} = spark.executor.cores * spark.executor.instances \quad (7.10)$$

Further, another configurations that are related to java such as spark.executor.extraJavaOptions and spark.driver.extraJavaOptions which helps in reducing the total memory potential garbage collection can be set as follow:

spark.executor.extraJavaOptions= -XX:+UseG1GC -XX:+UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark -XX:InitiatingHeapOccupancyPercent=35 -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'
spark.driver.extraJavaOptions= -XX:+UseG1GC -XX:+UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark -XX:InitiatingHeapOccupancyPercent=35 -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'

On the other side, assigning configurations for virtual and physical memory of YARN might play a prime role in enhancing execution time of an application. In general, yarn.nodemanager.vmem-check-enabled and yarn.nodemanager.pmem-check-enabled check the amount of virtual and physical memories of YARN, respectively. They are preferred to be set as false, thereby getting rid of checking time.

The most critical part of setting Apache Spark configurations is where to set them, using spark-submit or SparkContext. As it previously stated that there are several configurations that are related to deployment or runtime control of Apache spark that must be neatly specified. Some of these configurations, if they are set using SparkContext, may not be useful at all as the Apache Spark application and the driver JVM have been started already. Therefore, when starting Apache Spark application,

it is highly recommended to add configurations directly to spark-submit method or to spark-defaults.conf as the configurations will be set before the begin of Apache Spark [45]. Moreover, the difference of the structure of the same configuration between cluster and client deployment modes and between SparkContext and spark-submit methods must be completely recognized. For instance, the number of containers per executor can be expressed by the configuration property of spark.executor.instances and –num-executors in SparkContext and spark-submit methods, respectively.

7.4.1 Apache Spark memory management based on Hadoop YARN

The different types of memory that is allocated for Apache Spark executor can be seen in Figure 7.2 of all, the memory of YARN NodeManger must be set. This can be easily achieved by using the configuration of yarn.nodemanager.resource.memory-mb. This configuration determines the amount of memory in megabyte, Apache Spark uses 1000 factor to convert gigabyte to megabyte. While calculating its value, it must be considered that its value is not equal to the value of the RAM of the whole machine as some of the of it will be used by system and other services. Moreover, two other configurations which are yarn.scheduler.maximum-allocation-mb and yarn.scheduler.minimum-allocation-mb can be used to specify maximum and minimum amount of container's RAM, respectively.

After that, two main components of the container's RAM must be well understood which are executor memory (heap memory) and executor yarn overhead memory. The first one can be set using the configuration of spark.executor.memory. Its equation was discussed in this chapter in the subsection. However, this configuration value is divided into two values which are reserved memory and total usable memory. The main purpose of reserved memory is storing sparks internal objects with a value of 300mb default. It can be changed using the configuration of spark.testing.reservedMemory as it stated in Equation 7.11.

$$\textbf{Reserved Memory} = spark.testing.reservedMemory \tag{7.11}$$

On the other hand, total usable memory consists of two different parts which are total spark memory and user memory. Its value can be determined by extracting Reserved

memory from spark.executor.memory as it is shown in Equation 7.12.

$$\textbf{Total Usable Memory} = spark.executor.memory - Reservedmemory \quad (7.12)$$

Furthermore, total spark memory depends on two components which are storage memory and execution memory. Total Spark Memory, see Equation 7.13, can be decided based on the configuration of spark.memory.fraction which is multiplied with total usable memory.

$$\textbf{Total Spark Memory} = Totalusablememory * spark.memory.fraction \quad (7.13)$$

First component of total spark memory is execution memory which is in charge of storing spark execution tasks until they are done. It can be calculated using Equation 7.14. It is important to note that this amount of memory is used while executing Apache spark different operations such as joins, shuffles, aggregations, etc.

$$\textbf{Execution Memory} = TotalSparkMemory*(1-spark.memory.storageFraction) \quad (7.14)$$

The other component of total spark memory is storage memory which mainly stores broadcast variables and cached data. Its value can be assigned using Equation 7.15. In this equation, if the value of the propriety of spark.memory.storageFraction is not set by the user, it will be equal to 0.5 as a default value.

$$\textbf{Storage Memory} = TotalSparkMemory * spark.memory.storageFraction \quad (7.15)$$

Nevertheless, user memory which is a part of the usable memory, is not used by Apache Spark. From its name it can be claimed that it is the amount of memory that the user can use in any way and it can be simply calculated by using Equation 7.16.

$$\textbf{User Memory} = Totalusablememory * (1 - spark.memory.fraction) \quad (7.16)$$

Finally, as Hadoop YARN is a service which needs Memory in order to work and deal with these complex steps of Apache Spark managemetn, some amount of memory must be allocated for it. This amount of memory is desirable to be 0.1 of the executor memory. Its value can be modified using the property of spark.yarn.executor.MemoryOverhead.

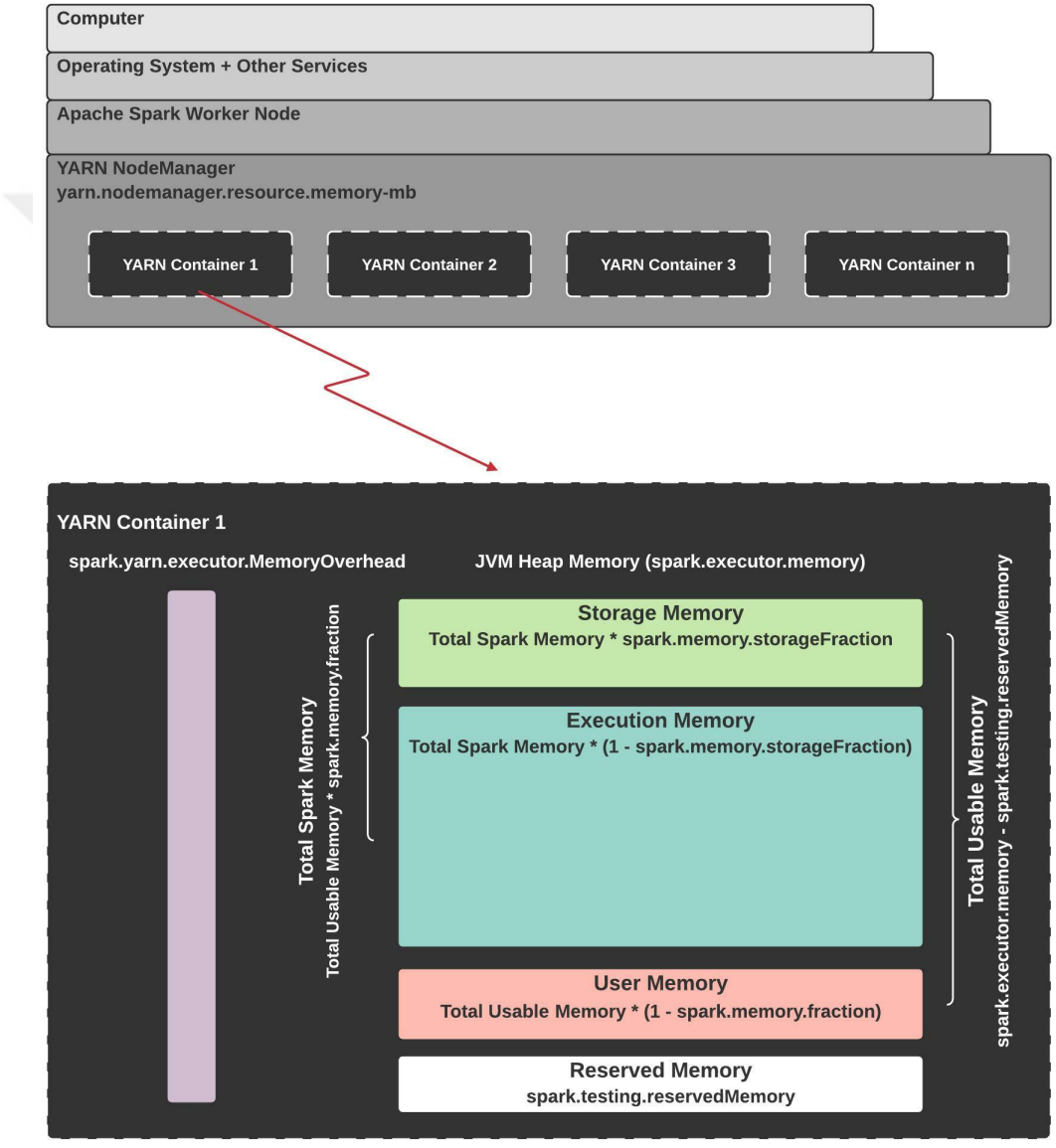


Figure 7.2: Apache Spark Memory Diagram

7.5 Resilient Distributed Dataset

Resilient Distributed Dataset (RDD) depictes the basic idea of Apache Spark. It can be considerd as an element with immutable and partitioned features that has the ability

to be processed in parallel.

7.5.1 RDD operations

There are two main operations of RDD which are Transformations and Actions. Transformations are lazy functions that can not be executed without applying Actions. Generally, Transformations take an input RDD and give a new output RDD without changing the input one. Transformations, in particular, based on the number of partitions are divided into two different types which are narrow and wide Transformations. Functions of Map, Filter, and Union are examples of narrow Transformations. While Distinct, Join(), Repartition(), and Coalesce() are examples of wide Transformations.

Contrary to Transformations, Actions are immediate functions that deals with the original input RDD and give non-RDD output. When an Action is applied, the processed data will be transfared from executor to driver. Some examples of several Apache spark Actions can be thought as reduce(), count(), and collect().

7.6 Comparison between RDD, Dataframe, and Dataset

Currently, Apache Spark Application Programming Interface (API) can be found in three different types which are Rdd, Dataframe, and Datasets. The difference between them can be noticed in Table 7.1, which compares them in the following features, Spark Release, Data Representation, Data Formats, immutability, interoperability, compile-time type safety, optimization, serialization, garbage Collection, lazy evolution, supported programming Language, schema projection, and speed of aggregation. Based on the comparison, it is better to utilize Dataset API. However, Datafram API is chosen in this project since it was done using Python Programming language, Dataset API does not support Python.

Table 7.1: Differences between RDD, Dataframe, and Dataset

Feature	RDD	Dataframe	Dataset
Spark Release	Spark 1.0	Spark 1.3	Spark 1.6
Data Representation	Java or Scala objects	Named Tables	Named Tables
Data Formats	Struc.+ Unstruc.	Struc.+Semi-struc.	Struc.+Unstru.
Immutability	Immutable	Non-immutable	Immutable
Compile-time type safety	Yes	No	Yes
Optimization	Unavailable	Catalyst optimizer	Catalyst optimizer
Serialization	Java serialization	Spark serialization	Spark serialization
Garbage Collection	High	No	No
Efficiency/Memory use	Low Efficiency	High Efficiency	High Efficiency
Lazy Evolution	Yes	Yes	Yes
Programming Language	Java, Scala, Python, R	Java, Scala, Python, R	Scala, Java
Aggregation	Low	Medium	High

Moreover, another study [46] was conducted on measuring the execution time of one mutual operation of RDD and Dataframe which its results can be seen in Figure 7.3. It is clearly that Dataframe operation is faster than the one of RDD. Therefore, the Apache Spark API for this project was chosen based on the information found in both Table 7.1 and 7.3 which is Dataframe.

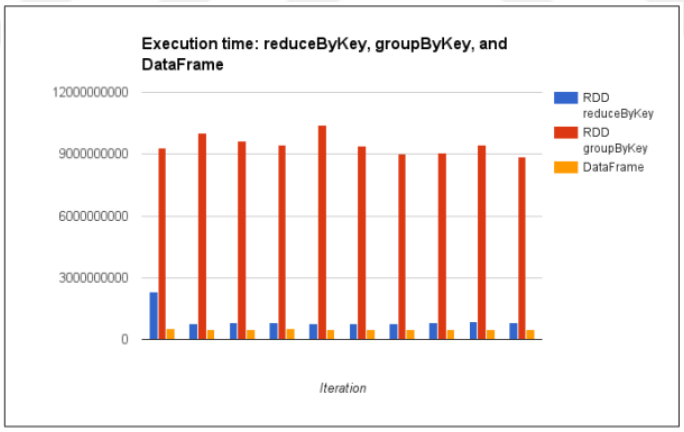


Figure 7.3: Performance Comparision Between RDD and Dataframe In Term of Exection Time.

8. CLOUD BASED CLUSTER

As it was previously mentioned that one purposes, which this project aims to fulfill, is making several experiments on different Apache Spark cluster sizes in order to find out the most suitable cluster size and feature to be applied to a new local cluster. It is not plausible to establish local clusters with different sizes and features since it will bring superfluous effort, time, and funds. Thus, the used clusters in this project are cloud based clusters that were established and configured on AWS. AWS might be defined as a cloud platform that gives ability of creating sundry user friendly services to its clients with lower costs. Only, specific services of AWS were harnessed to attain the desired processes.

8.1 Amazon S3

Amazon S3 acronym refers to Amazon Simple Storage Service which can be thought as an object storage service that enable clients to store their data in special buckets. It is characterized by its high scalability, large data availability, and ultimate security. Therefore, s3 is believed to be the storage of this project as AWS based Apache Spark clusters can get data and write it easily.

8.2 Amazon Elastic MapReduce

Amazon Elastic MapReduce or EMR is an analytic platform-as-a-service (PaaS) that is built on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3) and accessible by AWS. The mentality of Amazon EMR can be recognized as the one of Hadoop since it divide large data-sets into small partitions and distribute them across established EC2 master/s and slave/s machines [48]. In other words, it can be regarded as a platform that enables clients to create clusters with several models, libraries, and services. At the same time, EMR hourly prices [47] are reasonable when they are compared with their potential. It is important to note that, in order to get the total price of that cluster, EMR price must be added to the price of each EC2 found in that cluster. The built-in services and engines that can be easily established by EMR are

Hadoop, Zeppelin, Livy, JupyterHub, Tez, Flink, Ganglia, HBase, Pig, Hive, Presto, ZooKeeper, upyterEnterpriseGateway, MXNet, Sqoop, Mahout, Hue, Phoenix, Oozie, Spark, HCatalog, and TensorFlow. In this project, some of them which are Hadoop, Ganglia, Hive, Zeppelin, and Spark were just utilized. Except Ganglia and Zeppelin, all the other platforms are widely demonstrated. Ganglia can be use in the purpose of monitoring the usage of cluster machines. On the other hand, Python codes can be written on Zeppelin which can be realize as a notebook.

9. OBTAINED RESULTS AND VIZUALIZATIONS

The general analytic system of this project [49] can be conspicuously seen in Figure 9.1. As it is previously mentioned that the used data-set in this project is a data that contains observations made by clients movements on internet which is stored on different DNS servers. At the same time, other data-sets that show clients specific information such as their connection duration, IP, port, place of residence, etc. are stored on database server. These data-sets have to be carefully collected and stored in a storage that cluster can reach. Amazon s3 was the main used storage in this project. After collecting all needed data-sets, the implementation of processing data can be efficiently started. A YARN-based Apache Spark cluster is utilized to perform this step which is established using Amazon EMR. However, different cluster sizes with various features were investigated in order to determine the proper one for executing processing step in the future. Moreover, these clusters were highly optimized based on the configurations that were stated in Chapter 6 and 7. Subsequently, the difference between the optimized cluster and the normal one will be widely discussed. Eventually, different visualizations, which are pie, bar, and line charts and choropleth maps, were efficiently made using two distinct visualization tools which are Elasticsearch-based Kibana and Plotly. On kibana, an interactive dashboard was established in order to make immediate filtering requests based on user's inputs.

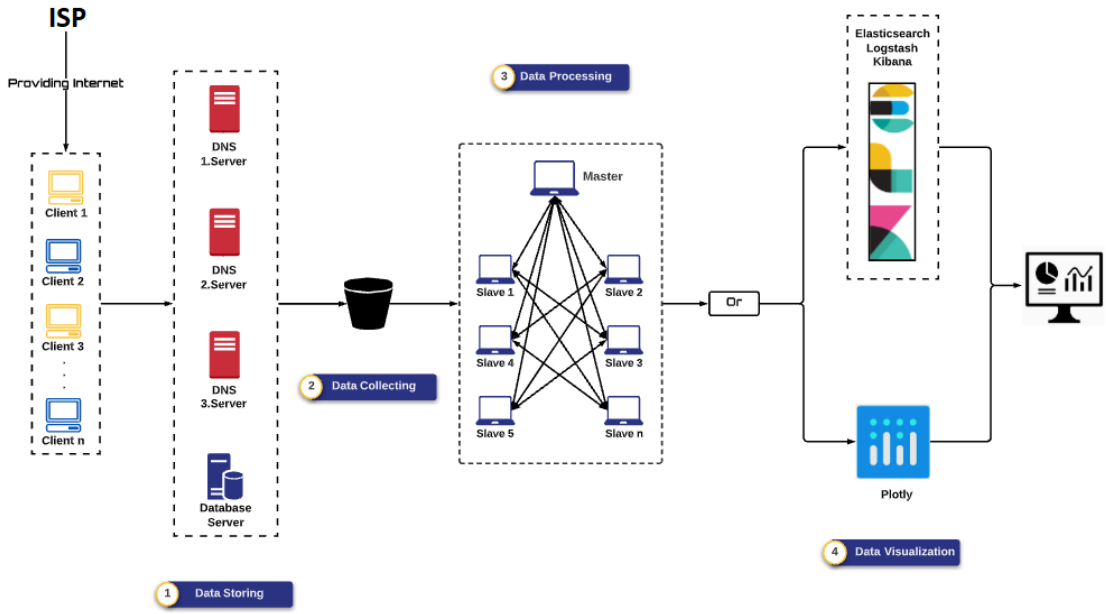


Figure 9.1: Project's General Analytic System.

A sample of three observations of output data can be noticed in Table 9.1. The meaning of each column can be stated as follow: observation capturing time, DateTime, Name of the DNS server that captured that raw, Server, and client's city, district, and neighbourhood are stored under columns with the same names. User's internet type is represented in T.Type. Finally, URL column shows the entered URL by client and Category column introduces the category of that URL.

Table 9.1: Sample of Output DNS Data-set

DateTime	Server	City	District	Neighborhood	T.Type	URL	Category
2021-09-16 14:09:33	1-b1	kutahya	merkez	saray	VADSL	a.root-servers.net	Suspicious
2021-09-16 23:11:01	2-b1	istanbul	besiktas	arnavutkoy	YAPA	get-express-vpn.com	VPN
2021-09-16 15:00:03	2-b1	bingol	merkez	yesilyurt	VADSL	graph.facebook.com	Social Media

9.1 Cluster Optimization

While trying to get the proper cluster, different sizes and different machines were used, See Table 9.2. In this table, used machine's names and features such as their virtual CPU number, amount of RAM, EBS bandwidth, Network Bandwidth and Network performance can be clearly noticed.

Table 9.2: Features of the Used EMR EC2 Instances

Instance	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Performance
m5.large	2	8	EBS-only	Moderate
m5.xlarge	4	16	EBS-only	High
m5.4xlarge	16	64	EBS-only	High

All machines that are stated in Table 9.2 were used in different cluster sizes. The results of processing 1.3 TB DNS data, one day of data, using three different clusters for each type of machines, without optimizing them, that are stated in Table 9.2, can be found in Table 9.3. It is obvious that the execution time of m5.large instance with its different sizes ranges between approximately 7 and 60 hours which is not rational to be utilized in processing big data due to its huge execution time. On the other hand, execution time of m5.xlarge, with 3.5 hours as lower limit and 15.1 hours as upper limit is somehow reasonable. However, it is still a huge time which could lead to undesirable situations such as inability to correct errors if occur and missing processing data. The final experiment was done over m5.4xlarge instance. Its execution time ranges between 0.87 and 4.7 hours which is plausible to consider. Thus, m5.4xlarge with 1+10 size was selected to be used and optimized in this project.

Table 9.3: Execution Times for Different Clusters

Node Type	Cluster Size (Master + Slave)	Input Data (TB)	Execution Time (m)
m5.large	1+2	1.3	3639
m5.large	1+6	1.3	921
m5.large	1+10	1.3	456
m5.xlarge	1+2	1.3	945
m5.xlarge	1+6	1.3	434
m5.xlarge	1+10	1.3	219
m5.4xlarge	1+2	1.3	283
m5.4xlarge	1+6	1.3	123
m5.4xlarge	1+10	1.3	60

Nevertheless, before starting our process step, the selected cluster must be optimized based on the properties that were previously discussed in chapter seven. Thus, After leaving some amount of memory and one Vcore for the operating system and other services, the remaining of RAM and Vcores per node are approximately 57gb and 15 core, respectively. These two values must be given to YARN resourcemanager. The first one can is given using the configuration of yarn.nodemanager.resource.memory-mb

and the later one is given using the property of yarn.nodemanager.resource.cpu-vcores.

After that, the number of tasks that can be achieved per one executor are specified. Based on several experiments that included different numbers of tasks, see Table 9.4, the most efficient number of tasks that one executor of our cluster can process in the same time is three tasks. It can be obviously seen from Table Table 9.4 that executor with three tasks has the highest number of parallelism, number of partitions. Accordingly, the propriety of spark.executor.cores is set to three which leads to have five executors per node, see Equiaton 7.1. Eventually, maximum number of parallel tasks was calculated using Equation 7.10 and number of parallelism was obtained using Equation 7.8.

Table 9.4: Experiments of determining the number of parallel tasks

# Tasks	# Executors per node	Max. number of parallel tasks	# Parallelism
2	7	138	276
3	5	147	294
4	3	116	232
5	3	145	290

Based on that, the value of the properties of spark.executor.memory = 9000m and spark.yarn.executor.memoryOverhead = 2000m were calculated using their equations, see Equation 7.3 and Equation 7.5, respectively. On the other hand, since the reserved memory was not considered while preparing the configuration file, its value remained as default which is 300mb. Afther that, using Equation 7.12 the total usable memory was determined. In addition, the cluster is only supposed to process given data-sets. Therefore, the value of user memory was assumed to be small since it will not be used by any user. This is done by setting the property of spark.memory.fraction to a high fraction of 0.9. Consequently, the value of total spark memory is directly set to a higher value. Based on this value, both total spark memory and user memory values can be found using their equations (Equation 7.13 and Equation 7.16). Finally, since the operation of Apache Spark does not need a lot of cache memory, the property of spark.memory.storageFraction was set to 0.1. Which means that spark execution memory, see Equation 7.14, will be higher than spark storage memory, see Equation 7.15. The values of Apache Spark memories can be detected in Table 9.5

All the configurations and their values are precisely mentioned in the end of the thesis in APPENDIX B.

Table 9.5: Used Apache Spark Memory values

Property	Value
Reserved Memory	300 MB
Total Usable Memory	8700 MB
User Memory	870 MB
Executor Driver Memory	1000 MB
Total Spark Memory	6830 MB
Spark Execution Memory	6147 MB
Spark Storage Memory	683 MB

In order to understand the benefits of optimizing clusters, a comparison between a default cluster and an optimized cluster will be closely stated. First of all, percent usage of CPU of the unoptimized cluster can be seen in Figure 9.2. It is clear from the figure that several CPU usage fluctuations were happening while executing code which in turn leads to inaccurate usage of the available CPU. As a result, some of worekr nodes' cores will be inactive and the runtime of the process will be increased. Inversely, the figure of the optimized cluster CPU usage, see Figure 9.3, shows that cluster was able to achieve steady CPU usage of roughly %90. The x-axis and y-axis of these two figures display time and percent of CPU usage, respectively. Usage of CPU by user can be found in blue distribution and by system is considered in red one.

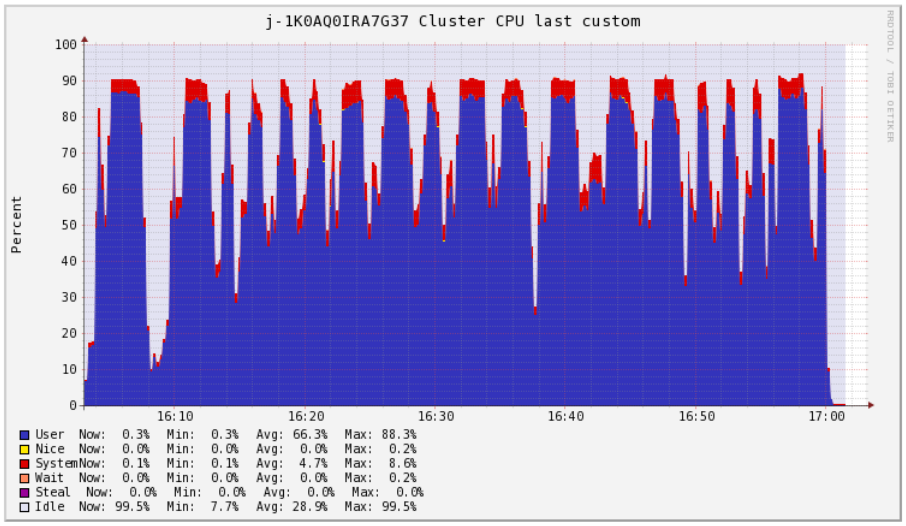


Figure 9.2: Unoptimized Cluster CPU Usage

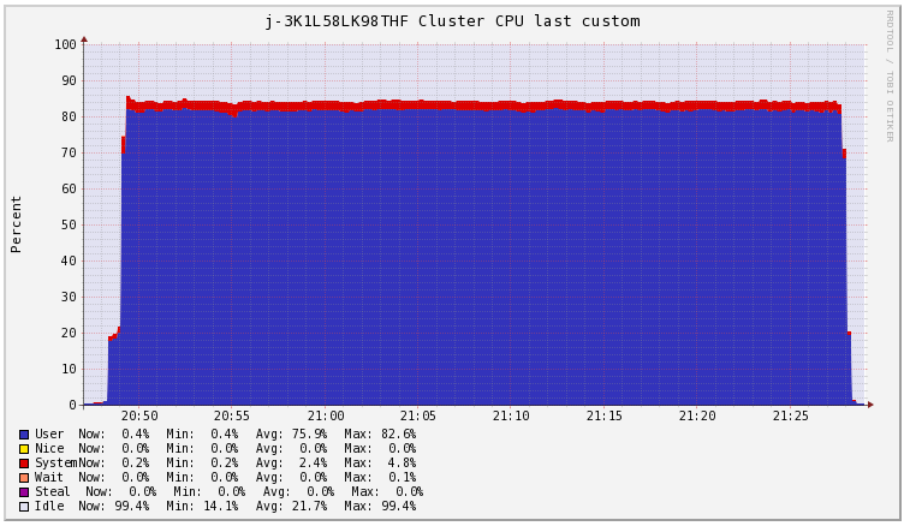


Figure 9.3: Optimized Cluster CPU Usage

Also, the unoptimized cluster was unable to utilize a huge part of the available memory which remained useless. Based on Table 9.5, each executor in Apache Spark cluster must be able to use 6830 MB of the available memory. Howver, in the unoptimized cluster, all executors were merely able to use 178.8 GB as a maximum value, see Figure 9.4. In addition, there is inconstancy in the buffer memory usage. In contrast, in the optimized cluster there were 50 executors (YARN containers). Each one of them must be able to use 6830 MB of the available memory. Yet, 1 GB of memory was left on each worker node to ensure no memory errors. Accordingly, there are 50 executor, 6830 MB of memory per executor, and 1 GB for worker nodes, see Figure A.2. Total used memory can be calculated as (50 * 6830) - (10 * 1000) which gives 331500 MB or 331.5 GB. As it is clear from Figure 9.5 that the optimized cluster was able to use 330.4 GB, which is very close to the calculated value. In memory figures, buffer memory is represented in green distribution and user usage is displayed in blue one. The axises of tese figures are as follow; x-axis shows execution time and y-axis contains amount of memory.

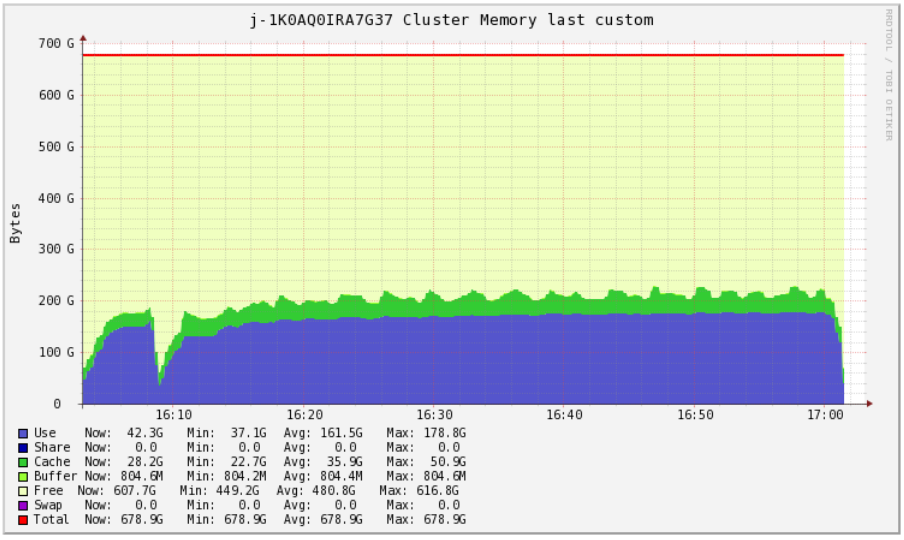


Figure 9.4: Unoptimized Cluster RAM Usage

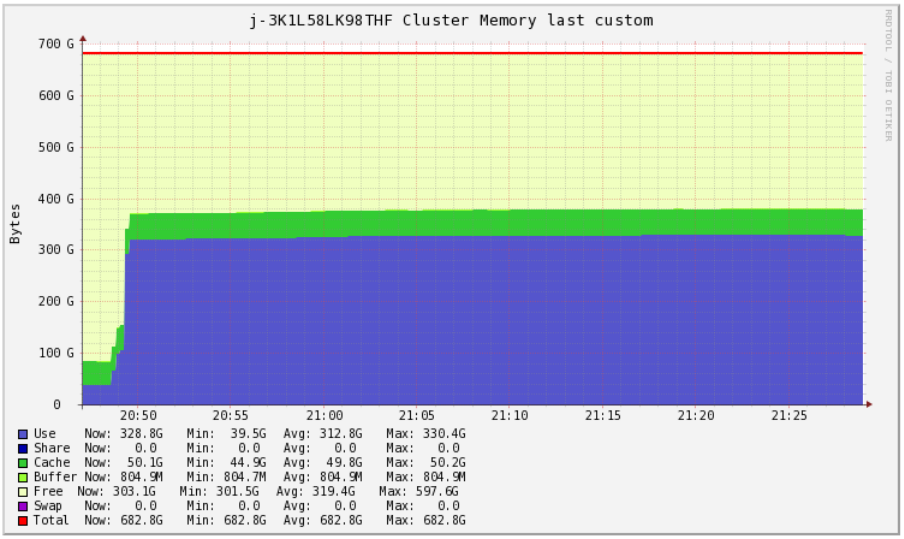


Figure 9.5: Optimized Cluster RAM Usage

Hence, based on the usage of CPU and memory, the load of the cluster would be affected positively or negatively. If the resources of the cluster were not optimized, the load of the cluster will exceed its limit and will lead to significant dumps thereby negaticely affecting the execution time. This can be plainly seen Figure 9.6 which represents the total load of the cluster. The individual load of unoptimized master and woreker nodes can be noticed in Figreü 9.7. In opposite, optimizing the cluster contributes to steady load distribution as the resources are utilized well. Based on the values that has been calculated in Table 9.4, using five executors with 3 cores each enables the cluster to process 147 tasks in parallel which is almost the same as the average value of the load in Figure 9.8. Moreover, the separate load of optimized master and worker nodes can

be demonstrated in Figure 9.9. As a result, the load of the cluster is steady without having fluctuations.

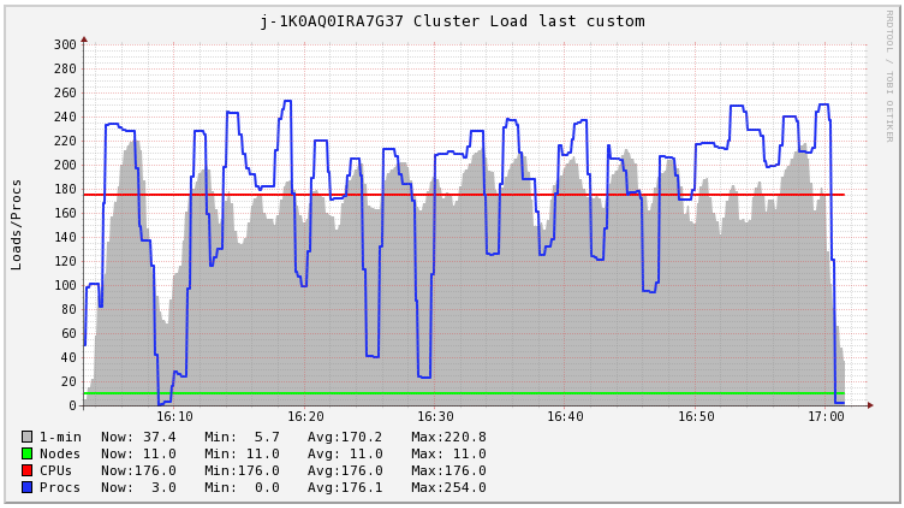


Figure 9.6: Unoptimized Cluster Total Load Distribution

Figure 9.7: Individual Load Distribution of Unoptimized Cluster Nodes

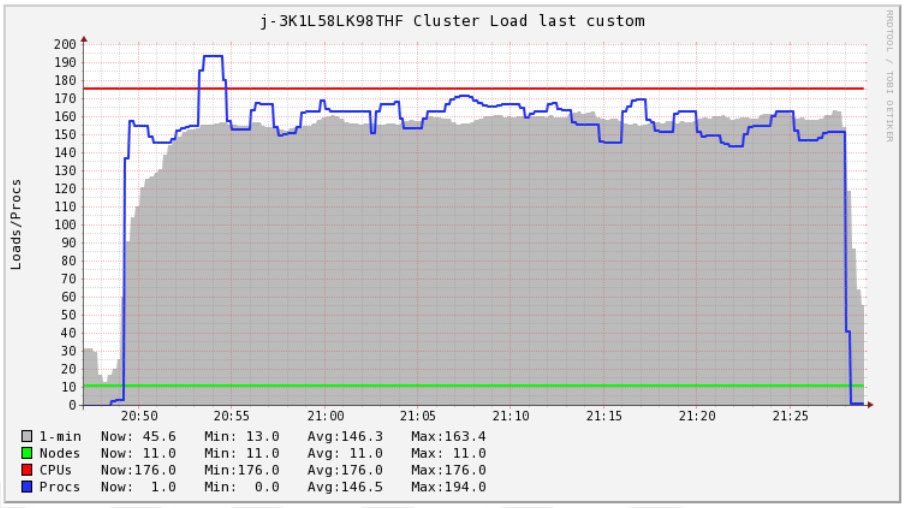


Figure 9.8: Optimized Cluster Total Load Distribution

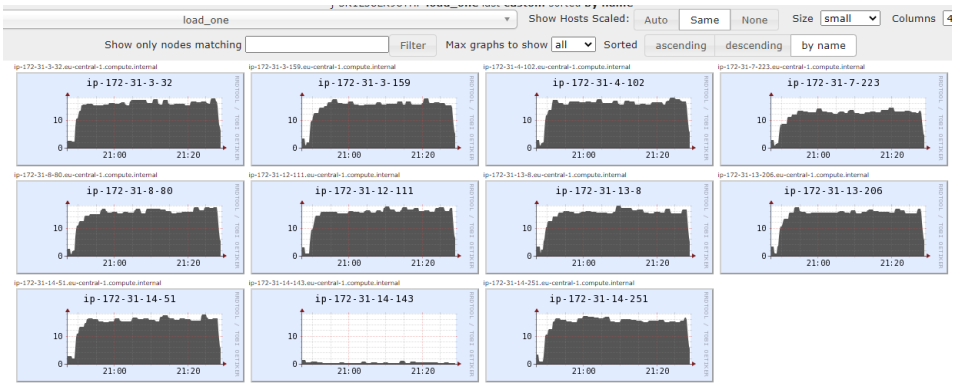


Figure 9.9: Individual Load Distribution of Optimized Cluster Nodes

Eventually, the results of processing 1.3 TB of DNS data-set using unoptimized and optimized clusters were obtained, see Table 9.6. These clusters consist of one master node and ten worker nodes with resources of 16 core and 64 GB RAM. Their Performances were measured using same data-set and execution code. Therefore, they are mutual in input size, input row, and output row. Nevertheless, since CSV format is used while writing output data in the unoptimized cluster, the size of output data is bigger than the one of the optimized cluster which employs Parquet format for output data-set. Also, the execution time of the optimized cluster was 38 minutes, and 60 minutes for the unoptimized one. Consequently, 22 minutes can be saved when processing one day of DNS data-set which means 660 minutes can be reduced monthly from the execution time by optimizing the cluster before using it. As a result of reducing output size and enhancing the execution time, the cost of the process can be affected approximately by %54.9 which saves almost 1.36$ hourly. If the process is supposed to be run on all hours of days of months for one year, 11750.4$ will be the total saved cost. Thus, the most rational thing that can be made while processing big data is establishing a local cluster which only has an initial cost and some other few costs.

Table 9.6: Comparison between Two Identical clusters (Optimized and Unoptimized)

Cluster	Input Size (TB)	Input Row (B)	Output Size (GB)	Output Row (B)	Execution Time (m)	Execution Cost ($)	Storage Cost ($)	Total cost ($)
Unoptimized	1.3	8.71	991.38	7.60	60	2.11	0.76	2.87
Optimized	1.3	8.71	254.2	7.60	38	1.33	0.18	1.51
Difference	**0**	**0**	**737.18**	**0**	**22**	**0.78**	**0.58**	**1.36**

9.2 Data Visualization

It can be said that output data will not be useful, without making clear visualizations from it, in the way it should be. Therefore, to take the full advantage of the output data, it must be found in a controllable dashboard that can make interactive charts from it when it is required. In this project, the used tools for visualization the output data were ELK, Plotly, and Matplotlib.

Initially, the processed data-set was transferred and visualized on a Kibana-based dashboard. The quires of this dashboard are made by using an Elasticsearch cluster. This dashboard has the ability to filter existing visualization based on a control panel that contains different filters such as filters based on city, district, neighborhood, transmission type, URL, and category. The first made visualization is Istanbul neighborhood map that can be found in Figure 9.10 below. This map shows the distribution of DNS traffic, red for high traffic and green for low traffic, between the user's neighborhood of the ISP.

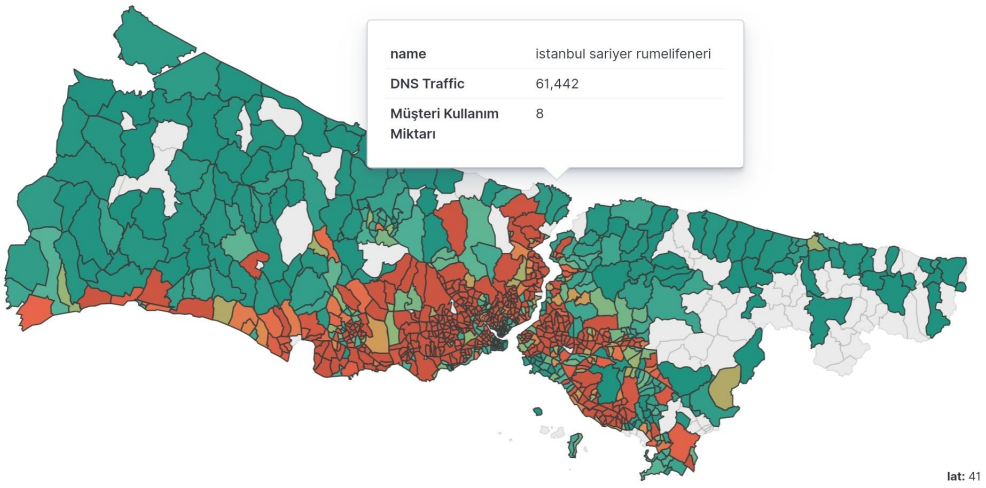


Figure 9.10: Istanbul Neighborhood Map In terms of DNS Traffic

At the same time, since the dashboard can be filtered as the user desires, a selected sample that shows the distribution of DNS traffic in only Besiktas districts' nighborhoods can be clearly seen in Figure 9.11

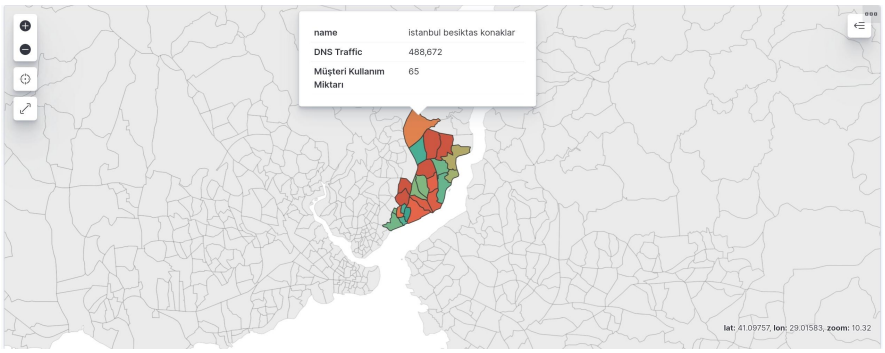


Figure 9.11: Istanbul Beşiktaş Neighborhood Map In terms of DNS Traffic

As the data of URLs' categories was extracted and joined with the DNS data, the category of each URL can be found in the output data. The percentage of each visited category in terms of unique click cab be clearly seen in Figure 9.12. Moreover, DNS traffic made by users is represented in Figure 9.13 which y-axsis shows visits count and x-axix shows the name of category.

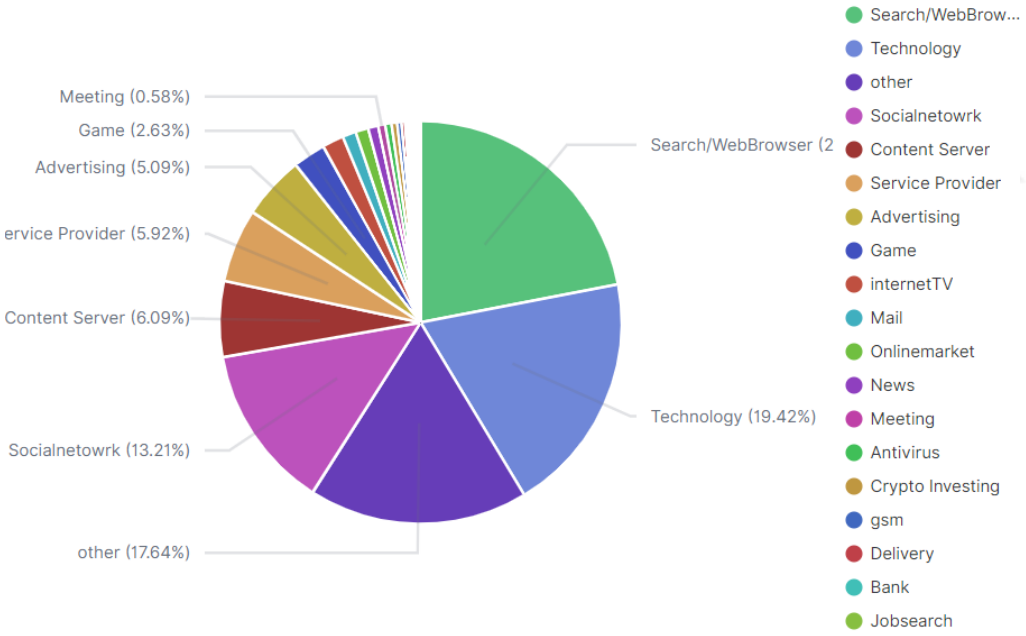


Figure 9.12: Pie Chart of Most visited URL Categories

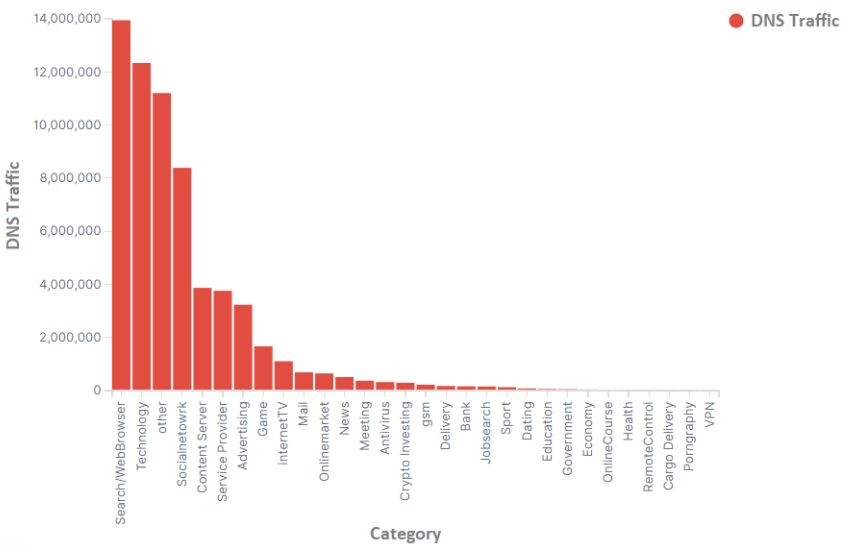


Figure 9.13: Line Chart of Most visited URL Categories

Furthermore, the percentage of used domains, not categories, is displayed in Figure 9.14 as a pie chart. The number of clics on these domains, visits count, is exposed in Figure 9.15 with y-axix as the visits count and x-axis with the name of that domain.

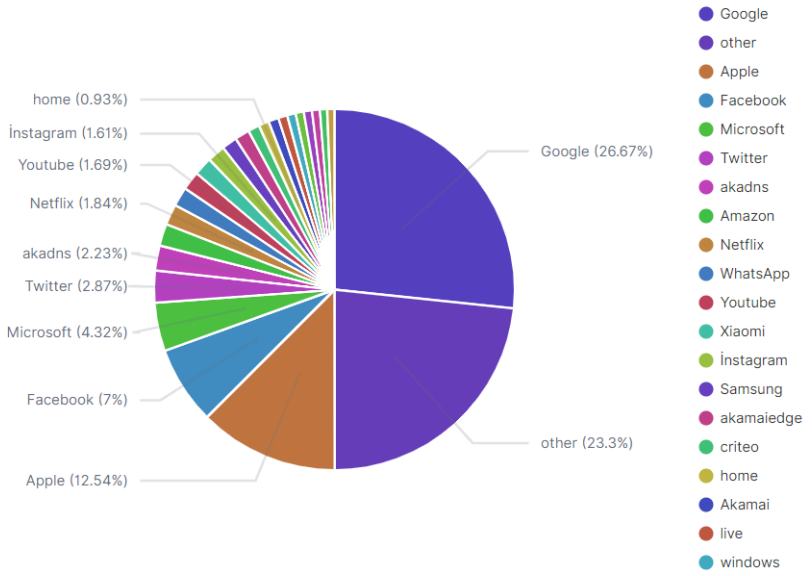


Figure 9.14: Pie Chart of Most visited Domains

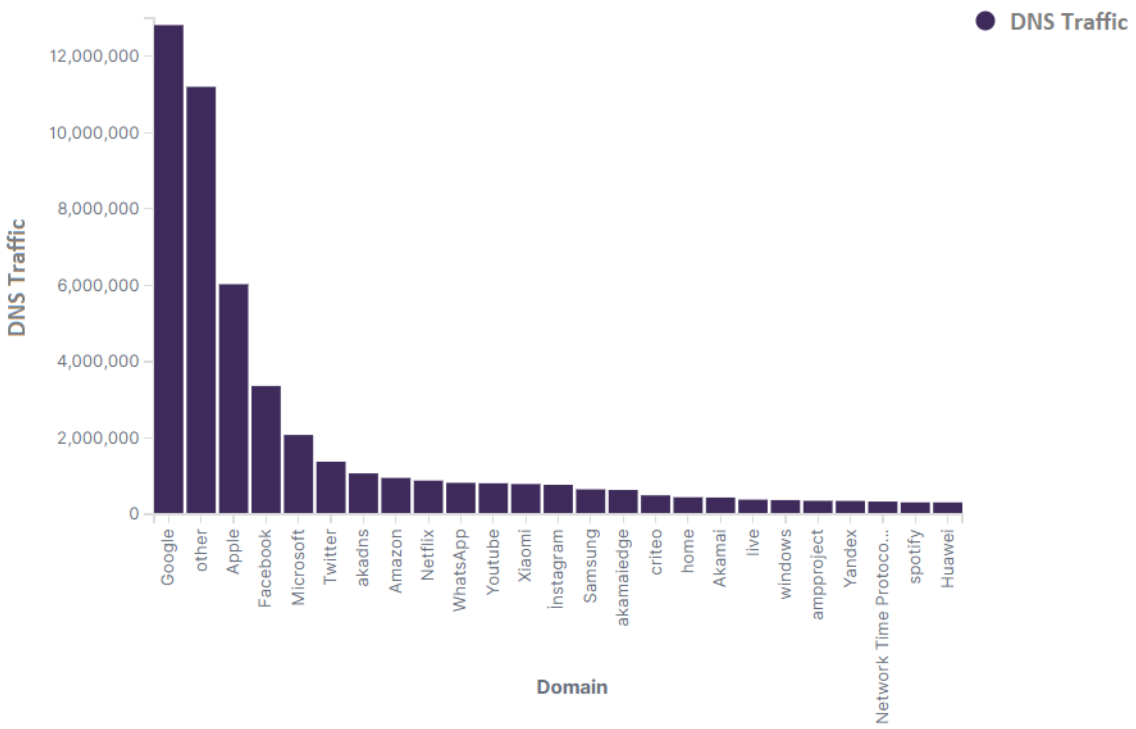


Figure 9.15: Line Chart of Most visited Domains

The previously discussed figures are taken from the dashboard of Kibana. The information of these figures can be changed to the needed one just by selecting the desired destination such as name of city, district, transmission type of customers, URL category, and date.

On the other hand, there are figures which can be made by using another Python libraries such as Plotly and Matplolib. For example, the distribution of DNS traffic over Turkey, Ankara, Istanbul, and Izmir between three specific banks can be noticed in Figure 9.16. As it is clear that, the DNS traffic of Isbank is dominant over Turkey's cities. Only, when looking at districts, another information could realized. For instance, bank at Izmir's districts is Ziraat bank.

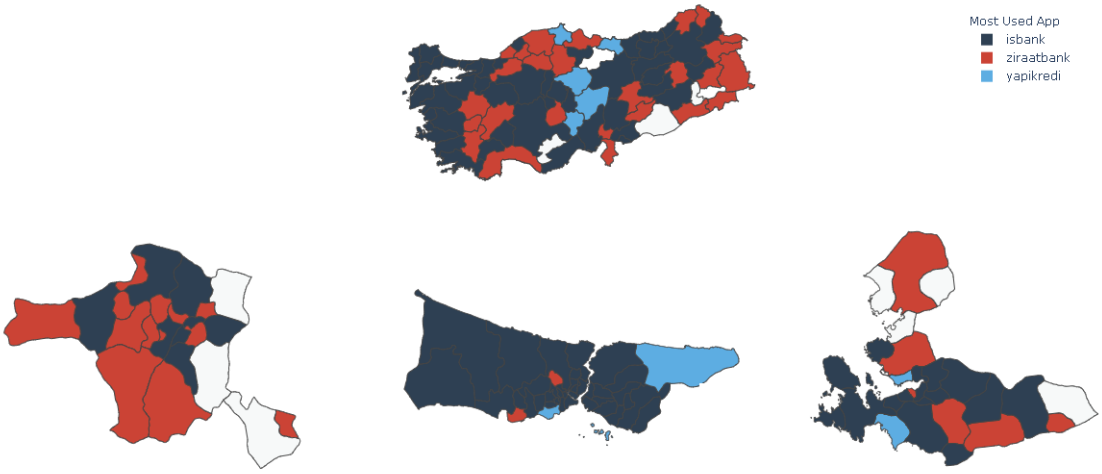


Figure 9.16: Comparison between Turkey, Ankara, Istanbul, and Izmir in terms of three banks.

Finally, based on the output data, a race bar chart that shows the cumulative sum of DNS traffic of defined domains over time can be made. This is summarized in Figure 9.17. It contains the cumulative sum of DNS traffic made by main market palces in Turkey over one month. The datetime of this figure includes the hourly change of DNS traffic.

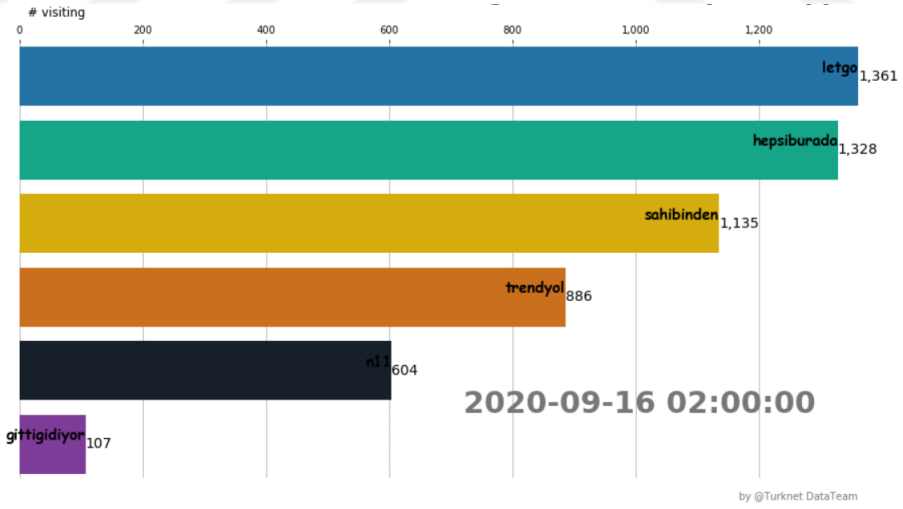


Figure 9.17: Race Chart Bar Between Specific Market places in Turkey.

9.3 Application of Time Series Data Forecasting

Recently, almost all companies etc. telecommunication, online stores, and hospitals started collecting various types of data-sets that mainly contain a column which represents the date time or timestamp of that collected data row. Storing such multiple

data-sets could play an essential role in helping companies to get familiar with their past and their present. It can attribute to make predictions for the future of the company as well. A Data that has a feature, column, of time period that is attached with other features is called time series data[50]. For example, any data that is collected from any medium, except single period mediums, cross-sectional data, can be considered as time series data. Analyzing data with several observations on a single individual at regular intervals is called time-series analysis[51]. However, in order to get future predictions from the data, a model ,that is able to efficiently deal with the data, must be build. There are different time series forecast models which can be implemented such as Autoregression (AR), Seasonal Autoregressive Integrated Moving-Average (SARIMA), Vector Autoregression (VAR), Holt Winter's Exponential Smoothing (HWES), and Fbprophet. In this project, the latter model, which is Fbprophet, is utilized using Python.

9.3.1 Fbprophet model

The term Fbprophet stands for Facebook predict Fit. It is a forecasting model implemented in Python that is created by Facebook company. It enables programmers to analyze their time series data-sets and get future predictions. Fbprophet model, generally, expects a data with two features which are ds and y. The first column, df, is a date-stamp that can be in YYYY-MM-DD format for a date or YYYY-MM-DD HH:MM:SS format for a timestamp. The other column, y, is the value of the measurements, such as counts, for that date-stamp which we are planning to forecast[52]. Nevertheless, two other columns can be also found in Fbprophet data. They are carrying capacity (cap) and floor columns. The Cap column is a value the controls the growth of the curve, however, floor column is used as saturating minimum.

The Fbprophet model is mainly based on three model components which are trend, seasonality, and holidays. Also, there is error term that has effects on the model. These components can be noticed in the equation 9.1 bellow:

$$\mathbf{y(t)} = g(t) + s(t) + h(t) + \epsilon t \tag{9.1}$$

The first component is g(t). It is responsible for considering the type of the growth of the curve which can be either linear or logistic. The other component is s(t) which represents the periodic changes that can be daily, weekly, monthly, and yearly seasonality or all of them. This component is critical and should be carefully used. Its equation 9.2 is seen below that has P variable as a period (7 days for weekly seasonality and 364.25 days for yearly seasonality and so on.) and a_n, b_n variables that should be estimated for N observations to determine seasonality.

$$\mathbf{s(t)} = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right) \tag{9.2}$$

The last component of Fbprophet model is h(t). It symbolizes the effects of holidays on the curve. In addition, the ϵ t is the error term which is in charge of detecting any abnormal changes occur in the curve. The idea of dealing with forecasting in Fbprophet is that the model considers forecasting problem as a curve-fitting. Unlike other models, it does not look directly at the time of each row within the data[53].

9.3.2 Optimizing the model

Any model can not be implemented in a powerfully way unless its parameters are well studied. If the model is performed without sufficient knowledge, then, even sometimes satisfied results my be obtained, the model could collapse suddenly and may not work for some other problems. consequently, After grasping the model, it becomes easy to execute it and get the best results.

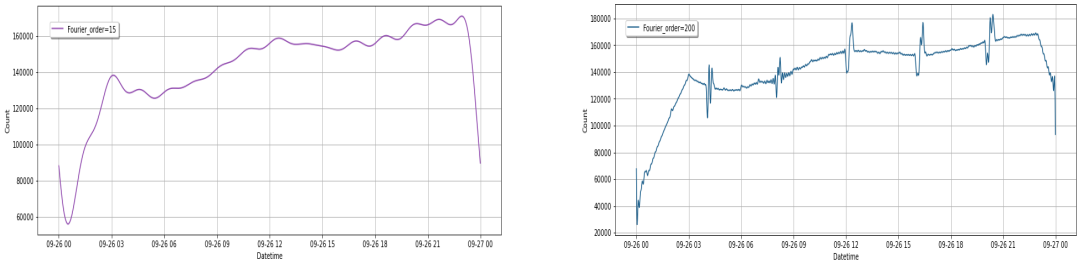
9.3.2.1. Growth and holidays

So as to select an accurate growth parameter, data should initially be plotted. Plotting data gives a glance about the way that the data follows. Growth parameter is specified according to the behavior of data and can have one of two values which are linear or logistic. Linear one, as the concept suggests, is chosen when the data is pure and has not some sudden different patterns or saturation points. Further, logistic is adopted when data exhibits unusual and unexpected patterns, thereby, the cap and floor columns have to be exist in the data to determine the maximum and minimum values that data can have. The other parameter is holiday. This parameter is useful when a

day-based forecasting, not hourly, is going to be implemented. Holidays, sometimes, has influential effects on data since people and markets behaviours differ in these days from the other normal days.

9.3.2.2. Change-points and seasonalities

When data have some points that have entirely different behaviour, compared to the other points of the data, then these points are called changepoints. They are automatically located by the model itself. However, instead of giving the right to the model to explicitly determine them, they can be set by the programmer with changepointes parameter, then, the model will just take the assigned points. This can be achieved by utilizing parameters such as changepoints, n_changepoints, changepoint_range and changepoint_prior_scale. Seasonality parameter plays an important role in developing the model to be more precise. This parameter must be carefully executed. There are three built-in seasonalites in Fbprohet model which are yearly_seasonality, weekly_seasonality and daily_seasonality. These important factors are given values depending on the scale of data and its behaviour. While designating seasonality, an essential parameter that should not be ignored is Fourier_order. It is the number of Fourier components to use[54]. In an easier way, as it is well known that, signals are composed of sum of some or several sine and cosine waves, then, Fourier_order is the number of these waves. Figure 9.18 is an clear example that displays the difference between low and high fourier_order for the same wave. It is obvious that Figure 9.18 (a) has less pulses or bends than Figure 9.18 (b) because of its low Fourier order. If this parameter is not precisely assigned the results of the model will be confined.



(a) Fourier Order $\bar{1}5$ **(b)** Fourier Order $\bar{2}00$

Figure 9.18: Difference Between Low and High Fourier Order

9.3.2.3. Uncertainty in seasonality and MCMC samples

Based on the history data of the forecast, if the future of it seems to be uncertain, then uncertainty in seasonality parameter must be taken in consideration. Fbprophet, in particular, realizes the average frequency and magnitude of trend changes in the past. Then, it considers the future values to be the same as they were in the past, and so, gives future predictions depending on the past behaviour[55]. To be able to apply uncertainty in seasonality in Fbprophet, Markov chain Monte Carlo (MCMC) must be defined in the model. MCMC, briefly, is a sampling method that makes characterizing of a distribution possible, in spite of unawareness of the distribution properties. MCMC accomplishes this by randomly sampling the distribution values. It is important to note that the operation of this parameter is so expensive on windows and may take several minutes to be done. Because of this disadvantage of Fbprophet on windows, it is recommended to use Linux when applying MCMC in the model.

9.4 Case Study

The previous stated forecasting model, Fbprophet, is employed with high efficiency for detecting active user numbers, which their data is stored on DNS servers. The purpose of this study is to establish forecasting model that could be used for forecasting missing data. Data, occasionally, can be missed due to abrupt reasons such as problems in storing or analyzing data, this model can be utilized to override such problems. Moreover, the model can be used to estimate the future number of active users of each minute over time. However, to achieve this study some steps must be accomplished to get the desired data before go forecasting. First of all, eight days of DNS data is processed and analyzed. Then, by applying a smart groupby method the number of active users for each minute of these eight days is obtained as it is displayed in the Figure 9.19 bellow.

9.4.1 Results

As it is well known that to check the model efficiency data should be split into two parts: train and text. After training the model, two cases of tests were regarded to emphasize the astonishing performance of the model. The first and the second cases can be noticed in Figure 9.20 and Figure 9.21, respectively. In the first case, the 1440 minutes of the 23 September 2020 day are picked out and considered as a test data.

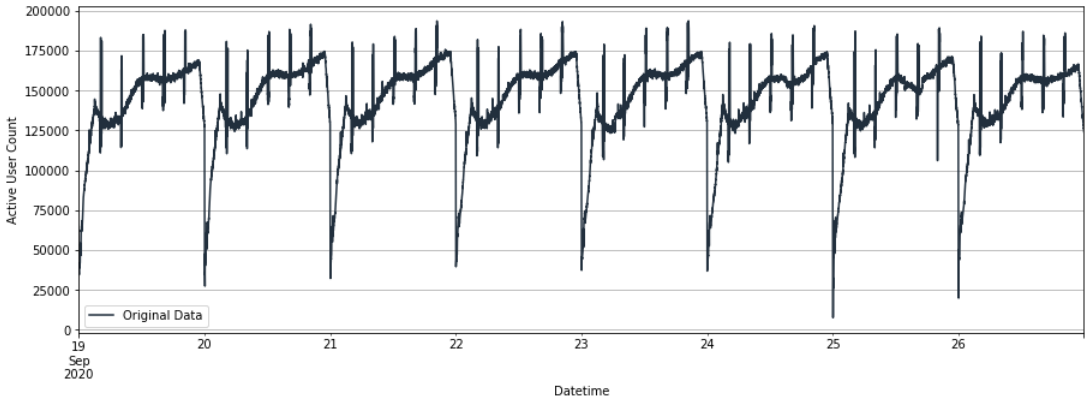


Figure 9.19: Original Data of Active User Numbers Over Eight Days

This case represents the case of missing data. The black line is the actual data and the blue one is the predicted data. Detected change points are shown as a vertical red lines.

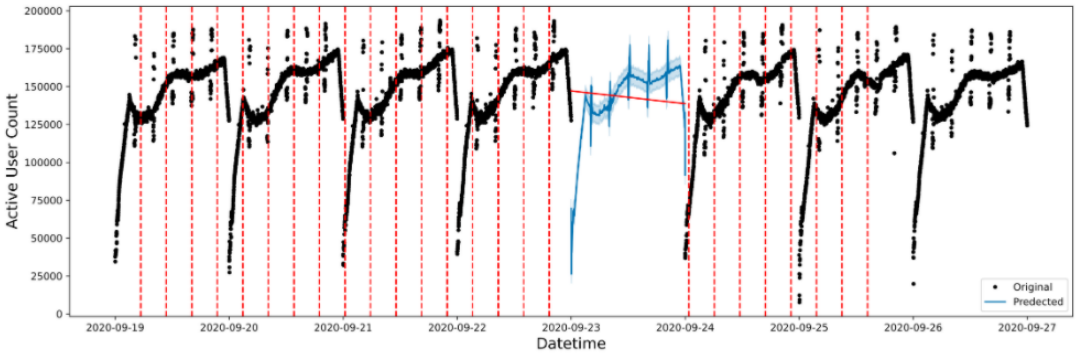


Figure 9.20: Forecasting the Data of Active Users (Middle)

The other test case is 26 September 2020 day which can be considered as a real example of prediction of the behaviour of the data. In other words, it helps in making accurate prediction for the number of users that will be active in the next day.

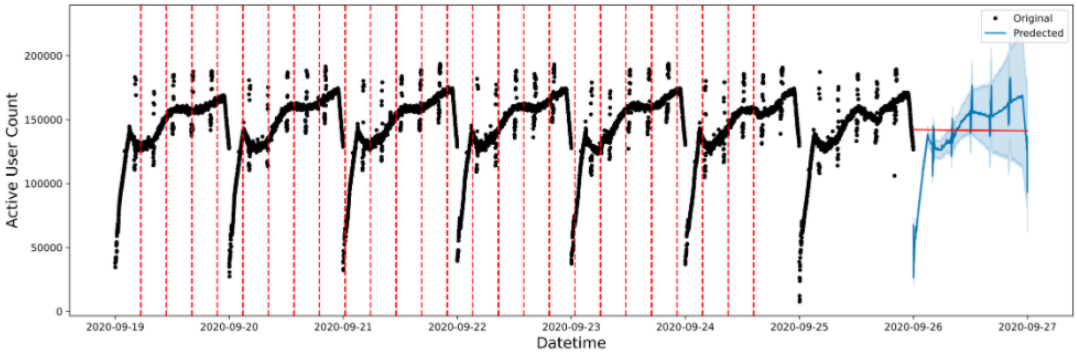


Figure 9.21: Forecasting the Data of Active Users (End)

Fitting the previously clarified parameters in a proper way into the model, gives green

light to the programmer to start using it. Checking the model over two samples of data will give a sight about the precision of the model. Furthermore, the acquired results demonstrates and assess the efficiency of the model.

The preceding two figures exhibit only the trained, predicted, data. It would be inadequate to evaluate the trained data without test it with the real one. Figure 9.21 is a line chart that combines the predicted and real data-sets together. The red line indicates the actual data and the blue line points out the predicted one.

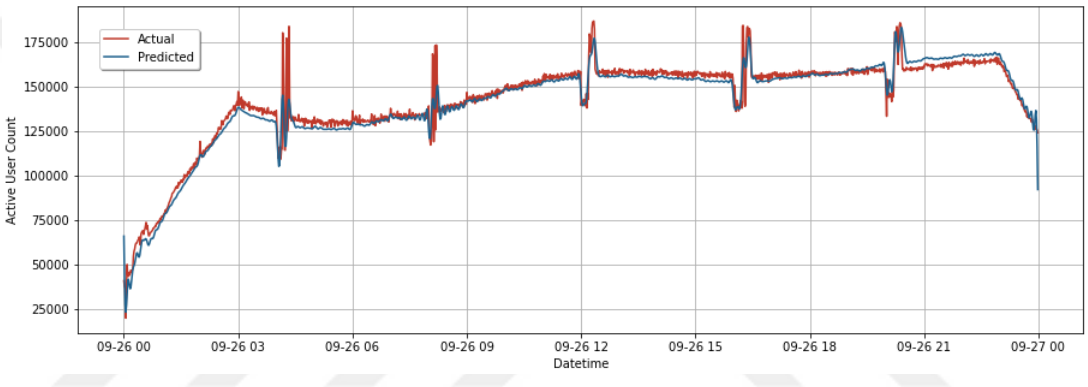


Figure 9.22: Comparison Between Actual and Predicated Active User Data-sets Over One day

it is evident from **Figure 9.21** that the actual and predicted data-sets follow each other. The chart, however, might be insufficient to judge the model. Therefore, calculating the mean absolute percentage error (MAPE),its formula is stated bellow, contributes to delicate knowledge of the way of how the model attains results.

$$\textbf{MAPE} = \frac{\%100}{n} \sum |\frac{y - y\prime}{y}| \tag{9.3}$$

A short explanation of the above equation 9.3 can be illustrated by declaring its parameters. The number of samples is symbolized by n. y' and y are the predicted and actual values, respectively. Finally, the MAPE value of the model that is created to forecast the data is %2.49. In fact, the gained result is stunning and looks like an error-free result. Only, there is no doubt that the repeated behaviour of the data has a major impact that led to such a perfect error value.

10. CONCLUSIONS AND RECOMMENDATIONS

As a conclusion it can be stated that the main target of this thesis, which is extracting significant results and meaningful visualizations from one month DNS data-set taken from an telecommunication enterprise using an optimized YARN based Apache Spark cluster, was completely accomplished. Captured visualizations have played a major role in determining useful information such as situation of DNS servers, customers with connections, distribution of DNS traffic across Turkey neighborhood, types of customers, most visited categories, most used URLs, and suitable places for advertising. In addition, the optimization of the used YARN-based Apache Spark clusters were efficiently done. The values that can be found on Resourcemanager interface were exactly the same as the calculated values of the allocated amount of RAM, number of vCores, number of containers, and parallel tasks which in turn confirms the efficient use of available resources. Therefore, a local based cluster was established with accurate size and features that were specified based on the experiences that were made on cloud. On the other hand, while preparing this project, several information that are related to Big Data was efficiently reported. A comprehensive introduction that included almost all important concepts of big data such as its history, importance, characteristics, and future was adequately reported. In addition, the architecture of the used services such as YARN, HDFS, Apache Spark, TEZ, Hive, ZooKeeper, MapReduce, were precisely discussed. In particular, a detailed practice of optimizing YARN-based Apache Spark cluster was minutely illustrated. In addition, results of various experiments, based on input data size, input data rows, number of operations, output data size, output data rows, and execution time using different cluster sizes were accurately obtained. Moreover, next to different cluster sizes, several node types that have different features such as amount of RAM, number of vCores, Network bandwidth, and Network performance were utilized. An application of time series foresting was applied to the DNS output data in the aim of predicting the future density of the used DNS servers and trying to compensate some of the lost data-sets, if any. Eventually, by using another optimized ELK cluster, the processed data, that is taken from DNS servers, was visualized on an

interactive dashboard. Benefiting from the established dashboard, several choropleth maps, race bar, vertical bar, and pie charts were taken and discussed.

REFERENCES

[1] **Samal, N., & Mishra, N.** (2015). Big data processing: Big challenges and opportunities. *Journal of Computer Sciences and Applications*, *3*(6), 177-180.

[2] **Chen, M., Mao, S., & Liu, Y.** (2014). Big data: A survey. *Mobile networks and applications*, *19*(2), 171-209.

[3] **Alexander, C. A., & Wang, L.** (2017). Big Data Analytics in Identification, Treatment, and Cost-Reduction of Hypertension. *American Journal of Hypertension*, *4*(1), 1-8.

[4] **Garlasu, D., Sandulescu, V., Halcu, I., Neculoiu, G., Grigoriu, O., Marinescu, M., & Marinescu, V.** (2013, January). A big data implementation based on Grid computing. In *2013 11th RoEduNet International Conference* (pp. 1-4). IEEE.

[5] **Slezak, D.** (2015, December). How to Make Big Data Analytics More Interactive?. In *2nd International Conference on Big Data Analysis and Data Mining* 4(3).

[6] **Kubina, M., Varmus, M., & Kubinova, I.** (2015). Use of big data for competitive advantage of company. *Procedia Economics and Finance, 26,* 561-565.

[7] **Bedi, P., Jindal, V., & Gautam, A.** (2014, September). Beginning with big data simplified. In 2*014 International Conference on Data Mining and Intelligent Computing (ICDMIC)* (pp. 1-7). IEEE.

[8] **Franks, B.** (2012). *Taming the big data tidal wave: Finding opportunities in huge data streams with advanced analytics* (Vol. 49). John Wiley & Sons.

[9] **Rajendran, P. K., Asbern, A., Kumar, K. M., Rajesh, M., & Abhilash, R.** (2016). Implementation and analysis of MapReduce on biomedical big data. *Indian Journal of Science and Technology, 9*, 31.

[10] **Yaqoob, I., Hashem, I. A. T., Gani, A., Mokhtar, S., Ahmed, E., Anuar, N. B., & Vasilakos, A. V.** (2016). Big data: From beginning to future. *International Journal of Information Management*, *36*(6), 1231-1247.

[11] **Gu, L., & Li, H.** (2013, November). Memory or time: Performance evaluation for iterative operation on hadoop and spark. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing* (pp. 721-727). IEEE.

[12] **Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F.** (2015). Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proceedings of the VLDB Endowment, 8*(13), 2110-2121.

[13] **Shi, X., Chen, M., He, L., Xie, X., Lu, L., Jin, H., ... & Wu, S.** (2014). Mammoth: Gearing Hadoop towards memory-intensive MapReduce applications. *IEEE Transactions on Parallel and Distributed Systems, 26*(8), 2300-2315.

[14] **Bansod, A.** (2015). Efficient big data analysis with apache spark in HDFS. *International Journal of Engineering and Advanced Technology (IJEAT), 4*(6), 313-315.

[15] **Verma, J. P., & Patel, A.** (2016). Comparison of MapReduce and spark programming frameworks for big data analytics on HDFS. *IJCSC, 7*(2), 180-184.

[16] **Singh, A., Mittal, M., & Kapoor, N.** (2019). Data processing framework using Apache and Spark technologies in big data. In *Big Data Processing Using Spark in Cloud* (pp. 107-122). Springer, Singapore.

[17] **Sang, G. M., Xu, L., & De Vrieze, P.** (2016, December). A reference architecture for big data systems. In *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)* (pp. 370-375). IEEE.

[18] **Pääkkönen, P., & Pakkala, D.** (2015). Reference architecture and classification of technologies, products and services for big data systems. *Big data research*, *2*(4), 166-186.

[19] **Mohammad, A., Mcheick, H., & Grant, E.** (2014, September). Big data architecture evolution: 2014 and beyond. In *Proceedings of the fourth ACM international symposium on Development and analysis of intelligent vehicular networks and applications* (pp. 139-144).

[20] **Li, X., & Li, Y. X.** (2018). Big data and its key Technology in the Future. *Computing in Science & Engineering, 20*(4), 75-88.

[21] **Waal-Montgomery, M. d.** (2015). World's data volume to grow 40% per year & 50 times by 2020: Aureus. https://aws.amazon.com/blogs/big-data/best-practices-for-successfully-managing-memory-for-apache-spark-applications-on-amazon-emr/. **Accessed on 03 may 2021**

[22] **Rialti, R., & Marzi, G.** (2019). *Ambidextrous Organizations in the Big Data Era: The Role of Information Systems*. Springer Nature.

[23] **Yaqoob, I., Hashem, I. A. T., Gani, A., Mokhtar, S., Ahmed, E., Anuar, N. B., & Vasilakos, A. V.** (2016). Big data: From beginning to future. *International Journal of Information Management*, *36*(6), 1231-1247.

[24] **Pence, H. E.** (2014). What is big data and why is it important?. *Journal of Educational Technology Systems, 43*(2), 159-171.

[25] **Sivarajah, U., Kamal, M. M., Irani, Z., & Weerakkody, V.** (2017). Critical analysis of Big Data challenges and analytical methods. *Journal of Business Research, 70*, 263-286.

[26] **Kabir, S. M. S.** (2016). Methods Of Data Collection: *Basic Guidelines for Research: An Introductory Approach for All Disciplines*. (pp. 201-275). Bangladesh, MI:Bangladesh

[27] **Hernandez-Suarez, A., Sánchez-Pérez, G., Toscano-Medina, K., Martínez-Hernández, V., Sanchez, V., & Meana, H.** (2018). A Web Scraping Methodology for Bypassing Twitter API Restrictions. *ArXiv, abs/1803.09875.*

[28] **Zheng, C., He, G., & Peng, Z.** (2015). A Study of Web Information Extraction Technology Based on Beautiful Soup. *JCP*, *10*(6), 381-387.

[29] **Karun, A. K., & Chitharanjan, K.** (2013, April). A review on hadoop—HDFS infrastructure extensions. In *2013 IEEE conference on information & communication technologies* (pp. 132-137). IEEE.

[30] **Buyya, R., Vecchiola, C., & Selvi, S. T.** (2013). *Mastering cloud computing: foundations and applications programming.* Newnes.

[31] **Ghazi, M. R., & Gangodkar, D.** (2015). Hadoop, MapReduce and HDFS: a developers perspective. *Procedia Computer Science, 48*, 45-50.

[32] **Perwej, Y., Kerim, B., Sirelkhtem, M., & Sheta, O. E.** (2017). An Empirical Exploration of the Yarn in Big Data. *International Journal of Applied Information Systems (IJAIS), 12*, 19.

[33] **Dimakopoulos, Vasilis.** (2018). Spark on Hadoop YARN & Kubernetes for Scalable Physics Analysis In collaboration with CERN Openlab, Intel, CMS Experiment, Fermilab.

[34] **Apache Hadoop** (2015). Apache Hadoop NextGen MapReduce (YARN) . https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html **Accessed on 28 april 2021**

[35] **Hortonworks** (n.d). Determine YARN and MapReduce Memory Configuration Settings. https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html **Accessed on 01 may 2021**

[36] **Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A., & Curino, C.** (2015, May). Apache tez: A unifying framework for modeling and building data processing applications. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data* (pp. 1357-1369).

[37] **Ivanov, T., & Beer, M. G.** (2015). Evaluating hive and spark SQL with BigBench. *arXiv preprint arXiv:1512.08417.*

[38] **Karau, H., Konwinski, A., Wendell, P., & Zaharia, M.** (2015). *Learning spark: lightning-fast big data analysis.* " O'Reilly Media, Inc.".

[39] **Sharma, M., Chauhan, V., & Kishore, K.** (2016). A review: Map reduce and spark for big data analytics. *International Journal of Advanced Technology in Engineering and Science, 4*(6), 42-50.

[40] **Veith, A. & Assuncao, A.** (2018). Apache Spark. In *Encyclopedia of Big Data Technologies* (pp.1-5). doi:10.1007/978-3-319-63962-8_37-1

[41] **Feng, W.** (2019). Learning Apache Spark with Python.

[42] **Chambers, B., & Zaharia, M.** (2018). S*park: The definitive guide: Big data processing made simple.* " O'Reilly Media, Inc.".

[43] **Apache Spark** (n.d). Cluster Mode Overview. https://spark.apache.org/docs/latest/cluster-overview.html **Accessed on 12 may 2021**

[44] **Shanmugam, K.** (2019). Best practices for successfully managing memory for Apache Spark applications on Amazon EMR. https://aws.amazon.com/blogs/big-data/best-practices-for-successfully-managing-memory-for-apache-spark-applications-on-amazon-emr/ **Accessed on 11 may 2021**

[45] **Apache Spark** (n.d). Spark Configuration. https://spark.apache.org/docs/latest/configuration.html **Accessed on 12 may 2021**

[46] **Karau, H., & Warren, R.** (2017). *High performance Spark: best practices for scaling and optimizing Apache Spark.* " O'Reilly Media, Inc.".

[47] **AWS** (n.d). Amazon EMR pricing. https://aws.amazon.com/emr/pricing/. Accessed on 15 may 2021

[48] **Deyhim, P.** (2013). *Best practices for Amazon EMR*. Technical report, Amazon Web Services Inc.

[49] **Çakır, A., Alkhanafseh, Y., Karabıyık, E., Kurubaş, E., Bunyak, R. B., & Bahçevan, C. A.** (2020, July). Cloud Based Big Data DNS Analytics at Turknet. In *International Conference on Intelligent and Fuzzy Systems* (pp. 833-841). Springer, Cham.

[50] **Velicer, W. F., & Fava, J. L.** (2003). Time series analysis. *Handbook of psychology*, 581-606.

[51] **Nielsen, A.** (2019). *Practical time series analysis: prediction with statistics and machine learning*. " O'Reilly Media, Inc.".

[52] **Navratil, M., & Kolkova, A.** (2019). Decomposition and Forecasting Time Series in the Business Economy Using Prophet Forecasting Model. *Central European Business Review, 8*(4), 26.

[53] **Choudhary, A.** (2018). Generate quick and accurate time series forecasts using Facebook's Prophet (with Python & R codes). *Analytics Vidya*.

[54] **Taylor, S. J., & Letham, B.** (2021). *Automatic Forecasting Procedure*. Retrieved from https://cran.r-project.org/web/packages/prophet/prophet.pdf

[55] **Taylor, S. J., & Letham, B.** (2018). Forecasting at scale. *The American Statistician, 72*(1), 37-45.

APPENDICES

APPENDIX A: Apache Spark Job and Hadoop YARN ResourceManager Interfaces.
APPENDIX B: Cluster Configurations.

APPENDIX A

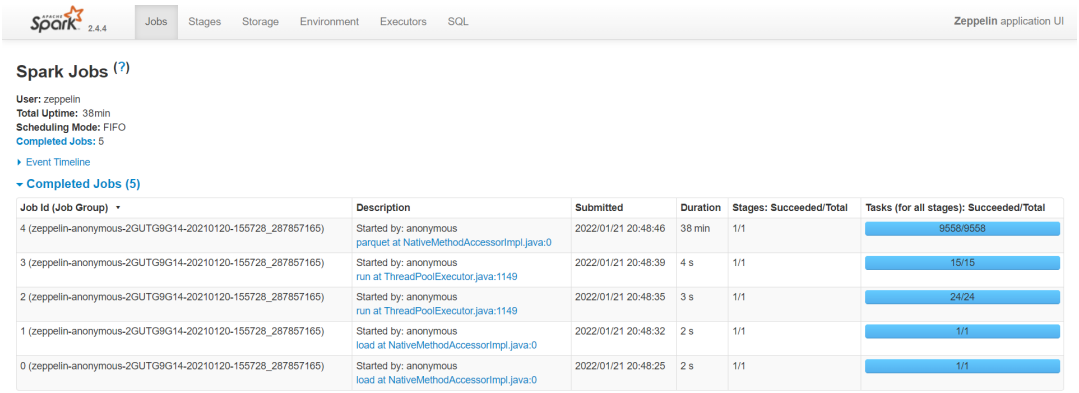


Figure A.1: Apache Spark Job Interface

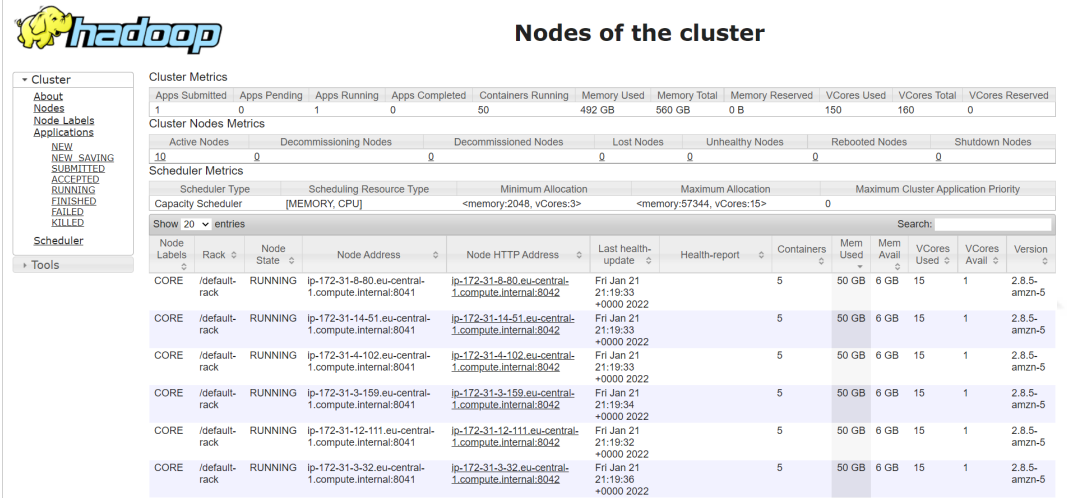
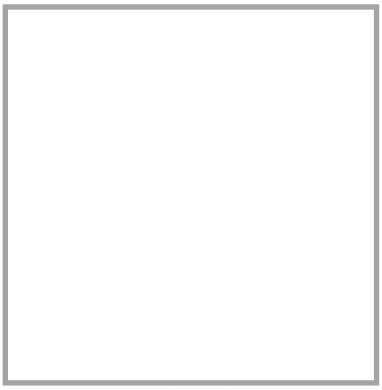


Figure A.2: Hadoop YARN ResourceManager Interface

APPENDIX B


```
[ { "Classification": "capacity-scheduler",
"Properties": {
"yarn.scheduler.capacity.resource-
calculator":"org.apache.hadoop.yarn.util.resource.DominantResourceCalculator"
}
},
{
"Classification": "yarn-site",
"Properties": {
"yarn.nodemanager.resource.memory-mb":"57344",
"yarn.nodemanager.resource.cpu-vcores":"15",
"yarn.scheduler.maximum-allocation-mb": "57344",
"yarn.scheduler.minimum-allocation-mb":"2048",
"yarn.scheduler.maximum-allocation-vcores":"15",
"yarn.scheduler.minimum-allocation-vcores":"3",
"yarn.nodemanager.vmem-check-enabled":"false",
"yarn.nodemanager.pmem-check-enabled":"false"
}
},
{
"classification": "spark",
"properties": {
"maximizeResourceAllocation":"false"
}
},
{
"Classification": "spark-defaults",
"Properties": {
"spark.dynamicAllocation.enabled":"false",
"spark.executor.heartbeatInterval":"60s",
"spark.network.timeout":"800s",
"spark.memory.storageFraction":"0.1",
"spark.memory.fraction":"0.9",
"spark.driver.memory":"9000m",
"spark.executor.memory":"9000m",
"spark.executor.cores": "3",
"spark.driver.cores": "3",
"spark.executor.instances": "50",
"spark.shuffle.compress": "true",
"spark.shuffle.spill.compress": "true",
"spark.scheduler.barrier.maxConcurrentTasksCheck.maxFailures": "5",
"spark.storage.level":"MEMORY_AND_DISK_SER",
"spark.serializer":"org.apache.spark.serializer.KryoSerializer",
"spark.sql.adaptive.enabled":"true",
"spark.executor.extraJavaOptions":"-XX:+UseG1GC -XX:+UnlockDiagnosticVMOptions
-XX:+G1SummarizeConcMark -XX:InitiatingHeapOccupancyPercent=35 -verbose:gc
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'",
"spark.driver.extraJavaOptions":"-XX:+UseG1GC -XX:+UnlockDiagnosticVMOptions
-XX:+G1SummarizeConcMark -XX:InitiatingHeapOccupancyPercent=35 -verbose:gc
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'" } } ]
```

CURRICULUM VITAE



Name SURNAME : Yousef ALKHANAFSEH

EDUCATION :

- **B.Sc. : 2020, Istanbul Technical University, Engineering Faculty, Electrical Engineering**
- **M.Sc. : 2022, Istanbul Technical University, Engineering Faculty, Electrical Engineering**

PUBLICATIONS ON THE THESIS:

• Çakır, A., **Alkhanafseh, Y.**, Karabıyık, E., Kurubaş, E., Bunyak, R. B., & Bahçevan, C. A. (2020, July). Cloud Based Big Data DNS Analytics at Turknet. In *International Conference on Intelligent and Fuzzy Systems* (pp. 833-841). Springer, Cham.

OTHER PUBLICATIONS:

• **Alkhanafseh, Y.**, & Akinci, T. C. (2021). A Python-Based Interface Design for Electric Power System Education. *International Journal of Smart Grid and Sustainable Energy Technologies*, 4(1), 163-168.

• Turkmen, A., Bahcevan, C. A., **Alkhanafseh, Y.**, & Karabiyik, E. (2020). User behaviour analysis and churn prediction in ISP. *New Trends and Issues Proceedings on Advances in Pure and Applied Sciences,* (12), 57-67.