REAL-TIME BIG DATA PLATFORM FOR DISTRIBUTED ENERGY LOAD FORECASTING WITH COMPUTING APPROACHES

A Dissertation

by

AMEEMA ZAINAB

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Ali Ghrayeb
Co-Chair of Committee,	Haitham Abu-Rub
Committee Members,	Erchin Serpedin
	Eyad Masad
	Le Xie
	Othmane Bouhali
Head of Department,	Miroslav Begovic

December 2021

Major Subject: Electrical Engineering

ABSTRACT

The proliferation of smart meters in the grids has resulted in an explosion of large energy datasets. Processing such big data is challenging and usually takes a longer time than the requirement of a short-term load forecast. In the era of big data, where information is one of the key factors in making decisions, this study is drawing attention to the need for data management in smart grids. For the utility to be able to plan the resources accurately and balance the electricity supply and demand, accurate and timely forecasting is required. Machine learning algorithms have been intensively applied to perform load forecasting to obtain better accuracies as compared to traditional statistical methods. However, with the huge increase in data size, sophisticated algorithms must be created which require big data platforms with adequate computational resources. Optimal and effective use of the available computational resources can be attained by maximizing the efficient utilization of the computational nodes of a big data platform. Parallel computing is demanded to allow for optimal resources utilization in dealing with smart grid big data. The work in this research addresses the concerns by deploying parallel computing capabilities to minimize the execution time while maintaining highly accurate load forecasting models. This work utilizes multi-node and multi-core processing to minimize the overall execution time of the forecasting models while ensuring acceptable accuracy by mapping simultaneous jobs to available processors. The obtained results demonstrate the efficacy of the proposed approach through real-time adoption of machine learning (ML) models, diminishing execution time, and enhancing scalability. This research will show how tree-based models have outperformed the other

models accomplishing a tradeoff between model accuracy and execution time. The proposed approach is validated on real big data provided by Iberdrola, a Spanish utility company. The data is acquired from one hundred thousand different data sources in the electrical distribution system and amounts to 2.2 billion records approximately. To enhance the analysis further, a master-slave parallel computing paradigm for load forecasting is deployed and experimentally verified. The work proposes a concurrent job scheduling algorithm in a multi-energy data source environment using Apache Spark. An efficient resource utilization strategy is developed for optimizing multiple Spark jobs to reduce job completion time. The clustering method is implemented to group the electrical distribution nodes into clusters to reduce the number of required forecasting models, additionally reducing computational time.

DEDICATION

This work is dedicated to my family.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Ali Ghrayeb, my committee co-chair, Dr. Haitham Abu-Rub, and my committee members, Dr. Othmane Bouhali and Dr. Le Xie for their guidance and support throughout the course of this research. I would like to thank Dr. Shady S. Refaat for his continuous support. I sincerely appreciate Mahdi's support and guidance throughout the project. Thanks also go to my friends, colleagues, the department faculty, and staff for making my time at Texas A&M University and Texas A&M University at Qatar a great experience.

Finally, thanks to my father and mother for their blessing and my husband for his support, patience, and love. Thanks to my siblings for their unending encouragement to pursue my goals.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Ali Ghrayeb, Professor Haitham Abu-Rub, Professor Erchin Serpedin, Professor Eyad Masad and Professor Le Xie of the Department of Electrical and Computer Engineering, Professor Othmane Bouhali of the Department of Physics and industry professionals Engineer Santiago and Mahdi Houchati. All the work conducted for the dissertation was completed by the student independently.

Funding Sources

This thesis was made possible by the NPRP grant [NPRP10-0101-170082] from the Qatar National Research Fund (a member of Qatar Foundation), an internal seed grant from Texas A&M University at Qatar, and the co-funding by IBERDROLA QSTPLLC. The findings achieved herein are solely the responsibility of the author[s].

NOMENCLATURE

ACID	Atomicity, Consistency, Isolation, and Durability
AMI	Advanced Metering Infrastructure
ANN	Artificial Neural Networks
API	Application Programming Interface
ARIMA	Auto-Regressive Integrated Moving Average
ARMSE	Average Root Mean Square Error
CAP	Consistency, Availability, Partition tolerance
CO	Number of Cores
COperE	Core per Executor
CSV	Comma Separated Values
D2R	Dynamic Demand Response
DAG	Directed Acyclic Graph
DBN	Dynamic Based Network
DNN	Deep Neural Network
DRNN	Deep Recurrent Neural Network
DT	Distribution Transformer
DTR	Decision Tree Regressor
DTE	Detroit-based Diversified Energy
ENEL	Ente nazionale per l'energia elettrica
ETL	Extract Transform and Load
FIFO	First In First Out

GBRT	Gradient Boosted Regression Trees
HDFS	Hadoop Distributed File System
HEALPix	Hierarchical Equal Area isoLatitude Pixelization
HPRC	High Performance Research Computing
IBM	International Business Machines
JCT	Job Completion Time
JVM	Java Virtual machines
LR	Linear Regression
MAPE	Mean Absolute Percentage Error
Mem	Memory
MemperE	Memory per Executor
ML	Machine Learning
MLP	Multi-Layer Perceptron
NN	Neural Network
ORC	Optimized Row Columnar
PMU	Phasor Measurement Unit
PSU	Power Supply Unit
RAM	Random Access Memory
RBF	Radial Basis Function
RDD	Resilient distributed datasets
RFR	Random Forest Regressor
RMSE	Root Mean Square Error

RMT	Random Matrix Theory
SATA	Serial Advanced Technology Attachment
SCADA	Supervisory Control and Data Acquisition
SLURM	Simple Linux Utility for Resource Management
STLF	Short Term Load Forecasting
SVM	Support Vector Machine
SVR	Support Vector Regreesor
TAMU	Texas A&M University
TAMU-Q	Texas A&M University at Qatar
TVA	Tennessee Valley Authority
vCPU	Virtual CPU
YARN	Yet Another Resource Negotiator

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION*[1]

The smart grid is re-engineering the electricity generation, transmission, and distribution throughout the world. Smart Grid is an amalgam of increased digital information with the electrical power grids. Managing the big data generated from the grid efficiently is the key to successful transformation to the smart grid era. Most of the scientific advancements are becoming data-driven and hence, big data management in smart grids is an emerging and interesting area of research for data scientists. The world is computationally challenged enough to develop new storage methods and processing technologies for such big data. Managing big data involves data cleaning, varied data source integration, and decision-making applications. This thesis focuses on the study of big data management and proposes a data management platform to help manage the big data in the smart grid. Data management tools and techniques have been leveraged to understand the sources and data types in the electric grid. The thesis work emphasizes the limitations of the existing computational solutions inclined towards applications for smart grid big data.

As the data size increases, the computations tend to be heavy and time-consuming, resulting in a challenging situation to forecast the load under short time constraints. Various computationally intelligent techniques have also been employed previously in the electric energy field [1]. Several ML algorithms were designed with the assumption that

[1] Reprinted with permission from "Distributed Tree-Based Machine Learning for Short-Term Load Forecasting With Apache Spark," by “Ameema Zainab, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat and Othmane Bouhali, 2021. IEEE Access, vol. 9, pp. 57372-57384, Copyright 2021 by Ameema Zainab.

the entire dataset fits in the memory. This assumption negatively affects the ML algorithms while impeding their performance. For instance, a support vector machine (SVM) has a space complexity of $O(m^2)$ and a training complexity of $O(m^3)$ [2], where m indicates the number of samples. Therefore, as the size of the data increases, parallel data structures, data reuse, and data partitioning become important characteristics. Resilient distributed datasets (RDDs) implemented in a Spark cluster computing framework exhibit in-memory characteristics [3]. This leads to the work of this thesis to use a typical architecture that can accommodate both the cluster computing framework and machine learning capabilities.

1.1 Problem Definition

Utility companies focus on enhancing the convergence rates to perform short-term and medium-term load forecasting. Smart meter data is collected at high velocity, variety, and volume; making the data characterized as big data. A large electrical grid with smart meters widely installed consists of distributed data stores. Distributed computing platforms with multiple nodes are required to process the data generated to create accurate load forecasting models. In a classical big data problem, historical data accumulated at one location is high in volume. In the current scenario of this problem statement, a large number of transformers are spread across the distribution network. This makes the problem statement challenging as the load forecast expects simultaneous execution of the forecasting models. This problem statement reflects two solutions, edge computing to forecast the load at the location of data generation or data concentrated at stored locations in the database and then processed by creating clusters. The problem

statement requires the short-term load forecasting of all the transformers simultaneously. It is highly crucial during the analysis to optimize the execution time of the forecast models and maintain the accuracy of forecasting models to obtain the economic operation of the power system.

1.2 Research Goal and Objectives

The goal of this research is to provide a big data management computing platform for convenient Extract Transform and Load (ETL) of the smart grid data to be able to make more informed decisions with high processing speed and highly accurate forecasting models.

1.2.1 Research Objectives

The following are the objectives of this work:

1. the use of parallel computing to perform simultaneous load forecasts in a multi-AMI environment to reduce the overall time needed for Short Term Load Forecasting (STLF).
2. Utilization of distributed machine learning models to perform load predictions with the highest possible accuracy.
3. Finding a suitable tradeoff between execution time to predict the load and the choice of machine learning model with the least error.
4. A scheduling algorithm to perform parallel and distributed execution of load forecast on the smart grid big data is proposed.
5. Spark parameter optimization in terms of the number of executors, number of cores per executor, and memory per executor.

6. ML model parameter optimization to gain high accuracy contained with measures to combat overfitting.
7. Testing the proposed methodology on one hundred thousand transformers' data without clustering, then make a comparison against the proposed clustering technique.

The main objective of this thesis is to provide a big data platform that is robust and reliable to be able to perform load forecasting and scalable to be able to handle data received from 100,000 transformers.

CHAPTER II

LITERATURE REVIEW*[2]

In this chapter, an introduction to the big data platforms and the literature review of the platforms utilized in the field of electric grids and smart grids are to be be presented.

2.1 Introduction

Big data offers potential insights and is crucial for the efficient functioning of the smart grid [4]. Information from big data is becoming more and more valuable, therefore many energy companies have invested in handling and utilizing the data to perceive, innovate and extract actionable insights. It is estimated in a preliminary assessment by a utility that the amount of data required to process transactions of its customers would reach about 25 gigabytes of data points per day [5]. The management of such a large data set is challenging. Energy companies such as ENEL, are moving towards new strategies and plans to be data-driven companies exploiting huge amounts of data obtained from the grid architecture, customers, etc. [6]. Many utilities and systems operators plan to migrate their data center to the cloud which may bring savings from this migration at €300,000,000 [7]. ENEL plans to focus on a platform model rather than a pipeline model involving data-driven networks. It is very crucial to timely manage the available smart grid data as it would help the utilities to understand the demand and perform a dynamic balance of demand and supply. Moreover, the correlations between different data features can identified [8]. Additionally, it is very crucial to identify different analytic and data

[2] Reprinted with permission from "Big Data Management in Smart Grids: Technologies and Challenges," by Ameema Zainab, Ali Ghrayeb, Dabeeruddin Syed, Haitham Abu-Rub, Shady S. Refaat and Othmane Bouhali, 2021. IEEE Access, vol. 9, 73046-73059, Copyright 2021 by Ameema Zainab.

management strategies for different applications and usage. The biggest players in the energy market have utilized big data technologies to manage the grid. National Grid, DTE Energy, and Ausgrid are some of the largest utilities which have used the International Business Machines (IBM) insights foundation to help improve their decision-making for monitoring assets health and maintenance [9]. Romeo project is a five-year and €16 million project led by Iberdrola Renowables Energia [10]. This project focuses on managing the data from the wind farms using predictive models and physical fault models to lower the operation and maintenance costs of the wind farms [11]. This thesis is conducted on a project that is co-sponsored by IBERDROLA which delivers real five-years consumption data of 100,000 transformers in Spain to perform one day ahead load forecasting. The importance and uses of managing big data from the grid are endless.

To perform the analytics on the data for required applications and visualization, relevant software technologies must be in place.

2.2 Related Work

Many frameworks have been proposed in the literature to understand the data flow, analyze the data, and manage the data in the smart grids. Previous works include the proposal and implementation of big data frameworks in the smart grids to take decisions on many aspects such as balancing demand, load forecasting, grid infrastructure optimization, asset management, consumer behavior analysis, state estimation, and service quality analytics, etc.

In [12], Mayilvaganan et al. proposed a cloud-based smart grid management architecture that analyses the big data for balancing the demand and supply to meet

customer needs. In [13], Yogesh et al. have proposed ‘Floe’s, a continuous data flow engine that utilizes a private cloud infrastructure. The proposed cloud based D2R (Dynamic Demand Response) platform performs intelligent dynamic demand response management relieving the load peaks in the power grid. In [14], Baek et al. proposed ‘Smart-Frame’ as a secure cloud-computing-based big data platform to analyze a voluminous amount of data acquired from power assets, smart meters, and distinct types of front-end devices in the grid. A popular cloud computing opensource platform called eucalyptus has been utilized for the prototype implementation [15]. TVA was selected by NERC (North American Electric Reliability Corporation) in 2009 as the repository for PMU data nationwide. America’s power grids at the TVA producing hundreds of terabytes of data have been handled with the help of apache Hadoop [16]. In [17], architecture named ‘SmartSantander’ which is a live city flexible big data platform has been introduced. In [18], ’SCOPE’ was presented as a smart city Cloud-based Open platform and ecosystem by Boston University. City Pulse [19] is proposed by Osborne Clarke, a smart city consulting firm from Europe [20]. FIWARE [21] is a smart energy platform for the development of intelligent applications in the future internet. It serves as an energy platform capable of supporting various business models for different smart energy industries. Wang et al. proposed wireless computing architecture for the processing and analysis of smart grid data [22]. Zhou et al. presented data mining and visualization techniques for smart grid data and achieved real-time monitoring of power consumption [23]. In [24] UlTraMan a unified platform for big data management and analytics for trajectory data is proposed. It offers a customized pipeline extension of

modules offering enhanced computing. ASTROIDE is a unified big data processing engine over spark for astronomical data. It introduces efficient query execution, by data partitioning with Hierarchical Equal Area isoLatitude Pixelization (HEALPix) on Spark [25]. Because the data is both complex and has different formats, handling the data is not straightforward. Big data technologies offer scalability, persistence, and are computationally efficient. Various technologies offer services that help in dealing with big data complexities. A comprehensive review of the storage and processing structures, database management systems, software technologies, architectures, systems benchmarking, and data indexing is outlined below.

2.2.1 Storage and Processing

2.2.1.1 Hadoop

Hadoop is a unified and centralized storage platform to manage various types of data. Hadoop augments itself by providing a repository where structured, semi-structured, and unstructured data can be processed together easily [26]. Along with being open-source software, Hadoop is fault-tolerant and has a very reliable storage system. Having a programmable storage system, it is flexible for users to analyse the data directly attached to the disk where it resides. However, Hadoop has limitations i.e. it supports only batch processing and is not efficient with real-time, iterative, and stream processing. The data collected from dispatched sources in the grid is stored in huge datasets. This data needs to be accessible by multiple users on multiple machines for analytics. The Hadoop framework helps in parallelizing the processing in cloud computing environments and permits users to attain a local copy of the stored data. The Hadoop distributed file system

is also well known for efficient storage of data as it provides fault tolerance, high availability, and scalability. For applications such as smart meter data analytics, load forecasting, and scheduling which require stream processing, hadoop is not very efficient as it cannot produce output in real-time with low latency. The Hadoop ecosystem is built of two components, MapReduce and Hadoop distributed file system (HDFS) and these are discussed in the next sections.

2.2.1.2 MapReduce

MapReduce is a parallel data processing system of Hadoop. It is the programming model used within Hadoop and it is efficient at processing huge volumes of data. MapReduce works on the concept of a job scheduler that assigns multiple tasks in parallel to data nodes in a single cluster or shared clusters and results are collated, filtered, sorted, and then passed out as an output. If the task assigned to a node is overloaded or failed in a cluster, then the task is executed by another server in the cluster as shown in Figure 1. MapReduce can execute in a potential number of high-level languages such as C, C++, and scripting programming languages i.e. Python, Perl, and PHP. It can also be noted that as MapReduce processes large datasets, it requires a large amount of time and might result in increased latency. Running on various clusters results in increased time and lesser processing speeds. This limitation can be overcome by the in-memory computation capability of the Hadoop spark. MapReduce does not have an interactive mode. However, this can be overcome by adding Hive Hadoop [27] or Pig Hadoop [28] and this enables users to have an interface to deal with the MapReduce paradigm without having to code complex java MapReduce programs.

2.2.1.3 HDFS and HopsFS

The file storage system in Hadoop is called the Hadoop distributed file system (HDFS). Because of its write once and read many models, it is best suited for data integrity when a read operation is performed. Many grid centers utilize Hadoop with HDFS file storage to collect various types of data from the grid such as phasor measurement units (PMU). HDFS however doesn't support random reading of small file sizes. It is designed in a way to support a small number of large datasets rather than a large number of small datasets. This can be overcome by merging the small files into one and then copying the bigger files to HDFS.

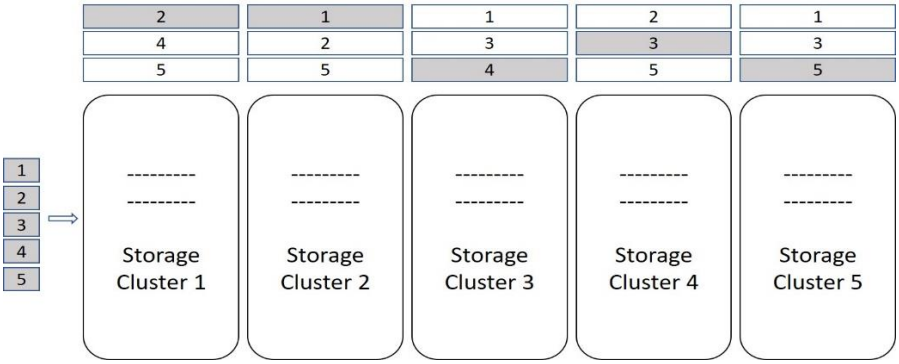

Figure 1 Software Framework – MapReduce

HopsFS is an open-source file system and it is an alternative to HDFS [29]. It uses the active and standby name nodes and thereby overcomes the deficiencies of HDFS. The name nodes in HopsFS can process the metadata not just locally in memory but also the metadata stored in the database. HopsFS works with different varieties of NewSQL

databases even if the databases have different licenses. It is since HopsFS uses Data Access Layer (DAL) as encapsulation to the database operations.

2.2.1.4 Apache Spark

Apache Spark is a lightning-fast framework that processes data that exists in data storage systems such as HDFS, Amazon S3 [30], MapR FileSystem [31], Cassandra [32], etc. The data processing also utilizes a cluster manager such as spark cluster, Apache Mesos, HadoopYARN, etc. [33]. Spark can process the data as it comes, even millions of events per second as it uses Resilient Distributed Datasets (RDDs) which reside in memory. The flexibility, speed, and scalable features of spark address the challenges of big data in smart grids. Spark also supports user-friendly APIs such as Python, Scala, Java, etc. and this makes developers easily use spark for machine learning libraries [34]. The very nature of data from smart grids (for example, the data from SCADA) is dynamic and anomalies in electrical systems tend to occur in milliseconds. Apache spark supports the real-time processing of the data and it can capture real-time information from the grid. Memory management in spark is crucial and involves various levels such as memory only, memory and disk, memory only serialization, and memory and disk serialization. Based on the size of the data, the memory allocation is altered.

2.2.1.5 Resource scheduler

A key to efficient utilization of a large asset is the choice of a suitable resource scheduler. Both supercomputers and big data systems use schedulers to allocate computing resources for the execution of submitted processes. The authors in [35] analyze 15 schedulers in both supercomputing and big data architectures. In [36], the

authors utilized up to 32 processors with the help of the Simple Linux Utility for Resource Management (SLURM) resource scheduler. Four of the most popular open-sourced schedulers include SLURM, Apache YARN, Apache Mesos, and Kubernetes.

2.2.2 Database Management Systems

Picking a relational SQL or a non-relational (NoSQL) database is one of the crucial decisions in choosing a database system. Both types of databases are suitable options, however, non-relational databases are constantly replacing relational databases as non-relational databases are efficient for big data applications. The cost of scaling relational databases is very high and the volume of data is ever-increasing in big data. Moreover, the ACID properties (Atomicity, Consistency, Isolation, and Durability) set unrequired constraints and hindrances to applications and these pose a challenge [37]. Therefore, relational databases are best avoided in big data applications.

NoSQL data storage has more ability to perform better adaptability, scaling, and performance when compared to relational databases. Although it must be noted that NoSQL does not have a universal query language that fits with all data models. Instead, it allows for RESTful coherence to the data and the query APIs. A comprehensive study explains the uses and performance comparisons between relational and non-relational databases [38]. Some of the non-relational databases include Redis, MemCached, Dynamo, Cassandra, PNUTS, MongoDB, CouchDB, Neo4j, HyperGraph DB, etc. The comparison between the relational and NoSQL databases is discussed in Table 1.

Table 1 Software framework – MapReduce

Characteristic	Relational Databases	NoSQL Databases
Data representation	Predefined schemas. The schema represents a logical view in which the data is organized & the relations are displayed.	Dynamic schema for unstructured data
Data Structure	Structured	Unstructured or lenient structure
Scaling	Vertically scalable. The amount of data stored depends on the physical memory of the system. Relational databases are scaled by increasing the hardware resources like CPU, RAM, SSD, etc. on a single server.	Horizontally scalable. No limit on data storage. NoSQL databases are scaled by increasing database servers.
Examples	MySQL, Oracle, SQLite, Postgres, MS-SQL, etc.	MongoDB, Bigtable, Redis, RavenDB, Cassandra, HBase, CouchDB, Graph databases like Neo4j, OrientDB, InfiniteGraph, AllegroGraph, etc.

Table 1 continued

Characteristic	Relational Databases	NoSQL Databases
Types	Table based databases	Column DB, Graph DB, Key-value pair DB, Document DB, etc.
Properties	ACID (Atomicity, Consistency, Isolation, Durability)	CAP (Consistency, Availability, Partition tolerance)
Language	Structured Query Language for data definition & manipulation	Unstructured Query Language
Development	Mix of open source (PostgreSQL) & closed (Oracle)	Open source
Complex Querying	Suitable for complex querying	does not have standards to perform complex queries.
Complexity	If records do not fit in the pre-defined schema tables, then the design of the database table becomes complex.	Schema is easily changed here as it is dynamic.

Table 1 continued

Characteristic	**Relational Databases**	**NoSQL Databases**
Community	Widely supported by vendors	Only community support
Normalization	Necessary	No constraint of normalization
Maintenance	High maintenance	Low maintenance with features like automatic repair, easier distribution of data & simpler data models is available. So, the administration is easy & so is the tuning requirement.
Consumer-friendly	GUI mode tools are available.	GUI mode tools are not available.

There are many other databases in the market that provide support to the requirements of huge data sizes, different data types, and high speed. The big databases include in-memory or main memory databases, object-oriented databases, time-series databases, and spatial and GIS (Geographical Information systems) databases. Even though in-memory databases are quite fast they are not durable, and they might be subject to data loss. The spatial databases are useful when data has geospatial attributes, but at the same time, it is hard to query upon [39]. Also, it requires good visualization to interpret the

data patterns. Streaming data from SCADA and oscillography data are usually stored in time-series databases.

2.2.3 Software Technologies

The evolution of big data technologies started way early in the 1990s. A boost to big data technology started with Hadoop in 2011 and it has been an open-source platform. Big data technologies have evolved in the past decade performing batch processing at one stage to real-time processing later. In [40], Sebnemet al. has explained the evolution of big data technologies starting with Google File system performing batch processing (2003) to Google Data Flow and Apache spark (2003) performing real-time analytical streams processing. Different software applications were released in the market and many were open-source, and these handled the high data volumes and high speeds while decreasing the latency of processing. One of the most widely used state-of-the-art lambda architecture has been discussed in the section below along with the system requirements to handle the software technologies:

2.2.4 Architectures

This section outlines the architectures utilized in the literature and the current streaming systems have been discussed.

2.2.4.1 Lambda Architecture

The advantages of data systems built with the assistance of lambda architecture go beyond just scaling and support real-time and batch processing on the distributed data. In support, the architecture will not just be capable of handling the data only but will also

be able to accumulate more data to interpret information from it. Increasing the number of data types and volumes stored will result in further opportunities to mine the data including, predicting performance, avoiding more than one version of a schema to be operative at the same time, and building new applications. Lambda architecture (Figure 2), a unique software design, is adopted to overcome the need to process two different systems considering batch processing and stream processing [41]. The batch layer is implemented using Hadoop well known open-source platform for batch processing.

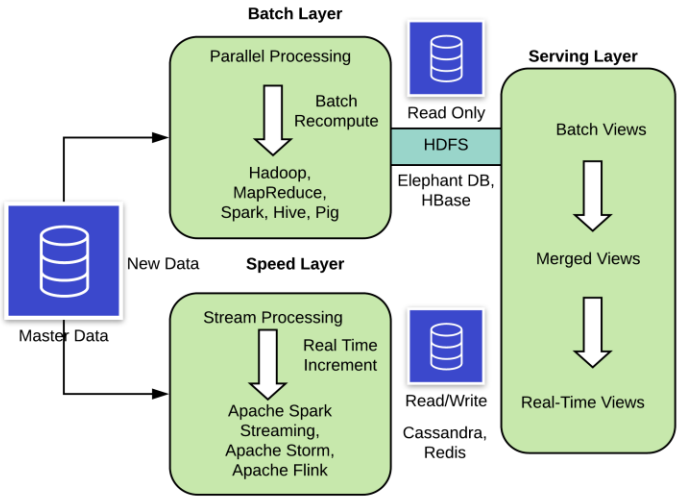

Figure 2 Lambda Architecture

Hadoop discussed in section 2.2.1.1 can handle the data at rest with the help of Hadoop's MapReduce functionality. The data received would be pulled into HDFS and MapReduce jobs are executed using Pig, Hive, etc. As all the data would be in HDFS, there will be a full view of the data available to process it. Streaming analytics engines such as spark and Flink will assist to perform processing and analytics on incomplete data or when data is being updated [42]. This engine's process ables the data as it comes

in and does it a lot faster. These help in processing the data even before the data is transferred to HDFS.

A portion of the data that is collected is analyzed instantaneously as and when the data is generated, and the rest of the data is stored for batch processing. Table 2 refers to some current systems in the field of stream analytics. Analyzing the data as it is available from the source to the memory of a distributed platform needs stream mining systems. If working with stream-only frameworks is desired, then apache storm [42] is one of the best-suited frameworks as it offers a great range of language support, but at the same time, it cannot guarantee to order in its default configuration. The best fit always relies on the data being analyzed, the required latency, and the application required. The three layers of Lambda Architecture are:

Table 2 Stream mining systems

Current Systems	**Year**
R's stream package (clustering only) [43]	2017
streamDM (github) [44]	2016
Moa.cs.woikato.ac.nz (Massive Online Analysis) [45]	2014
Samoa-project.net [46]	2014
lambda-architecture.net [41]	2013
Spark.apache.org/streaming [3]	2012
Rapid Miner stream plugin [47]	2012
Apache Samza [48]	2012
Apache Storm [49]	2011

- Batch layer: stores all the data as ‘master data’, manages it, and precomputes batch views.
- Speed Layer: processes the incoming streaming data as per user-defined requirements and increments the real-time views.
- Serving Layer: a linearly scalable data management system on top of the batch layer and speed layer exposing queried views by the user.

2.2.5 Systems benchmarking

Big data in the smart grid sector involves not only data at rest but also real-time data. Owing to the data being real-time and continuous, additional resources and high computational speeds are required.

As discussed earlier, the use of cloud computing helps electrical companies to reduce cost and power requirements. Table 3 shows the minimum requirements needed to install the platforms Hadoop, Strom, Spark, and Flink and work with the big data frameworks [50]. A minimum of 8 GB RAM is required to have any of the mentioned software technologies to be installed. A supercomputer will help the processes to run faster as it consists of vast computational capability and high-speed interconnect between multiple nodes.

2.2.6 Data Indexing

Indexing plays an important role when it comes to big data management. The speed of data retrieval from a database system is vital for efficient data access. Time-series data is one of the massive types of smart girds. An index format is chosen based on the type of

storage system. Table 4 shows a summary of advanced data indexing techniques that exhibit comprehensive distributed functionality. As the paper suggests the utilization of a distributed framework, the section focuses on distributed data indexing techniques.

Table 3 Big data framework hardware requirements

Framework	**Hadoop**	**Storm**	**Spark**	**Flink**
RAM(Min)	64 GB	64 GB	64 GB	64 GB
CPU (at least)	2	8	8	8
Hard Disk (for each 1TB at least)- Disks per node	12-24	6	4-8	12-24
Operating Sytems	64 bit: SUSELinux Enter-priseServer	CentOS, Red HatEnterprise Linux, Windows	WindowsXP/7/8, Windows (Cygwin), Linux, MacOSX, CentOS, Linux	Linux

2.3 Smart Grid Data

An automated big data management pipeline for a smart grid must have the following qualities:

- The platform should be able to support the acquisition of dynamic data at variable rates and high volumes.
- The platform should be adaptive to the operational needs of current data sources.

Table 4 Distributed data indexing techniques

Indexing	**Year**	**Property**	**Underlying storage system**
FITing-Tree	2019	A data-aware index structure that captures data trends and fits an index to a dataset with the help of piecewise linear functions.	-
Parallel B+ trees [52]	2019	Tree-based: maximizing terminal nodes and minimizing the height of a B+ tree	Hadoop
FastPM [53]	2018	Extends k-d tree indexing to a distributed framework	
IndexedHBase [54]	2014	Historical and streaming data scalable indexing	HBase
E3 [55]	2013	Avoiding irrelevant data splits accesses	Hadoop
HIndex [56]	2013	Secondary Index (server side)	HBase
HAIL [57]	2012	Less index creation cost	Hadoop

Table 4 Continued

Indexing	**Year**	**Property**	**Underlying storage system**
MD-HBase [58]	2011	Quad-Tree and K-d based multi-dimensional index	HBase
Trojan Index [59]	2010	Created at data load time and at query time no penalty	Hadoop

The data sources in the smart grid fall under four categories i.e. historical (archived), real-time, multimedia, and time series [4]. Data sources from SCADA, PMUs, Automated Metering Infrastructure (AMIs), smart meter, Digital Fault Recorders (DFRs), Digital Protective Relays (DPRs), Intelligent Electronic Device (IEDs), Asset management, operational and weather are real-time data sources. The real-time data flows in high volumes and the data is either collected at once or streamed in chunks continually. For instance, standard SCADA polls every 4 seconds. PMU, weather or lightning, and GIS are mostly historically based. The data is usually available in bursts from devices in the grid or as files stored in any of the storage devices and this data can be captured when there is a triggered event. On-demand, this data is transferred by the utility for different kinds of analyses. Data in the form of text, voice, and video (e.g., video surveillance cameras) are multimedia and PMU data is time series. Most often event messages are generated in response to any unusual physical events. These

responses might be in the form of commands communicated to the grid devices by grid-control systems, e.g., an asynchronous business process such as meter ping [60].

Big data management deals with finding the hidden patterns in the data to get meaningful information as an output. As the data grows in volume, variety, and velocity, it tends to be multi-dimensional. To handle big data with multiple dimensions, Random Matrix Theory (RMT) is particularly useful [51]. The most fundamental concepts of dealing with big data account for the representation and modeling of big data.

The random matrices are natural building blocks in modeling big data [4]. The non-asymptotic theory is a unified treatment to a lot of big data problems, which was proposed to model the datasets as large random matrices in 2010 [61]. A single dataset can be expressed as an $m \times n$ matrix given by equation (1)

$$X = U \wedge V \tag{1}$$

$U(m \times n)$ - Orthonormal rows matrix

$\wedge (n \times n)$ - Diagonal matrix with real and non-negative entries

$V(n \times n)$ - Unitary matrix

Where XX^H and X^HX are Hermitian matrices with diagonal entries of $\wedge 2$ correspondings to the eigenvalues. U and V correspond to the eigenvectors. When it comes to large random matrices $m \to \infty$, $n \to \infty$, both Hermitian and Non-Hermitian are utilized in various applications based on the variety of data [62]. Some of the differences between Hermitian and non-Hermitian matrices have been stated in Table 5.

In a high dimensional setting, it is often desired to cut down the dimension of the matrix by working on a low-rank matrix approximation and often require solving for

eigenvalues. The most prevalent methods are Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). PCA is one of the most widely used dimensionality reduction techniques [63].

Table 5 Dealing with big data matrices in smart grid

Operations	**Hermitian matrices**	**Non-Hermitian matrices**
Diagonalization	$XV = V\wedge$	$XX_R = X_R\wedge$ and $X_L X = \wedge X_L$ X_R right-hand eigenvectors X_L left-hand eigenvectors
Eigenvalues	Real	Real or complex conjugate pairs
Eigenvectors	Orthonormal	Not orthonormal

It is used to reduce the number of features in the data. It selects the features which have the most variance in the data and neglects the features that have the least information in the data. We can explicitly specify the number of principal components or features that we wish to consider. The reduction in the features decreases the training and the testing time to a great extent and this knowledge can help in the reduction of data that is to be managed. A centralized process flow has been proposed in this work to manage the data in the smart grids.

CHAPTER III*[3, 4]

BIG DATA PLATFORMS

The computational capacities utilized in the experimental work consist of three platforms.

1. Texas A&M University (TAMU) High-Performance Research Computing (HPRC) Ada, Terra clusters
2. Microsoft Azure cloud computing resource
3. Texas A&M University at Qatar (TAMU-Q) research raad2 cluster with 6 compute nodes (1 master node and 5 data nodes) with Spark on top of Hadoop.

Each of the three platforms' hardware requirements is described in detail in the following sections.

3.1 High-performance research computing platform

The Texas A and M University supercomputer has been used during the process of performing the experiments and the specifications of the supercomputer are outlined below.

TAMU Supercomputer:

Processor - Intel Xeon E5-2680 v4 2.40GHz

Operating System - Linux (CentOS 7)

---

[3] Reprinted with permission from "A Multiprocessing-Based Sensitivity Analysis of Machine Learning Algorithms for Load Forecasting of Electric Power Distribution System," by Ameema Zainab, Dabeeruddin Syed, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat, Mahdi Houchati, Othmane Bouhali and Santiago Bañales Lopez, 2021. IEEE Access, 31684-31694, Copyright 2021 by Ameema Zainab.

[4] Reprinted with permission from "Distributed Tree-Based Machine Learning for Short-Term Load Forecasting With Apache Spark," by “Ameema Zainab, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat and Othmane Bouhali, 2021. IEEE Access, vol. 9, pp. 57372-57384, Copyright 2021 by Ameema Zainab.

Batch Scheduler - SLURM

Frequency - 64 GB DDR4, 2400 MHz

Table 6 Size of the Linux virtual machines in azure

Symbol	Quantity
VM Size	D32s_v3
Offering	Standard
Family	General purpose
vCPUs	32
RAM (GiB)	128
Data disks	32
Max IOPS	51200
Temporary	256
storage (GiB)	yes

Interconnect - Intel Omni-Path Fabric 100 Series switches.

Memory per node utilized - 2560M

3.2 Cloud computing service - Microsoft Azure

vCPUs varying from 8 - 32 are used to run experiments in the azure environment. The specifications are indicated in Table 6.

3.3 Texas A and M Qatar Research computing - Raad2 supercomputer

The apache-spark platform, on which all the distributed computations are performed, consists of one master node and 5 slave nodes as shown in Figure 3. Each of the 5 compute nodes is Linux-based and contains 24 physical CPU cores -- 2 processor sockets with 12

cores per socket -- and 128GB of RAM. The interconnect is comprised of the Cray Aries network, which is employed both for MPI as well as for storage traffic [64]. Hadoop 2.8.0 and spark 3.0.0 are installed on both the master and slave nodes. The load forecasting algorithm is implemented in Python 3.6.4.

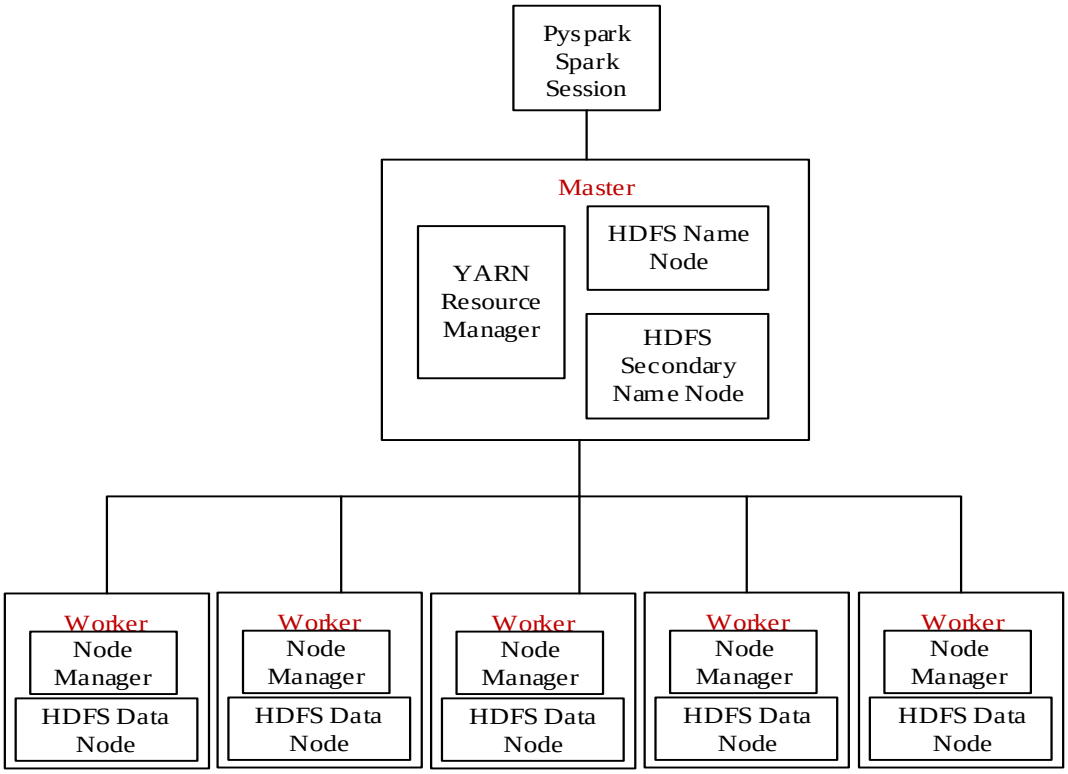

Figure 3 Experimental setup of Spark framework for load forecasting

CHAPTER IV*[5]

MULTIPROCESSING

In this chapter, the choice of multiprocessing method to perform load forecasting is discussed in detail. This computational methodology supports the simultaneous execution of ML models on multiple processors. These methods will be used as a layer in our research in the data processing layer of the proposed multi-layer big data platform proposed in Chapter VI section 6.2 for data processing.

4.1 Introduction

Accurate short-term load forecast with the help of available energy consumption data is essential for demand management and to deliver the required load energy while minimizing the redundancy for power generation and operational costs. A variety of ML approaches have been proposed in the literature for short-term load forecasting such as semi-parametric additive model [65], neural-network-based engine [66], novel radial basis function (RBF) algorithm [67], random forest-based day-ahead load forecast [68] [69], novel wavelet-based ensemble method [70], multi-dimensional XGBoost and fireworks algorithm [71], and average ensembles model [72]. The benefits of load forecasting can not only be utilized by utilities but also by the energy markets. The electricity market's short-term forecasting is realized to improve the management of their resources and to enhance the economics of energy trade. Choosing the best ML model is based on two

[5] Reprinted with permission from "A Multiprocessing-Based Sensitivity Analysis of Machine Learning Algorithms for Load Forecasting of Electric Power Distribution System," by Ameema Zainab, Dabeeruddin Syed, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat, Mahdi Houchati, Othmane Bouhali and Santiago Bañales Lopez, 2021. IEEE Access, 31684-31694, Copyright 2021 by Ameema Zainab.

main criteria, namely, the forecasting accuracy and the convergence rate of the model.

As the smart grid data grows exponentially with the transformation to the smart grid era, a fusion of big data availability becomes a challenge for utilities and other electric sector players. Various research has been concentrated in this area of application. Step-by-step architectural planning for data extraction, data querying, and effective solution for improved accuracy and load modeling was proposed in [73]. Syed et al [74] performed a survey on big data technologies and applications in the field of smart grids. Detection of fault using data-driven approaches utilizing machine learning techniques was proposed in [69]. Gao et al [75], performed load forecasting by proposing a multifactorial framework that compares the effects of various features on forecasting accuracy. This approach is critical when the number of features is increased by adding weather conditions, energy rating of the transformer, etc. Structural dynamics of materials are analyzed with the help of software tools that perform streaming and are anticipated to be time-consuming.

However, one of the biggest persisting challenges in the field of big data is the efficient utilization of the available computational resources while still achieve acceptable results. In this thesis, a methodology that aims at reducing the training time of the ML model whilst enhancing the prediction accuracy is proposed. The load forecasting in the proposed approach is performed with the help of machine learning in absolute run-times of seconds to minutes for an energy data of 3 years for 1,000 transformers. This scale down of training time is intended to offer a significant advantage in load forecasting models with high precision and low execution time.

The accuracy of the proposed approach can also be determined by comparing the

smart meter measurements with the simulated voltages. Additionally, the proposed methodology can be deployed in cloud systems or data centers that are geographically closest to the meters to reduce communication and data transmission delay.

4.2 Related Work

In [76], a moldable parallel task was proposed to perform forecasting, where each sub-task within a parallel task supports a time slice. The authors in [76] focused on reducing the wait time which holds resources until all the tasks are completed. However, the proposed method of multiprocessing in the thesis does not wait for all the next batch of jobs to be submitted but instead picks the next available processor and submits the undone job immediately. However, the method in [14] is applied to a single dataset as compared to multi AMI datasets in the proposed work. Hence to achieve the best performance, workstations with multicore processors are utilized to essentially parallelize the data to achieve fast processing [77]. The spark regression python libraries have been utilized to evaluate the performance of distributed computing in smart meter data management [78][34]. An accurate and fast converging STLF model was proposed in [79] by devising modifications in the artificial neural networks (ANNs) training process to attain a tradeoff between the convergence rate (decrease by 52.38%) and the forecasting accuracy (99.5%). The utilization of spark-based parallel computing has been performed for mid-term load forecasting on historical data of size ~88 million rows to gain an advantage in the calculation time [80]. The published works in [77] – [81] have all focused on processing a unified dataset, however, the proposed approach handles multi-AMI dataset simultaneous load forecast.

4.3 Multi AMI load forecasting algorithm

Load forecasting is performed on a high aggregation level of thousands of transformers or at a low-aggregation level of individual transformers or on a cluster of transformers. Load prediction is performed using machine learning models instead of traditional techniques such as time series, autoregressive models, time of the day model, etc. The problem statement here expects the hourly day-ahead load forecast of individual transformers with a load sampling frequency of 15 mins to be performed. In the STLF scenario of this work, the load of 1,000 distribution transformers needs to be forecasted at the same time. This issue does not involve accumulating a very large dataset but involves distributed large datasets to be worked upon individually. The factors that have to be considered are if the task being performed is heavy and can be a part of the multiprocessing technique or can be a part of a multi-threading technique that involves any form of input-output (IO). The task of fitting the data in memory is challenging for big data. As the models under consideration do not have any bottleneck of I/O being CPU-intensive task multiprocessing will be a logical choice.

Let the data D comprise of $n * d$ matrix, where n is the total number of rows in the data and d is the number of features X in the dataset.

$$D = \begin{pmatrix} & X_1 & X_1 & \dots. & X_d \\ x_1 & x_{11} & x_{12} & \dots. & x_{1d} \\ x_2 & x_{21} & x_{22} & \dots. & x_{2d} \\ \vdots & & & & \vdots \\ x_n & x_{n1} & x_{n1} & & x_{nd} \end{pmatrix} \quad (2)$$

where x_n in (2) represents the n^{th} line of the dataset. Figure 4 indicates the load forecasting being performed on the aggregated data and the time taken to process all the

models sequentially is outlined. The aggregated data is split as per the transformer ID and the models are trained to start with dataset 1 (which is D_1) and then the processing of dataset 2 is initiated. The time taken to train model 1 is given as t_1. Then the data from transformer 2 is trained taking time t_2. Each time t_t also involves the dataset D_t filtered from the aggregated data. The total processing time depicted in Figure 4, is given by equation (3):

$$\Sigma_{totaltime} = t_1 + t_2 + \cdots \cdot t_T \tag{3}$$

where t_t is the time taken to train the ML model on the transformer t, and T is the total number of transformers. Figure 5 shows the use of parallel processing to process each of the x meter ID's data concurrently. The total time of processing all the ML models will change as shown in equation (4).

$$\begin{aligned} &\Sigma_{totaltime} \\ &= \frac{t_1 + t_2 + t_3 + \cdots t_x}{x} \\ &+ \frac{t_{x+1} + t_{x+2} + t_{x+3} + \cdots t_{x+x}}{x} + .. \frac{t_{T-x+1} + t_{T-x+2} + \cdots t_T}{x} \end{aligned} \tag{4}$$

which equates to equation (5)

$$\Sigma_{totaltime} = \frac{t_1 + t_2 + t_3 + \cdots t_T}{x} \tag{5}$$

where x is the number of processors utilized. In this method, x transformers equal to the number of processors are running on multiple processors. Once the x batch is completed, the next x batch is submitted. The time taken for each of the x transformers is reduced from $t_1 + t_2 + \cdots \cdot t_T$ to $\frac{t_1+t_2+t_3+\cdots t_T}{x}$. Submitting the processing

of the transformers as a pool to multiple processors shows the reduction in the time to process the data by ~$\boldsymbol{x}$ times not considering the costs of communication.

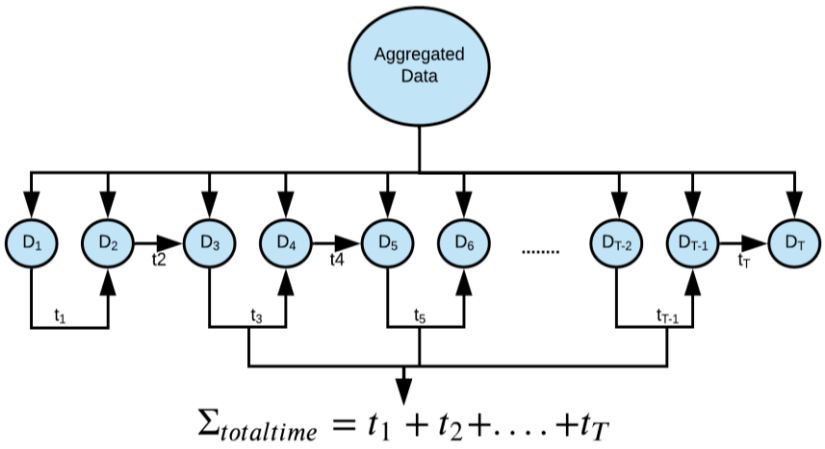

Figure 4 Processing the data without parallel processing

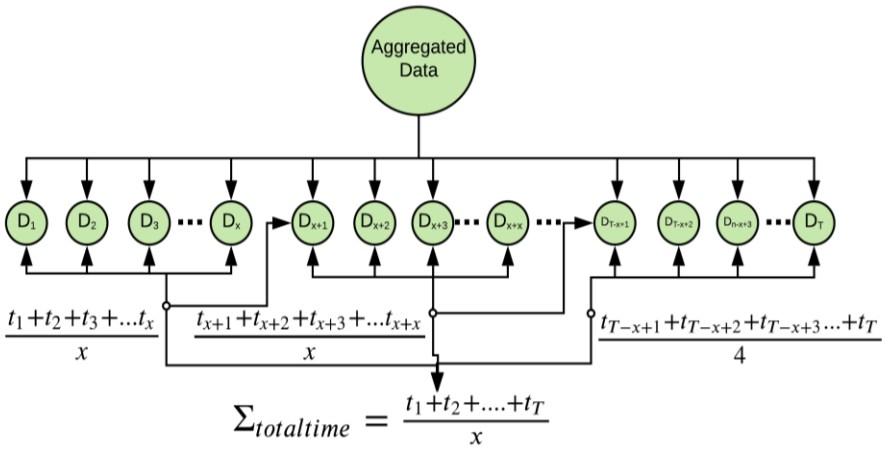

Figure 5 Processing the data with parallel processing

The forecasting strategy is outlined in the form of a block diagram in Figure 6. The strategy can be broadly categorized into 4 stages outlined below :

Stage1 : Collection of complete data D

Stage2 : Filter the data at transformer level resulting in T D_T's.

Stage3 : Parallel processing of each of 1000 D_T's is submitted in batches of x. The total number of batches submitted can be given as $\frac{T}{x} : \frac{T}{x} \in \mathbb{Z}$ and $\frac{T}{x} + 1 : \frac{T}{x} \notin \mathbb{Z}$

Stage4 : Accumulation of forecasting results simultaneously. As the forecasting is a day ahead hourly load forecast, the accumulated results consist of $[Y]_{1000\times24}$ values.

4.4 Parallelization using python multiprocessing

Multiprocessing is a technique of executing tasks in parallel utilizing multi-core or multi-processors in a computing cluster. Python multiprocessing follows a similar approach to spawning the processes over multiple workers. While running the processes in parallel, it needs communication between the processes, and python uses the pickle module for these communications [82]. Distributing the job to parallel machines expects fast

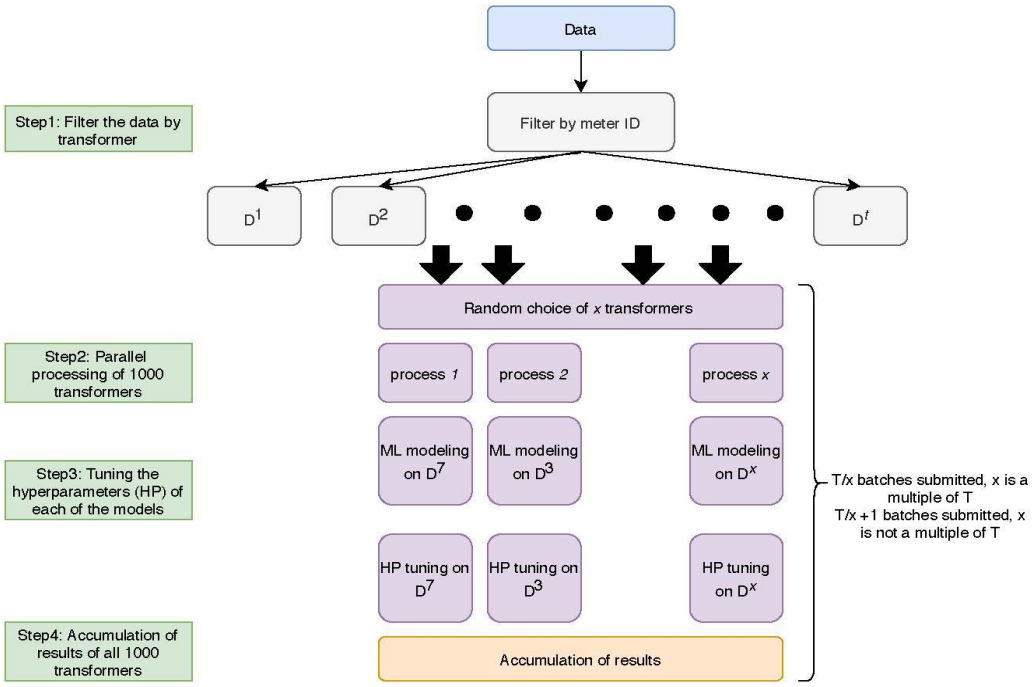

Figure 6 Proposed forecasting algorithm based on multiprocessing. The steps 2 and 3 are repeated for T/x or $T/x + 1$ times to complete the forecasting ofall the 1000 models under consideration.

communication among the machines, which results in high communication costs if data needs to transfer among the processors. The pickling and unpickling usually result in a considerable amount of overhead, hence it is best to keep as few arguments as possible between the processes. Furthermore, the pool class in multiprocessing provides a convenient approach to utilize parallel processing tasks. The worker processes in these processes utilize the pool.map method, which considers only a single iterative argument for processing. Since the computations in the current problem statement do not need any communication between the processes, the order of processes follows the First In First Out (FIFO) queue for gathering the results.

If M_i is the ML model i utilized to train the data D resulting in prediction Y_i, the time taken to train the model is given as $t(Y_i)$ and the error of the prediction variable is given as $RMSE(Y_i)$. The trade-off i value between these two parameters $t(Y_i)$ and $RMSE(Y_i)$ is the value we are trying to attain. To attain a model M_i with the tradeoff between run time and the error, the model M_i falling under the square with Min$\{t(Y_i)\}$, Min$\{RMSE(Y_i)\}$ will be considered as the best model for the analysis (Figure 7).

To utilize the proposed parallel processing strategy in this work to handle the data from multiple transformers and to be able to forecast energy consumption simultaneously, it can be assumed that the matrix D is split into t transformer datasets which can be given in equation (6):

$$D^t = \begin{pmatrix} & X_1 & X_1 & \ldots. & X_d \\ x_1^t & x_{11}^t & x_{12}^t & \ldots. & x_{1d}^t \\ x_{1.}^t & x_{21}^t & x_{22}^t & \ldots. & x_{2d}^t \\ \vdots & & & & \vdots \\ x_n^t & x_{n1}^t & x_{n1}^t & & x_{nd}^t \end{pmatrix} \quad (6)$$

where D^t is the data belonging to each of the transformers. The strategy discussed in section 4.3 is applied to the training of ML models where each of the x D^ts are submitted to x processes to gain advantages in the processing speed. The results are accumulated from all the processes and accumulated to calculate the average $RMSE$.

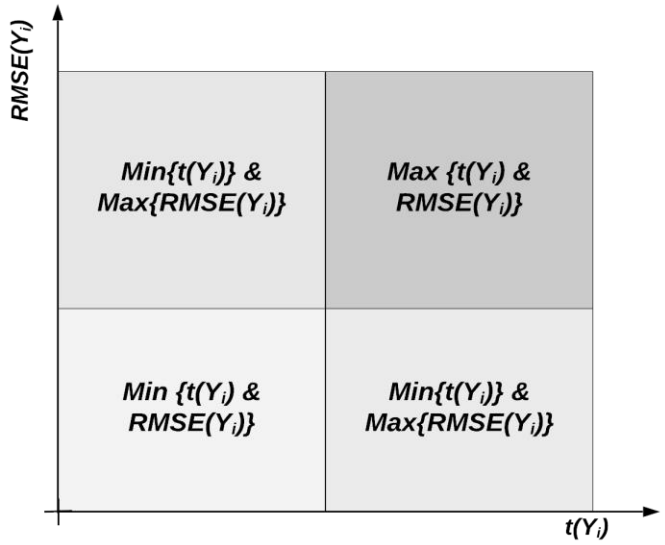

Figure 7 Graph for assessing the training time and root mean square error trade-off

CHAPTER V*[6]

DISTRIBUTED LOAD FORECASTING WITH APACHE SPARK

In this chapter, works on the apache spark platform to perform distributed computing are presented. This work will be used later in the research in the data processing layer of the proposed methodology in chapter VI section 6.3. The distributed computing of apache spark enables the data collection from the distributed storage in Hadoop with the help of HDFS. The performance shows unprecedented improvements in the ETL process and data processing thereby enhancing the forecasting accuracy of load forecasting models.

5.1 Introduction

With the development of the smart infrastructure in the electrical grids, the data collected from various units and locations over time have begun to receive the attention of grid operators and research centers. Data centers usually collect 15-minutes to the one-hour frequency of captured data, which creates enormous amounts of data streams. The power grid operators are looking forward to creating data analytics solutions to benefit from these enormous amounts of collected data. Processing large amounts of data and deriving insights from them will help in the purpose of knowledge discovery and better decision making. Machine learning (ML) techniques help in the decision-making processes and big data provides power in better decision making.

---

[6] Reprinted with permission from "Distributed Tree-Based Machine Learning for Short-Term Load Forecasting With Apache Spark," by “Ameema Zainab, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat and Othmane Bouhali, 2021. IEEE Access, vol. 9, pp. 57372-57384, Copyright 2021 by Ameema Zainab.

To manage big data and perform ML with big data, most of the researchers have been focusing on handling large size of data stored historically in the data centers [83]. To improve the performance of the machine learning algorithms in big data, processing infrastructure and manipulations in terms of the way ML algorithms execute is necessary. Among the ML paradigms in big data, the proposed work focuses on tree-based methods and ensemble learning techniques. Splitting a deluge of data into multiple datasets to perform training with the ML models has gained significant improvement in the learning process in terms of the big data context. For example, the authors in [84] applied ensemble learning to subsamples of big data improving learning accuracy and simultaneously decreased the computation time.

The multi-AMI infrastructure mostly concentrates on forecasting the load of all the distribution transformers (DT's) at the same time. In this methodology, a novel scheduling technique with the help of the Apache spark platform is proposed to short-long term forecast the load of all the one thousand transformers simultaneously. The spark cluster submits big data analytics tasks as spark jobs and the computational resources are allocated optimally to these spark jobs. The amount allocated to these jobs is customizable by the user which affects the Job Completion Time (JCT) significantly. ML algorithms such as Spark Random Forest and Spark Gradient boosted regression trees for training and forecasting the load are utilized. The proposed method performs load forecasting by submitting multiple jobs concurrently on the data sets utilizing the cluster resources optimally.

The main contributions of this distributed load forecasting methodology can be summarized as follows:

1) A scheduling algorithm to perform parallel and distributed execution of load forecast on the smart grid big data is proposed.
2) Testing the proposed methodology on all the one thousand transformers' data without grouping, then make a comparison against the proposed grouping technique.
3) Tuning the ML models to gain high accuracy along with measures to combat overfitting.

5.2 Related Work

Many works have proposed benchmarking results with the use of ML for load forecasting, but in this section, the essence of big data smart grid load forecasting using spark is outlined. The widely installed smart meters collect huge amounts of load data for each of the grid's distribution transformers. Many computing frameworks [85], [86], [87], [88] have been developed for the analysis of big data but, MapReduce [85] is the most famous one because of its features of fault-tolerance, parallel computation, and flexibility. Apache Spark [89] proposed by the Zaharia et al. emerged to overcome the drawbacks in MapReduce. It is an open-source framework and is 100 times faster than Hadoop MapReduce [90]. Spark can execute over several cluster managers such as Hadoop YARN [91], Apache Mesos [92], and spark's standalone scheduler. Spark can also interface with a variety of data storage repositories such as Hadoop Distributed File System (HDFS) [93], Hive [27], Hbase [94], to name a few. However, spark supports distributed computing resulting in a communication overhead increase. Previous research has

observed that by only increasing the computational capability, JCT reduces but then starts increasing in communication overhead [95]. Hence the proposed scheduling algorithm focuses to utilize the available computation capability and still be able to submit multiple jobs with the help of an optimal scheduling algorithm and not losing on communication overhead.

Highly cited algorithms for forecasting smart grid data include linear regression, SVM and its variants [96], and artificial neural networks (ANN). A pooling-based deep recurrent neural network (DRNN) was proposed to learn the spatial information, which outperformed Support Vector Regressor (SVR), Auto-Regressive Integrated Moving Average (ARIMA), and the classical deep recurrent neural network (RNN) [97]. In [98], Happy et al, proposed a statistical approach for load forecasting using quantile regression random forest, risk assessment index, and probability map. In [80], Wei et al performed midterm load forecasting of power supply unit (PSU) considered as a collection of distribution transformers. The authors have utilized a dynamic-based network (DBN), with a peak load of all the distribution transformers within a PSU summed. All of the summed load values are utilized to train and forecast the load using sparks standalone clusters. However, the use of complete data to train instead of summed load values can result in better training accuracies but will require optimized scheduling method which is achieved in this work.

The proposed solution focuses on an hourly day-ahead load forecast with the use of spark ML tree-based algorithms. The models are trained with the spark.ml Application

Programming Interface (API) of spark which is data frame based facilitating ML pipelines and feasible feature transformations [99].

5.3 Load forecasting methodology for optimized computation with apache-spark

5.3.1 Introduction

The Spark ML library supports tree-based models namely spark ml decision trees and ensemble models namely spark ml random forest and spark ml gradient boosted regression trees [100]. Spark session connects to the master node to submit jobs, where each job is split into stages, and stages are further split into tasks. Adding more tasks to a single job if possible is recommended as compared to starting new jobs to avoid start-up costs. In the case of data from multiple transformers, each dataset can be assigned as a job. To reduce the execution time of the load forecasting models, multiple DTs' load forecasting is performed simultaneously with the help of parallel job submission in spark. Moreover, the shortest job submitted may consume fewer resources as compared to the other jobs submitted. To overcome this, python's thread pool concurrency feature in addition to the spark fair scheduler can be used. A solution is to decompose the complete dataset into a cluster of transformers IDs and use multiple computing nodes to train the clustered model with an added sequential step to test the model of each of the transformers within the clusters. However, it is necessary to train clustered models first and then test the individual models within the clusters. This attempts to add multiple layers of parallel processes executed sequentially as iterated in Figure 8. For n clusters, the number of iterations to train the clustered data is n/j, where j is the number of jobs submitted simultaneously. Similarly, for t transformers, the number of iterations to test the holdout data of each of

the transformer (TF) is $n * (t/j)$. As the value of $n * (t/j)$ is larger than n/j in all cases of t, the time in the previous case (with n clusters) is much less than without clustering, provided the data size for each of the jobs in both cases is the same. The proposed parallel and sequential approach of the tree-based ensemble model is deployed on the spark. As

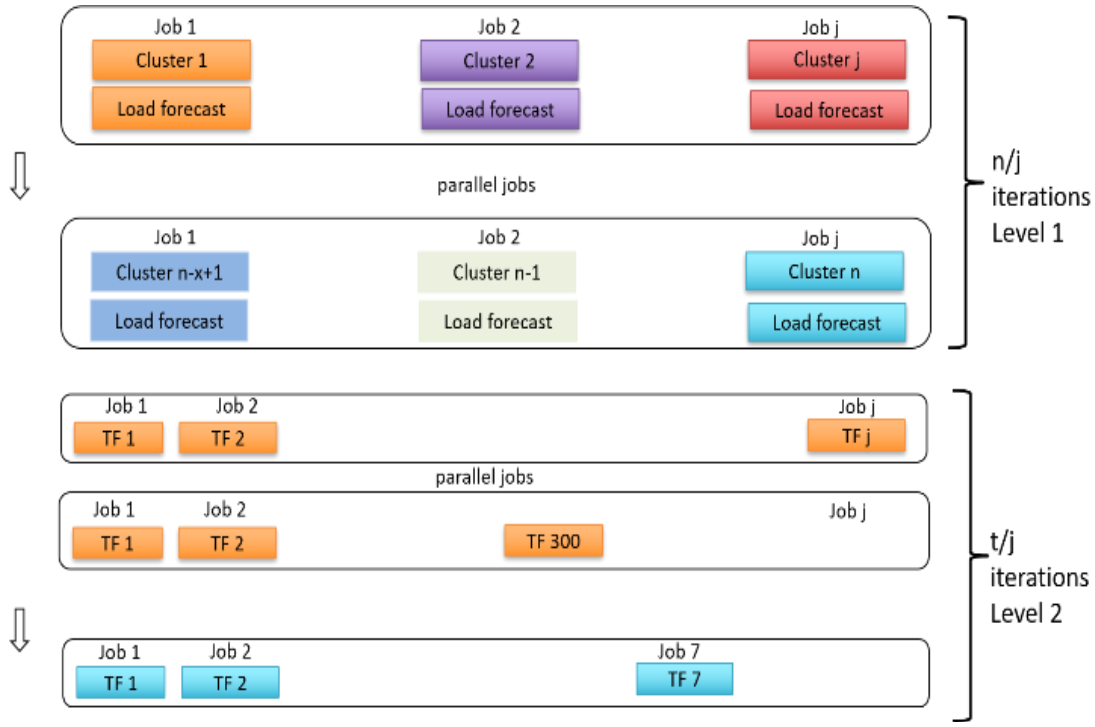

Figure 8 Nested parallelism with spark (sequential and parallel runs)

shown in Figure 3, spark adopts a single master and multiple slave's model. To incorporate the proposed methodology parallelism in datastore and training are discussed further.

5.3.2 Datastore parallelism

The big data of the transformers' load values with the timestamp is stored in the HDFS with replication factor 3. The resulting load data partitions are constructed into RDDs and stored in the corresponding data nodes. The number of partitions is automatically set by

Spark as one partition for a block of file, however, the data is repartitioned to 20 which is equal to the number of cores in each of the nodes using the pyspark programming interface.

5.3.3 Training parallelism

The data from HDFS is read into the spark data frame for the analysis. By using the data frame API only, all the physical execution is compiled in native spark using Java Virtual Machine (JVM), while only the logical plan is constructed in pyspark [101]. The use of data frame API in pyspark results in efficient execution as it avoids the creation of key-value pairs that occur in Scala. Data frames in spark are immutable like RDD and are conceptually similar to a panda's data frame or a relational database. However, the important difference is the execution of transformations and actions in spark. The spark's catalyst optimizer creates an optimized logical plan before sending an instruction to the spark driver. As the catalyst optimizer functions are the same across all the language APIs, data frames provide equivalent performance to all the spark API. Once a logical plan is created, it visualizes it as a Directed Acyclic Graph (DAG) as shown in Figure 9, and is distributed among all the tasks in a job to be able to perform each of the stages concurrently.

Considering the merits of spark, it is used as the big data processing platform in our application for two of the main computing tasks:

1) Average load matrices calculation: the elements of the average load matrix consist of load averaged for 1 lag day, 7 lag days, etc. The data is inputted into the matrix

calculation from the historical data stored in HDFS and the computations are carried out in pyspark.

2) Simultaneous training of DTs' load forecasting models with the help of thread pools in python and multiple jobs in spark utilizing a FAIR scheduler.

5.3.4 Data partitioning in a distributed environment

The RDDs in spark are distributed in partitions and spread across different nodes. Some operations in spark such as rank, count, and window for example can result in serious performance degradation as it results in all the records to shuffle into a single

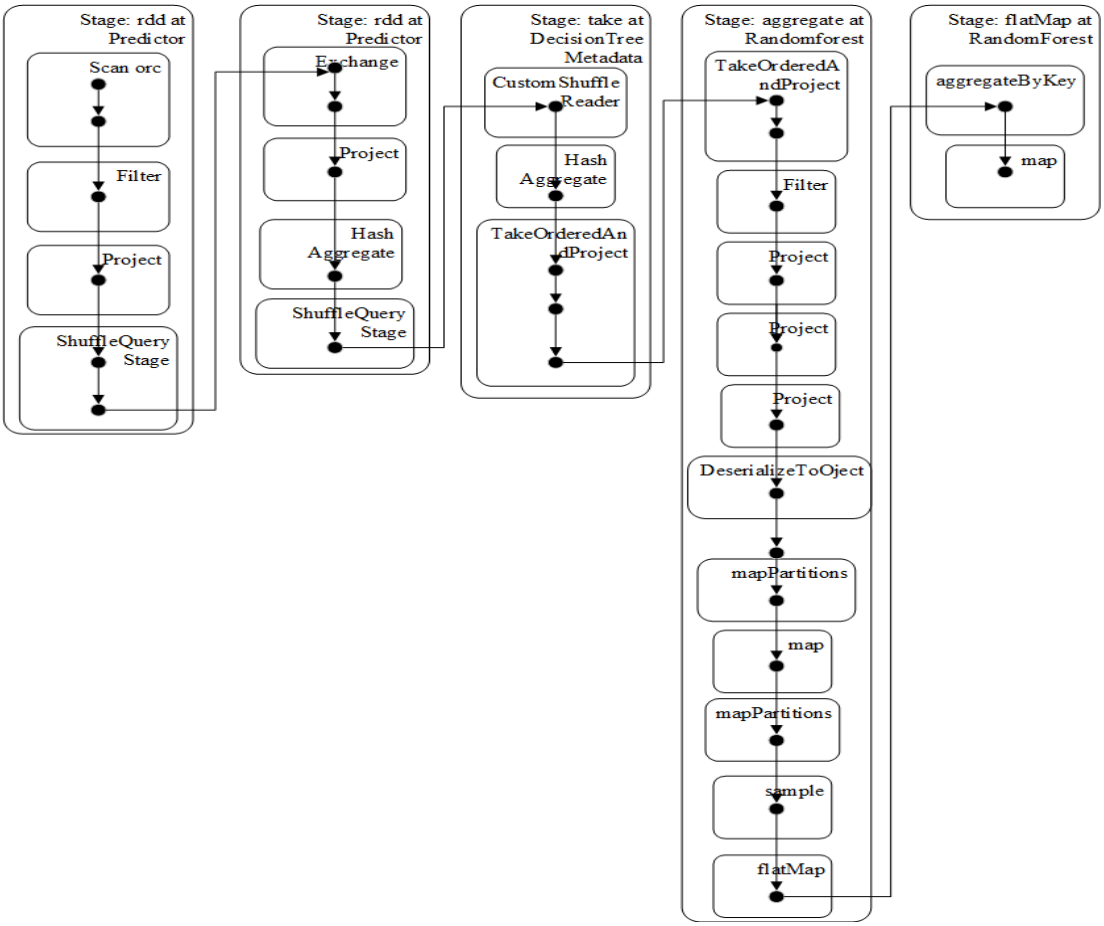

Figure 9 Spark-based DAG visualization for random forest regressor

partition. In this work, a window function has been used to utilize the F.lag function in spark. To overcome the performance degradation in such a scenario, the data has been partitioned by the column month. Partitioning the data by month column has resulted in data being stored into a subdirectory for each partition. Figure 10 illustrates the approach of partitioning the data by column in a distributed environment.

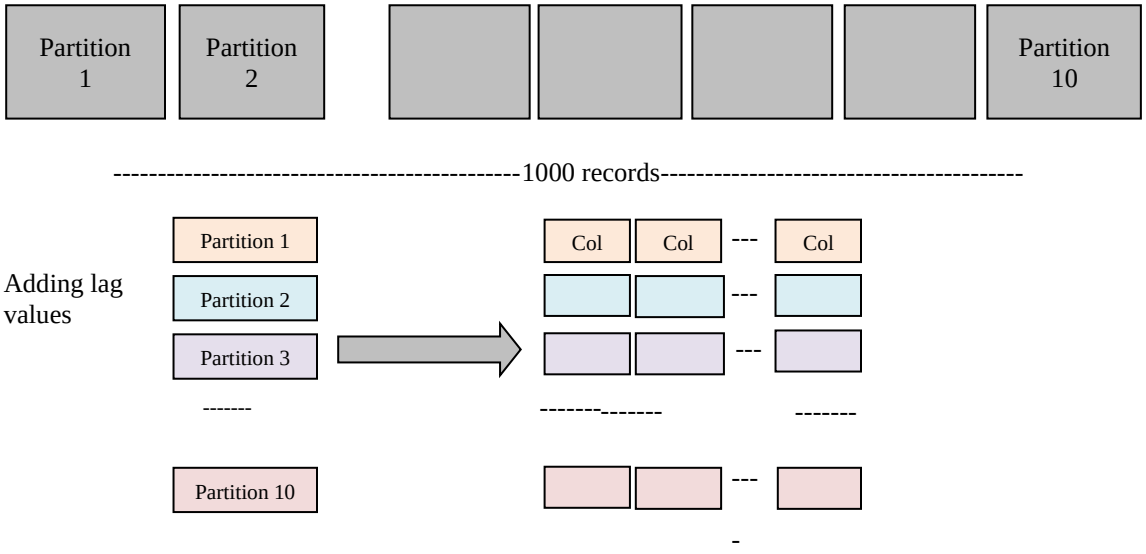

Figure 10 Data partitioning in a distributed environment using window operation on the month column

5.4 Optimal scheduling algorithm

5.4.1 Introduction

Scheduling jobs considering the available resources is challenging. An optimal scheduling algorithm is necessary to schedule jobs to be able to reduce the execution time. As per the requirement of load prediction of multiple transformers at the same time, two scheduling

algorithms are leveraged in this thesis. In this section, the solutions of optimal scheduling when communication costs are ignored and when considered are discussed.

5.4.2 Ignoring communication costs

Considering w workers available and M jobs to be executed, three cases can be obtained where $(w < M)$ & $(wx = M)$, $(w < M)$ & $(wx < M)$, and $w \geq M$ where x is a multiple of M resulting in $wx = M$. The algorithm in this section is structured as follows.

Step 1: Submit the array of tasks to the w workers

Step 2: w jobs are submitted to the available w workers

Step 3: Whenever a processor becomes available, assign it the unexecuted ready job with the highest priority.

Submitting the jobs with the help of a thread pool as discussed in section 5.3, w jobs are submitted at the same time. Considering the three cases, the algorithm flow can be elaborated for three of the cases as below:

Case I:

$pool_{\omega}(train, [1, 2, \ldots . w])$

$(w < M)$ & $(wx = M)$

$T_m^1 \;\; T_m^2 \;\; T_m^3 \;\; T_m^4 \;\; .. \;\; T_m^w$

.

$. \; x \; times$

.

$T_m^1 \;\; T_m^2 \;\; T_m^3 \;\; T_m^4 \;\; .. \;\; T_M^w$

Case II:

$pool_{\omega}(train, [1, 2, \ldots . w])$

$(w < M)$ & $(wx < M)$

$T_m^1 \;\; T_m^2 \;\; T_m^3 \;\; T_m^4 \;\; .. \;\; T_m^w$

.

$. x - 1 \; times$

.

$T_m^1 \;\; T_m^2 \;\; T_m^3 \;\; .. \;\; T_m^{M-(x-1)w}$

Case III:

$pool_{\omega}(train, [1, 2, \ldots . w])$

$(w \geq M)$

$T_m^1 \;\; T_m^2 \;\; T_m^3 \;\; T_m^4 \;\; .. \;\; T_M^w$

where T_m is the time taken for individual job execution and is assumed to be the same. In case III, the computational capabilities are not as high most of the time when the number of jobs to be submitted is in the thousands. The total execution time in all three cases can be summarized in (7) as:

$$T_{totaltime} = \begin{cases} xT & where\ (w < M)\ and\ (wx = M) \\ xT & where\ (w < M)\ and\ \ (wx < M) \\ T & where\ w \geq M \end{cases} \quad (7)$$

Because of the way the number of concurrent jobs is submitted, w workers are assigned for each step of parallel runs. Although at the last step of execution $wx < M$ still takes the same amount of time, as w workers are assigned to perform the job.

5.4.3 Considering communication costs

The main idea of this scheduling task is to augment the scheduling with new precedence relations to be able to compensate for the communication time. By clustering the jobs into C clusters and submitting them to the same worker, the overall communication between clusters will be minimized. If $\tilde{T}$ is the time taken by a cluster including the communication costs, and y is a multiple of the total number of clusters resulting in $wy = C$, $y\tilde{T}$, is the time taken for all the jobs where $y < x$.

5.4.4 Objective function

This section attempts to create the theoretical functions for parallel and sequential training approaches and to propose an implementation solution based on the spark platform. The collected transformers power data is denoted as D where $D^1, D^2, D^3, \ldots D^M$ denote the data for meter m. The data D^m consists of F features namely month, day, year, etc.

Therefore, the chunk of data for a meter ID can be expressed by D^mas in the following equation (8) and (9):

$$D^m = [X_1^m, X_2^m, \ldots\ldots X_F^m] \tag{8}$$

$$D = \bigcup_{m=1}^{M} D^m = \bigcup_{m=1}^{M} \bigcup_{n=1}^{N^m} D_n^m \tag{9}$$

where X_f^m is the feature f of the chunk of the data for a meter ID m; N^m is the size of the m^{th} dataset. This chunk of data is trainable input to the machine learning model. Additionally based on the data decomposition shown in (9), the means square error

(MSE) for regression of the parallel training of the ML model is represented as shown in equation (10) [102] [103]

$$RMSE^{OOB} = \min \frac{1}{N} \sum_{m=1}^{M} J^m$$

$$= \min \frac{1}{N} \sum_{m=1}^{M} \sum_{n=1}^{N^m} J_n^m \tag{10}$$

And the loss function J_n^m of the sample n in data with subset m is given by (11)

$$J_n^m = \sqrt{\frac{1}{N \sum_{n=1}^{N} \| y_n^m - \hat{y}_n^{OOB}(X_n^m) \|^2}} \tag{11}$$

where J^m in (12) is the loss function of the m^{th} data set

$$\mathrm{J^m} = \sum_{\mathrm{n=1}}^{\mathrm{N^m}} \mathrm{J_n^m} \tag{12}$$

y_n^m and $\hat{y}_n^m$ are the observed and the predicted load values, respectively, of sample n in data subset m; and N is the dimension of each of the output samples. The ML model training is performed to minimize the $RMSE^{OOB}$ in (10) and obtain the trees using the dataset D. Similar procedures are performed for the subset dataset D^m concerning the data subset m for transformer level load forecasting.

CHAPTER VI*[7, 8]

PROPOSED METHODOLOGY

This chapter presents the complete big data management platform proposed in this thesis for managing the data from the smart meters to perform short-term load forecasting in large electrical networks. The main aim of the work is to analyze big data and perform load forecasting with high accuracy for electrical energy forecasts. One day ahead hourly energy forecasts are the goal of the work.

A detailed description of the various steps involved in the proposed methodology is described in subsequent sections starting with the description of acquisition data sets used for the case studies.

6.1 Data

The data was collected from an advanced smart metering infrastructure (AMI) of the Iberdrola network. The collected dataset is the energy consumption data of transformers at the distribution level for a period of 33 months (from January 2017 to September 2019). It consists of the hourly energy consumption of transformers located in different municipalities of Spain. Each data point includes the summertime, meter ID reading timestamp, and consumption. This investigation is focused on forecasting the day-ahead hourly load of each of the DTs.

---

[7] Reprinted with permission from "A Multiprocessing-Based Sensitivity Analysis of Machine Learning Algorithms for Load Forecasting of Electric Power Distribution System," by Ameema Zainab, Dabeeruddin Syed, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat, Mahdi Houchati, Othmane Bouhali and Santiago Bañales Lopez, 2021. IEEE Access, 31684-31694, Copyright 2021 by Ameema Zainab.

[8] Reprinted with permission from "Distributed Tree-Based Machine Learning for Short-Term Load Forecasting With Apache Spark," by “Ameema Zainab, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat and Othmane Bouhali, 2021. IEEE Access, vol. 9, pp. 57372-57384, Copyright 2021 by Ameema Zainab.

6.1.1 Dataset 1

Number of DTs – 1,000

Data points - 24,072,709

Dates - 2017-01-01 11:00:00 to 2019-10-01 09:00:00

Format – CSV, size - 2.85 GB

Format – ORC files, size – 200 MB

6.1.2 Dataset 2

Number of DTs – 10,000

Data points - 206,660,033

Dates - 2017-01-01 11:00:00 to 2019-10-01 09:00:00

Format – CSV, size – 23.7 GB

Figure 11 indicates the load values of the data. It can be noted that for a quantile more than 0.9999 the values of load are more than ~700 kWh. Keeping this in view load values above a quantile of 0.99995 have been equated to 0.9995. Quantile statistics after removal of outliers in indicated in Figure 12.

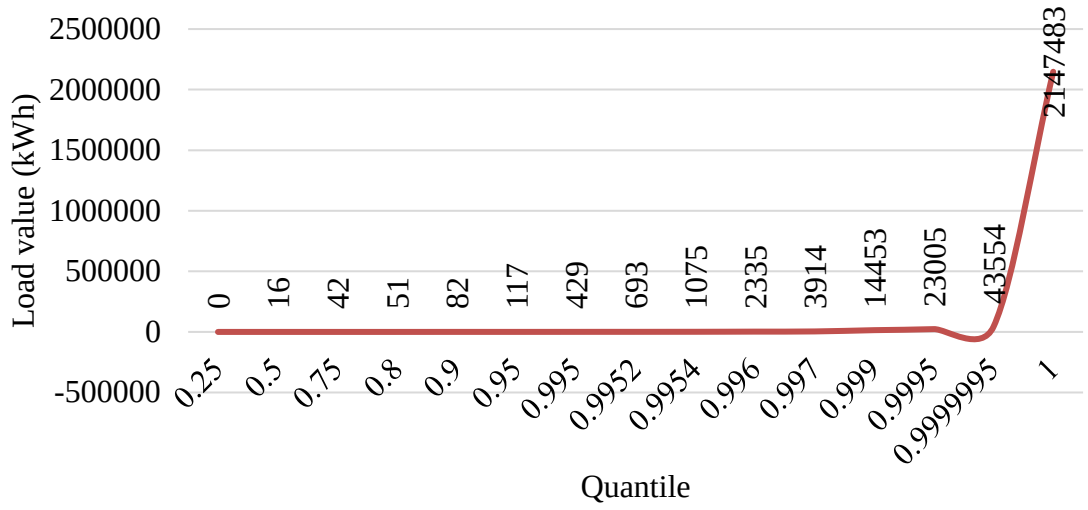

Figure 11 Descriptive quantile statistics with outliers

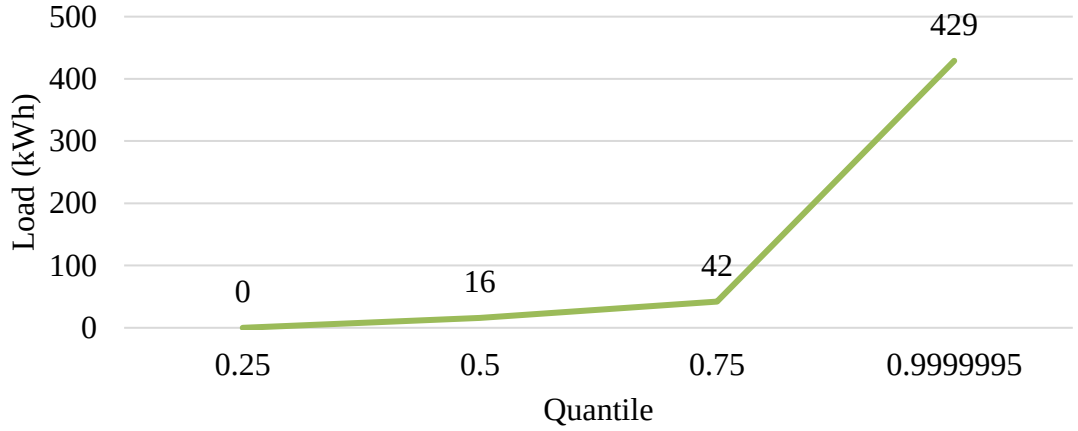

Figure 12 Descriptive quantile statistics without outliers

Transformer rating was also obtained for Dataset2. Out of the rating of the 10k transformer for 354 transformers rating was not obtained. Figure 13 shows the count of transformers on the y-axis and the rating values on the x-axis.

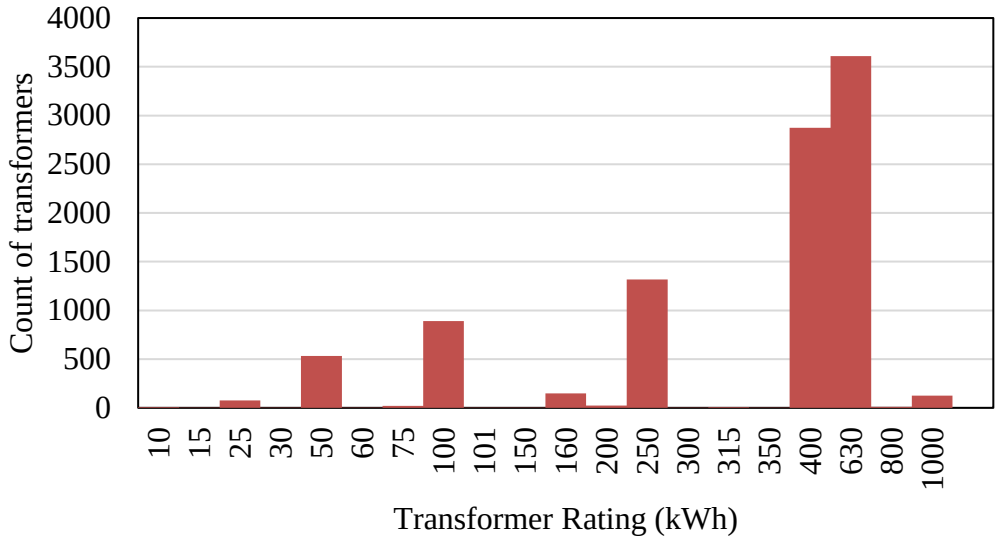

Figure 13 Statistics on transformer rating data

The transformer rating maximum value of 1000 kWh shows that no transformer can have a load value of more than 2000kWh considering the transformer is loaded twice the capacity. Also, 63% of the transformers have a rating ranging between 400 kWh to 630 kWh.

6.1.3 Dataset 3

Number of transformers – 105,148

Data points - 2,166,910,300 (~2.2 billion records)

Dates - 2017-01-01 11:00:00 to 2019-10-01 09:00:00

Format – CSV, size – ~250 GB

Figure 14 indicates the count of the number of transformers against the number of days the data is available for each of the transformers. The x-axis indicates the difference of the end date and the start date for the data available for each of the transformers and the y-axis indicates the count of the transformers. It can be noted from the figure that

although the majority of transformers have complete data for 3 years (=3*365*24 = ~24000 records) there are transformers for which the data is unavailable indicating that either these meters were installed at a later stage or the data has not been collected for these transformers for the complete range of the timeframe.

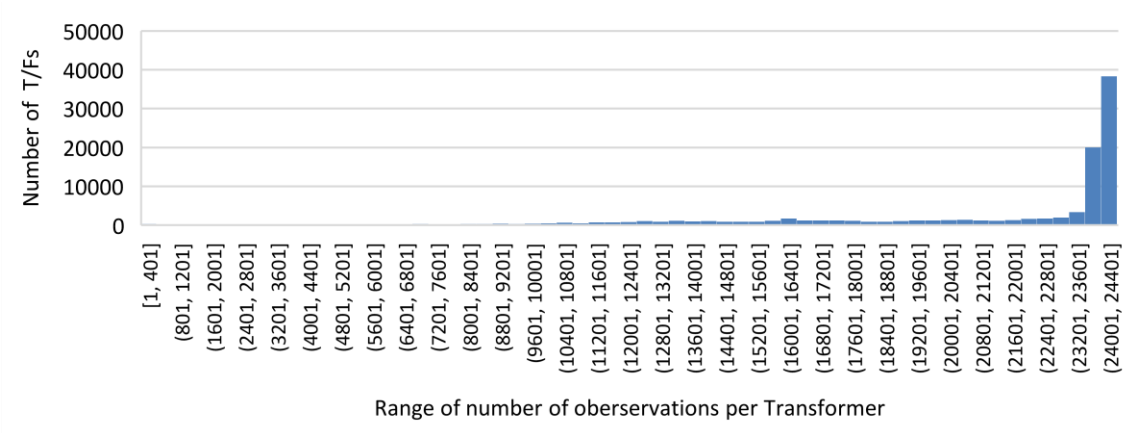

Figure 14 Descriptive quantile statistics of 100k transformers dataset

6.2 Data Statistics and Pre-processing

In the experiments on dataset 1, the data consists of load value and timestamp of 1000 transformers meters of the Iberdrola network [104]. The data is split into 90% (Jan 2017 to Jun 2019) of training and 10% (July 2019 to September 2019) holdout dataset. The total dataset counts to around ~24,000,000. The data was collected from the utility company in an Optimized Row Columnar (ORC) format and was stored in the HDFS storage on 5 data nodes and replicated 3 times. Currently, spark supports timestamp input with the help of flint time series as flint context and not flint session. Because of this limitation, the timestamp is split into the year, month, day, and hour.

Figure 15 shows the power consumption pattern for all the 3 years in the top left, data with large load values on the top right, and the frequency of the load values in the bottom

left and bottom right graphs. It can be noted that the bottom left graph is right-skewed, and after log normalization, the spread of the data is more diverse comparatively but still not normally distributed. The bottom right graph also has log+1 normalization as the data consists of load values of 0. It can be noticed that the data is right-skewed.

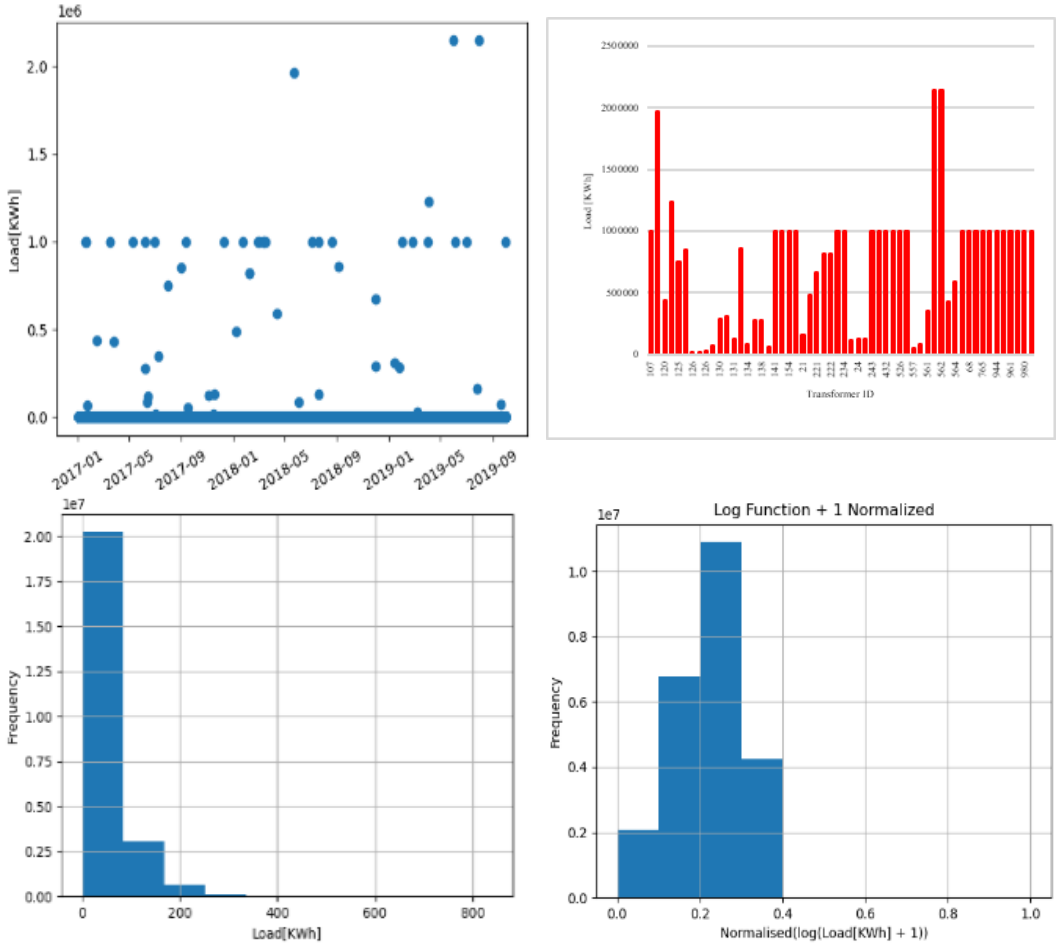

Figure 15 Top left - Load distribution across all the three years (The vertical axis indicates the load value in kWh and the x axis indicates the time stamp). Top right – Data with large load values greater than 1000 kWh (The vertical axis indicates the transformer id the data belongs to and the x axis indicates the load value in kWh. Bottom left – Frequency of the load distribution limiting to 1000 kWh. Bottom right – Frequency of log normalized load plus 1.

6.3 Proposed Methodology

The performance of the forecasting models in terms of execution time is assessed on various big data management proposed methodologies. After the assessment, an end-to-end data management and analytics platform is proposed to perform load forecasting. The tested methods can be categorized into 4 main blocks as depicted in Figure 16. The blocks depicted in Figure 16 are listed below:

- ETL
- The hardware layer
- Data processing layer
- ML modeling

The reason for the choice of each of the components in these four blocks is described in detail below:

6.3.1 ETL

There are various data sources in the smart grid. A few examples are SCADA, Advanced Metering Infrastructure (AMI), smart meters, sensors, PMUs, distributed generation units, weather, customers, etc. The data is collected in the data storage location or streamed through secure channels. In the proposed methodology both Hadoop storage and a physical SATA disk are integrated.

6.3.2 Big data platform layer

In the big data platform layer, multiple compute nodes are added. The current infrastructure discussed in chapter III involves 5 compute nodes with 120GB each. These compute nodes are utilized in a distributed manner to help in the processing of

data. Multiprocessing supports 8 concurrent computation nodes whereas the spark platform supports as many numbers of executors as design in the platform upon the 5 compute nodes. To enhance the computation speed and test the performance of multiprocessing azure cloud infrastructure has also been utilized with 32 simultaneous processes for testing.

6.3.3. Data processing layer

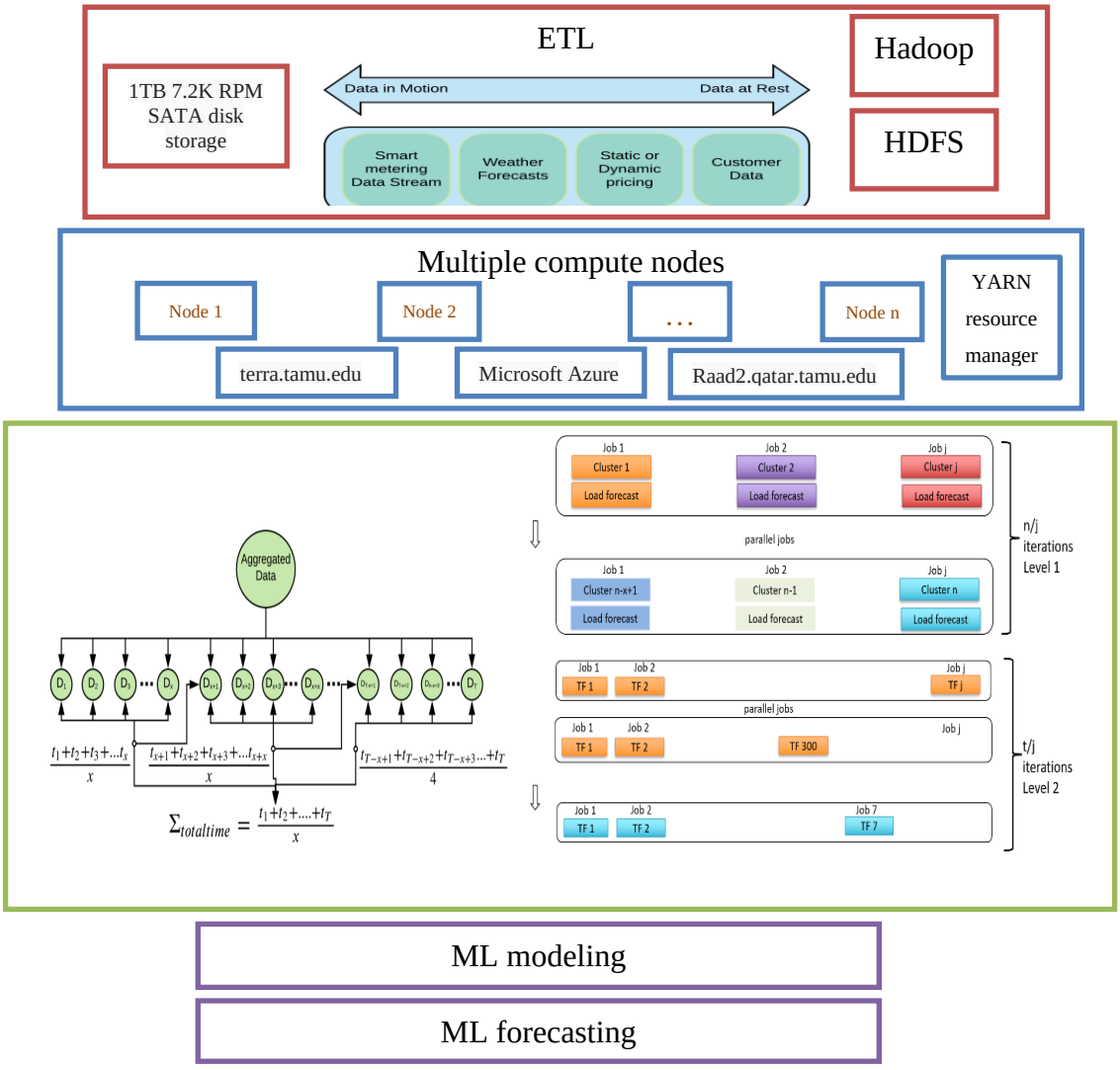

Figure 16 Proposed methodology for big data management in the smart grids

Depending on the data source, the data processing is performed. For processing the data two methodologies have been proposed. With the help of multiprocessing, multiple jobs are submitted in parallel to the processors. The complete process of filtering the data, preprocessing, and ML model is performed on the multiple processors simultaneously and the results are aggregated. This methodology does not involve any communication between the processes and hence there is also no memory share involved. Only messages are passed to each of the processors with the meter IDs. Based on the index of the meter ID the complete processes are initiated and the forecasting result is delivered. In the second methodology of distributed processing apache-spark platform is utilized. Spark is lightning fast and utilizes all the nodes under consideration with the help of a cluster manager. The work has focused upon tuning the spark executors, number of cores per executor, amount of memory per executor, amount of memory in the driver node, scheduler mode, number of shuffle partitions, off-heap memory size, and blocksize compression. The concept of multiple job submissions has been utilized from multiprocessing to be utilized in spark to perform analysis on all the transformers, followed by ML modeling and accumulation of results.

6.3.4 ML Modeling

While performing ML modeling on the transformers data, the choice of ML model has been performed. A sensitivity analysis has been performed by choosing various ML models to observe the performance in terms of both accuracy and execution time. Measures to avoid overfitting have also been considered. Hyperparameter tuning has been performed to choose the ML model with the best parameters. As the job

submission pattern is simultaneous, the accumulation of results is done in such a way that the results are posted online to a single repository.

CHAPTER VII*[9, 10]

RESULTS AND DISCUSSION

7.1 Performance Evaluation Metrics

7.1.1 RMSE, MAPE, and R-squared

The mean absolute percentage error (MAPE) and the root mean square error (RMSE) are chosen as the evaluation criteria. However, MAPE is a widely used error statistic in energy forecasting [105]. Therefore, most of the presented results are compared based on the MAPE values to have a clear comparison with the existing techniques and results. The used MAPE and RMSE are shown in equations (13) and (14):

$$MAPE_t = \frac{1}{n^t} \sum_{t=1}^{n^t} \left| \frac{Y_t - Y_p}{Y_t} \right| \tag{13}$$

$$RMSE_t = \sqrt{\frac{\sum_{t=1}^{n^t} \left(Y_t - Y_p\right)^2}{n^t}} \tag{14}$$

where $RMSE_t$ and $MAPE_t$ are the error metrics belonging to transformer t with data size of n^t. Y_t is the true load value and Y_p is the predicted load value.

[9] Reprinted with permission from "A Multiprocessing-Based Sensitivity Analysis of Machine Learning Algorithms for Load Forecasting of Electric Power Distribution System," by Ameema Zainab, Dabeeruddin Syed, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat, Mahdi Houchati, Othmane Bouhali and Santiago Bañales Lopez, 2021. IEEE Access, 31684-31694, Copyright 2021 by Ameema Zainab.

[10] Reprinted with permission from "Distributed Tree-Based Machine Learning for Short-Term Load Forecasting With Apache Spark," by "Ameema Zainab, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat and Othmane Bouhali, 2021. IEEE Access, vol. 9, pp. 57372-57384, Copyright 2021 by Ameema Zainab.

For non-linear models, the R-squared measure is a choice measure for regression models and is given in the equation. R-squared goodness of fit measure belongs to a class of exponential family and generally leads to a value of [0, 1] [106].

$$R - squared_t = 1 - \frac{\sum_{t=1}^{n^t} \left(Y_t - Y_p\right)^2}{\sum_{t=1}^{n^t} (Y_t - \bar{Y}_t)^2} \tag{15}$$

where $\bar{Y}_t$ is the mean of the true load value for transformer *t*. The R-squared measure is also called the coefficient of multiple determination and is given by the division of regression sum of squares against the total sum of squares (RSS/TSS).

7.1.2 Average RMSE

The objective of future load consumption is to predict the load with high precision and speed to have near real-time processing ability. Root mean square error (RMSE) is used as the error metric because of its wide use. To evaluate the predictive performance, the training dataset is separated from the holdout dataset (data never used for training). All the models are built on the training data and optimized to obtain as low $RMSE_{train}$ as possible and predicted on the holdout dataset to note the $RMSE_{holdout}$. Moreover, to evaluate the performance on all the holdout datasets for different transformers, the average $RMSE$ $(ARMSE)$ is calculated as described in equation (16):

$$ARMSE = \frac{1}{M}\sum_{i=1}^{M} RMSE_{holdout}\ 1 < i < M \quad (16)$$

The ARMSE shows how well the ML model learns the data for all the distribution transformers. The reason for choosing ARMSE is to have high average accuracy across all the distribution transformers and not just one or a few.

7.1.3 Execution time

An important objective of choosing the proposed methodology is to reduce the processing time of the transformer's data. To improve the performance in terms of execution time, total time $T_{totaltime}$ is first measured by submitting individual jobs and $\tilde{T}_{totaltime}$ by considering a cluster of jobs. The time is compared in both cases to choose the methodology with the lowest execution time and still retain the skewed distribution of the multiple meters data.

7.1.4 Spark optimization

Besides spark being an in-memory computing framework, it runs on top of the Java Virtual machines (JVMs). Hence tuning the JVM parameters is necessary to improve the performance of the spark. In this work, three key spark parameters that impact the utilization of resources to reduce the workload execution time have been identified. The work is focused on parameters that impact the memory serialization, data compression, caching, and repartitioning of data. Experiments are conducted considering: i) various combinations of several executors, ii) the number of cores per executor, and iii) the amount of memory for each of the executors. If CO is the total number of cores in the configuration as shown in equation (17)

$$CO = E * COperE \quad (17)$$

where E is the total number of executors assigned and $COperE$ is the number of cores assigned per executor in the spark configuration. The distribution of total memory in the spark configuration is given in equation (18).

$$MEM = (MEMperE * E) + (0.1 * MEMperE) \quad (18)$$

where $MEMperE$ is the memory assigned per executor. The second term in (18) is the overhead memory allocated to each of the executors which accounts for virtual machine overheads or other native overheads. Further, the $MEMperE$ is divided into two fractions, one for memory and the other for storage. The memory fraction handles the data structures, out-of-memory error and the storage fraction handles the cached blocks of data. The values of CO and MEM can vary and are very specific to the cluster used for configuring spark. Choosing a larger value of E results in reducing the $COperE$ to balance the CO. Similarly, choosing a larger value of E reduces the $MEMperE$ to balance the

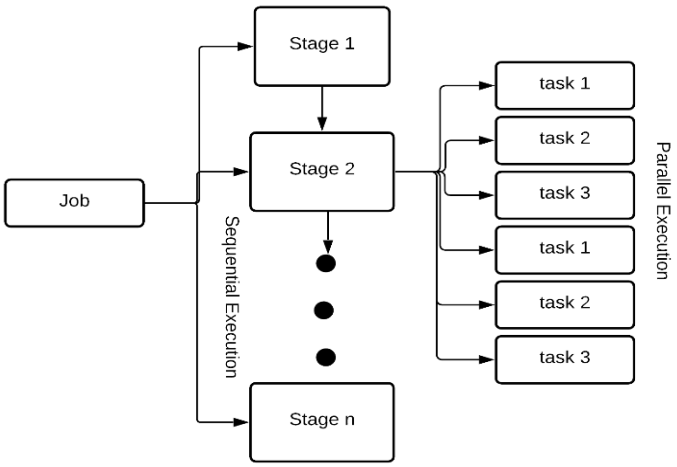

Figure 17 Spark job anatomy single job

MEM.

The topology of a single job function of spark is described in Figure 17. A job connects with the HDFS to read the data and perform computations on it by partitioning it into stages. All the stages are executed sequentially one after the other. Each stage has multiple tasks which are performed by multiple executors. These executors consist of multiple cores depending on the optimization performed on sparks topology. Each of the tasks is performed on a logically divided partition.

Figure 18, shows the architecture of the design which involves a series of parallel execution followed by sequential execution and finally followed by parallel execution. The proposed methodology utilizes an optimized scheduling strategy to incorporate the parallel execution of multiple jobs [27]. Considering the spark platform, with total cores as CO, the number of cores per executor is given in equation (17) where $\boldsymbol{COperE}$ is denoted as the number of cores per executor. When a single job is submitted in spark, the resources are allocated as per the configuration of the platform.

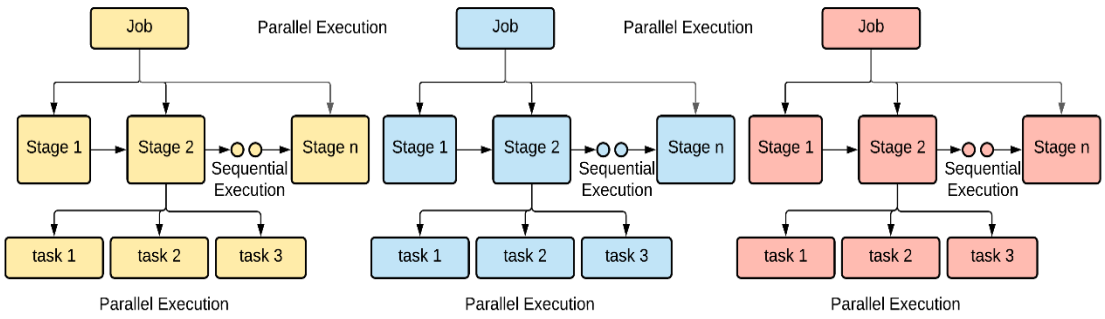

Figure 18 Proposed k-means parallel in-memory clustering component with parallel jobs.

7.2 Experimental Results

The future load can be predicted by determining the relationship between the load and the variables that influence the load i.e. time of the day and weather [107]. The parameters of the model are precalculated to accelerate the forecasting on the larger volumes of the DTs. 90% of the data is taken for training, and 10% for forecasting to determine the least root mean square error (RMSE) and mean absolute percentage error (MAPE).

7.2.1 Multiprocessing Layer

Six machine learning regression models are chosen for comparison, viz, Linear Regression, Support Vector Machine, MLP Regressor, Decision Tree, Random Forest, and Gradient Boosting Regressor. Table 7 lists the average RMSE and MAPE values generated for the hourly day-ahead load forecast of the 1,000 transformers. The table

Table 7 Results of 6 ML models on hourly load [kWh].

Accuracy	DTR	LR	NN	SVR	GBRT	RFR
Average RMSE	252.26	111.27	6527.39	94.54	224.03	194.27
Average RMSE (without outliers)	5.67	5.91	40.78	4.87	4.26	4.02
MAPE (%)	10.91	10.0	553.29	5.42	11.99	10.64
Fit time(s)	781.27	28.63	16121.48	39040.04	5240.23	45570.63
Forecast time(s)	1.957	1.702	5.004	2716.54	6.02	92.73
Number of parameters	12	4	8	11	21	16

The time in sec indicated above is the accumulated time for all the 1000 transformers data which is close to ~24 million rows

contains the training fit time, the forecast time, and the number of parameters. Therefore, this assessment is based on the average of all the 1000 transformers, which helps in portraying the comparison of model performances. The limitation of the approach is that the best model cannot be generalized to determine a tradeoff between accuracy and time. None of the used models exhibit the best accuracies, which is consistent with the fact that the values are an average of all the transformers' results. This inconsistency indicates the need for individual models to be hyperparameter tuned for each of the ML models across all the transformers. The SVR shows the least error followed by decision trees and gradient boosted random forest. Even though linear regression shows fewer error rates, LRs cannot be considered as the best model because of the nonlinear patterns in the data. The elimination of random spikes in a few of the transformers results in a more accurate load forecast of the transformer with maximum load. Figure 19 shows the RMSE of the day-ahead hourly forecast for all the 1,000 transformers while removing outliers. Among all the RMSE values obtained in Figure 19, approximately 20 transformers with peak values above 1,000 kWh were excluded. As the results indicate 1,000 transformers, some samples at random have been filtered in Figure 20.

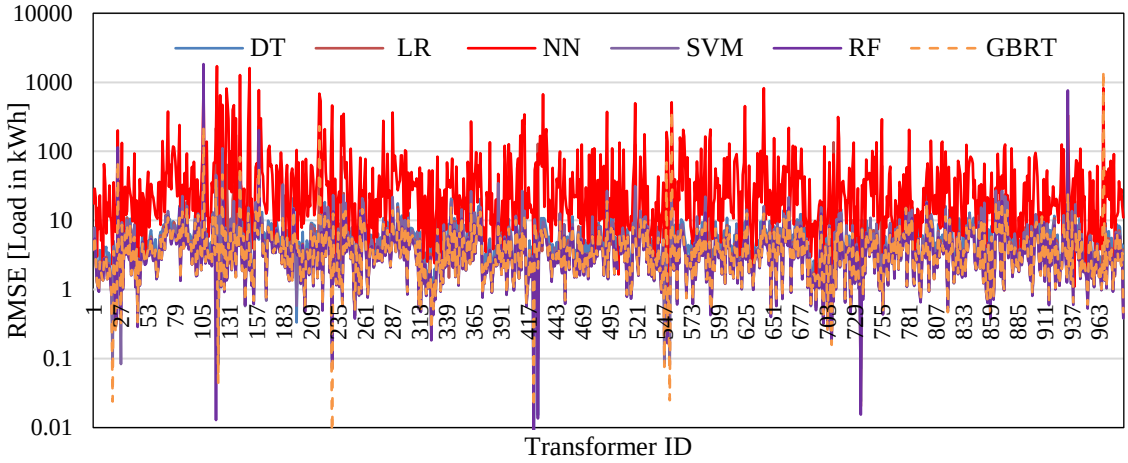

Figure 19 Day ahead hourly load forecast RMSE of 1,000 transformers.

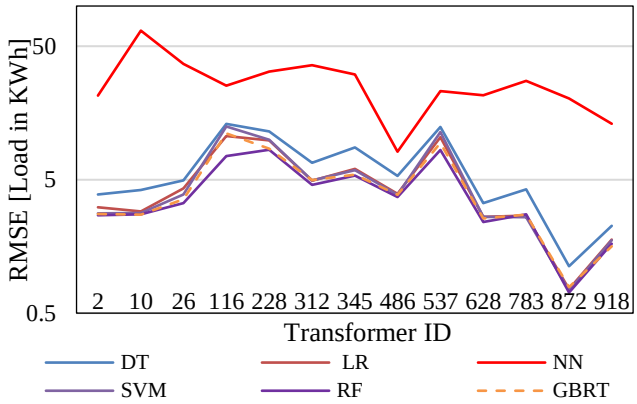

Figure 20 Transformer RMSE for random with proper resolution and good interpretations

7.2.1.1 Data size / ML model / Execution time

To validate the proposed approach and the performance evaluation, the three following measures are considered that impact the big data load forecasting strategy:

Data size – The frequency at which the data is generated affects the size of the data. Bigger data volumes result in larger execution times and more memory requirements.

ML model **–** Better forecasting accuracies can be attained by choosing the best suited ML model. A complex model with multiple optimizing parameters results in higher execution times.

Execution time **–** An important aspect that must be considered while evaluating a prediction model. A shorter execution time is necessary for short-term load forecasting in a real-case scenario.

Short-term load forecasting can be performed only if the big data generated can be processed rapidly, for example within an hour for an hourly load forecast. Most of the research conducted on STLF targets the load prediction within an hour horizon [108]. The computing time should be less than the targeted forecasting timeline. To attain this goal, load forecasting is performed along with parallel computing. The parallel processing model helps in the execution of the ML models parallelly on multiple nodes reducing the run time as compared to the traditional method. Figure 22 presents the comparison results of concurrent computation and non-concurrent computation with dataset 1 for 1,000 transformers for randomly chosen 3 ML models utilizing 20 processors. The measure of computational speedup is calculated in equation (19) based on Amdahl's law [109]:

$$speedup = \frac{run\ time\ of\ the\ standalone\ algorithm\ (sec)}{run\ time\ of\ the\ parallel\ algorithm\ (sec)} \tag{19}$$

The speedup values for three ML models NN (3 layers), DT, and LR are 20.55, 20.95, and 17.30, respectively considering average run times. The average speedup of all the models in a parallel environment as compared to the stand-alone environment is 19.06.

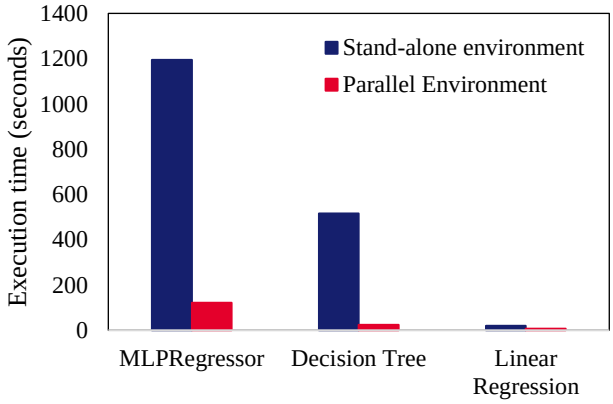

Figure 22 Execution time of 3 different ML models in different environments for ~24 million samples.

Hence, the parallel execution of the model attains a faster processing speed in total for all the models as compared to the series computing environment. As the number of experiment samples increases from 1 million to 64 million, the average execution time of the Spark-DBN model increases from 3.35 to 113.12 seconds, however, the processing speed of 24 million samples to perform linear regression is only 56 seconds

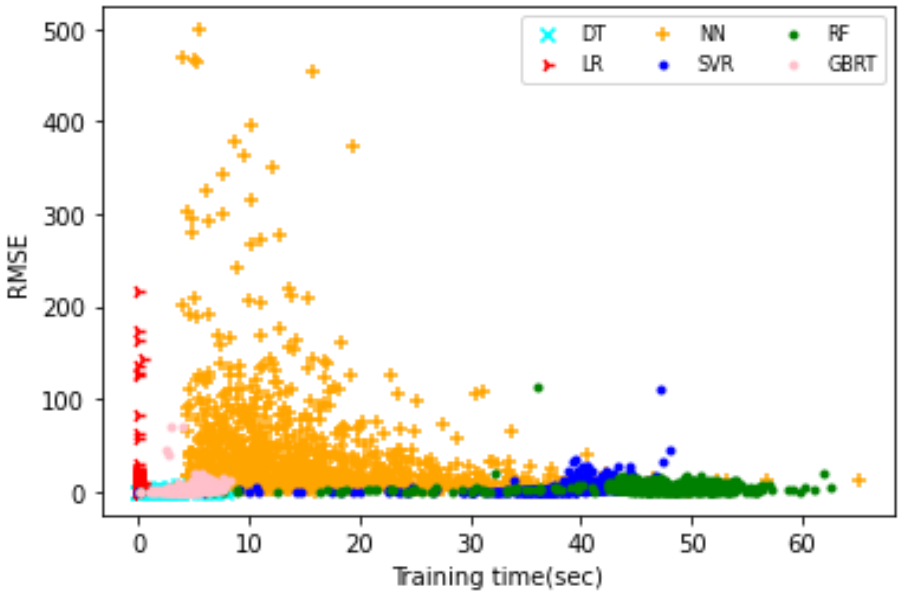

Figure 21 Comparison of the forecast strategy based on RMSE, training time and 6 ML models.

as compared to ~120 seconds in [80]. SVR stopped execution with a kill time error while running in the stand-alone environment. This is because of a system problem on the HPRC due to the processing limit of one hour on the HPRC login node.

7.2.1.2 ML model/Execution time

The experimental results of prediction error in terms of RMSE and the time taken to train the models of ML models under consideration are depicted in Figure 21. As per the proposed methodology of parallelization using python multiprocessing in section 4.4 of chapter IV, Min $\{t(Y_i)\}$, Min$\{RMSE(Y_i)\}$ will be considered as the best model. The neural network model indicated with a plus sign is spread across both in terms of RMSE and training time. Whereas, DT and GBRT and LR fall under the Min$\{t(Y_i)\}$, Min$\{RMSE(Y_i)\}$ bucket. Random forest, an ensemble model indicated in green and support vector regressor indicated in blue, takes more time to execute as compared to Linear Regression indicated in red. RFs and SVRs exhibit lower RMSE but have higher training times. The error distribution for Neural Networks ranges gives an insight into the high forecasting errors but showcase lower training times as compared to SVR.

7.2.1.3 Data Size

Execution times on the data of a different number of DTs for the period data ranging from January 2017 to September 2019 is considered. The numbers of distribution transformers considered in various experiments are 10, 100, 500, and 1,000. When the number of transformers increases from 10 to 1,000, the execution time of the ML models increases. The execution times for all the 6 ML models for various numbers of processors with 8, 12, 16, and 32 are shown with top-left, top-right, bottom-left, and

bottom-right in Figure 23. The execution time on the Y-axis for all the graphs as shown sums the time is taken for transformers' data filtering from the aggregated dataset, model training time, prediction time, and model evaluation time. For all the four cases considering 8, 12, 16 and 32 processors run time is averaged to compare the ML model in terms of run time. The average execution time of linear regression increases from an

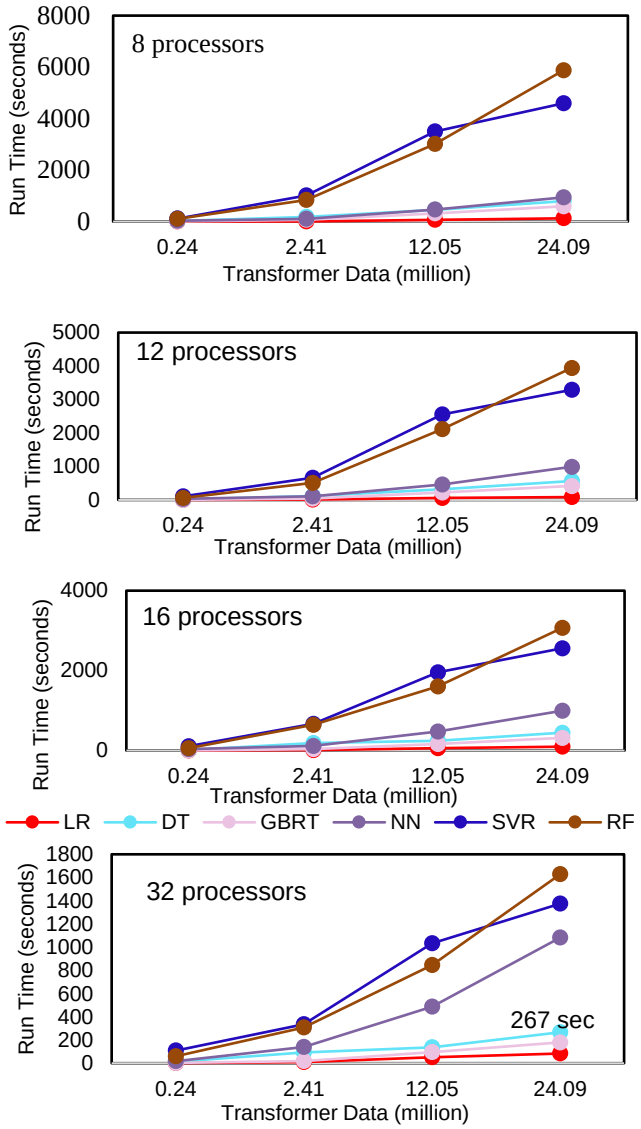

Figure 23 Total execution time of all the 1,000 transformers for different ML models and varying datasets.

average of 1 second to 271 seconds, the average execution time of SVR increases from an average of 110 seconds to 80 minutes, the average execution time of MLPRegressor increases from an average of 30 seconds to 18 minutes, the average execution time of GBRT increases from an average of 4 seconds to seven minutes, the average execution time of random forest increases from an average of 72 seconds to 61 minutes, while the average execution time of DTs increases from an average of 16 seconds to 9 minutes. From Figure 23, it can also be noted that from 8 processors to 32 processors the range of the Y-axis changes from 0-8000 sec to 0-1800 sec. The time required is the shortest for 32 processors in all the experimental trials.

Overall, considering a tradeoff between the accuracy and the execution times of the models, decision trees have outperformed as compared to the other ML models and can complete the training of the ML models for all the 1,000 transformers in nine minutes approximately. Figure 24 exhibits a snapshot of the decision tree with 24 lag hours with a max depth of a tree restricted to 8, and the maximum leaf node to be 50. Similarly, Figure 25, depicts the architecture of the neural network used. The 30 features X are given as input to the neural network to obtain the forecast value Y. The results show an intuitive understanding of the benefits of using decision trees in the scope of big data over other Machine Learning algorithms. Real-time processes of large data streams for utility applications are hence possible with decision trees. The observations confirm the findings that this method is also one of the suitable methods for data streams that can be adapted to fast execution times [110].

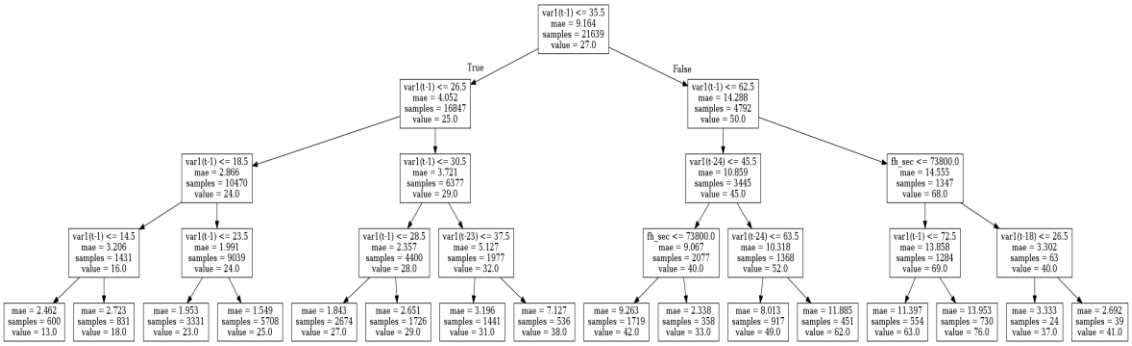

Figure 24 Decision tree for predicting load for one of the transformers. Note var(t-1): 1-hour lag value, var(t-2): past 2nd hour lag value, previous value of load consumption.

The results after comparison in terms of ML model, execution time, and data size indicate that decision trees have outperformed the other models with an execution time of 9 minutes utilizing 32 processors to run all the 1,000 models with 24-hour lag day features added, and a data size of ~24 million records.

7.2.2 Distributed processing with spark executors

In this section, the metrics discussed in the section 7.1.2, 7.1.3 and 7.1.4 are evaluated on the datasets to showcase the benefits of the optimal scheduling algorithm.

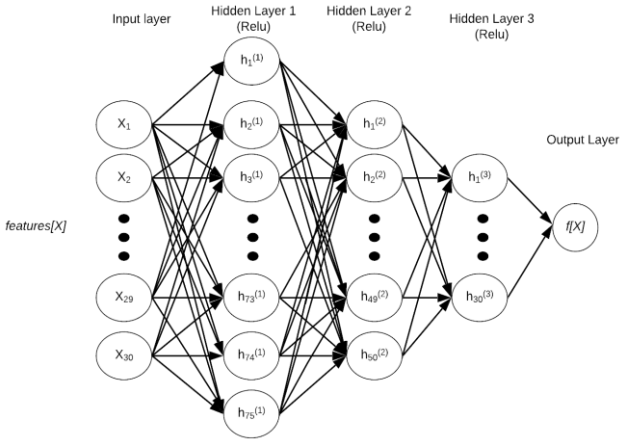

Figure 25 Neural network architecture used to forecast the model. The activation function is chosen as relu and the solver as adam optimizer.

The $ARMSE$ and the execution times are noted under experiments to determine the robustness of load forecasting methodology using spark. All the experiments are conducted on spark 3.0.1 on top of the Hadoop platform with 5 worker nodes and 1 name node each with 120GB of memory and 20 cores. YARN is installed on top of Hadoop as the resource manager and HDFS is used for the distributed storage of data.

7.2.2.1 Validation of execution time

In this experiment, the proposed optimal scheduling method is validated in terms of the training time and the forecasting time. The total time T and $\tilde{T}$ are measured for both cases of x and y number of jobs submitted. The proposed scheme is tested for x using the k-means clustering algorithm to group the data to obtain clusters with higher accuracy. To validate the proposed method, various chunks of y values are considered and compared against the time taken for x number of chunks of data. For the given data as the value of x is 1000, values ranging from 750 to 25 are chosen as shown in Figure 26(a). The speedup is calculated for the various combinations by performing $T/\tilde{T}$. As the value of y increases, the size of the data is distributed among the y chunks which also affects the processing time for different sizes of y. For varying values of y the speedup is increasing, indicating the time $\tilde{T}$ is always less than T for all the values of y. Choosing a lesser value of y and still not losing on speedup is recommended, as in practice it will help in reducing the execution time in cases of performing representative clustering. Hence the proposed optimal scheduling algorithm stated in section iv improves the performance by reducing the time to perform the analysis. Similarly, for a y value of 93, varying values of the thread pool are performed to analyze the speedup as shown in

Figure 26(a). Compared with a single job submission, the calculation time tends to decrease gradually as the number of concurrent jobs submitted increases based on the left axis in Figure 26(b). Massive jobs are distributed across the slave nodes, which reduces the computational load. The spark computing platform captures the intermediates results to memory resulting in the inefficiency of iterative processing where each data frame is called multiple times for various processing stages. As shown in Figure 26(b), based on the right y-axis, the speedup is approximately increasing linearly up to a value close to some cores and makes it less linear after a value of 18-20. When the number of jobs submitted increases above the threshold of a possible number of concurrent threads that can be submitted, the data transfer among the processes increases communication overhead which eventually increases the parallel management overhead. Hence a trend of less linearity can be observed clearly after a value of jobs=18. The training time is 43.8% faster for a pool value of 18 than the traditional method of submitting jobs sequentially in spark.

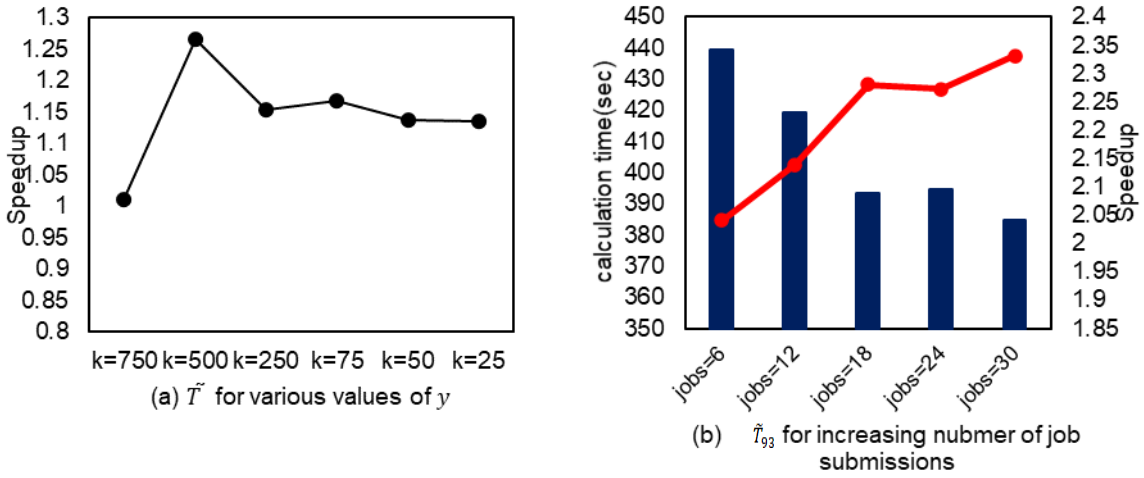

Figure 26 Performance evaluation. (a) shows the speedup for various cluster sizes for a concurrent job submission size of 18 and (b) presents the speedup of

Additionally, the clustering time, training time, and testing time for a value of k = 93 is estimated based on the clustering performed to choose the best cluster number value as shown in Figure 27(a). The total execution time (includes the training time of grouped clusters, testing time of individual transformers with clustering, the training time of the individual transformers, and the testing time for individual transformers) for the 1000 models is shown in Figure 27(b). The time taken by the gradient boosted algorithm is the highest compared to the other spark ml algorithms. Although both random forest and gradient boosted trees are ensemble models, the random forest takes noticeably lesser time as compared to random forest. Inference out of this observation is that gradient boosted is a boosting algorithm that is quite sequential and is intended to take more execution time whereas multiple trees in the random forest can be run parallelly across the nodes to speed up the execution. The times observed in Figure 27(a) show the lowest training time for spark decision tree regressor. It can be noted that the time taken to perform testing is almost close to the training time. This is evident from the proposed

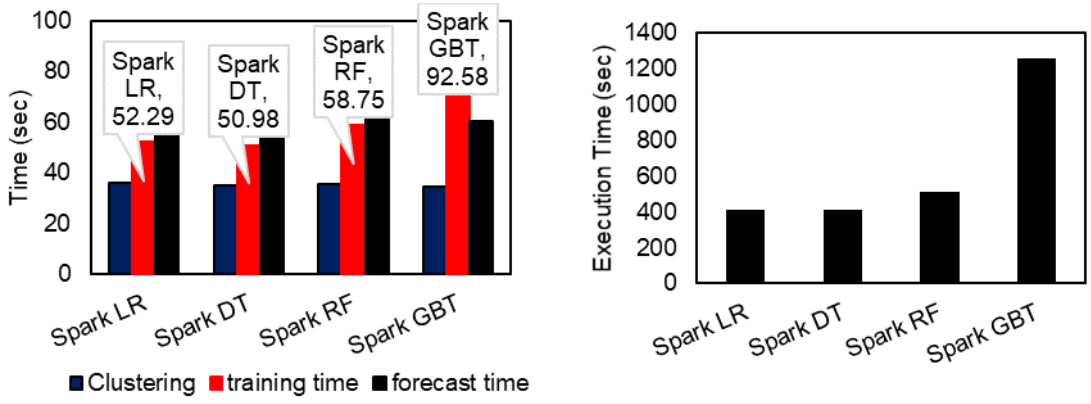

Figure 27 Comparison of compute time at various stages of load forecasting. (a) Results obtained for the time taken to perform clustering, training time and testing time on the holdout dataset for SLR(spark LR), SDT, SRF and SGBT. (b) The execution time involves

methodology which states that performing analysis on grouped data is preferred over individual transformers data. However, as the testing has to be performed on all the DT's datasets, grouping cannot be performed to reduce execution time.

7.2.2.2 Validation of spark optimization

To validate the use of an optimal number of $COperE$, experiments are conducted based on various combinations of $COperE$ and E which in turn affects the $MEMperE$. Figure 28 displays the comparison of run-time for various combinations of executors and cores per executor. The combination with the largest number of cores per executor shows the lowest run time as per the secondary y-axis in Figure 28. As the job submission computes multiple jobs at the same time more number of workers helps in the distribution of the jobs to more number of workers. Hence a choice of 5 executors and 20 cores per executor is decided as an optimized combination of the spark configuration. It is worth mentioning that as the number of executors is increased, the $MEMperE$ is reduced as it is distributed among the executors, to sum up to MEM. Other than time, communication overhead and

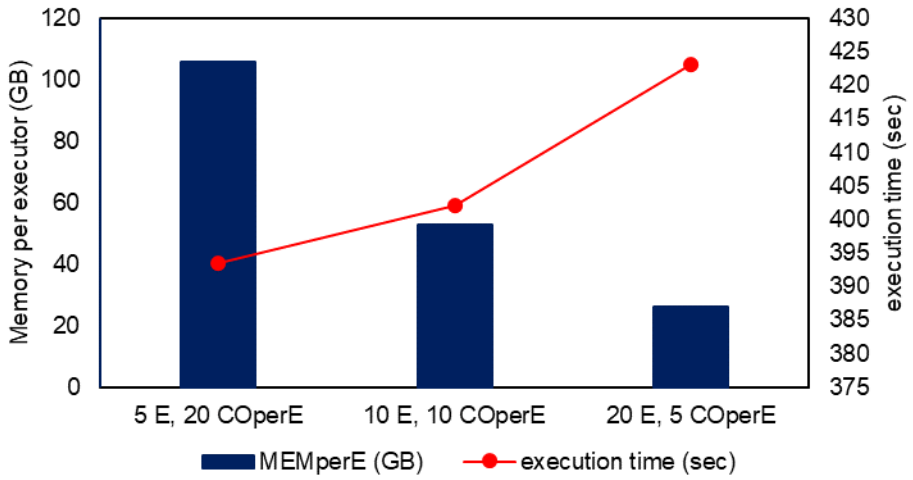

Figure 28 Run time comparison for various spark optimization parameters

data transfer is also a concern in distributed computing. Other than time, communication overhead and data transfer are also a concern in distributed computing. By increasing the depth of a decision tree (refer to Table 8), it is noticed that as the tree grows larger, after a max depth of 10, a large task transfer warning is shown by spark indicating that deep models with a large number of tree nodes are being transferred across the tasks which result in more data transfer. Referring to Table 8, it can be noted that by increasing the depth of the model, the training accuracy is reducing, and the time taken is close to each other. However, after a max depth of 10, there is a jump in the time and the time is gradually increasing. This indicates that more amount of time is being utilized in transferring data, hence such a scenario has to be avoided during the execution or the spark parameters have to be tuned further to accommodate large task binaries.

Table 8 Performance of ML model in terms of RMSE and training time to monitor the effect of deep networks

Max depth	Training accuracy (kWh)	Holdout accuracy (kWh)	Time (sec)
2	5.850740	9.205257	28.711398
4	5.422266	9.738556	25.833542
6	4.759868	11.30912	24.382604
8	4.401291	17.37286	25.479490
10	4.153886	12.44175	26.977820
12	4.083622	12.50476	29.376672
14	4.077183	17.77995	32.170327
16	4.081471	17.79192	35.396624

7.2.2.3 Overfitting

Most of the machine learning models perform accurately post tuning of hyperparameters. However, excess tuning of parameters tends to fit the training data so accurately that the model is overfitted. Once overfit, the models do not perform as expected on the new forecasting dataset. To avoid such a case many measures are taken to avoid overfitting in the training data. Consideration of holdout data set which has never been used in the training is one of the measures to prevent overfitting. In the case of tree ML models, the depth of the tree or the number of nodes while training can be regulated. An experiment is performed by increasing the depth of the tree and the num of nodes in the trained model is monitored (refer to Figure 29). The x-axis shows the number of nodes, and the y-axis is the performance measure in terms of RMSE. It can be noted that the black line in Figure 29, which indicates the training RMSE, is decreasing with an increase in the number of nodes by fitting the dataset onto the trees as deeper as possible. Whereas the red line which

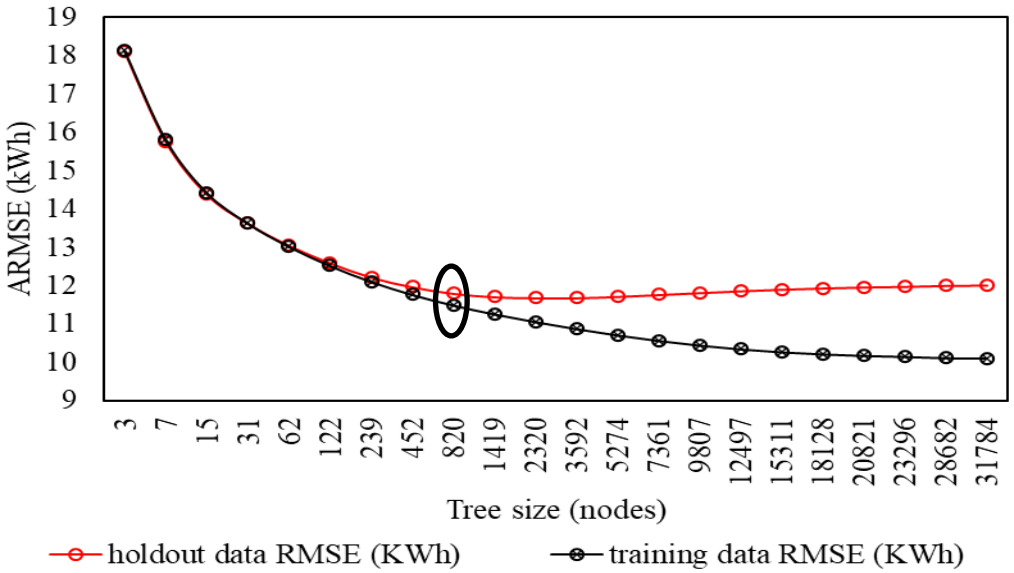

Figure 29 ARMSE of training and holdout dataset for spark decision tree. The spot above 820 nodes result in overfitting of the datasets.

denotes the holdout RMSE does not show such a trend. After a point, the models' performance starts deteriorating. This point is called the sweet spot where the models tend to start overfitting. Hence the depth of the model must be restricted below this point which indicates a value of 820 nodes in Figure 29. By taking such a measure on the average RMSE of all the transformers data, a max depth of 8 is chosen to perform training on the spark ML models.

7.2.2.4 Validation of historical data

The use of historical data has been used in the field of data mining to discover regularities to improve the decision-making processes [111]. For the dataset at hand, the data is collected for 3 years over the years 2017 to 29. Experiments have been run to understand the effect of historical data in forecasting the models. A sample of transformers data has been collected with varying horizons such as data from 2019, 2018, 2017, 2018+2019, and 2017+2018+2019. The dataset sizes vary for each of the three cases as shown in Table 9.

Table 9 Number of records for a sample transformer for the varying horizon

Dataset	Number of records
2019	5,895
2018	8,760
2017	8,759
2018+2019	14,654
2017+2018+2019	23,389

Considering each of the scenarios discussed in Table 9 mean square error and mean absolute percentage error has been measured for all the transformers in Dataset 1. The results for mean squared error have been described in Figure 30 and the mean absolute percentage error in Figure 31. The results indicated in both the figures are averaged for all the 1000 transformers. 24-hour lag day values have been added as a pre-processing step for the analysis. The x-axis for both the figures indicates the average time taken to fit the models on the training data and the y-axis indicates the measure in KWh. The bars in blue and green belong to DTR and RFR respectively. It can be observed in both the figures that the average time is taken to train increases as the data is added. It can be observed that both DTR and RFR average MSE is least considering only data from 2019 as compared to the cases of 2019+2018 and 2017+2018+2019. The training time for the RFR is more than DTR in all the cases as it is an ensemble model and runs a lot of computations as compared to a single decision tree. Average MAPE as shown in Figure 31 is still the least considering only data from 2019.

From the experiments of adding historical data, although considering only data from 2019 gives results better than the rest of the cases. This discovery of mining the load data gives an insight that only considering data from 2019 is sufficient to generate models with high accuracy whilst saving a lot of execution time needed to process the historical data.

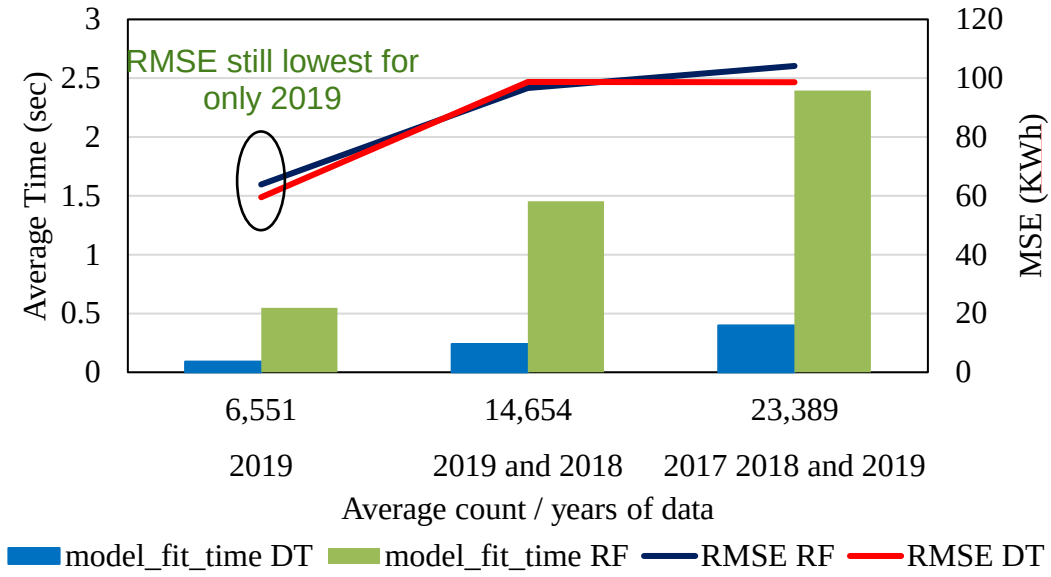

Figure 30 MSE Results on Average of decision tree and random forest(1000 t/f)

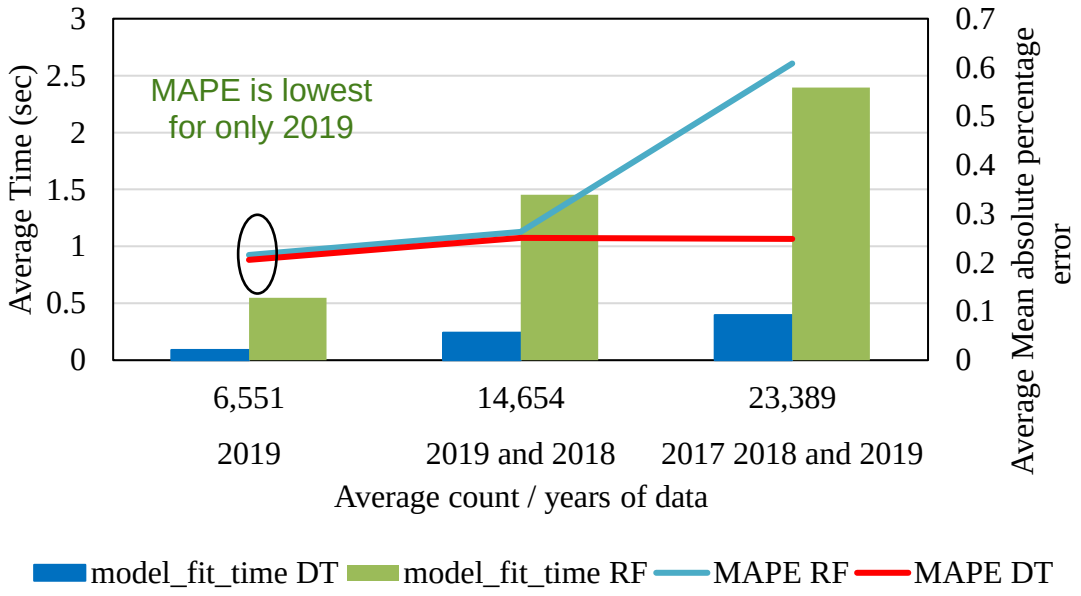

Figure 31 Mean absolute percentage error results on decision tree and random forest (Average of 1000 t/f)

7.2.2.5 Validation of accuracy

This section discusses the results after considering the validation of accuracy, computation time, and overfitting in the previous sections. For each of the spark ml models, the performance of the training dataset and holdout dataset are compared. Table 10 reports the $ARMSE$ computed for all the 1000 datasets. Holdout $ARMSE$ indicates the quality of load forecasting. Thus, a lower value of $ARMSE$ indicates a better load forecasting model. The table indicates that the values of holdout $ARMSE$ are the lowest for the spark random forest regression model.

Referring back to Figure 27(b), the execution time for the random forest is not the lowest but is comparable to the spark DTR model. Even though the random forest is an ensemble model, the execution time is not as large compared to other models. This is because the way spark performs its execution is that it utilizes its parallel computing capability to execute each of the decision trees individually and gives back the result. The actual power of spark in terms of execution can be observed here. Thus, it can be concluded that the spark RF performs better than the other spark ml models under comparison.

Table 10 Final ARMSE, for training and holdout dataset after choosing tuned parameters.

Algorithm	Training $ARMSE$ (kWh)	Holdout $ARMSE$ (kWh)
Spark Decision Tree Regression Model	9.17193954	10.80716307
Spark Random Forest Regression Model	8.01070855	10.60056138
Spark Gradient-Boosted Trees	4.20657451	11.88559708

Figure 32 shows the plot of $RMSE$ of all the distribution transformers under consideration. For randomly chosen DTs, those indexed as 0, 78, 208, 91, 13, 39, 104, 1, 52 present the training $RMSE$ and holdout $RMSE$. The plots indicate that the forecasting accuracy follows the training accuracy closely attributing to the fact that the built ML model is quite robust in terms of performance while increasing the speedup when a large number of jobs is performed.

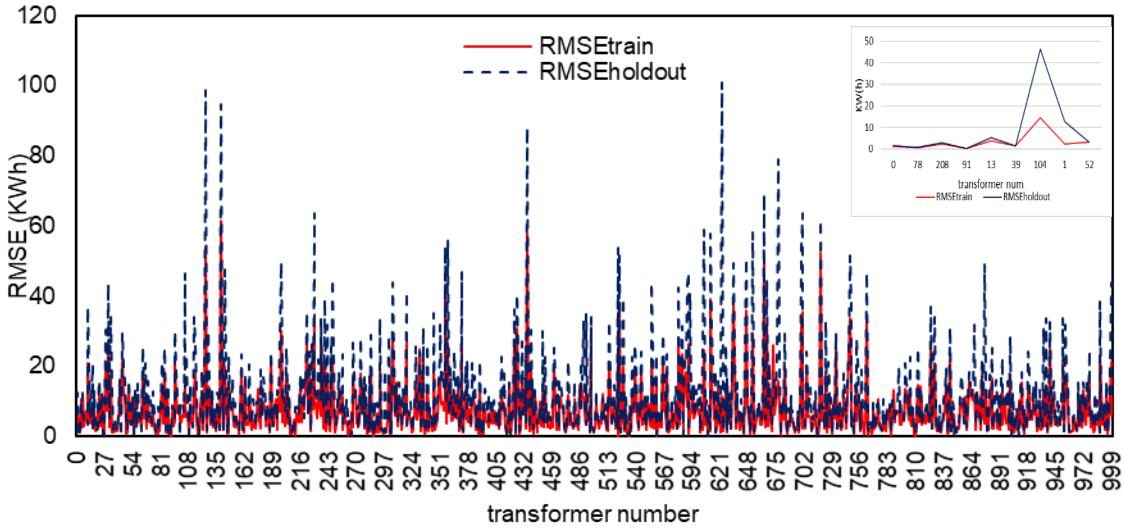

Figure 32 ARMSE comparison of training and holdout dataset for all the DT's

7.2.3 Experimental results on Scalability

This section outlines the results of the proposed methodology on a dataset obtained for dataset 2 with 10 thousand transformers and dataset 3 with100 thousand transformers. The results focus on the execution time and accuracy of the ML models based on the proposed methodology. All the experiments are conducted on spark 3.0.1 on top of the Hadoop platform with 7 worker nodes and 1 name node each with 120GB of memory and 20

cores. YARN is installed on top of Hadoop as the resource manager and HDFS is used for the distributed storage of data.

7.2.3.1 Estimating Number of Clusters Utilizing Optimized Scheduling of Parallel Jobs

A parallel implementation of the k-means is discussed in this section. The Lloyd's iterations as part of the k-means clustering methodology can be parallelized in the MapReduce framework, hence can be utilized in the spark framework. The algorithm is a parallel version of the k-means++ clustering technique. The details of the algorithm are presented in Algorithm 1 [112]. The initial center is picked up at random and it computes the initial cost ψ, followed by subsequent $log(\psi)$ iterations. Given each set of c centers, the algorithm samples with probability $\frac{ld^2(x,C)}{\phi_X(C)}$ for each sample x, and l is the oversampling factor give as $\Omega(k)$. It can be noticed from the algorithm that the size of C is

Algorithm 1 k-means||

Step1: Sample a point uniformly.

Step2: $\psi \Leftarrow$ cost of C for X.

Step3: Repeat Step4 and Step5 for $O(log\psi)$ times.

Step4: Sample new points with probability $\frac{\boldsymbol{ld^2(x,C)}}{\boldsymbol{\phi_X(C)}}$

Step5: Add the new points to the points sampled at step1.

Step6: End repetition.

Step7: Choice of $\boldsymbol{\omega_x}$ number of points closer to x than any other point in c.

Step8: form k clusters by reclustering the weighted points in C.

less than the input size significantly.

Concerning the problem statement of the optimal value of k while performing clustering, the data is clustered for all the values in the range $[2, K]$ to choose the best value of k resulting in an iterative process. The work proposes these iterations k to be submitted as parallel jobs. If θ is the batch of jobs submitted, then the total number of iterations can be reduced to equation (20)

$$Number\ of\ iterations\ =\ \frac{(K-2)+1}{\theta} \tag{20}$$

By submitting θ jobs the CO value will be reduced to $COperJ$ (cores per job) as shown in equation (21)

$$COperJ\ =\ \frac{CO}{\theta} \tag{21}$$

The resource allocation per job is now reduced, however, if each of the jobs requires only a limited number of resources, then multiple jobs can be run to expedite the iterative process of finding the optimal value of k. Algorithm 2 is stated with the pseudo-code describing the functional behavior of the scheduling strategy to deliver the main idea.

In this section, the efficiency of the proposed algorithm on a 10k transformers and 100k transformers real-world dataset provided by Spanish utility has been evaluated. The results are investigated on scalability, execution time, and evaluation score. All the experiments have been performed with pyspark programming language to integrate python supported ML techniques with the Spark SQL module which supports distributed query engines and data frames.

Algorithm 2. The proposed k-means parallel in-memory optimal job scheduling algorithm

Input:

j: the number of batches

w: number of workers (indicates $CoperE$)

K: Maximum value of k.

r: step value of k

csv: an empty csv file to accumulate all the results

Initialize:

def *cluster* (k):

Perform k-means parallel clustering;

Evaluate clustering on the evaluation measures;

Update the clustering scores;

end def

Output:

csv: the WSSE and Silhouette of data for all values of k.

Assign the number of workers and create a thread pool equal to the number of workers;

Create an array $[2, K, r]$;

Call pool.map function for cluster function with the array of k and $cluster$ function as variables;

The function cluster is called in j batches n times resulting in n/j iterations. The jobs are assigned to the next available processors in a random fashion. The WSSE and silhouette evaluation scores for each of the values of k are simultaneously updated after each execution;

close pool;

return csv

The experiments have been conducted on [6.1.1 Dataset 1] and 6.1.2 Dataset 2 provided by Spanish utility Iberdrola. In both datasets, duplicates have been dropped and

the time stamp variable has been segregated into the year, month, and day of the timestamp. The meter IDs have been anonymized with an index, while a dictionary is created to refer to the meter IDs corresponding to the index if needed. The total number of features in the data is 8. To perform clustering, the prediction variable (load value) with the help of timestamp has been averaged hourly and daily to obtain a matrix with each row indicating the averaged load. The data is stored in a compressed format of Optimized Row Columnar (ORC) and collected from hive datastores of the utility storage facility. Both the datasets were preprocessed and the preprocessing time of both Dataset 1 and Dataset 2 individually was less than a minute.

To measure the goodness of our clusters, we use WSSE which is a measure of how far each point X is from its centroid. The WSSE is calculated as shown in (22)

$$\sum_{i=1}^{N} d(X, \bar{C}_i) = \sum_{i=1}^{N} \left(\sum_{j=1}^{D} \left(C_{ij} - \bar{C}_{ij} \right)^2 \right) \tag{22}$$

where X is each of the points in the data with D dimension, C_i is the point X belonging to the cluster, C_{ij} is the j^{th} dimension of i^{th} point in the cluster and $\bar{C}_{ij}$ is the j^{th} dimension of the cluster center. The computational complexity of calculating the distances is given by $O(N * D)$, where N is the cardinality of the dataset. To ease the cluster evaluation complexity, equation (23) is expanded as

$$\sum_{i=1}^{N} d(C_i, \bar{C}_i) = \sum_{i=1}^{k} \left(\sum_{j=1}^{D} \left(\sum_{l=1}^{P} \left(C_{lj} - \bar{C}_{lj} \right)^2 \right) \right) \tag{23}$$

where k is the number of clusters and P is the number of records within a single cluster. Equation (23) holds when

$$\sum_{l=1}^{k} P_l = N \tag{24}$$

After rearranging equation (23), the computations for each of the k clusters can be distributed as broadcast variables to the worker nodes w to incorporate distributed processing. The platform computes the sum of one cluster or segment of the data on one worker, then a sum of a different segment or cluster over on another worker, and then combines those two sums as the final result. The computational complexity is hence reduced from $O(N * D)$, to $O\left(\frac{D*N}{w}\right)$ by using a parallel implementation of the WSSE by changing (22) to (23).

Silhouette Score is used to measure the clustering consistency within the clusters as indicated in equation (25) [113]. A value of 1 indicates all the data to be appropriately clustered.

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} = \begin{cases} 1 - \frac{a_i}{b_i} & if\, a_i \leq b_i \\ \frac{bi}{a_i} - 1 & if\, a_i > b_i \end{cases} \tag{25}$$

where a_i is the average distance of i^{th} point in the cluster to all the other points within the cluster and b_i is the average distance of i^{th} point with all the other points to which the point i does not belong. If a_i the measure is small and the b_i measure is large, the value of s_i will be close to 1 indicating the points are appropriately clustered. The algorithm computes the distance of each of a couple of points in the dataset by the equation (26) below.

$$\sum_{i=1}^{N} d(X, C_i) = \sum_{i=1}^{N} \left(\sum_{j=1}^{D} \left(x_j - C_{ij} \right)^2 \right) \tag{26}$$

This measure is not scalable in N, hence, to ease the cluster evaluation computational complexity, Equation (26) is expanded as equation (27)

$$\sum_{i=1}^{N} d(X, C_i) = \sum_{i=1}^{N} \left(\sum_{j=1}^{D} x_j^2 + \sum_{j=1}^{D} C_{ij}^2 - 2 \sum_{j=1}^{D} x_j C_{ij} \right) \tag{27}$$

The equation is further rearranged as shown in (28)

$$= \sum_{i=1}^{N} \sum_{j=1}^{D} x_j^2 + \sum_{i=1}^{N} \sum_{j=1}^{D} C_{ij}^2 - 2 \sum_{j=1}^{D} \left(\sum_{i=1}^{N} C_{ij} \right) x_j \tag{28}$$

$$= N\xi_X + \psi_\Gamma - 2 \sum_{j=1}^{D} Y_{\Gamma j} x_j \tag{29}$$

where ξ_X is the sum of squares of each of the X points and is a constant. ψ_Γ is also a constant for each of the clusters Γ. $Y_{\Gamma j}$ is a vector for all the N points and is fixed for each cluster Γ. The average distance of a point is given in (30)

$$= \xi_X + \frac{\psi_\Gamma}{N} - 2 \sum_{j=1}^{D} \frac{Y_{\Gamma j} x_j}{N} \tag{30}$$

where the constant ξ_X can be precalculated for each of the points X, and ψ_Γ, Y_Γ for each cluster Γ for k clusters. These pre-computed values for the k clusters are distributed as broadcast variables to the worker nodes w. The computational complexity is hence reduced from $O(N^2 * D)$, to $O\left(\frac{k*D*N}{w}\right)$ by using a parallel implementation of the Silhouette by changing (26) to (30).

Extensive experiments have been performed to examine the speedup of the parallel jobs. To measure the performance of load forecasting, Root Means Square Error (RMSE) has

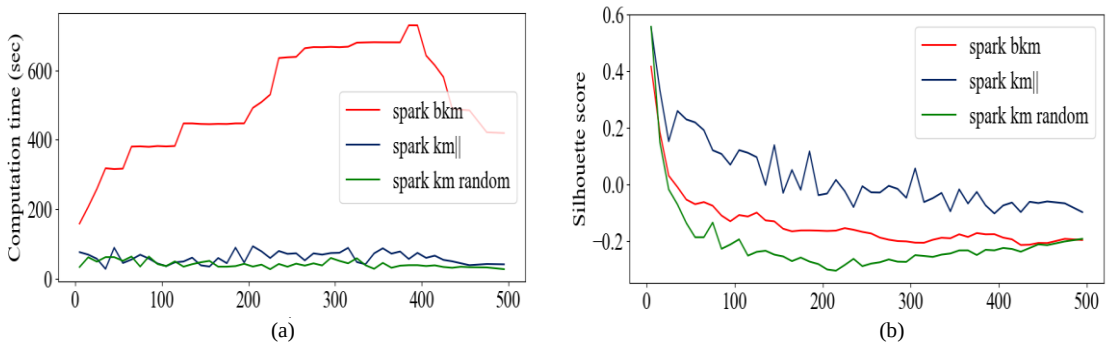

Figure 33 Computation time and silhouette score of three clustering techniques on spark for varying values of k

been utilized as the evaluation measure. For 6.1.2 Dataset 2, we compare the performance of spark k-means random, k-means parallel, and bisecting k-means using the parameters of k ranging from 2 up to 500. An outer bound of 500 is chosen as the number of clusters beyond 500 is undesirable and does not serve the purpose of clustering to reduce the number of models to be trained. Figure 33 illustrates the computation time and silhouette score for various values of k. From Figure 33(a) it can be observed that the computation time increases for spark bisecting k-means till a k value of 400 and then drops. In the case of clustering, as the value of k increases, the number of partitions in the data also increases which is distributed across workers and can result in a reduction in runtime. It is observed from Figure 33(a), that for both k-means parallel and k-means random the computation time is less as compared to bisecting k-means. The silhouette score in Figure 33(b), indicates that for all values of k, k-means parallel shows superior performance. The higher

the silhouette score better the clustering quality. Hence spark k-means parallel is a choice algorithm to perform clustering on big data.

Experiments of hourly load consumption and daily load consumption for different values of k have been conducted and the execution time has been measured. The data for evaluation consists of averaged load matrix of size [1000,1003] and [1000,24000] for daily load consumption and hourly load consumption, respectively. Clearly, from Figure

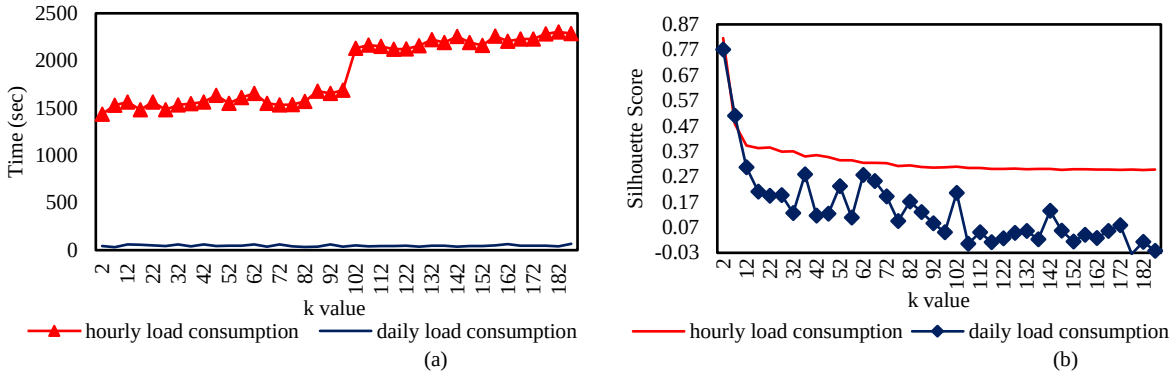

Figure 34 Computational time and clustering efficiency measure of hourly and daily load consumption of Dataset1.

34(a), the hourly load consumption takes more time as compared to daily load consumption. In [114], the authors suggest that the optimal value of k value is 93 utilizing the k-Medoid clustering algorithm. Regarding the silhouette scores, the result in Figure 34(b) emphasizes that more data with hourly load patterns leads to better clustering as the silhouette score is closer to 1 for the majority values of k the hourly load as compared to daily load.

The efficiency of the proposed k-means parallel in-memory clustering component parallel jobs to the RMSE has been evaluated. Note that the idea behind this algorithm is to

distribute the load forecast of all the transformers across the worker nodes of the big data platform. The data is divided into clusters and the choice value of k is discovered. Once the value of k is known this value is used to perform clustering on the data. Once the clusters are known the transformer closest to the center is chosen as the representative cluster. This representative cluster is chosen as the training dataset and the load is forecasted for the rest of the transformers. Thus, the value of k only decides the right clustering value and not the accuracy of the load forecasting model. To perform the load forecasting accuracy analysis, the decision tree algorithm is chosen based on its performance considering a tradeoff between accuracy and execution time [36]. The focus is on the effect of clustering, hence a deeper analysis on the choice of ML model is not considered in the evaluation. To test the clustering strategy firstly 90% of the data of all the transformers is used for training and 10% as the holdout dataset. The training of all the transformers is performed individually and the data is not collected at one place for all the transformers for the results indicated in Figure 35(a). In the results of Figure 35, the red line indicates training RMSE, and the blue line indicates RMSE for the holdout dataset. As shown in Figure 35(a), the RMSE for holdout exceeds the training data RMSE by not more than 20KWh approximately for the majority of the transformers. This shows a good performance of the trained model. It can also be noted that most of the transformers' RMSE ranges between 0 to 10 kWh. Figure 35(b) shows the results on representative clustering, X-axis shows the cluster number, and each cluster number is repeated for many cases showcasing the different transformers within a cluster. In representative clustering as only 90% of data of the transformer closest to the centers is used for training, the RMSE

for training will remain the same for all the transformers within the cluster. This model is used to test the 10% data of the remaining transformers within the cluster. The advantage of representative clustering is that only k transformers closest to the cluster centroids are trained instead of all the transformers reducing the training time further. These results show that the RMSE for almost all the transformers is in the range of 0 to 40 kWh for training and more than 40kWh for the holdout data. With the use of clustering, the RMSE of the holdout data farther from the cluster center is expected to deteriorate as the model may not be completely suitable for all the transformers within the cluster as the clustering quality is not 100% in a real-time scenario.

The performance in terms of scalability by running the tests on dataset 2 which is 10 times

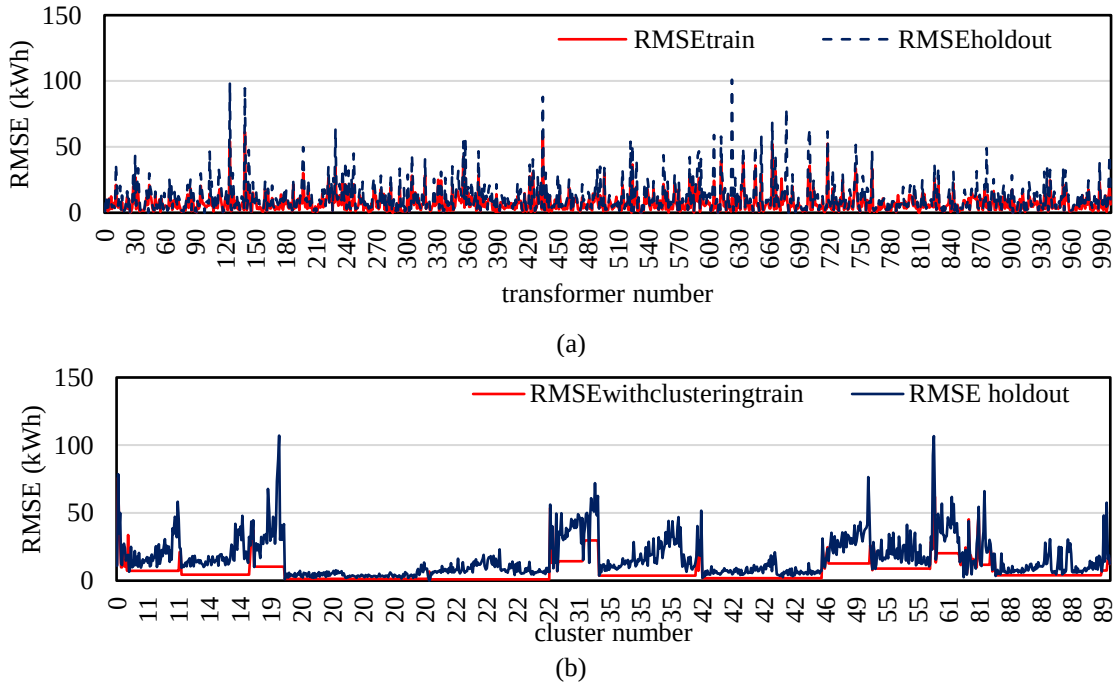

Figure 35 RMSE in kWh of 1000 transformers with representative clustering and without representative clustering

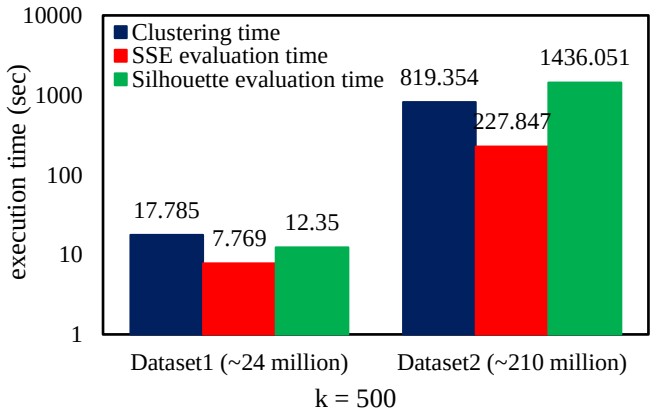

Figure 36 Comparison of run time of spark parallel k-means with varying values of k on different datasets.

larger than dataset 1 has been compared. To access the scalability, the total time taken to cluster the data for the various value of *k* for both the datasets is measured. The experiments aim to verify that an exponential increase in big data does not have an exponential increase in the run time for all the values of *k* clusters. From Figure 36, it can be observed that as we increase the data size from ~24 million records to ~210 records both consisting of 8 features, for a *k* value of 500 we obtain results for clustering along with the evaluation measures. In terms of comparison of evaluation scores, the silhouette score takes longer than the proposed SSE score. An SSE evaluation time of 227 sec for dataset 2 with 210 million records prove the scalability of the proposed SSE evaluation measure in this paper. The results also showcase that the spark platform has a noticeable effect on the parallel k-means algorithm as we scale up the size of the data by almost 10 times. Hence, the parallel k-means algorithm with Spark on top of Hadoop is very suitable for large scale and can handle large datasets for clustering smart meter data.

To evaluate the proposed job scheduling of parallel k-means in-memory processing the experiments are run on the various value of k for both spark k-means parallel and the

Table 11 Time taken for spark k-means parallel and proposed scheduling methodology with various k values

k	spark k-means||			Proposed job scheduling for k-means||		
	Prediction time (sec)	SSE calculation time (sec)	Silhouette score calculation time (sec)	Prediction time (sec)	SSE calculation time (sec)	Silhouette score calculation time (sec)
250	179.633	5.913	11.719	194.035	27.572	40.924
500	11.467	4.106	9.644	209.527	17.477	28.826
750	14.257	4.237	9.895	195.819	25.790	36.600
1000	12.578	4.606	10.262	194.020	27.587	40.149
2500	13.670	6.629	13.241	201.884	21.556	36.470
5000	14.185	8.446	15.700	209.013	20.744	33.398
7500	15.028	10.425	18.300	199.965	23.290	38.010
10000	15.525	11.926	20.379	201.218	28.500	41.122
Total time	416.694			248.297		

Total time includes the prediction time, SSE calculation time and Silhouette score calculation time for all the values of k

proposed method on daily average load values of 6.1.2 Dataset 2. Table 11 tabulates the time taken to execute 8 values of k ranging from 0 to 10000 with a step size of 250. The time taken for clustering (prediction time) is lower for each of the values of k as compared to the proposed method, however, with the proposed method, the total time is lower than

the spark k-means parallel. The run time for both the evaluation measure WSSE and silhouette score has been computed.

It is observed that the time taken for SSE calculation in both cases is lesser than the silhouette score as expected. Silhouette score has a greater number of distance measures to be evaluated as compared to SSE. However, it can be noted that both the evaluation measures are scalable and can be used to measure the clustering quality. The total time indicates the time taken to cluster and evaluate both the measures for all the values of k. It is the time taken for clustering, SSE evaluation, and silhouette score evaluation for all the values of k. The total time in the proposed optimal scheduling method is 40.4% lesser than the spark k-means parallel method without optimal scheduling. In the proposed method as the workers are distributed across various jobs each worker gets a lesser share of the memory as compared to the spark k-means parallel classical method. However, the overall time to run all the iterations of k is lower in the proposed methodology. Thus, we can conclude that the proposed optimized scheduling strategy performs better than the spark k-means parallel algorithm in terms of execution time.

The number of clusters has also been estimated on Dataset 3 for 100k transformers data using kmeans|| as the clustering technique. The results have been shown in Figure 37. The graph on the x-axis shows the various value of k for which the clustering technique has been tested to obtain the elbow curve. The primary x-axis shows the measure of error using SSE and the secondary y-axis indicates the times taken to cluster the data in seconds. The suggested value of k is chosen to be 15,050 considering a dip in the elbow at this k

value. It can also be observed that the time that it takes to cluster the data for a value of *k* of 15,050 is ~2800 sec or ~46 minutes.

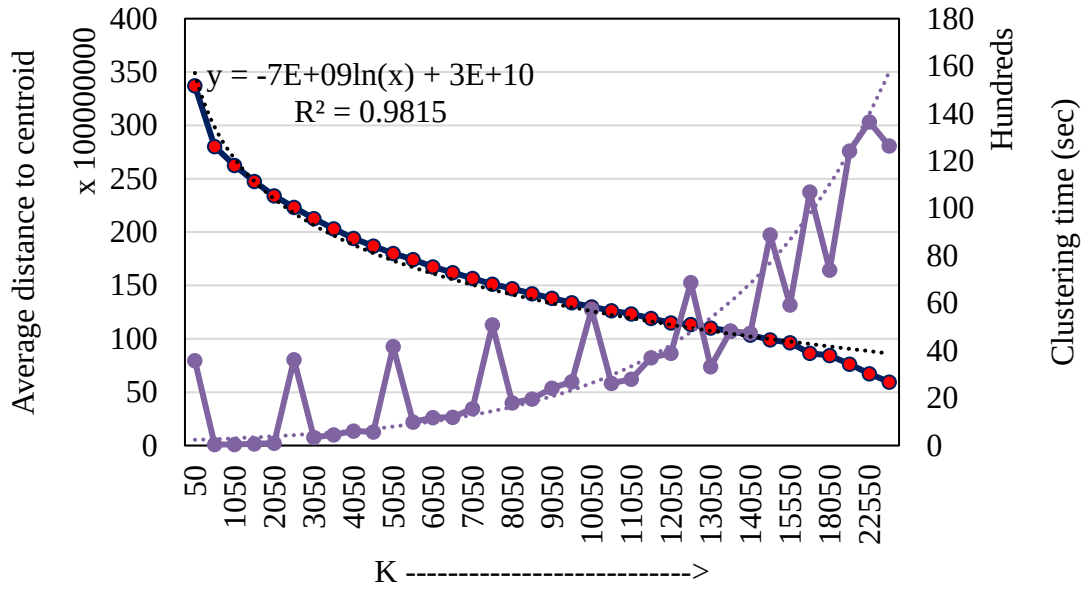

Figure 37 Kmeans|| result on Dataset3 - elbow curve

The spikes in the graph for clustering time indicate that when any of the workers are busy performing other tasks the time it takes increases. However, as the tests are run based on the proposed scheduling methodology discussed in section 7.2.3.1 all the values of *k* are submitted simultaneously as multiple jobs and are executed in parallel. The time to evaluate the clusters with the proposed parallel implementation of the evaluation score (SSE) with the help of workers is shown in Figure 38. All the values of *k* are distributed as broadcast variables to the worker nodes *w*. The time taken to evaluate each value of k is fairly less and does not reach more than 60 secs for a k value as large as 25000.

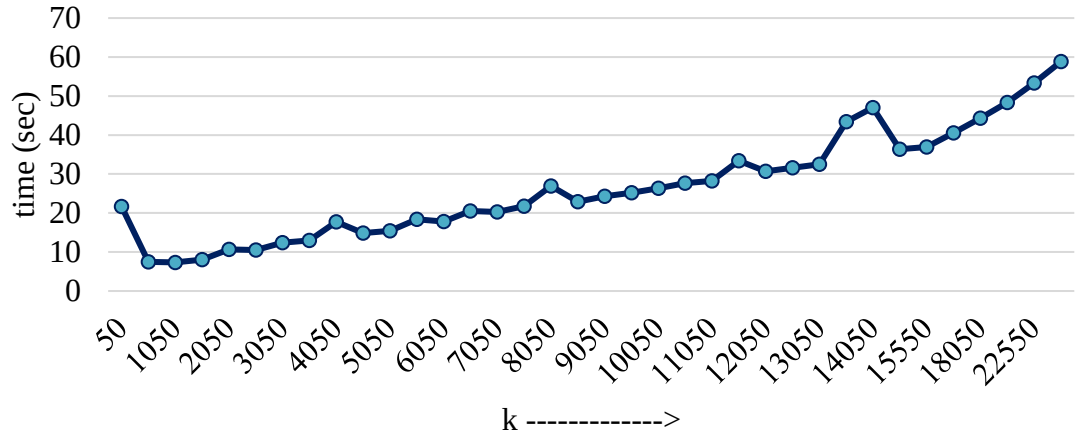

Figure 38 Time to evaluate the clusters using the parallel sum of square error on Dataset3

For a chosen k value of 15,050, the distribution of the number of transformers per cluster is noted. Figure 39 indicates the bins for the number of transformers on the x-axis and the number of transformers falling under each of the bins on the primary y-axis. The secondary y-axis indicates the % of the total number of transformers falling under the respective bins.

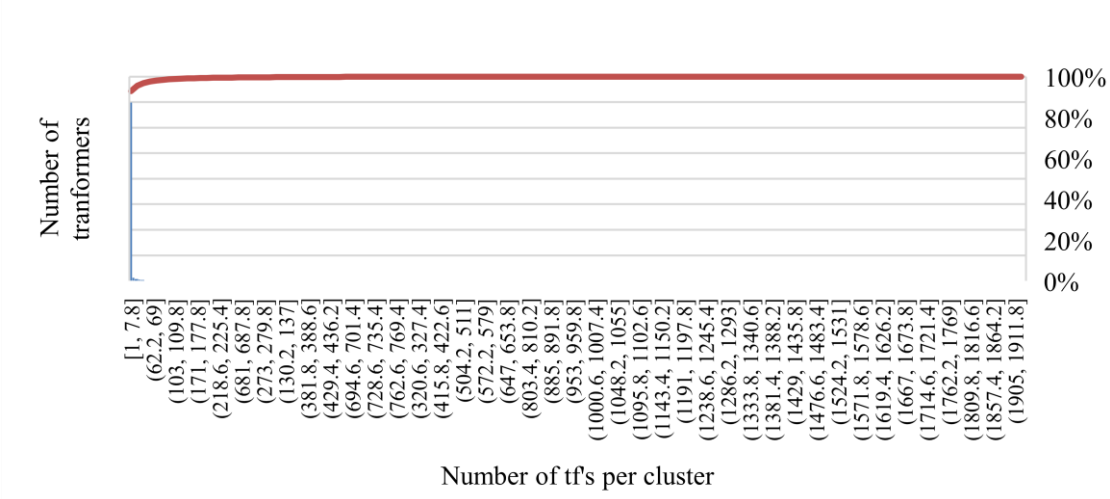

Figure 39 Distribution of transformers per cluster

The majority of the transformers (>90%) fall under the 1-10 bin width considering 15k clusters. The bin with a majority number of transformers is a fair distribution of the transformers considering 105k transformers in total distributed across 15k clusters. However, there are clusters with only 1 transformer per cluster and also clusters with 1928 transformers belonging to a single cluster. Considering the k value of 15,050 load forecasting has been performed considering the transformer closest to the centroid as the training model and the rest of the transformers data is used for testing. A sample cluster is considered with 10 transformers to observe the performance of R-square in this scenario. Figure 40 shows the comparison of with and without clustering for a sample cluster with 10 transformers. The x-axis shows the names of the transformers are positioned as per the distance of the transformer from the centroid. The transformer closest to the centroid is ZIVS004475051. The blue line indicates the R-square value without performing any clustering and the red line indicates the R-square value considering the trained model of transformer ZIVS004475051. The blue is on a record high above 0.98 R-square value for all the transformers, however as expected the red line with R-square indicating the goodness of fit measure values decrease. To evaluate the performance statistically an average of R-square without and with clustering is calculated to be 0.991962064 and 0.95100845 respectively. The %loss on average can be evaluated to be 4% by clustering data to be able to generate a lesser number of trained models and save computation time eventually. In this particular scenario, only a single model will be trained instead of training 10 different ML models.

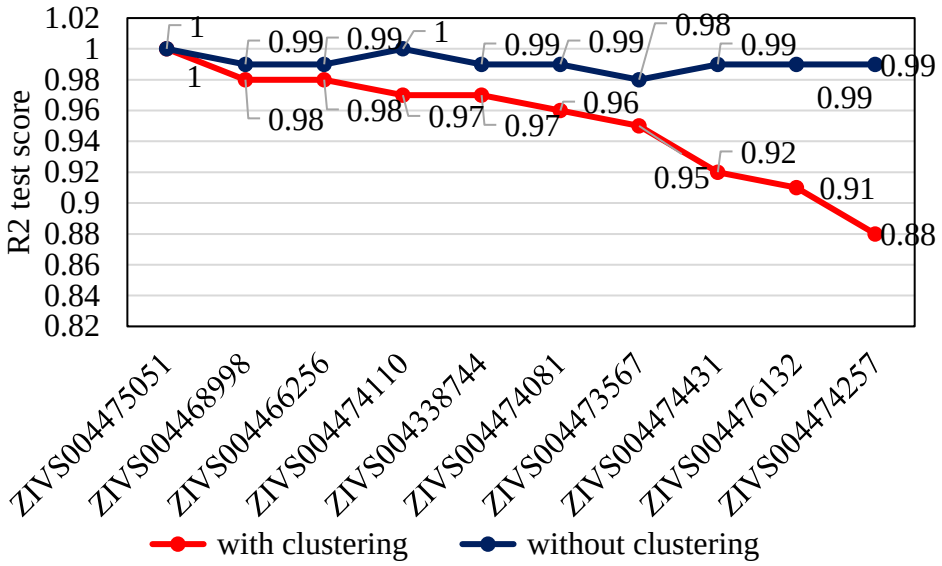

Figure 40 R2 Results for load forecasting with clustering

7.2.3.2 Load Forecasting results on Dataset 2 (10k transformers)

The results outlined in this section are for Dataset 2. The results indicated are with regards to a quantile range of [0.25 – 0 kWh, 0.5 – 16 kWh, 0.75 – 42 kWh, 0.9995 - 1751 kWh] utilizing decision tree regression for a maxDepth of 8. The results are indicated in Table 12 are not only in terms of RMSE but also in terms of the R-squared statistic. Out of the ~10k transformers, as indicated in 6.1.2 Dataset 2, columns 2&3, 5&6 in Table 12 indicate the number of transformers falling under the designated range in columns 1 and 4 respectively. RMSE is an absolute measure of fit as indicated in equation (14), and is observed to be low for a majority of transformers for both DTR and random forest (RF) regressor. The results with the R-squared measure on the same dataset are also captured in Table 12. It can be noticed that a majority of transformers for both DTR and RF fall under the R-squared range of [0.5, 1]. An R-squared value as

close as possible to 1 makes the model more accurate. Hence to improve the models for a majority of the transformers load values of the lag day have been added to the data in the preprocessing step.

Table 12 Load Forecasting results in terms of RMSE and R-squared on Dataset2

RMSE			R-Squared		
Range	tf's DTR	tf's RF	Range	tf's DTR test	tf's RFR test
			$R2 < 0$	18	23
$0 \leq RMSE \leq 10$	9655	9612	$0 \leq R2 < 0.2$	67	66
$10 < RMSE \leq 50$	262	314	$0.2 \leq R2 < 0.5$	1281(13%)	957(10%)
$50 < RMSE \leq 100$	6	10	$0.5 \leq R2 < 0.8$	5860(61%)	4334(45%)
$100 < RMSE \leq 200$	18	5	$0.8 \leq R2 \leq 1$	2250(23%)	4102(43%)

Considering the data partitioning technique discussed in 5.3.4 Data partitioning in a distributed environment lag values for 24-hour load values are added to the dataset and results have been obtained. The Quantile range for 0.25, 0.5, 0.75, and 0.9995 is 0 kWh, 16 kWh, 42 kWh, and 1751 kWh respectively. The tuning parameters for random forest regressor have been performed for a maxDepth = [6, 8, 10] and numTrees [6, 12, 24]. Based on the results obtained random forest regressor with a maxDepth of 8 and numTrees as 20 has been considered and the results are tabulated in Table 13. The total time taken to obtain the results for 5900 t/f's is 8.5 hours.

Table 13 Load Forecasting results in terms R-squared on Dataset2 with 5900 t/f's adding lag values

R−squared (tuning individual models)			R−squared (adding 24−hour lag values)		
Range	tf's RF test	tf's RF test tuned	Range	tf's RF test	tf's RF test tuned
$R^2 < 0$	28	23	$R^2 < 0$	28	34
$0 \leq R^2 < 0.2$	56	51	$0 \leq R^2 < 0.2$	28	29
$0.2 \leq R^2 < 0.5$	948 (16%)	731 (12%)	$0.2 \leq R^2 < 0.5$	34	35
$0.5 \leq R^2 < 0.8$	3540 (60%)	3267 (55%)	$0.5 \leq R^2 < 0.8$	79	73
$0.8 \leq R^2 \leq 1$	1279 (21%)	1778 (30%)	$0.8 \leq R^2 \leq 1$	5698 (96%)	5704 (96%)

It can be observed from Table 13 that most transformers are now moved from an R-squared range of [0.5, 1.0] to [0.8, 1.0]. It clearly shows that adding lag values has resulted in a drastic improvement of the model accuracies for a majority of transformers.

7.2.3.3 Load Forecasting results on Dataset 3 (100k transformers)

In this section Dataset3 with 105k transformers has been considered to evaluate the performance of the proposed techniques to prove scalability. For feasibility, the dataset has been divided into 10 batches with approximately 10k transformers each and the tests have been performed. The results of load forecasting on 105k t/f's are shown in Table 14. The results indicate that for the test dataset almost 90% of the transformers

have an R-square value greater than 0.9. The time taken to train the models is shown in Table 15.

Table 14 Load forecasting results on dataset 3 (102,988 t/f)

R−squared (adding 24-hour lag values)				
	R-squared train		R-squared test	
	t/f number	% of total	t/f number	% of total
$R^2 < 0$	4	0	1075	1%
$0 \leq R^2 < 0.2$	5	~0	777	1%
$0.2 \leq R^2 < 0.5$	9	~0	644	1%
$0.5 \leq R^2 < 0.8$	461	~0	1,248	1%
$0.8 \leq R^2 < 0.9$	1,505	1%	1,440	1%
$0.9 \leq R^2 < 1$	100,311	97%	94,739	91%

RFR with a maxDepth of 8 and numTrees as 20 has been considered with a lag value of 24 hours. The total Time taken – ~4 days 5 hours for Data read, Pre-processing, ML modeling, Training, Prediction, and saving the results. Considering the techniques discussed the time taken to perform the computations for a total cycle model inference can also be estimated for a platform with specifications other than the one used for experimentation. Total time for testing 10% of each transformer – 2,216,366 sec (36,939 min or 615 hours). If 20 cores are utilized on each of the executors, then the computation time can be reduced to 31 hours. To perform hourly day-ahead prediction model inference for 24 data points can be estimated to 18.42 min. Comparing the platform

specifications with a possible platform in production the time can be estimated to be 7min for model inference. The details are estimated and explained in Table 16.

Table 15 Time taken to read the data, perform pre-processing, ML modeling, Training, Prediction, and Saving the results for Dataset3.

Batch1	~12 hours
Batch2	~12 hours
Batch3	~12 hours
Batch4	~12 hours
Batch5	9hrs, 31mins
Batch6	9hrs, 27mins
Batch7	10hrs, 6mins
Batch9	6 hrs
Batch10	8hrs, 39 mins
Total	~4 days 5 hours

Table 16 Model inference estimate for Dataset3.

Current platform configuration / node	Production platform configuration / node	Time estimate for model inference
7 nodes 20 cores 120 GB RAM 3 TB of disk Storage	38 nodes 96 cores 252 GB RAM 22 TB of disk Storage	7 min

CHAPTER VIII

CONCLUSION AND FUTURE WORK

8.1 Conclusion and Future Work

This thesis proposed a big data platform for managing the big data from the real distribution transformers to perform short-term load forecasting while utilizing multiprocessing and distributed computing techniques. Several factors that affect the choice of the machine learning model for the load forecast were taken into consideration. The parallel processing platform was set up to address the problem of simultaneously improving the short-term forecasting accuracy and speed. The obtained results indicate that the decision trees demonstrate superior performance in terms of accuracy and calculation speed. The findings of the work can be generalized and utilized for any number of transformers with similar load patterns. In this work, a smart scheduling algorithm to perform load forecasting on multiple distribution transformers was proposed. The proposed approach was implemented on Apache spark to not only deal with the challenges associated with computation time while handling the big data but also to optimize deployed jobs in a parallel environment. One of the distinctive characteristics of the proposed approach is its capability to submit as many jobs in parallel as that is achievable for efficient memory utilization. The processed big data was partitioned into various chunks and cached to improve the performance of big data storage that is too large to be stored. The other significant accomplishment of this work is the use of thread pool and fair scheduler in spark to speed up the processes with in-memory processing which resulted in a 43% improvement in execution time. This is a good optimization strategy for load

forecasting utilizing multi-AMI big datasets. Several experiments were performed to evaluate the scheduling strategy in terms of ML model forecasting error and execution time. The results with the use of spark ml libraries have shown superior performance in terms of both accuracy and execution time. The proposed ML models achieved higher accuracies over the previously proposed iterative algorithms. The merits shown in the experiments indicated that there is a great potential for the proposed method to be used in big data processing of multi-AMI environments.

In section 7, the experiments have been extended to an electrical network with 10,000 transformers and 100,000 transformers to prove the scalability of the proposed distributed computing methodology using a big data platform. Scaling the data to more than 1000 DTs requires more than a minimum of 100 jobs to be scheduled. However, the experiments have been conducted with restricted computational resources and intuitions have been developed with scaled spark cluster sizes. The tests have been performed to investigate the scalability of the clustering approach and to determine the optimal number of clusters for larger data sets.

In this thesis, a scalable big data platform is proposed to uncover the patterns in the load data and to perform forecasting. The future data will become larger, complex, and will be collected at a velocity. Analysis of streaming data is a step in that direction. This extension will improve the real-time computations of forecasting models.

REFERENCES

[1] S. N. Fallah, M. Ganjkhani, S. Shamshirband, and K. wing Chau, “Computational intelligence on short-term load forecasting: A methodological overview,” *Energies*, vol. 12, no. 3, 2019, doi: 10.3390/en12030393.

[2] I. W. Tsang, J. T. Kwok, and P. M. Cheung, “Core vector machines: Fast SVM training on very large data sets,” *J. Mach. Learn. Res.*, vol. 6, no. 13, pp. 363–392, 2005, Accessed: Jan. 12, 2021. [Online]. Available: http://jmlr.org/papers/v6/tsang05a.html.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” *2nd USENIX Work. Hot Top. Cloud Comput. HotCloud 2010*, vol. 10, no. 10–10, p. 95, 2010.

[4] Qiu, R. C., & Antonik, P. (2017). Smart grid using big data analytics: a random matrix theory approach. John Wiley & Sons.

[5] D. Alahakoon and X. Yu, “Advanced analytics for harnessing the power of smart meter big data,” in *2013 IEEE International Workshop on Inteligent Energy Systems (IWIES)*, 2013, pp. 40–45.

[6] M. Nisi *et al.*, “Transparently mining data from a medium-voltage distribution network: A prognostic-diagnostic analysis,” in *CEUR Workshop Proceedings*, 2019, vol. 2322.

[7] D. Arkhipova and C. Bozzoli, “Digital Capabilities,” in *CIOs and the Digital Transformation*, Springer, 2018, pp. 121–146.

[8] I. S. Group and others, “Managing big data for smart grids and smart meters,” *IBM Corp. whitepaper (May 2012)*, 2012.

[9] N. B. Reinprecht, G. White, and M. Peters, “Enabling European electrical transmission and distribution smart grids by standards,” *IBM J. Res. Dev.*, vol. 60, no. 1, pp. 1–3, 2016, doi: 10.1147/JRD.2015.2482878.

[10] “Romeo project lands in East Anglia ONE and Wikinger.” https://www.iberdrola.com/about-us/romeo-project. Online: accessed, 2019.

[11] H. Owen, M. Avila, A. Folch, L. Cosculluela, and L. Prieto, “A high performance finite element model for wind farm modeling in forested areas,” in *EGU General Assembly Conference Abstracts*, 2015, vol. 17.

[12] M. Mayilvaganan and M. Sabitha, “A cloud-based architecture for Big-Data analytics in smart grid: A proposal,” in *2013 IEEE International Conference on Computational Intelligence and Computing Research, IEEE ICCIC 2013*, 2013, pp. 1–4, doi: 10.1109/ICCIC.2013.6724168.

[13] Y. Simmhan *et al.*, “Cloud-based software platform for big data analytics in smart grids,” *Comput. Sci. Eng.*, vol. 15, no. 4, pp. 38–47, 2013, doi: 10.1109/MCSE.2013.39.

[14] J. Baek, Q. H. Vu, J. K. Liu, X. Huang, and Y. Xiang, “A secure cloud computing based framework for big data information management of smart grid,” *IEEE Trans. cloud Comput.*, vol. 3, no. 2, pp. 233–244, 2015.

[15] R. Kumar and S. Gupta, “Open source infrastructure for cloud computing platform using eucalyptus,” *Glob. J. Comput. Technol. Vol*, vol. 1, no. 2, pp. 44–50, 2014.

[16] Christophe Bisciglia. The smart grid: Hadoop at the tennessee valley authority (tva). http://blog.cloudera.com/blog/2009/06/smart-grid-hadooptennessee-valley-authority-tva/, 2009.

[17] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs, “Building a big data platform for smart cities: Experience and lessons from santander,” in *2015 IEEE International Congress on Big Data*, 2015, pp. 592–599.

[18] A. C. C. Bestavros, L. Hutyra, and E. Terzi, “SCOPE: Smart-city Cloud Based Open Platform and Ecosystem,” *Bost. Univ. Boston, MA, USA*, 2016.

[19] D. Puiu *et al.*, “Citypulse: Large scale data analytics framework for smart cities,” *IEEE Access*, vol. 4, pp. 1086–1108, 2016.

[20] Osborne Clarke (2016). OC: The smart cities law firm. Retrieved from http://smartcities.osborneclarke.com

[21] T. Zahariadis *et al.*, “FIWARE lab: managing resources and services in a cloud federation supporting future internet applications,” in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 792–799.

[22] K. Wang *et al.*, “Wireless Big Data Computing in Smart Grid,” *IEEE Wirel. Commun.*, vol. 24, no. 2, pp. 58–64, 2017, doi: 10.1109/MWC.2017.1600256WC.

[23] Y. Zhou, P. Li, Y. Xiao, A. Masood, Q. Yu, and B. Sheng, “Smart grid data mining and visualization,” in *PIC 2016 - Proceedings of the 2016 IEEE International Conference on Progress in Informatics and Computing*, 2017, pp. 536–540, doi: 10.1109/PIC.2016.7949558.

[24] X. Ding, L. Chen, Y. Gao, C. S. Jensen, and H. Bao, “Ultraman: a unified platform for big trajectory data management and analytics,” *Proc. VLDB Endow.*, vol. 11, no. 7, pp. 787–799, 2018.

[25] M. Brahem, K. Zeitouni, and L. Yeh, “Astroide: A unified astronomical big data processing engine over spark,” *IEEE Trans. Big Data*, 2018.

[26] M. Olson, “Hadoop: Scalable, flexible data storage and analysis,” *IQT Quart*, vol. 1, no. 3, pp. 14–18, 2010.

[27] A. Thusoo *et al.*, “Hive: a warehousing solution over a map-reduce framework,” *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1626–1629, 2009.

[28] A. F. Gates *et al.*, “Building a high-level dataflow system on top of Map-Reduce: the Pig experience,” *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1414–1425, 2009.

[29] M. Ismail, S. Niazi, M. Ronstrom, S. Haridi, and J. Dowling, “Scaling HDFS to

more than 1 million operations per second with HopsFS," in *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, 2017, pp. 683–688, doi: 10.1109/CCGRID.2017.117.

[30] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science grids: a viable solution?," in *Proceedings of the 2008 international workshop on Data-aware distributed computing*, 2008, pp. 55–64.

[31] M.C. Srivas et.al,Map-Reduce Ready Distributed File System, Patented Document Appl. No.: 13/162,439; Dec 2011.

[32] A. Lakshman and P. Malik, "Cassandra - A decentralized structured storage system," *Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010, doi: 10.1145/1773912.1773922.

[33] M. Zaharia *et al.*, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Nov. 2016, doi: 10.1145/2934664.

[34] © [2020] IEEE. Reprinted, with permission, from, A. Zainab, S. S. Refaat, H. Abu-Rub, and O. Bouhali, "Distributed Computing for Smart Meter Data Management for Electrical Utility Applications," in *2020 Cybernetics & Informatics (K&I)*, Jan. 2020, pp. 1–6, doi: 10.1109/KI48306.2020.9039899.

[35] A. Reuther, C. Byun, W. Arcand, … D. B.-J. of P. and, and undefined 2018, "Scalable system scheduling for HPC and big data," *Elsevier*.

[36] A. Zainab *et al.*, "A Multiprocessing-Based Sensitivity Analysis of Machine Learning Algorithms for Load Forecasting of Electric Power Distribution System," *IEEE Access*, vol. 9, pp. 31684–31694, 2021, doi: 10.1109/ACCESS.2021.3059730.

[37] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of 'big data' on cloud computing: Review and open research issues," *Inf. Syst.*, vol. 47, pp. 98–115, 2015.

[38] A. Makris, K. Tserpes, V. Andronikou, and D. Anagnostopoulos, “A classification of NoSQL data stores based on key design characteristics,” *Procedia Comput. Sci.*, vol. 97, pp. 94–103, 2016.

[39] S. Le, Y. Dong, H. Chen, and K. Furuse, “Balanced Nearest Neighborhood Query in Spatial Database,” in *2019 IEEE International Conference on Big Data and Smart Computing, BigComp 2019 - Proceedings*, 2019, pp. 1–4, doi: 10.1109/BIGCOMP.2019.8679425.

[40] H. A. Abdelhafez, “Big Data Technologies and Analytics,” in *International Journal of Business Analytics*, 2014, vol. 1, no. 2, pp. 1–17, doi: 10.4018/ijban.2014040101.

[41] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co., 2015.

[42] Carbone and Asterios Katsifodimos and Stephan Ewen and Volker Markl and Seif Haridi and Kostas Tzoumas, “Apache Flink {^{TM}} : Stream and Batch Processing in a Single Engine Paris,” *Undefined*, vol. 36, no. 4, 2016.

[43] M. Hahsler, M. Bolaños, and J. Forrest, “Introduction to stream: An extensible framework for data stream clustering research with R,” *J. Stat. Softw.*, vol. 76, no. 1, pp. 1–50, 2017, doi: 10.18637/jss.v076.i14.

[44] A. Bifet, S. Maniu, J. Qian, G. Tian, C. He, and W. Fan, “Streamdm: Advanced data mining in spark streaming,” in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015, pp. 1608–1611.

[45] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive Online Analysis,” *J. Mach. Learn. Res.*, vol. 11, no. May, pp. 1601–1604, 2010.

[46] G. D. F. Morales and A. Bifet, “SAMOA: scalable advanced massive online analysis.,” *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 149–153, 2015.

[47] C. Bockermann and H. Blom. The streams Framework. Technical Report 5, TU Dortmund University, 12 2012..

[48] G. Hesse and M. Lorenz, “Conceptual survey on data stream processing systems,” in *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 2016, vol. 2016-Janua, pp. 797–802, doi: 10.1109/ICPADS.2015.106.

[49] “Apache Storm Project.”, http://storm.apache.org/. Online: accessed, 2018.

[50] H. Daki, A. El Hannani, A. Aqqal, A. Haidine, and A. Dahbi, “Big Data management in smart grid: concepts, requirements and implementation,” *J. Big Data*, vol. 4, no. 1, p. 13, 2017, doi: 10.1186/s40537-017-0070-y.

[51] X. He, Q. Ai, R. C. Qiu, W. Huang, L. Piao, and H. Liu, “A big data architecture design for smart grids based on random matrix theory,” *IEEE Trans. Smart Grid*, vol. 8, no. 2, pp. 674–686, 2017.

[52] H. C. V. Ngu and J.-H. Huh, “B+-tree construction on massive data with Hadoop,” *Cluster Comput.*, vol. 22, no. 1, pp. 1011–1021, 2019.

[53] D. Yang *et al.*, “Fastpm: An approach to pattern matching via distributed stream processing,” *Inf. Sci. (Ny).*, vol. 453, pp. 263–280, 2018.

[54] X. Gao and J. Qiu, “Supporting queries and analyses of large-scale social media data with customizable and scalable indexing techniques over NoSQL databases,” in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2014, pp. 587–590.

[55] M. Y. Eltabakh, F. Özcan, Y. Sismanis, P. J. Haas, H. Pirahesh, and J. Vondrak, “Eagle-eyed elephant: Split-oriented indexing in Hadoop,” in *ACM International Conference Proceeding Series*, 2013, pp. 89–100, doi: 10.1145/2452376.2452388.

[56] Glänzel W: “On the H-index – A mathematical approach to a new measure of

publication activity and citation impact". Scientometrics 67(2), to appear.

[57] J. Dittrich, J.-A. Quiané-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad, "Only aggressive elephants are fast elephants," *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1591–1602, 2012.

[58] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi, "MD-HBase: A scalable multi-dimensional data infrastructure for location aware services," in *Proceedings - IEEE International Conference on Mobile Data Management*, 2011, vol. 1, pp. 7–16, doi: 10.1109/MDM.2011.41.

[59] J. Dittrich, J. A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing)," *Proc. VLDB Endow.*, vol. 3, no. 1, pp. 518–529, 2010, doi: 10.14778/1920841.1920908.

[60] B. Taube, S. G. Solutions, and V. Corporation, "Leveraging big data and real-time analytics to achieve situational awareness for smart grids."

[61] G. W. Andersonf, A. Guionnet, and O. Zeitouni, "An introduction to random matrices, volume 118 of Cambridge Studies in Advanced Mathematics." Cambridge University Press, Cambridge New York.

[62] A. Basak and M. Rudelson, "Invertibility of sparse non-Hermitian matrices," *Adv. Math. (N. Y).*, vol. 310, pp. 426–483, 2017, doi: 10.1016/j.aim.2017.02.009.

[63] L. Xie, Y. Chen, and P. R. Kumar, "Dimensionality reduction of synchrophasor data for early event detection: Linearized analysis," *IEEE Trans. Power Syst.*, vol. 29, no. 6, pp. 2784–2794, 2014, doi: 10.1109/TPWRS.2014.2316476.

[64] "TAMUQ Research Computing Policies - Research Computing @ TAMUQ." https://rc-docs.qatar.tamu.edu/index.php/Main_Page (accessed Jan. 11, 2021).

[65] S. Fan and R. J. Hyndman, "Short-term load forecasting based on a semi-

parametric additive model," *IEEE Trans. Power Syst.*, vol. 27, no. 1, pp. 134–141, 2012, doi: 10.1109/TPWRS.2011.2162082.

[66] S. Kulkarni, S. P. Simon, and K. Sundareswaran, "A spiking neural network (SNN) forecast engine for short-term electrical load forecasting," *Appl. Soft Comput. J.*, vol. 13, no. 8, pp. 3628–3635, 2013, doi: 10.1016/j.asoc.2013.04.007.

[67] C. Cecati, J. Kolbusz, P. Rózycki, P. Siano, and B. M. Wilamowski, "A Novel RBF Training Algorithm for Short-Term Electric Load Forecasting and Comparative Studies," *IEEE Trans. Ind. Electron.*, vol. 62, no. 10, pp. 6519–6529, Oct. 2015, doi: 10.1109/TIE.2015.2424399.

[68] A. Lahouar and J. Ben Hadj Slama, "Day-ahead load forecast using random forest and expert input selection," *Energy Convers. Manag.*, vol. 103, pp. 1040–1051, 2015, doi: 10.1016/j.enconman.2015.07.041.

[69] © [2020] IEEE. Reprinted, with permission, from, A. Zainab, S. S. Refaat, D. Syed, A. Ghrayeb, and H. Abu-Rub, "Faulted Line Identification and Localization in Power System using Machine Learning Techniques," in *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*, Dec. 2019, pp. 2975–2981, doi: 10.1109/BigData47090.2019.9006377.

[70] S. Li, P. Wang, and L. Goel, "A novel wavelet-based ensemble method for short-term load forecasting with hybrid neural networks and feature selection," *IEEE Trans. Power Syst.*, vol. 31, no. 3, pp. 1788–1798, May 2016, doi: 10.1109/TPWRS.2015.2438322.

[71] G. Suo, L. Song, Y. Dou, and Z. Cui, "Multi-dimensional short-term load forecasting based on XGBoost and fireworks algorithm," *Proc. - 2019 18th Int. Symp. Distrib. Comput. Appl. Bus. Eng. Sci. DCABES 2019*, pp. 245–248, 2019, doi: 10.1109/DCABES48411.2019.00068.

[72] © [2019] IEEE. Reprinted, with permission, from, D. Syed, S. S. Refaat, H. Abu-

Rub, O. Bouhali, A. Zainab, and L. Xie, “Averaging Ensembles Model for Forecasting of Short-term Load in Smart Grids,” in *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019, Los Angeles, CA, USA*, Dec. 2019, pp. 2931–2938, doi: 10.1109/BigData47090.2019.9006183.

[73] H. Zheng, N. Jin, C. Ji, Z. Xiong, and K. Li, “Analysis technology and typical scenario application of electricity big data of power consumers,” *Dianwang Jishu/Power Syst. Technol.*, vol. 39, no. 11, pp. 3147–3152, Nov. 2015, doi: 10.13335/j.1000-3673.pst.2015.11.020.

[74] D. Syed, A. Zainab, S. S. Refaat, H. Abu-Rub, and O. Bouhali, “Smart Grid Big Data Analytics: Survey of Technologies, Techniques, and Applications,” *IEEE Access*, pp. 1–1, Nov. 2020, doi: 10.1109/access.2020.3041178.

[75] Y. Gao, Y. Fang, H. Dong, and Y. Kong, “A Multifactorial Framework for Short-Term Load Forecasting System as Well as the Jinan’s Case Study,” *IEEE Access*, vol. 8, pp. 203086–203096, Nov. 2020, doi: 10.1109/access.2020.3036675.

[76] J. Li, Y. Zhong, and X. Zhang, “A Scheduling Method of Moldable Parallel Tasks Considering Speedup and System Load on the Cloud,” *IEEE Access*, vol. 7, pp. 86145–86156, 2019, doi: 10.1109/ACCESS.2019.2925429.

[77] S. K. Abeykoon, M. Lin, and K. K. Van Dam, “Parallelizing x-ray photon correlation spectroscopy software tools using python multiprocessing,” *2017 New York Sci. Data Summit, NYSDS 2017 - Proc.*, 2017, doi: 10.1109/NYSDS.2017.8085042.

[78] © [2020] IEEE. Reprinted, with permission, from, D. Syed, S. S. Refaat, and H. Abu-rub, “Performance Evaluation of Distributed Machine Learning for Load Forecasting in Smart Grids,” in *Cubernetics and Informatics 2020*, 2020, p. In Publishing.

[79] A. Ahmad, N. Javaid, M. Guizani, N. Alrajeh, and Z. A. Khan, “An Accurate and

Fast Converging Short-Term Load Forecasting Model for Industrial Applications in a Smart Grid," *IEEE Trans. Ind. Informatics*, vol. 13, no. 5, pp. 2587–2596, 2017, doi: 10.1109/TII.2016.2638322.

[80] W. Jiang, H. Tang, L. Wu, H. Huang, and H. Qi, "Parallel processing of probabilistic models-based power supply unit mid-term load forecasting with apache spark," *IEEE Access*, vol. 7, pp. 7588–7598, 2019, doi: 10.1109/ACCESS.2018.2890339.

[81] W. Jiang, H. Tang, L. Wu, H. Huang, and H. Qi, "Parallel processing of probabilistic models-based power supply unit mid-term load forecasting with apache spark," *IEEE Access*, vol. 7, pp. 7588–7598, 2019, doi: 10.1109/ACCESS.2018.2890339.

[82] P. Prettenhofer,) Datarobot, and G. Louppe, "Gradient Boosted Regression Trees scikit Motivation Motivation," 2014. Accessed: Apr. 21, 2020. [Online]. Available: https://orbi.uliege.be/handle/2268/163521.

[83] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, "Machine Learning with Big Data: Challenges and Approaches," *IEEE Access*, vol. 5, no. 1, pp. 7776–7797, 2017, doi: 10.1109/ACCESS.2017.2696365.

[84] Y. Tang, Z. Xu, and Y. Zhuang, "Bayesian network structure learning from big data: A reservoir sampling based ensemble method," in *International Conference on Database Systems for Advanced Applications, Dallas, Texas, USA*, 2016, vol. 9645, pp. 209–222, doi: 10.1007/978-3-319-32055-7_18.

[85] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.

[86] P. Mika, "Flink: Semantic Web technology for the extraction and analysis of social networks," *Web Semant.*, vol. 3, no. 2–3, pp. 211–223, 2005, doi:

10.1016/j.websem.2005.05.006.

[87] A. Baldominos, E. Albacete, Y. Saez, and P. Isasi, “A scalable machine learning online service for big data real-time analysis,” in *IEEE Symposium on Computational Intelligence in Big Data (CIBD), Orlando, FL, USA*, 2014, pp. 1–8, doi: 10.1109/CIBD.2014.7011537.

[88] Y. Zhang, S. Chen, Q. Wang, and G. Yu, “I2MapReduce: Incremental mapreduce for mining evolving big data,” *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1906–1919, 2016, doi: 10.1109/TKDE.2015.2397438.

[89] M. Zaharia *et al.*, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in *Proceedings of NSDI 2012: 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012, pp. 15–28.

[90] N. Bharill, A. Tiwari, and A. Malviya, “Fuzzy Based Scalable Clustering Algorithms for Handling Big Data Using Apache Spark,” *IEEE Trans. Big Data*, vol. 2, no. 4, pp. 339–352, 2016, doi: 10.1109/tbdata.2016.2622288.

[91] V. K. Vavilapalli *et al.*, “Apache hadoop YARN: Yet another resource negotiator,” in *Proceedings of the 4th Annual Symposium on Cloud Computing, SoCC 2013, Santa Clara, CA, USA*, 2013, vol. 13, pp. 1–16, doi: 10.1145/2523616.2523633.

[92] B. Hindman *et al.*, “Mesos: A platform for fine-grained resource sharing in the data center,” *Networked Syst. Des. Implement.*, vol. 11, no. 2011, pp. 295–308, 2011, Accessed: Dec. 09, 2020. [Online]. Available: https://www.usenix.org/event/nsdi11/tech/full_papers/Hindman.pdf.

[93] T. White, *Hadoop: The definitive guide*, Third ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2012.

[94] L. George, *HBase: the definitive guide: random access to your planet-sized data*, First ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2011.

[95] Z. Hu, D. Li, and D. Guo, “Balance resource allocation for spark jobs based on prediction of the optimal resource,” *Tsinghua Sci. Technol.*, vol. 25, no. 4, pp. 487–497, 2020, doi: 10.26599/TST.2019.9010054.

[96] R. E. Edwards, J. New, and L. E. Parker, “Predicting future hourly residential electrical consumption: A machine learning case study,” *Energy Build.*, vol. 49, no. 1, pp. 591–603, Jun. 2012, doi: 10.1016/j.enbuild.2012.03.010.

[97] H. Shi, M. Xu, and R. Li, “Deep Learning for Household Load Forecasting-A Novel Pooling Deep RNN,” *IEEE Trans. Smart Grid*, vol. 9, no. 5, pp. 5271–5280, 2018, doi: 10.1109/TSG.2017.2686012.

[98] H. Aprillia, H.-T. Yang, and C.-M. Huang, “Statistical Load Forecasting Using Optimal Quantile Regression Random Forest and Risk Assessment Index,” *IEEE Trans. Smart Grid (Early Access)*, pp. 1–1, Oct. 2020, doi: 10.1109/tsg.2020.3034194.

[99] “Classification and regression - Spark 3.0.1 Documentation.” https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-trees (accessed Nov. 26, 2020).

[100] X. Meng *et al.*, “Mllib: Machine learning in apache spark,” *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[101] M. Armbrust *et al.*, “Spark SQL: Relational data processing in spark,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Melbourne Victoria Australia*, May 2015, pp. 1383–1394, doi: 10.1145/2723372.2742797.

[102] M. Manivannan, B. Najafi, and F. Rinaldi, “Machine Learning-Based Short-Term Prediction of Air-Conditioning Load through Smart Meter Analytics,” *Energies*, vol. 10, no. 11, p. 1905, Nov. 2017, doi: 10.3390/en10111905.

[103] A. Cutler, D. R. Cutler, and J. R. Stevens, “Random Forests,” *Ensemble Mach.*

Learn., pp. 157–175, 2012, doi: 10.1007/978-1-4419-9326-7_5.

[104] "STAR Project - Iberdrola." https://www.iberdrola.com/about-us/lines-business/flagship-projects/star-project (accessed Jan. 11, 2021).

[105] T. Hong, J. Wilson, and J. Xie, "Long term probabilistic load forecasting and normalization with hourly information," *IEEE Trans. Smart Grid*, vol. 5, no. 1, pp. 456–462, 2014, doi: 10.1109/TSG.2013.2274373.

[106] M. Lewis-Beck, A. S.-P. Analysis, and undefined 1990, "The R-squared: Some straight talk," *cambridge.org*, 2021, doi: 10.1093/pan/2.1.153.

[107] B. Parhami. Introduction to Parallel Processing: Algorithms and Architectures. Kluwer Academic, 2002.

[108] S. Sepasi, E. Reihani, A. M. Howlader, L. R. Roose, and M. M. Matsuura, "Very short term load forecasting of a distribution system with high PV penetration," *Renew. Energy*, vol. 106, pp. 142–148, Jun. 2017, doi: 10.1016/j.renene.2017.01.019.

[109] J.L. Gustafson, "Reevaluating Amdahl's Law," Comm. ACM, May 1988, pp. 532-533

[110] A. Bifet *et al.*, "Extremely fast decision tree mining for evolving data streams," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2017, vol. Part F1296, pp. 1733–1742, doi: 10.1145/3097983.3098139.

[111] T. M. Mitchell, "Machine Learning and Data Mining," *Commun. ACM*, vol. 42, no. 11, pp. 30–36, Nov. 1999, doi: 10.1145/319382.319388.

[112] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable κ-means++," 2012. doi: 10.14778/2180912.2180915.

[113] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation

of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, no. C, pp. 53–65, Nov. 1987, doi: 10.1016/0377-0427(87)90125-7.

[114] D. Syed *et al.*, "Deep Learning-Based Short-Term Load Forecasting Approach in Smart Grid With Clustering and Consumption Pattern Recognition," *IEEE Access*, vol. 9, pp. 54992–55008, Apr. 2021, doi: 10.1109/access.2021.3071654.

APPENDIX

A. Published/Accepted Journal Papers

- Zainab, A., Ghrayeb, A., Abu-Rub, H., Refaat, S. S., & Bouhali, O. (2021). Distributed tree-based machine learning for short-term load forecasting with apache spark. IEEE Access, 9, 57372-57384.
- Zainab, A., Syed, D., Ghrayeb, A., Abu-Rub, H., Refaat, S. S., Houchati, M., ... & Lopez, S. B. (2021). A multiprocessing-based sensitivity analysis of machine learning algorithms for load forecasting of electric power distribution system. IEEE Access, 9, 31684-31694.
- Zainab, A., Ghrayeb, A., Syed, D., Abu-Rub, H., Refaat, S. S., & Bouhali, O. (2021). Big Data Management in Smart Grids: Technologies and Challenges. IEEE Access, 9, 73046-73059.
- Syed, D., Zainab, A., Ghrayeb, A., Refaat, S. S., Abu-Rub, H., & Bouhali, O. (2020). Smart grid big data analytics: Survey of technologies, techniques, and applications. IEEE Access, 9, 59564-59585.
- Zainab A, S. Refaat S, Bouhali O. Ensemble-Based Spam Detection in Smart Home IoT Devices Time Series Data Using Machine Learning Techniques. Information. 2020; 11(7):344. https://doi.org/10.3390/info11070344

B. Published/accepted conference papers

- A. Zainab, S. S. Refaat, D. Syed, A. Ghrayeb and H. Abu-Rub, "Faulted Line Identification and Localization in Power System using Machine Learning Techniques," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 2975-2981, doi: 10.1109/BigData47090.2019.9006377.
- Ameema Zainab, Ali Ghrayeb, Mahdi Houchati, Shady S. Refaat, Haitham Abu-Rub "Performance Evaluation of Tree-based Models for Big Data Load Forecasting using

Randomized Hyperparameter Tuning" In 2020 IEEE International Conference on Big Data (Big Data). IEEE, 2020. (Accepted)

- D. Syed, S. S. Refaat, H. Abu-Rub, O. Bouhali, A. Zainab and L. Xie, "Averaging Ensembles Model for Forecasting of Short-term Load in Smart Grids," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 2931-2938, doi: 10.1109/BigData47090.2019.9006183.
- A. Zainab, S. S. Refaat, H. Abu-Rub and O. Bouhali, "Distributed Computing for Smart Meter Data Management for Electrical Utility Applications," 2020 Cybernetics & Informatics (K&I), Velke Karlovice, Czech Republic, 2020, pp. 1-6, doi: 10.1109/KI48306.2020.9039899.

C. Submitted Journal papers (under review)

- Ameema Zainab, Dabeeruddin Syed, Ali Ghrayeb, Haitham Abu-Rub, Shady S. Refaat, Othmane Bouhali, Mahdi Houchati, “Estimating Number of Clusters Utilizing Optimized Scheduling of Parallel Jobs to Manage Smart Grid Big Data” in IEEE Systems Journal [Submitted, Under Review].
- Dabeeruddin Syed, Othmane Bouhali, Shady S. Refaat, Ameema Zainab, Haitham Abu-Rub, “Enhancement of the Performances of Cross-model Power Forecasting in Smarts Grids using Transfer Learning”. In IEEE Systems Journal [Submitted, Under Review].

D. Submitted Conference papers (under review)

- Dabeeruddin Syed, Haitham Abu-Rub, Ameema Zainab, Mahdi Houchati, Othmane Bouhali, Ali Ghrayeb, and Shady S. Refaat. “Investigation on Optimizing Cost Function to Penalize Underestimation of Load Demand through Deep Learning Modeling”. In 47th Annual Conference of the IEEE Industrial Electronics Society [Submitted, Under Review].