Empirical evaluation of parallelizing correlation algorithms for sequential telecommunication devices data

Kevin Kim

A thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science (Electrical and Computer Engineering)
Concordia University
Montréal, Québec, Canada

April 2020

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By:	**Kevin Kim**
Entitled:	**Empirical evaluation of parallelizing correlation algorithms for sequential telecommunication devices data**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Wahab Hamou-Lhadj	Chair
Dr. Peter Chen	External Examiner
Dr. Yan Liu	Supervisor

Approved by Sheryl Tablan ________ Graduate Program Director

June 21st 2019 Dr. Amir Asif ________ Dean of Faculty

Abstract

Empirical evaluation of parallelizing correlation algorithms for sequential telecommunication devices data

Kevin Kim

Context: Connected devices within IoT is a source of generating big data. The data measured from devices consists of large number of features from hundreds to thousands. Analyzing these features is both data and computing intensive. Distributed and parallel processing frameworks such as Apache Spark provide in-memory processing technologies to design feature analytic workflows. However, algorithms for discovering data patterns and trends over time series are not necessarily ready to cooperate issues such as data partition, data shuffling that rise from distribution and parallelism. **Aim:** This thesis aims to explore the relation between algorithm characteristics and parallelisms as well as the effects on clustering results and the system performance. **Method:** System level techniques were developped to address particularly the data partition, load-balancing and data shuffling issues. Furthermore, these techniques are applied to adopt clustering algorithms on distributed parallel computing frameworks. In the evaluation, two workflows were built in which each consists of a clustering algorithm and its corresponding metrics for measuring distances of any two time series data. **Result:** These system level techniques improve the overall performance and execution of the workflows. **Conclusion:** The distribution and parallel workflows address both algorithmic factors and parallelism factors to improve accuracy and performance of processing big time series data of connected devices.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. Yan Liu for the continuous support of my M.A.Sc. study and research, for her patience, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. She went above and beyond to encourage me throughout my high and lows throughout these years and I could not have imagined having a better advisor and mentor for my M.A.Sc. study.

Special gratitude goes out to the Thomas Triplet, Merlin Davies, and David Cote for giving me the incredible learning opportunity to work along their side and for providing me guidance throughout my research.

I am forever grateful for all the close friends and siblings who have provided me ongoing encouragement and emotional support along the way.

Last but not least, I would like to thank my amazing parents for supporting me throughout all my life and constantly putting my needs before their own so that I could have a better life and education.

Contents

List of Figures

Chapter 1

Introduction

In this era of Big Data, 90 % of the data today has been created in the past two years and the global population is generating 2.5 quintillion bytes of data daily [45]. In 2017, there were 3.7 billion users on the internet generating a vast amount of data constantly whether they are at work, exercising, commuting, running errands, watching their favorite series and even sleeping. There are various definition and usage of the term Big data. The use of these terms generally refers to "Information assets characterized by a High Volume, Velocity, and Variety to require specific Technology and Analytical Methods for its transformation into Value" [46]. This flood of data becomes a liability unless the right technologies and analytical methods are adopted to navigate through this flood of information in order to drive value and insight.

The impact of data are numerous and are benefiting those who embrace this challenge and make sense of it. In this digital age, businesses are able to bring this further by processing more data, from many more sources, faster and with better accuracy than ever before. For many reasons such as the ones stated, Data has been declared as a new class of economic asset along with currency and gold. With the right technology and skilled workforces, businesses can drive value by becoming data-driven by using this data to make better decisions and to drive operational improvements through data analysis. Bigger players who had a leg start in the data business bring this further by seeing opportunities to commercialize the data. The data that they generated for their own businesses that gave them a competitive edge over their markets are many times highly sought for and can be accessed and sold to other related or unrelated businesses. In 2015, IBM acquired most of the Weather Network, which owns weather.com and weather underground for a reported US $ 2 billion for the quality and vast historical weather dataset which can be used to sell to other companies who need to know about how the weather influences their businesses.

Data analytics, the process of drawing conclusions and confirming a hypothesis by examining the data, is not a novel science and has benefitted and drove innovation for many companies from various industries for many decades. The novel aspect which is a differentiator and the fundamental challenge of big data is its volume. The rate of generation of the data has outpaced the standard capability of processing and storing it. Traditional technologies were never conceived to handle, store and process such vast volume of data.

In the midst of this era, technologies have gained many grounds to face these challenges. There are various industry adopted tools and large vendor products which tackles these challenges through parallel processing enabled technologies. The idea of parallel processing is to tackle the scalability challenge by running the same computational task on multiple processors at the same times in order to achieve a common task. By leveraging a computing cluster, multiple processing nodes acting as a single computing unit, these massive datasets can now be broken down, sent to each processing nodes and processed in parallel which would otherwise be limited by the traditional computing technologies.

A popular open source tool that provides a solution to big data challenges is Hadoop which quickly became an industry standard for big data and parallel processing.

1.1 Internet of Things

Internet of Things (IoT) stands for the evolution of home, mobile and embedded application being connected to the internet to integrate greater computing capabilities. Millions of devices have already been connected to the internet. These intelligent devices and groups of devices acting as systems share data over the cloud. The data generated from these devices have been analyzed to transform the operations of many industrial domains [1] [27] [28] [34].

IoT promises much application to human life, making it easier and safer and smarter. Applications range from smart cities, smart homes, transportation, energy, smarter healthcare, and many more. An example of smarter healthcare is the continuous monitoring of a patient health condition through an IoT device, where the physiological data are gathered through sensors and sent to the cloud for analysis. Once the data is stored and analyzed, it is returned to both patient and health care professional for a constant flow of information and more effective health care measures. Another application is the use of smart home IoT products to make human lives easier, more convenient and more comfortable. Connected home energy equipment such as lighting and thermostat can be tuned to the tenant's preferences when he is present or tuned down during his absence to lower electric bill and energy consumption.

Data gathered directly from these devices usually are unformatted and have thousands of features. Such raw data need to go through transformation, pre-processing and intensive analysis in order to understand the underlying insights of data. A common type of data generated is time-series, which is defined as data that is acquired at a fixed interval over a period of time. Examples are datasets of the home temperature of a smart thermostat over the course of many days; the heart rate gathered by a smart wristband over a run; the GPS location of a smart car driver over the span of the run, and networking infrastructure metrics from a telecommunication service provider reflecting the demand incurred by the end-users. On their own, individual devices can be processed by any traditional analytical tools. However, with millions of devices each producing a sequence of time series [39] [40], more scalable tools are essential for applications relying on timely and accurate analysis. An illustrating case of the above challenges is observed from data sets collected by a telecommunication service company based in the USA where datasets on networking devices have been collected to measure the key performance indicators of the operational networks. A record of each feature of the device is producing a reading at each time stamp. There are over 350 features of each type of devices. These features have correlations but with no prior information. The motivation behind the research methods in this thesis is to identify patterns within this dataset and to find correlations between different devices. Such findings lead to applications that yield business values.

1.2 Challenge

With such a vast amount of data generated daily, fully realizing the potential of this new asset does not come without a battle. There are many technical challenges when trying to unlock its value. Creation of value through big data is a process made of multiple stages [47] [48] which usually consists of data acquisition, information extraction and cleansing, data integration, modeling and analysis, and interpretation. Challenges are found at any step in this process.

Moreover, the complexity of the data itself is overwhelming. High Dimensionality, a term describing data with a large number of attributes, are often encountered when dealing with Big Data. High Dimensional data sets could be images, videos, medical records, historical stock market time-series for instances. For scalability and performance reasons, only a curated view of the raw dataset should be processed for optimal outcomes.

Even curated, these data sets are too large for traditional modeling and analysis tools. When dealing with a large dataset, it is often required to leverage a cluster of resources to process, transform and apply models on these datasets. Traditional algorithms need to be adapted and tuned for these parallel workflows. Algorithms which were designed to have all the data available on a single node will not yield statistically accurate results on a parallel and distributed environment without considerable effort to parallelize and tune algorithms for this distributed environment.

The research questions of scalable analysis raise from three aspects upon distributed and parallel development, including:

1. **High dimentionality** - How to process multiple features in parallel so that operations on features such as *join* or *group* are executed on partitioned datasets?

2. **Unknown data correlation** - What are the metrics to measure the correlation? How relevant data features are identified for an application?

3. **Algorithm diversity** - How does an algorithm affect parallelism, potentially causing data skews and eventually degrading scalability?

This thesis presents parallel clustering methods to identify the correlation pattern. Techniques are developed to realize a clustering method using different types of algorithms. Two parallel workflows are designed where each consists of clustering algorithms and its corresponding distance metrics adapted for the distributed parallel computing framework. To horizontally scale the system, techniques are developed to address data partitioning, load-balancing and data shuffling. Each clustering workflow and their integrated algorithms are extensively evaluated to identify the factors related to data locality, data skew and overall system scalability.

1.3 Contribution

The work presented in this Thesis is primarily focused on understanding the data set in order to build accurate production ready applications at a large scale.

The contributions of this thesis are four-fold:

1. Developed scalable and parallel workflows by combining distance measures and statistical algorithms for time series analysis with system performance evaluation;
2. Integrate statistical models into parallel workflows and developed a parallel version of the Neighbor-Joining Hierarchical Clustering Algorithm;
3. Identify limitation from experiments on parallelism raised by the relation between workflows and the underlying system framework;
4. Provide Spark tuning recommendations on performing data dependent partitioning when parallelizing algorithms using Spark's framework

The first contribution is the creation of parallel workflows, a sequence of data processing tasks, using Spark's Distribution framework to enable time series processing, time series comparison through similarity measures and time series clustering capabilities through unsupervised learning algorithms. These capabilities were selected and prioritized based on the complexity and the structure of the data in-scope of this study. The time series processing capabilities, referred as pre-processing in this study due to being the first step of the parallel workflows, consists of turning raw log data sampled at inconsistent time intervals into a standardized data format that can be further processed. This standardized format for a time series object consists of a series of timestamps, a series value corresponding to each timestamp, and a label. Additional processing is enabled through binning, which reduces the dimensionality of the time series, or concatenating, which group related time series by appending them sequentially and creating an artificially longer timeline. Once standardized, comparison and clustering can be achieved. Time series are compared through similarity measures and in this study, the comparison capabilities enabled were parallel comparisons of time series through Euclidean distance and fast Dynamic Time warping, an optimized version of this commonly used measure. Results are stored using a distributed Distance matrix. An intermediate process is enabled on the distance matrix to reduce its dimension before being clustering through Principal Component Analysis. In this study, clustering capabilities enabled were parallelized DBSCAN and Neighbor-joining as these were suited for identified unknown patterns in the dataset. These capabilities were enabled in Spark in this study and can be selected and tuned based on the dataset. Although each capability at each step of the workflows can be combined differently, two distinct use case were identified for comparative purposes which are detailed in section 4 and 5 respectively.

The second contribution builds on top of the first contribution. In the process of creating parallel workflows and enabling parallel processing of time series, there was an inherent need to perform and apply statistical algorithms in a distributed environment. Statistical models and algorithms are traditionally derived and computed on a single computer with the assumption of processing manageable sized datasets on traditional non-distributed technologies. In this thesis, the second contribution is adapting statistical algorithms for distributed and parallel application using Spark's Framework for enabling analysis of large time series datasets. In this thesis, Fast Dynamic time Warping, Principal Component Analysis, Pearson's Correlation, and parallel DBSCAN implementation were leveraged from open source libraries, integrated and adapted to be enabled in the spark workflows for parallel time series analysis. In the case of Neighbor-joining (NJ), there was no available open source parallel version of this algorithm in Spark. In this thesis, a parallel version of this algorithm is presented which was implemented in Spark. This algorithm was inspired by the latest publication and research papers on parallelizing the NJ, which is then integrated and enabled as part of the Spark Time series analysis workflows.

The third contribution consists of identifying limitation from experiments on parallelism raised by the relation between workflows and the underlying system framework. Spark excels at fast in-memory processing. Optimal performance and scalability can only be achieved if the application itself can be adapted to its system's framework. In this study, parallelizing Neighbor-Joining, a matrix-based recursive hierarchical clustering algorithm, has shown to be a challenge for multiple reasons. For starters, the recursive nature of the algorithm slows down the execution of the overall application. Dependencies on its previous iteration prevent the ability to execute each iteration in parallel. Parallelization was achieved "intra"-iterations, was the calculation on each iteration was processed in parallel. The results of each iteration needed to be completed first in order to proceed to the next iteration. The second challenge was to perform more granular matrix manipulations in Spark. At the time of this writing, Spark's default data structure was not suitable for matrix-based algorithms out of the box. Custom representation of a matrix using Sparks RDD is presented in this thesis which provided more flexibility to perform the required algebra to implement the Neighbor-Joining algorithm in Spark.

The fourth contribution consists of performance tuning recommendations and key lessons learned that were identified throughout the development and system evaluation of these Spark parallel workflows.

- Leverage the Spark broadcast method to reduce data movement which translates by shortening the execution time by removing unnecessary shuffle read and shuffle write time for the related tasks.

- Redundant RDD calculations should be discarded to remove repetitive. For instance, due to its diagonal symmetry of a distance matrix, calculating a distance matrix translates into duplicated spark tasks which can be discarded and inferred by either upper or lower diagonal.

- Naive Recursive algorithm implementation in spark can exponentially increase the complexity of each iteration. Internally, due to its in-memory processing framework, Spark has to re-calculate every previous iteration before addressing the current iteration. Temporally storing in-memory the output of each iteration truncates the spark lineage which prevents exponential complexity of each iteration.

- Different spark methods such as groupbyKey, reducebykey and Union can create unwanted data skew in the partitions. Adequate use of coalescing can solve data skew hurdles by optimizing data partitioning and reducing shuffle read and shuffle write time for the related tasks.

The publications from this thesis includes:

- **IEEE Access Paper** entitled *"Distributed and Parallel Clustering Methods to Discover Patterns of IoT Device Time Series Data" (under major revision at the time of this writing)*
- **US Patent** entitled *"Systems and methods for automated feature selection and pattern discovery of multi-variate time-series" - Filing Date: September 9 2018*

The remaining of this thesis is structured as follows. Chapter 2 will provide an overview of related topics and outlying a lack of similar papers using the methods and approach. Chapter 3 will present the problem statement which was the root of this research where more details on the datasets and its challenges to transform it from raw data into application-ready data are discussed. Chapter 4 and 5 go into the details of the two workflows derived. Chapter 6 presents the system evaluation and the results of the experimentations. Chapter 7 will present reflections and discussions. Chapter 8 presents closing words on this thesis and the related work. Finally, Chapter 9 provides more implementations details in the Annex and figures which may complement the understanding of the audience.

Chapter 2

Background

This chapter provides preliminary background material foundational for the theory and study derived in this thesis. The chapter starts by presenting the Parallel and Distribution Paradigm, data partitioning, data locality, and data skew which are key principles and challenges often encountered in distributed environment applications. Then, the chapter presents the Map Reduce model, a key concepts which contributed heavily to make scalable big data processing possible. Finally, the chapter provides a brief literature survey of similar publications around the topics of IoT, time series analysis research space, and the existent tools for time series analysis.

2.1 Parallel and Distribution Paradigm

Mining patterns on this dataset involve pair-wise operations of all the data records. The volume of the dataset explored in this thesis is only a small portion of the production environment data. The preprocessing operations as Extract-Transform-Load (ETL) and the machine learning algorithms become data and computing intensive [2]. Parallelism and distribution are both computing paradigms that address large data volume applications. Parallelism is the process of launching multiple computing instances of the same task in parallel. Distributed computing stems from the need to leverage a networked cluster of processing nodes in order to achieve a common computing goal. Parallelism can be achieved on both a single multi-core node or on a cluster of commodity nodes. By leveraging the elasticity of cloud computing that computing nodes are added and removed on demands, it is obvious that combining both computing is more scalable than single node parallelism. Under this paradigm, the dataset as the input is partitioned over a number of working nodes, and all the working nodes are executing operations in parallel.

Frameworks such as Apache Hadoop, Apache Spark, and Apache Storm provides various types of distributed parallel runtimes that allow the programming of data processing and analysis operations. In this thesis, Apache Spark [29] was selected for building the distributed and parallel data operations that involve functions for ETL and machine learning algorithms. This choice is justified in three folds. Ease of access to a data center that has virtual instances that contain a full stack of software for running a Spark cluster was the first deciding factor. Second, the programming model of a resilient distributed dataset (RDD) of Spark is suitable to develop the proposed data clustering methods with data models and the associated operations requirements. Finally, the machine learning library of Apache Spark was beneficial to implement critical steps of the clustering methods.

2.2 Data Locality

In its general term, Data locality refers to the process of moving the data close to where the computation is performed in order to maximize throughput and faster processing of the data. In a distributed environment such as Spark data locality can have a significant impact on the jobs performance [49]. Generally, the closer the data is to the computation, the faster the job tends to be. In Apache Spark, the different levels of locality are presented below in order of closest to farthest:

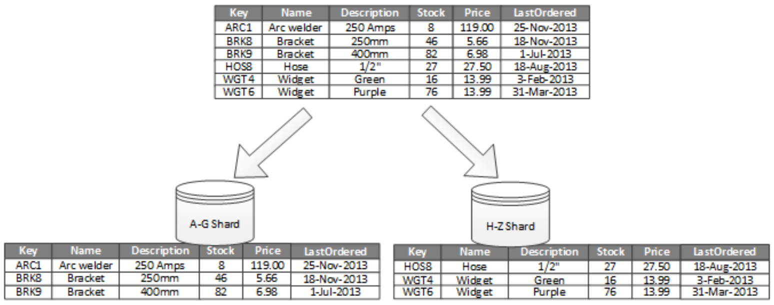
1. Data is in the same JVM as your code;
2. Data is on the same node as your running processes. Data may need to move between processes on the same node;
3. Data is on the same rack server. Data may need to be moved through the network.
4. Data is anywhere on the network and needs to be sent over the network.

By default, Spark tries to optimize data locality whenever possible by waiting for a free CPU where the data is found. If it times out, the data is then moved to the CPU. Spark provides the ability to overwrite default behaviors specifying which level of data locality it should prioritize.

2.3 Data Partitionning

Data partitioning is the process of dividing large chunks of data into more manageable smaller chunks and accessed separately [50]. This has multiple benefits on scalability and performance when applied configured optimally for based on your dataset and the complexity of your computation. Big Data frameworks are built on the key concept of data partitioning and processing and by default partitions large data chunks and processes seamlessly. There are three strategies to partition data:

1. **Horizontal partitionning** (Sharding) where each partition is in a different data store but has the same schema (Figure 1).
2. **Vertical partitionning** where each partition holds a subset of fields for items in a data store. The select field subset can be designed by a pattern of usage such as frequent to less frequent access (Figure 2).
3. **Functional partitionning** where each partition is organized by the bounded context in the system (Figure 3).

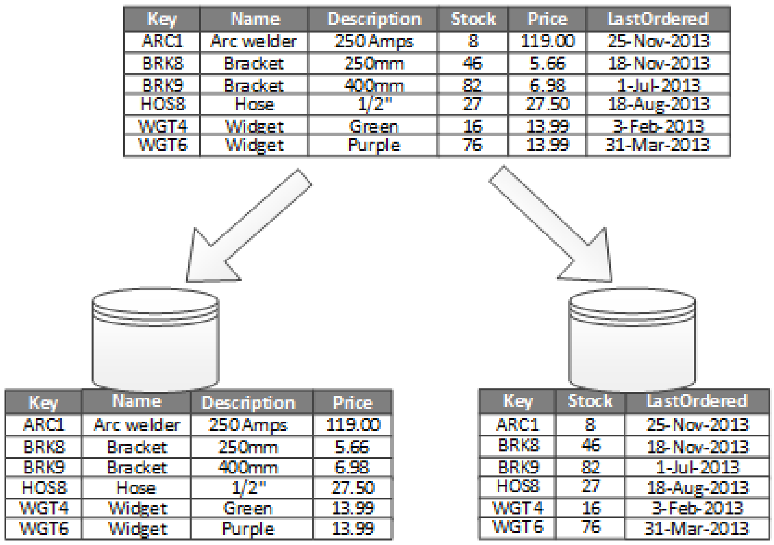


Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Figure 1: Sample of Horizontally partitioning (sharding) data based on a partition key. [50]



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

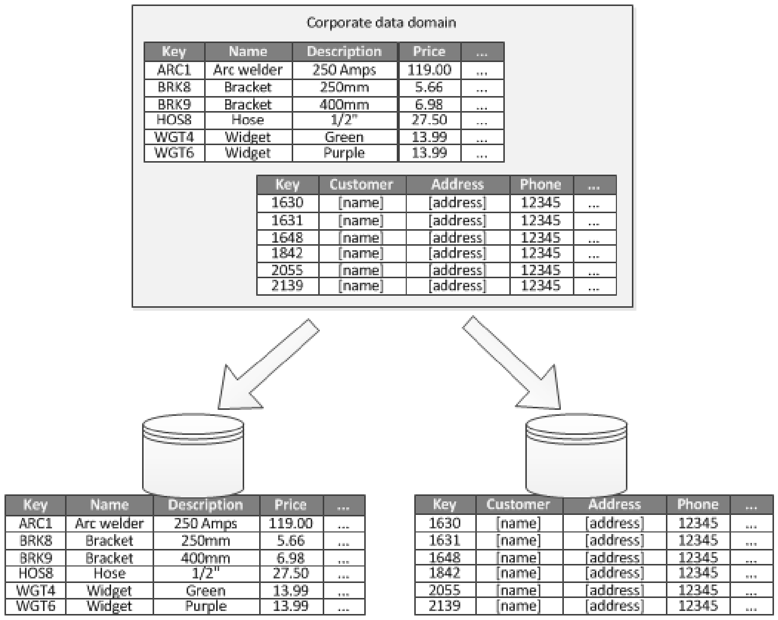
Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

Figure 2: Sample of Vertically partitioning data by its pattern of use. [50]

Big data frameworks such as Hadoop and Apache spark automatically partitions large datasets. In Spark, Spark processes data through RDD's - resilient distributed datasets which divide datasets and spreads them out across multiple nodes because they cannot fit a single one. By default, the framework automatically performs the partitioning across the cluster and resources available at the time of the execution.

2.4 Data Skew

In its general term, Data skew primarily refers to the nonuniform distribution of a database [69]. In the context of distributed and parallel computing, data skew represents a condition of uneven distribution of data across the cluster [52]. Data skew can severely impact the performance of a job and the cluster as some nodes will store more data than others which negatively affects the queries.



Key	Name	Description	Price	...
ARC1	Arc welder	250 Amps	119.00	...
BRK8	Bracket	250mm	5.66	...
BRK9	Bracket	400mm	6.98	...
HOS8	Hose	1/2"	27.50	...
WGT4	Widget	Green	13.99	...
WGT6	Widget	Purple	13.99	...

Key	Customer	Address	Phone	...
1630	[name]	[address]	12345	...
1631	[name]	[address]	12345	...
1648	[name]	[address]	12345	...
1842	[name]	[address]	12345	...
2055	[name]	[address]	12345	...
2139	[name]	[address]	12345	...

Key	Name	Description	Price	...
ARC1	Arc welder	250 Amps	119.00	...
BRK8	Bracket	250mm	5.66	...
BRK9	Bracket	400mm	6.98	...
HOS8	Hose	1/2"	27.50	...
WGT4	Widget	Green	13.99	...
WGT6	Widget	Purple	13.99	...

Key	Customer	Address	Phone	...
1630	[name]	[address]	12345	...
1631	[name]	[address]	12345	...
1648	[name]	[address]	12345	...
1842	[name]	[address]	12345	...
2055	[name]	[address]	12345	...
2139	[name]	[address]	12345	...

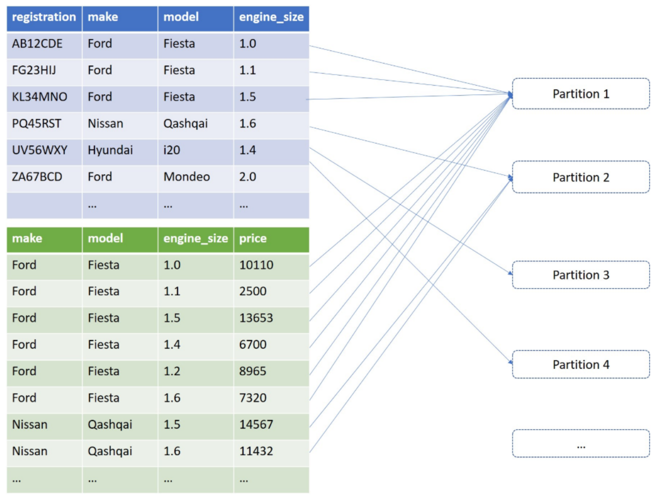
Figure 3: Sample of Functionally partitioning data by bounded context or subdomain. [50]

Processing with Apache Spark default partitioning [53] configurations can present skewed data due to aggregation operations involving data shuffle.

In Apache Spark, symptoms of data skew are noticeable when a tasks, a partial computation of part of a spark job, takes longer than its other tasks or is stuck and hangs the whole job. A Symptom of data skew can be observed when all tasks are completed but one outstanding task of the same type can hold back and stall a job for hours.

Data skew can be caused for various reasons. The most common source of data skew using Spark's distributed framework arises when the data prior to an operation requiring a shuffle such as a join was already skewed as depicted in Figure 4. By default, Spark hashes the join column and sorts it and tries to keep the records with same hashes in a single partition [62]. The example in Figure 4 depicts a join operation between two sample tables [62] with car registration and car maker data. Analysis of the Two tables reveals a nonuniform distribution favoring entries values $Ford$ and $Fiested$ in the $Make$ and *model* columns. When performing a join operation using the *make* and *model* column as join keys, the default spark behavior will attempt at partitioning entries with common entries within the same partition. As depicted in Figure 4, due to the naturally skewed starting tables, the outcome of the join operation will aggregate the majority of the rows into $Partition1$ and the task assigned to this partition will have a longer execution time due to the greater amount of data to process. Data skew can arise for a multitude of reasons based on the dataset and the operations performed as depicted by this simple scenario. Data skews will negatively impact the spark program as the computing cluster is not leveraged optimally.

The most intuitive approach at solving a data skew scenario is to ensure prior to an operation involving data shuffle such as reduceBy, groupBy, Join, and more to partition the data evenly. Spark



registration	make	model	engine_size
AB12CDE	Ford	Fiesta	1.0
FG23HIJ	Ford	Fiesta	1.1
KL34MNO	Ford	Fiesta	1.5
PQ45RST	Nissan	Qashqai	1.6
UV56WXY	Hyundai	i20	1.4
ZA67BCD	Ford	Mondeo	2.0
	...	...	...

make	model	engine_size	price
Ford	Fiesta	1.0	10110
Ford	Fiesta	1.1	2500
Ford	Fiesta	1.5	13653
Ford	Fiesta	1.4	6700
Ford	Fiesta	1.2	8965
Ford	Fiesta	1.6	7320
Nissan	Qashqai	1.5	14567
Nissan	Qashqai	1.6	11432
...	...	...	...

Figure 4: Sample skewed Spark RDD [62]

provides built-in methods to facilitate this tuning process of repartition through the repartition and coalesce method. These methods will perform data repartition based on the rule provided. Although proper partitioning of the data can neglect the hurdle of data skews, improper use of these methods can backfire and perform the opposite desired result and become a performance bottleneck. In later sections of this thesis, data skew scenarios were encountered and the approach and techniques used to solve them are presented in the System Evaluation chapter.

2.5 Map Reduce Model

Map Reduce is a programming paradigm that made possible processing scalability across a large number of nodes. This programming model was the solution to process terabytes of data in parallel to achieve faster analysis and was first used and built at Google in 2003 to analyze their search queries [55].

Map Reduce is at the heart of Apache Hadoop, the widely popular framework for distributed storage and big data processing. Map Reduce is at the core component and integral part of this framework's key functionalities.

MapReduce facilitates the use of parallel and distributed systems. It can concurrently process petabytes of data by dividing it into smaller chunks and automatically processing them in parallel on Hadoop commodity servers. By default, the program does not have to configure data partitioning, scheduling of the program on all the nodes of the cluster, handling machine failure, and inter-machine communication as these are all handled automatically by the runtime system. [57]

Figure 5 outlines an example of how MapReduce processes data in parallel. MapReduce can be defined by two key sequential functions: map and reduce. Figure 5 presents a sample use case for counting the recurrence of each letter from the input dataset.

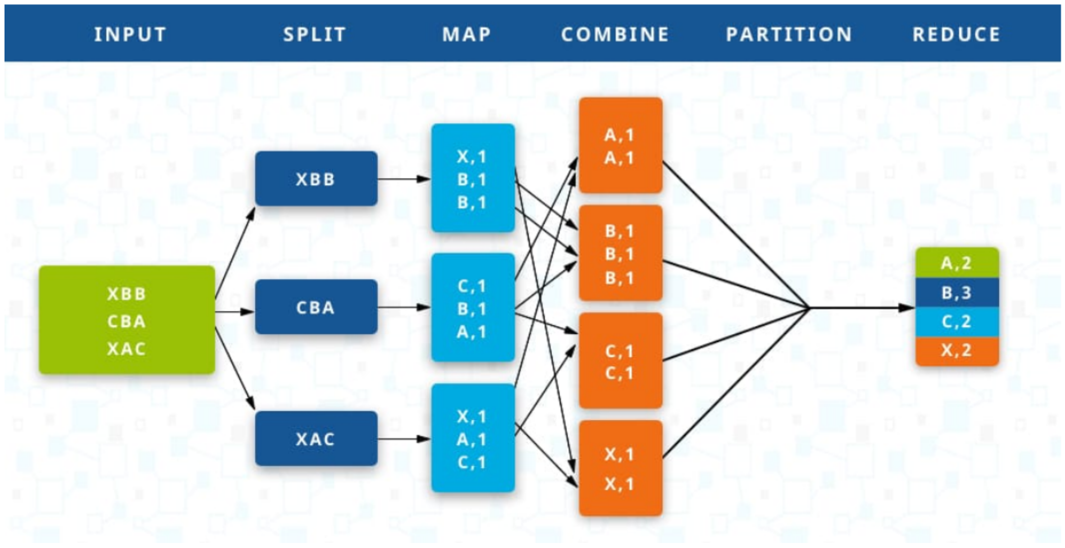


Figure 5: Sample diagram portraying how map reduce works [56]

- **Map** - The input data is first split into smaller blocks. Each block is then assigned to a mapper for processing. The goal of the mapper function is to map each input key/value pair to a set of intermediate key/value pairs. In the example, each block's key's are mapped to the integer 1.
- **Reduce** - When all mappers complete processing, the framework shuffles and sort the results before passing them on to the reducers. A reducer cannot start while a mapper is still in progress. The reducer has 3 primary phases: shuffle, sort and reduce. In this example, the output of the mapper is shuffled and grouped by common key, sorted then all key reduces are aggregated on a single node (reduced) to provide the count of each letter in the input dataset.

Generally, there are two key operations, but MapReduce can have optional intermediary steps such as **Combine** and **Partition** which reduces the output on each mapper and defines how data is presented to the reducer respectively.

2.5.1 Map Reduce and Apache Spark

Apache Spark is a system developed by UC Berkeley as a research project and became part of the Apache umbrella in June 2013. Apache Spark is a general-purpose data processing engine suitable for in-memory processing. This capability is essential and more suitable for performance advance analysis and parallel implementation of algorithms. [59]

Apache Spark Framework comes with a suite of impressive tools such as machine learning tool M Lib, structured data processing, Spark SQL, graph processing took Graph X, stream processing engine called Spark Streaming, and Shark for fast interactive question device as presented Figure 7 reinforcing its position in advance analysis and parallel algorithms implementation over Hadoop.

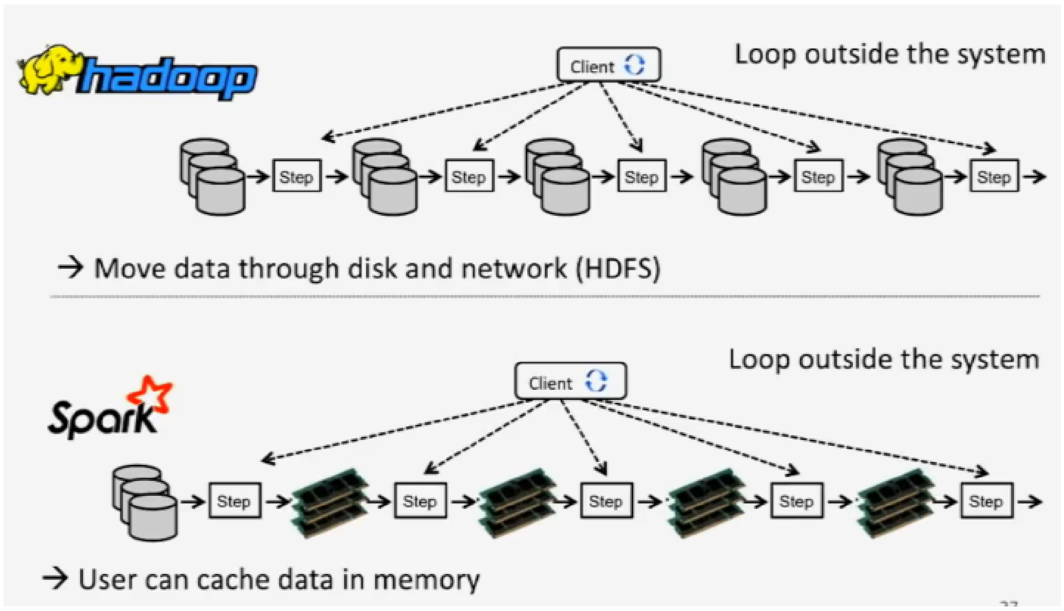


Figure 6: Apache Spark vs Hadoop's Map reduce data processing comparison [60].

Although Apache Spark is more suitable than Hadoop for parallel algorithm implementation, they should not be seen as competitors but as complements. Spark provides capabilities that Hadoop lacks, while Hadoop has features such as a distributed file system and strong batch processing to complement Spark. [59].

2.6 Related Work

This section provides an overview of the related literature around the challenges and the available solutions to analyze Internet of Things datasets, more specifically time series.

2.6.1 Internet of Things Challenges

Challenges from mining large datasets from IoT devices have been recognized in aspects of system capabilities, algorithmic design and business models [3], [7] [28] [35]:

- Analytics Architecture - What is the optimal architecture for data analytics remains unclear;
- Distributed Machine Learning - Standard machine learning techniques are not trivial to deploy in a distributed environment and thus require research to scope the problems and generate suitable solutions;
- High Dimensionality - Handling high dimensionality of data requires approaches to compression, sampling, and feature engineering that trade-off information accuracy and computing capability.

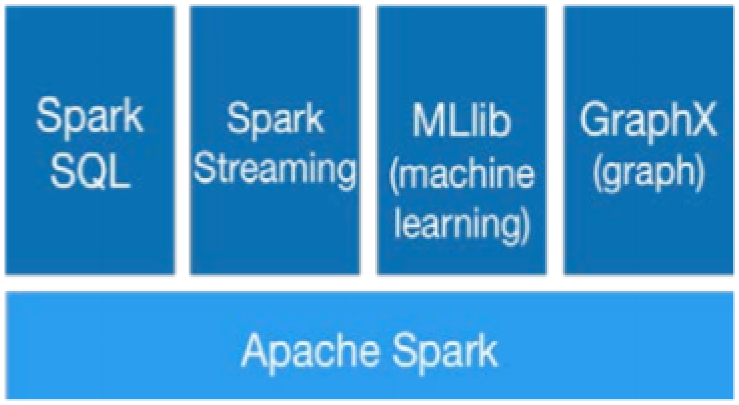


Figure 7: Apache Spark Capabilities [68]

2.6.2 Machine Learning on the Internet of Things

The proliferation of IoT devices capable of sensing, measuring, inferring and sharing these pieces of information across platforms is fueled by a variety of enabling wireless technologies transforming the internet into a fully integrated one [16]. Machine learning of IoT device data involves analyzing patterns in large time series to discover hidden information using suitable algorithms based on the problem and the application proposed [17] [18]. These datasets come from various sources, sensors, and devices in a multitude of formats. The machine learning techniques include classification, clustering, association analysis, time series forecasting, and outlier detection [3] Although there are many personal, profession and economic benefits from the rise IoT, there are also various challenges associated with the development of IoT [19] [20].

2.6.3 Time Series Analysis Surveys

The interest of time series analysis has been on the rise for numerous reasons as there are numerous world application which output this type of data. The rise of Big data and IoT is contributing to the effervescence in this research space and is generating a lot of research interest in finding methods to make sense of Time series datasets [34] [36] [38]. A paper named ”Recent techniques of clustering of time series data: a survey.” [41] surveyed time series clustering papers in various domains such as science, engineering, finance and more summarizes the permutation of distance measures and clustering algorithms researched and presents their limitations. This same paper also performed a survey on the most relevant 50 time series paper and attempted at reproducing all their results. Their conclusion has been that there is a need for a benchmark in this space and that the results claimed in most of the papers do not have significant value in real-world applications due to data bias and experimental flaws. In another paper named ”A Decade Review” [42], a survey is performed on this research space through another angle which seeks on summarizing the research done on time Series time series clustering. The conclusion was that the work in this space addresses two main focus: the ability to reduce the high dimensionality of Time series for the use of conventional clustering algorithms and distance measure evaluation. The findings in these survey papers have served as general guidelines to pivot this thesis in the right directions. The derived workflows offer a novel approach at clustering times series with distinct and novel combinations of similarity measures and clustering methods. The work presented in this thesis stemmed from the work performed between academia and an industry partnership. Contrary to what most research in this space, the objective of this research has been to drive value through real-world applications using industry-wide technologies and tackling common challenges.

2.6.4 Time Series Test Data Survey

There is a lack of substantial literature around Time series analysis for large scale data. The literature in this space use either proprietary data not available to the community or open source datasets available which are collectively in an archive [43] of papers. A call to the community presented as a survey [44] of 340 recent papers on time series revealed that 89% of the test data used could fit on a 1.4 MB floppy disk.

2.6.5 Tools for Time Series Data Analysis

A number of open source and commercial tools [37] with rich sets of libraries are available for time series analytics [12] [13] [14] [15]. Programming languages such as R and Python both support comprehensive algorithms and methods designed for statistical computing and visualization [6]. Apache Moa is a non-distributed data stream mining framework that includes implementations of classification, regression, clustering and frequent pattern mining [5]. Apache Mahout is a distributed data mining platform based on MapReduce/Hadoop [4] in the batch mode. Therefore, Mahout is not directly suitable for data stream analysis. To overcome the limitations of Mahout on data streams, SAMOA [25] features a pluggable architecture that runs several distributed stream processing engines such as Storm, S4, and Samza. In addition, SAMOA (Scalable Advanced Massive Online Analysis) is a library for many machine learning algorithms and statistical methods. Comparing to the architecture designed in this paper, the distributed and parallel framework of Spark was utilized to akin to the role of SAMOA. Even with a handful choice of distributed data stream processing platforms, the principles of converting existing algorithms into parallel and distributed environment still remain under-addressed. This is addressed in this thesis as a systematic parallel workflow are designed and optimized for parallel processing along each stage of a workflow by scrutinizing the data level operations and their effects on system performance and scalability.

Commercial players such as IBM tackled this analytics business need by offering their own platform named Physical Analytics Integrated Repository and Services (PAIRS) [31] as a service. This platform is built on top of open source software and enables business-friendly manipulation, aggregation, and analytics on various datasets with space and/or time dimensions. SAS also has a dedicated module with a full suite of time series analytics capabilities with libraries of methods to process, apply statistics, clustering and many more [32]. These platforms are that they are designed for Businesses and are not available to the opensource community.

2.6.6 Time Series Anomaly Detection

Anomaly detection is the predictive analytics capability of finding unexpected patterns in a dataset with the end goal of taking action or gaining insight for better outcomes. Data mining on time series can be translated to application such as ecosystem modeling, network traffic monitoring, medical diagnosis, and other domains by detecting events such as heart arrythmia in electrocardiogram, intrusions in network data, congestion in traffic data, and ecosystem disturbances in Earth science data [74].

In general, time series anomolys can be characterized by high or low values or unusual subsequences and patterns. Characterizing normal and abnormal behaviour is a challenge in anomaly detection which often requires industry or domain experts to provide or guide towards that information.

DBSCAN has demonstrates several advantages over statistical approach on discovering anomolies on time series dataset. According to his paper, DBSCAN can identify anomolies even if there are no extreme values.

The EGADS Framework at Yahoo is a large scale anomoly detection which automatically monitors and alert on millions of time-series on different use cases [76]. This framework is leverages Hadoop and Storm for real time stream processing. Their approach at detecting anomalous time-series leverages multiple time series features such as spectral entropy, autocorrelation euclidean distance and involves clustering times series into a set of clusters. Their cluster analysis involves intra or inter-cluster time-series anomaly detection by measuring the deviation within or among the cluster centroids and the time-series. This paper is provides an high level overview of their general use case framework, but does not provide much information on the use of algorithms that are discussed in this thesis and the system level approach and methods used towards scalability [76].

Chapter 3

Industry Background

This section will provide an overview of the partnertship with Ciena Corporation, a United States-based global supplier of telecommunications networking equipment, software, and services, and provide context on processing platform and industry datasets made available for our academic research. This section will start by providing the partnership motive, provide details the big data platform which stands up the experimental environment, and finally, describe the industry provided datasets made available by our partner and their clients for our research.

3.1 Partnership motive

Capitalizing on big data analytics provides many opportunities for innovation and value creation for customers. Industry and Academia partnership commonly drives benefits for both parties through synergies which increases the rate at which innovation turns to customer value.

Academia is where a lot of the theoretical and experimental work takes place. In the analytics space, the challenges for academia have been to access real world usage data that can be used to train models and to access platform in which AI and ML algorithms can be deployed and tested at scale [70].

Industry players can greatly benefit from academic partnership as academic's scientific method used in research are needed in the application of machine learning to real industry problems. Implementing machine learning is not like implementing new software functionality. For most functionality you can specify requirements and come up with a development schedule. Instead, machine learning requires lots of experimentation, which looks more a like an academic science experiment. In this way, the academic culture of establishing a hypothesis followed by iterative experiments is what is is needed to successfully deploy machine learning in industry [70].

3.2 Big Data platform with parallel analysis of telecommunication Data

In the wake of the industrial revolution 4.0, disruption from technologies such as big data, cloud, IoT and 5G presents an urgent need to provide and support product and services capable to drive customer value in this new era of digital businesses. Current telecommunication service providers are facing many challenges in their key business processes which involves IT and networking functions. Common and ongoing challenges are the overwhelming and ever increasing demand from their clients imposing an stressfull workload on their network, siloed it operations, and the orchestration and management of complex multi-vender fragmented network to handle the ever increasing needs of today's connected world. Such challenges reinforce the need for Intelligent automation which will provide the capabilities to address the needs of tomorrow.

Our partner, being at the forefront of innovation, invested time and resource into their big data and analytics platform which provides deep insights in order to optimize network and IT performance through intelligent software-based network automation. Their solution achieves intelligent automation by leveraging analytics, multi-domain orchestration, federated inventory management, and route optimization and assurance. This open, data-driven approach optimally aligns and accelerates mission-critical business processes that span the network and IT to deliver faster time to revenue, reduced costs, and ultimately a better customer experience [71].

Fundamentally, all these capabilities are enabled by a strong awareness of the network state and the ability to analyze performance data from a multi-vendor network. Through this partnership, a development platform similar to their big data analytisc plaform was made available for the research in this thesis where algorithm experimentation and deployment can take place and analyzed.

3.3 Access to real world datasets

As a telecommunication equipment supplier and service provider, our partner has access to a wide and vast variety of communication network data generated from their own operations or client operations.

These datasets can be sourced and generated from many devices on a communication network and can capture information from many angles such as the network components, the communication channels, and other entities such as business context, end users, and more Network components, whether they are physical or virtual, can generate data about performance monitoring (PM), alarms, and logging data such as power levels, error counters, received, transmitted, or dropped packets, CPU utilization, and many more [73].

Communication channels can also generate performance monitoring data for all layers of the open systems interconnection model. For example, the performance of the optical layer is characterized by optical signal-to-noise ratio, chromatic dispersion, polarization-mode dispersion, transmit and received power, and bit error rate. The performance of the IP layer is characterized by bandwidth, throughput, latency, jitter, and error rate [73].

Whenever the data described above is collected, it is important to record a timestamp for every reading. This added time dimension enables opportunities for time analysis such correlating independent data sources and providing trends through time-series. In general, communication networks are connecting many different devices types from different venders producing data in different formats which can be collected at different frequencies for variable and prolonged periods of time. Communication Networks sourced can be described as “Big Data” for its high variety, velocity and volume of data.

Throughout this partnership, a few datasets were made available for our academic research. Due to non-disclosure agreements, security reasons, partner priorities and time restrictions, the focus of this paper will be on a selected dataset described in Chapter 4. This dataset is a sample of a real world communication network data provided by our partner’s client. This dataset size provided is only 350 mb. Although this dataset is on the smaller side, proving value and scalability on such dataset builds confidence when scaling to larger datasets.

Due to the nature of the partnership, our partners influenced many aspects of this thesis and assumptions and injected industry knowledge will be stated throughout this thesis whenever the academic research presented in this paper was influenced as such.

Chapter 4

The Problem Statement

This chapter starts with presenting the dataset that influenced the scope of this thesis. The problem statement is presented to outline key challenges encountered through the study and analysis of such dataset. Then, preprocessing steps and derived techniques to standardize the data suitable to enable further analysis are presented. Finally, the chapter presents an overview of the spark workflow and solution derived to address the problem statement.

4.1 The Dataset

As part of a predictive analytics proposal for a third party, our partner was provisioned a trial dataset to demonstrate their methodologies and capabilities. This trial dataset originated from the syslogs of an Alcatel-Lucent router and was identified as a important part of their business processes.

Syslogs, a standard for message logging, holds event logging data from network devices event which about where, when, and why the log was sent. Limited knowledge on this data set was provided as the intention was for Ciena to demonstrate their methodologies and their Big data platform capabilities. Subsequently, as their academic partners, our research attention was focused on this dataset and only this dataset will be covered in this thesis.

The dataset under study consists of 358 columns and 305078 rows occupying approximately 350 MB. Each columns represents a specific type of alarm and each row number represents how many times a the alarm was raised based on the syslog within a certain time period. The frequency at which each row was collected was assumed to be at consistent interval of time. Hence, this dataset is a collection of columns-based time-series of alarm counter values representing network performance data over an unknown period of time coming from a third party's network.

4.2 Problem Statement

As part of a proposal, our partner intents to use this dataset as input to machine learning models for predictive analytic. It is good practice to limit the number of input used by machine learning classifiers to what is truly necessary, and not blindly use all features in order to increase robustness against noise or bad-quality data, to avoid scalability bottlenecks with bigger datasets, and to accelerate iterations of the model-building process.

Analyzing these features is both data and computing intensive. The high dimensionality of this dataset impose the technical challenges on exploring the answers to questions such as

Q1. Do different time series datasets correlate to each other?

Q2. Which features can be discarded to help reduce the computational burden while keeping the analysis accuracy desired?

Q3. Do the clustering methods discover recurrent patterns in the telecommunication device based time series data?

These questions can be resolved through a pairwise analysis of features. Suppose there are n number of features and the sample size is m, the computation size of the dataset can be estimated at the level of $n^2 \times m$. When the computation of analysis exceeds the capacity of a single computing node, it is essential to scale out the analysis to a distributed cluster.

With the paradigm of distributed computing, the dataset is partitioned at two levels. At the first level, dataset partitions m samples to p partitions, thus each node has $m \div p$ time series. At this level, the partition is evenly distributed in terms of the data size.

At the second level, operations on pair wised features such as merging or joining further produce new sets of intermediate data. The new dataset is further aggregated by keys and shuffled across the partitions. When the partitions are distributed on separate nodes, the shuffling cause network traffic.

From the above analysis, the problem is further scoped into two issues that this thesis is focused on:

1. Data locality. According to an algorithm, the data with correlations are ensured to locate in the same partition or with close partitions. Hence, the analysis tasks at runtime are moved to the location where necessary data is located to complete the computation.

2. Data Repartition. At the intermediate stage of the analysis, newly produced data may cause an imbalance of data partitions and thus leads to data skew. Hence, repartition at runtime is performed. In addition, the repartition needs to be tuned to have optimal system performance, such as tuning the frequencies of repartition.

4.3 The Method Overview

The industry partner effort was focused on deriving predictive analytics application for their client whereas the academic research presented in this thesis focused on the work derived to explore, understand and prepare the data feeding into the industry partner's machine learning models. Throughout this process driven by the research questions, the thesis explores the relation between algorithm characteristics and parallelisms as well as the effects on clustering results and the system performance.

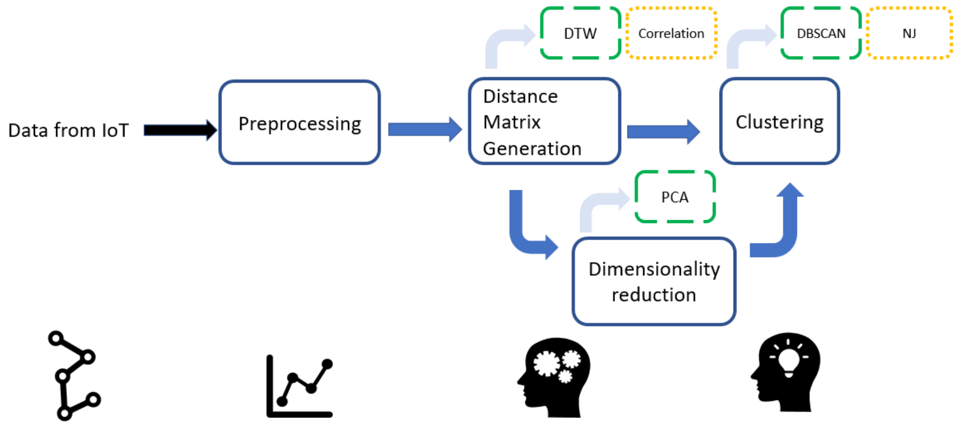


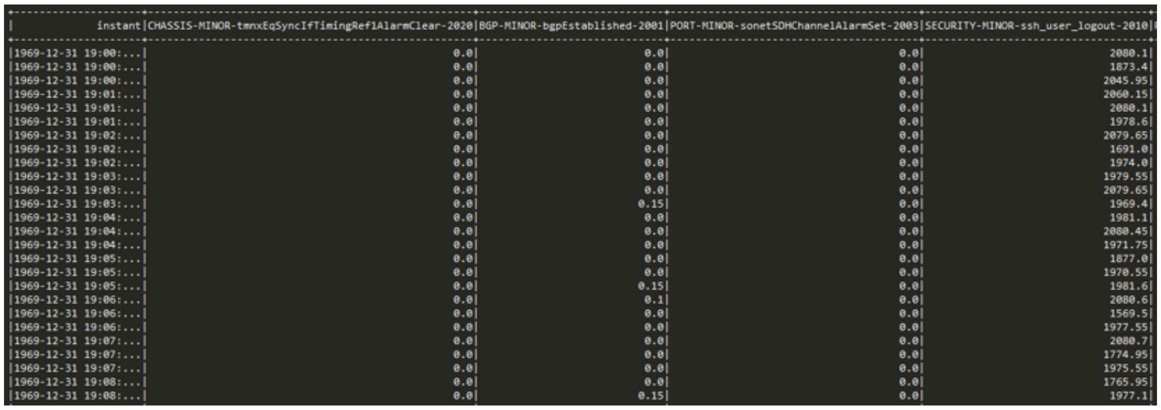
Figure 8: Overview of the multi-stage workflow, from data pre-processing, distance metrics generation, dimentionality reduction, to clustering

Although, the work was initiated based on a specific dataset, the contributions of this thesis can be generalized to similar time series processing problems.

The distributed and parallel machine learning method derived and driven by the research questions will be presented in the following chapters structured as the key four stages of the method namely (1) Preprocessing, (2) Distance Matrix Generation, (3) Dimensionality Reduction, and (4) Clustering.

Preprocessing handles issues such that devices have different sampling intervals, and the samples in time series have missing data. **Distance Matrix Generation** estimates the correlation among features. Two distance matrix is computed. Each is used as input for one clustering algorithm. **Dimensionality Reduction** solves the issue of the high dimensionality of features and prepares the time series for clustering analysis. Finally, **Clustering** analysis applies two algorithms, Neighbor-Joining (NJ) and Density-based spatial clustering of applications with noise (DBSCAN), to validate the results. Since different matrix and algorithms are applied, the method is realized by two parallel workflows. Figure 8 presents a high-level overview of components and sequences two workflows: Workflow 1) *Pre-processing*; *Distance Matrix Generation* using Correlation and *Clustering* using Neighbor-Joining; and Workflow 2) *Pre-processing*; *Distance Matrix Generation* using Dynamic Time Warping (DTW); *Dimensionality Reduction* using Principal Component Analysis (PCA); *Clustering* using DBSCAN.

Industry provided Recommendation: The distance-based method Neighbor-Joining was proposed by our industry partner as a suitable algorithm to find patterns in sequences of data such as time series. To complement our Industry recommendations and challenge their perspective, the second workflow using spatial density based clustering method DBSCAN was explored to see if it would find complementary information.



```
+--------------------+---------------------------------------------------+-----------------------------+---------------------------------------+-----------------------------------+
|             instant|CHASSIS-MINOR-tmnxEqSyncIfTimingRef1AlarmClear-2020|BGP-MINOR-bgpEstablished-2001|PORT-MINOR-sonetSDHChannelAlarmSet-2003|SECURITY-MINOR-ssh_user_logout-2010|
+--------------------+---------------------------------------------------+-----------------------------+---------------------------------------+-----------------------------------+
|1969-12-31 19:00:...|                                                0.0|                          0.0|                                    0.0|                             2080.1|
|1969-12-31 19:00:...|                                                0.0|                          0.0|                                    0.0|                             1873.4|
|1969-12-31 19:00:...|                                                0.0|                          0.0|                                    0.0|                            2045.95|
|1969-12-31 19:01:...|                                                0.0|                          0.0|                                    0.0|                            2060.15|
|1969-12-31 19:01:...|                                                0.0|                          0.0|                                    0.0|                             2080.1|
|1969-12-31 19:01:...|                                                0.0|                          0.0|                                    0.0|                             1978.6|
|1969-12-31 19:02:...|                                                0.0|                          0.0|                                    0.0|                            2079.65|
|1969-12-31 19:02:...|                                                0.0|                          0.0|                                    0.0|                             1691.0|
|1969-12-31 19:02:...|                                                0.0|                          0.0|                                    0.0|                             1974.0|
|1969-12-31 19:03:...|                                                0.0|                          0.0|                                    0.0|                            1979.55|
|1969-12-31 19:03:...|                                                0.0|                          0.0|                                    0.0|                            2079.65|
|1969-12-31 19:03:...|                                                0.0|                         0.15|                                    0.0|                             1969.4|
|1969-12-31 19:04:...|                                                0.0|                          0.0|                                    0.0|                             1981.1|
|1969-12-31 19:04:...|                                                0.0|                          0.0|                                    0.0|                            2080.45|
|1969-12-31 19:04:...|                                                0.0|                          0.0|                                    0.0|                            1971.75|
|1969-12-31 19:05:...|                                                0.0|                          0.0|                                    0.0|                             1877.0|
|1969-12-31 19:05:...|                                                0.0|                          0.0|                                    0.0|                            1970.55|
|1969-12-31 19:05:...|                                                0.0|                         0.15|                                    0.0|                             1981.6|
|1969-12-31 19:06:...|                                                0.0|                          0.1|                                    0.0|                             2080.6|
|1969-12-31 19:06:...|                                                0.0|                          0.0|                                    0.0|                             1569.5|
|1969-12-31 19:06:...|                                                0.0|                          0.0|                                    0.0|                            1977.55|
|1969-12-31 19:07:...|                                                0.0|                          0.0|                                    0.0|                             2080.7|
|1969-12-31 19:07:...|                                                0.0|                          0.0|                                    0.0|                            1774.95|
|1969-12-31 19:07:...|                                                0.0|                          0.0|                                    0.0|                            1975.55|
|1969-12-31 19:08:...|                                                0.0|                          0.0|                                    0.0|                            1765.95|
|1969-12-31 19:08:...|                                                0.0|                         0.15|                                    0.0|                             1977.1|
```

Figure 9: Sample of the input data stored in a Spark RDD

4.4 Data Preprocessing

The initial limited industry expert provided knowledge of the dataset reveals that:

1. Data comes from a single router device
2. All rows represent a set of 358 features extracted from the Syslogs at an unknown timestamp
3. Data is highly sparse with over 40% empty data.
4. Data sequences is sampled at successive equally spaced point in time, but no timestamps are provided

Figure 9 provides a sample of the input file stored in a Spark Time Series RDD after it has gone through a standardization process of replacing empty fields with zeros.

4.4.1 Data Preprocessing Techniques

A sample of the plotting of the data set is depicted in Figure 10. The plot shows an example of 3 features from 3 devices that consist of 9 time series. The sampling interval is at every 5 seconds and is observed from 0 to 60 seconds. Data coming from the same feature have a similar curve irrespective of the device they are from. Different features also show different behaviors. The three bottom time series show constant low values with little fluctuation. The middle three series have a spike starting at the 25 seconds that last till the 45 seconds before they set back to normal. The upper three time series are of higher values with constant fluctuation.

Figure 10 serves only as a high-level visual representation tool and is representative conceptually to describe the structure of the dataset available. In this case, the dataset headers suggest that each row represents a set of concurrent Alarm counts, and each column represents a specific type of alarm, presumably coming from the IoT network device. In the context of this dataset, an alarm represents a state of a property describing the health of the device. Hence, empty cells are assumed

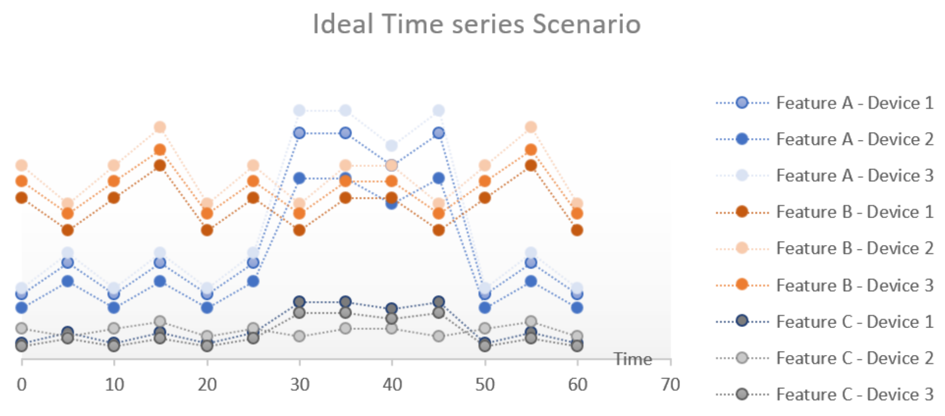


Figure 10: Plots of 9 dummy time series of 3 features on 3 devices used solely in this section for visual representation of the preprocessing techniques

to mean that no alarm was raised, and was filled with a zero. This figure will be used as a reference in the subsequent chapter on data pre-processing and will provide visuals that were built on top of this one.

Data preprocessing helps to assess the correlation between features. The purpose is to identify which of over 358 features are redundant that should be further merged for dimensional reduction. In fact, the gathered time series have different sampling intervals. Also, these time series have missing data. The preprocessing techniques below are highlighting the transformations steps applied on the raw dataset prior to applying the algorithms downstream in the next stage of the workflows. This ensures that the data prior to analysis is formatting adequatly for further analysis.

Throughout the research effort, multiple data processing techniques were explored and derived. Below are some of the key data preprocessing techniques explored throughout this thesis.

1. Column-based Raw time series data converted into distributed Vector of values. Each sequence of values were converted into a Vector data type in spark in order to maintain preserve the information provided by the order of those values. When loaded into an RDD, ll values of a time series are distributed as a single vector and are not broken down in parts across the cluster.

2. Filling sparse data with a replacement value a 0. This technique, also called imputation, is a common approach at handling sparse data. For univariate timeseries, there are various widely used techniques and algorithms to replace empty values. The more common ones are replace by value, interpolation, mean, moving average, kalman smoothing to name a few which all comes with their own sets of benefits and trade offs. **Industry provided Recommendation:** This processing technique was applied based on recommendation from a industry domain expert stating that empty values corresponds to no alarms raised in the dataset which can be safely replaced by 0.

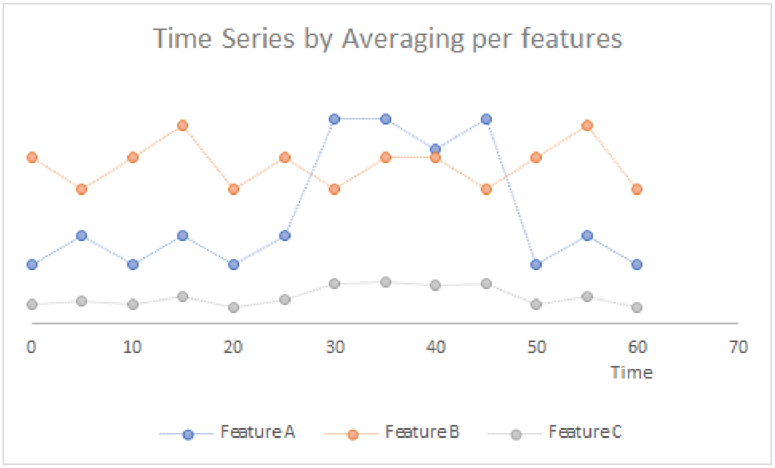


Figure 11: The original 9 time series in Figure 10 are aligned into 3 time series, one for each feature that is generated by averaging the readings per time stamps from 3 devices.

3. Data binning by averaging the values at every time stamp. At each time stamp, there are multiple data records for each feature from different devices. This approach first performs alignment of all-time series of the same feature by time stamps. If a timestamp has no readings from any devices, the missing value is filled as zero. Then the average value is taken for each feature at each timestamp. A sample plot of the dataset in Figure 10 after the alignment is shown in Figure 11. This technique is also usefull when dealing with sparse data because as the values are aggregated, there are less empty values per data points. Although this technique can reduce the dimensionality of the input, there is a drawback of information loss the more the binning window is large.

   The window size of data readings was a tunable parameter throughout this thesis as it was a driving performance factor for the workflows. Tuning was required to find the right trade-off balance between the processing time and information retention. Figure 13 and Figure 14 show the time series after the binning technique for the window size of 10 seconds and 20 seconds respectively. **Industry provided Recommendations:** This processing technique was derived based on recommendation from a industry domain expert stating that using the average aggregation function is sufficient for their data exploration use cases. For this reason, no alternate aggregate functions and their trade-offs where explored in this thesis as further exploration with variants with aggregate function would have required more time.

4. Concatenating the time series from different devices sequentially. This approach first creates a list of all devices. Then time series is created for each feature by concatenating the time series of the devices by respecting the order of that list. This is achieved by creating an artificial and longer timeline of readings. Again, empty readings are filled by zero. The order in which the time series is appended must stay consistent across different features. Figure 12 plots the concatenated time series of Figure 10. The window size is a tunable parameter as it becomes a factor for a trade-off between the processing time and information retention. Figure 13 and

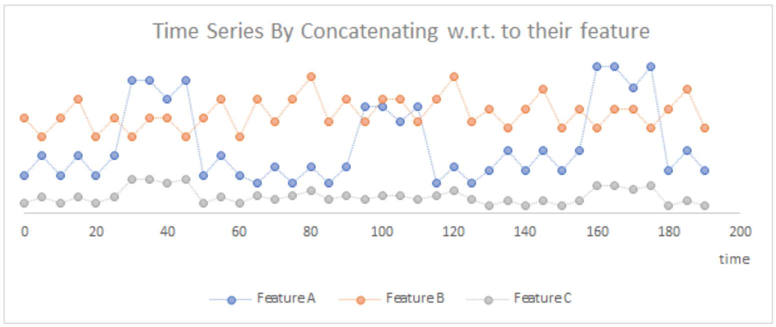


Figure 12: The original 9 time series in Figure 10 are aligned into 3 time series. The 3 devices of the same feature are appended one after the other. Device one has the time serieis from the data points ranging between 0 to 60 seconds; device 2's values range between 65 to 125 seconds; and devices 3' values are from 130 to 190 seconds respectively.

Figure 14 show the time series after the binning technique for the window size of 10 seconds and 20 seconds respectively.

Limitation: This technique was explored for datasets that are not presented in this thesis . The need for this technique steemed from the existence of sub-categories of time series in alternate datasets which presented the need to compare the main categories of time series but also, sub-categories of time series in parallel. This approach would concatenate the related sub-categories of a timeseries into single main time series.

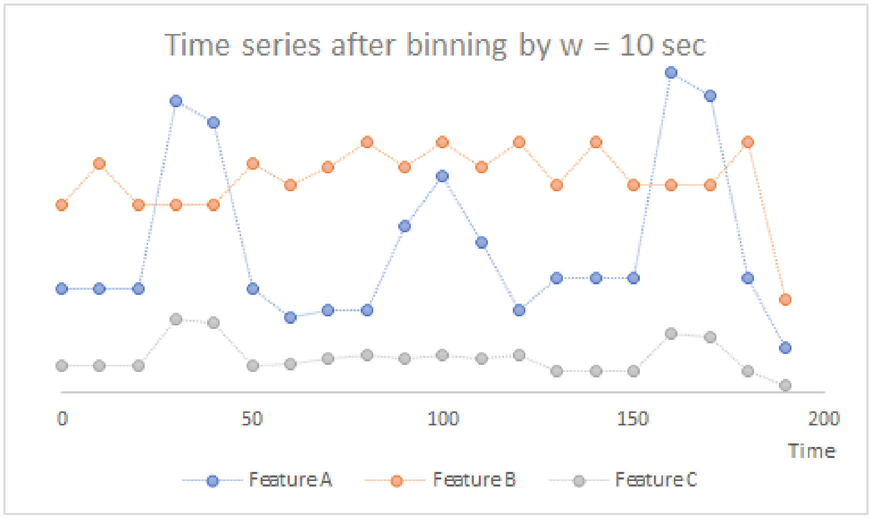


Figure 13: Times series from Figure 12 after applying binning by average on a window of 10 seconds. Technique applied for a different dataset and use case not covered in this thesis

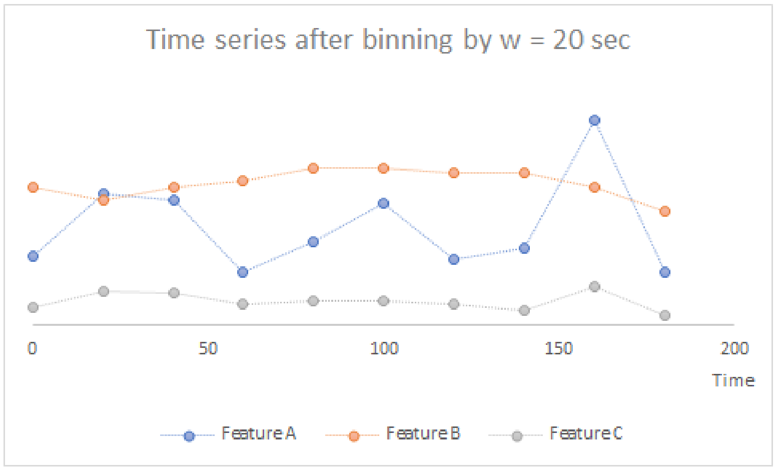


Figure 14: Times series from Figure 12 after applying binning by average on a window of 20 seconds.

Chapter 5

The Correlation and Neighbor-Joining Clustering Method

This Chapter presents a clustering method integrating statistical models such as Pearson's Correlation the Neighbor-Joining Hierarchical Clustering Algorithm into a parallel workflow. This method provides a tool which can be used to address the problem statements questions **Q1**, **Q2**, and **Q3**.

The Chapter starts by providing an overview of the algorithms integrated into this method. The algorithm definition and traditional non-distributed applications are introduced to prepare for the subsequent sections which will focus on their adaptation and integration to the parallel workflow. One of the section will highlight the implementation of PCA using the Spark Framework. The chapter continues with the Parallel Neighbor-Joining which was designed in this thesis and highlights key challenges from adapting a matrix-based statistical algorithm into a distributed framework such as Spark. The chapter ends with the presenting output of the Correlation and Neighbor-Joining Clustering Method.

5.1 Algorithms Overview

This section provides an overview of the algorithms discussed and part of the Correlation and Neighbor-Joining Clustering Method by providing their standard definition.

5.1.1 Correlation

In statistics, Correlation can be broadly defined as the degree of association between two variables and the direction of their relationship. The result of the correlation given by equation 1 is called the correlation coefficient and will be in the range of -1 to $+1$. The direction of the relationship is defined by the sign of the correlation coefficient. The most widely used correlation statistic is Pearson's correlation which measures the degree of relationship between linearly related variables [66].

$$\rho = CORR(X,Y) = \frac{\sum[(X-\mu_X)(Y-\mu_Y)]}{\sigma X \sigma Y} \tag{1}$$

where X and Y are two random variables; μ_X and μ_Y are their respective mean, and σX and σY are their respective standard deviation

In practice, correlations provide a statistical measure to compare the relationship between two variables which can be exploited for predictive analysis.

5.1.2 Neighbor-Joining

Neighbor-Joining is a recursive clustering algorithm where each iteration finds the closest two nodes, joins them by replacing with a new node, and finally calculates the distance of this new node to all other existing nodes. The algorithm contains four stages.

The first stage of the algorithm computes the sum of rows of the input distance matrix, called R_i computation. The variable R_i holds the sum of every row of the input distance matrix, dm.

The second stage of an iteration is finding the minimum Q_h calculation given by [10]. Minimum Q_h is calculated by application equation (1) on a vectorized version of the distance matrix [26]. This vectorized version of the distance matric DV_h formats the distance matrix into a vector to simplify calculation without affecting performance.

$$Q_h = (N-2)DV(h) - (R(i) + R(j)) \tag{2}$$

where $i = 1...N-1; j = i+1...N; h = ((i-1)(2n-i))/(2+j-i)$.

Equation (1) performs transformations on the distance matrix dm to derive the matrix Q_h. The smallest value of Q_h, denoted as $minQ_h$, identifies the two closest nodes to merge.

The third stage involves calculating the new distance values that result from merging the nodes identified by the minimum Q_h. It calculates the distance of a node k to the newly merged node U_1 as the given formula of Equation (2). The new values are then inserted at their respective distance matrix indexes for the next iteration.

$$D_{U_1k} = \frac{D_{ik} + D_{jk}}{2} \tag{3}$$

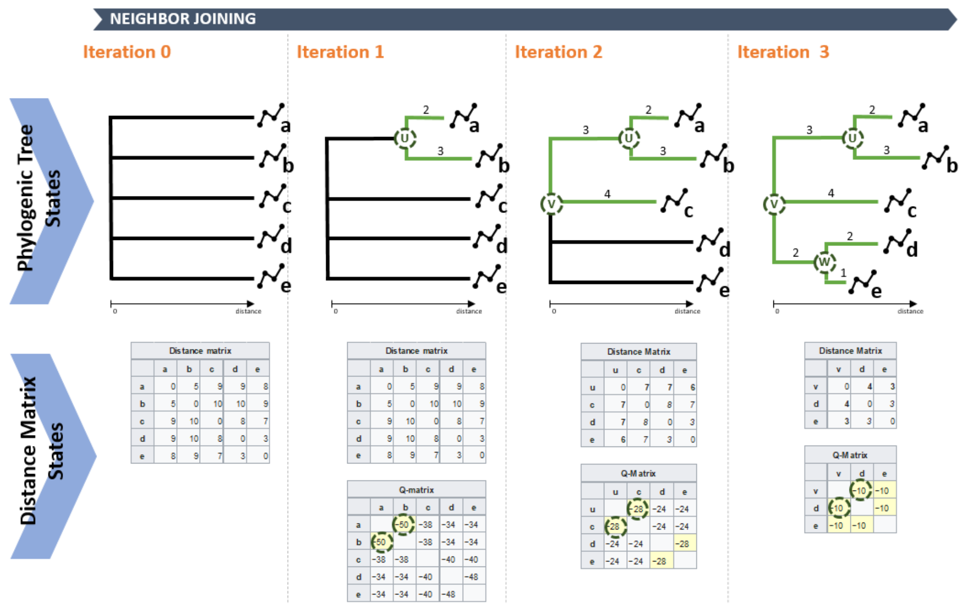
where $k = 1...N$; $k \neq i, j$ and

$$D_{iz} = \frac{\sum\limits_{k=1}^{N} D_{ik}}{N-2}$$

and

$$D_{jz} = \frac{\sum\limits_{k=1}^{N} D_{jk}}{N-2}$$

The formula to calculate the branch lengths L_{iU_1} and L_{jU_1} against this newly joint node are given by equations



Iteration 0

Distance matrix	a	b	c	d	e
a	0	5	9	9	8
b	5	0	10	10	9
c	9	10	0	8	7
d	9	10	8	0	3
e	8	9	7	3	0

Iteration 1

Distance matrix	a	b	c	d	e
a	0	5	9	9	8
b	5	0	10	10	9
c	9	10	0	8	7
d	9	10	8	0	3
e	8	9	7	3	0

Q-matrix	a	b	c	d	e
a		-50	-38	-34	-34
b	-50		-38	-34	-34
c	-38	-38		-40	-40
d	-34	-34	-40		-48
e	-34	-34	-40	-48	

Iteration 2

Distance Matrix	u	c	d	e
u	0	7	7	6
c	7	0	8	7
d	7	8	0	3
e	6	7	3	0

Q-Matrix	u	c	d	e
u		-28	-24	-24
c	-28		-24	-24
d	-24	-24		-28
e	-24	-24	-28	

Iteration 3

Distance Matrix	v	d	e
v	0	4	3
d	4	0	3
e	3	3	0

Q-Matrix	v	d	e
v		-10	-10
d	-10		-10
e	-10	-10	

Figure 15: Neighbor-Joining example on an arbitrary distance matrix. In iteration 1, the phylogenic tree initializes all the nodes with equal distances. The Q matrix is computed from the initial distance matrix, where time series a and b are identified as the closest nodes. These nodes are joined as node u. Their branch length is calculated and the distance matrix is updated. In iteration 2, time series c and u are joined as node v. Their branch length is calculated and the distance matrix is updated. In iteration 3, branch node v and time series d are joined as node w and their branch length is calculated. The distance between w and e is finally calculated.

$$
\begin{aligned}
L_{iU_1} &= \frac{D_{ij} + D_{iz} - D_{jz}}{2} \\
L_{jU_1} &= \frac{D_{ij} + D_{jz} - D_{iz}}{2}
\end{aligned}
\tag{4}
$$

The final stage updates the distance matrix for the next iteration.

Figure 15 illustrates a simple example of the neighbor-joining process. The result is a phylogenic tree that visually shows the relationship between time series. Similar to a binary tree, the leaves of this phylogenic tree represents a time series while nodes are connected to others through branches proportional to their level of similarity measured as the distance metrics applied.

5.2 Parallel Processing of Feature Correlation

The Neighbor-Joining algorithm requires a distance measure for every pair-wise combination of time series. These distance measures are stored in a distance matrix that serves as a similarity measure between any two time series.

The distance matrix is of the size of $N \times N$, where N is the number of time series in the dataset. The correlation coefficient of a pair-wise combination of the time series is used as the distance

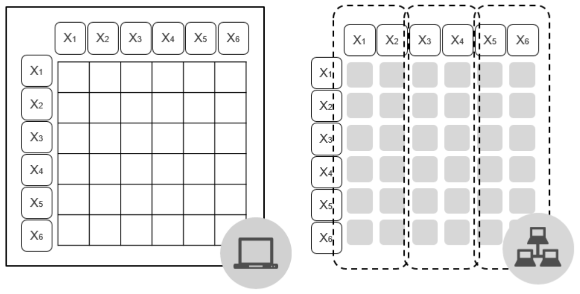


Figure 16: Left: Standard matrix data structure on a single core. Right: the matrix abstraction on a distributed environment where RDDs that represent cells of the matrix are spread out amongst three nodes.

measure. The correlation coefficient between two random variables X and Y is defined as

$$\rho(X,Y) = \frac{\mathbf{Cov}(X,Y)}{\sqrt{\mathbf{Var}(X)\mathbf{Var}(Y)}}$$

The naive approach of generating a distance matrix is to iterate through each cell of the matrix and apply the above formula to corresponding cells.

Cells of a distance matrix are represented using Resilient Distributed Datasets (RDDs) from Apache Spark [9]. There is no native matrix in the Spark. However, operations crucial to the Neighbor-Joining algorithm are not fully supported at the time being of this project. Therefore, RDDs of the type [(Long, Long), Double] are defined as the main distributed matrix abstraction in this workflow and use it as the input to the Neighbor-Joining algorithm.

Each cell is represented by an RDD of type [(Long, Long), Double], where both Long variables correspond to indexes of row i and column j. The Double variable is a placeholder for the distance measure. Such an RDD is a distributed variable.

RDDs are first created by reading the dataset from a stable storage such as HDFS into partitioned collection of records. These RDDs are further transformed by operations such as map, filter, groupBy, reduce.

The parallelization breaks down a matrix into units of elements. Figure 16 illustrates the partition of the initial matrix into a distributed matrix.

This distance matrix populated with the correlation coefficient of the pairwise combination of time series provides the ability to benchmark their level of similarity. By doing so, the problem statement **Q1** and also **Q2** are addressed by providing a tool which can be used, if desired, for preliminary feature reduction by discarding time series that are highly correlated to reduce unnecessary computation effort downstream.

In listing 5.1, the source code written in scala depicts how the correlation matrix is first computed for the Upper Diagonal. From there, the already calculated values are assigned to the opposite index of this distance matrix which is achieved by setting the values of a tuple (i,j) to the tuple (j,i). Listing 5.2, provides the source code for the correlation matrix calculation on the pairwise sets of time series which was achieved by integrating the Spark MLlib statistic library method.

```
val (dmResult,labels) = DistanceMatrix.upperCorrelationMatrix(preprocessedData)

dmResult.collect().foreach{
                     case((i,j),v) =>
                     denseMatrix(i.toInt,j.toInt) = v;
                     denseMatrix(j.toInt,i.toInt) = v;
```

Listing 5.1: Calculating Correlation Matrix in Spark

```
def upperCorrelationMatrix(input:RDD[(String,Vector)]):(RDD[((Long,Long),
Double)],RDD[(Long,String)]) =
    {
        val indexed = input.zipWithIndex().cache()
        val vectors = indexed.map{case(s,v)=> (s._2)}.cache()
        val labels = indexed.map{case(s,v)=> (v,s._1)}

        val cnt = indexed.count.toInt
        val correlMatrix: Matrix = Statistics.corr(transpose(vectors), "pearson")

        //isolate upper matrix
        val upperDmRDD = sqlContext.sparkContext.parallelize(
            for { i <- 0 until cnt
                  j <- 0 until cnt
                  if( i < j)
            } yield ((i.toLong, j.toLong), 1 - Math.abs(correlMatrix.apply(i,j)))
        )

        //Substitite NaN
        val result = upperDmRDD.mapValues{ case(v) => if ( v.isNaN ) 1.0 else (1.0 - v)}

        (result,labels)

    }
```

Listing 5.2: Correlation Matrix implementation in Spark Code Snippet

5.3 Parallel Neighbor-Joining Algorithm

The parallelization of Neighbor-Joining is limited to the level of each iteration because each iteration depends on its previous one. In every iteration of the algorithm, matrix Qh is calculated using equation (2) on the input distance matrix used to identify the closest two nodes to join.

The parallelization of this algorithm comes down to parallelizing the calculation of Q_h. This is achieved by mapping each cell of the initial distance matrix to its new value of Q_h. The mapping operation is convenient in this process because the new Q_h matrix is of the same size as the distance matrix at the beginning of each iteration. The parallelization version of the algorithm is presented in Algorithm 1.

Algorithm 1 Parallel Neighbor-Joining Algorithm

```
Require:
    The number of time series $N$;
    The distance matrix $dm$;
Ensure:
    The updated distance matrix $dm$;
 1: Broadcast $dm$
 2: for $h = 1$ to $N - 2$ do do
 3:     Compute $R_i$ in parrallel.
 4:     Broadcast $R_i$;
 5:     for all $i$ , $j$ do
 6:         Compute $Q_h$ using broadcasted $R_i$ in parallel using Spark's map transformation. Equation
            (2) is applied to all cells of the distance matrix $dm$;
 7:     end for
 8:     Compute the minimum $Q_h$ to get neighbors i and j. The Spark filter transformation is applied
        to obtain the minumum $Q_h$;
 9:     Nodes i and j are joined as a new Node $U_h$;
10:     Compute updated distance $D_{uhk}$ according to Equation (3);
11:     Compute branch lengths $L_{iU_h}$ and $L_{jU_h}$ according to Equation (4);
12:     Delete nodes i and j, and add $U_h$ to current node lists;
13:     Update the distance matrix $dm$;
14: end for
15: Join the last two nodes N-2 and N-1
16: return $dm$
```

Figure 17 depicts the Neighbor-Joining workflow implemented in Spark. This flow summarizes the key spark transformations of each iteration of the algorithm. The workflow can be broken down into 4 logical stages where the first stage computes Ri a variable used throughout subsequent calculations, the second stage implements the NJ algorithm which identifies the nodes to join, the third stage prepares the distance matrix with the joined node, while stage 4 prepares other variables for the next iteration.

5.3.1 Parallel Neighbor-Joining implementation in Spark

This section provides an overview of the spark implementation of the parallel neighbor-joining algorithm. The following listing highlights substeps of the pseudo-code presented in Algorithm 1.

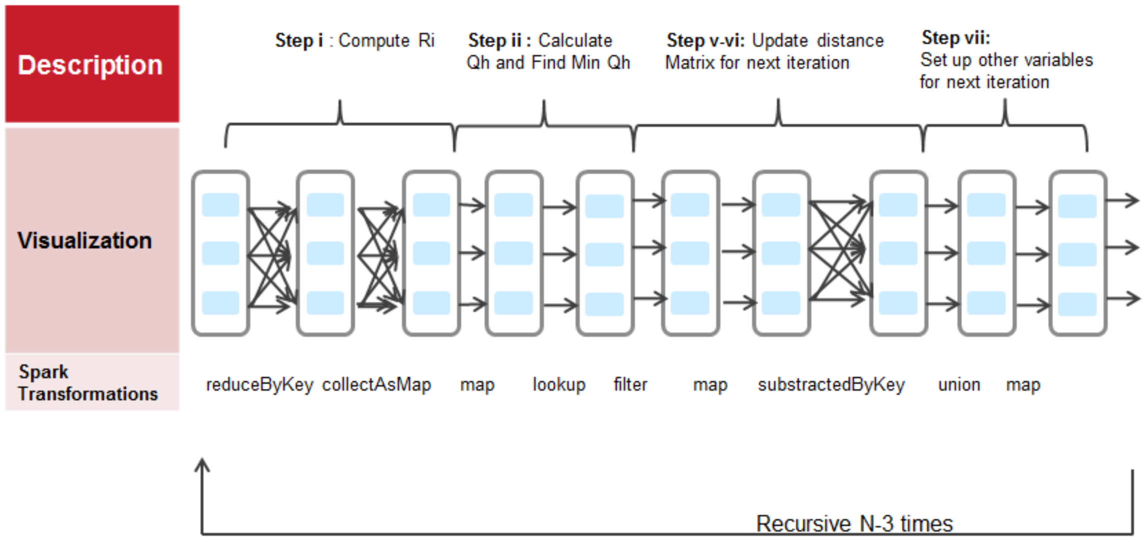


Figure 17: The neighbor-joining parallel workflow with 4 main stages through every iteration

Algorithm 1 captured the high level flow of the parallel neighbor joining algorithm derived for this thesis but was not directly implemented in the same exact pattern. The following Listings will serve as a medium to highlight key spark implementation steps tackling the challenge of parallelizing a statistical algorithm.

Initiation of the Parallel Neighbor-Joining is presented in Listing 5.3 which highlights the Scala source code written in Spark. This method picks off from the result of the correlation matrix created in the previous section.

```
    val (dmResult, labels) = DistanceMatrix.upperCorrelationMatrix(filtereddata)
    NeighborJoiningParallel.init(dmResult, labels)
    result = NeighborJoiningParallel.convertTreeInNewick()
```

Listing 5.3: Neighbor-Joining initialization In Spark Snippet

The body of the NJ algorithm coordinating and sequencing each iteration as defined in Line 2 and 13-16 in Algorithm 1 is highlighted in the Listing 5.4. The recursive nature of the implemented algorithm can be observed as the *performNeighborJoining* method is iteratively called on the same distance matrix variable. It can be observed that *localcheckpoint* is used to cache the distance matrix variable before each iteration for performance reasons discussed in the next section. *Coalesce* is also used to address data partitioning and data shuffling impacting performance of the workflow. More details on this tuning technique are provided in Section 6.4.4.

```
object NeighborJoiningParallel {
    ...
    //Initialize the tree to store the results of NJ and start NJ
    def init(dm: RDD[((Long, Long), Double)],labels:RDD[(Long,String)]) = {

        //setup variables
        initTree(labels)
        InitialTaxaCount = labels.count()
        var currentDm = dm
        currentDm.localCheckpoint()
        TaxaOrder = IndexedRDD(labels)
        initialNumPartitions = currentDm.getNumPartitions
        var currentOptimalNumPartition = initialNumPartitions

        for( h <- InitialTaxaCount to 3 by -1){
            N = h

            currentDm = performNeighborJoining(currentDm)
            currentDm.localCheckpoint()

            //coalesce required to repartition data due to unwanted partition creation from
                union
            if(h % 1 == 0 ) {
                currentDm = currentDm.coalesce(initialNumPartitions)
            }
        }

        val remainingDist = currentDm.first()._2
        TREE(1).setBranchLength(remainingDist)
    }
...

}
```

Listing 5.4: Neighbor-Joining initialization implementation in Spark snippet

Line 3-13 of Algorithm 1 embodies the definition of Neighbor-Joining algorithm. It can be observed in Listing 5.5 that the spark *broadcast* variable was leveraged to calculate the summation variables of Equation (2) defined in the method *findMinQh*. The summation variable required to be calculated prior to perform spark mapping operations because a summation operation performed on a single data partition will not be equivalent to the summation operation performed on the whole data set. This is sample hurdle of parallelizing a statistical algorithm, where the design must ensure that the signifiance of the algorithm is not loss from performing operations on partitionned data.

```
    //Execute a set of methods that consists of an iteration of the NJ algorithm
```

```
def performNeighborJoining(dm: RDD[((Long, Long), Double)]): RDD[((Long, Long),
     Double)] = {
    //Calculate Qh and return the smallest Qh as minQh
    val minQh = findMinQh(dm)
    //Calculate Duhk
    val updatedDM = computeDuhk(dm,minQh._1._1,minQh._1._2,minQh._2)

    //Update tree and DM to store result of this iteration
    updateTaxaOrder(minQh)
    updateTree(minQh)

    //Return updated Distance Matrix for the next iteration
    updatedDM
}

// Calculates Qh inspired by equation (2) and return the smallest value of Qh
def findMinQh(dm:RDD[((Long,Long),Double)]):
((Long,Long),Double) = {
    // Generate a map variable from Ri for quick access during Qh calculation in the
        next below
    val riMap = computeRi(dm).cache().collectAsMap()

    // Broadcast Ri to worker nodes
    val Ri = sqlContext.sparkContext.broadcast(riMap)

    //finding Ri using broadcast
    val Qh = dm.map{ case(k,v)=>

    //Use broadcasted Ri
    val ri = Ri.value.getOrElse(k._1,0.0)
    val rj = Ri.value.getOrElse (k._2,0.0)

    val Q = ((N-2) * v) - (ri - rj)
    (k,Q)}.localCheckpoint()

    val minQh = Qh.min()(new Ordering[Tuple2[(Long,Long), Double]]() {
    override def compare(x: ((Long,Long), Double), y: ((Long,Long), Double)): Int =
    Ordering[Double].compare(x._2, y._2)})

    val minDist = dm.lookup(minQh._1)

    //I and J of smallest Qh with value of their corresponding Dij;
    return (minQh._1,minDist(0))
}
```

```
//Calculate once the sum of rows
def computeRi(dm: RDD[((Long,Long),Double)]):RDD[(Long,Double)] = {

    val summedRowDm = dm.map{case((i,j),v)=> (i,v)}
                        .reduceByKey( _ + _)

    summedRowDm
}
```

Listing 5.5: Neighbor-Joining source code in Spark capturing Line 3-13 of Algorithm 1

Another challenge of parallelizing a statistical algorithm was to performing matrix operations on a distributed data structure. The Neighbor-Joining algorithm requires reducing by one column and one row conceptually every iteration as the result of merging two closest nodes, as shown in Figure 18.

Since the distance matrix is an RDD of indexes-value pairs, deleting a row and a column becomes the operation on query RDDs with keys corresponding to a row and column indexes. Adding the merged node is done by creating a new RDD with the merged value of distance. Consequently, the remaining RDDs' key is updated to reflect the changing indexes of rows and columns in adjacent sequential order with no gap.

Listing 5.6 highlights *Line 9-13* of Algorithm 1 capturing necessary matrix operations for the Neigbor-Joining algorithm. Matrix dimension reduction technique is present in the *computeDuhk* method. In this method, the spark variable *updatedIndexDm* is the final product of removing any occurrence of the *i* and *j* to create variable *subtractedDm*, by merging the new *Duk* values in *unionDm*, and by re-indexing in the rdd distance matrix abstraction for the next iteration. These matrix operations were made possible from the distance matrix abstraction presented in this thesis.

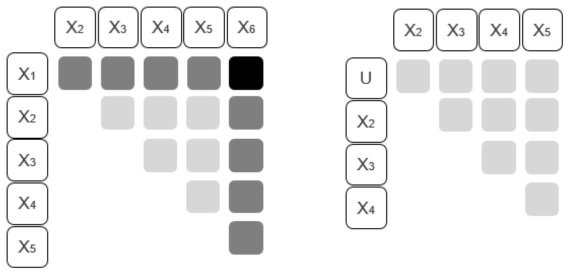


Figure 18: Scenario of distance matrix update in one iteration. Left: The black cell corresponds to the smallest value of Q_h obtained. Hence the closest nodes X_6 and X_1 need to be merged. Right: The corresponding row and column that contain nodes X_6 and X_1 are deleted from the distance matrix. A new cell U is placed at the smallest index position, row 1 in this case. The new distance between node U and remaining nodes are computed.

```
//Calculate Duhk as defined by Equation (2)
def computeDuhk(dm:RDD[((Long,Long),Double)],i:Long,j:Long, Dij:
    Double):RDD[((Long,Long),Double)]= {
    val I = Math.min(i,j) //I is assumed to be the smaller index
    val J = Math.max(i,j)

    val dmWithI = dm.filter( t => t._1._1 == I || t._1._2 == I).localCheckpoint()
    val Dik = dmWithI.map{ t =>
                  val i = if (t._1._1 == I) t._1._1 else t._1._2
                  val k = if (t._1._1 != I) t._1._1 else t._1._2
                  (k,(i,t._2))
               }.localCheckpoint()

    val dmWithJ = dm.filter( t => t._1._1 == J || t._1._2 == J).localCheckpoint()
    val Djk = dmWithJ.map{ t =>
                  val j = if (t._1._1 == J) t._1._1 else t._1._2
                  val k = if (t._1._1 != J) t._1._1 else t._1._2
                  (k,(j,t._2))
               }.localCheckpoint()

    //Return an RDD containing all pairs of elements with matching keys
    val Duk = Dik.join(Djk,initialNumPartitions)
               .map{t =>

                  val u = t._2._1._1 //u will be smaller index
                  val k = t._1
                  val newDist = (t._2._1._2 + t._2._2._2) /2
```

```
                        ((u,k),newDist)}

       //Remove trace of old I and J values
       val subtractedDm = dm.subtractByKey(dmWithI)
                         .subtractByKey(dmWithJ)

       //Add newly calculated Duk
       val unionDm = subtractedDm.union(Duk)

       // Update index values of the Dm due to the matrix size reduction from the node
           merge
       val updatedIndexDm = unionDm.map{ case ((i,j),v) =>
                                 val updatedI = if( i > J) i -1 else i
                                 val updatedJ = if( j > J) j -1 else j
                                 ((updatedI,updatedJ),v)}

       //Calculate Branch length
       calculateBranchLength(Dij,Dik,Djk)

       return updatedIndexDm
    }

 calculation steps in Spark },captionpos=b]
    //Calculate new node branch length following Equation (3)
    def calculateBranchLength(Dij:Double ,dik:RDD[(Long,(Long,Double))],
        djk:RDD[(Long,(Long,Double))]) = {
       val DizSum = dik.map{case(k,(i,v))=> (v)}
                           .sum()
       val Diz = DizSum/ (N-2)

       val DjzSum= djk.map{case(k,(j,v))=> (v)}
                        .sum()
       val Djz = DjzSum / (N-2)

       val Liu = (Dij + Diz - Djz) / 2
       val Lju = (Dij + Djz - Diz) / 2
       BranchLiu = Math.abs(Liu)
       BranchLju = Math.abs(Lju)
    }

}
```

Listing 5.6: Neighbor-Joining source code in Spark capturing Line 9-13 of Algorithm 1

5.4 Performance Tuning

Throughout the development of the Parallel Neighbor-Joining algorithm in Spark, two performance tuning opportunities were identified.

The first performance tuning was addressing a limitation from the Spark distributed framework when provided a recursive function. Internally, due to its in-memory processing framework, Spark has to re-calculate every previous iteration before addressing the current iteration. This translates into a very long and expansive RDD lineage. Although the neighbor-joining function complexity gradually reduces over its iterations, the naive implementation of this recursive function displayed significantly increasing execution time for each iteration. In order, to reduce the complexity of this operation, the localcheckpoint method was introduced at key stages of each iteration in order to truncate the lineage. The checkpoint performs a memory cache of the variable and making the RDD at the point of caching to "remember" its contents the first time it passes by there. As depicted Listing 4.1, the localcheckpoint method was applied variable Q. This translates into storing in memory the result of Q so that subsequent iterations can leverage off this computed value and not recompute it.

The second performance tuning opportunity was to leverage the broadcast spark method to reduce redundant computing efforts of a common variable required by parallel tasks. As depicted Listing 5.7, Ri is computed first on the driver then broadcasted to executors for subsequent parallel tasks similar to a global variable in traditional programming. This variable is used to calculate ri and rj when calculating Qh through a map operation. Through this map operation, each task can directly read and access the indexed value through the broadcasted variable Ri instead of each tasks having to recalculate Ri making this computing step more scalable and efficient.

To compute R_i, the cells of the distance matrix were grouped by their row index i and summed using the reduceByKey transformation. A collectAsMap transformation is applied to R_i at the beginning of the next stage in order to obtain a map variable with the row i as key and the sum of the row as values. Once broadcasted, each task can access the relevant sum of row value using the row index as a key.

```
def findMinQh(dm:RDD[((Long,Long),Double)]):
    ((Long,Long),Double) = {

    // Generate a map variable from Ri for quick access during Qh calculation in the next
        below
    val riMap = computeRi(dm).cache().collectAsMap()

    // Broadcast Ri to worker nodes
    val Ri = sqlContext.sparkContext.broadcast(riMap)

    //finding Ri using broadcast
    val Qh = dm.map{ case(k,v)=>
    //Use broadcasted Ri
    val ri = Ri.value.getOrElse(k._1,0.0)
    val rj = Ri.value.getOrElse (k._2,0.0)

    val Q = ((N-2) * v) - (ri - rj)
    (k,Q)}.localCheckpoint()

    val minQh = Qh.min()(new Ordering[Tuple2[(Long,Long), Double]]() {
    override def compare(x: ((Long,Long), Double), y: ((Long,Long), Double)): Int =
    Ordering[Double].compare(x._2, y._2)})

    val minDist = dm.lookup(minQh._1)

    //I and J of smallest Qh with value of their corresponding Dij;
    return (minQh._1,minDist(0))
}
```

Listing 5.7: Broadcast and localcheckpoint in Neigbor-Joining in Spark

5.5 The Workflow Output

The output of the workflow follows the Newick tree, a format to represent a graph-theoretical trees with edge lengths. This output is further converted to a cladogram based on industry recommendation which is a diagram that shows the relation between entities based on the branch length. The shorter the branch length, the more related the two entities are. Figure 19 plots the clustering result. In this plot, small regions with multiple short branch lengths correspond to time series that are closely related. It can be observed that time series with common labels are grouped into clusters.

Figure 20 and Figure 21 provides enlarged section of Figure 19. It can be observed that the enlarged sections of Figure 19 that neighboring features from the same cluster tend to have similar label names, such as starting with SVCMGR or BGP, which are unknown prefixes found in the feature names in the dataset. Without injecting telecom domain knowledge, the workflow manages to group these features solely from their statistical correlations.

The output of this Newick tree generated from this workflow provides the ability to visualize a cladogram of time series where the proximity and the length of the branches outline the strength of their relationship and similarity. In this thesis, the derived Neighbor-Joining workflow and its output provide a tool which can address the problem statements questions **Q1**, **Q2**, and **Q3**. Through this particular use case using this dataset and workflow, the resulting cladogram provided an early pattern of similarity based on common prefixes **Q3**. Such a pattern can help lessen the technical challenge on exploring high dimensional datasets by correlating time series **Q1** and discarding **Q2** highly similar time series to reduce computational burden down the road.

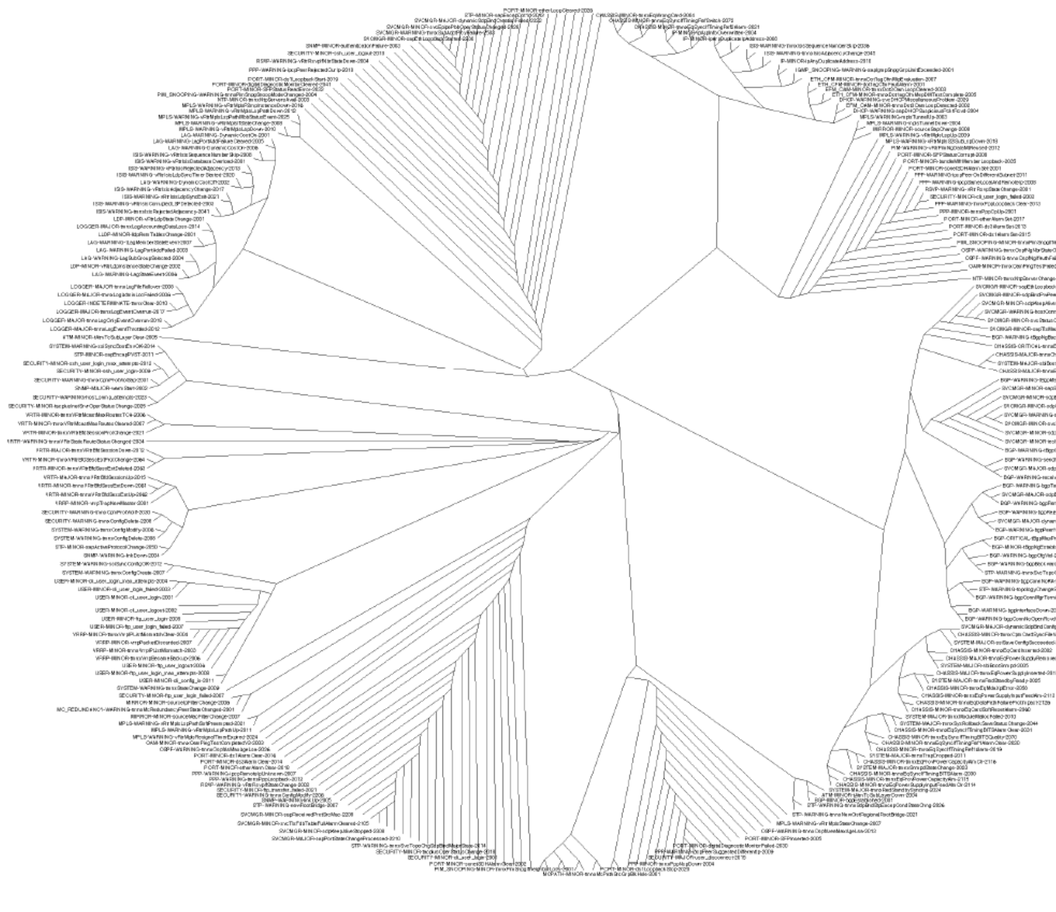


Figure 19: Visualization of Neighbour-Joining clustering of devices.

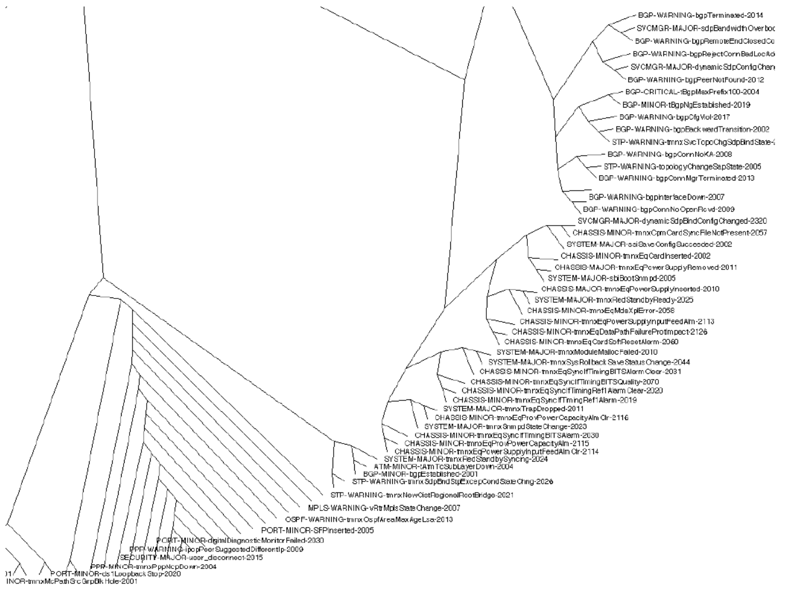

BGP-WARNING-bgpTerminated-2014
SVCMGR-MAJOR-sdpBandwidthOverboo
BGP-WARNING-bgpRemoteEndClosedCo
BGP-WARNING-bgpRejectConnBadLocAd
SVCMGR-MAJOR-dynamicSdpConfigChang
BGP-WARNING-bgpPeerNotFound-2012
BGP-CRITICAL-tBgpMaxPrefix100-2004
BGP-MINOR-tBgpNgEstablished-2019
BGP-WARNING-bgpCfgViol-2017
BGP-WARNING-bgpBackwardTransition-2002
STP-WARNING-tmnxSvcTopoChgSdpBindState-
BGP-WARNING-bgpConnNoKA-2008
STP-WARNING-topologyChangeSapState-2005
BGP-WARNING-bgpConnMgrTerminated-2013
BGP-WARNING-bgpInterfaceDown-2007
BGP-WARNING-bgpConnNoOpenRcvd-2009
SVCMGR-MAJOR-dynamicSdpBindConfigChanged-2320
CHASSIS-MINOR-tmnxCpmCardSyncFileNotPresent-2057
SYSTEM-MAJOR-ssiSaveConfigSucceeded-2002
CHASSIS-MINOR-tmnxEqCardInserted-2002
CHASSIS-MAJOR-tmnxEqPowerSupplyRemoved-2011
SYSTEM-MAJOR-sbiBootSnmpd-2005
CHASSIS-MAJOR-tmnxEqPowerSupplyInserted-2010
SYSTEM-MAJOR-tmnxRedStandbyReady-2025
CHASSIS-MINOR-tmnxEqMdaXplError-2058
CHASSIS-MINOR-tmnxEqPowerSupplyInputFeedAlm-2113
CHASSIS-MINOR-tmnxEqDataPathFailureProtImpact-2126
CHASSIS-MINOR-tmnxEqCardSoftResetAlarm-2060
SYSTEM-MAJOR-tmnxModuleMallocFailed-2010
SYSTEM-MAJOR-tmnxSysRollbackSaveStatusChange-2044
CHASSIS-MINOR-tmnxEqSyncIfTimingBITSAlarmClear-2031
CHASSIS-MINOR-tmnxEqSyncIfTimingBITSQuality-2070
CHASSIS-MINOR-tmnxEqSyncIfTimingRef1AlarmClear-2020
CHASSIS-MINOR-tmnxEqSyncIfTimingRef1Alarm-2019
SYSTEM-MAJOR-tmnxTrapDropped-2011
CHASSIS-MINOR-tmnxEqProvPowerCapacityAlmClr-2116
SYSTEM-MAJOR-tmnxSnmpdStateChange-2023
CHASSIS-MINOR-tmnxEqSyncIfTimingBITSAlarm-2030
CHASSIS-MINOR-tmnxEqProvPowerCapacityAlm-2115
CHASSIS-MINOR-tmnxEqPowerSupplyInputFeedAlmClr-2114
SYSTEM-MAJOR-tmnxRedStandbySyncing-2024
ATM-MINOR-tAtmTcSubLayerDown-2004
BGP-MINOR-bgpEstablished-2001
STP-WARNING-tmnxSdpBndStpExcepCondStateChng-2026
STP-WARNING-tmnxNewCistRegionalRootBridge-2021
MPLS-WARNING-vRtrMplsStateChange-2007
OSPF-WARNING-tmnxOspfAreaMaxAgeLsa-2013
PORT-MINOR-SFPInserted-2005
PORT-MINOR-digitalDiagnosticMonitorFailed-2030
PPP-WARNING-ipcpPeerSuggestedDifferentIp-2009
SECURITY-MAJOR-user_disconnect-2015
PPP-MINOR-tmnxPppNcpDown-2004
PORT-MINOR-ds1LoopbackStop-2020
INOR-tmnxMcPathSrcGrpBlkHole-2001

Figure 20: Visualization of Neighbour-Joining clustering of devices. Zoom in on the bottom right of the whole cladogram to display that neighbour features from the same cluster tend to have similar names

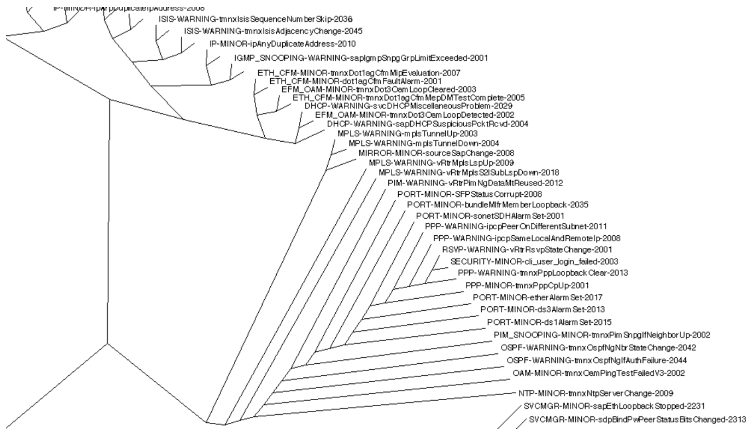

ISIS-WARNING-tmnxIsisSequenceNumberSkip-2036
ISIS-WARNING-tmnxIsisAdjacencyChange-2045
IP-MINOR-ipAnyDuplicateAddress-2010
IGMP_SNOOPING-WARNING-sapIgmpSnpgGrpLimitExceeded-2001
ETH_CFM-MINOR-tmnxDot1agCfmMipEvaluation-2007
ETH_CFM-MINOR-dot1agCfmFaultAlarm-2001
EFM_OAM-MINOR-tmnxDot3OamLoopCleared-2003
ETH_CFM-MINOR-tmnxDot1agCfmMepDMTestComplete-2005
DHCP-WARNING-svcDHCPMiscellaneousProblem-2029
EFM_OAM-MINOR-tmnxDot3OamLoopDetected-2002
DHCP-WARNING-sapDHCPSuspiciousPcktRcvd-2004
MPLS-WARNING-mplsTunnelUp-2003
MPLS-WARNING-mplsTunnelDown-2004
MIRROR-MINOR-sourceSapChange-2008
MPLS-WARNING-vRtrMplsLspUp-2009
MPLS-WARNING-vRtrMplsS2lSubLspDown-2018
PIM-WARNING-vRtrPimNgDataMtReused-2012
PORT-MINOR-SFPStatusCorrupt-2008
PORT-MINOR-bundleMlfrMemberLoopback-2035
PORT-MINOR-sonetSDHAlarmSet-2001
PPP-WARNING-ipcpPeerOnDifferentSubnet-2011
PPP-WARNING-ipcpSameLocalAndRemoteIp-2008
RSVP-WARNING-vRtrRsvpStateChange-2001
SECURITY-MINOR-cli_user_login_failed-2003
PPP-WARNING-tmnxPppLoopbackClear-2013
PPP-MINOR-tmnxPppCpUp-2001
PORT-MINOR-etherAlarmSet-2017
PORT-MINOR-ds3AlarmSet-2013
PORT-MINOR-ds1AlarmSet-2015
PIM_SNOOPING-MINOR-tmnxPimSnpgIfNeighborUp-2002
OSPF-WARNING-tmnxOspfNgNbrStateChange-2042
OSPF-WARNING-tmnxOspfNgIfAuthFailure-2044
OAM-MINOR-tmnxOamPingTestFailedV3-2002
NTP-MINOR-tmnxNtpServerChange-2009
SVCMGR-MINOR-sapEthLoopbackStopped-2231
SVCMGR-MINOR-sdpBindPwPeerStatusBitsChanged-2313

Figure 21: Visualization of Neighbour-Joining clustering of devices. Zoom in on the top right of the whole cladogram to display that neighbour features from the same cluster tend to have similar names

5.5.1 Cluster Analysis

In this section, analysis of the correlation neighbor-joining parallel workflow output is presented in order to confirm the design and implementation validity. If valid, the parallel workflow output should contain clusters where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other. Although initial observations of the clusters are provided, data mining effort to search and generate information from this dataset was not in scope in this thesis.

A dendrogram is a tree-like display that lists the objects which are clustered along the x-axis, and the distance at which the cluster was formed along the y-axis. A cladogram can be abstracted to a circular dendogram. In order to identify clusters in a dendrogram, a line must be drawn across the branches of the diagram in order to delimit the clusters. Following the same methodology, given the circular shape of the cladogram, a circle delimiter was placed in the middle of the cladogram as depicted in Figure 22.

The placement of the circle was determine based on the current structure of the cladogram. It can be noticed that some aggregations of words are tighly related due to their short branch length - see clusters 1-3-6-7-8-9, whereas other clusters are aggregated with longer branch length - see clusters 2-4-5-10. In this analysis, the placement of the circle was determined in order to delimit those distinct 10 groups of objects.

Without industry knowledge on the dataset, it is challenging to assess how the objects within clusters are similar. What can be achieved is through their labeling. To provided statistical significance to the observation above regarding common prefixes within clusters, each objects within each clusters were compared based on their labeling.

In order to generate the numbers in Figure 23, the size of the clusters based on the unique labels were compared to the count of unique prefixes. In this dataset, a prefix was the string of text before the second occurance of a dash "-" character . In example, for the object labeled "port-minor-SonetsdhdAlarmSet-2001", the prefix is "port-minor". In this case, when a cluster has a low count of distinct prefix, it implies that many of the objects have similar prefix. In this analysis, a lower percentage of distinct prefix (highlighted in green) indicates that the objects within the cluster have similar prefixes meaning they are similar based on the labeling pattern. Furthermore, it can be noticed that tightly aggregated clusters due to their short branch length clusters (1,3,6,7,8,9) have on average a higher rate of similar labels from their prefixes than the clusters (2,4,5,10) with the longer branch length.**In summary, in Figure23, the lower the % of distinct prefix, the higher the similarity for time series within the respective cluster.** This analysis confirms that the clustering is valid because it was able to cluster objects with more similarity together based solely the values of the time series and without injection of telecommunication knowledge in the workflow.

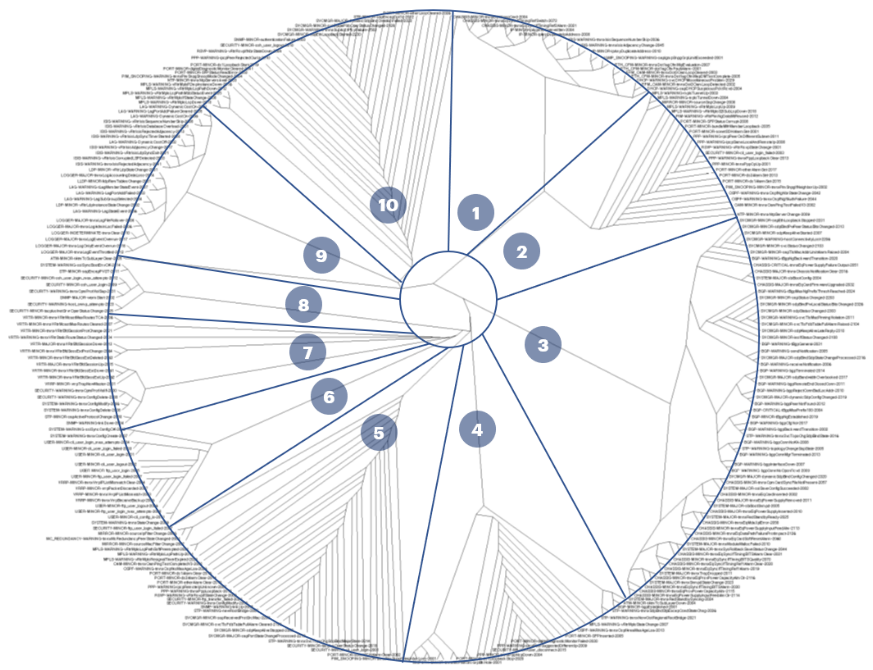
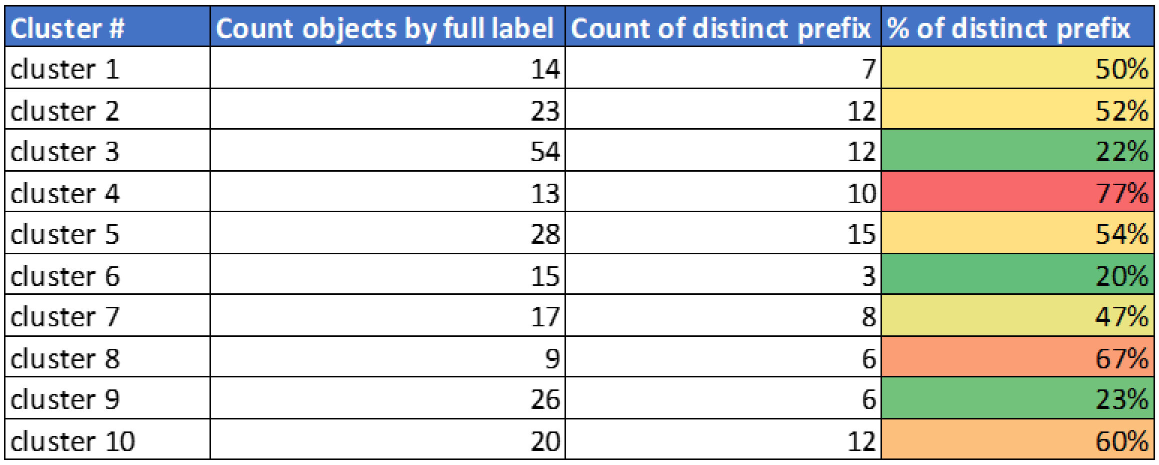


Figure 22: Cluster delimitation on Figure 20



Cluster #	Count objects by full label	Count of distinct prefix	% of distinct prefix
cluster 1	14	7	50%
cluster 2	23	12	52%
cluster 3	54	12	22%
cluster 4	13	10	77%
cluster 5	28	15	54%
cluster 6	15	3	20%
cluster 7	17	8	47%
cluster 8	9	6	67%
cluster 9	26	6	23%
cluster 10	20	12	60%

Figure 23: Cluster analysis results based on labelling.

Chapter 6

The DBSCAN Clustering Method

This chapter presents the exploration of an alternate approach at analyzing the time series data through spatial distance in contrast to correlation distance in the previous method. The motivation in developing a alternate method is to explore if different clustering methods could find complimentary results as the Neighbor-Joining Method. As a result, a scalable and parallel workflow for time series analysis using density based clustering was developed with system performance evaluation. This workflow features a transformation step which applies DTW to evaluate he similarity between time series and projects the distance matrix into a spatial dimension using PCA. The DBSCAN algorithm selected in this workflow performs unsupervised learning in spatial dimension and excel with noisy data.

This Chapter further explores a clustering method which explores the relation between the pairwise time series by evaluating their association level and explores potential patterns amongst the dataset. This method results in a tool which can be used to address our problem statements questions **Q1**, **Q2**, and **Q3**. An overview of the workflow is presented in Figure 24.

The Chapter starts by providing an overview of the algorithms integrated into this method. The algorithm definition and traditional non-distributed applications are introduced to prepare for the subsequent sections which will focus on their adaptation and integration to the parallel workflow. Three stages of the parallel workflow are then presented in their respective section. The workflow first

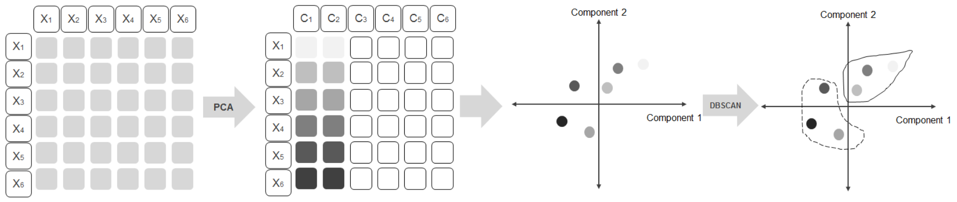


Figure 24: An overview of the DBSCAN clustering method. The leftmost represents the distance matrix of the input time series measured with DTW. PCA is then applied to reduce the dimensionality. These pairs of principal components from PCA are then projected into a two-dimensional space where the X axis is component 1 and the Y axis is component 2. Finally, DBSCAN is applied to these points to find clusters based on the principal components.

computes the distance of any two pairs of time series. This produces a distance matrix that follows the Dynamic Time Warping (DTW) coefficients of any two time series. This distance matrix is further transformed by the algorithm of Principal Component Analysis (PCA). PCA transforms the distance matrix into a set of linearly uncorrelated variables, called principal components. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm clusters the output from PCA. The chapter ends by presenting the output of the DBSCAN clustering method.

6.1 Algorithms Overview

This section provides an overview of the algorithms discussed and part of the DBSCAN Clustering method by providing their standard definition.

6.1.1 Dynamic Time Warping

DTW measures the similarity between sequences of any two time series that may vary in speed referred to as a lag (or delay). DTW corrects the lag by finding the minimal distance $D(i, j)$ between two time series X_i and Y_j. $D(i, j)$ is defined as a recursive function with the mapping starting from $(1, 1)$ and ending at (i, j).

$$D(i, j) = |X_i + Y_j| + \min\Big\{D(i-1, j); D(i-1, j-1); D(i, j-1)\Big\} \tag{5}$$

The computing complexity of DTW is of $O(n^2)$.

6.1.2 Principal Component Analysis

Principal Component Analysis is a mathematical procedure used orthogonal transformations that converts a number of variables assumed to be correlated into a smaller set of linearly uncorrelated variables named *Principal Components* [63].

Each possible principal component is defined in such a way that the preceding component has a higher variance leading to the first principal component having the largest possible variance.

By definition, PCA is obtained by 1) evaluating the covariance matrix X of the dataset and then 2) finding the eigenvectors of X through eigendecomposition.

$$C = Covariance(X) = XX^T/(n-1).$$

Since C is a symmetric matrix, it can be diagonalized such that

$$C = VIV^T$$

where V is a matrix of eigenvectors and I is a diagonal matrix of eigenvalues. The *principal components* of X are now given by XV. The i-th principal component is given by the i-th column of XV. The only parameter specified through PCA is the number of principal components k expected. If the initial matrix X is of dimension $n \times m$, the result of PCA on X is an n x k matrix, where each column corresponds to a principal component.

Patterns of data can be difficult to grasp in a higher dimension. To facilitate the analysis and pattern identification of the data, Principal Component Analysis can serve as a tool for dimensionality reduction on high dimension datasets in which the reduced dimensions retains most of the information from its source [64].

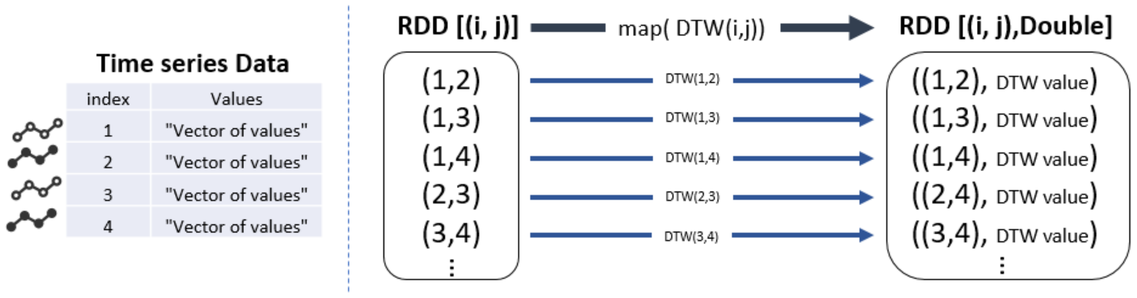


Figure 25: The DTW Distance Matrix Computation in Spark. A mapping transformation calculating the DTW algorithm is applied to an RDD of indexes. Each of these indexes corresponds to vectors of values which defines the time series.

6.1.3 Density-Based Spatial Clustering of Applications with Noise

Density-based spatial clustering of applications with noise (DBSCAN) clusters a set of spatial data points based on their proximity. This unsupervised machine learning method does not require prior knowledge of the count of clusters in the dataset and can find clusters of arbitrary shape [24]. Two parameters are expected for the DBSCAN algorithm, namely $minP_{ts}$ and eps. $minP_{ts}$ is the minimum number of objects required to form a dense region. eps is a measure of proximity that defines if a spatial point should be considered into a cluster.

DBSCAN clustering method distinguishes itself with the ability to find patterns and structures in the data that are not arbitrary shape in spatial databases with noise [65].

6.2 Parallel Processing of Dynamic Time Warping

In The DBSCAN Clustering Method, the first stage of the workflow consists of generating the Distance Matrix using the Dynamic Time Warping Algorithm. The parallel processing of the Dynamic Time Warping and details how it was integrated into the DBSCAN Clustering Method are presented in this section.

A naive approach of generating a distance matrix is to iterate through each cell of the matrix and apply Equation 5. Figure 25 illustrates the parallel processing to generate a distance matrix applying Equation 5. The distance matrix abstraction in this thesis is represented by an RDD of type [(Long, Long),Double], where both Long variables correspond to indexes of row i and column j. The Double variable is a placeholder for the distance measure that is the DTW measurement in this case. Indexes i and j refer to two specific time series. Hence, the distance matrix RDD is generated by means of a map transformation that applies Equation 5 to all instances of an RDD.

While implementing the DTW from equation 5 and applying it as a Spark Mapping Operation to all pairwise time series, it was quickly observed in the Spark UI, Spark's application interface, that this operation was taking too long to complete to the point where the workflow seemed to stall. This stage was a clear bottleneck of the workflow because it would take an extended amount of time to process this task and it would delay the overall execution time of the job. Due to the high

Algorithm 2 FastDTW

```
Require: Time series $X_i, X_j, (i \neq j)$ and $radius$ - distance to search outside of the projected warp
    path of the previous resolution when refining the warp path
Ensure: $dtwCoefficient$ between warp path of $X_i$ and $X_j$
 1: minTSsize = radius +2
 2: if (|$X_i$| ≤ minTSsize OR |$X_j$| ≤ minTSsize) then
 3:     //For small input time series, run traditional DTW
 4:     RETURN DTW($X_i, X_j$)
 5: else
 6:     $X_i = X_i$.reduceByHalf()
 7:     $X_j = X_j$.reduceByHalf()
 8:     lowResPath = FastDTW($X_i$,$X_j$, radius)
 9:     searchWindow = ExpandedResWindow(lowResPath,$X_i$, $X_j$,radius)
10:     RETURN DTW($X_i, X_j$,searchWindow)
11: end if
```

dimensionality of the dataset and the quadratic computing complexity of DTW is $O(n^2)$, it became apparent that this distance matrix generation step required revision.

The solution was to adopt an approximation of FastDTW [22] with computing complexity $O(n)$. The authors of the paper propose a linear version of the traditional quadratic complexity DTW algorithm that approximates accurately the optimal path between two times series while discarding unnecessary computation. This implementation of the distance measure is more suitable for larger time series. The algorithm of FastDTW is detailed in Algorithm 2. Integration of this optimized algorithm to this method was achieved by leveraging the open source Java implementation made available by the authors [67]. Listing 6.1 provides the details of this stage of the workflow using Spark. Once FastDTW was imported as a library, all pairwise combination of time series were applied a Spark mapping operation with FastDTW as the mapping function.

This optimization step significantly lessened the performance bottleneck previously experienced by making the distance matrix generation using DTW a linear complexity task instead of a quadratic complexity one.

By measuring the distance between all the time series using the distance measure FastDTW, the thesis addresses **Q1** by providing a measure of similarity on the input time series dataset and also **Q2** by providing initial benchmarking criteria for, if needed, discarding time series that are identified as highly similar through small FastDTW coefficients.

```
//FastDTW distance matrix generation using Open Source Java implementation
def fastDtwDistanceMatrixWithBroadcast(input:RDD[(String,Vector)]):
RDD[(Long,(Array[Double],String))] =
{
    //Broadcast TS RDD
    val tsMap = input.zipWithIndex().map{case(v,i)=>(i,v)}
                                   .collectAsMap()
```

```
  val broadcastedTsMap = sqlContext.sparkContext.broadcast(tsMap)

  //Prepare indices corresponding to Upper Diagonal cells
  val cnt = input.count.toInt

//Pruning the lower diagonal indexes
  val indices = sqlContext.sparkContext.parallelize(
     for { i <- 0 until cnt
          j <- 0 until cnt
          if( i < j)
     } yield (i.toLong, j.toLong)
  )

  val distFn = DistanceFunctionFactory.getDistFnByName("EuclideanDistance")
  val fastDtwRDD = indices.map{ case (i, j) => {

     val seq1 = new
         TimeSeries(broadcastedTsMap.value.get(i).get._2.toArray.toBuffer.asJava)
     val seq2 = new
         TimeSeries(broadcastedTsMap.value.get(j).get._2.toArray.toBuffer.asJava)
     val fastDtwCoefficient = FastDTW.getWarpDistBetween(seq1,seq2,distFn)

     ((i,j),fastDtwCoefficient)
     }
  }

  //Invert results for to obtain other triangle of the matrix
  val otherDtwTriangle = fastDtwRDD.map{ case ((i,j),v) => ((j,i),v) }

  // Adding the diagonal of the dtw matrix since a dtw of a Variable with itself is
      equal to 0
  val diagonal = sqlContext.sparkContext.parallelize(for { i <- 0 until cnt} yield
      ((i.toLong,i.toLong),0.0))

  // Combine all parts of the distance matrix
  val distanceMatrix = fastDtwRDD.union(otherDtwTriangle).union(diagonal)
  val initialSet = mutable.ArrayBuffer.empty[(Long,Double)]
  val addToSet = (s: mutable.ArrayBuffer[(Long,Double)], v: (Long,Double)) => s += v
  val mergePartitionSets = (p1: mutable.ArrayBuffer[(Long,Double)], p2:
      mutable.ArrayBuffer[(Long,Double)]) => p1 ++= p2

  def mapFunc(lines:Iterator[((Long,Long),Double)])={
     lines.map{case ((i,j),v) => (i,(j,v))}
  }
```

```
    //Replaced map by mapPartitions for optimization
    val formattedDm = distanceMatrix.mapPartitions(mapFunc)
                                 .aggregateByKey(initialSet)(addToSet,
                                     mergePartitionSets)
                                 .mapValues(_.toArray.sortBy(_._1).map(_._2))

    //Combine Labelled with FastDTW DM Results
    val formattedDmLabelled = formattedDm.map{ case (i, v) => (i, (v,
        broadcastedTsMap.value.get(i).get._1))}

    formattedDmLabelled
}
```

Listing 6.1: FastDTW in Spark Code Snippet

6.3 Principal Component Analysis in Parallel

In the DBSCAN Clustering Method, PCA is then applied to perform dimensionality reduction on the distance measure generated using a Dynamic Time Warping algorithm. PCA is used to projects the high dimensional distance matrix into a lower linear space to perform spatial clustering in a visualizable space. PCA performs an orthogonal projection of the distance matrix into a lower dimensional linear space with the maximized variance of the data.

The implementation of PCA is derived from the Spark MLLibs library [11] and its Spark Source code are detailed in Listing 6.2. Using Spark MLLibs, PCA is performed in two stages. The first stage generates the covariance matrix of X. The second stage is the eigenvalue decomposition achieved through Singular Value Decomposition (SVD). This method accepts a single parameter k that specify how many principal components the user seeks. The input is the distance matrix formatted in RDD[vector], whereby each vector contains the data points of one time series. Internally, the method converts the input RDD [Vector] into Spark RowMatrix, an abstraction of a row-oriented distributed matrix with properties defining the number of rows and columns. RowMatrix consists of a set of numerical algorithms on matrixes that applies breeze [23], a generic, and powerful numerical processing library. Finally, the principal components are stored as a local matrix of size $n \times k$, where each column corresponds for one principal component. The columns are in descending order of component variance. This matrix is used as input to the DBSCAN algorithm.

By integrating the PCA in the DBSCAN Clustering Method, this part of the workflow addresses **Q2** by staging the data for spatial clustering in a lower dimension found in the next stage of this method and **Q3** by reducing the computational burden of processing a high dimensional distance matrix while maintaining the information and potential patterns from the higher dimension.

Limitation: However, a limitation should be noted. The DBSCAN implementation in this thesis is restricted to 2-dimensional space clustering which requires the dimensionality reduction through PCA to only use the first two principal components. This constraint will have impacts on this workflow output as information in higher principal components are loss due to potentially risk over reducing the dimensionality.

```
def fit(sources: RDD[Vector]): PCAModel = {
    val numFeatures = sources.first().size
    require(k <= numFeatures, s "source vector size $numFeatures must be no less than
        k=$k")

    require(PCAUtil.memoryCost(k, numFeatures) < Int.MaxValue,
    "The param k and numFeatures is too large for SVD computation. " +
    "Try reducing the parameter k for PCA, or reduce the input feature " +
    "vector dimension to make this tractable.")

    val mat = new RowMatrix(sources)
    val (pc, explainedVariance) = mat.computePrincipalComponentsAndExplainedVariance(k)
    val densePC = pc match {
        case dm: DenseMatrix => dm
        case sm: SparseMatrix => sm.toDense /*Convert a Spark matrix to dense. */
        case m => throw new IllegalArgumentException("Unsupported matrix format.
            Expected " + s"SparseMatrix or DenseMatrix. Instead got: ${m.getClass}")
    }
    val denseExplainedVariance = explainedVariance match {
        case dv: DenseVector => dv
        case sv: SparseVector => sv.toDense
    }

    new PCAModel(k, densePC, denseExplainedVariance)

}
```

Listing 6.2: PCA Spark Implementation

6.4 The Parallel DBSCAN Algorithm

The last stage in the DBSCAN Clustering Method consists of applying the DBSCAN algorithm in order to identify potential patterns and/or structures in the dataset.

This thesis adopts an open source implementation of DBSCAN on Spark [8] for a two-dimensional space clustering. This dbscan implementation imposes a technical requirement for the input spatial points to be in two dimensions in this thesis. This is achieved by using PCA with a number of principal components $k = 2$ on the distance matrix. In contrast to the traditional DBSCAN algorithm, this parallel version introduces the third parameter $maxP_{ts}$ that limits the maximum

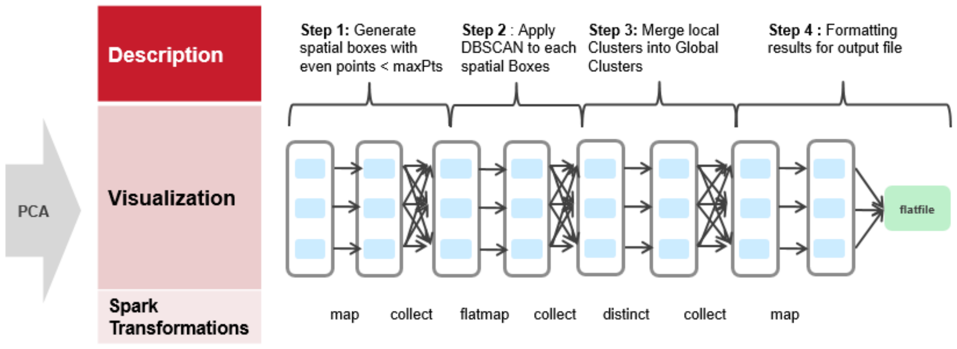

Description
Visualization
Spark Transformations
PCA
Step 1: Generate spatial boxes with even points < maxPts
Step 2 : Apply DBSCAN to each spatial Boxes
Step 3: Merge local Clusters into Global Clusters
Step 4 : Formatting results for output file
flatfile
map
collect
flatmap
collect
distinct
collect
map

Figure 26: Overview of DBSCANs workflow adapted for Spark. The distance matrix is the starting point of the workflow where each cell is the coefficient obtained from the distance function applied to X_i and X_j. Then the distance matrix is transformed by PCA. In this case, the first two principal components are maintained. These two components are used as spatial coordinates. Each pair of cells is represented as a point in the spatial plot. Eventually, DBSCAN is applied to identify clusters based on the principal components.

number of points. The implementation of the entire workflow using Spark is depicted in Figure 26. More details of this Parallel DBSCAN Implementation in Apache Spark is provided below.

The parallel implementation of DBSCAN follows the Map-Reduce principles by mapping (or breaking down) the global spatial area that englobes all input data spatial points into smaller areas with an even number of spatial points that can be processed in parallel. The results of each smaller areas are then reduced (or aggregated) to obtain the final DBSCAN result for the entire spatial area. The parallel DBSCAN algorithm is broken down into three main stages:

1. Data partitioning - consists of spatial points divided evenly into partitions based on their proximity;
2. Local clustering - application of a traditional naive DBSCAN to each partition containing the spatial points;
3. Global merging - merging of local clusters of each partition to generate global clusters.

Step 1 Data Partitionning as detailed in Listing 6.3 distributes evenly the input spatial points into smaller rectangular spaces based on spatial proximity. The input is a RDD[Vector], where each vector is of size two containing a horizontal x and vertical y coordinates of a spatial point. This RDD[Vector] goes through transformations based on spatial density in order to identify their initial partition that is associated with a rectangle subspace space in the global search area. As depicted in Figure 27, a rectangle is an object defined by dialog coordinates. These rectangles are partitioned input to local clustering where DBSCAN is applied to each partition in parallel.

The rectangle object is generated by mapping each spatial point in the RDD[Vector] to a rectangle space of the size $eps * eps$, where eps is the measure of proximity. AggregateByKey is then used to group and count all the points that fit within the same rectangular space of the size $eps * eps$. The resulting variable minimumRectangleWithCounts is a list of spatial rectangles and the count spatial points it contains and serves as the initial step to data partitioning.

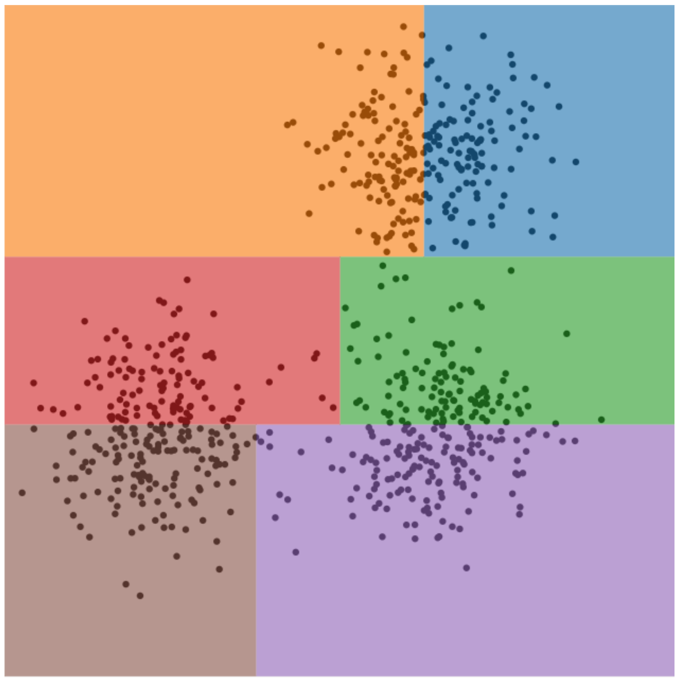


Figure 27: Representation of spatial partitioning in smaller rectangles of the global rectangle area.

The variable minimumRectangleWithCounts, is then partitioned using the custom partition method of EvenSplitPartitioner. This recursive method iterates through each rectangle to ensure a rectangle should contain spatial points less than the maxPointsPerPartition. Otherwise, the partition method recursively splits the rectangle and resizes it until the rectangle meets that criteria.

The partitions are finally merged into common clusters. These spatial points are formatted with a label of their respective partition. This variable is a RDD[id,point], where id is an integer referring to a partition; and point is a tuple of values identifying the spatial coordinates of a data point.

```
//Generate the smallest rectangles that split the space and count the points contained
    in each rectangle
val minimumRectanglesWithCount = vectors
   .map(toMinimumBoundingRectangle)
   .map((_, 1))
   .aggregateByKey(0)(_ + _, _ + _)
   .collect()
   .toSet

//Find the best partitions for the data space
val localPartitions = EvenSplitPartitioner
   .partition(minimumRectanglesWithCount, maxPointsPerPartition, minimumRectangleSize)

val localMargins = localPartitions
   .map({ case (p, _) => (p.shrink(eps), p, p.shrink(-eps)) })
   .zipWithIndex

//Broadcast local variable localMargins to be used for parallel processing in the next
    step
val margins = vectors.context.broadcast(localMargins)

// Assign each point to its proper partition
val duplicated = for {
   point <- vectors.map(DBSCANPoint)
   ((inner, main, outer), id) <- margins.value
   if outer.contains(point)
} yield (id, point)

val numOfPartitions = localPartitions.size
```

Listing 6.3: Data Partitionning in parellel DBSCAN

Step 2 Local DBSCAN applies Local DBSCAN to each partition of the data as detailed in Listing 6.4. Spatial points are first grouped into their partition using the Spark groupByKey transformation. Then, a traditional local DBSCAN is applied to each partition using the Spark flatMapValues transformation on an array of spatial points in this case.

```
// Perform local dbscan
val clustered = duplicated
```

```
    .groupByKey(numOfPartitions)
    .flatMapValues(points =>

new LocalDBSCANNaive(eps, minPoints).fit(points)).cache()
```

Listing 6.4: Local DBSCAN

Step 3 Global Clustering merges above local clusters into global clusters as detailed in Listing 6.5. The RDDs in step 2 are labelled by a local cluster id from the local DBSCAN transformations. The map transformation is applied to the local cluster id by assigning a distinct id to unique clusters and a common cluster id to connected clusters.

```
//find all cluster ids
val localClusterIds = clustered
    .filter({ case (_, point) => point.flag != Flag.Noise })
    .mapValues(_.cluster)
    .distinct()
    .collect()
    .toList

val (total, clusterIdToGlobalId) = localClusterIds.foldLeft((0, Map[ClusterId, Int]()))
    {
case ((id, map), clusterId) => {
    map.get(clusterId) match {
        case None => {
            val nextId = id + 1
            val connectedClusters = adjacencyGraph.getConnected(clusterId) + clusterId
            logDebug(s"Connected clusters $connectedClusters")
            val toadd = connectedClusters.map((_, nextId)).toMap
            (nextId, map ++ toadd)  }
        case Some(x) => (id, map)
        }
    }
  }
```

Listing 6.5: Global Clusters generation source code snippet

6.5 The Workflow Output

Finally, the output of DBSCAN Clustering method results in the time series labeled with their associated cluster. The cluster is plotted in Figure 28. The points represent the spatial projection generated by PCA on the distance matrix. To compare with the Neighbor-Joining clustering method, the data points are plotted with labels starting with BGP. It can be observed that the objects

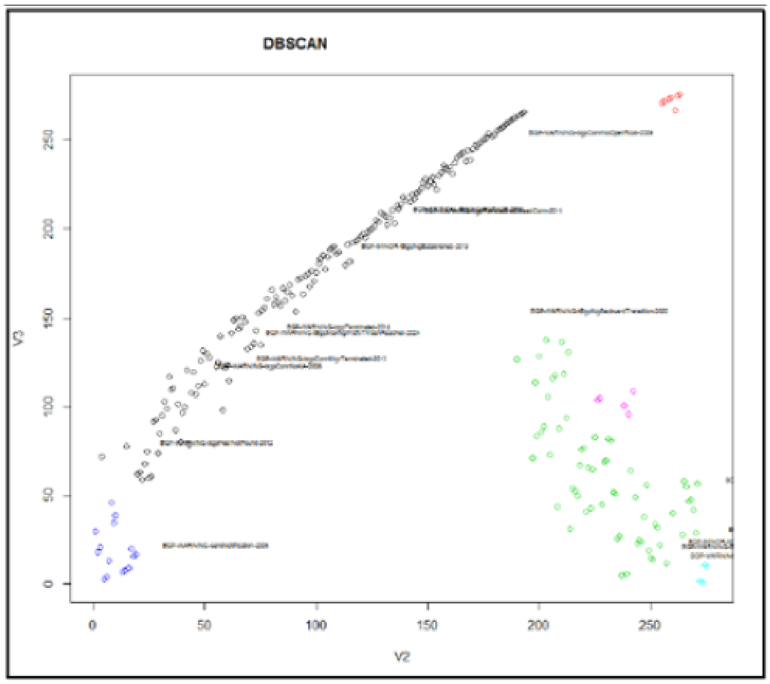


Figure 28: Visualization of DBSCAN output resulting in 6 clusters based on Eps 40

similarly labelled further spread into five clusters of different color schemes.

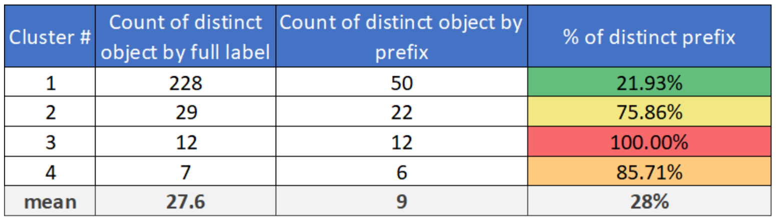
The output of this workflow provides the ability to visualize the spatial positioning of the time series and assess the presence of any unknown clusters. The output of this workflow is a tool which can be used to address the problem statements questions **Q1**, **Q2**, and **Q3**. In this particular use case using this dataset and workflow, the resulting clusters do not provide a conclusive pattern that can be used to correlate or discarding features. Furthermore, to cross-validate with the findings from the Neighbor-joining output, Figure 28 27 provides two plots where each plots highlights labeled points with similar Prefixes. As a reminder, the Neighbor-joining workflow managed to group together time series with common prefixes which were cannot observe in these figures.

Although this particular workflow use case and dataset did not contain distinct clusters, the derived DBSCAN workflow provides another tool to help address the technical challenges on exploring high dimensional time series datasets and the questions captured in the problem statements **Q1**, **Q2**, and **Q3**.

6.5.1 Cluster Analysis

In this section, analysis of the DBSCAN workflow output is presented in order to confirm the design and implementation validity. If valid, the parallel workflow output should contain clusters where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other. Although initial observations of the clusters are provided, data mining effort to search and generate information from this dataset was not in scope in this thesis.

The results in Figure 28 was selected as it shown the most count of clusters. This particular



Cluster #	Count of distinct object by full label	Count of distinct object by prefix	% of distinct prefix
1	228	50	21.93%
2	29	22	75.86%
3	12	12	100.00%
4	7	6	85.71%
mean	27.6	9	28%

Figure 29: Visualization of DBSCAN output resulting in 4 clusters based on eps 100

results was presented due to its clear clustering based on the visually observed data spatial density. **Further exploration and Industry input would be needed in order to ensure clusters are deterministic.**

Although, data mining was not the objective of this thesis, the observation from the Neighbor-joining output was cross validated with the sample dbscan output in Figure 28 . Selected prefix labels were displayed on their respective spatial data points and it can be observed that data points with similar prefix are not aggregated in any particular clusters, but are in fact spread out across all the data space. The absence of the observation found in the Neighor-joining workflow does not invalidate the ability to cluster data points based on density, but that this observation is not found in using the dbscan workflow results with these specific parameters and 2D Spatial dimension.

Furthermore, The motivation was to compliment results from NJ, the same hypothesis was applied on these results: "Are similarly labeled time series clustered together?". Due to the number of data points and the inability to clearly label each points with their respective labels, only one subset of data points with similar prefix was labelled on the figure. It can be observed that these similarly labeled data points are spread out across all clusters. The methodology to statistically evaluate our hypothesis is to count the numbers of points in each cluster was also applied on dbscan outputs. In Figure 29, we observe the statistical evaluation on the dbscan output where 4 clusters were observed. By Counting ratio by which the similarly labelled timeseries are found within each cluster, we can notice that cluster one has the lowest ratio of distinct Prefix. **In summary, In Figure 29,using the same logic as in the NJ workflow, the lower the % of distinct prefix, the higher the similarity for time series within the respective cluster. Hence, it can be noticed that clusters from the dbscan output do have high similarity within the first cluster, but subsequent clusters seem to be very distinct. From a statistical point of view, some similarity can be derived, but visual as shown in In Figure 28 similarly labelled time series do not seem to fall under the same spatial cluster when using 2 Principal components**

With further research time and access to our partner's development environment, exploration of this dataset using this workflow in higher dimensions would have been considered for more exhaustive analysis of the clustering results due to the assumption that information was loss by over reducing the dimensionality of the data given the implemention constraints at the time of this writting.In addition to explore different dimensions, more extensive cluster analysis would have been applied using the same methodology of statistically evaluating the distribution of similarly labeled times series and observe their cluster distribution for all times series.

Chapter 7

The System Evaluation

In this section, the thesis aims to evaluate the system performance and scalability of the two workflows developed by varying:

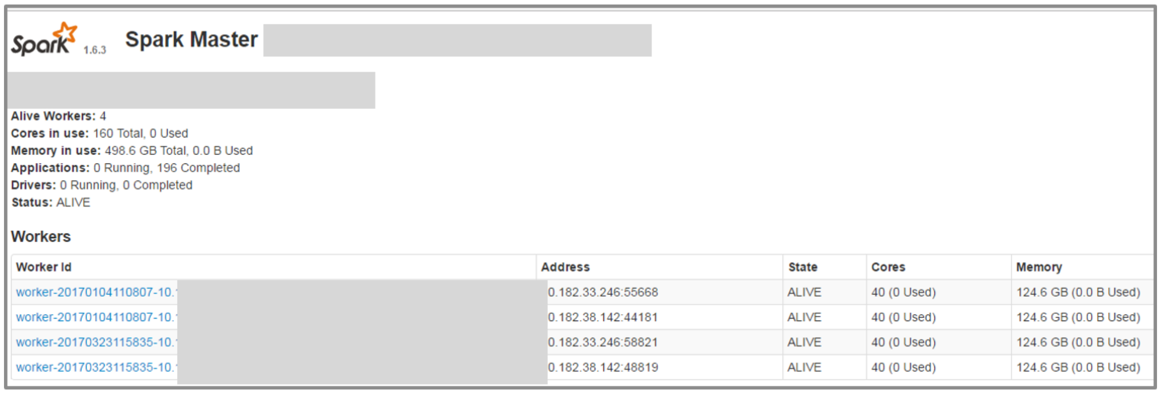
1. the cluster size
2. the parallelism
3. data size through resampling
4. partition through Spark transformations

These four factors impact the workload on the system. To further identify the performance bottlenecks, the thesis will experiment applying the workflows through different scenarios which will simulate different configurations of the four parameters above.

Scalability is a big data performance requirement as the data captured is exponential. Scalability is the capability of a system, network or a process to accommodate to rapid changes in the growth of data either in traffic or in volume. In algorithm design, scalability is said to be suitably scalable when facing a large processing load. If the algorithm fails when the load increases it does not scale. There are two general methods to add more resources:

1. Scaling horizontally - to add more nodes (or remove nodes) such as a server to a distributed cluster
2. Scaling vertically - to add more processing resources such as GPU or memory to a single computer or node in a system.

The chapter starts with presenting the experiment setup and the total resources available in the cluster. The chapter then presents the evaluation approach of each workflow derived and their results. The system level findings are then presented.



Spark 1.6.3 Spark Master [redacted]

[redacted]

Alive Workers: 4
Cores in use: 160 Total, 0 Used
Memory in use: 498.6 GB Total, 0.0 B Used
Applications: 0 Running, 196 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20170104110807-10.[redacted]	0.182.33.246:55668	ALIVE	40 (0 Used)	124.6 GB (0.0 B Used)
worker-20170104110807-10.[redacted]	0.182.38.142:44181	ALIVE	40 (0 Used)	124.6 GB (0.0 B Used)
worker-20170323115835-10.[redacted]	0.182.33.246:58821	ALIVE	40 (0 Used)	124.6 GB (0.0 B Used)
worker-20170323115835-10.[redacted]	0.182.38.142:48819	ALIVE	40 (0 Used)	124.6 GB (0.0 B Used)

Figure 30: Spark Master resources view with no jobs running.

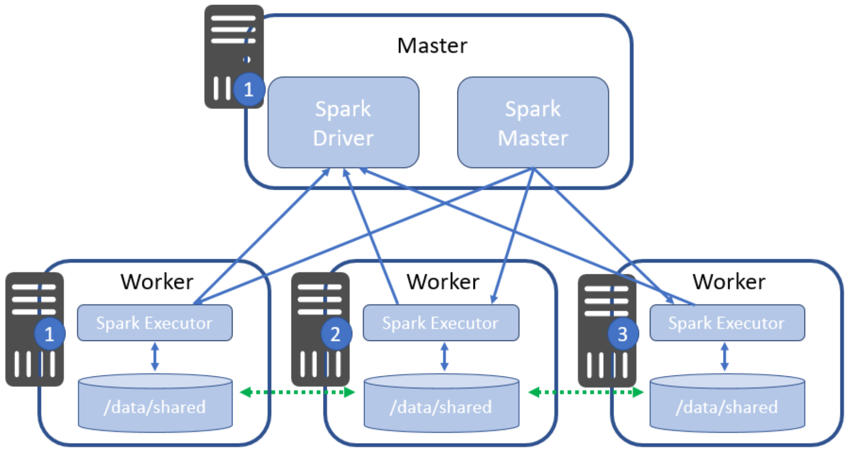


Figure 31: Spark Cluster Architecture in this experiment setup

7.1 The Experiment Setup

7.1.1 Environment

The experimental environment was an on-premise cluster made out of 3 in-house servers configured with Apache Spark 1.6.3. Each of those servers has 2 CPUs that translated into 10 Virtual Cores, 128 GB of RAM, and 2 TB of storage as shown in Figure 33. The cluster's total resource is provided in Figure 30. This cluster was configured with 1 master node and 2 worker nodes as depicted in Figure 31. Apache sparks require a storage area for parallel processing and intermediary file storage. In this on-premise cluster, the master node was leveraged as the distributed file system. To achieved this, a specific directory on the master node was mounted on the other two slave nodes using NFS - Network File System, a distributed file system protocol that allows you to mount remote directories on your server. Each node has the ability to write to this defined directory for any write operations raised by the execution of a Spark job. The resulting shared file system is depicted in Figure 32.

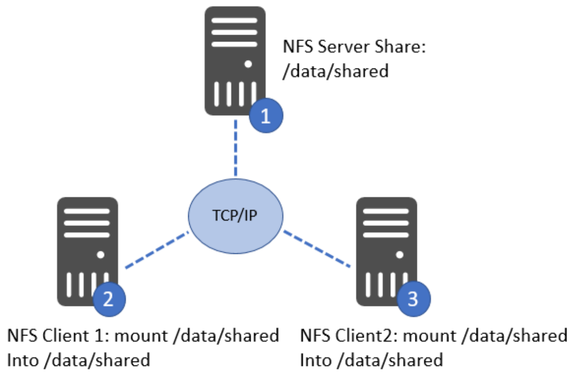


Figure 32: On-premise NFS Shared file system used for common storage between nodes in this experiment setup

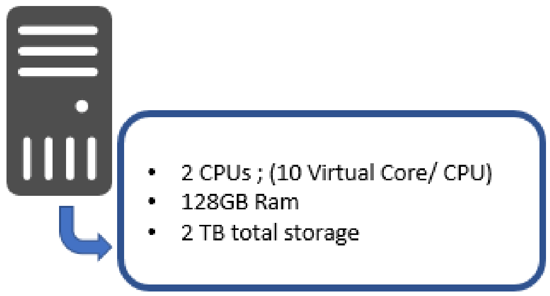


Figure 33: Individual Specifications of each server in the experiment setup

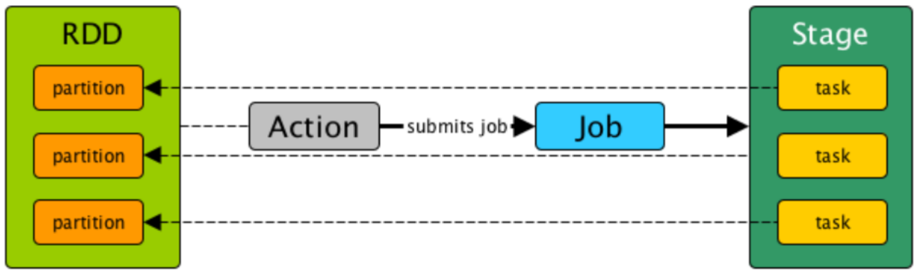


Figure 34: Job, Stages, Tasks of an RDD

7.1.2 Execution Terminologies

To assess and study the performance bottlenecks of the workflows, it is essential to understand how Spark evaluates a submitted program and breaks it down through its engine as Spark Jobs, Spark stages and Spark Tasks The three main building blocks of a Spark unit of executions are Jobs, Stages, and Tasks in order of magnitude.

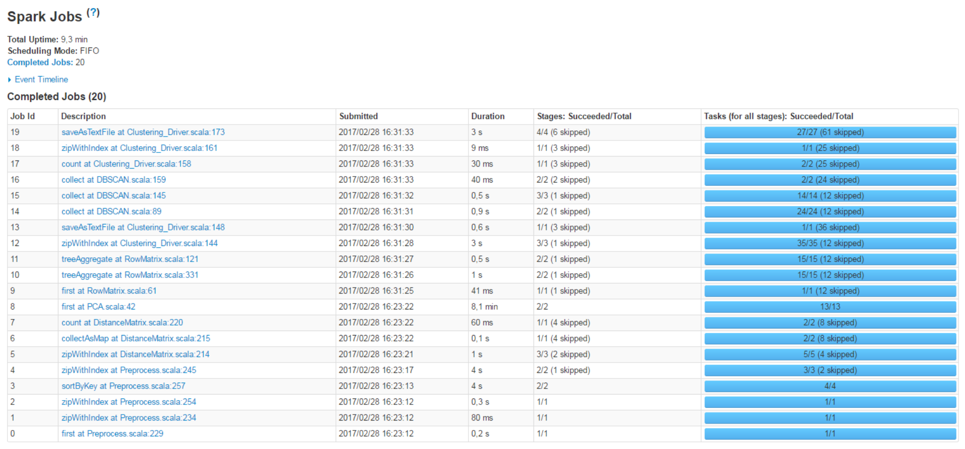
1. **Jobs** are computations that physically move data of an RDD in order to produce some result. These computations are sliced into Stages based on the presences of shuffle or reduce operations.
2. **Stages** compute partial results of a function executed as part of a Spark job. A Stage is composed of a set of parallel tasks of a single RDD
3. **Tasks** are the smallest unit of computational work in Spark which is associated to a Stage. Each task are performed on a single partition specific to an RDD

Basically, as presented in Figure 34 each job gets divided into Stages delimited by operations requiring data shuffle and each of the stages is broken down into smaller sets of tasks for execution on each partition of a specific RDD.

7.1.3 Performance Metrics

There are common evaluation metrics for each task - the smallest unit of execution in Spark. A task's execution time can be broken up as the Scheduler Delay, Deserialization Time, Shuffle Read Time, Executor Runtime, Shuffle Write Time, Result Serialization Time, and Getting Result Time. Assessment and tuning of these aspects can help optimize performance and understand system-level bottlenecks. [54].

1. **Executor Computing Time** consists of the following parts: data read/write time from and to the file system, CPU execution time, and Java garbage collector (GC) time
2. **Getting Result Time** consists of the time to collect the results from all the partition to the driver node.
3. **Result Serialization Time** consists of the duration for Spark to perform object serialization.



Spark Jobs (?)

Total Uptime: 9,3 min
Scheduling Mode: FIFO
Completed Jobs: 20

▸ Event Timeline

Completed Jobs (20)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
19	saveAsTextFile at Clustering_Driver.scala:173	2017/02/28 16:31:33	3 s	4/4 (6 skipped)	27/27 (61 skipped)
18	zipWithIndex at Clustering_Driver.scala:161	2017/02/28 16:31:33	9 ms	1/1 (3 skipped)	1/1 (25 skipped)
17	count at Clustering_Driver.scala:158	2017/02/28 16:31:33	30 ms	1/1 (3 skipped)	2/2 (25 skipped)
16	collect at DBSCAN.scala:159	2017/02/28 16:31:33	40 ms	2/2 (2 skipped)	2/2 (24 skipped)
15	collect at DBSCAN.scala:145	2017/02/28 16:31:32	0,5 s	3/3 (1 skipped)	14/14 (12 skipped)
14	collect at DBSCAN.scala:89	2017/02/28 16:31:31	0,9 s	2/2 (1 skipped)	24/24 (12 skipped)
13	saveAsTextFile at Clustering_Driver.scala:148	2017/02/28 16:31:30	0,6 s	1/1 (3 skipped)	1/1 (36 skipped)
12	zipWithIndex at Clustering_Driver.scala:144	2017/02/28 16:31:28	3 s	3/3 (1 skipped)	35/35 (12 skipped)
11	treeAggregate at RowMatrix.scala:121	2017/02/28 16:31:27	0,5 s	2/2 (1 skipped)	15/15 (12 skipped)
10	treeAggregate at RowMatrix.scala:331	2017/02/28 16:31:26	1 s	2/2 (1 skipped)	15/15 (12 skipped)
9	first at RowMatrix.scala:61	2017/02/28 16:31:25	41 ms	1/1 (1 skipped)	1/1 (12 skipped)
8	first at PCA.scala:42	2017/02/28 16:23:22	8,1 min	2/2	13/13
7	count at DistanceMatrix.scala:220	2017/02/28 16:23:22	60 ms	1/1 (4 skipped)	2/2 (8 skipped)
6	collectAsMap at DistanceMatrix.scala:215	2017/02/28 16:23:22	0,1 s	1/1 (4 skipped)	2/2 (8 skipped)
5	zipWithIndex at DistanceMatrix.scala:214	2017/02/28 16:23:21	1 s	3/3 (2 skipped)	5/5 (4 skipped)
4	zipWithIndex at Preprocess.scala:245	2017/02/28 16:23:17	4 s	2/2 (1 skipped)	3/3 (2 skipped)
3	sortByKey at Preprocess.scala:257	2017/02/28 16:23:13	4 s	2/2	4/4
2	zipWithIndex at Preprocess.scala:254	2017/02/28 16:23:12	0,3 s	1/1	1/1
1	zipWithIndex at Preprocess.scala:234	2017/02/28 16:23:12	80 ms	1/1	1/1
0	first at Preprocess.scala:229	2017/02/28 16:23:12	0,2 s	1/1	1/1

Figure 35: Sample DBSCAN workflow completed Spark jobs summary

4. **Scheduler Delay** consists of time gaps between tasks where the Spark framework is searching for the best executor satisfying it's configured data locality preference.

5. **Shuffle Read Time** consists of the duration for shuffle data read operations.

6. **Shuffle Write Time** consists of the duration for shuffle data write operations.

In this study, performance bottlenecks will be assessed against these performance metrics as a variation to the factors affecting the workload (cluster size, parallelism, data size, and partition) are applied. The following section addresses the performance study of the two workflows derived. The evaluation approach was to extract the data for all the tasks from each execution summary provided from the Spark UI as seen in Figure 34, consolidate them and analyze their distribution by metrics and their fluctuation as the factors that impact the workload (cluster size, parallelism, data size, and partition) vary.

7.2 Scalability of Neighbour-Joining Workflow

Scalability is the ability of a system to sustain increasing workloads by making use of additional resources. In the context of the clustering workflows, the workload is produced by data processing tasks that load the data into RDDs and perform transformations on RDDs. A data processing task is broken up across stages, therefore generates new RDDs and thus new partitions as the result of each stage. Each task is assigned to a single or multiple cores. The number of cores affects the level of parallelism by driving the number of Spark executors required for a task.

Each Spark executor hosts a single or multiple data partitions that the tasks are executed. Thus the number of executors for a stage is driven by the factors of (1) the number of tasks of a stage, which is driven by the transformations within a workflow; (2) spark.task.cpus, the number of cores allocated for each task, which is one by default; and (3) spark.executor.cores, the number of cores to use on each executor, which is one by default.

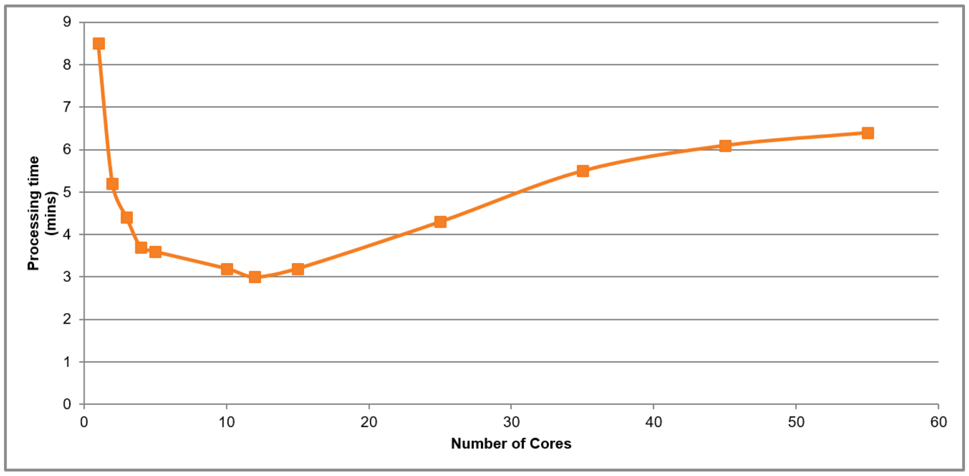


Figure 36: Scalability of the Neigbhour-Joining workflow

In the evaluation experiments, the same dataset was maintained and the total number of cores was varied to observe the processing time of the same workload. Figure 36 plots the processing time of the Neighbour-Joining workflow as the number of cores scales from 1 to 55. Beyond 55 cores, it has not been observed any improvement. The processing time is more than 8 minutes when only a single core is set. The optimal processing time is reduced to approximately 3 minutes at 12 cores allocated. Over 12 cores, the overhead of data partition on distributed cores surpasses the performance gain of parallelism.

To further identify the tasks that contribute most to the processing time, the processing time is decomposed as plotted in Figure 37 when the number of cores changes from 10 to 20. The plot shows the most significant change is the percentage of Shuffle Read/Write Time that increases from 7.6% to 17.18%, more than twice. This confirms that increasing the number of cores incurs the higher level of parallelism but more overhead of data shuffling.

7.3 Scalability of DBSCAN Workflow

The DBSCAN workflow is evaluated in the same experimental setup. The processing time of the workflow is measured by varying the number of cores. The measured processing time includes the stages of DTW and PCA. The plot in Figure 38 shows the processing time of the workflow is approximately 6 minutes. After the system reaches 20 cores, the processing time remains a plateau value of approximately 1 minute even when the number of cores increases from 20 to 60 cores as opposed to the scalability plot of the Neighbour-Joining workflow where the processing time goes back up past the optimal amount of cores. The plot shows changing the number of cores that increases the level of parallelism has no significant effect on the processing time.

To identify the intrinsic factors leading to this scalability behavior, the processing time of each job is decomposed as shown in Figure 39. As mentioned previously, a task's execution time can be broken up as the Scheduler Delay, Deserialization Time, Shuffle Read Time, Executor Runtime,

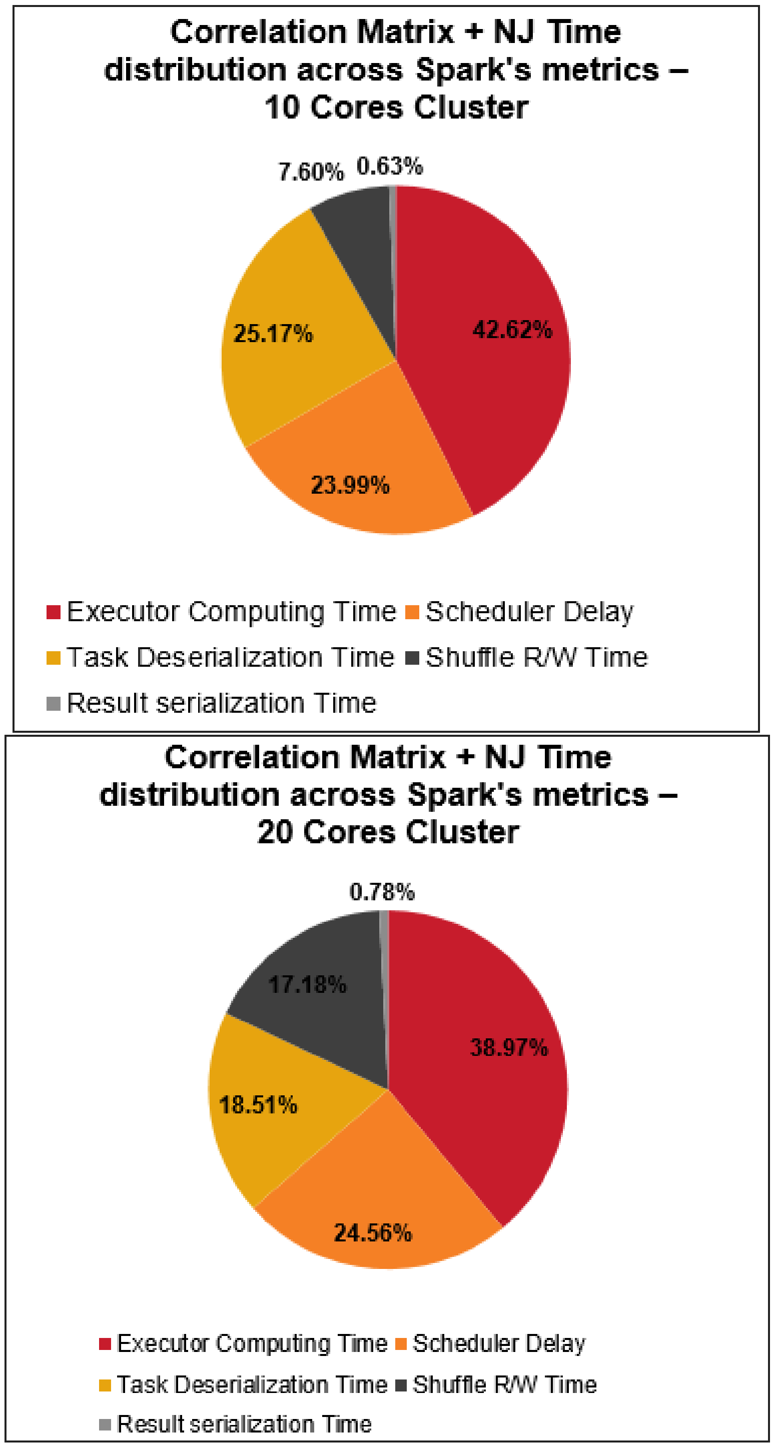


Figure 37: Processing time decomposition of Neighour-Joining tasks by varying number of cores

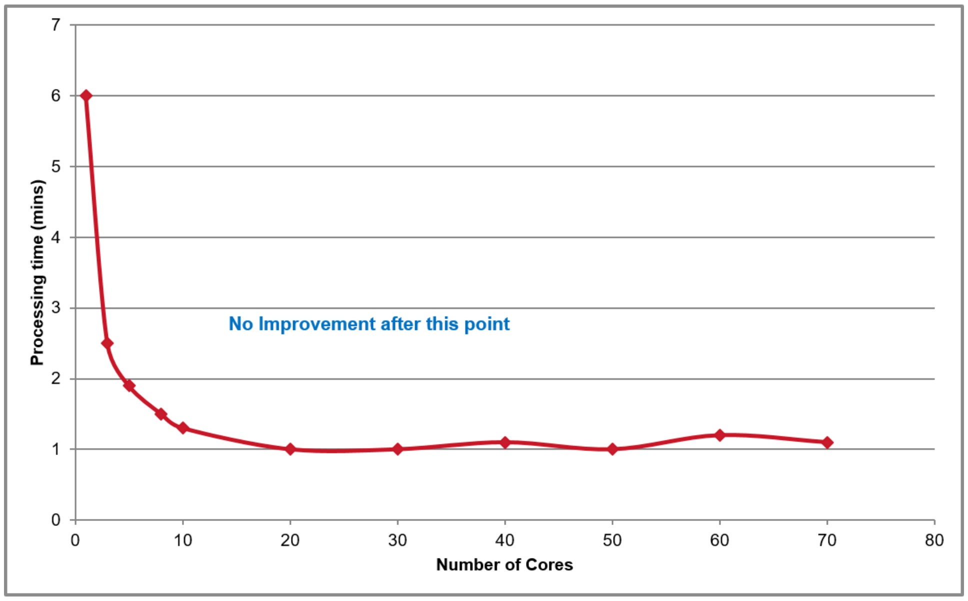


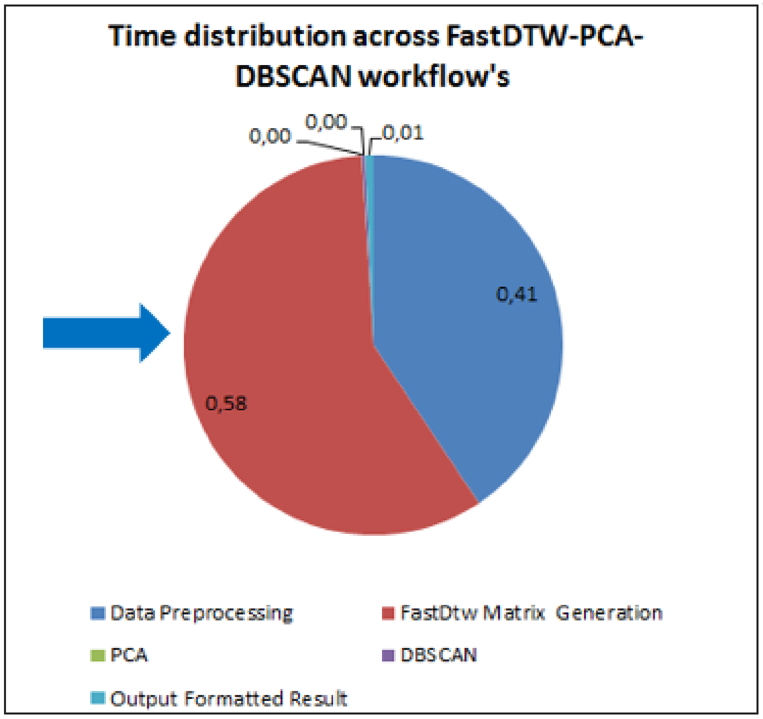
Figure 38: Scalability of the DBSCAN workflow

Shuffle Write Time, Result Serialization Time, and Getting Result Time. Figure 39 was generated by aggregating all these tasks level execution details in order to understand which type of system level operation impact the length of the jobs processing time. The top chart of Figure 39 provides the job execution breakdown for the DBSCAN workflow using the Correlation matrix. It can be observed that the majority of the processing time is spent on data preprocessing and generation of distance metrics as inputs to PCA and DBSCAN.

The bottom chart of Figure 39 was generated by using the DBSCAN workflow with FastDTW distance measure instead of the Correlation. The goal in this figure was to observe and compare how a different distance measure would affect the performance of the workflow.

It can be observed in Figure 39 a larger data processing time for the workflow using FastDTW than its counterpart. This large difference in processing time can be attributed and justified by the computational complexity of the distance measures. Fast DTW has a linear $\mathcal{O}(n)$ complexity [22] while Pearson's correlation is a constant $\mathcal{O}(c)$ complexity. Figure 39 shows the fast DTW occupies approximately 58% of the total process time, while the correlation method only incurs 15% of the total processing time.

Above results indicate the operations in the DTW-PCA-DBSCAN workflow reach to the optimal level of parallelism running on 20 cores given the dataset. The data shuffling load in this workflow is not a dominating factor and thus increasing the number of cores does not adversarially cause overhead on the processing time.



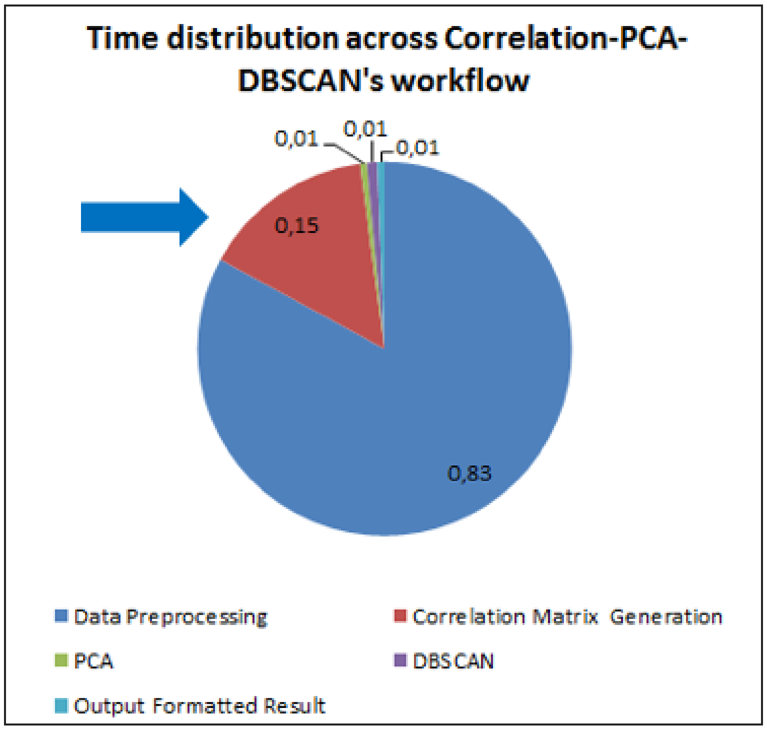


Figure 39: Processing time decomposition of DBSCAN tasks with different distance metrics

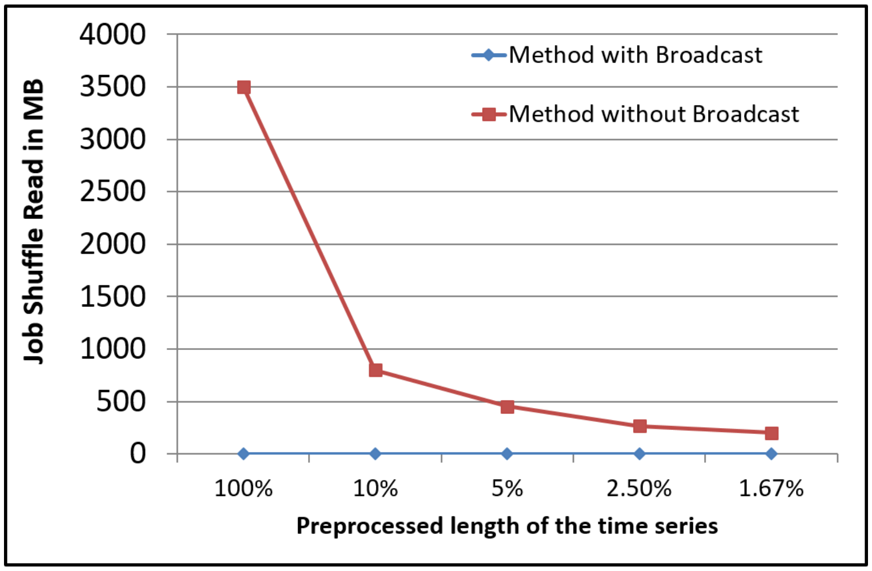


Figure 40: Effect of a Spark transformation using Broadcast on Job Shuffle Read

7.4 Parallism and Performance Tuning

The performance of both workflows are affected by common factors related to research questions presented in the introduction section, namely **(R1)** data correlation; **(R2)** in-memory representation of high dimension and partitioned data sets, and **(R3)** algorithms to transform the data. In this section, performance tuning techniques are discussed for each factor identified within the workflows.

7.4.1 Reduce Data Shuffling

Data shuffling is caused when tasks cross multiple stages need the same data, the data on demand is transferred across the network and passes through the software stack. An example of these operations are ReduceByKey, GroupByKey from Spark. In these operations, all the key value tuples from all partitions are shuffled across the cluster to conform to the reducing rule. This generates significant data transferred over the network and negatively affects the end-to-end performance of the analytic workflow.

The optimization technique is using Spark broadcast variables to keep read-only data cached on each node. The preprocessed time series data (see Figure 13 and Figure 14) are stored in broadcast variables on each worker nodes. The broadcast variables are also used in the Neighbour-Joining method (see Algorithm 1).

These broadcast variables are stored and retrieved from hashmap by the corresponding keys. It was observed that the byte size of Shuffle Read reduces to zero during those tasks when the broadcast variables are applied as plotted in Figure 40.

7.4.2 Prune Matrix Operations

Pruning matrix operations further optimize performance. The pairwise matrix in Neighbour-Joining and DTW is diagonally symmetric that means the upper diagonal contains the same information as the lower diagonal. Therefore, the computation on the lower matrix was pruned by mapping the computation from the upper matrix to the lower matrix. This pruning operation was performed for distance matrix generation as the parallel processing of pairwise time series along with a similarity measure is computationally expansive. A sample of such pruning operation can be observed in Listing 6.1 which highlights the FastDTW distance Matrix Generation source code. At the comment section mentioning *prune*, the upper diagonal matrix indexes are generated and only the time series referenced by these indexes are applied the $FastDTW$ mapping function. From it can be observed that the results are inverted to the lower diagonal by assigning the similarity value to the inverted indexes.

7.4.3 Truncate RDD Lineage Graph

An RDD lineage is a graph of all the parent RDDs of an RDD. The lineage graph is built when certain transformations are executed on the RDD. Meanwhile, a logical execution plan involving the parent RDD is created. This lineage graph also serves the base of resiliency as it allows to recompute missing or damaged partitions due to node failures. One issue encountered is the lineage graph is repetitively re-processing transformations within all previous iterations in every following iteration of the Neighbour-Joining algorithm. This is observed as the phenomenon that each iteration takes longer delay than its previous one. The solution is to truncate the RDD lineage graph at the end of each iteration, using the method of RDD.checkpoint() to save the previous computing result in RDDs to an HDFS filesystem.

7.4.4 Coalesce and Repartition

The intermediate transformation generates extra data even larger than the original input data size. An example is the RDD.union() the transformation used in each iteration of the Neighbor-Joining algorithm to combine RDDs that causes shuffling. If similar keys or range of keys are stored in the same partition then the shuffling is minimized and the processing of union() becomes substantially fast. However, it was observed that the number of partitions keeps doubling each time the RDD.union() transformation is invoked. This causes unnecessary data shuffling from/to new partitions. To solve this problem, a repartition was explicitly enforced on the RDDs. There are two partition methods:

1. Repartition allows increasing or decreasing of the number of partitions;
2. Coleasce only decreases the number of partitions.

The selection of the repartition method considers two factors: (1) the processing time of the algorithm; (2) when and how often the repartition should be applied.

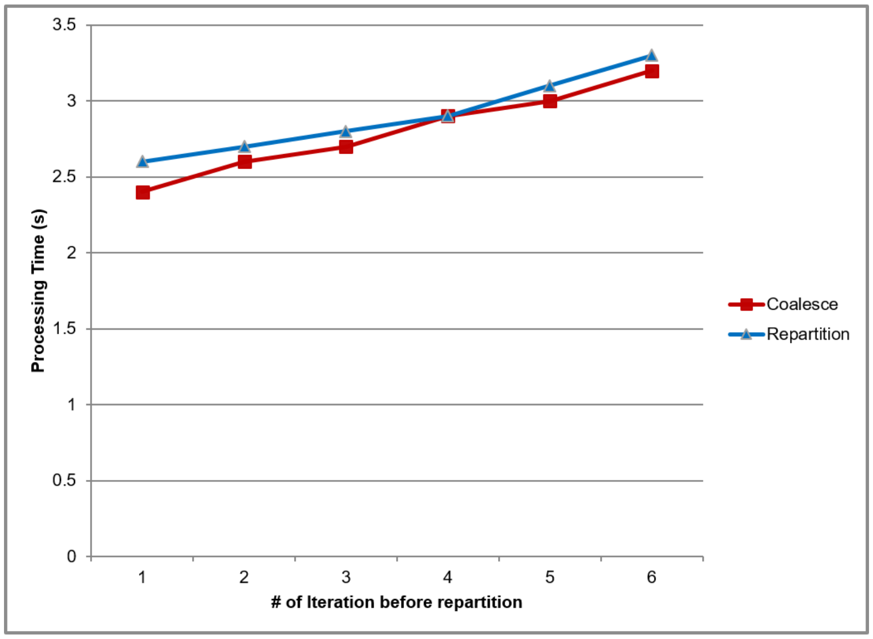


Figure 41: Comparing Coalesce and Repartition Effect for Processing Time

For the evaluation purpose, both methods were compared by measuring the processing time by changing the number of iterations before one partition method is invoked. The plot in Figure 41 shows that coalesce performs up to 8.5% better than repartition when it is applied at every iteration. The plot also indicates that repartition at every iteration in the case of the Neighbor-Joining algorithm on the dataset delivers better performance results.

7.4.5 Reduce Time Series Binning

Time series binning techniques were applied in the data preprocessing step of the workflows. In a nutshell, the binning technique consists of reducing a window of values into a single value. The bigger the window, the less information kept on the actual time series. The plots in Figure 42 displays a different level of binning with window sizes of 10, 20, 40 and 60. It was observed as the window size increases, less outlining points appear in the PCA projection. The binning window size becomes a tuning parameter as the trade-off between processing time and information retention.

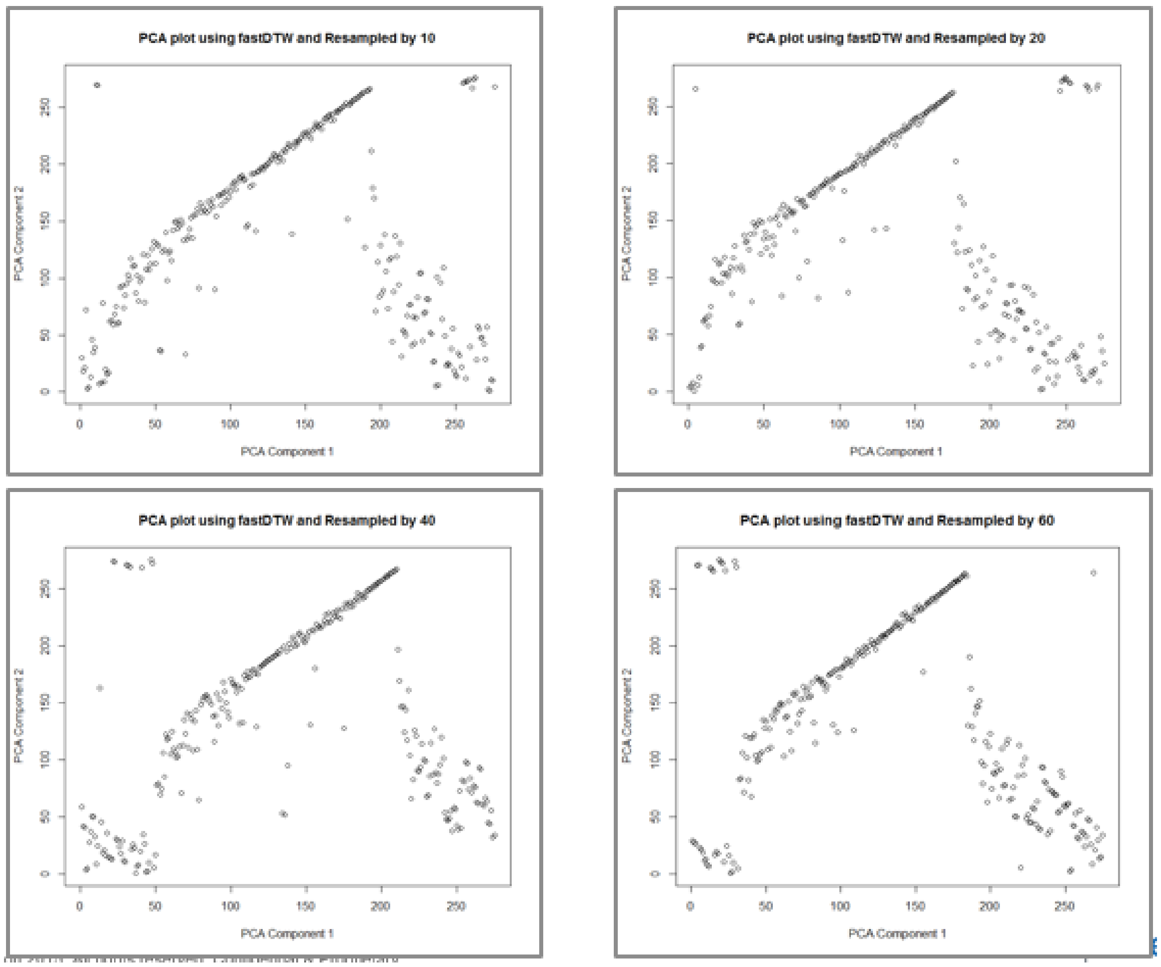


Figure 42: Time series in PCA projection with different levels of binning

Chapter 8

Reflection and Discussion

In this thesis, two workflows were developed integrating a different set of algorithms for clustering analysis. On the same dataset, parallelism was observed for each workflow and the effectiveness of data partition on the scalability and performance of each workflow. Based on the above system evaluation results, the insights applicable to distributed time series analysis, in general, were reflected.

Multiple Models. From raw data to extracted insights, there remains a chain of analysis steps. Each step has multiple choices of models. Composing multiple workflows of different models helps to produce an insightful understanding of the data, as well as efficiency (in term of parallelism) and effectiveness (in term of accuracy) of the analysis. For example, in the workflow of DBSCAN, the DTW model was identified as computationally intensive to generate distance metrics as an input to a clustering method. While in the workflow of Neighbour-Joining, the parallelization of this algorithm was achieved to an extent, since each iteration had dependencies on the previous one. In this case, the calculation steps between two iterations were designed to be parallelized using Spark.

Model Monitoring. In fact, any model eventually turns into operations and transformations on data in a data structure, such as a hash map, vector, and matrix. When such a data structure is partitioned across distributed nodes, the way of data partition is determined by the size of the data, the available nodes and cores, and the transformations. It is necessary to monitor the data partition size and the data shuffling ratio at run-time to make a decision on adjusting the data partition.

8.1 Further work

Due to the time limitation, a more extended evaluation of the workflows through more time series datasets were not accomplished. Different datasets of different sizes would have provided a more thorough analysis and evaluation on the effectiveness and the scalability of the workflows. The study and the derived workflow is general enough to be applicable to different datasets as long as the dataset provided is in time series format. Otherwise, pre-processing steps need to be considered to make sure that data inputted the workflows can be processed.

Although the workflow derived are generic enough for various datasets, obtaining the best performance of each dataset will require tuning of the certain components of the workflows mainly the DBSCAN parameters, the resampling rate, and the level of parallelism.

Experimentation with different datasets would have impacted the scalability and would have affected the workflow similarly to the experiments with different resampling rates. From a data mining perspective, it would have been particularly interesting to study different dataset to see what interesting patterns or insights can potentially be uncovered on various datasets.

Chapter 9

Conclusion

This thesis presents an investigation of the issues involved in distributed and parallel machine learning of IoT data streams. A systematic experimental principle was followed in order to develop two end-to-end workflows that consist of two sets of algorithms. The dominating factors of parallelism at the level of data operations and transformations that are common to a wide range of algorithms were investigated. Hence the observations are generally applicable, not specific to the algorithms examined. Based on the observations and experiences, this thesis advocate the practices of (1) the development principle of having multiple models to analyze the same datasets to produce complementary insights of the data; (2) reducing data shuffling by means of partition methods and choice of transformation; (3) monitoring at run-time the intermediate data partitions that are subject to re-partition; (4) increasing the parallelism by adding more computing cores is not necessarily improve the performance nor scalability. The bottleneck lies on the internal dependent iterations of algorithms.

In this thesis, the tuning and optimization of the workflow are still manually devised. Future research focuses on building a middleware layer that encapsulates the monitoring and re-partition so that they are automatically integrated to scale a workflow.

Bibliography

[1] Zeinab, Kamal Aldein Mohammed, and Sayed Ali Ahmed Elmustafa. "Internet of Things applications, challenges and related future technologies." World Scientific News 2.67 (2017): 126-148.

[2] Rusitschka, S., Eger, K. and Gerdes, C., 2010, October. Smart grid data cloud: A model for utilizing cloud computing in the smart grid domain. In Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on (pp. 483-488). IEEE.

[3] MR, M.S., 2017. Data Mining for Internet of Things. International Journal of Current Trends in Science and Technology, 7(12), pp.20556-20560.

[4] Owen, Sean, and Sean Owen. "Mahout in action." (2012).

[5] Bifet, Albert, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. "Moa: Massive online analysis." Journal of Machine Learning Research 11, no. May (2010): 1601-1604.

[6] Team, R. Core. "R: A language and environment for statistical computing." (2013): 201.

[7] Verleysen, Michel, and Damien François. "The curse of dimensionality in data mining and time series prediction." In International Work-Conference on Artificial Neural Networks, pp. 758-770. Springer, Berlin, Heidelberg, 2005.

[8] Cordova, Irving, and Teng-Sheng Moh. "Dbscan on resilient distributed datasets." In High Performance Computing & Simulation (HPCS), 2015 International Conference on, pp. 531-540. IEEE, 2015.

[9] Zaharia, Matei, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng et al. "Apache spark: a unified engine for big data processing." Communications of the ACM 59, no. 11 (2016): 56-65.

[10] Al-Neama, Mohammed W., Naglaa M. Reda, and Fayed FM Ghaleb. "Accelerated guide trees construction for multiple sequence alignment." International Journal of Advanced Research 2, no. 4 (2014): 14-22.

[11] Meng, Xiangrui, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman et al. "Mllib: Machine learning in apache spark." The Journal of Machine Learning Research 17, no. 1 (2016): 1235-1241.

[12] Ahmed, Nesreen K., Amir F. Atiya, Neamat El Gayar, and Hisham El-Shishiny. "An empirical comparison of machine learning models for time series forecasting." Econometric Reviews 29, no. 5-6 (2010): 594-621.

[13] Kalpakis, Konstantinos, Dhiral Gada, and Vasundhara Puttagunta. "Distance measures for effective clustering of ARIMA time-series." In Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, pp. 273-280. IEEE, 2001.

[14] Aghabozorgi, Saeed, Ali Seyed Shirkhorshidi, and Teh Ying Wah. "Time-series clustering–A decade review." Information Systems 53 (2015): 16-38.

[15] Montero, Pablo, and José A. Vilar. "Tsclust: An r package for time series clustering." Journal of Statistical Software 62, no. 1 (2014): 1-43.

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

[17] Alsheikh, Mohammad Abu, Shaowei Lin, Dusit Niyato, and Hwee-Pink Tan. "Machine learning in wireless sensor networks: Algorithms, strategies, and applications." IEEE Communications Surveys & Tutorials 16, no. 4 (2014): 1996-2018.

[18] Khan, Rafiullah, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. "Future internet: the internet of things architecture, possible applications and key challenges." In Frontiers of Information Technology (FIT), 2012 10th International Conference on, pp. 257-260. IEEE, 2012.

[19] Chen, Yen-Kuang. "Challenges and opportunities of internet of things." In Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific, pp. 383-388. IEEE, 2012.

[20] Bandyopadhyay, Debasis, and Jaydip Sen. "Internet of things: Applications and challenges in technology and standardization." Wireless Personal Communications 58, no. 1 (2011): 49-69.

[21] Earley, Seth. "Analytics, machine learning, and the internet of things." IT Professional 17, no. 1 (2015): 10-13.

[22] Salvador, S., and P. Chan. "Fastdtw: Toward accurate dynamic time." Warping in Linear Time and Space (2007).

[23] Hall, D., and D. Ramage. "Breeze: numerical processing library for Scala." (2009).

[24] Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise." In Kdd, vol. 96, no. 34, pp. 226-231. 1996.

[25] Morales, Gianmarco De Francisci, and Albert Bifet. "SAMOA: scalable advanced massive online analysis." Journal of Machine Learning Research 16, no. 1 (2015): 149-153.

[26] Al-Neama, Mohammed W., Naglaa M. Reda, and Fayed FM Ghaleb. "Fast vectorized distance matrix computation for multiple sequence alignment on multi-cores." International Journal of Biomathematics 8, no. 06 (2015): 1550084.

[27] Shyam, R., Bharathi Ganesh HB, Sachin Kumar, Prabaharan Poornachandran, and K. P. Soman. "Apache spark a big data analytics platform for smart grid." Procedia Technology 21 (2015): 171-178.

[28] Namiot, Dmitry. "On big data stream processing." International Journal of Open Information Technologies 3, no. 8 (2015).

[29] Salloum, Salman, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. "Big data analytics on Apache Spark." International Journal of Data Science and Analytics 1, no. 3-4 (2016): 145-164.

[30] Talavera, R., R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, and F. Martínez-Álvarez. "A nearest neighbours-based algorithm for big time series data forecasting."

[31] Klein, Levente J., Fernando J. Marianno, Conrad M. Albrecht, Marcus Freitag, Siyuan Lu, Nigel Hinds, Xiaoyan Shao, Sergio Bermudez Rodriguez, and Hendrik F. Hamann. "PAIRS: A scalable geo-spatial data analytics platform." In Big Data (Big Data), 2015 IEEE International Conference on, pp. 1290-1298. IEEE, 2015.

[32] Yaffee, Robert Alan, and Monnie McGee. An introduction to time series analysis and forecasting: with applications of SAS® and SPSS®. Elsevier, 2000.

[33] Mavridis, Ilias, and Helen Karatza. "Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark." Journal of Systems and Software 125 (2017): 133-151.

[34] Chen, Feng, Pan Deng, Jiafu Wan, Daqiang Zhang, Athanasios V. Vasilakos, and Xiaohui Rong. "Data mining for the internet of things: literature review and challenges." International Journal of Distributed Sensor Networks 11, no. 8 (2015): 431047.

[35] Díaz, Manuel, Cristian Martín, and Bartolomé Rubio. "State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing." Journal of Network and Computer applications 67 (2016): 99-117.

[36] De Mauro, Andrea, Marco Greco, and Michele Grimaldi. "What is big data? A consensual definition and a review of key research topics." In AIP conference proceedings, vol. 1644, no. 1, pp. 97-104. AIP, 2015.

[37] Botta, Alessio, Walter De Donato, Valerio Persico, and Antonio Pescapé. "On the integration of cloud computing and internet of things." In Future internet of things and cloud (FiCloud), 2014 international conference on, pp. 23-30. IEEE, 2014.

[38] Hashem, Ibrahim Abaker Targio, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. ”The rise of “big data” on cloud computing: Review and open research issues.” Information systems 47 (2015): 98-115.

[39] Cryer, Jonathan D., and Natalie Kellet. Time series analysis. Royal Victorian Institute for the Blind. Tertiary Resource Service, 1991.

[40] Chatfield, Chris. The analysis of time series: an introduction. CRC press, 2016.

[41] Rani, Sangeeta, and Geeta Sikka. ”Recent techniques of clustering of time series data: a survey.” International Journal of Computer Applications 52, no. 15 (2012).

[42] Aghabozorgi, Saeed, Ali Seyed Shirkhorshidi, and Teh Ying Wah. ”Time-series clustering–A decade review.” Information Systems 53 (2015): 16-38.

[43] Chen, Yanping, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. ”The ucr time series classification archive.” (2015): 23.

[44] Keogh, Eamonn, and Shruti Kasetty. ”On the need for time series data mining benchmarks: a survey and empirical demonstration.” Data Mining and knowledge discovery 7, no. 4 (2003): 349-371.

[45] Domo Inc., ”Data Never Sleeps 5.0”, `https://www.domo.com/learn/data-never-sleeps-5?aid=ogsm072517_1&sf100871281=1`, [accessed May 2019]

[46] De Mauro, Andrea, Marco Greco, and Michele Grimaldi. ”What is big data? A consensual definition and a review of key research topics.” In AIP conference proceedings, vol. 1644, no. 1, pp. 97-104. AIP, 2015.

[47] Katal, Avita, Mohammad Wazid, and R. H. Goudar. ”Big data: issues, challenges, tools and good practices.” In 2013 Sixth international conference on contemporary computing (IC3), pp. 404-409. IEEE, 2013.

[48] Labrinidis, Alexandros, and Hosagrahar V. Jagadish. ”Challenges and opportunities with big data.” Proceedings of the VLDB Endowment 5, no. 12 (2012): 2032-2033.

[49] Apache Spark, ”Tuning Spark”, `https://spark.apache.org/docs/latest/tuning.html#data-locality`, [accessed January 2019]

[50] Microsoft Contributors, November 03 2019, ”Horizontal, vertical, and functional data partitioning”, https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning, [accessed January 2019]

[51] Jacek Laskowski,”Partitions and Partitioning”, https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-partitions.html, [accessed May 2019]

[52] Databricks, ”Skew Join Optimization”, https://docs.databricks.com/spark/latest/spark-sql/skew-join.html, [accessed Januuary 2019]

[53] Lokesh Poojari Gangadharaiah, "An Intro to Apache Spark Partitioning: What You Need to Know", March 05 2018, https://www.talend.com/blog/2018/03/05/intro-apache-spark-partitioning-need-know/ , [accessed January 2019]

[54] IBM Knowledge Center, "Tuning Spark applications Tasks", https://www.ibm.com/support/knowledgecenter/en/SSZU2E_2.3.0/performance_tuning/application_spark_parameters.html, [accessed January 2019]

[55] IBM, "What is MapReduce?", https://www.ibm.com/analytics/hadoop/mapreduce, [accessed January 2019]

[56] Talend, "What is MapReduce?", https://www.talend.com/resources/what-is-mapreduce/, [accessed January 2019]

[57] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: a flexible daurlta processing tool." Communications of the ACM 53, no. 1 (2010): 72-77.

[58] Shoro, Abdul Ghaffar, and Tariq Rahim Soomro. "Big data analysis: Apache spark perspective." Global Journal of Computer Science and Technology (2015).

[59] Jim Scott,"Apache Spark: A Great Companion to Modern Hadoop Cluster Deployment", September 23 2015, https://mapr.com/blog/apache-spark-great-companion-modern-hadoop-cluster-deployment/, [accessed January 2019]

[60] Sumit Anand, "Hadoop vs Spark: The Best Big Data Frameworks", October 28 2016, https://acadgild.com/blog/hadoop-vs-spark-best-big-data-frameworks, [accessed January 2019]

[61] Raj Vardhan, "Spark Protip: Joining on skewed dataframes", March 9 2018, https://medium.com/simpl-under-the-hood/spark-protip-joining-on-skewed-dataframes-7bfa610be704, [accessed January 2019]

[62] Cox automotive Solution, "Data Skew", https://coxautomotivedatasolutions.github.io/datadriven/spark/data\%20skew/joins/data_skew/, [accessed January 2019]

[63] NCSU, "Introduction to Principal Components and FactorAnalysis", ftp://statgen.ncsu.edu/pub/thorne/molevoclass/AtchleyOct19.pdf, [accessed January 2019]

[64] Lindsay I Smith, "A tutorial on Principal Components Analysis",February 26, 2002,http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf,[accessed January 2019]

[65] M.Ester, H.P.Kriegel, J.Sander and Xu. , "DBSCAN: Density-based Spatial Clustering of Applications with Noise",http://www.cs.fsu.edu/~ackerman/CIS5930/notes/DBSCAN.pdf,[accessed January 2019]

[66] Statistics Solutions, ”Correlation (Pearson, Kendall, Spearman)”,https://www.statisticssolutions.com/correlation-pearson-kendall-spearman/,[accessed January 2019]

[67] Stan Salvador and Philip Chan., ”Google Code Archive: fast-dtw”,https://code.google.com/archive/p/fastdtw/,[accessed January 2019]

[68] Apache Spark, ”Lightning-fast unified analytics engine”,https://spark.apache.org/,[accessed January 2019]

[69] Encyclopedia of Database Systems, ”Data Skew”, https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_1088,[accessed January 2019]

[70] AI News Greg Benson,”How industry collaboration with academia advances the field of AI”, https://artificialintelligence-news.com/2018/02/22/industry-collaboration-academia-advances-field-ai/,[accessed January 2019]

[71] Ciena, ”Aligning Blue Planet with the future of intelligent network automation”, https://www.ciena.com/insights/articles/Aligning-Blue-Planet-with-the-future-of-intelligent-network-automation.html?srv, [accessed March 2019]

[72] Ashraf, Muhammad, Sevia Idrus, Farabi Iqbal, Rizwan Butt, and Muhammad Faheem. ”Disaster-resilient optical network survivability: a comprehensive survey.” In Photonics, vol. 5, no. 4, p. 35. Multidisciplinary Digital Publishing Institute, 2018.

[73] Côté, David. ”Using machine learning in communication networks.” Journal of Optical Communications and Networking 10, no. 10 (2018): D100-D109.

[74] Cheng, Haibin, Pang-Ning Tan, Christopher Potter, and Steven Klooster. ”Detection and characterization of anomalies in multivariate time series.” In Proceedings of the 2009 SIAM international conference on data mining, pp. 413-424. Society for Industrial and Applied Mathematics, 2009.

[75] Çelik, Mete, Filiz Dadaşer-Çelik, and Ahmet Şakir Dokuz. ”Anomaly detection in temperature data using dbscan algorithm.” In 2011 International Symposium on Innovations in Intelligent Systems and Applications, pp. 91-95. IEEE, 2011.

[76] Laptev, Nikolay, Saeed Amizadeh, and Ian Flint. ”Generic and scalable framework for automated time-series anomaly detection.” In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1939-1947. ACM, 2015.