RESIDENTIAL AREA ENERGY CONSUMPTION BIG DATA ANALYTICS AND VISUALIZATION

by

Ragini Gupta

A Thesis presented to the Faculty of the
American University of Sharjah
College of Engineering
In Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates

June 2018

Approval Signatures

We, the undersigned, approve the Master's Thesis of Ragini Gupta.
Thesis Title: Residential Area Energy Consumption Big Data Analytics and Visualization.

Signature	**Date of Signature** (dd/mm/yyyy)
_____	_____
Dr. Abdulrehman Al-Ali Professor, Department of Computer Science and Engineering Thesis Advisor	
_____	_____
Dr. Imran Zualkernan Associate Professor, Department of Computer Science and Engineering Thesis Co-Advisor	
_____	_____
Dr. Rana Ejaz Ahmed Associate Professor, Department of Computer Science and Engineering Thesis Committee Member	
_____	_____
Dr. Lotfi Romdhane Professor, Department of Mechatronics Engineering Graduate Program Thesis Committee Member	
_____	_____
Dr. Fadi Aloul Head, Department of Computer Science and Engineering	
_____	_____
Dr. Ghaleb A.Husseini Associate Dean for Graduate Affairs and Research	
_____	_____
Dr. Richard Schoephoerster Dean, College of Engineering	
_____	_____
Dr. Mohamed El-Tarhuni Vice Provost for Graduate Studies	

Acknowledgement

First and foremost, I would like to express my heartfelt gratitude to my advisor and mentor, Dr. Abdulrehman Al-Ali for his immense guidance, support, and motivation throughout the course of my thesis. I am truly grateful for his unwavering support and encouragement. Without his continual enthusiasm in the research interest, persistent help and guidance, this work would not have been completed.

I would like to extend my thanks to my co-advisor, Dr. Imran Zualkernan for providing knowledge and assistance whenever I needed help. His guidance into the world of big data has been a significant input during this work.

I want to thank Dr. Rana Ahmed and Dr. Lotfi Romdhane for serving as the examinee committee and their informative feedback and guidance.

I would like to thank the professors at the Computer Engineering department of American University of Sharjah who have taught me during the course of my undergraduate and graduate program for their dignified advices and help.

I would like to thank the Graduate Program at American University of Sharjah for offering me the Graduate Teaching Assistantship during the course of my master's program.

I must express my profound gratitude to my friend, philosopher, and guide my father and my mother for believing in me and nurturing me with unconditional love, support, and motivation throughout the years.

Dedication

To my brother, who is my best friend and pillar of strength through thick and thin..

To my advisor, guide and mentor for the past six years, who has continually motivated me in my determination to find and realize my potential, and has been a constant source of knowledge and inspiration..

Abstract

As Internet of Things (IoT) technology and open source file distributed system applications are evolving, home appliances can be monitored and controlled via an IoT-based home gateway. These gateways collect energy consumption from home appliances and hence create a large amount of data. Due to the large amount of data being generated, utility companies require platforms that enable them to store, process, analyze, visualize, and monetize the energy consumption data, and to gain meaningful insights into load profiles. This thesis proposes a residential area smart energy management system that enables home owners and utilities to monitor consumption patterns of each home, community, state, and country. Using an open source file distributed file system tools, home owners can monitor their home appliances energy consumption on a periodic basis. Additionally, utilities can also monitor the neighborhoods, communities, states, and country's consumption. The architecture was tested to process data from one million smart meters. This data was synthetically generated based on one year of real consumption data from a home. The big data was stored in a Hadoop cluster of four nodes. Dimensional modeling was used to develop benchmarking queries to create a real time dashboard consisting of charts, graphs, and reports for home owners and utilities. Both Spark and Hive were used to implement the benchmarking queries and it was found that Spark outperformed Hive in terms of latency and processor throughput. Spark's average latency was fifteen minutes with an average throughput of 2400 MBps while Hive's average latency was thirty-four minutes with an average throughput of 2200 MBps for processing one million smart meters in a four nodes cluster. To validate the proposed system outcomes, the results were compared with existing proprietary tools such as IBM's TimeSeries and relational database management systems. Spark and Hive have an intermediate performance in comparison to IBM's proprietary tool and relational database management system. The results demonstrate that the proposed solution can be utilized to provide energy data consumption visualization for consumer and utility provider stakeholders, while implementing Spark as the backend processing engine for low latency, performance gain, and a high throughput.

Search Terms: ***Internet of Things; Big data; Hadoop; Smart energy management system; Spark; Hive***

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

This chapter provides an overview of big data and its relevance in the growing Smart Grid energy sector.

1.1. Big Data in a Smart Grid

Electrical energy and environment have been observed as the mainspring for the growing energy economy. Attaining a better quality of life requires a high quality electricity generation, transmission, distribution, consumption, and management. There are many challenges currently faced by the electric power systems such as increased cost of fossil fuels, increased carbon footprint, aging equipment, energy resources security, environmental impacts, and inability to accommodate new trending technologies. Each year very large volumes of data gets collected in the home energy sector due to continuous application of sensors, wireless networks, communication networks, and power distribution outlets. Due to these reasons, smart energy technologies and green environment have been recognized as a fundamental research priority in many research centers. Motivated by the aforementioned reasons, the energy sector is deviating towards the era of big data in Smart Grids. In order to achieve valuable insights of this accumulated energy big data, big data driven smart energy management technologies have become an increasing trend. The future smart energy management systems should be able to seamlessly accommodate data from the renewable and non-renewable energy generation units in residential, business, and industrial networks. It should optimize energy consumption for air conditioning, lighting, heating and cooling systems in order to ensure that the electricity is consumed only when needed. Such smart energy management system will provide a remote access to the utility providers in order to view their power generation pattern in close relation to the clients' electricity consumption behavior. For example, the streaming data from residential areas will be visually represented in reports that will enable home owners and utilities to monitor the supply and meet the demand to ensure that the rapid changes in demand are anticipated beforehand and upgrade the power network efficiently. Significant research efforts are needed in order to recognize this vision.

Big Data in a smart grid comprises of any complex, diverse form of structured and unstructured data in large massive volume. There are five distinguished characteristics of Big Data (5Vs) [1-2]:

- **Volume:** Increasing volumes of datasets in size of petabytes and terabytes.
- **Variety:** Different types and formats of datasets
- **Velocity:** The rate of data generation which is being generated at an alarming rate.
- **Value:** Data has the potential to be mined to generate and value and meaningful results.
- **Veracity:** The inconsistencies and uncertainties (heterogeneity) in the data.

For a Smart Grid system, big data sources are residential, building, factories, power plants, renewable energy resources, electric vehicles, and environmental events as shown in Figure 1.1. Figure 1.1 demonstrates a lifecycle for Smart Grid initiating from Data Generation to Data Analytics at the top layer.

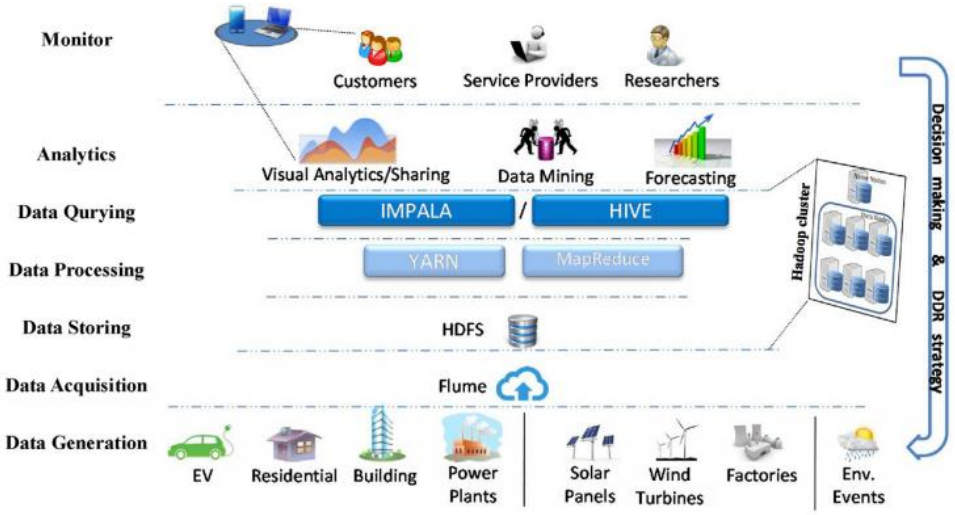

Figure 1.1: Lifecycle of smart grid data from data generation to data analytics with a learn and response loop [3]

Smart Grid has the potential to capture a massive volume of data through electrical and communication networks which can be harnessed to make strategic control decisions. The big data of a smart grid can be acquired, stored, processed, analyzed, visualized and monetized in different ways. When big data is stored and processed, additional dimensions come into action, such as management, security,

cost and governance. The smart grid data requires effective analytics, management in order to enhance grid reliability, operational efficiency and meet the increasing consumer demand with the help of information exchange and monitor in real-time.

The different sources of energy data that contribute to forming big data in a smart grid are elaborated in [3]. The residential units play a significant role in Smart Grid. The smart meters installation numbers report smart devices consumption in each house and its profound impact on the smart grid.

1.2. Research Problem

Each day smart meters collect power consumption data generated from residential areas home appliances. These produce large volumes of data flow in the size of petabytes and terabytes. This massive volume of big data cannot be handled by the conventional Relational Database Management Systems (RDMS) in Utility data centers. The staggering rate of growth in smart home devices enabled with Internet of Things (IoT) technology, and the need to perform data analytics on the captured datasets has challenged the use of utility data centers data management capability such as RDMS. Open source big data platforms such as Apache Hadoop can be utilized to assist in storage and parallel processing of this big data utilizing Map Reduce algorithm (MRA). MRA is a solution that not only helped in the data aggregation but was also useful in gaining meaningful insights from the collected information. Hadoop storage popularly known as, Hadoop Distributed File System Storage (HDFS) and RDMS serve the same roles in terms of collection, storage, and interpretation of collected datasets. However, the two platforms are very different in terms of the 'type' of data each platform handles. RDMS focuses on small to medium scale well-structured data by storing tables in the form of rows and columns with primary keys and foreign keys. Different query languages such as SQL can be used in a RDMS to access, retrieve, store, process, manipulate, and interpret data. On the other hand, HDFS concentrates on semi structured and unstructured data such as videos, audio files, and text files and different file formats such as XML, JSON, etc. RDMS is not powerful enough when it comes to horizontal scalability and delivers results with high latency with addition of more storage systems and/or CPUs. On the contrary, Hadoop's infrastructure is of distributed file system with clusters of many systems in horizontal scalability.

Thus, there were the following major challenges in using RDMS that gave birth to Hadoop framework:

- Large amount of unstructured and semi structured data adding to the storage and replication cost.
- Parallel Processing and horizontal scaling.
- As the data grows vertically, there is an increasing query performance related problems with RDMS due to high latency with the large number of rows in a table.
- The traditional RDMS cannot be utilized in processing requests from customers for monitoring real time usage data.

Table 1.1 demonstrates the major differences between a RDMS and Hadoop framework.

Table 1.1: Difference between RDMS and Hadoop HDFS

Characteristics	Hadoop Distributed File System (HDFS)	Relational Database Management System (RDMS)
Basic Description	Distributed file-based system to store data across a range of nodes allowing parallel processing	Traditional column oriented database for transactional systems, operational DBs, and archiving
Storage Capability	Store TBs and PBs of data	Store GBs of data
Processing Type	Supports batch processing, in-memory	Supports batch processing and interactive process
Best for Application	Does not allow UPDATION, follows WORM (Write Once Read Many) paradigm	Allows UPDATION, multiple writes and reads
Table Schema Requirement	Schemas not required	Schemas must be defined
Type of Data Support	Supports multi types of data (structured and non-structured)	Only structured data
Throughput and Latency	High throughput but high latency also as data is distributed across nodes in a network	Throughput depends on data size. Low latency as data is stored in a centralized server.

Hadoop is a framework that allows distributed processing of huge data sets across clusters which hosts thousands of computer nodes using simple programming models. It brings together different components under one umbrella to form a scalable system. Hadoop provides a platform for storage and analysis on massive scale of data. The Current Apache Hadoop ecosystem consists of the Hadoop Kernel, MapReduce, HDFS and various components like Apache Hive, Base, Pig, Sqoop, and Zookeeper [72]. The core of Hadoop consists of a storage part, known as Hadoop Distributed file

system (HDFS), and the processing part is called Map Reduce. Hadoop is not really a type of database but a software ecosystem that provides a platform for massive parallel computing.

1.3. Thesis Statement

While there are diverse sources of big data in a smart grid, this research focuses on the smart meter data in residential areas. This thesis emphasizes on the performance evaluation of Hadoop with respect to data querying using two distributed computing frameworks; Apache Spark and Apache Hive [1]. Apache Spark is an in-memory, open source fast processing engine on top of Hadoop for big data analytics and machine learning. Spark Queries are executed as individual jobs that runs in multiple stages, each stage comprising of multiple tasks utilizing Directed Acyclic Graphs (DAG). Apache Hive is an open source distributed batch processing engine that facilitates map reduce programming model utilizing the map reduce engine. Both frameworks will be deployed for the same data querying and data reporting smart home uses cases in order to analyze which engine outperforms in terms of latency and throughput. Both processing engines will share a unified data storage logical layer of Hadoop distributed file system (HDFS) enabling parallel storage and processing across several Hadoop cluster nodes. To analyze the performance between the two engines, the cluster will be scaled from one node to four nodes, with each cluster-size processing logarithmic batch of files from ten to one million smart meters. This thesis will also provide a solution for storing, processing, analyzing and visualizing smart meter data that will benefit the consumers and utility providers in gaining insights from the data and discovering significant consumption patterns for each housing unit. Additionally, big data and analytics can help the utility providers to uncover anomalies at energy network that can help to predict and prevent power outages and help in planning. The proposed solution can help home owners, community owners, state owner, and country owner to efficiently monitor and manage the energy supply-demand. This will improve the power consumption, reduce the power outages, reduce the operational cost, and improve the grid performance at large. Stakeholders can access the smart meter real time data through a web portal and visualize the consumption pattern of home appliances and the grid performance.

Chapter 2. Background and Literature Review

This chapter describes the theoretical background and the main motivation that fostered the current research. A number of applications of Smart Grid in the energy sector utilizing big data solutions can be found in the literature, wherein the main objective is to process large volumes of data and estimate the particular load profiles of consumers depending on different contexts. Smart Grids produce gigabytes of data which leads to several issues due to storage reasons. Thus, it becomes instrumental to map this Gigabytes of data into a few Kilobytes or Megabytes, which is enabled by utilizing the technique of big data for efficient storage. Different researches and surveys highlight the significance of big data for better energy management and control decisions in distributed systems of smart grid [4-6].

For instance, in [4] the authors emphasize the different benefits of big data in power distribution smart grids such as reduced electric grid network losses, large scale optimization of distributed energy system resources, maintain reliable power supply during peak load hours and critical loads, and determining the root causes of failure. However, the authors focus on different criteria that should be adopted in selecting the best tool for big data management. There are many challenges involved in power distribution systems to extract relevant information from the huge data volume and foster online decision making. Some of these challenges include the data challenges corresponding to data scalability, velocity, veracity, volume, volatility and data quality. The big data tool should have the ability to handle large volumes of structured, semi structured and unstructured data. Next are the processing and analytical challenges since the tool should be able to identify the relevant data sources to extract useful insights and analyze information in order to satisfy the smart grid objectives. Other challenges involve the big data security and the cost of deploying a big data system as it is crucial to use the right and cost effective infrastructure for big data.

Alternatively, in [6] the authors propose a four layered IoT based architecture that utilizes big data techniques in the Management layer to obtain energy efficiency in the buildings of a smart city. The paper focuses on different Big Data analytic techniques that can be used for intelligent decision making process in energy management. Big data analytic algorithms such as Artificial Neural Networks, and

Support Vector Machines, can be used to predict energy requirements, particularly for HVAC units. The results obtained after developing the energy consumption predictive models for buildings showed that with the application of predictive models there was a 23% of energy savings per month on HVAC units without compromising on the consumer comfort.

In [6], authors provide a comprehensive study of big data driven smart energy management. Different sectors of energy management such as power generation side management, micro grid and renewable energy management, asset management, and demand side management (DSM) have been taken into study for the application of big data analytics techniques. The authors emphasize on the issues that need to be addressed in adoption of big data analytics techniques such as data processing and analysis. The traditional techniques such as data mining, machine learning, and statistical analysis may face several complexities in dealing with energy data. To enable smart energy management tasks, effective and efficient data mining analysis techniques are required to establish models and obtain simulation results that can be well interpreted for strategic decision making.

In response to the different requirements to be fulfilled by big data, many researches have been aligned towards using several big data frameworks such as Apache Hadoop and Apache Spark in energy domain. Authors in [8], performed an evaluation of cloud-based log file analysis using Apache Hadoop and Apache Spark. Both the frameworks were utilized to extend the Map Reduce programming model to process the log files generated as big data from the HTTP server. The key difference between the two ecosystems is that Hadoop performs on-the-disk data processing whereas Spark does that in-memory. Experiments were conducted to compare the two platforms in terms of parameters such as execution time, scalability, resource utilization, cost, and power consumption. For Scalability, the mean execution time of the two frameworks was evaluated with increasing size of log file and increase in number of node clusters. It was concluded that both Hadoop and Spark showed a similar pattern and supported great scalability with up to 1000 GB of data within 5 cluster nodes. For other performance indicators such as execution time, resource utilization and power consumption Apache Spark outperformed Hadoop. However, Spark needs a large amount of memory for caching the data. Additionally, the requirement to use the main memory for processing adds to the total cost of

infrastructure in case of Spark. Thus, the authors conclude that since Hadoop Map Reduce is oriented towards batch processing it is a more preferable and cost-effective framework for any big data processing application.

Several other researches in [8-10] have focused on mass log processing based on Hadoop Map Reduce model to achieve high scalability, reliability and better performance. The log files are first stored in HDFS and processed using Map Reduce model and Hadoop mining system. This provides the administrator to analyze the log data efficiently and be more decisive in taking decisions. The Map Reduce programming model provided an administrative monitoring system for problem identification and future trend prediction of the system.

The utility companies have deployed smart meters that can measure the energy consumption of water, gas, and electricity at regular intervals. These smart meters tend to generate a large volume of interval data that requires being stored, processed and analyzed. The utilities also run large, sophisticated systems that generate power and each grid comprises of smart sensors that can monitor the current, voltage, and other operating parameters. To achieve operating efficiency, the organization should monitor the data delivered by the sensors. Consequently, a big data solution can help to analyze power generation (demand) and power consumption (supply) data makes sense [11].

Many researches proposed different programming models using Map Reduce on Hadoop framework for transforming massive amounts of data on distributed systems. In [12] and [13], Hadoop Map Reduce framework is utilized where the Big Data generated as a result of periodic audit files is stored in Hadoop environment using map reduce. In [12], the audit files or the log files are continuously generated from Advanced Metering Infrastructure (AMI) installed in Smart Grids that corresponds to large bulk of data or Big Data. This data is stored in Hadoop Cluster nodes, called the Data Nodes which are controlled by the Name Node instances. Additionally, during simulation a Hadoop environment is created with a generator unit, solar panel unit, processor unit, industrial unit, power plant and industrial wind turbines which are deployed in the network. The data from each individual unit it stored on Hadoop HDFS. Each unit sends its log files to a specific allocated node for storage in HDFS. The log files generated from all units are applied with Map Reduce algorithm wherein each log file is moved into the mapper function from HDFS line by

line. The Mapper maps each entry into the equivalent <key, value> pair and generates small pieces of data. During the Reduce stage, all the output from the mapper section is accumulated and the reducer creates a new set of output and stores it in the HDFS. This solves the storage limitation issue. One of the major advantages of Hadoop Map Reduce programming model is that the algorithm is performed at the position of the data and the data storage and computations are synchronized on all nodes in the cluster, thus decreasing the overhead of network traffic involved in shifting data. This research was concluded with performance analysis based on parameters such as latency, packet delivery ratio, throughput, total energy consumption of all nodes in cluster, overall residual energy present in the nodes of Smart Grid in Hadoop configuration with respect to the packet size in bytes. Thus, Map Reduce algorithm proved instrumental in storing large Gigabytes of data efficiently by mapping it into a few kilobytes or Megabytes as the experimental results also showed great improvement in Hadoop Map Reduce framework.

In [13], the authors provide a comprehensive overview on using Hadoop Map Reduce in energy sector for analyzing momentary outage data and power theft. The authors compare the Hadoop Map Reduce framework with other data storage and analytics platforms such as Spark and SAP HANA. However, the latter cause an overhead on the machines as they are all in-memory platforms. Thus, the authors have recommended on using Hadoop for deploying safety analytics by assessing the meter temperature data and predicting the transformer or meter mishaps and fire before the event happens. Other factors that make Hadoop outperform other analytic platforms is its ability for large data processing on HDFS, processing data on the fly, distributed performance and faster access. This ensures Hadoop as one of the most cost effective scalable analytics platform available today.

More researches have been steered in the realm of smart meter data analytics using Hadoop HDFS in [14-18]. In [18], authors implemented an information integrated pipeline for collecting data from smart meters periodically and utilizing a scalable platform like Hadoop for processing and mining big data sets along with a web portal for end users to visualize analytical results. Additionally, Hive (an open source SQL-based distributed warehouse system) is used on top of Hadoop framework for off-line analytic queries in SQL like script language called, Hive –QL. The analytic queries are internally translated into Map Reduce jobs and the results are

stored into a Hive table, which is a data organizational structure in HDFS. From HDFS, the results are exported into PostgresSQL for interactive analytics and visualization via web application. Five categories of analytics modules were considered: load profiling of customers to study monthly and annual consumption variability of customers, pattern discovery and segmentation to perform k-means clustering of customers with a similar consumption pattern, load disaggregate to segregate energy consumption into temperature-dependent and temperature-independent loads, forecasting to predict the period of peak demands for energy utilities, and online anomaly detection to allow customers to detect their unusual consumption compared to their compared to their consumption history and/or neighborhood consumption. Several off-the-shelf analytics functions (such as percentile, mean, min, max and median) and algorithms (such as clustering and regression) were exported from Apache Common Math library and Spark streaming on top of Hadoop to provide analytic results in the form of graphs, charts and reports. The effectiveness and efficiency of this solution was tested extensively with real time data and synthetic data with successful results.

Similarly, [14] presents smart meter data as time series data utilizing open source tools on Hadoop and HBase for storage and analytics. HBase is an open source column-oriented store modelled on top of Hadoop framework for read/write access and storage of very large tables with millions of rows and columns on HDFS clusters. Smart meter data is tracked as time series data that is collected continuously but in short intervals of time and stored in HBase. The setup comprised of a fully distributed mode of Hadoop comprising of several nodes with different requirements based on scalability, reliability, and availability. HBase is utilized for big data storage which is in direct communication with the HDFS layer of Hadoop. An open source tool, OpenTSDB comes with web based UI that is setup to directly interact with HBase and retrieve data accumulated in HDFS. Using OpenTSDB, a request is sent to Hadoop with time stamp details and tags and the results are received in graphical or report format. OpenTSDB served as an efficient platform to generate graphs on the fly with an easy interaction to access data from Hadoop HDFS.

It is worth mentioning that most data centers and data warehouse systems are infamous energy consumers. Hadoop or other data centers whether real or virtual (in cloud) execute large computational jobs such as Map Reduce which is often very

energy exhaustive. However, to combat the energy management in Hadoop cluster nodes, Hadoop has evolved with state-of-the art resource management features that help in improving performance through dynamic resource control as described in [19] and [20]. In [19] the authors propose the energy-aware dynamic node management technology for Map Reduce jobs on Hadoop that ensure the switching ON/OFF the cluster nodes to reduce the energy consumption while meeting the Service Level Agreement (SLA). This is achieved by deploying YARN on top of Hadoop framework. With YARN scheduler, the time-varying workload is predicted based on the Map Reduce job history information. Based on this predictive workloads and average execution time of nodes, energy aware dynamic node management is used on YARN scheduler to assign the suited number of nodes for Map Reduce Tasks. The nodes that are retained in the idle state for long periods are eventually switched OFF automatically to save energy. Thus, this dynamic scaling-up / scaling-down capability in Hadoop utilizing Map Reduce YARN will help in adjusting the size of the cluster for big data applications in a specified period of time.

Different approaches were utilized in the context of performance analysis of Hadoop in terms of memory utilization, CPU usage, and scalability in [21], [22], [23]. In [22], HiBench stress test tool was deployed on Hadoop stack as a benchmark suite to generate a combination of synthetic and real world data. Different HiBench workloads of varying sizes were selected for memory usage tracing and scalability. Other parameters such as CPU utilization, memory usage, power consumption of name node and data nodes were evaluated using two different tools Cloudera manager and Datacenter manager Console. From the results, it was inferred that approximately 96GB of RAM was sufficient enough for the largest size of workload, i.e., 61,600 MB in Hadoop data nodes. Moreover, addition of larger memory had little or no improvement in the execution time for all workloads. The amount of memory usage for each workload was dependent on factors such as algorithm used in processing the data set and CPU utilization. In case of CPU intensive Map Reduce jobs such as word count, the execution time and power consumption of Hadoop data nodes incremented linearly with the increase in size of data set.

Besides Map Reduce models on Hadoop for Smart Energy Management, dimensional models utilizing OLAP (On-Line Analytics Processing) to analyze customer consumption data and deliver real time reporting with trend forecasting is

studied in [24],[25],[26]. For instance in [24], researchers propose a system framework comprising of data warehouse construction, ETL (Extraction-Transformation-Loading) process, and information representation in the context of electricity management of buildings. The framework incorporates multi-dimensional modelling for cube construction with the help of the designed star schema. This enables the aggregation of data aggregation at different levels and hierarchies concerned with monetizing energy consumption. On the side of end-users, a web applet UI is introduced to cater to aggregated data representation at different granularity levels in order to meet user requirements. Similarly, in [26] a data warehouse modelling system is implemented by developing a user-developed agent called EDWH agent that aggregated the collected sensor data and generates cubes. The data cubes are generated by applying one of the aggregate functions such as SUM, AVG, or COUNT or one of the OLAP operations once each day and stored in the Data Warehouse agent. The stored cubes are published to the Building Data cloud which leverages the data model and exposes it on dashboard in the form of reports and graphs. This dashboard visualization helps the users to identify areas of energy leaks or electricity usage pattern within their respective buildings. A similar experimental setup is deployed for OLAP modelling using a different data warehouse agent called RDF-DCV. The two platforms are evaluated against selected criteria such as storage size (latency), query execution time in generating reports, execution time in storage and generating cubes. From the results, EDWH outperformed the RDF-DCV data warehouse agent in all evaluation parameters.

Thus it can be stated that Hadoop plays an instrumental role in big data analytics due to two main reasons; HDFS and Map Reduce. HDFS provides a fault tolerant, scalable, and a reliable platform due to its distributed storage and processing capability along with automatic recovery across different cluster nodes. Similarly, the map reduce programming logic provides parallel processing of data across several nodes, thus reducing the execution time significantly and making the system more efficient. Due to its several advantages such as scalability, reliability, availability, and fault tolerance, Hadoop serves as an optimum platform for deploying it to analyze energy data and comprehend the users' consumption pattern and appraise the different factors which can directly or indirectly affect the consumption pattern of the users. On the other hand, OLAP processing model is also extensively used in the domain of

Smart Energy management for extensive consumption analysis, trend identification and demand prediction in big data. Real time information feedback to consumers on different levels of granularity (Home consumers to Energy utility providers) regarding consumption pattern and associated energy costs can help in redistribution and adjustment in Demand Response of energy with lower power consumption. Thus, both the computing models Map Reduce and Dimensional Modelling (OLAP) cater to the same type of business services to the end users but with different processing approaches. Data warehousing architecture utilizing OLAP is a traditional platform for modelling and processing data in contrast to the newer technologies of Hadoop Map Reduce. The primary objective of this research is to explore which computing model is more efficient for handling Energy Big Data in terms of the selected criteria as discussed later.

This chapter presented a survey of previous research where most of the work implemented small dataset in a few GBs with tailored modeling techniques and commercial database systems for storage. The processing models proposed have not been implemented in the domain of smart energy utilizing big data analytical techniques. The consumers, community, state, and country stakeholders are not informed about the data insights and analysis in real time. The conventional SQL querying continues to be the most popular query language for big data analysis. The Hadoop based SQL engines namely Hive and SparkSQL can be utilized for benchmarking study as well as to provide data analysis and visualization to consumers and utility providers.

Chapter 3. Methodology

This chapter is divided as follows: the first section defines a set of non-functional requirements that were deduced from the works presented in Chapter 2. The requirements are then used to construct a system definition, which is further presented in this section.

3.1. Functional and Non-Functional Requirements

To formulate a system definition and architecture for the proposed methodology, it is important to outline the functional and non-functional requirements of such a system. The functional requirements of a system are defined by the functionalities and technical characteristics that the system should provide. On the other hand, the nonfunctional requirements outline the qualities and the criteria that define the operation of the system. The system requirements are defined as follows:

- Creation of a Hadoop environment cluster that comprises of at least one master node and two slave nodes.
- Dynamic addition of slave nodes to an existing Hadoop cluster
- Deletion of slave nodes from an existing Hadoop cluster
- Automated load balancing of cluster machines
- Provide high availability and redundancy.
- Triggering automatic failover in the event of a node crash
- Provide a user-friendly interface to configure and manage Hadoop environment features, and to interact with the cluster scheduler.
- Provide an interface for graphical visualization to present the status of one home appliances consumption for stakeholders on different levels; home owner and utility providers on community, state, and national level.

The non-functional requirements of the system are as follows:

- A scalable distributed storage required so that the cluster is able to store and process up to one million smart meter data files.
- The nodes (commodity machines) should have specifications of at least 8GB RAM, between 500GB-1TB storage, and high-speed processors. These specifications are a critical part for Hadoop cluster planning that perform CPU intensive i.e. query workloads efficiently in the Hadoop environment.

- Ease of deployment and management for large volumes of data.
- The execution time (latency) of the queries should be as small as possible.
- Processor throughput should be as high as possible.
- Should provide an interactive platform for executing queries using disk caching and in-memory caching for map reduce programming model.
- The Hadoop framework should provide an intuitive navigation to end users.

3.2. System Definition

Figure 3.1 shows the system diagram in accordance to the requirements and the use cases for the proposed two architecture.

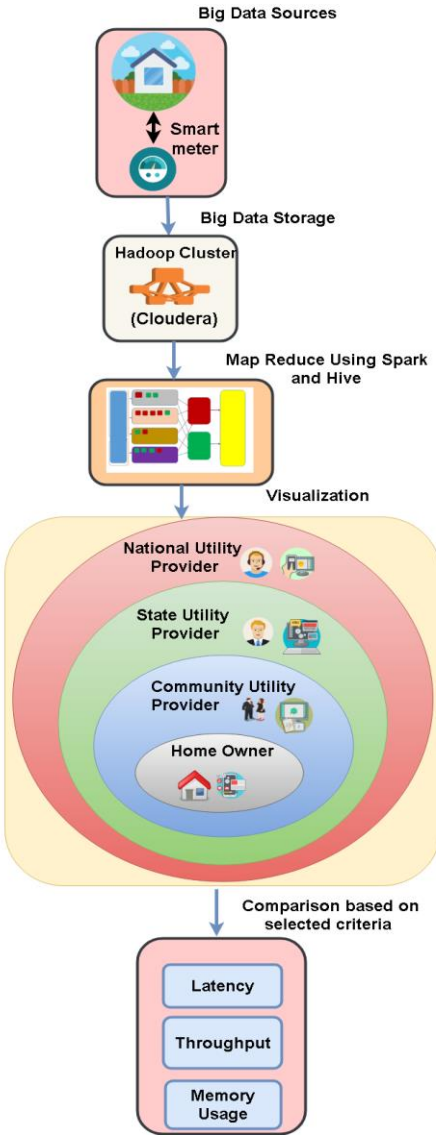

Figure 3.1: System diagram

The system consists of six stages:

- Big data sources: The big data sources will be aggregated from smart meters to obtain a large volume of data in the size of terabytes.
- Model OLAP Queries: OLAP refers to Online Analytic Processing representing multidimensional data structures that can be utilized for fast processing in business intelligence. It's primarily a data warehousing concept that provides a flexible way to make advanced analytics on multidimensional data. The smart meter data should be modeled as an OLAP cube with different dimensions of time, location, and home appliance. OLAP operations on the smart meter data cube will be applied. The dimensional modelling for OLAP and map reduce querying to generate the reporting data will be based on different use cases for each of the stakeholders concerning monitoring of energy consumption. Different use cases and business processes are incorporated on a national level to analyze the data on a four level hierarchy of stakeholders, i.e., the lowest level for the home owner followed by the community utility provider and the state utility provider, and the highest level for the national utility provider. The OLAP operations will be mapped to the equivalent SQL language queries to benchmark querying results on a big data platform such as Hadoop.
- Big Data Storage: The energy big data from smart meters will be accumulated and loaded on to Hadoop Distributed File System (HDFS) for efficient storage across cluster nodes and parallel processing.
- Big Data Processing: Two SQL processing engines namely, SparkSQL and Hive will be used for executing SQL queries on the data stored in HDFS storage layer. The two processing engines share a unified processing paradigms of map reduce tasks running in parallel across the cluster nodes. Spark utilizes in-memory computation for map reduce tasks while Hive utilizes disk map reduce processing.
- Visualization: Visualization capabilities will be provided from the smart meter data to consumers and utility providers on community, state, and national level. The visualization graphs and charts will allow the end users to analyze the consumption pattern of home appliances on a periodic basis. For

visualization, an interactive interface is provided by Hadoop application, Hue, to render query results in the form of graphs, charts, tabulated reports.

- Comparison based on selected criteria: Several performance analysis experiments will be conducted to evaluate the Hadoop based SQL processing engines in terms of latency and throughput. These experiments will also provide an insight into how well the two processing engines can scale up with the increase in dataset volume.

3.3. Data Generation

Smart Meter Energy data is collected for one residential unit from [27]. This data is the energy consumption in KWh that spans a period of one year (2014) and is collected every day for each home appliance of the house namely, Furnace, Cellar, Washing Machine, Fridge, Outlets, Cooking Range, Dishwasher, Lights, and Heater. The data is in a structured format. The data will be sampled to generate synthetic big data for one million houses using ARIMA modelling. Each file corresponds to a data size of 1.5MB summing up to the total size of 1.5TB for one million data files. This data spans a period of one year and is collected on an everyday basis from a home. There are several techniques to synthetically generate a dataset for one million smart meters; Arima Modelling [28] and Markov Chain Monte Carlo [29]. It was found that ARIMA modelling provides the best fit for simulating data for accurate representation of smart meter referenced dataset.

3.3.1. ARIMA modelling. The data obtained for one house is a time series data as it is a collection of power consumption values for different household devices achieved through repeated measurements over a period of one year. ARIMA stands for auto-regressive integrated moving average and is specified by these three order parameters: *(p, d, q)*. The process of fitting an ARIMA model is sometimes referred to as the Box-Jenkins method. The auto-regressive parameter *p* specifies the number of lags used in the model [28]. For example, ARIMA expression for Time Series Data Y_t can be expressed as:

$$Y_t = c + \emptyset_1 y_d + \emptyset_p y_d + \ldots + \theta_1 e_{t-1} + \theta_q e_{t-q} + e_t$$

Ø1, Ø2, Θ1, Θ2, are model parameters, y_d is differenced *d* times between current and previous value, e_t is previous error terms, and *c* is a constant. The'*d*' represents the degree of differencing in the integrated (I (*d*)) component. Differencing a series involves simply subtracting its current and previous values d times. Often, differencing is used to stabilize the series when the stationarity assumption is not met. For simulating time series data, we need to choose an optimal ARIMA model (*p, d, q*). For this, we used the auto.arima () function in R language as it searches through combinations of order parameters and picks the set of *p, d, q* values that optimizes model fit criteria. Auto.arima () function in R performs repeated tests from Likelihood Estimation (MLE). The parameter *d* is set to 0 or 1 if it improves the Akaike Information Criterion (AIC) value to estimate relative quality of statistical model for the new dataset w.r.t the old data [29]. Figure 3.2 is the flowchart for ARIMA algorithm.

According to UAE national statistics [30], the UAE population in 2008 was 4.1 million with a breakdown of Sharjah comprising of 17%, Abu Dhabi contributing to 36%, Dubai contributing to 32%, Ajman contributing to 5%, Umm AlQuain comprising of 1.1%, Ras AlKhaima contributing to 5%, and Fujairah consisting of 3% of the country's population. Using these emirates statistics, one million smart meter files were synthetically generated utilizing ARIMA simulation method. From the total smart meters data generated for a million smart meter homes, 170,000 smart meters data was generated for Sharjah, 360,000 smart meters data was generated for Abu Dhabi, 320,000 smart meters was generated for Dubai, 50,000 smart meters for Ajman, 11000 smart meters for Umm Al Quain, 50,000 smart meters for Ras Al Khaima and 30,000 smart meters for Fujairah. Additionally, the heater consumption in the synthetic data is considered similar to an AC unit for simulating smart meter tailored according to UAE. Four distinct communities were taken to generate data in each emirate. Table 3.1. Illustrates the data files generated following the UAE population statistics.

3.4. Identifying the Key Business Processes

Different business processes are incorporated on a national level to analyze the data on a four level hierarchy of stakeholders, i.e., the lowest level for the home owner followed by the community utility provider and the state utility provider, and the highest level for the national utility provider. An efficient energy management system should ensure monitoring of energy consumption from a holistic point that renders real time monitoring capabilities to home owners and divisions of utility providers. The two important business processes are considered for an efficient home energy management system.

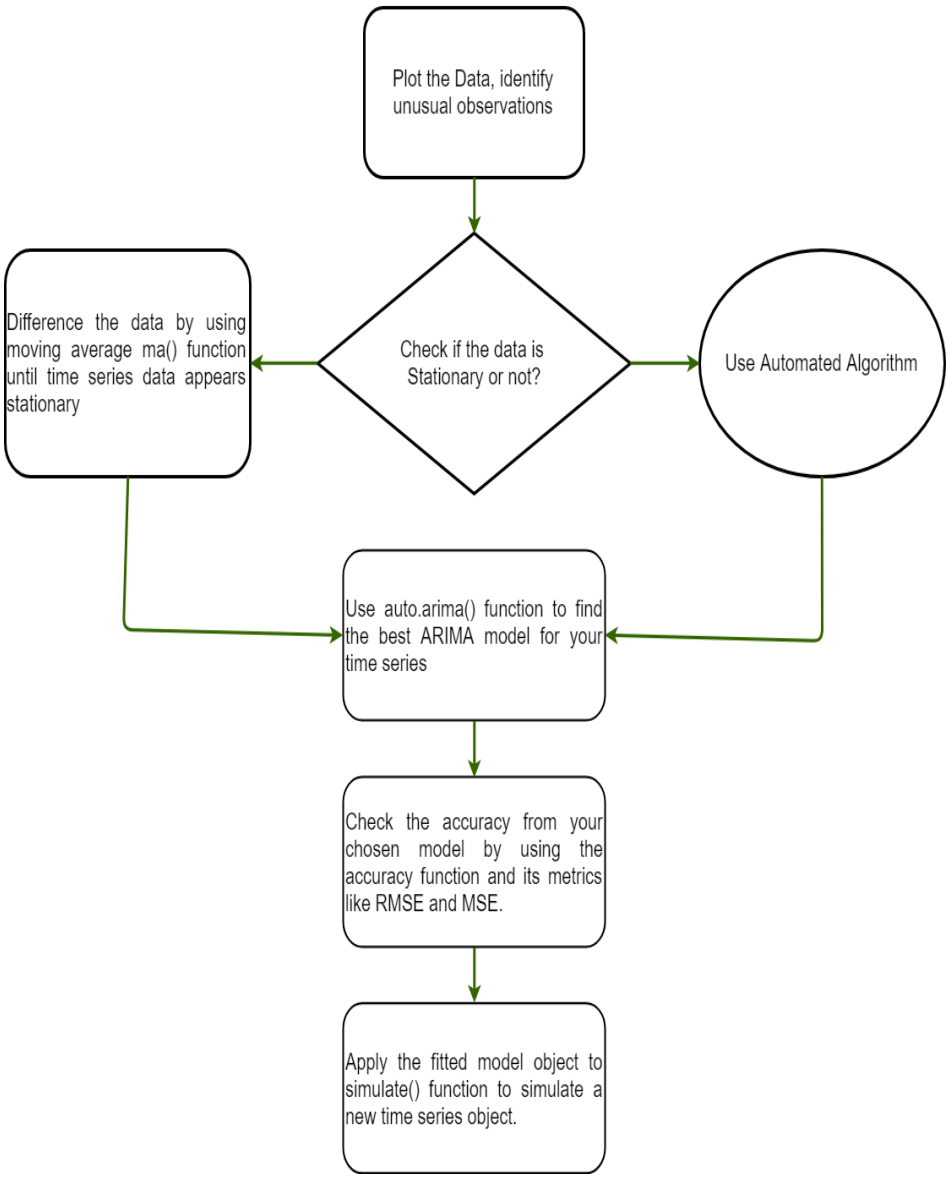

Figure 3.2: ARIMA flowchart

Table 3.1: Synthetically Generated UAE Smart Meters Data

Number of Smart Meters	Emirate	Community	Range
60000	Sharjah	Dasman	SH1-SH60000
80000	Sharjah	Maliha	SH60001-SH140000
20000	Sharjah	Rola	SH140001-SH16000
10000	Sharjah	UniversityCity	SH160001-SH170000
200000	Dubai	Mirdiff	DH170000-DH370000
80000	Dubai	JebelAli	DH370001-DH450000
30000	Dubai	Jumeirah	DH450001-DH480000
10000	Dubai	AlBarsha	DH480001-DH490000
180000	Abu Dhabi	Khalifa City	AUH490001-AUH670000
120000	Abu Dhabi	Baniyas	AUH670001-AUH790000
40000	Abu Dhabi	Saadiyat	AUH790001-AUH830000
20000	Abu Dhabi	Reem	AUH830001-AUH850000
10000	Ajman	Nakhil	AH850001-AH860000
20000	Ajman	Manama	AH860001-AH880000
15000	Ajman	Hamdiya	AH880001-AH895000
5000	Ajman	Rumailah	AH895001-AH900000
4000	UmmAlQuain	Duor	UH900001-UH904000
2000	UmmAlQuain	Sinniyah	UH904001-UH906000
3000	UmmAlQuain	Qaram	UH906001-UH909000
2000	UmmAlQuain	AlSow	UH909001-UH911000
20000	RasAlKhaima	Masafi	RH911001-RH931000
10000	RasAlKhaima	Khatt	RH931001-RH941000
10000	RasAlKhaima	Hamra	RH941001-RH951000
20000	RasAlKhaima	Rams	RH951001-RH971000
12000	Fujairah	Bidya	F971001-F983000
15000	Fujairah	Bidnah	F983001-F998000
2000	Fujairah	Qaram	F998001-F1000000
Total: 1,000,000 Smart Meters in UAE			

3.4.1. Energy consumption process for consumers. This business process caters to the monitoring services for home owners. For the consumers, access to the smart metering data is necessary in order to comprehend their own electricity demand. Smart Meters installed in residential units capture the electricity consumption of different devices and send it to central substations every day over a period of time. The efficiency of different devices in a house building can largely depend upon the time (month/week) of the year. Using different benchmarking techniques, ideal efficiency levels for a device can be computed and compared with specific cases which involve excessive energy consumption. Analytical techniques can help identify power consumption discrepancies across different home devices over a span of time.

3.4.2. Energy capacity planning process. This business process caters to the utility providers. There are four divisions of utility providers; Community Owner, State Owner, and Country Owner. The utility providers can collect smart meters data from all houses and analyze this large data to generate useful insights and make

efficient energy supply decisions based on the consumer usage pattern. They are eligible for monitoring total power consumption within each house, community, state, and country. This energy usage will be depicted through different types of graphs and timeline. Based on the data analyzed, peak time analysis as well as load scheduling events can be planned and executed by the Utility Providers. The aggregated energy consumption can be compared to the overall energy generation from the Central power station. This will assist the state jurisdiction to set climate goals, prioritize or implement energy strategies, and make data driven energy decisions.

3.5. Identifying the Use Cases for Stakeholders

Based on the above mentioned business four stakeholders (Consumer, Community Utility Provider, State Utility Provider, and National Utility provider) are deemed important for providing real time visualization on smart homes energy consumption. A detailed description of these stakeholders is discussed as below.

3.5.1. Consumer. A home owner is entitled for viewing the power consumption of all devices in his/her house with respect to time on a daily, weekly, monthly, and yearly basis. He/she can also evaluate the asset efficiency of a device and monitor the aggregate power consumption of all devices on a periodic basis.

3.5.2. Community energy utility provider. A community utility provider can monitor the aggregated power consumption of each household in his/her respective community. The analysis will help the community utility providers to identify trends in energy consumption of each household on a timely basis and the subsequent relationship between energy supply and demand. This information will be represented in the form of timeline graphs on a monthly, quarterly, and yearly basis. Using this analysis, the utility providers can discover the peak load hours and plan and execute load scheduling events accordingly.

3.5.3. State energy utility provider. A state utility provider can supervise the cumulative power consumption of all communities within his/her corresponding state on a monthly, quarterly, and yearly time period. They can adjust the energy prices at peak demand using multiple tariff rates dynamically.

3.5.4. National energy utility provider. A national utility provider is the highest level of authority in the hierarchy. They can observe and compare the aggregated power consumption of each state within the country on a periodic basis

(monthly, quarterly, annually). The cumulative power consumption from all states can be compared to the total power generation from the Central power station. This will help the utility providers to prioritize different energy saving strategies and execute data driven energy actions accordingly. Three granularities of time have been considered to perform different types of analysis on the data periodically; daily, weekly, monthly, and yearly.

3.6. Dimensional Model Construction

Data Warehouses are designed on the fundamentals of OLAP, whereas relational databases systems are modelled on the concept of OLTP (On-Line Transactional Processing). OLAP data querying tools allows the users to analyze different dimensions of the data such as time series, trend analysis, etc. OLTP models utilizes two-dimensional data in the form of rows and columns, whereas OLAP opens new avenues into looking up the data from more than two dimensions. OLAP supports filtering, slicing, sorting, dicing operations of the data structure. Figure 3.3 below demonstrates the visual representation of OLTP vs. OLAP data models. In an OLAP data warehouse, there is an aggregated data stored in multi-dimensional schemas, popularly known as star schemas. It provides multi-dimensional views on the historical stored data. This multi-dimensional model is alternatively known as a cube. Each logical cube illustrates a real word business entity comprising of different measures, dimensions, dimension attributes, levels, and hierarchies. Depending on the analytics requirement, OLAP operations are performed on this cube to examine the different measures and other meaningful dimensions and attributes. Figure 3.4 demonstrates the different objects relationship in a multi-dimensional model.

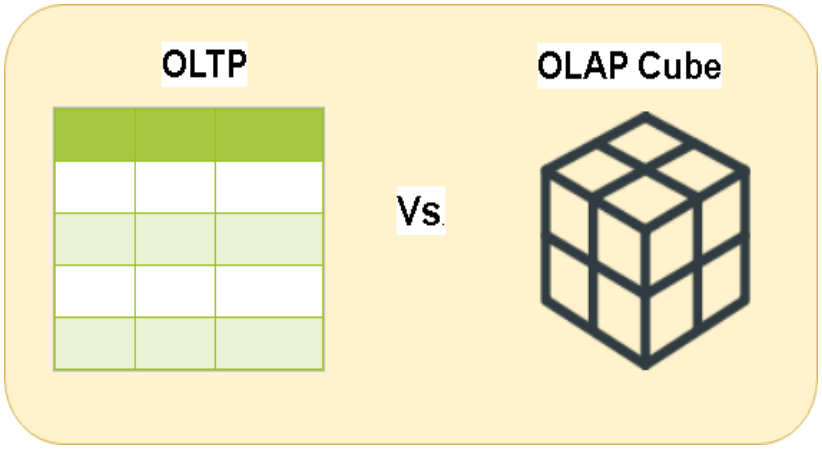

Figure 3.3: OLAP vs. OLTP

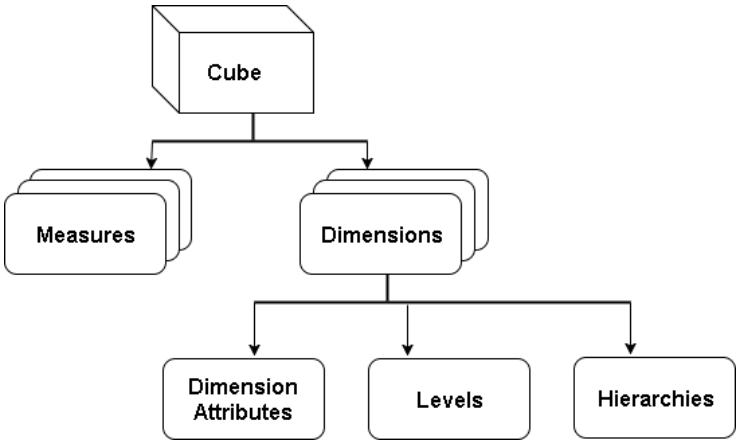

Figure 3.4: Logical dimensional model; relationship among objects

3.6.1. Dimensions in a logical cube. Dimensions correspond to the unique values that are used to categorize the data [31]. These form the edges of a cube. Thus, each numeric value of a Measure is quantified with a dimension or attribute corresponding to a measure value. Since a cube is modelled in 3D, each cube has 3 dimensions across X axis, Y axis and Z axis. Dimensions of the cube correspond to different data attributes and parameters such as time, location, and appliances. These dimensions can be further categorized into hierarchies, levels, and dimension attributes. Hierarchies and Levels are a way to organize data into different levels of aggregation [31]. The analysts can drill down to lower levels or roll up to higher levels to identify consumption trends in smaller and larger sector of population respectively. On the other hand, dimension attributes are utilized to provide additional information about each dimension.

3.6.2. Measures in a logical cube. Measures correspond to different values that populate all cells in a dimensional model. These measures in a cube are organized based on different dimensions.

3.6.3. Relational implementation of a cube: star schema. A Star Schema is a method of organizing Cube data in the form of one Facts Table, several dimension tables, and materialized views. A dimension table comprises of different dimension attributes, levels and hierarchies. A Facts table and materialized views comprise of measures. Each Dimension table has a primary key that associates it to the Facts table. The Materialized Views are instantiated for displaying a combination of levels. Figure 3.5 illustrates a generalized Star Schema implementation:

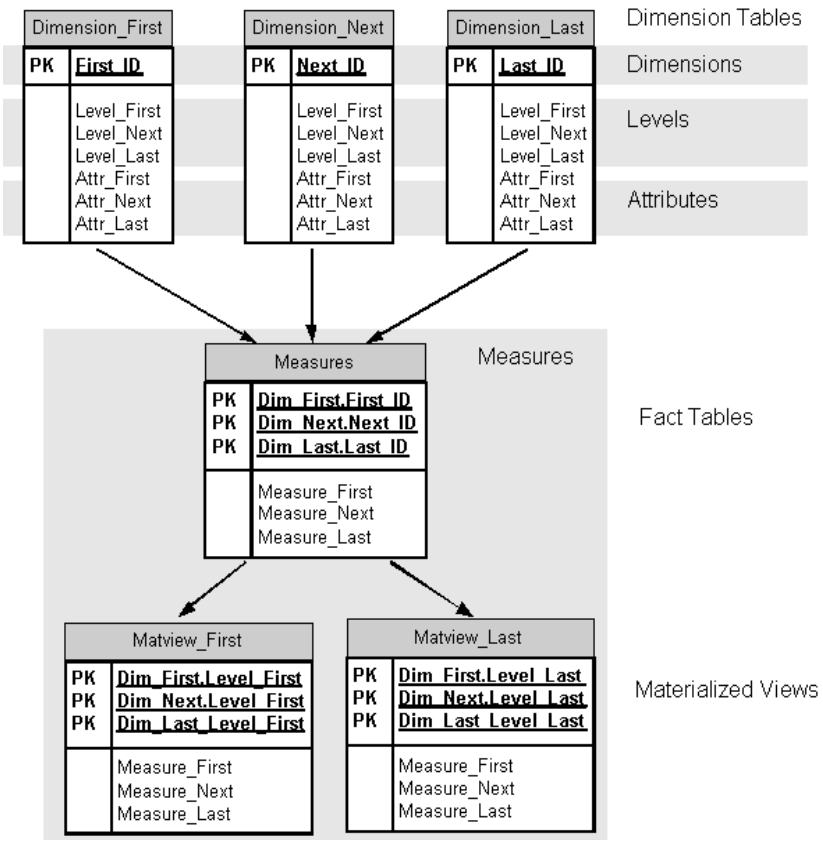

Figure 3.5: A standard star schema implementation [31]

In view of the business processes and stakeholders described in earlier section, the conceptual dimensional model of a smart meter data is as follows:

- Hierarchy in data: The smart meter data includes each house device consumption data on an everyday basis. This device-level data can be aggregated up to community-level, state level, and national level. . Thus, the user consumption can be divided into four user granularities; home owner, community owner, state owner, and country owner. Each level is relatively independent of each other, data however, the underlying user data forms the very foundation for the upper layer data.
- Time granularity in data: A home smart meter data represents the device consumption details for each day in a year. This time dimension can have hierarchy with relative time periods such as levels of week, month, and year.
- Dimensions design: Dividing the complete data structure into dimensions provides structured information for analysis as well as reporting. Each

Dimension table will further comprise of different attributes related to that specific dimension. This is demonstrated with the help of a schema that provides a logical description of the entire data structure. Dimension corresponds to the attribute of the facts or the factual information from the data. In our design, there three primary dimensions; Location corresponding to House IDs , Time corresponding to Days attribute and Appliances dimension referring to types of devices in a house.

- Metric formulation: Metric is the fact that forms the basis of the data. From our design, the device consumption in kW (Cellar, Furnace, Fridge, Heater, Washing Machine, Lights, Outlets, Cooking Range, and Dishwasher) is the metric value.
- Model formulation: A star schema is used to design the OLAP data model. This star schema is a relational database model that can be stored in any conventional data mart. The dimensions of this star schema are stored in dimension tables and the metrics are stored in fact table. Figure 3.7 represents the OLAP star schema model for the smart meter data. This model plays an important in data warehousing techniques as it helps understanding the physical layout of the underlying data sources or tables and how they are related to each other. In this Star Schema, each cube dimension is represented with one dimension table consisting of several attributes related to the corresponding dimension and a private key. Moreover, there is only one Facts table comprising of several dimension keys linking to every dimension table and measures that correspond to attribute values such as power consumption in this case.

A star schema for smart meter data shown in Figure 3.6. The constructed cube in the Figure 3.7 represents how the smart meters data from several houses is modelled in the form of cube. OLAP operations will be executed on top of this cube to render consumption query results to different levels of stakeholders. The cube is represented with three dimensions, namely, Location House IDs, Time (Days), and Appliances (Type).

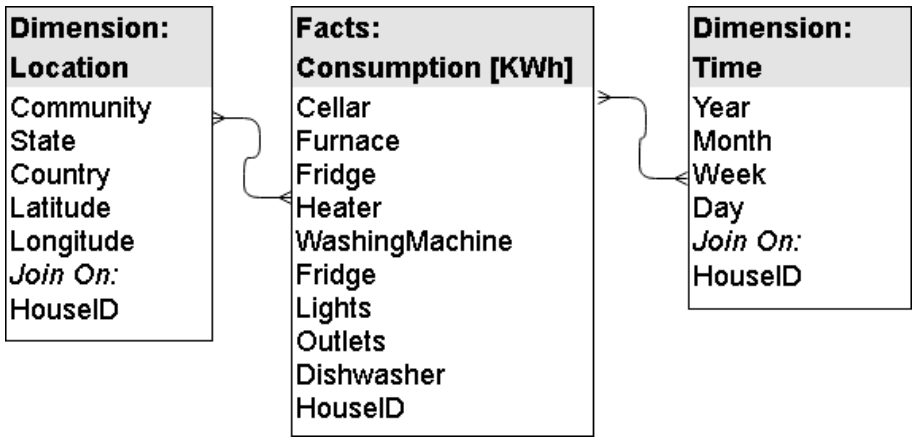

Figure 3.6: A star schema for smart meter data

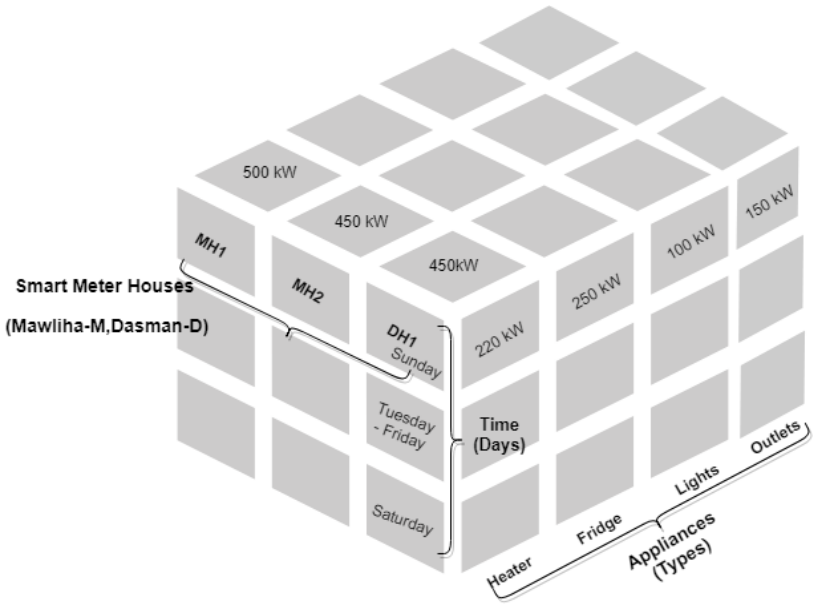

Figure 3.7: OLAP Cube for smart meter dataset

Each dimension is further categorized into dimension attributes such as MH1, MH2, and DH1 for the dimension Location house IDs, dimension Time has Days, and the dimension appliance represents four types pf appliances, namely Heater, Fridge, Outlets, and Lights. The cells of the cube are populated with the power consumption values with respect to each dimension. There are five different operations that can be performed on this OLAP Cube:

- Roll Up: Roll Up operation performs aggregation on a data cube by climbing up the hierarchy for a dimension. Figure 3.8 shows the roll up operation on the dimension of Location House IDs to represent the rolled up consumption data from individual houses in a community to total power consumption all houses in each community.

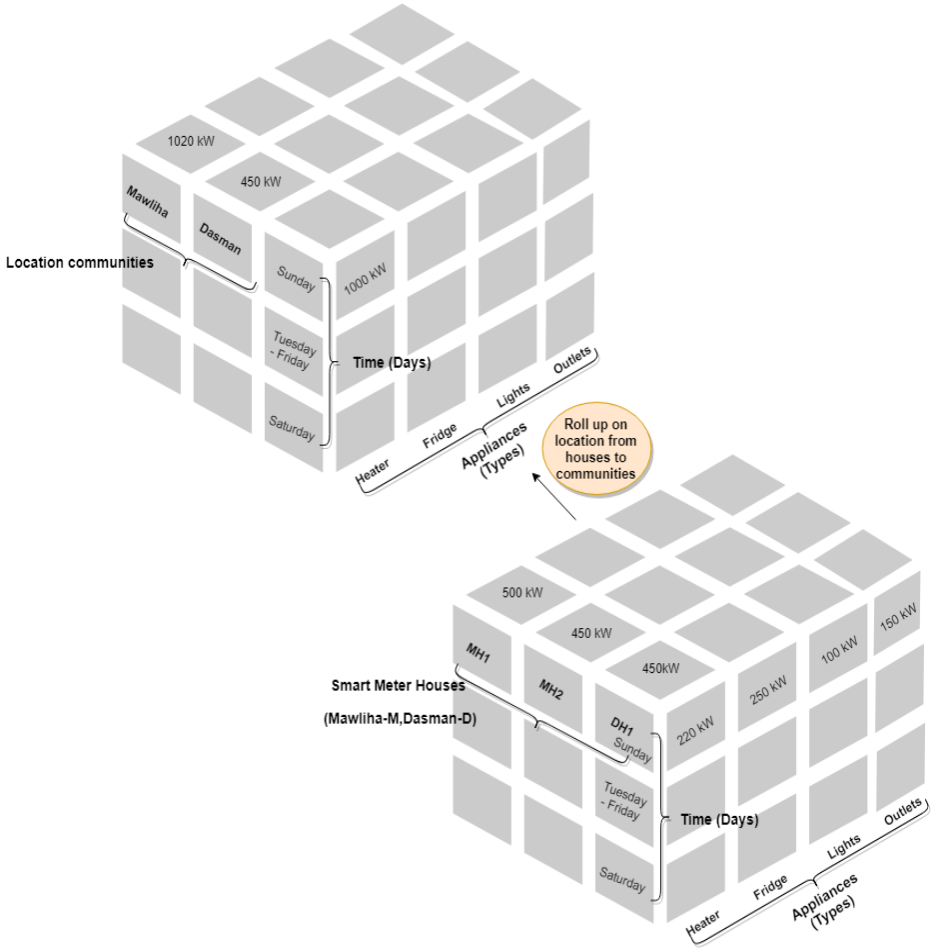

Figure 3.8: Roll Up on an OLAP cube on location dimension to aggregate total houses' consumption in a community

Similarly, Figure 3.9 illustrates a roll up operation on dimension time to aggregate the day's consumption of a house into weekly smart home consumption. Figure 3.10 represents a roll up operation from weekly smart home consumption to monthly consumption. Figure 3.11 represents a three level hierarchical roll up operation on dimension location from house consumption to community consumption to state consumption. Each higher

level stakeholder consumption is the aggregate power consumption of the lower granularity attributes.

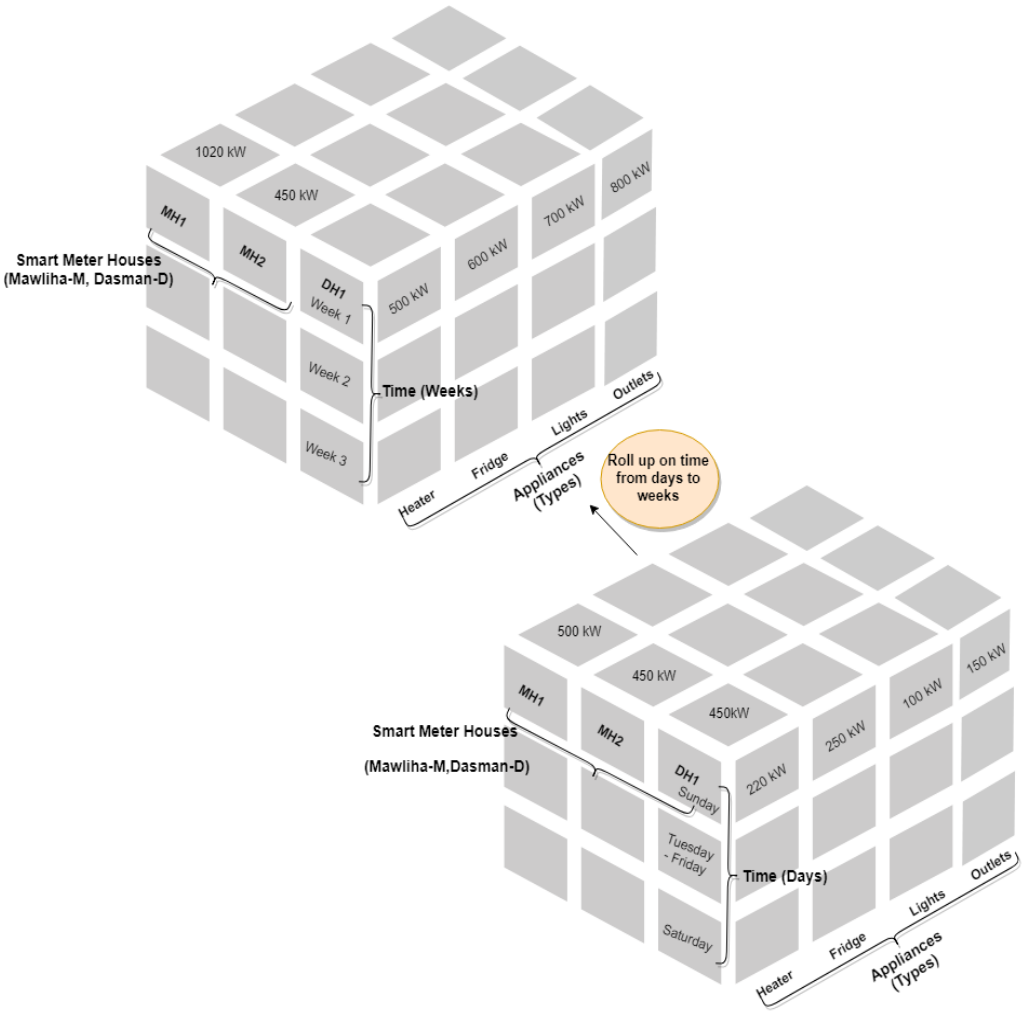

Figure 3.9: Roll Up on an OLAP cube on time dimension to aggregate house MH1 daily consumption to weekly consumption

- Drill Down: Drill down is the opposite of Roll Up operation. It either introduces new dimension in the model or steps down the concept hierarchy for a dimension. Figure 3.12 demonstrates the drill down operation on the Time dimension from monthly power consumption to weekly power consumption.

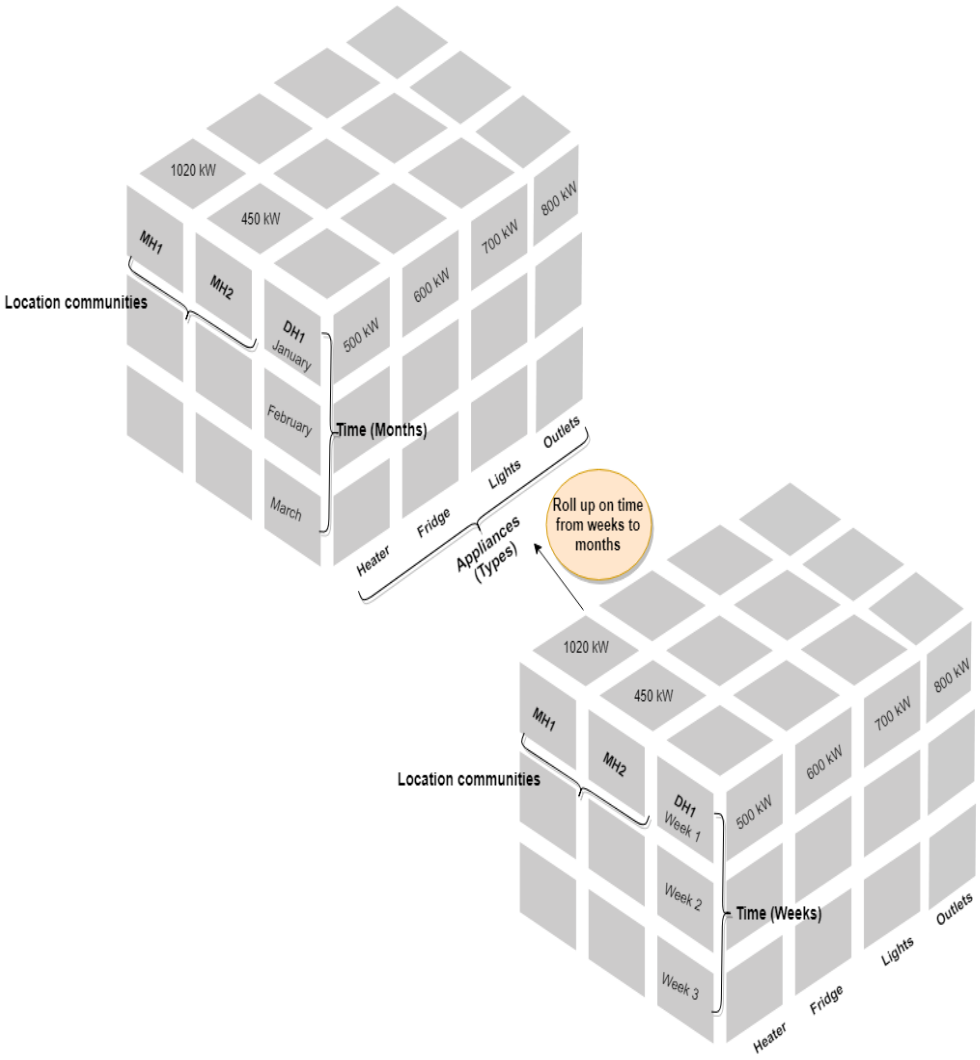

Figure 3.10: Roll Up on an OLAP cube on time dimension to aggregate house MH1 weekly consumption to monthly consumption

- Slice: Slicing operation provides a new sub cube from one specific dimension of a given cube. Figure 3.13 illustrates the resulting cube after performing slice operation on dimension location for House ID, DH1.
- Dice: Dicing operations provide a new sub cube from two or more dimensions of given cube. Figure 3.14 illustrates the resulting cube after performing dice operation on all three dimensions of Location, Time, and Appliances. Dice operations is performed for location House ID=MH1, Time=Sunday, and Appliances=Fridge.

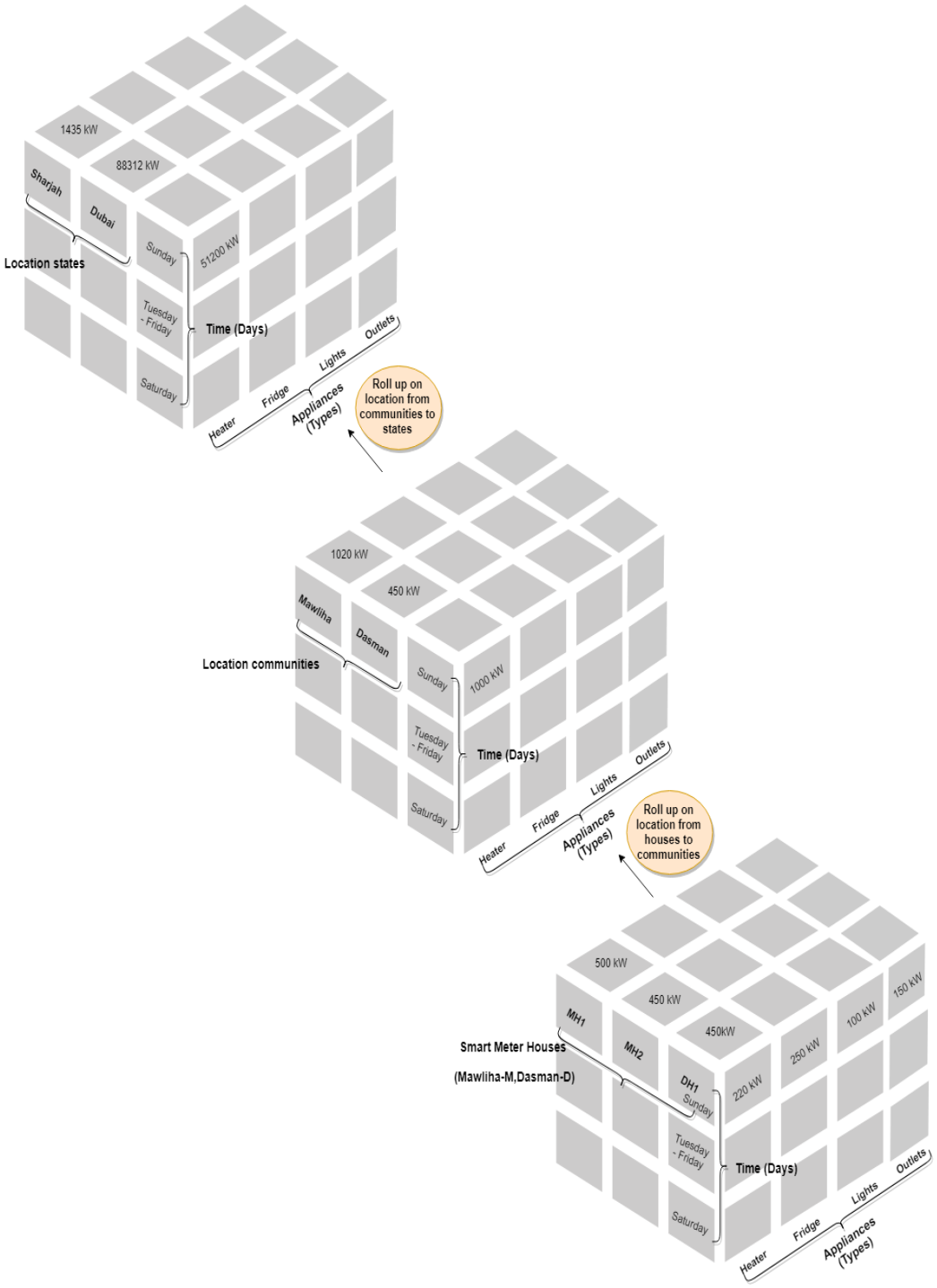

Figure 3.11: Roll Up on an OLAP cube on location dimension to aggregate house consumption to community consumption to state consumption

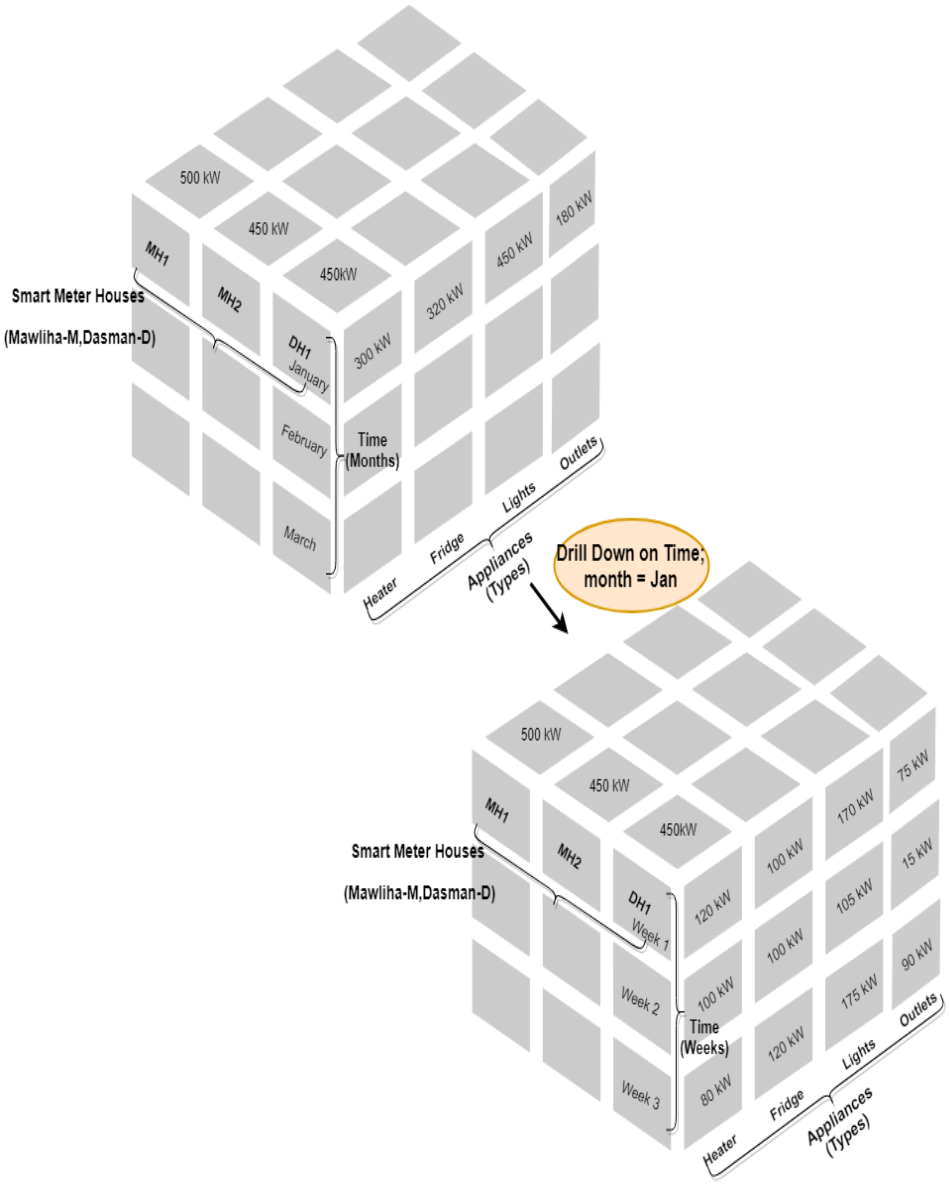

Figure 3.12: Drill Down on an OLAP cube

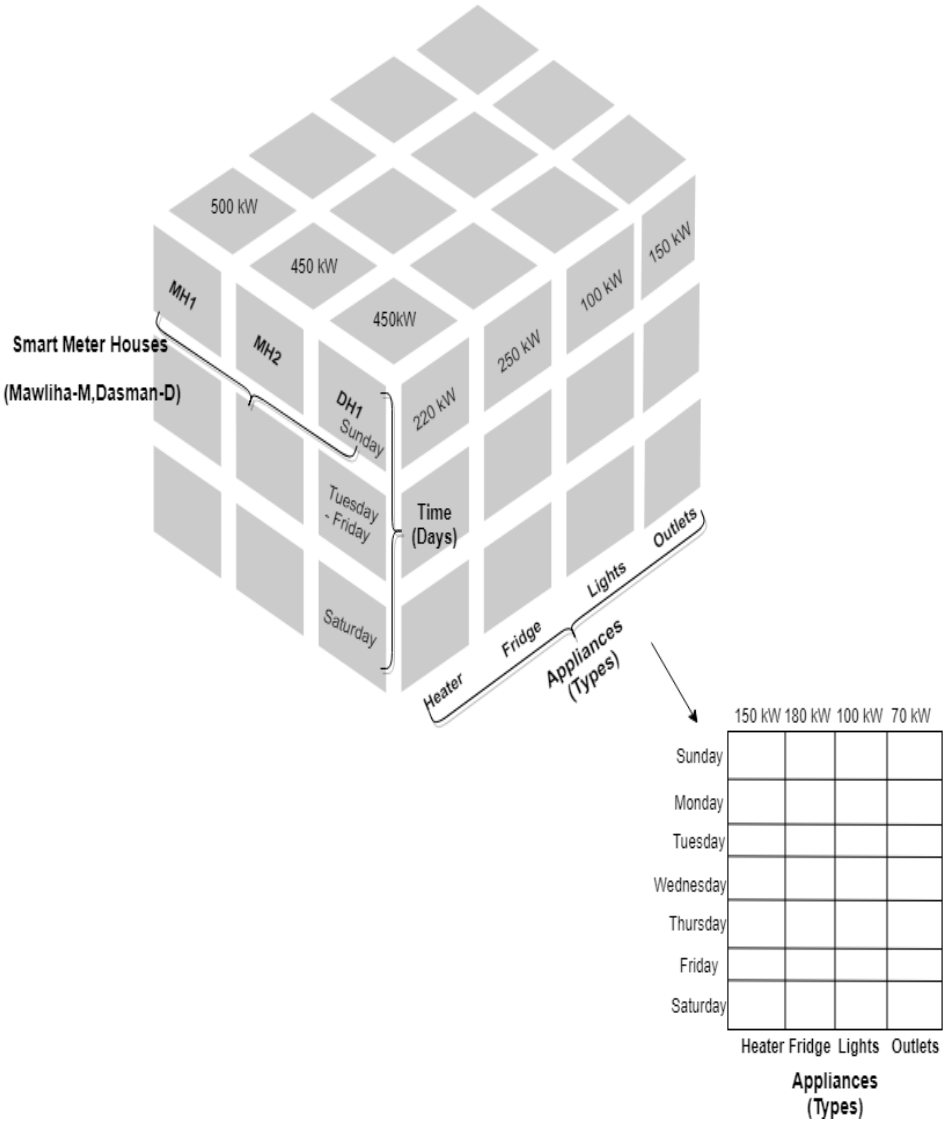

Figure 3.13: Slicing on an OLAP cube

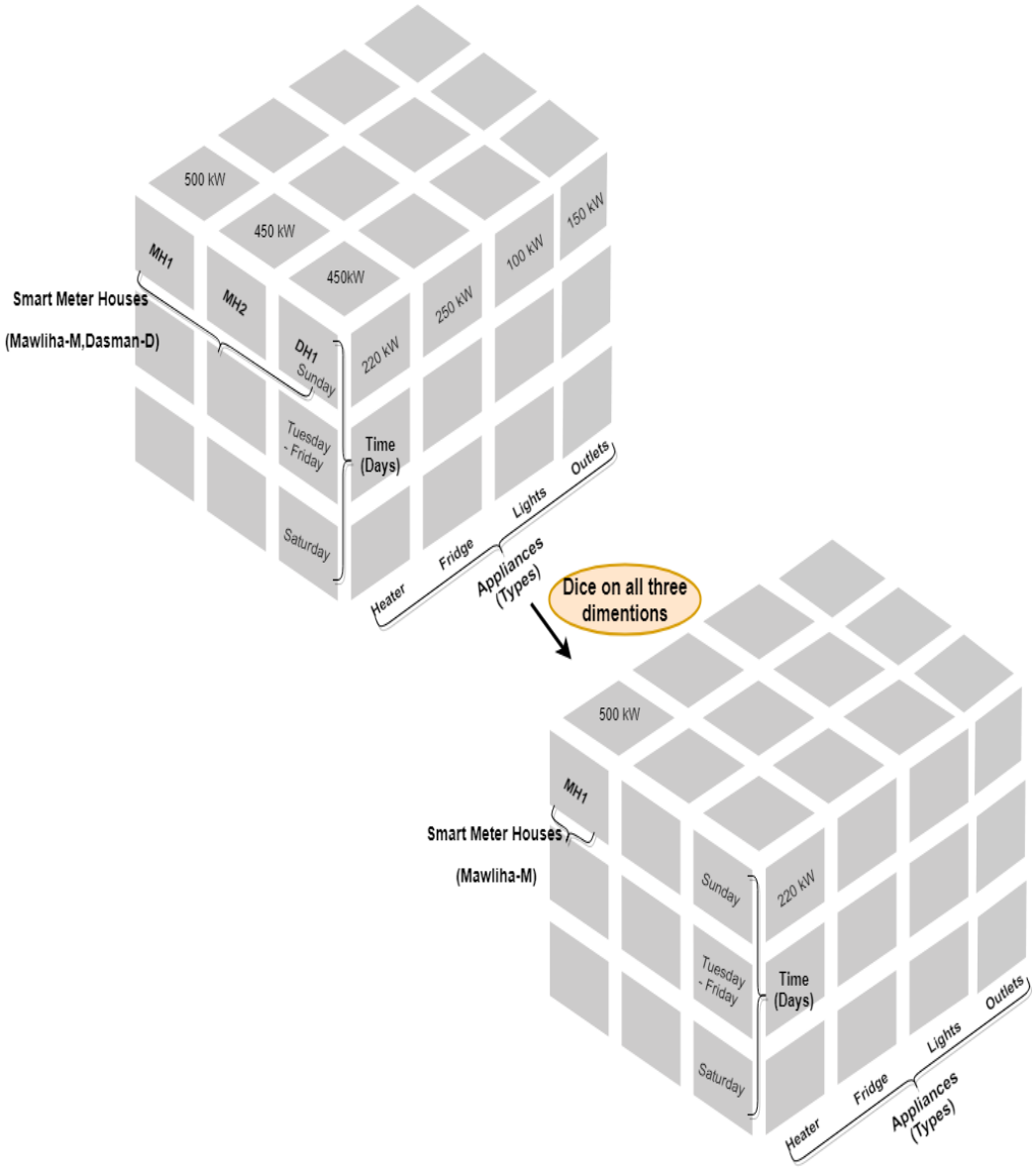

Figure 3.14: Dicing on an OLAP cube

3.7. Hadoop Distribution Selection

Apache Hadoop is an open source platform that allows distributed processing of large volume of datasets across multiple nodes in a cluster/multi cluster environment. The three modes of installation are [32]:

- Standalone Mode: Single node cluster.
- Pseudo Distributed Mode: Simulated multi node environment on a single node machine.
- Distributed Mode: Multi node cluster.

Depending on the research requirement, we chose to setup distributed mode for installing Hadoop in a multi node cluster. A Hadoop cluster comprising of one master and three slave nodes is created using the open source Hadoop distribution, Cloudera Distribution Hadoop (CDH) 5.2. CDH is a complete tested 100% open source Hadoop solution that offers different batch processing, interactive SQL querying, and continuous availability.

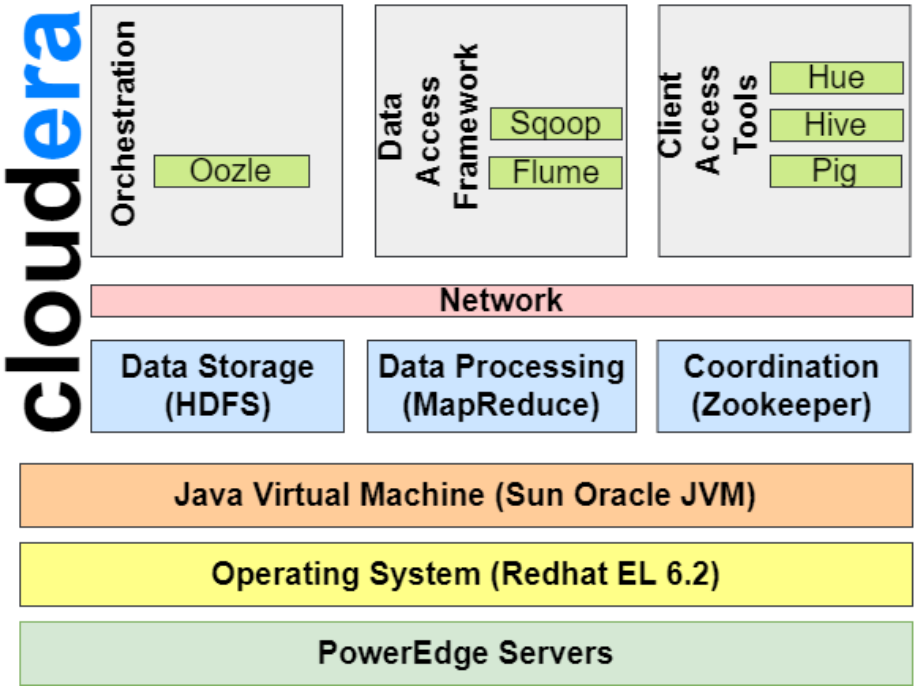

Figure 3.15: Cloudera taxonomy [32]

As shown in Figure 3.15, the Java Virtual machine (JVM) makes up the foundation on which Cloudera stack runs. The dark blue components depict the two core Hadoop frameworks:

- Data Storage: It is a distributed, scalable Hadoop file system storage that is used to store data on the cluster nodes.
- Map Reduce processing framework: It is a parallel computing framework introduced by Google.

The next layer is the Hadoop network layer as Hadoop makes extensive utilization of network based services in sending and fetching data across cluster nodes such DNS lookup for name resolution of data nodes, interference of Hadoop traffic from other network applications, etc.

One of the main aspects of an Apache Hadoop [33] ecosystem is its capability of distributed parallel processing. To achieve this, Hadoop consists of a fault tolerant storage system called the Hadoop Distributed File System, also known as HDFS [34]. HDFS is designed to store big data sets across low cost machines in a distributed fashion. It splits the data files into multiple "blocks" and stores them redundantly across a relatively cheap hardware. Additionally, HDFS can scale up incrementally with increasing data and survive the failure of the storage infrastructure without any data loss. In order to empower distributed processing, Hadoop creates multi-node clusters of machines and coordinates work among them. Each cluster comprises of its respective high quality name node and multiple data nodes. The concept of creating multi-node clusters in HDFS is because it makes Hadoop fault-tolerant. This is due to the fact that in the case of a data node failure within a cluster, Hadoop continuous to work by transferring the same load to other data nodes in the same cluster.

Following points will highlight the type of nodes and architecture of a Hadoop Cluster [35]:

3.7.1. Name node. Hadoop has one centerpiece machine called a Name node server, which stores and manages the metadata for HDFS. It is the master node in Hadoop distributed framework. Name Nodes are configured with large RAM as it is the directory for all the files and blocks stored in the file system across several data nodes. These Name Nodes run a Job Tracker process that assigns map reduce tasks among data nodes.

3.7.2. Data nodes. The slave nodes in a Hadoop cluster are called Data nodes.

These Data Nodes store the datasets and perform the read/write operations from the clients. These data nodes can be controlled and accessed by the master/slave node. It

also performs the replication of the datasets on the instructions from Name node. Data nodes execute a Task tracker process that performs and computes the map or reduce jobs as allotted by the Job tracker process of the Name node [35].

3.7.3. Master slave architecture. Hadoop cluster is designed as a master-slave architecture as shown in Figure 3.16. The master node comprises of the Job tracker process while the slave nodes run the task tracker process. The Job tracker on the master node is responsible for managing cluster resources, scheduling, and monitoring progress and fault-tolerance mechanisms [36]. The Job tracker receives the jobs submitted by the users and appoints it to the Task trackers. The Task tracker process on each slave node initiates parallel tasks and registers the task status to the Job tracker constantly [37]. To execute either map or reduce jobs, the slave nodes are statically divided into different computational tasks depending on the respective machine's RAM and memory usage capacity. Additionally, each node can be segregated into computational (Map Reduce) layer and storage (HDFS) layer that helps in scaling out the memory of the corresponding node. In short, following are a few demons that run on multi node standardized Hadoop cluster:

One Master node executes the following modules:

- Job tracker (MapReduce layer)
- Task tracker (MapReduce layer)
- Name node (HDFS layer)

Several slave nodes consist of the following modules:

- Task tracker (MapReduce layer)
- Data node (HDFS layer)
- MapReduce layer contains the job tracker and task tracker processes
- HDFS layer contains the name node and data nodes

Figure 3.16 demonstrates a master-slave architecture paradigm in a distributed Hadoop cluster and its respective daemon processes.

3.8. Distributed Processing Paradigms on Hadoop

With the constant increase in volume of energy big data, it is challenging to forecast the size of this data and the computing power required to store and process such huge data quantity.

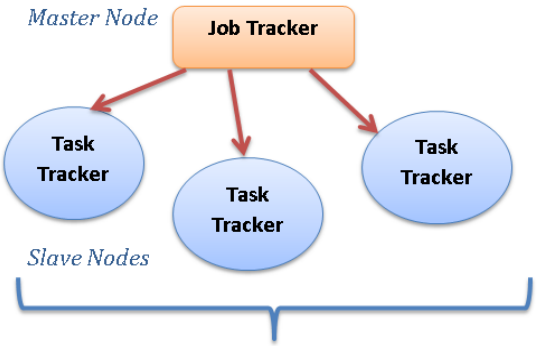

Figure 3.16: Master slave architecture in a hadoop cluster

In a non-distributed platform, when the computing power increases scaling up is one of the option where a single computing system with additional number of cores, hard disks and large memory is used in parallel. However, such an architecture is a lot more expensive with a single point of failure and large dependency. Alternatively, a more cost effective option with fault tolerance capability is to scale out where the computing tasks are divided amongst less powerful machines with moderate computing power and resources running in parallel. Map Reduce is one parallel batch processing framework that provides programmers with an abstraction from low level hardware complexities, thus enabling a reliable and fault tolerant scheme for big data applications. The concept of Map Reduce programming model was introduced by Google to enable web indexing. Apache Hadoop is one open source software with different components steered on Map Reduce.

3.8.1. Parallel processing paradigm on hadoop using map reduce. The computation of map reduce is such that it takes an input of key/value pairs and outputs a set of key/values pairs. The mechanism involves a number of stages. An example of Map Reduce model is demonstrated in Figure 3.17 below to generate power consumption report of two communities, namely, Maliha and Dasman on a quarterly basis for a State Energy Utility Provider. The Figure 3.17 illustrates how power consumption from each device of every house in the community is stored on HDFS Data Nodes followed by splitting and extraction of total power consumption data within each community using the Map Reduce algorithm.

When a MapReduce is run by Hadoop, the job is sent to a master node, the Job tracker which has multiple slaves, or Task trackers that report to it, and ask for new work when they are idle. Job tracker split the map and reduce tasks among the Task

trackers, so all of them work in parallel. The Job tracker keeps a track of which Task trackers fail.

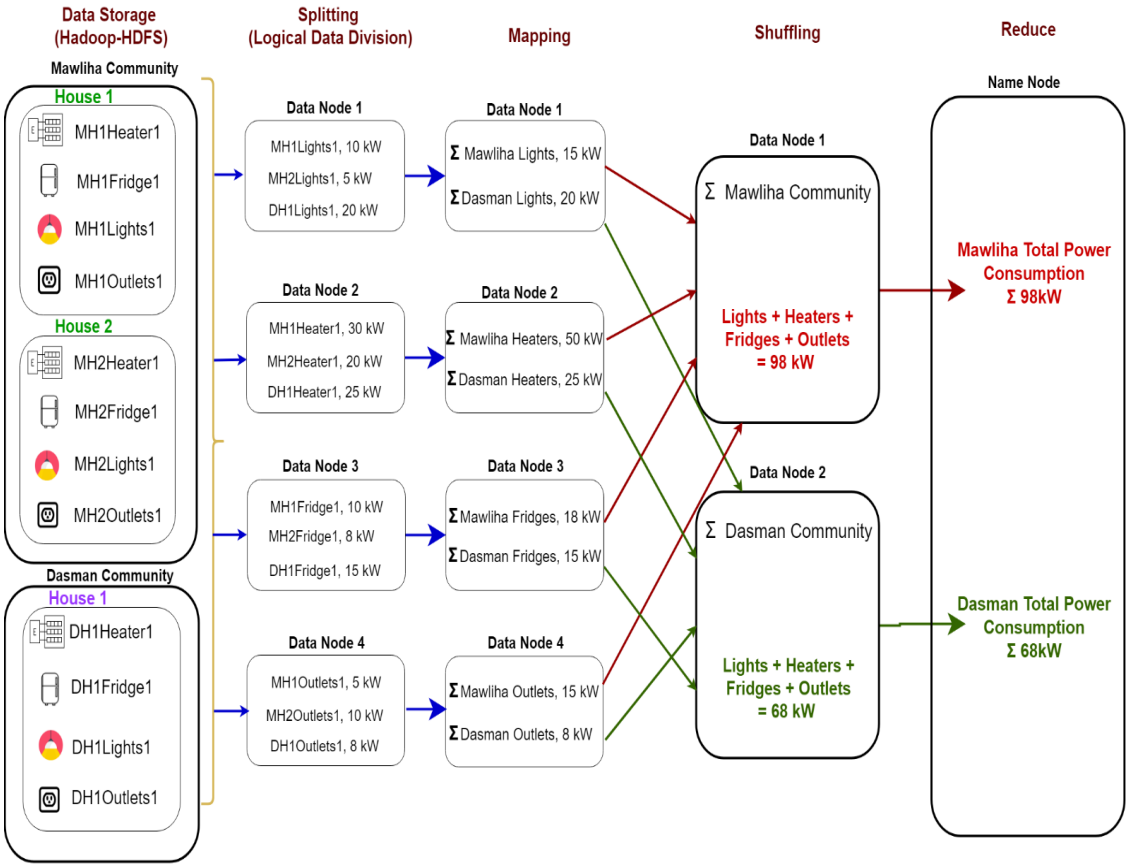

Figure 3.17: Map Reduce processing model on hadoop

Figure 3.18 shows the Hadoop cluster architecture and its work can be explained in the following steps [37]:

1. The Map-Reduce library first split the data into small pieces from 16 MB to 64 MB , then it makes copies on the cluster of the machine, and one special copy to the master, and others to the workers. There are M map tasks and R reduces tasks; the masters chooses the idle workers and give tasks to them.
2. A worker with a map task reads the content of the input split and parses key/value pairs out of the input data and passes it to the map function. The

intermediate key/value pairs produced by the map are buffered in the memory.

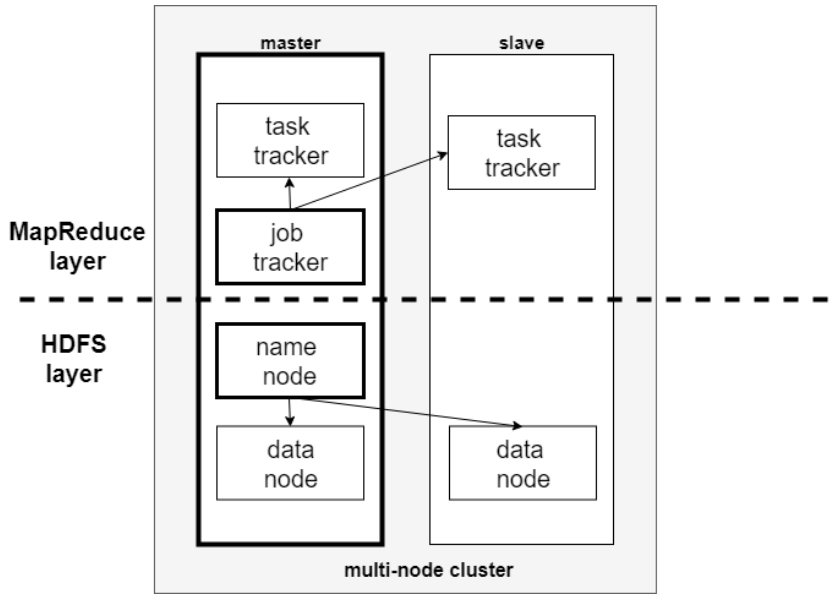

Figure 3.18: Hadoop cluster architecture [37]

Those buffered pairs are written to the disk periodically, partitioned into R regions by the partitioning function. The location of these pairs on the disk are passed to the master, who is responsible for passing these pairs to the reduce workers.

3. When a reduce worker is modified by the master about these locations.it uses remote procedures to read the buffered data from the local disk. Then it sorts it .Once it finishes, the reduce worker iterate over the sorted intermediate data, and for each unique key it passes the key and the corresponding set of data to the user reduce function, the output of reduce function is append to the output file.
4. When all map and reduce tasks done, the master wakes up the user program.

3.8.2. SQL engines on top of Hadoop facilitating map reduce. Map reduce jobs on Hadoop are coded by programmers in different languages such as Scala, Java, etc. depending on the data analysis requirement. However, such coding is a time consuming task as the map reduce performance completely depends on the code and

requires a lot of manual effort in practice. It is therefore more practical to implement high level abstraction queries that autonomously develop and optimize the map reduce jobs such as SQL. With the help of SQL queries, dynamically the map reduce jobs are generated and the data analysis of HDFS data can be directly accomplished using SQL. The most commonly used SQL engines in a Hadoop framework are SparkSQL and Hive. Both platforms share a common architecture similar to client/server paradigm [38].

3.8.2.1. ***Apache hive*.** Hive is one of the open source Hadoop data warehousing applications that supports querying languages such Hive-SQL or H-QL for data analysis in a Hadoop ecosystem. It provides a structure to the unstructured data stored in HDFS for querying data. The H-QL language directly translates the queries into map reduce jobs for execution in a Hadoop cluster. The stored HDFS data is organized into Hive tables, thus, providing a low level structure to the unstructured data in HDFS. The metadata of the Hive tables such as schema of the data is stored as Hive metastore in the master node of the cluster.

- Hive Architecture: Hive architecture comprises of command line interface (CLI), a JDBC/ODBC middleware, and a Graphic User Interface (GUI) as the application layer on top of Hadoop for executing Hive-SQL queries [38]. The middleware comprises of the Hive Thrift server that allows clients outside the Hadoop framework to send SQL requests to hive data tables. As shown in Figure 3.19, the driver between the thrift server and the HDFS store is the database layer. Once the HQL queries are posted via the CLI or from the thrift server, the driver makes a call to the compiler that translates the queries into map reduce jobs. The sequence of query execution in Hive is as follows: Hive-SQL -> Parse Tree to validate SQL statement-> Query representation using a logical plan comprising of a tree of operators -> rule based optimizer for optimizing the logical plans for determining how to execute a query -> Map Reduce tasks. The Hive metadata is stored in Hive metastore that is usually generated during the time of Hive table creation and it is accessed when the records are being read from or written to the Hadoop cluster. This Hive metastore is stored on to the client's operating system or the master node of the Hadoop cluster [38].

- Data model in Hive: The data files stored in HDFS are abstracted as Hive tables when stored in HDFS. A table schema is applied to the structured data files stored in HDFS by generating a hive table inside the /hive/warehouse directory on HDFS by default [38].

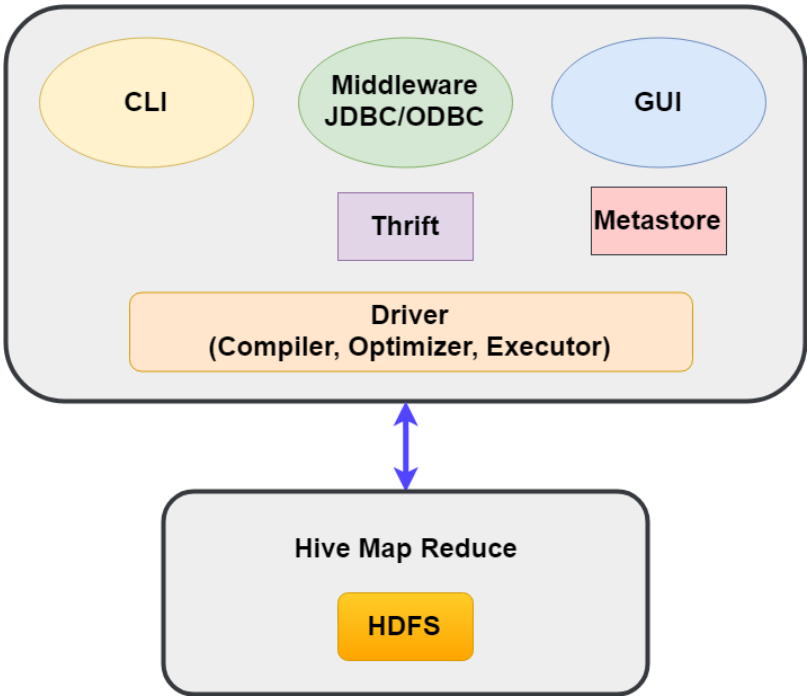

Figure 3.19: Hive architecture [38]

When a hive table is created, it's dynamically generates a metadata of the table that consists of the all the information regarding data tables, databases, table columns, column types, etc. When hive queries are executed, the metadata helps in mapping the hive table to the actual HDFS data and then process it using the map reduce job.

- Hive Query Language: Hive-QL is the SQL oriented Hive query language for Big data designed to query data from the hive tables in a Hadoop ecosystem. Once the hive queries are submitted through CLI, the compiler translates them into corresponding map reduce tasks for processing in the map reduce engine on top of Hadoop. Since map reduce paradigm is very low, most developers and analysts use the map reduce abstraction with Hive-SQL queries. Hive-QL differs from the SQL of relational database management system in the way that it supports the feature of Serialization and deserialization using the SerDe

library. SerDe [39] is a library built-in to Hadoop API that allows Hive to read in data from a table, and write it back out to HDFS in any custom format. It is designed to read unstructured data from any custom file format where the data is separated by a delimiter character

Hive encapsulates the following steps during the course a HiveQL statement [40]:

- Step 1: After submitting the query on a CLI or UI, the UI calls the execute interface to the Hive driver.
- Step 2: The driver creates a session handle for handling the query and sends the query to the compiler in order to produce an execution plan for the corresponding map reduce tasks.
- Step 3: The compiler creates an acyclic graph of map reduce tasks and validates the query semantics using the metadata from metastore.
- Step 4: The plan generated by the compiler is a DAG (Directed Acyclic Graph) that comprises of different stages with each stage being a map or a reduce job or an operation on HDFS.
- Step 5: The execution engine submits these stages to their task specific components. In each task for the mapper/reduce, the map reduce engine is reads in the rows from the Hive table stored in HDFS. Once the output is produced, these are written out into a temporary file onto the HDFS.

3.8.2.2. *Apache spark*. Apache Spark is currently one of the most popular big data processing engines on top of Hadoop. It is much faster than the conventional batch processing and map reduce programming model in a Hadoop ecosystem. One of the major reasons for it to be faster than map reduce is the in-memory computational feature in Spark. With the help of in-memory processing, the intermediate results in Spark are stored in memory and this allows reuse of the data available in memory thus reducing the disk I/O. This significantly leverages the execution time of queries in Spark and particularly after executing queries for the first iteration [38]. Spark provides an easy interface for programing Scala language and can be integrated with Java, R, and even python language. The Scala code written on a spark API corresponds to one fifth of the code written in map reduce. In contrast to Hive, Spark can fetch and process data not only from HDFS but also from the client's local file system, or cloud based storage services [38] The Spark stack as shown in Figure 3.20

comprises of several components such as Spark Core that provides an API's for Java, R and python programming languages. Other components developed on top of spar core are; Spark SQL, Spark Streaming and MLLib. Spark SQL is integrated to utilize data analysis and querying from the structured data. The Spark streaming is responsible for processing live stream of real time data and MLLib provides a machine learning paradigm on top of Hadoop framework. Spark performs unstructured computations using Resilient Distributed Datasets, RDD [38]. RDDs are stored and distributed across cluster nodes in memory as well as on the disk. The motivation to use RDD's is that they can be persisted in memory or on the disk. Caching the RDDs in memory allows high-speed processing. In addition to unstructured processing, Spark also supports structured relational functionality, which utilizes an extensible catalyst optimizer for faster operations than RDDs. Spark SQL is one such Spark module that provides faster structured computation and allows the users to integrate SQL queries into the Spark programs [38]. Spark SQL catalyst optimizer is cost-based query optimization and code generation that allows for the query computation to be agile enabling fault-tolerance. Spark SQL provides data processing on top of Hadoop utilizing structured data processing functionalities. Additionally, its syntax is similar to SQL-like statements. The Spark SQL queries can be executed through a Spark SQL module on command line interface similar to hive's H-SQL. Spark SQL allows the users to import data from the relational hive data tables, run SQL queries, and write the results on to the Hive tables.

- Spark Architecture: Spark's architecture as shown in Figure 3.21 is dominated by master-slave pattern in a Hadoop cluster. The master of the cluster contains the cluster manager, popularly known as the Yarn scheduler.

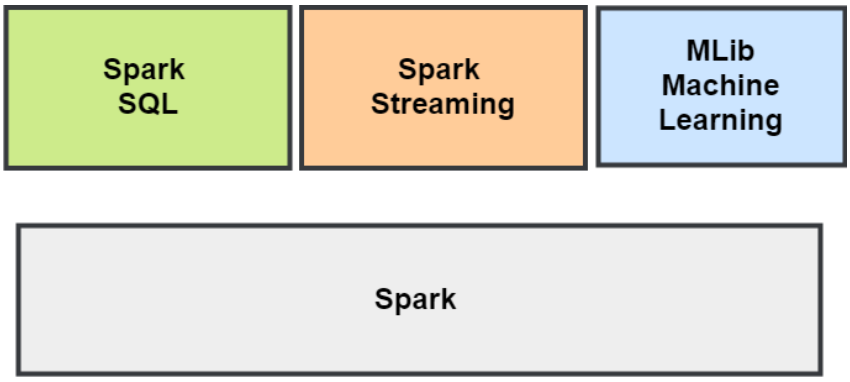

Figure 3.20: Spark Components [41]

- For each spark query submitted, the Yarn allocates resources by initiating executor (container) process on each worker node. The executors on each slave node can be seen as a background process that will be processing and storing the data on each slave node of the cluster. The executors have their own thread pools wherein each thread executes multiple tasks in parallel on its respective node [38].

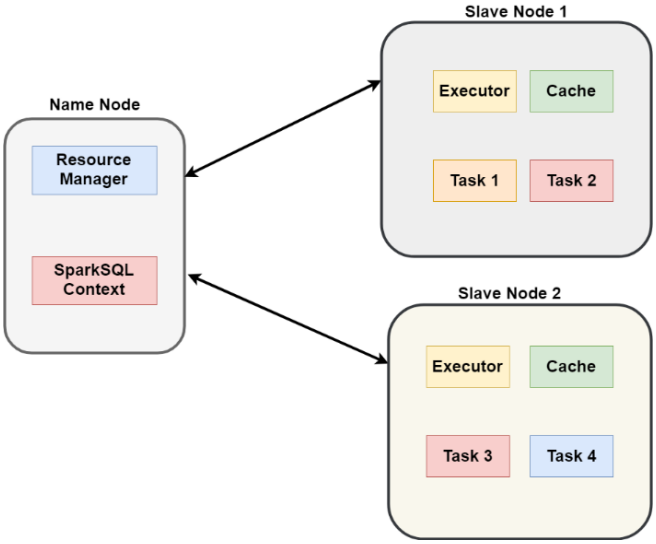

Figure 3.21: Spark client-server architecture [41]

Moreover, the spark's in-memory computing feature is implemented for executors wherein each executor has a read-only cache in memory. This cache is shared by all executor tasks generated by the same spark query or application. The in-memory processing in Spark improves the execution time of Spark queries in contrast to Hive map reduce processing that entails splitting the data onto the disk with heavy disk I/O cost. In-memory processing is highly beneficial when it comes to iterative processing as it allows data to be re-fetched from the cache rather than from the disk after the first iteration. In order to write a spark application program, the user has to write a driver program that implements the execution flow across different tasks of the worker nodes in parallel. The driver program of the spark SQL application is called the Spark SQL Context. The SparkSQL context encapsulates all the structured and relational processing features in Spark in the form SQL like statements. The general context to SparkSQL is as follows:

```
var sqlContext = new org.apache.spark.sql.sql.SQLContext( )

var query = sqlContext.sql(" SELECT ");
```

- Execution workflow in SparkSQL: Spark provides a second-generation map reduce model on top of Hadoop. It applies the idea of map reduce programming model by segregating the submitted queries into map-reduce tasks. Spark SQL context module is used to execute Spark queries on top of Hadoop cluster. Using the Spark SQL API, Hive-SQL queries are written and executed that computes results utilizing the same spark execution engine. Two kinds of operations are performed on SparkSQL queries; transformations and actions. Transformations refer to the mapper function of Spark engine that perform data splitting and mapping the data according to <key, value> pairs. The actions correspond to the reduce function of Spark engine that execute reducer tasks on the output of the transformations and send the result to the name node [41].

3.9. SparkSQL vs. Hive

Both Spark SQL and Hive are big data Hadoop based processing engines that support querying languages of SQL for data analysis. For Hive, the Hive SQL queries are directly translated into directed acyclic graphs (DAG) that comprises of map reduce tasks processes in parallel across the cluster nodes. Hive's execution is strictly bound to HDFS. Hive provides parallel processing with disk caching. On the other hand, Spark SQL queries are translated into Directed Acyclic Graphs (DAG) that comprise of multiple transformations and actions besides just map and reduce functions for more optimized execution of Spark queries. It facilitates in-memory caching with several threads executing in parallel in memory [38]. Additionally, Spark paradigm is fault tolerant as it keeps a track of all the partitions created by mapper and reducer tasks in a lineage graph. This graph can be reutilized to regain any lost partitions. For e.g. if a data partition in-memory is lost due to node failure while data processing, Spark has the capability to regenerate the lost partition by applying the same mapper and reducer function to the corresponding data block in HDFS file [42]. This recovery saves the overhead time of rerunning the queries since the failed node contains multiple partitions which can recreated with in-parallel on cluster nodes. In terms of resource management, both SparkSQL and Hive depend on

the Hadoop ecosystem's resource manager Yarn for dynamic allocation of container. Containers are the JVM heap memory that are allocated to execute map and reduce tasks. Table 3.2 shows how SparkSQL and Hive are different from each other for different characteristics.

Table 3.2: Comparison between SparkSQL and Hive [38]

Characteristics	SparkSQL	Hive
On top of Hadoop HDFS	Yes	Yes
MapReduce processing	Yes	Yes
DAG generation	Yes	Yes
In-Memory	Yes	No
Fault Tolerant	Yes	No
Support for SQL	Yes	Yes

3.10. Formulation of Queries w.r.t. Stakeholders

To provide a holistic view of energy consumption from home owner to country utility providers, a total of eighteen queries are written in SQL language. These big data SQL queries will be spun off on the two SQL engines of the Hadoop ecosystem; SparkSQL and Hive. The query description for each stakeholder is are tabulated in Table 3.3. Further breakdown of queries in SQL language is shown in Table 3.4.

Table 3.3: High-level queries per stakeholder

No	Stakeholders	Title	Description
1.1.	Home Owner	1.1.1. Load Profiling (Aggregation Functions)	Load Profiling can be performed by home owners to execute queries with aggregation functions such as SUM to obtain SUM consumption data for all home appliances. This data can be reported with respect to different granularities of time; daily, weekly, monthly, and annually. The weekly and monthly data can be collected and reported rigorously from selected start and end dates.
1.1.	Home Owner	1.1.2. Compare my consumption with neighborhood	A customer can compare SUM monthly/annual consumption of the house with the SUM consumption of his/her neighborhood (community). The customer can save energy and control devices based on this information.
2.1.	Community Level Utilities	2.1.1 Consumption Variability Analytics for Houses using Aggregation Functions	For a distributed smart grid, utilities must provision for the peak demand power consumption. Thus, it is critical for utilities to identify consumers that have large variation in their consumption and consequently offer them incentives that can help in smoothing out the demand. To cater this, the Utilities must be able to view SUM energy consumption distribution of customers in their respective community w.r.t different time granularities (weekly, monthly and annually) and geographic location dimension (latitude and longitude). The Utilities must be able to analyze SUM power usage for each community on a periodic basis to predict the demand in advance and balance the Demand Response chain. Aggregated power consumption using other aggregate functions such as sum, min or max can be used to view consumption details of houses within a community (not allowed to view within an individual's each house appliance consumption due to privacy reasons).
2.1.	Community Level Utilities	2.1.2. Load shapes on Weekdays/Weekends for each month	The Utilities must be able to view load shapes on weekdays/weekends for the required month of the year.
2.1	Community Level Utilities	2.1.3 Device-wise consumption in a community	The community utility provider should be able to view the total consumption of individual device from all houses within the vicinity of the respective community.
2.1	Community Level Utilities	2.1.4 Additional Statistical Information using Ad Hoc Queries	Ad Hoc queries are queries that are not predetermined and can be generated dynamically to obtain information when the need arises.
3.1	State level Utilities	3.1.1. Consumption Variability Analytics for Communities using Aggregation Functions.	The State Level Utilities should be entitled to view the SUM (SUM) power consumption variation for communities within each state on time granularity levels and geographic location dimensions/community name.
4.1	National level Utilities	4.1.1. Consumption Variability Analytics for Communities	The National Level Utilities should be entitled to view power consumption variation of all states (on a Histogram) with respect to time granularity levels (monthly, annually) and geographic location dimension /state name.

Table 3.4: Big Data SQL Queries

Query Sequence	Stakeholder	Query Breakdown	Hadoop SQL Queries
1	Home Owner	Total Consumption of each appliance of myhouseID=H1 every day in a week.	Select date format(DateTime,'EEEE') AS Days, round(sum(Furnace),2) AS Furnace, round(sum(Cellar),2) AS Cellar , round(sum(Fridge),2) AS Fridge , round(sum(Heater),2) AS Heater , round(sum(Lights),2) AS Lights, round(sum(Outlets),2) AS Outlets, round(sum(CookingRange),2) AS CookingRange, round(sum(WashingMachine),2) AS WashingMachine, round(sum(DishWasher),2) AS DishWasher from smarthomeconsumption WHERE HouseID='"H1"'AND weekofyear(DateTime)=3 group by date_format(DateTime,'EEEE');
2	Home Owner	Total Consumption of each appliance of myhouseID=H1, each week of a month= January	Select weekofyear(DateTime,'EEEE') AS weeks, round(sum(Furnace),5) AS Furnace, round(sum(Cellar),5) AS Cellar , round(sum(Fridge),5) AS Fridge ,round(sum(Heater),5) AS Heater , round(sum(Lights),5) AS Lights, round(sum(Outlets),5) AS Outlets, round(sum(CookingRange),5) AS CookingRange, round(sum(WashingMachine),5) AS WashingMachine,round(sum(DishWasher),5) AS DishWasher from smarthomeconsumption WHERE HouseID='"H1"'AND month(DateTime)=2 group by weekofyear(DateTime);
3	Home Owner	Total consumption of each appliance of myhouseID=H1, monthly for the given year.	Select month(DateTime) AS monthly, sum(Furnace) AS Furnace, sum(Cellar) AS Cellar ,sum(Fridge) AS Fridge ,sum(Heater)AS Heater , sum(Lights) AS Lights, sum(Outlets) AS Outlets, sum(CookingRange) AS CookingRange, sum(WashingMachine) AS WashingMachine,sum(DishWasher) AS DishWasher from smarthomeconsumption WHERE HouseID='"H1"' group by month(DateTime);
4	Home Owner	Total power consumption of each appliance of myhouseID=H1, annually.	Select houseid AS myhouseid, round(sum(Furnace),5) AS Furnace, round(sum(Cellar),5) AS Cellar , round(sum(Fridge),5) AS Fridge , round(sum(Heater),5) AS Heater , round(sum(Lights),5) AS Lights, round(sum(Outlets),5) AS Outlets, round(sum(CookingRange),5) AS CookingRange, round(sum(WashingMachine),5) AS WashingMachine, round(sum(DishWasher),5) AS DishWasher from smarthomeconsumption WHERE HouseID='"H1"' group by houseid;
5	Home Owner	TOTAL (total of all appliances) ANNUAL consumption of myhouseID=H1. AND TOTAL ANNUAL consumption of all houses in that community for the same year.	select ((b.totalofmyhouse/d.totalofmycommunity)*100) AS MyHouseConsumptionPercent,b.totalofmyhouse AS TotalHouseCOnsumption,d.totalofmycommunity AS TotalCommunityConsumption from (Select MyFurnace+MyCellar+MyFridge+MyHeater+MyLights+MyOutlets+MyCookingRange+MyWashingMachine+MyDishWasher AS TotalOfMyHouse from (Select avg(Furnace) AS MyFurnace, avg(Cellar) AS MyCellar ,avg(Fridge) AS MyFridge ,avg(Heater)AS MyHeater , avg(Lights) AS MyLights, avg(Outlets) AS MyOutlets, avg(CookingRange) AS MyCookingRange, avg(WashingMachine) AS MyWashingMachine,avg(DishWasher) AS MyDishWasher from smarthomeconsumption WHERE HouseID='"H1"') as a) as b JOIN (Select MyFurnace+MyCellar+MyFridge+MyHeater+MyLights+MyOutlets+MyCookingRange+MyWashingMachine+MyDishWasher AS TotalOfMyCommunity from (Select avg(Furnace) AS MyFurnace, avg(Cellar) AS MyCellar ,avg(Fridge) AS MyFridge ,avg(Heater)AS MyHeater , avg(Lights) AS MyLights, avg(Outlets) AS MyOutlets, avg(CookingRange) AS MyCookingRange, avg(WashingMachine) AS MyWashingMachine,avg(DishWasher) AS MyDishWasher from smarthomeconsumption WHERE community='"Maliha"') as c) as d;

6	Community Level Utilities	Total Consumption of each house in a community every day in a week.	SELECT days, HouseID, round((MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+ MyCookingRange+MyWashingMachine+MyDishWasher),5) AS TotalHouseConsumption from (Select date_format(DateTime,'EEEE') AS days, houseid AS HouseID, round(sum(Furnace),5) AS MyFurnace, round(sum(Cellar),5) AS MyCellar ,round(sum(Fridge),5) AS MyFridge ,round(sum(Heater),5) AS MyHeater , round(sum(Lights),5) AS MyLights, round(sum(Outlets),5) AS MyOutlets, round(sum(CookingRange),5) AS MyCookingRange, round(sum(WashingMachine),5) AS MyWashingMachine,round(sum(DishWasher),5) AS MyDishWasher from smarthomeconsumption WHERE community='"Maliha"'AND weekofyear(DateTime)=3 group by HouseID, date_format(DateTime,'EEEE'))a;
7	Community Level Utilities	Total Consumption of each house in a community every week of a month	SELECT weeks, HouseID, round((MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+ MyCookingRange+MyWashingMachine+MyDishWasher),5) AS TotalHouseConsumption from (Select weekofyear(DateTime) AS weeks, houseid AS HouseID, round(sum(Furnace),5) AS MyFurnace, round(sum(Cellar),5) AS MyCellar , round(sum(Fridge),5) AS MyFridge ,round(sum(Heater),5)AS MyHeater , round(sum(Lights),5) AS MyLights, round(sum(Outlets),5) AS MyOutlets, round(sum(CookingRange),5) AS MyCookingRange, round(sum(WashingMachine),5) AS MyWashingMachine, round(sum(DishWasher),5) AS MyDishWasher from smarthomeconsumption WHERE community='"Maliha"'AND month(DateTime)=2 group by HouseID, weekofyear(DateTime))a;
8	Community Level Utilities	Total power consumption of each house in a community on weekdays	Select date_format(DateTime,'EEEE') AS weekday,houseid, Furnace AS CommunityFurnace, Cellar AS CommunityCellar ,Fridge AS CommunityFridge ,Heater AS CommunityHeater , Lights AS CommunityLights, Outlets AS CommunityOutlets, CookingRange AS CommunityRange, WashingMachine AS MyWashingMachine,DishWasher AS MyDishWasher from smarthomeconsumption WHERE community='"Maliha"' AND month(DateTime)=3 AND date_format(DateTime,'u') between 1 and 5 ;
9	Community Level Utilities	Total power consumption of each device in a community.	Select month(DateTime) AS monthly, sum(Furnace) AS Furnace, sum(Cellar) AS Cellar ,sum(Fridge) AS Fridge ,sum(Heater)AS Heater , sum(Lights) AS Lights, sum(Outlets) AS Outlets, sum(CookingRange) AS CookingRange, sum(WashingMachine) AS WashingMachine,sum(DishWasher) AS DishWasher from smarthomeconsumption WHERE community='"Maliha"' group by month(DateTime);
10	Community Level Utilities	How many users exist with a power consumption between 20 and 100 in the date range from"2014-03-01" to "2013-05-01"?	SELECT COUNT(HouseID) AS NumberOfHouses FROM (SELECT HouseID, round((MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+ MyCookingRange+MyWashingMachine+MyDishWasher),5) AS TotalHouseConsumption from (Select houseid AS HouseID, sum(Furnace) AS MyFurnace, sum(Cellar) AS MyCellar ,sum(Fridge) AS MyFridge ,sum(Heater)AS MyHeater , sum(Lights) AS MyLights, sum(Outlets) AS MyOutlets, sum(CookingRange) AS MyCookingRange, sum(WashingMachine) AS MyWashingMachine,sum(DishWasher) AS MyDishWasher from smarthomeconsumption WHERE Community='"Maliha"' AND month(DateTime) between 3 and 5 group by houseid)a)b WHERE TotalHouseConsumption between 20 AND 100 ;

11	Community Level Utilities	Total Consumption of each community of a state everyday in a week.	SELECT community,days, round(sum(myFurnace+myCellar+MyWashingMachine+MyOutlets+MyDishWasher+MyHeater+MyLights),5) as CommunityConsumption FROM (SELECT community,date_format(DateTime,'EEEE') as days, round(sum(furnace),5) as myFurnace, round(sum(cellar),5) as myCellar, round(sum(washingmachine),5) as MyWashingMachine, round(sum(outlets),5) as MyOutlets, round(sum(dishwasher),5) as MyDishWasher, round(sum(heater),5) as MyHeater, round(sum(lights),5) as MyLights from smarthomeconsumption where state='"Sharjah"'AND weekofyear(DateTime)=1 group by community,date_format(DateTime,'EEEE'))a GROUP BY community,days ;
12	State level Utilities	Total Consumption of each community of a state on a weekly basis.	SELECT community,weekly, round(sum(myFurnace+myCellar+MyWashingMachine+MyOutlets+MyDishWasher+MyHeater+MyLights),5) as CommunityConsumption FROM (SELECT community, weekofyear(DateTime) as weekly, sum(furnace) as myFurnace,sum(cellar) as myCellar,sum(washingmachine) as MyWashingMachine,sum(outlets) as MyOutlets,sum(dishwasher) as MyDishWasher,sum(heater) as MyHeater,sum(lights) as MyLights from smarthomeconsumption where state='"Sharjah"'AND month(DateTime)=1 group by community,weekofyear(DateTime))a GROUP BY community,weekly ;
13	State level Utilities	Total Consumption of each community of a state on a monthly basis.	SELECT community,months, round((myFurnace+myCellar+MyWashingMachine+MyOutlets+ MyDishWasher+MyHeater+MyLights),5) as CommunityConsumption FROM (SELECT community, month(DateTime) as months, sum(furnace) as myFurnace,sum(cellar) as myCellar,sum(washingmachine) as MyWashingMachine,sum(outlets) as MyOutlets,sum(dishwasher) as MyDishWasher,sum(heater) as MyHeater,sum(lights) as MyLights from smarthomeconsumption where state='"Sharjah"' group by community,month(DateTime))a GROUP BY community,months ;
14	State level Utilities	Total Consumption of each community of a state annually.	SELECT CommunityName, round((MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+ MyCookingRange+MyWashingMachine+MyDishWasher),5) AS TotalCommunityConsumption from (Select community AS CommunityName, sum(Furnace) AS MyFurnace, sum(Cellar) AS MyCellar ,sum(Fridge) AS MyFridge ,sum(Heater)AS MyHeater , sum(Lights) AS MyLights, sum(Outlets) AS MyOutlets, sum(CookingRange) AS MyCookingRange, sum(WashingMachine) AS MyWashingMachine,sum(DishWasher) AS MyDishWasher from smarthomeconsumption WHERE state='"Sharjah"' group by community)a ;
15	National level Utilities	Total Consumption of each state of a country everyday in a week.	SELECT mystate,daysOfWeek, round(sum(MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+MyCookingRange+MyWashingMachine+MyDishWasher),5) AS TotalStateConsumption from (Select state AS mystate,date_format(DateTime,'EEEE') as daysOfWeek, sum(Furnace) AS MyFurnace, sum(Cellar) AS MyCellar ,sum(Fridge) AS MyFridge ,sum(Heater)AS MyHeater , sum(Lights) AS MyLights, sum(Outlets) AS MyOutlets, sum(CookingRange) AS MyCookingRange, sum(WashingMachine) AS MyWashingMachine,sum(DishWasher) AS MyDishWasher from smarthomeconsumption WHERE weekofyear(DateTime)=5 group by state,date_format(DateTime,'EEEE')) a GROUP BY mystate,daysOfWeek;

16	National level Utilities	Total Consumption of each state of a country on a weekly basis.	SELECT mystate,weekly, sum(MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+MyCookingRange+MyWashingMachine+MyDishWasher) AS TotalStateConsumption from (Select state AS mystate,weekofyear(DateTime) as weekly, sum(Furnace) AS MyFurnace, sum(Cellar) AS MyCellar ,sum(Fridge) AS MyFridge ,sum(Heater)AS MyHeater , sum(Lights) AS MyLights, sum(Outlets) AS MyOutlets, sum(CookingRange) AS MyCookingRange, sum(WashingMachine) AS MyWashingMachine,sum(DishWasher) AS MyDishWasher from smarthomeconsumption where month(DateTime)=3 group by state,weekofyear(DateTime))a GROUP BY mystate,weekly;
17	National level Utilities	Total Consumption of each state of a country every month.	SELECT mystate, (MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+MyCookingRange+MyWashingMachine+MyDishWasher) AS TotalStateConsumption from (Select state AS mystate, avg(Furnace) AS MyFurnace, avg(Cellar) AS MyCellar ,sum(Fridge) AS MyFridge ,sum(Heater)AS MyHeater , sum(Lights) AS MyLights, sum(Outlets) AS MyOutlets, sum(CookingRange) AS MyCookingRange, sum(WashingMachine) AS MyWashingMachine,sum(DishWasher) AS MyDishWasher from smarthomeconsumption WHERE month(DateTime)=2 group by state)a ;
18	National level Utilities	Total Consumption of each state of a country annually.	SELECT mystate,monthly, sum(MyFurnace+MyCellar+MyHeater+MyLights+MyOutlets+MyCookingRange+MyWashingMachine+MyDishWasher) AS TotalStateConsumption from (Select state AS mystate,month(DateTime) as monthly, sum(Furnace) AS MyFurnace, sum(Cellar) AS MyCellar ,sum(Fridge) AS MyFridge ,sum(Heater)AS MyHeater , sum(Lights) AS MyLights, sum(Outlets) AS MyOutlets, sum(CookingRange) AS MyCookingRange, sum(WashingMachine) AS MyWashingMachine,sum(DishWasher) AS MyDishWasher from smarthomeconsumption group by state,month(DateTime))a GROUP BY mystate,monthly;

Chapter 4. Experimental Design

Four machines are taken for setting up a cluster consisting of one master-four slave nodes as shown in Figure 4.1. The replication factor for each node is set to 2 and the block size is 64MB. For a million smart meters data (~1.5TB), the name node generates 23438 blocks (1.5TB/64MB) of data. Each data nodes stores a total of 5860 blocks that correspond to a total size of 375GB. The block distribution from B1 to B5860, B5861 to B11720, and B11721 to B17580, and B17581 to B23440 across node 1, node 2, node 3, and node 4 respectively is shown in Figure 4.1. The hardware specifications for deploying a single master or a slave node machine is in Table 4.1.

Table 4.1: Commodity Hardware Specifications

	CPU RAM		Storage	OS	Ethernet speed (Cat6 cables)	IP address (local)
Master Node	8 cores at 2GHz	16GB	1TB	Ubuntu 16.4	1000Mbps	10.25.34.176
Slave Node 1	8 cores at 2GHz	16GB	1TB	Ubuntu 16.4	1000Mbps	10.25.34.176
Slave Node 2	4 cores at 1.5GHz	8GB	730GB	Ubuntu 16.4	1000Mbs	10.25.34.177
Slave Node 3	4 cores at 1.5GHz	8GB	500 GB	Ubuntu 16.4	1000Mbps	10.25.34.178
Slave Node 4	4 cores at 1.5GHz	8GB	500GB	Ubuntu 16.4	1000Mbs	10.25.34.179

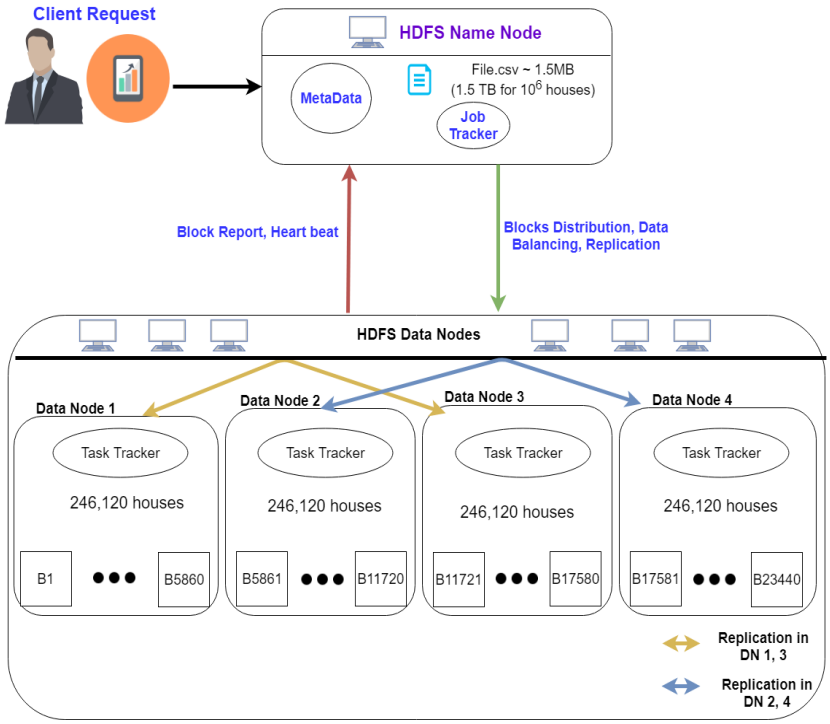

Figure 4.1: One master-four slaves hadoop cluster

4.1. Evaluation Criteria

Performance is critical in Hadoop clusters whether it's deployed on bare metal or virtualized environment. Our cluster was deployed on physical environment of the machines. Three types of variables are determined for evaluating cluster performance: input variables, output variables, control variables as shown in Table 4.2.

Table 4.2: Experimental Variables under study

Input Variables	Control Variables	Output Variables
Number of Nodes	Number of Cores	Latency
Number of Data files	Memory Size	Throughput
Number of Queries	Network Bandwidth	

The control variables are based on the physical environment and machine specifications as already mentioned in the previous section. The input variables are the optimization parameters that can be controlled by the user during Hadoop Job submission and queries to optimize the performance. Four output variables will be assessed by varying different input variables; latency and throughput. The term 'latency' refers to the completion time of rendering query results across the cluster data nodes. Similarly, 'throughput' denotes the number of reads from HDFS completed per unit time (tasks completed/minute). The data files will be imported in log scale of 10, 100, 1000, 10000, 100000, and 1,000,000 smart meter CSV files. Each batch of files will be loaded across one, two, and three data nodes to examine the latency, throughput, and memory usage of write to HDFS. To evaluate the cluster performance, experiments will be focused on the data querying and reporting on Hadoop cluster using the two frameworks; Apache Spark and Apache Hive.

4.2. Experiments and Evaluation Use-Cases

Two experiments are designed to analyze performance of the SQL querying engines on top of Hadoop in terms of latency and throughput.

4.2.1. Experimental objective. To determine the total latency for data querying each query is scheduled to run 100 times using the Oozie workflow [43] tool on top of Hadoop. For each query run, there are two variables i.e. number of data files and number of nodes. Experiments will be conducted to evaluate the overall query

performance across the two data querying frameworks of Hadoop; Apache Spark and Apache Hive.

4.2.2 Experimental objective. To determine the system throughput in data querying throughput is based on the file size written (or read) by the individual map tasks and the elapsed time to do so.

Two test cases will be generated to determine the elapsed time (latency) and throughput for executing queries on Spark and Hive. The test cases are as follows:

- Number of Smart meter files: The impact of the data files size during the querying process is determined by measuring the elapsed time (latency), throughput for each batch of data files.
- Number of Data nodes: The impact of the cluster size on querying is determined by measuring the latency, throughput, for submitting query jobs across one, two, three, and four data nodes.

Chapter 5. Results and Discussion

This chapter comprises of quantitative results discussion from the performance evaluation as discussed in the previous chapter. Section 1 elaborates on two performance metrics; query execution time and throughput. Section II discusses the Hadoop processing engines performance with proprietary tools and relational database management system.

5.1. Quantitative Evaluation

In order to obtain statistically accurate data, each query was scheduled in a workflow using the Oozie Workflow Scheduler system to manage Apache Hadoop jobs. Oozie [43] is a scalable and a reliable system integrated on top of the Cloudera Hadoop Stack supporting different types of Hadoop jobs such as map reduce, multidimensional processing, etc. Each query workflow is scheduled to run every fifteen minutes on a day, and for each query execution hundred points of latency are logged to gain statistical significance. Throughput is calculated from the latency points obtained from the experiments.

5.1.1. Latency. Hadoop is designed and developed to process large number of files. Hadoop's mean execution time was evaluated by investigating different factors, such as the number of active slave nodes and dataset size that affect it, i.e.,

$$Hadoop\ execution\ time(seconds)\ \alpha\ \ Number\ of\ Smart\ Meters$$

$$Hadoop\ execution\ time(seconds)\ \frac{1}{\alpha}\ Number\ of\ Active\ Slave\ Nodes$$

As a query job is submitted via Spark or Hive CLI, the job is executed in multiple stages where each stage contains multiple Map Reduce tasks. We use the following notation to represent an big data SQL query job:

$$Job = \{Stage_i : 0<i<M\}$$

$$Stage = \{Task_{i,j} : 0<j<N\}$$

Here M is the number of stages in a job and N is the number of Map Reduce tasks in a stage. The resource manager of master node distributes these stages across the cluster nodes. The map/reduce task processes assigned to each stage are executed

in parallel across several data notes. Latency corresponds to the total execution time taken by all the mapper/reducers tasks within each stage when run in parallel to process results.

Since the standard deviations are unevenly distributed, the query performance will be evaluated based on the average values of latency as well as throughput. Figure 5.1 and Figure 5.2 depict Query-wise mean execution time for Hive and Spark across 1 node and nodes 4 for one million files. From the two graphs, it is observed that the query execution time largely depends on the query selectivity characteristic as well. For queries formulated with large selectivity parameters such as Query #1, Query #11, Query #15, Query #16, Query #17, and Query #18 take longer time to process than other queries for the same processing volume across each node. Query #11 takes a long execution time because of multiple JOIN clauses used in this query to find the total house consumption and its respective neighborhood community consumption. Due to multiple JOINs, there is a strong interdependency between the files. With larger interdependency between smart meters dataset, more numbers of reducers strongly dependent on each other. With this increase in the number of reducers, the execution time increases sharply depending on varying computing needs. The maximum time is taken from Q #15 to Q # 18 as these queries have a wider selectivity requirement to aggregate more number of records on a national level spread across different cluster nodes, and hence larger latency. For one node, Hive and Spark take roughly 2322 seconds and 2170 seconds for Query # 18 respectively. Conversely, for four nodes Hive and Spark take 843 seconds and 712 seconds for same query respectively. Additionally, it can be observed from both the figures that in case of Spark, Query #1 takes longer than other queries with the same selectivity clause such as Query #2 because for the first iteration Spark launches the container memory and once the processing is completed other memory containers are not stopped until the session is closed which saves a significant amount of overhead time. However, for Hive when a query is submitted the map and reduce tasks will be launched on the execution stage and these tasks are alive until the period of execution. Once the processing is completed, the map and reduce memory containers will get stopped and will be relaunched again for the subsequent query session.

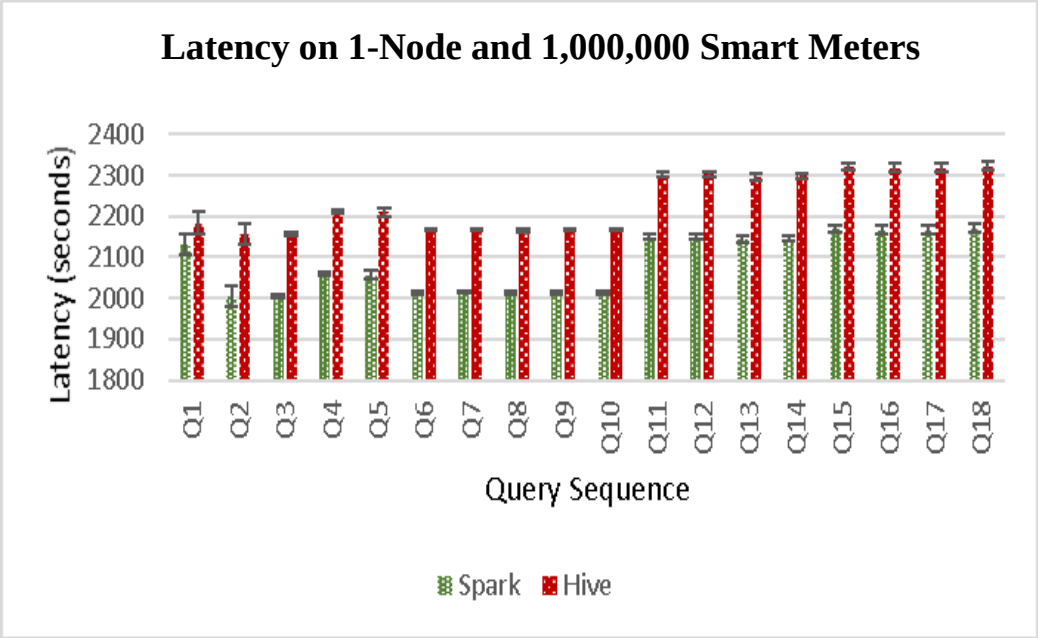

Figure 5.1: Mean latency per query for 1 million smart meters across 1-Node

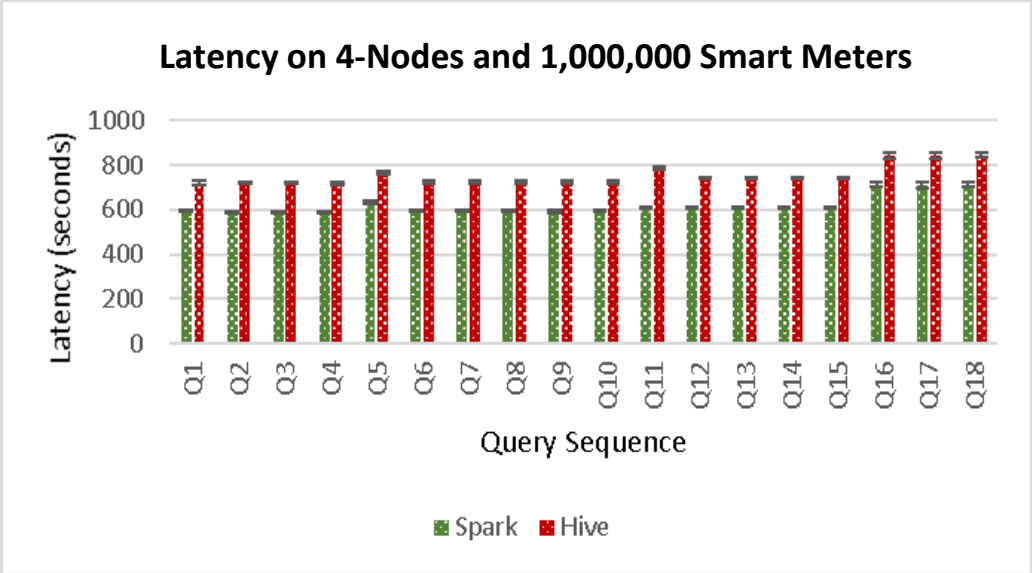

Figure 5.2: Mean latency per query for 1 million smart meters across 4-Nodes

The results for the query execution time on Apache Spark and Apache Hive are shown in Figure 5.3. The results are discussed for cluster size of 1-Node, 2-Nodes, 3-Nodes, and 4-Nodes as follows:

5.1.1.1. ***1-node cluster.*** For processing 10 smart meters on one 1 node, the execution time for Spark and Hive is 34 seconds and 39 seconds with a standard deviation of 4.6 and 15.1 seconds respectively. This accounts to Spark being 14%

faster than Hive. For processing 100 smart meters, Spark's and Hive's execution time is 232 seconds and 265 seconds with standard deviation of 28.78 and 12.289 respectively. Thus, Spark being 14.8% faster than Hive. For a 1000 smart meters, the processing time for Spark and Hive is 829 and 954 seconds with a standard deviation of 114.1992 and 60.47 respectively, yielding Spark as 15% faster than Hive. For a 10000 smart meters, Spark and Hive latency is reported as 1201 seconds and 1364 seconds with a standard deviation of 40.79 and 25.6579. Thus, Spark is 13.5 % faster than Hive. For 100,000 meters, Spark and Hive's response time is 1476 and 1675 seconds with standard deviation of 43.8 and 51.07 respectively implying that Spark is 13.4% faster than Hive. For a million smart meters, Spark and Hive's processing time is 2172 and 2221 seconds with a standard deviation of 70.31 and 70.66 respectively. This infers that Spark is 2.3% faster than Hive. The reason for Spark being faster than Hive is that the Spark processes the data in -memory while the Hive processing utilizes disk access. With in-memory data processing on Spark, the processing speed is increased substantially. This is because using Spark, the data is cached in the first query run and it does not require to be fetched from the disk again for the subsequent query sessions. For the first 10, 100, and 1000 smart meters Spark's performance is increasing linearly with respect to Hive wherein Spark is 14%, 14.8% and 15% faster than Hive. However, as the number of input volume increases to 10,000, 100,000 and 1,000,000 smart meters, the performance of Spark is only 13.5%, 13.4% and 2.3% faster than Hive. This is because of the memory constraint on a single node comprising of 540 GB RAM for processing 1.5 TB data that interferes with the Spark's in memory processing. The reason being that for an increased input size coupled with a smaller cluster design (~1 node), Spark SQL is unable to handle the intermediate result sets (i.e. 1.5 TB for 1 million files, 0.15 TB for 100,000 files) in the available container memory whereas Hive is able to cope with such a situation due to read/write fetch from the disk. Due to this memory bottleneck, all records from the smart meter files cannot be cached into the memory due to which they are forced to be written on to the disk, causing high latency in Spark.

5.1.1.2. ***2-nodes cluster.*** The processing time for 10 smart meters on two nodes, the execution time for Spark and Hive is 31 seconds and 35.04 seconds with a standard deviation of 4.44 and 5.56 respectively. This accounts to Spark being 13.61% faster than Hive. For processing 100 smart meters, Spark's and Hive's

execution time is 108 seconds and 120 seconds with a standard deviation of 5.96 and 5.93 respectively. Thus, Spark being 11% faster than Hive. For a 1000 smart meters, the processing time for Spark and Hive is 734 and 809 seconds 83.63 and 70.39 respectively, yielding Spark as 12.2% faster than Hive. For a 10000 smart meters, Spark and Hive latency is reported as 1031 seconds and 1140 seconds with a standard deviation of 76.75 and 71.42. Thus, Spark is 10.5 % faster than Hive. For 100,000 meters, Spark and Hive's response time is 1202 and 1364 seconds with a standard deviation of 48.33 and 51.13 respectively implying that Spark is 11.4% faster than Hive. For a million smart meters, Spark and Hive's processing time is 1751 and 1927 seconds with a standard deviation of 99.14 and 102.56 respectively. This infers that Spark is 10.5% faster than Hive. For every set of files, it is observed that Spark's performance is greater than Hive. Thus, it can be deduced that even though Spark outperforms Hive for every batch of files, the increase in performance declines as the number of input files change from 10,000 to one million in a small cluster set of two nodes. This is due to the memory intensive computation power of Spark wherein it splits some of the intermediate records onto the disk when it falls short of commodity hardware memory. For a cluster of two nodes, the total available memory for processing was 960 GB (Node-1) and 589 GB (Node-2) RAM, that is, 1549 MB ~ 1.5 GB whereas the size of input volume for 10,000 , 100,000 and 1 million files was 15 GB, 150 GB, and 1.5 TB.

5.1.1.3. ***3-nodes cluster.*** The processing time for 10 smart meters on three nodes node, the execution time for Spark and Hive is 10.5 seconds and 12 seconds with a standard deviation of 2.68 and 2.85 seconds respectively. This accounts to Spark being 14.28% faster than Hive. For processing 100 smart meters, Spark's and Hive's execution time is 97 seconds and 120 seconds with a standard deviation of 11.05 and 9.2 respectively. Thus, Spark being 23.7% faster than Hive. For a 1000 smart meters, the processing time for Spark and Hive is 379 and 482 seconds with a standard deviation of 19.09 and 45.76 respectively, yielding Spark as 27% faster than Hive. For a 10000 smart meters, Spark and Hive latency is reported as 454 seconds and 592 seconds with a standard deviation of 12.38 and 10.28 respectively. Thus, Spark is 30.5 % faster than Hive. For 100,000 meters, Spark and Hive's response time is 789 and 1093 seconds with a standard deviation of 32.46 and 30.75 respectively implying that Spark is 38% faster than Hive. For a million smart meters, Spark and Hive's

processing time is 977 and 1295 seconds with a standard deviation of 87.166 and 93.175 respectively. This infers that Spark is 32.5% faster than Hive. Thus, with the increase in nodes, the execution time for both Spark and Hive increased linearly, and there is a substantial increase in the performance upgradation of Spark in contrast to Hive as the volume of dataset increases from 10 to one million smart meters without any memory bottleneck.

5.1.1.4. ***4-nodes cluster.*** The processing time for 10 smart meters on three nodes node, the average execution time for Spark and Hive is 5.4 seconds and 6.3 seconds with a standard deviation of 1.53 and 1.55 respectively. This accounts to Spark being 15.66% faster than Hive. For processing 100 smart meters, Spark's and Hive's execution time is 87 seconds and 101 seconds with a standard deviation of 19.49 and 18.69 respectively. Thus, Spark being 16.7% faster than Hive. For a 1000 smart meters, the processing time for Spark and Hive is 258 and 329 seconds with a standard deviation of 43.788 and 45.54 respectively, yielding Spark as 28% faster than Hive. For a 10000 smart meters, Spark and Hive latency is reported as 400 seconds and 540 with a standard deviation of 10.921 and 10.27 seconds correspondingly. Thus, Spark is 35 % faster than Hive. For 100,000 meters, Spark and Hive's response time is 546 and 758 seconds with a standard deviation of 69.01422 and 75.065 respectively implying that Spark is 44.3% faster than Hive. For a million smart meters, Spark and Hive's processing time is 603 (~13 minutes) and 1089 (~21 minutes) seconds with a standard deviation of 44.433 and 45.25 respectively. This infers that Spark is 76% faster than Hive. To summarize, for 10, 100, 1000, 10000, 100,000 and 1,000,000 smart meters Spark is 15.6%, 16.7%, 28%, 35%, 43%, and 76% faster than hive respectively. The percentage increment is less for small batch of files such as 10, 100, and 1000. This is because running a query on dataset which has only a few hundreds of MBs written to disk is not much different than transferring the same in memory. Now coming to processing a volume of size in thousands of MBs or in TBs i.e. 10000, 100000, and one million smart meters, Hive's intermediate results from the map outputs are written to the disk and then transferred to the reducers which take a significantly large amount of time. Additionally, when a hive query is submitted, the mapper and reducer tasks will be launched. These tasks However, in Spark this is not the case as the intermediate results are cached in-memory avoiding any hefty disk I/O. Thus, overall it can be inferred that with a cluster of four nodes the

Spark performance increases substantially for each set of files. The best processing performance for a million meters is achieved with a cluster size of four nodes. Thus, for processing a larger batch of files, the addition of nodes to the cluster has a big impact on reducing the execution time.

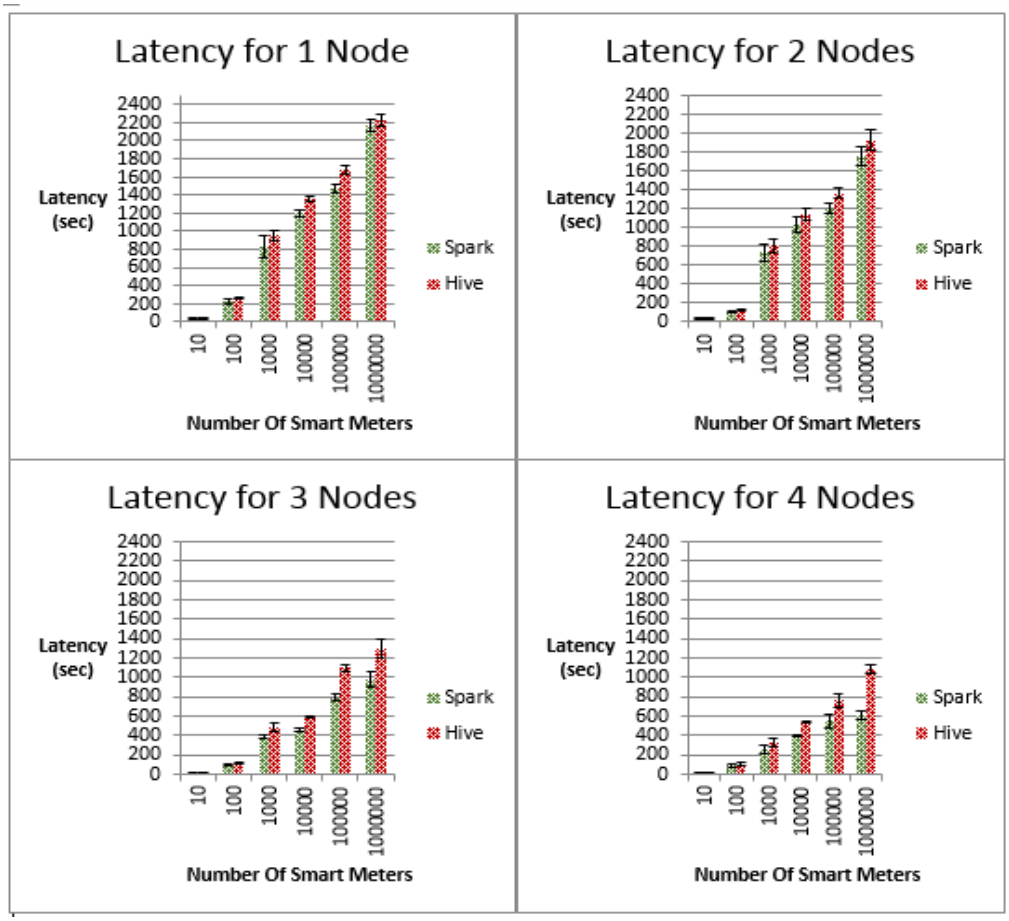

Figure 5.3: Mean latency across 1-Node, 2-Nodes, 3-Nodes, and 4-Nodes

Additionally, the latency for processing 100,000 files across one node in Spark and Hive is 2172 and 2221 seconds respectively. Conversely, the processing time for two nodes and the same processing engines (Spark and Hive) is 1751 and 1927 seconds respectively. Thus, for two nodes Spark is roughly 7 minutes faster than one node computation. On the other hand, Hive processing across two nodes is approximately 4 minutes faster than on a single node. For three nodes and one million smart meters, the latency on Spark and Hive is 977 seconds and 1295 seconds. Thus, Spark on three nodes is nearly 12 minutes faster than two nodes. Alternatively, Hive processing on three nodes is roughly 10 minutes faster than that on two nodes. Similarly, for four nodes the Spark and Hive processing time is 603 and 1029 seconds

respectively. This infers that Spark on four nodes is nearly 6 minutes faster than its processing the same set of million files on three nodes. Conversely, Hive on four nodes is approximately 6 minutes faster than computation time on three nodes. This small variation in the increase in processing time between three and four nodes is attributed to the network bottleneck (~1 Mbps Ethernet bandwidth and CAT6 cables providing 100 Mb/s) in a cluster as it takes longer time to read and write the data stored across more number of nodes of the cluster in the network. To minimize this network constraint for achieving lower latency, high speed Ethernet cables and higher bandwidth switch can be deployed for connecting the cluster machines. Thus, it can be inferred that latency reduces as the number of active nodes increase and the processing time increases with the increase in number of files. The obtained results are reasonable as it is expected that with increase in the volume of data files on each node, the processing time will also increase significantly. Theoretically, a linear improvement is observed in the performance of Spark and Hive with increase in cluster size and subsequent memory resources. Although, in the practical implementation there will be some overhead due to network communication and synchronization between the cluster nodes, yet the performance for a large cluster size will be better than a small cluster. The performance gain is inversely proportional to the latency or the query execution time in seconds, and can be expressed as follows:

$$Performance = \frac{1}{Latency\ in\ Seconds}$$

In practicality, the performance gain also depends upon other parameters such as the optimization and execution plan generated by the processing engines. Figure 5.4 represents the performance gain achieved for processing one million records using Spark and Hive across 1 node, 2 nodes, 3 nodes, and 4 nodes. The processing gain achieved for a small cluster is lower in contrast to the gain obtained with the increase in cluster size. Spark's performance gain is always higher than Hive. Additionally Figure 5.5 illustrates the performance of Query # 1 in on a million smart meters on a cluster of four nodes. From the graph we can observe the effect of Spark in an iterative querying environment. It is surmised that with 1.5 TB (million meters) data, Query #1 performance, Hive takes a constant time per iteration of about 720 seconds.

On the other hand, Spark takes 780 seconds in first iteration to load data in-memory and only ~120 seconds in subsequent iterations.

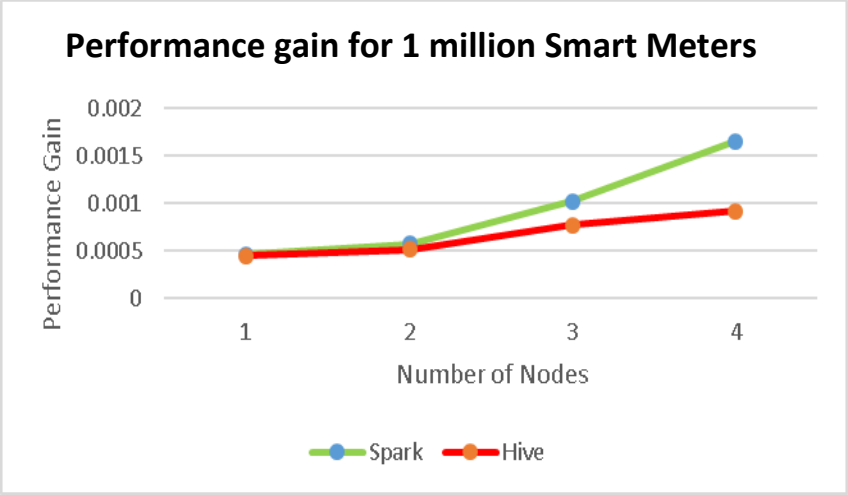

Figure 5.4: Performance gain in Hive and Spark for a million smart meters

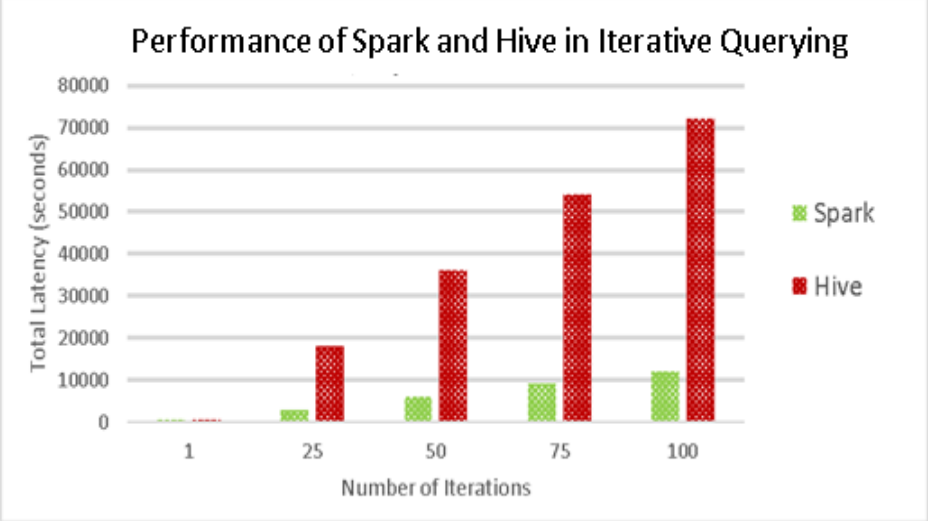

Figure 5.5: Performance of Spark and Hive in iterative querying

5.1.2. Throughput. Throughput refers to the amount of data executed, per second, for each query execution. Following expression is used to calculate throughput based on input file size and latency measurements. Figure 5.6 and Figure 5.7 represent the

$$Total\ throughput\ \left[\frac{MB}{sec}\right] = \frac{Number\ of\ Smart\ Meter\ Files}{Latency\ (sec)}\ X\ Size\ of\ 1\ File\ (MB)$$

$$Total\ throughput\ \left[\frac{MB}{sec}\right] = \frac{\sum File\ Size}{Latency\ (sec)},\ \text{File size=1.5MB}$$

throughput of Hive and Spark for processing one million files across the cluster size of node 1 and nodes 4 respectively. It can be summarized that Queries with larger latency have a smaller throughput such as Query #1, Query #11, Query #15, Query #16, Query #17, and Query #18 with throughput values 686.669 MBps, 651.0047 MBps, 646.1469 MBps, 646.9373 MBps, 646.9897 MBps, and 645.8848 MBps respectively across 1-node. Alternatively for Figure 5.7, it is observed that the throughput measurements are higher across the cluster size of 4-nodes in contrast to 1-node cluster. This result matches the theoretical expectation as latency across four nodes is reduced in comparison to node one, hence higher throughput. Hadoop scalability to accommodate large volume of data across multiple nodes using Spark is excellent with an optimum amount of execution time of around 15 minutes to process a million smart meter data. The mean throughput across each node for every batch of file across Spark and Hive is shown in Figure 5.8. From the experimental results, it is observed that for a set of files (e.g. 1,000,000 files) the maximum throughput is achieved at the addition of four nodes in the cluster. Mean throughput for Spark and Hive with 1 million processing volume across four nodes is recorded as 2433 MBps and 1977 MBps with a standard deviation of 158.42 MBps and 110 MBps. This could be accounted for the fact that with the addition of a node the processing time decreases for executing queries on the same number of files. Moreover, as the processing volume across each node increases, the throughput is also observed to be increased sharply. Overall, Spark outperforms Hive in latency but with a tradeoff in available memory resources.

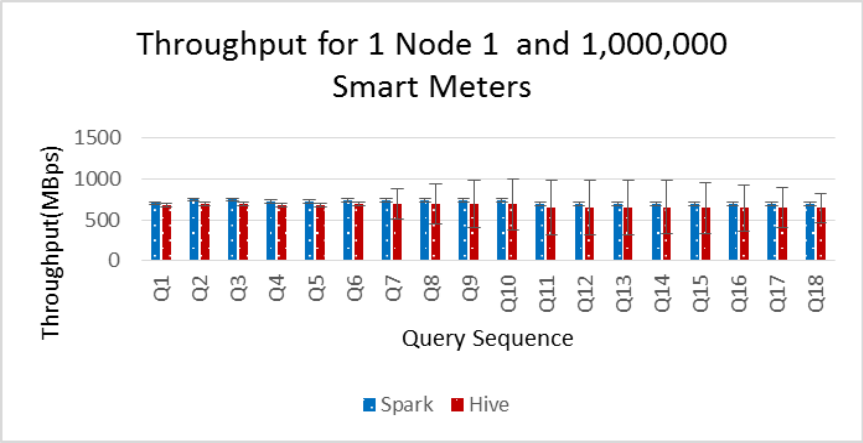

Figure 5.6: Mean throughput result for Spark and Hive across Node 1- 1 million smart meters

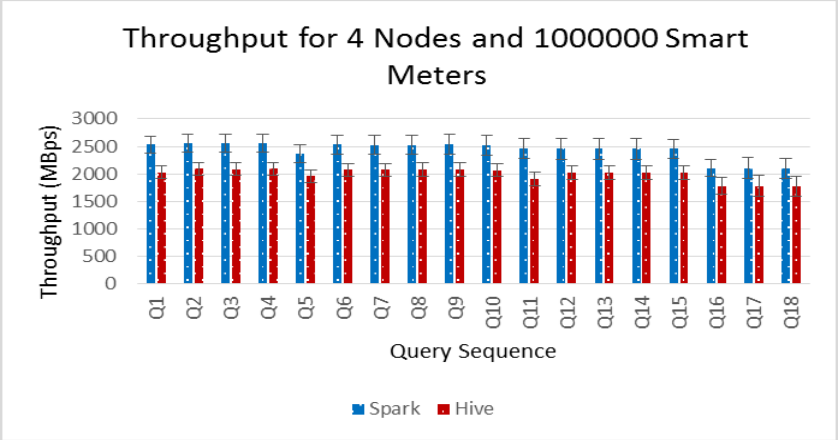

Figure 5.7: Mean throughput result for Spark and Hive across 4 Nodes and 1 million smart meters

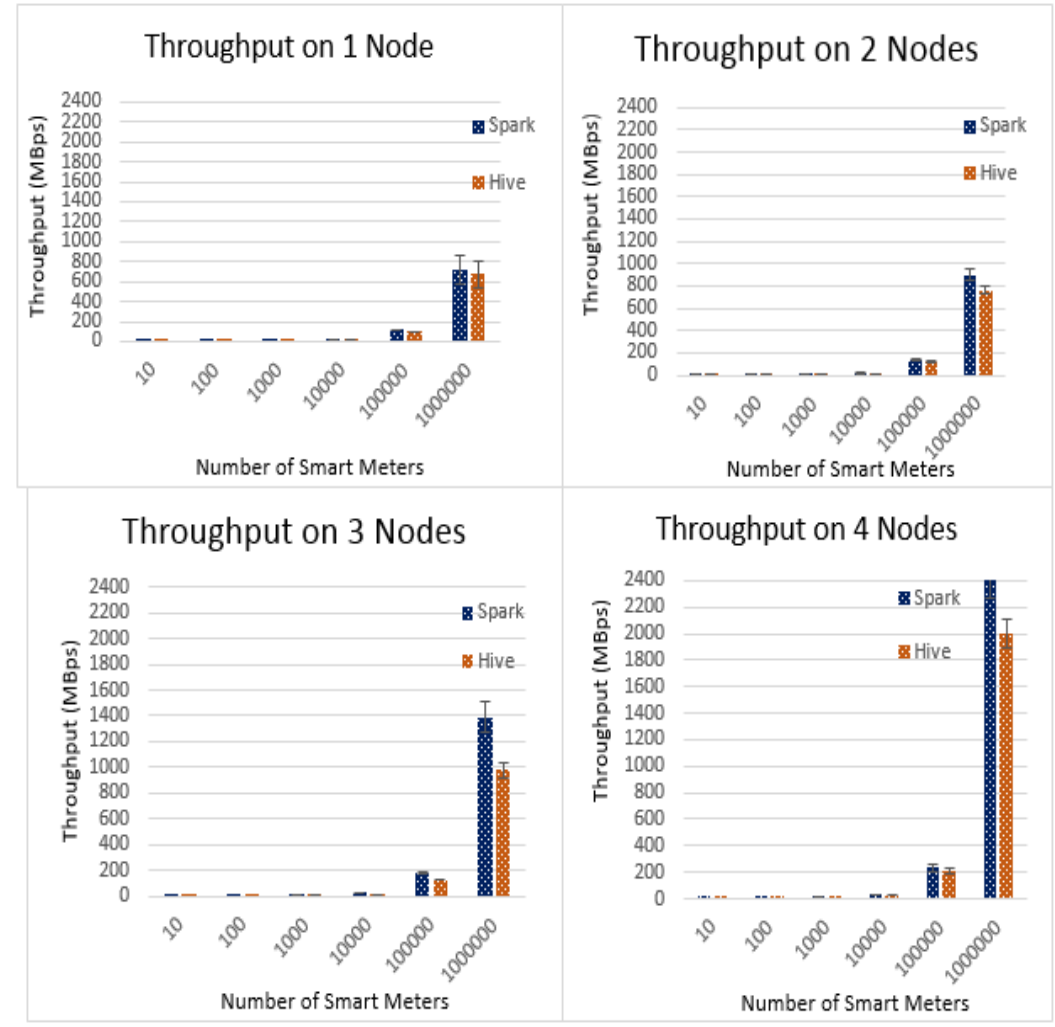

Figure 5.8: Mean throughput result for Spark and Hive across 1 Node, 2 Nodes, 3 Nodes, and 4 Nodes

5.2. Comparison of Experimental Results with Proprietary Tools and RDMS

IBM's Informix big data tool and relational databases from [45] were used as a benchmark to compare the processing time for running queries on one million smart meters data. Table 5.1 provides a synopsis of performance comparison between the experimental results for Spark and Hive with respect to IBM's proprietary tool and relational database management system.

Table 5.1: Comparison of Hadoop processing engines with IBM proprietary tool and relational database management system [45]

	IBM Proprietary Tool	Relational Database	Spark	Hive
Run time for 1 million meters	25 sec to 6 min	2-7 hours	12 to 15 minutes	28-34 minutes
Storage required for 1 million meters	350GB	1.3TB	1.5TB (w/o replication)	1.5 TB (w/o replication)

From Table 5.1, we can deduce that for a million smart meters processing, Spark and Hive have an intermediate performance in comparison to IBM's proprietary tool and relational database management system. Relational database management system perform the worst for one million smart meters processing that require a storage of 1.3 TB. For 1.5 TB files size, Spark's processing time was 12-15 minutes for one million smart meters whereas Hive took about 28-34 minutes. On the other hand, IBM's processing tool took the least time ranging from 25 seconds to 6 minutes. However, the size of one million smart meters files was only 350 GB for IBM's tool while it was 1.5 TB for Spark and Hive. Thus, it can be concluded that for the same storage of 1.5 TB for one million smart meter dataset, the IBM proprietary tool could take longer than Spark and Hive processing time.

A large data processing procedure which might take hours of processing time on a centralized relational database might take just roughly 15 minutes when the same data is distributed across Hadoop cluster with several nodes given all processing be done in parallel. The residual graphs of latency and throughput for query wise execution across each node can be found in the appendix section.

Chapter 6. Visualization on Hadoop for Smart Meter Data

Hue is a web-based interactive query editor in the Hadoop stack of Cloudera that allows data visualization in real time [44]. Hue utilizes hive SQL Query engine to generate graphs for different levels of stakeholders as discussed in Chapter 3. For visualization purpose, we sampled 10 smart meters data from the generated dataset in order to demonstrate an example of how visualization can be performed on Hadoop ecosystem.

6.1. Consumer

- Query #1: Total consumption of each appliance for consumer's house everyday in a week. Figure 6.1 represents the graphical output for Query #1.

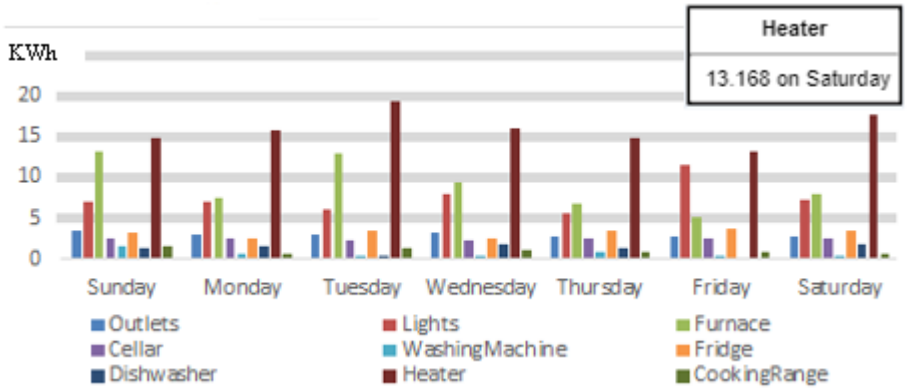

Figure 6.1: Consumer's home appliances consumption for each day in a week

- Query #2: Total consumption of each appliance for consumer's house each week in a month. Figure 6.2 represents the output for Query #2 in tabular format.

				KWh					
week	furnace	cellar	heater	lights	outlets	cooking range	washing machine	fridge	dishwasher
1	74.5	19.3	128.1	58.4	23.7	7.9	4.3	24.6	4.3
2	57.1	20.7	107.6	60.6	22.5	5.6	3.9	25.8	3.9
3	70.8	20.8	124.2	44.7	21.7	6.6	7.9	26.5	7.8
4	67.6	14.8	93.5	39.2	16.2	5.4	3.7	21.8	3.7

Figure 6.2: Consumer's home appliances consumption for each week in a month

- Query #3: Total consumption of each appliance for consumer's house on a monthly basis. Figure 6.3 represents the graphical output for Query #3.

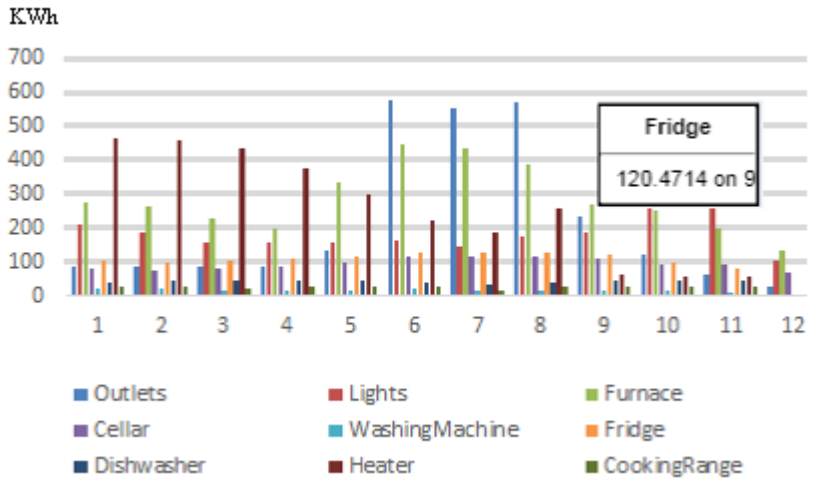

Figure 6.3: Consumer's home appliances consumption for each month of the year

- Query #4: Total annual power consumption of each home appliance of the consumer. Figure 6.4 represents the graphical output for Query #4.

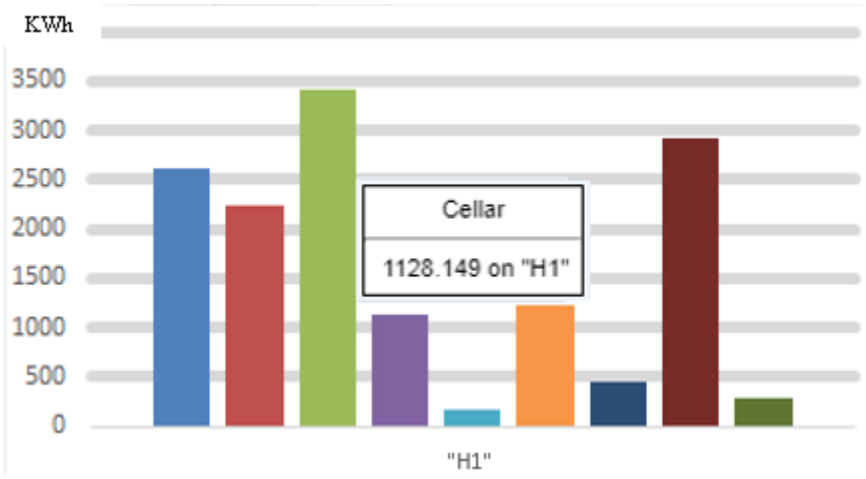

Figure 6.4: Total annual consumption of each home appliance for the consumer

- Query #5: Total annual consumption of consumer's house and total annual consumption of all houses in that respective community for the same year. . Figure 6.5 represents the output for Query #5 in tabular format.

myhouseConsumption.percent	b.totalofmyHouse [KWh]	b.totalofmyCommunity [KWh]
23.46	14440.95	60585.28

Figure 6.5: Consumer's annual consumption in percentage with respect to the community's total consumption

6.2. Community Utility Provider

- Query #6: Total consumption of each house in a community for everyday in a week. Figure 6.6 represents the graphical output for Query #6.

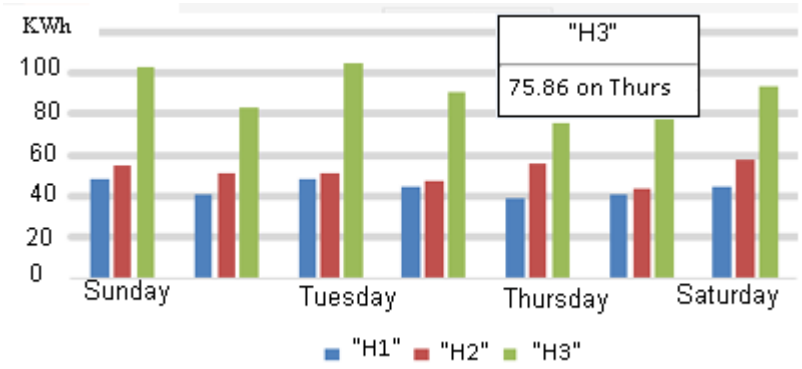

Figure 6.6: Total annual consumption for all houses in a community each day

- Query #7. Total consumption of each house device in a community a week. . Figure 6.7 represents the graphical output for Query #7.

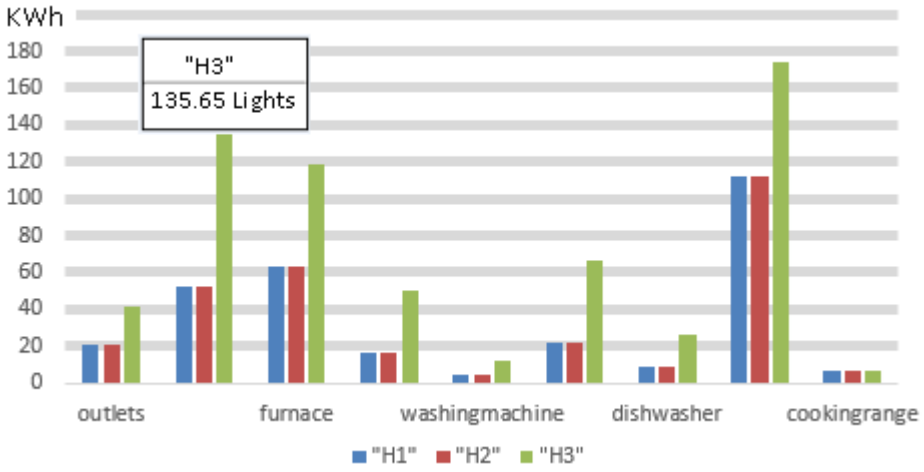

Figure 6.7: Appliance consumption for all houses in a community for one week

- Query #8. Total consumption of each house in a community every week of a month. Figure 6.8 represents the graphical output for Query #8.

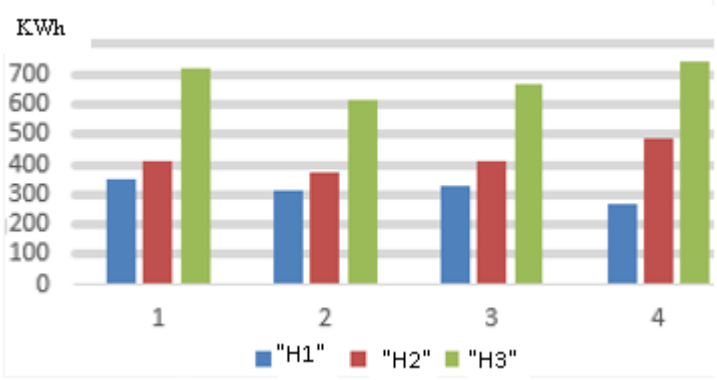

Figure 6.8: Total consumption of each house in a community every week of the month

- Query #9: Total consumption of each house in a community every month of the year. Figure 6.9 represents the graphical output for Query #9.

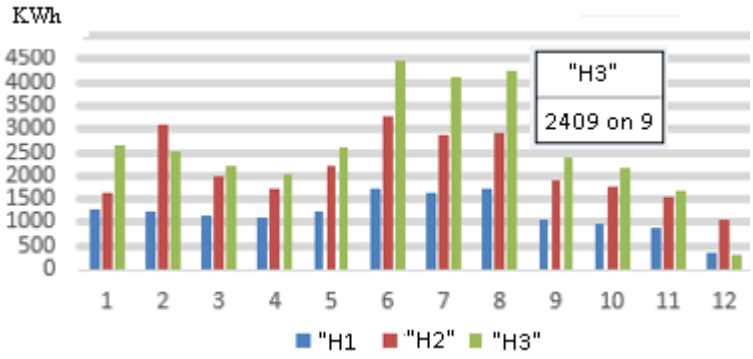

Figure 6.9: Total consumption of each house in a community on a monthly basis

- Query #10: Total consumption of each house in a community annually. . Figure 6.10 represents the graphical output for Query #10.

6.3. State Utility Provider

- Query #11: Total consumption of each community of a state every day in a week. . Figure 6.11 represents the graphical output for Query #11.

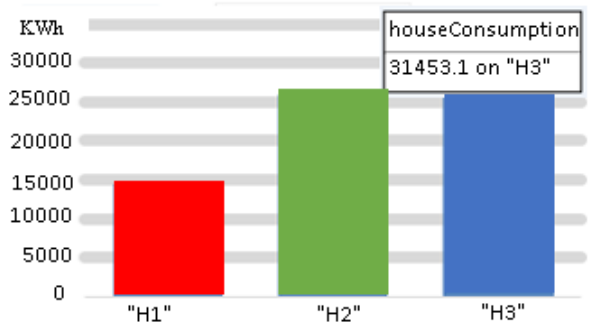

Figure 6.10: Total annual consumption of each house in a community

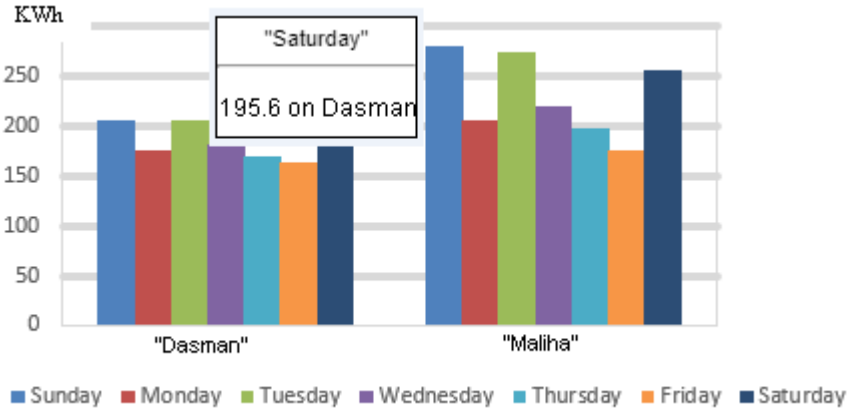

Figure 6.11: Total consumption of each community of a state every day in a week

- Query #12: Total consumption of each community of a state for each week in a given month. Figure 6.12 represents the graphical output for Query #12.

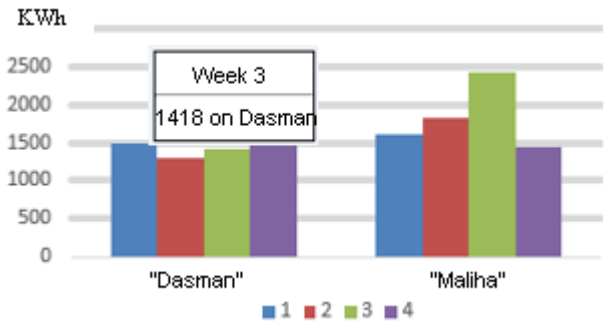

Figure 6.12: Total consumption of each community of a state every day in a week

- Query #13: Total consumption of each community of a state on a monthly basis. Figure 6.13 represents the graphical output for Query #13.

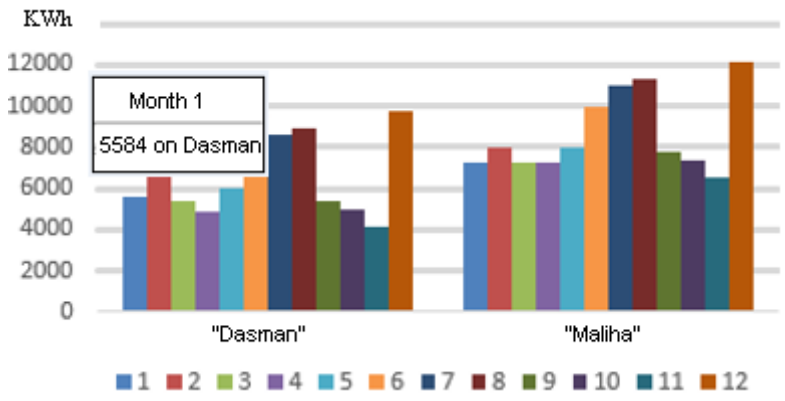

Figure 6.13: Total consumption of each community of a state on a monthly basis

- Query #14: Total annual consumption of each community of a state. Figure 6.14 represents the graphical output for Query #14.

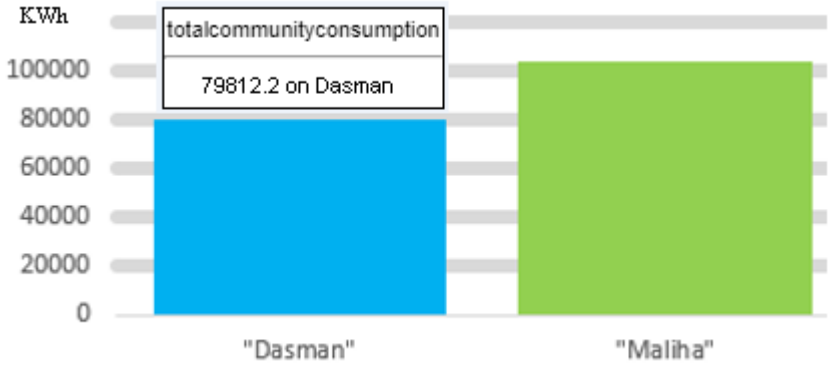

Figure 6.14: Total annual consumption of each community of a state

6.4. National Utility Provider

- Query #15: Total consumption of each state of a country everyday in a week. Figure 6.15 represents the graphical output for Query #15.

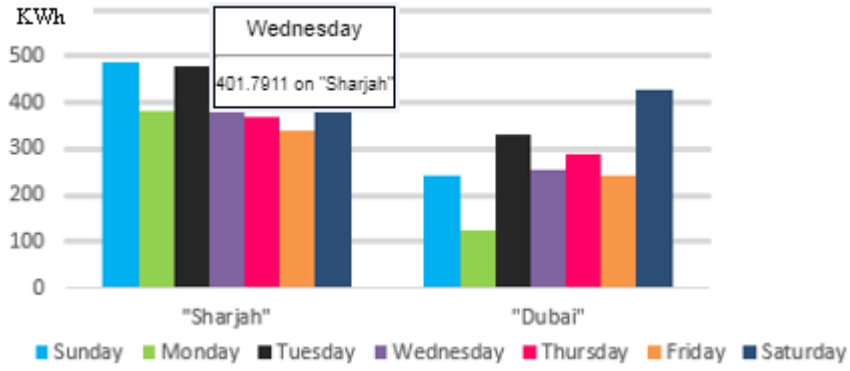

Figure 6.15: Total consumption of each state of a country every day in a week

- Query #16: Total consumption of each state of a country on a weekly basis. Figure 6.16 represents the graphical output for Query #16.

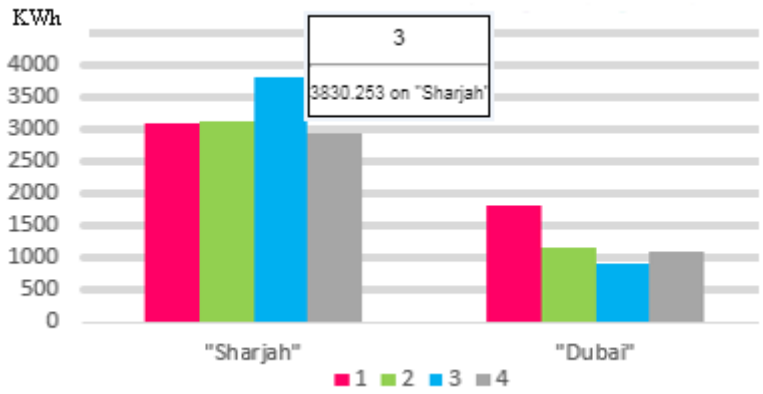

Figure 6.16: Total consumption of each state of a country weekly basis

Chapter 7. Conclusion, Limitations and Future Work

The goal of this thesis was to build a high end commodity hardware computing cluster and utilizing the open source data storage, processing, analyzing, and visualizing residential area big data. A dataset for one year of everyday energy consumption using smart meter was obtained from a research team at University of Massachusetts. In this thesis, using ARIMA modeling synthetic data was generated for one million smart meters using the sample dataset. The one million meter was geographically distributed into the seven UAE emirates based on the latest population percentage in each emirate.

A four nodes computing cluster was designed and configured using high end quad core CPU's and large RAM and disk storage. The open source file distributed system (Hadoop) was installed on the cluster, configuring one machine as the name node (master) and three machines as data nodes (slaves). The one million smart meter big data was modeled as a data cube using the On-Line Analytical Processing technique to develop SQL queries for processing on top of the file distributed system (hadoop). These queries were utilized for the smart meter big data analysis using two processing engines on top of hadoop; Spark and Hive for their performance evaluation in terms of latency, throughput, and memory usage. The outcome of this was compared with two traditional and proprietary existing energy management techniques for smart meter big data. It was found that the proposed system outperformed the traditional system 20 times but it's less good than the proprietary system. It's worth mentioning that the proprietary system is not an open source platform and used specialized cluster resources that cost more money compared to our open source commodity hardware cluster.

One major additional contribution was to develop a visualization interface using open source Hadoop based data visualization tool, called Hue. Different stakeholder queries are executed on Hue to generate graphs and tabular data corresponding to each stakeholder. These graphs can be used by home consumers and utility providers to gain useful insight into the periodic consumption trend of different home appliances, houses, community, state, and country at large. This helps home owners and utility to better manage the demand response of energy. This outcome was

compared with the existing Google home energy management system called, Google power meter which only catered to the visualization of individual appliances of the home owners but lacked a unified visualization platform for utility providers on community, state, and country level.

In addition to the residential area energy management visualization, a performance evaluation of two analysis and processing engines; Spark and Hive was also achieved. In-memory processing provides a significant performance gain and speed boost in Spark as compared to Hive. Additionally, storing the input data in-memory in subsequent query iterations greatly benefits the Spark latency in an iterative querying environment. From the experimental results, Spark proved to have performed almost 10 times better than Hive. Optimum selection of SQL engine in reality is very subjective. Even though Spark provides in-memory computation that fosters low latency, the memory constraint and limited network bandwidth should be taken into consideration. Hive can be chosen for analysis if the available hardware resources are limited. For processing small datasets on a small cluster, Hive can be a good choice to obtain stable query response without any out-of-memory exceptions. Moreover, the execution times for writing a few MBs on disk or in-memory do not have much difference so Hive can be preferred for small datasets in a cluster. In a large size cluster for processing medium to large datasets, Spark is strongly recommended over Hive.

Some limitations were also identified in this work such as the use of a structured data format for processing. Hadoop is designed for processing bulk volume of unstructured and semi structured data efficiently. Since the available dataset for one smart home was in structured CSV file format, we used the available dataset for generating, storing, and processing one million smart meters data.

For the future work, we recommend that focus can be given on efficient and optimized performance of Hive and SparkSQL by tuning in the default configuration settings and parameters in a Hadoop cluster. Additionally, it would be interesting to investigate the query plans for both processing systems to study which database algorithms are being invoked for the query execution.

References

[1] "What is Hadoop?", *IBM Big Data & Analytics Hub*, 2016. [Online]. Available: http://www.ibmbigdatahub.com/blog/what-hadoop. [Accessed: 29- Jun- 2018].

[2] H. Harshawardhan, S. Bhosale1 and D. Gadekar, "A Review Paper on Big Data and Hadoop," *International Journal of Scientific and Research Publications*, vol. 4, no. 10, pp. 1-3, 2014.

[3] P. Patil and R. Phursule, "Survey Paper on Big Data Processing and Hadoop Components," *International Journal of Science and Research (IJSR)*, vol. 3, no. 10, pp. 585-590, 2014.

[4] S. S. Refaat, H. Abu-Rub and A. Mohamed, "Big Data, Better Energy Management and Control Decisions for Distribution Systems in Smart Grid," in *2016 IEEE International Conference on Big Data (Big Data)*, Washington, DC, 2016, pp. 3115-3120.

[5] M. V. Moreno et al., "Applicability of Big Data Techniques to Smart Cities Deployments," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 800-809, April 2017.

[6] Zhou, Kaile, Fu, Chao, Yang and Shanlin, "Big Data Driven Smart Energy Management: From Big Data to Big Insights," *Renewable and Sustainable Energy Reviews*, vol. 56, pp. 215-225, 2016.

[7] I. Mavridis and H. Karatza, "Performance Evaluation of Cloud-Based Log File Analysis with Apache Hadoop and Apache Spark," *Journal of Systems and Software*, vol. 56, pp. 133-151, March 2017.

[8] C. Wang, C. Tsai, C. Fan, S. Yuan, "A Hadoop based Weblog Analysis System," in *7th International Conference on Ubi-Media Computing and Workshops (U-MEDIA 2014)*, Ulaanbaatar, Mongolia, 2014, pp. 72–77.

[9] S. Narkhede and T. Baraskar, "HMR Log Analyzer: Analyze Web Application Logs over Hadoop MapReduce." *International Journal of UbiComp (IJU)*, vol. 4, pp. 41–51, July 2013.

[11] D. Mysore, S. Khupat and S. Jain, "Introduction to Big Data Classification and Architecture", *IBM*, 2013. [Online]. Available: https://www.ibm.com/developerworks/library/bd-archpatterns1/. [Accessed: 29- Jun- 2018].

[10] H. Yu and D. Wang, " Mass Log Data Processing and Mining Based on Hadoop and Cloud Computing," in *7th International Conference on Computer Science and Education (ICCSE 2012)*, 2012, pp. 14–17.

[12] V. Fanibhare and V. Dahake, "SmartGrids: MapReduce framework using Hadoop," in *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, Noida, 2016, pp. 400-405.

[13] M. Vaidya and S. Deshpande, "Distributed Data Management in Energy Sector using Hadoop," in *2015 IEEE Bombay Section Symposium (IBSS)*, Mumbai, 2015, pp. 1-6.

[14] S. Prasad and S. B. Avinash, "Smart Meter Data Analytics using OpenTSDB and Hadoop," in *2013 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia)*, Bangalore, 2013, pp. 1-6.

[15] R. T. Kaushik, M. Bhandarkar and K. Nahrstedt, "Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, Indianapolis, IN, 2010, pp. 274-287.

[16] 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*, 2013, pp. 1-4.

[17] D. Alahakoon and X. Yu, "Smart Electricity Meter Data Intelligence for Future Energy Systems: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 1, pp. 425-436, Feb. 2016.

[18] Xiufeng Liu, Per Sieverts Nielsen, "A Hybrid ICT-Solution for Smart Meter Data Analytics," *Journal of Energy*, vol. 115, pp. 1710-1722, 2016.

[19] Y. Shao, C. Li, W. Dong and Y. Liu, "Energy-Aware Dynamic Resource Allocation on Hadoop YARN Cluster," in *2016 IEEE 18th International Conference on High Performance Computing and Communications*, Sydney, NSW, 2016, pp. 364-371.

[20] J. Choi, M. Kim and J. Yoon, "Implementation of the Big Data Management System for Demand Side Energy Management," in *2015 IEEE International Conference on Computer and Information Technology*; *Ubiquitous Computing and Communications*; *Dependable, Autonomic and Secure Computing*; *Pervasive Intelligence and Computing*, Liverpool, 2015, pp. 1515-1520.

[21] T. Ivanov, R. Niemann, S. Izberovic, M. Rosselli, K. Tolle and R. V. Zicari, "Performance Evaluation of Enterprise Big Data Platforms with HiBench," in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, 2015, pp. 120-127.

[22] I. Alzuru, K. Long, B. Gowda, D. Zimmerman and T. Li, "Hadoop Characterization," in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, 2015, pp. 96-103.

[23] Y. Samadi, M. Zbakh and C. Tadonki, "Comparative Study between Hadoop and Spark based on Hibench Benchmarks," in *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, Marrakech, 2016, pp. 267-275.

[24] Y. Li, L. Wang, L. Ji and C. Liao, "A Data Warehouse Architecture Supporting Energy Management of Intelligent Electricity System", in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*, 2013.

[25] José Cavalheiro and Paulo Carreira, "A Multidimensional Data Model Design for Building Energy Management," *Advanced Engineering Informatics*, vol. 30, pp. 619-632, 2016.

[26] M. Mehdi, R. Sahay, W. Qu, S. Debloch and E. Curry, "Real-time Generation of Linked Sensor Data and Multidimensional Data Cubes for Smart Environments", *INSIGHT Centre for Data Analytics*, National University of Ireland, Galway Ireland, 2009.

[27] T. Weibel, "Smart - UMass Trace Repository," *Traces.cs.umass.edu*, 2017. [Online]. Available at: http://traces.cs.umass.edu/index.php/Smart/Smart

[28] R. Dalinina, "Introduction to Forecasting with ARIMA in R", *Datascience.com*, 2018. [Online]. Available: https://www.datascience.com/blog/introduction-to-forecasting-with-arima-in-r-learn-data-science-tutorials. [Accessed: 29- Jun- 2018].

[29] P. Müller, "Markov Chain Monte Carlo Methods," in *International Encyclopedia of the Social & Behavioral Sciences*, edited by Neil J. Smelser and Paul B. Baltes, Pergamon, Oxford, 2001, pp. 9236-9240, ISBN 9780080430768, https://doi.org/10.1016/B0-08-043076-7/00469-1.

[30] "United Arab Emirates", *En.wikipedia.org*, 2018. [Online]. Available: https://en.wikipedia.org/wiki/United_Arab_Emirates. [Accessed: 2- Jul- 2018].

[31] "2 The Multidimensional Data Model", *Docs.oracle.com*, 2017. [Online]. Available: https://docs.oracle.com/cd/B13789_01/olap.101/b10333/multimodel.htm. [Accessed: 29- Jun- 2018].

[32] J. Kestelyn, "It's All About You: New Community Forums for Cloudera Customers and Users - Cloudera Engineering Blog", *Cloudera Engineering Blog*, 2018. [Online]. Available: http://blog.cloudera.com/blog/2013/07/its-all-about-you-new-community-forums-for-cloudera-customers-and-users/.

[33] K. Singh and R. Kaur, "Hadoop: Addressing Challenges of Big Data," in *2014 IEEE International Advance Computing Conference (IACC)*, Gurgaon, 2014, pp. 686-687.

[34] D. Borthakur, "HDFS Architecture Guide", *Hadoop*, 2013. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Accessed: 29- Jun- 2018].

[35] K. Singh and R. Kaur, "Hadoop: Addressing Challenges of Big Data," in *2014 IEEE International Advance Computing Conference (IACC)*, Gurgaon, 2014, pp. 686-689.

[36] P. Patil and R. Phursule, "Survey Paper on Big Data Processing and Hadoop Components", *International Journal of Science and Research (IJSR)*, vol. 3, no. 10, pp. 585-590, 2014.

[37] S. Humbetov, "Data-intensive Computing with Map-Reduce and Hadoop," *in 2012 6th International Conference on Application of Information and Communication Technologies (AICT)*, Tbilisi, 2012, pp. 1-5.

[38] Rui Xue, "SQL Engines for Big Data Analytics: SQL on Hadoop", Master's thesis, Aalto University, Espoo, 2015.

[39] A. Thusoo *et al.*, "Hive - A Petabyte Scale Data Warehouse Using Hadoop," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, Long Beach, CA, 2010, pp. 996-1005.

[40] N. Pushpalatha and P. Sudheer, "Data Processing in Big Data by Using Hive Interface," *International Journal of Advanced Research in Computer Science and Advanced Studies*, vol. 3, no. 4, pp. 272-277, 2015.

[41] "Apache Spark - Unified Analytics Engine for Big Data", *Spark.apache.org*, 2018. [Online]. Available: https://spark.apache.org/. [Accessed: 29- Jun- 2018].

[42] M. Zaharia, M. Chowdhury, T. Das, A. Dave, M. Mccauley, M. J.Franklin, S. Shankar and I. Stoica, "Fast and Interactive Analytics over Hadoop Networked Systems Data with Spark", *usenix.org*, 2018. [Online]. Available: https://www.usenix.org/system/files/login/articles/zaharia.pdf. [Accessed: 29- Jun- 2018].

[43] "Oozie - Apache Oozie Workflow Scheduler for Hadoop", *Oozie.apache.org*, 2018. [Online]. Available: http://oozie.apache.org/.[Accessed: 29- Jun- 2018].

[44] "Get Started with Hue | 5.9.x | Cloudera Documentation", *Cloudera.com*, 2018. [Online]. Available: https://www.cloudera.com/documentation/enterprise/5-9-x/topics/hue.html. [Accessed: 29- Jun- 2018].

[45] "Managing Big Data for Smart Grids and Smart Meters", *IBM.com*, 2012. [Online]. Available at: http://www-935.ibm.com /services/multimedia/ Managing_big_data_for_smart_grids_and_smart_meters.pdf. [Accessed: 29- Jun- 2018].

Appendix A: Experimentation Results

This appendix provides average latency, throughput, and memory usage values for experiment 1 and 2.

A.1. Latency for query wise execution

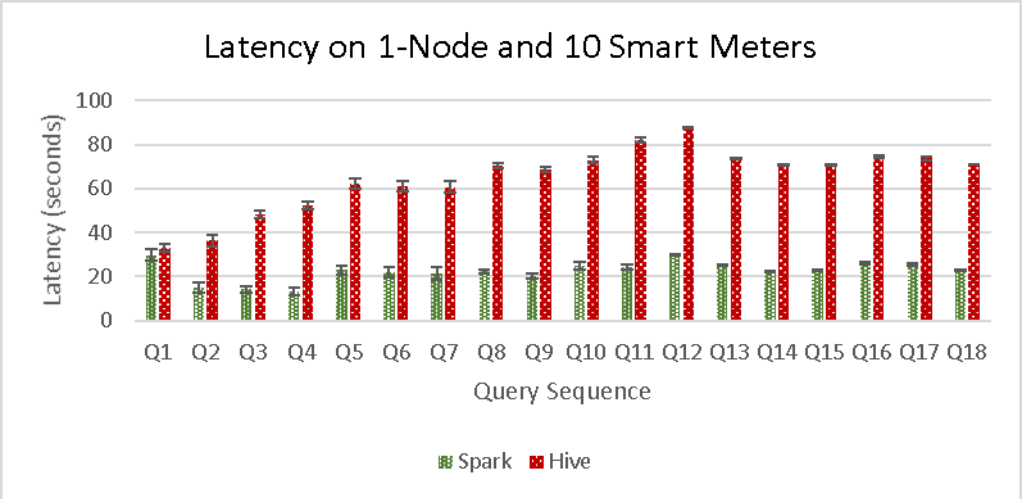

Figure A.1.1: Mean Latency for Spark and Hive across 1 Node, 10 smart meter files per query

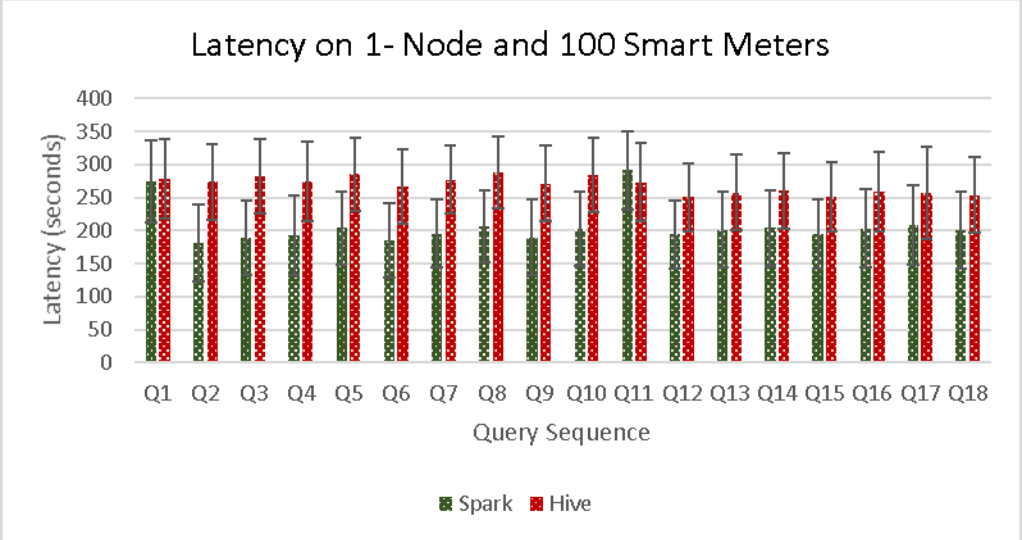

Figure A.1.2: Mean Latency for Spark and Hive across 1 Node, 100 smart meter files per query

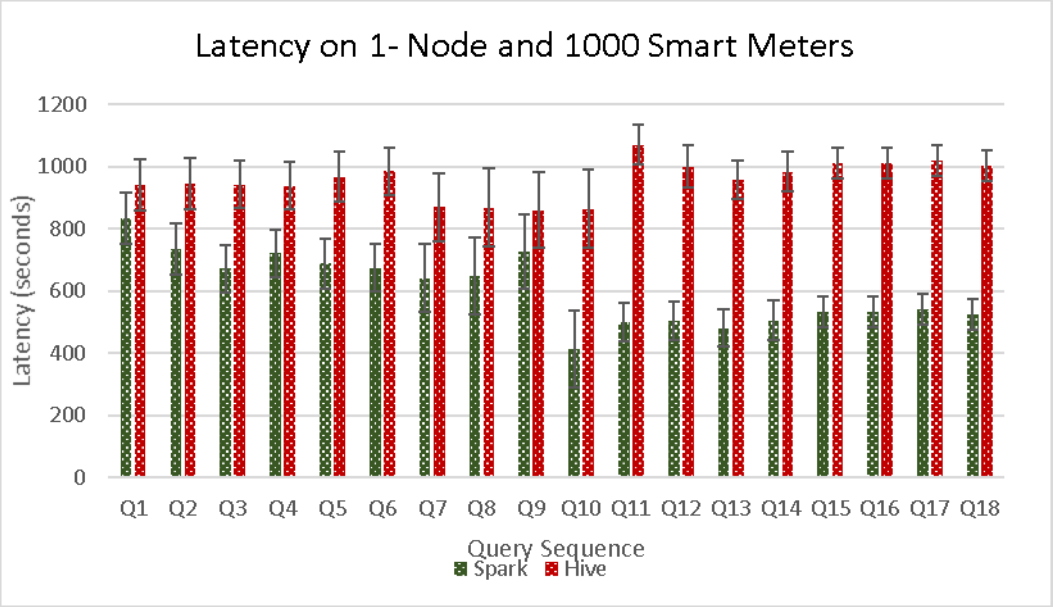

Figure A.1.3: Mean Latency for Spark and Hive across 1 Node, 1000 smart meter files per query

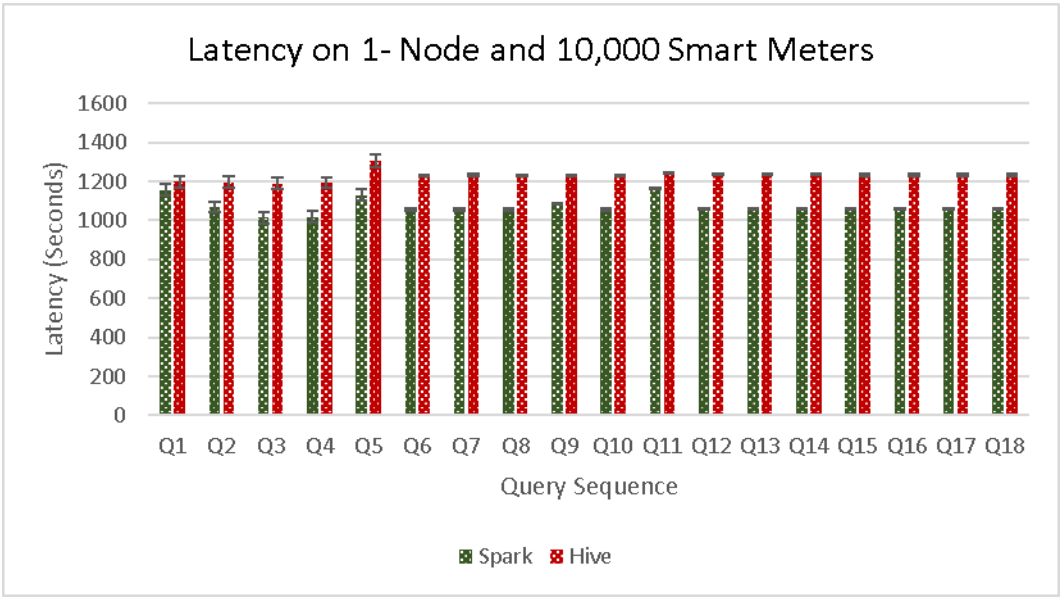

Figure A.1.4: Mean Latency for Spark and Hive across 1 Node, 10000 smart meter files per query

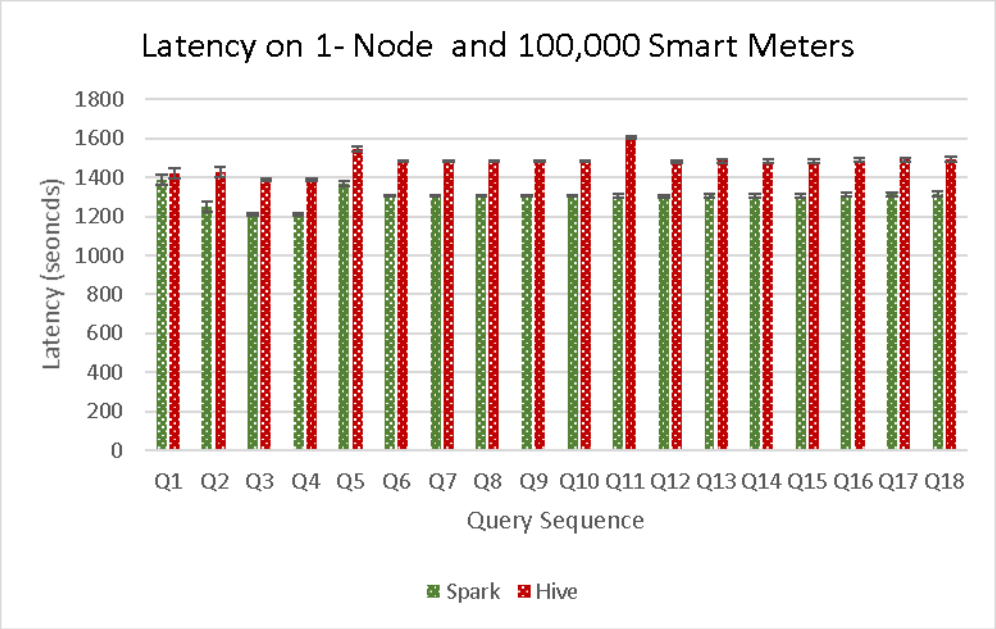

Figure A.1.5: Mean Latency for Spark and Hive across 1 Node, 100000 smart meter files per query

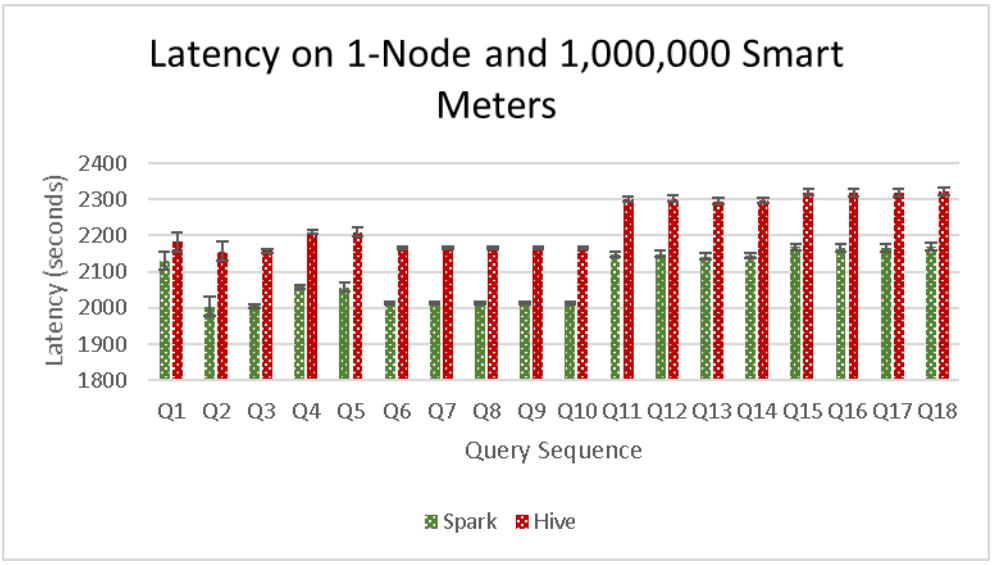

Figure A.1.6: Mean Latency for Spark and Hive across 1 Node, 1,000,000 smart meter files per query

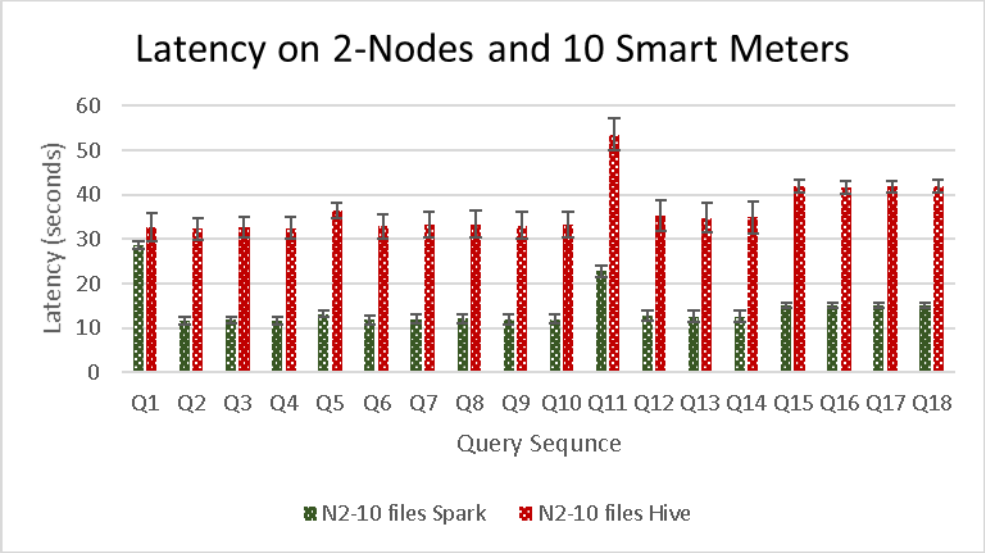

Figure A.1.7: Mean Latency for Spark and Hive across 2 Nodes, 10 smart meter files per query

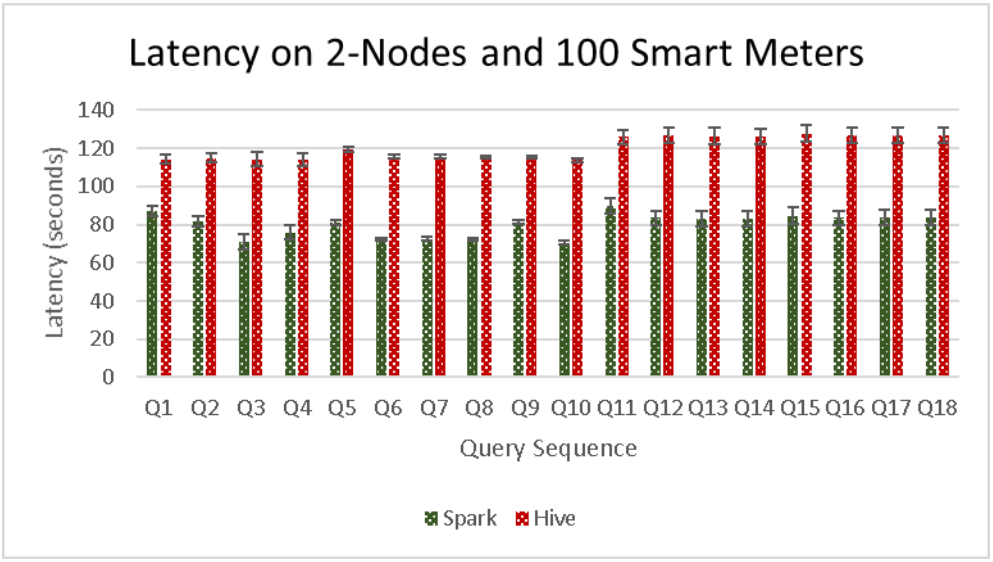

Figure A.1.8: Mean Latency for Spark and Hive across 2 Nodes, 100 smart meter files per query

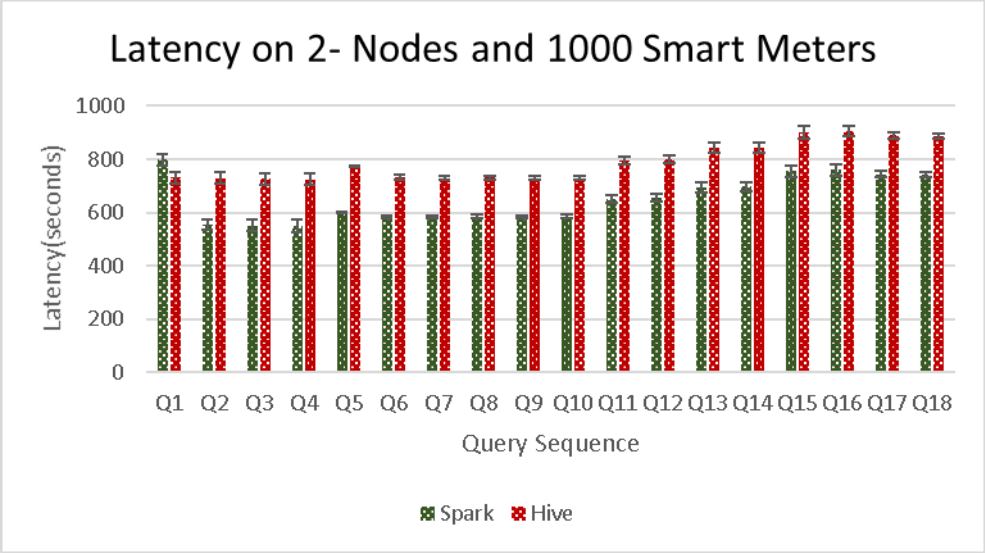

Figure A.1.9: Mean Latency for Spark and Hive across 2 Nodes, 1000 smart meter files per query

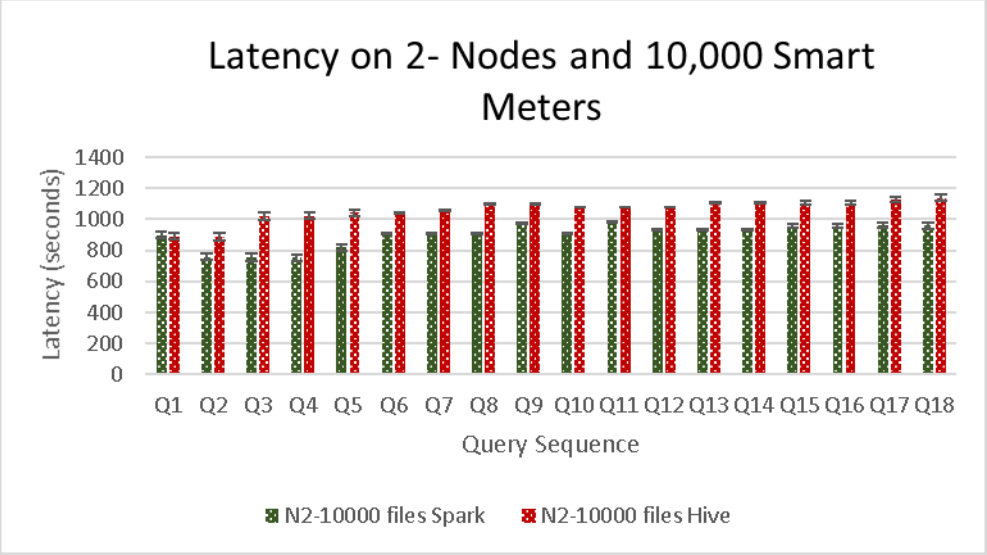

Figure A.1.10: Mean Latency for Spark and Hive across 2 Nodes, 10000 smart meter files per query

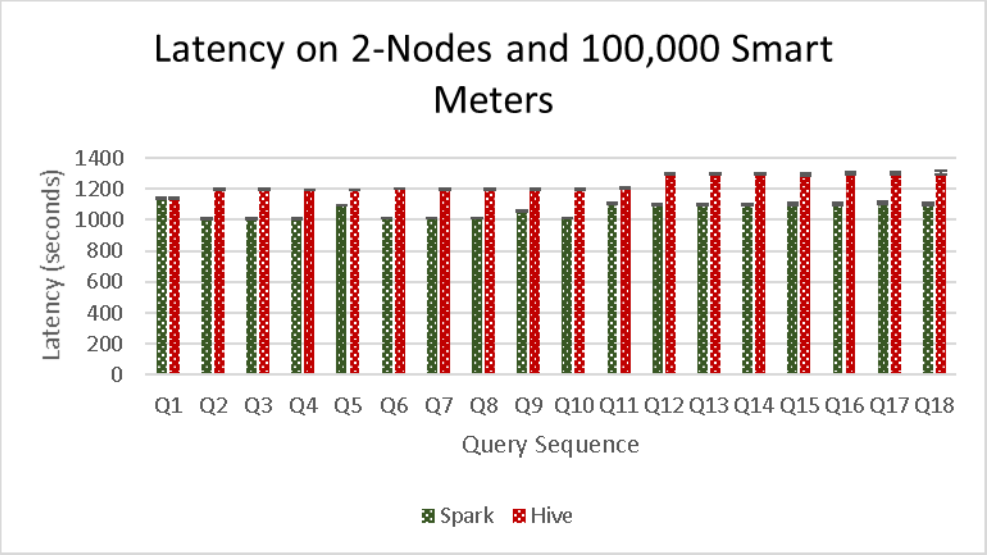

Figure A.1.11: Mean Latency for Spark and Hive across 2 Nodes, 100000 smart meter files per query

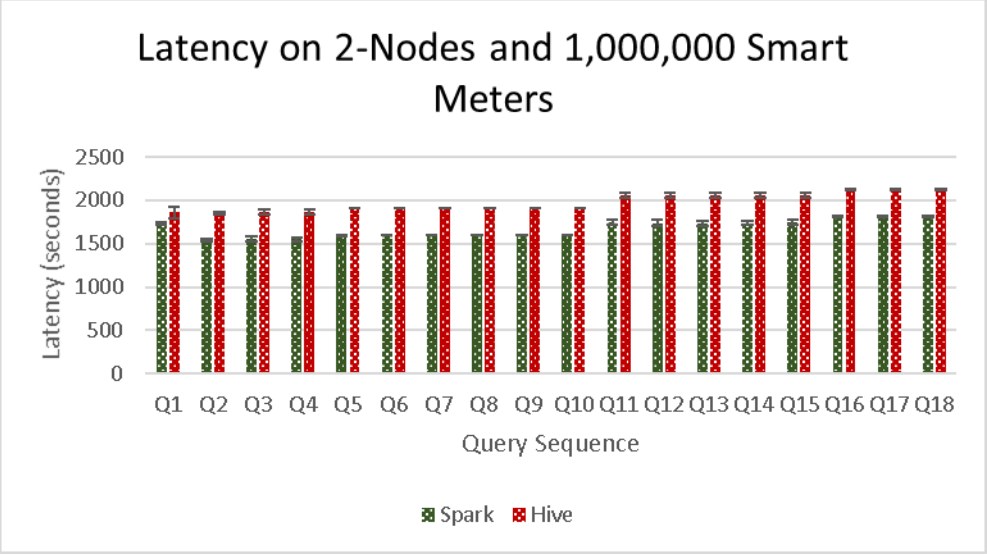

Figure A.1.12: Mean Latency for Spark and Hive across 2 Nodes, 1,000,000 smart meter files per query

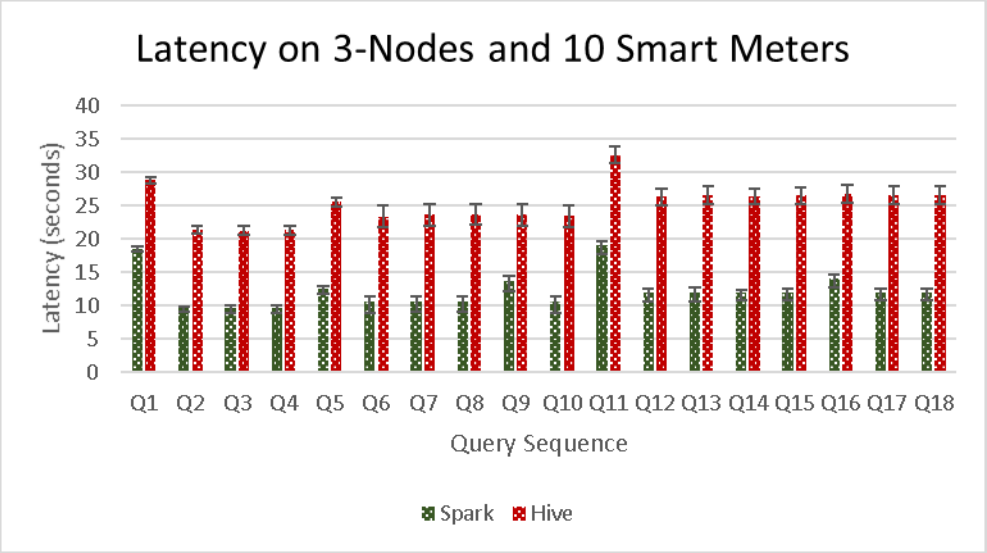

Figure A.1.13: Mean Latency for Spark and Hive across 3 Nodes, 10 smart meter files per query

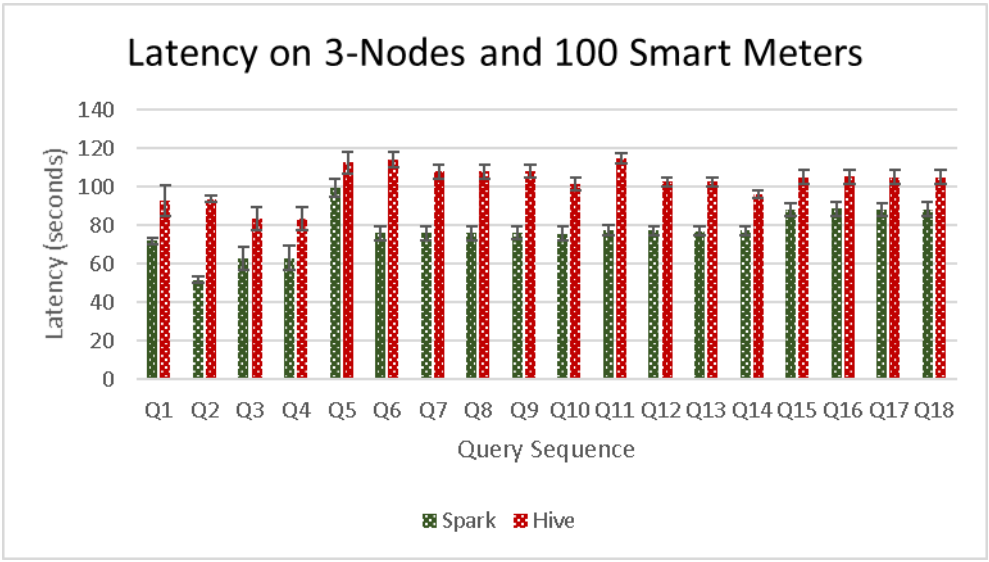

Figure A.1.14: Mean Latency for Spark and Hive across 3 Nodes, 100 smart meter files per query

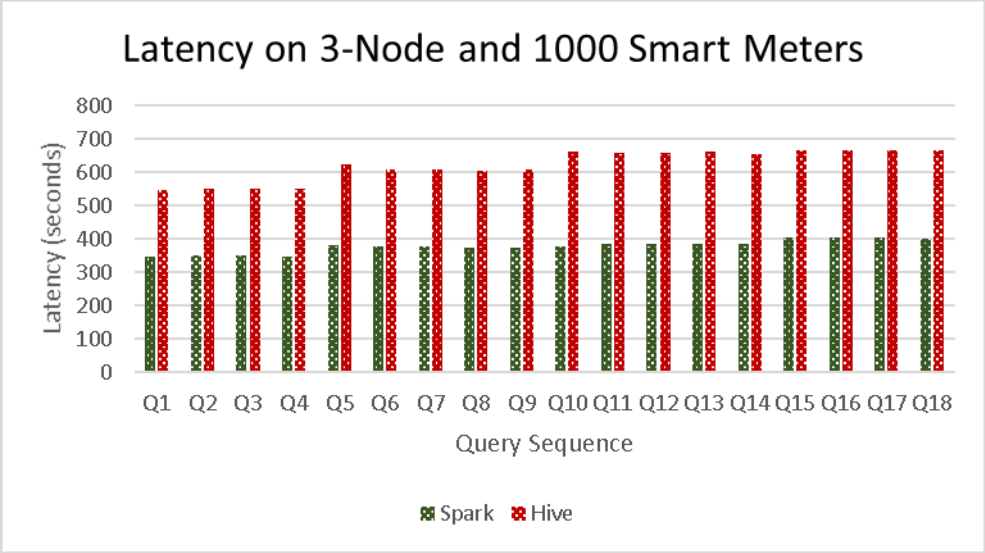

Figure A.1.15: Mean Latency for Spark and Hive across 3 Nodes, 1000 smart meter files per query

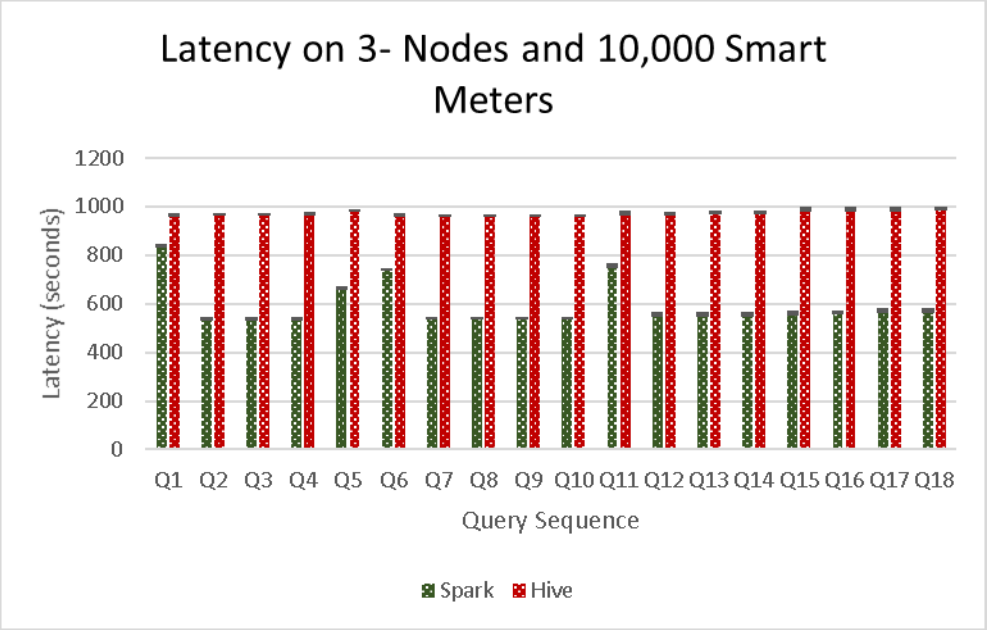

Figure A.1.16: Mean Latency for Spark and Hive across 3 Nodes, 10000 smart meter files per query

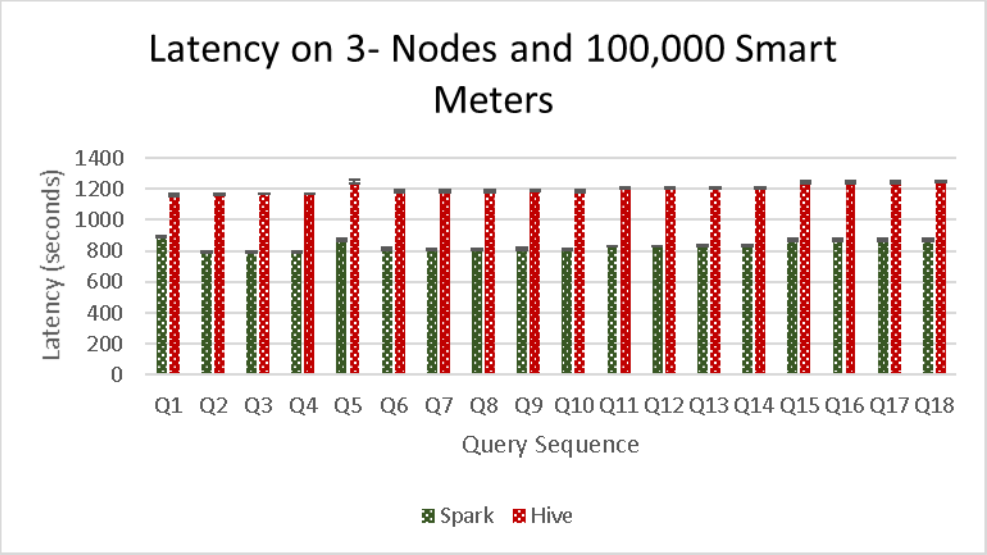

Figure A.1.17: Mean Latency for Spark and Hive across 3 Nodes, 100000 smart meter files per query

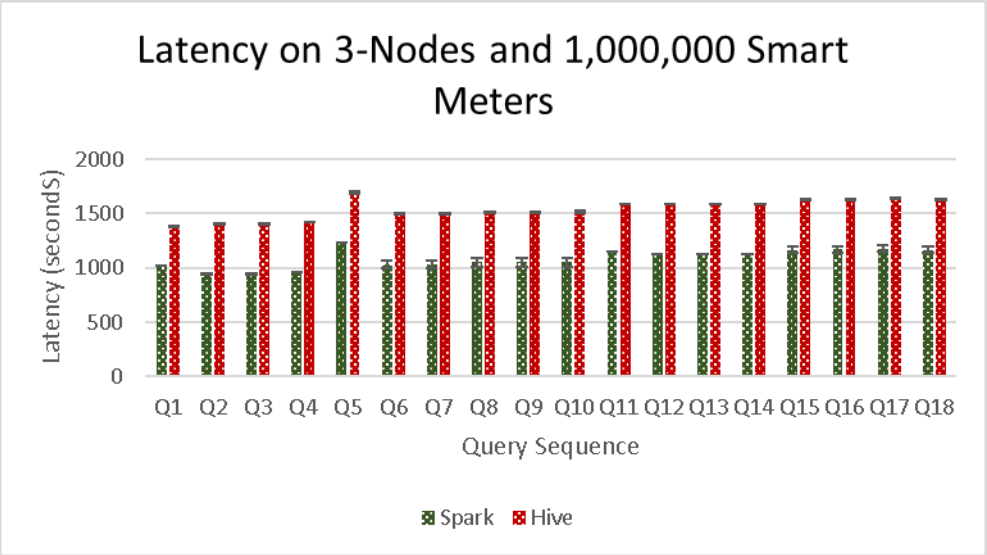

Figure A.1.18: Mean Latency for Spark and Hive across 3 Nodes, 1,000,000 smart meter files per query

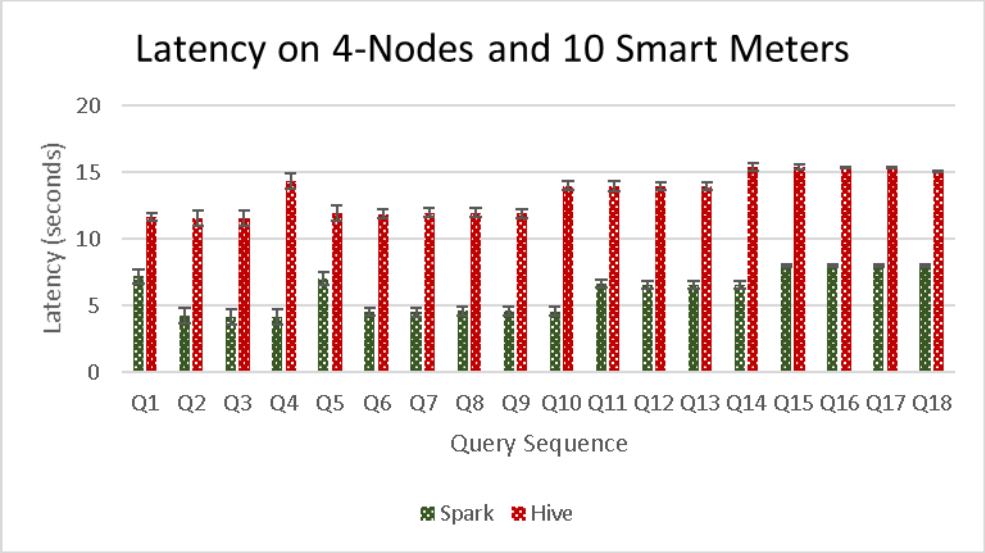

Figure A.1.19: Mean Latency for Spark and Hive across 4 Nodes, 10 smart meter files per query

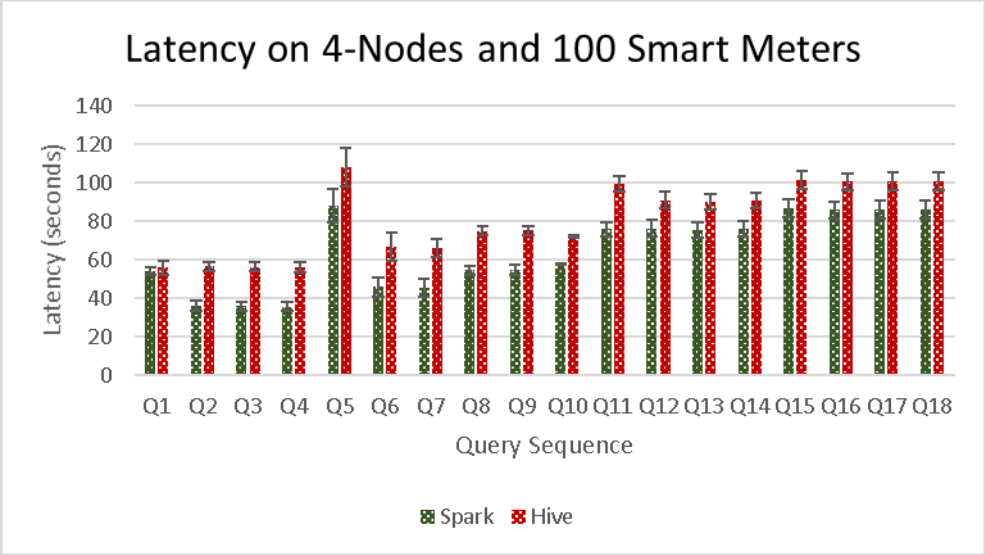

Figure A.1.20: Mean Latency for Spark and Hive across 4 Nodes, 100 smart meter files per query

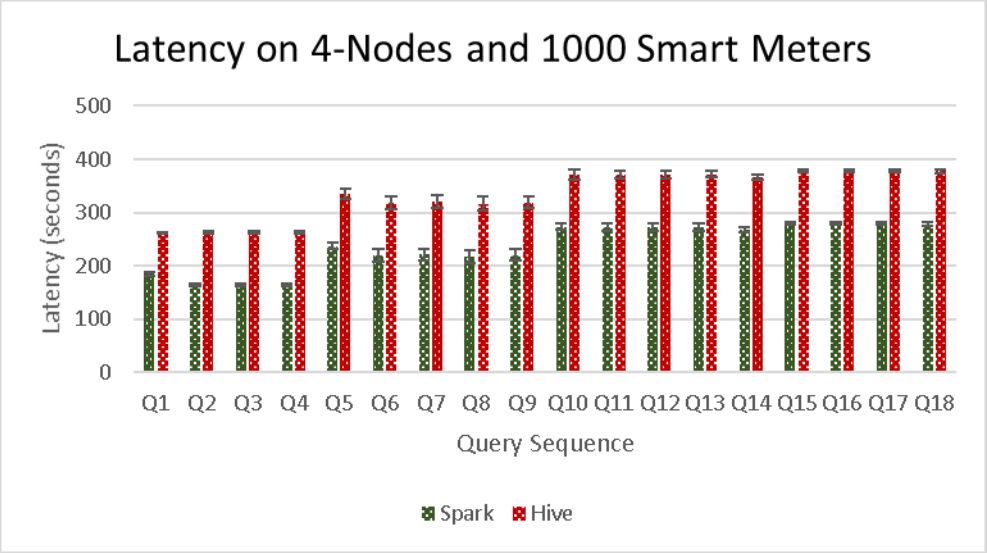

Figure A.1.21: Mean Latency for Spark and Hive across 4 Nodes, 1000 smart meter files per query

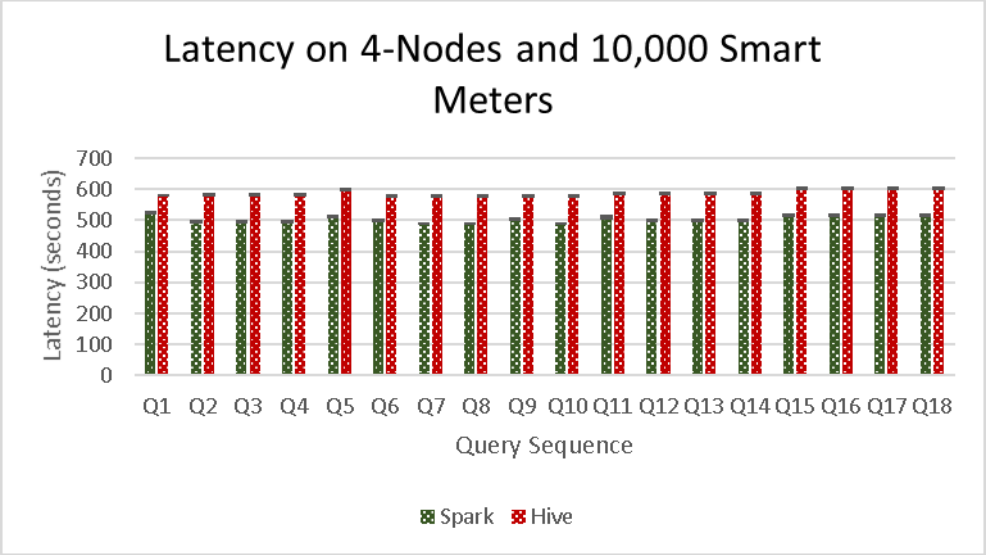

Figure A.1.22: Mean Latency for Spark and Hive across 4 Nodes, 10000 smart meter files per query

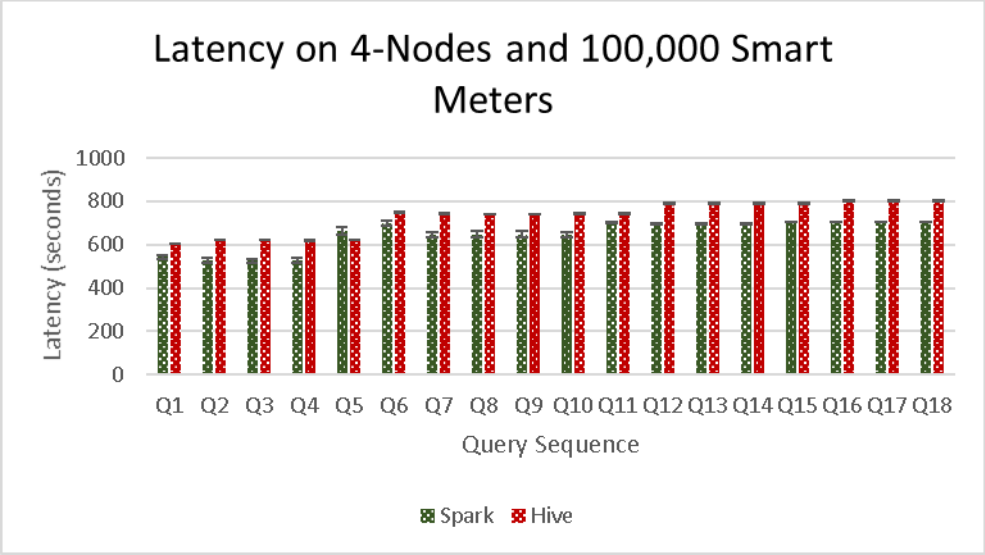

Figure A.1.23: Mean Latency for Spark and Hive across 4 Nodes, 100000 smart meter files per query

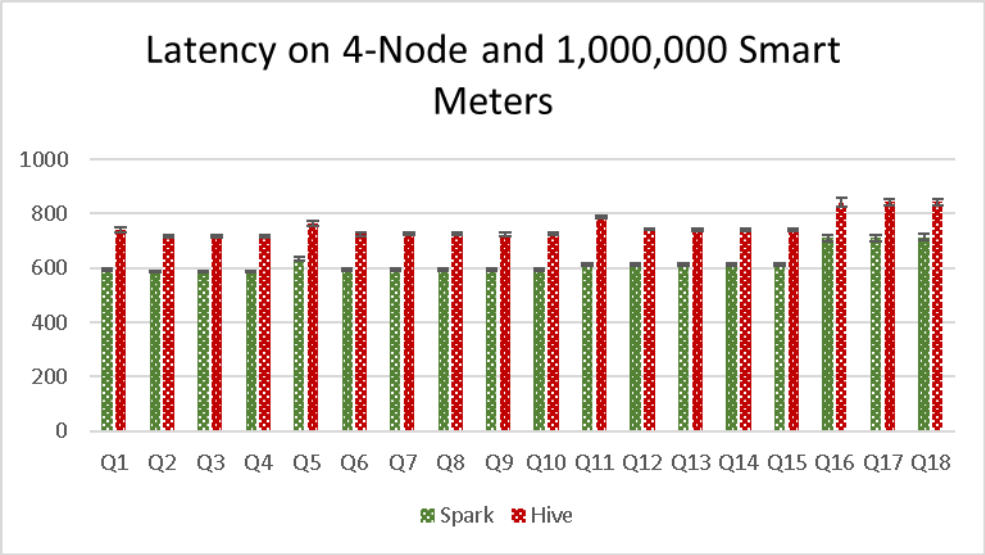

Figure A.1.24: Mean Latency for Spark and Hive across 4 Nodes, 100000 smart meter files per query

A.2. Throughput for query wise execution

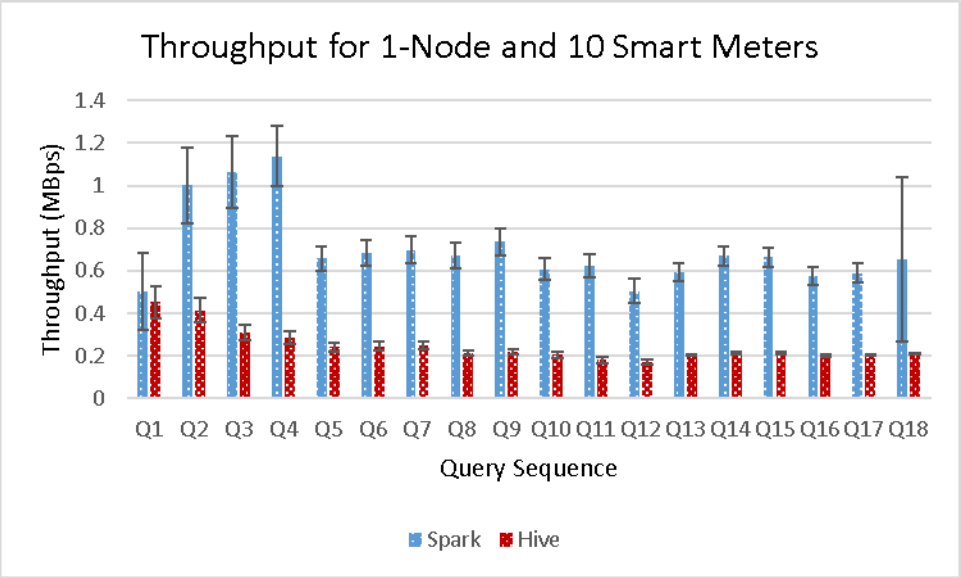

Figure A.2.1: Mean Throughput for Spark and Hive across 1 Node, 10 smart meter files per query

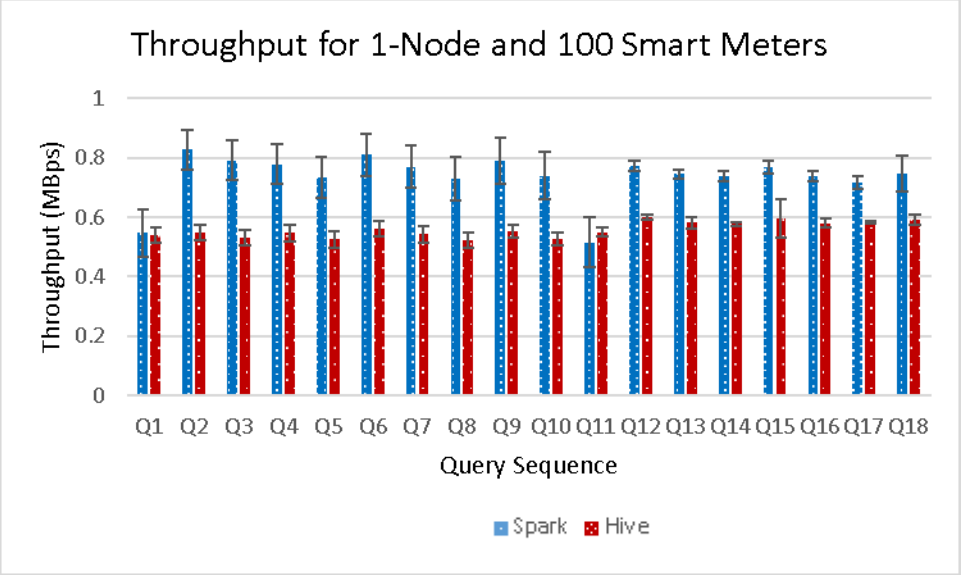

Figure A.2.2: Mean Throughput for Spark and Hive across 1 Nodes, 100 smart meter files per query

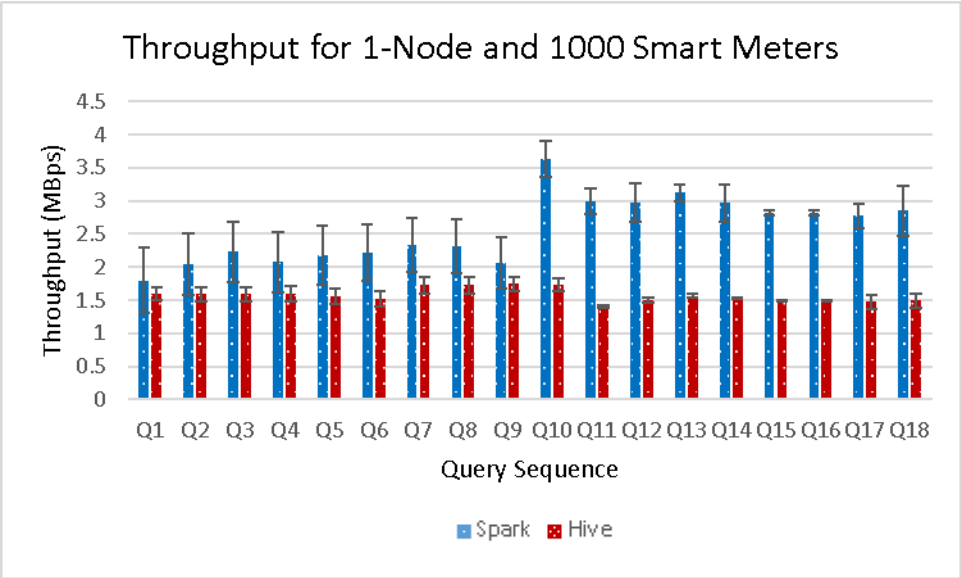

Figure A.2.3: Mean Throughput for Spark and Hive across 1 Nodes, 1000 smart meter files per query

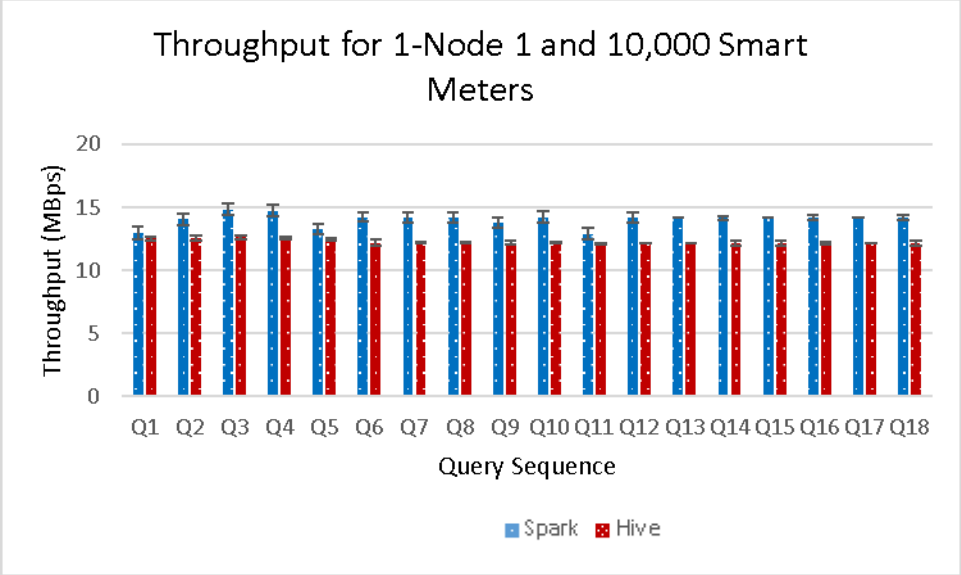

Figure A.2.4: Mean Throughput for Spark and Hive across 1 Nodes, 10000 smart meter files per query

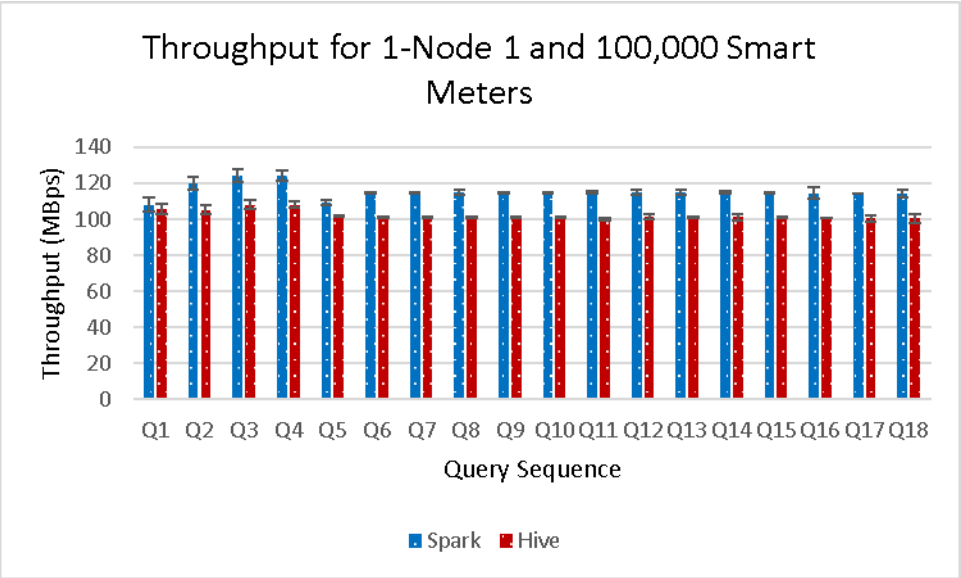

Figure A.2.5: Mean Throughput for Spark and Hive across 1 Node, 100000 smart meter files per query

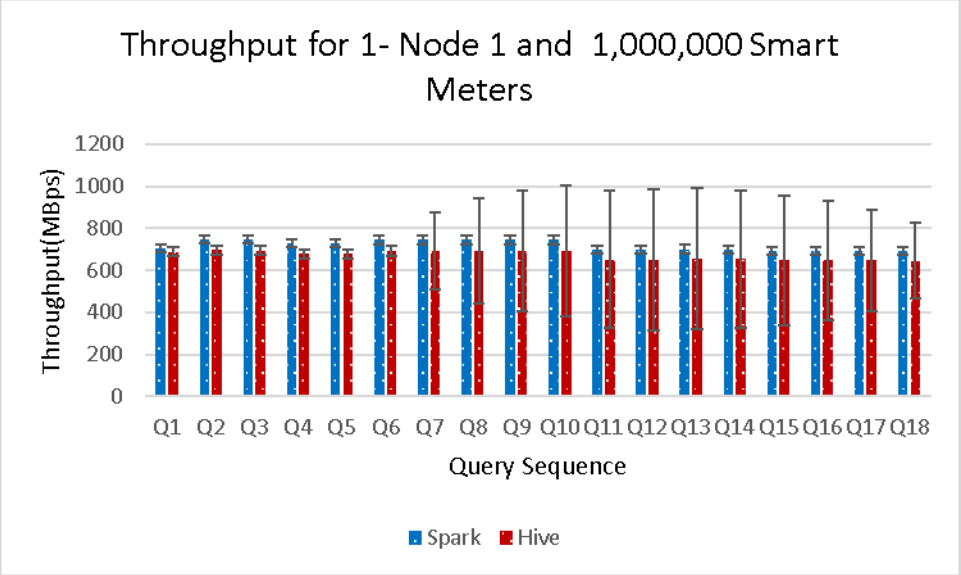

Figure A.2.6: Mean Throughput for Spark and Hive across 1 Node, 1,000,000 smart meter files per query

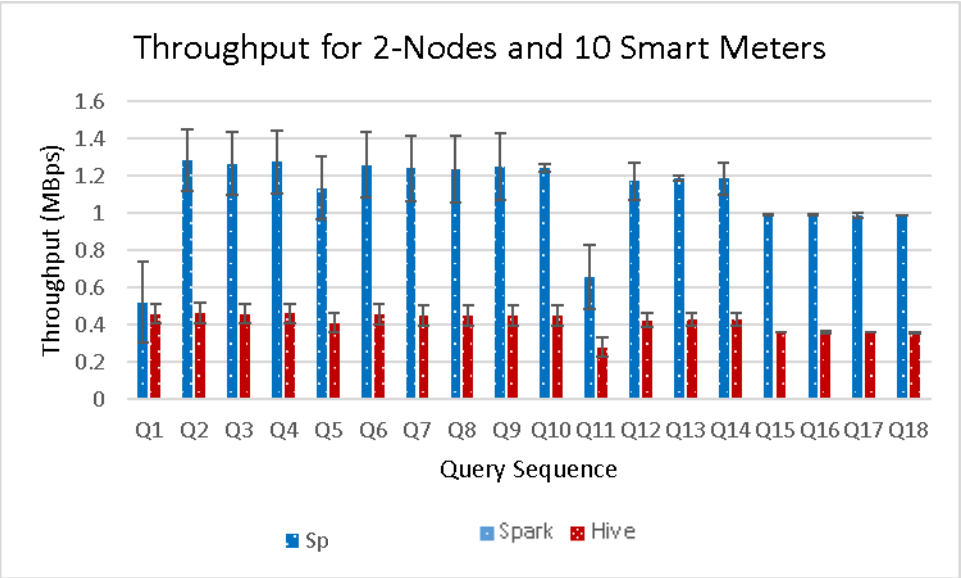

Figure A.2.7: Mean Throughput for Spark and Hive across 2 Nodes, 10 smart meter files per query

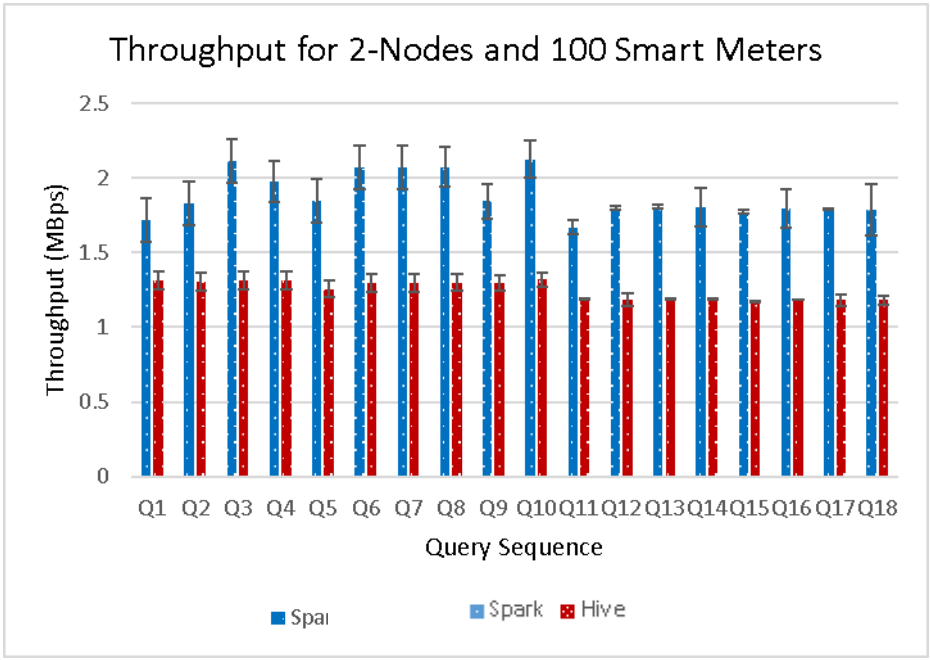

Figure A.2.8: Mean Throughput for Spark and Hive across 2 Nodes, 100 smart meter files per query

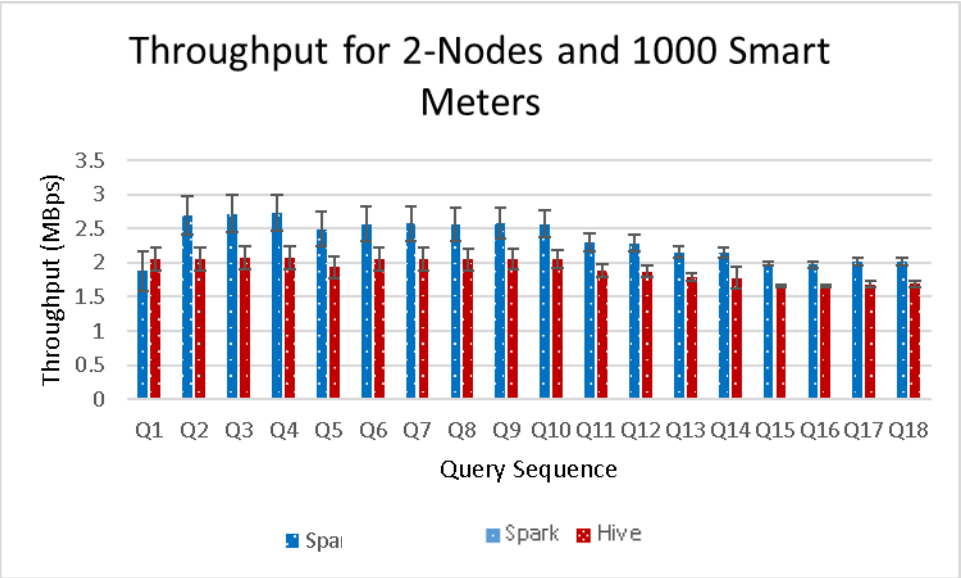

Figure A.2.9: Mean Throughput for Spark and Hive across 2 Nodes, 1000 smart meter files per query

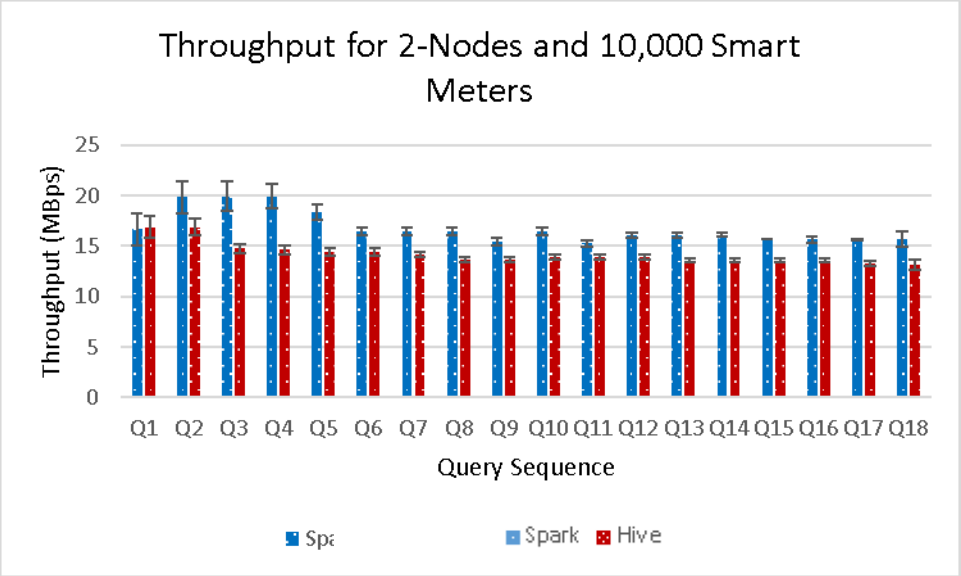

Figure A.2.10: Mean Throughput for Spark and Hive across 2 Nodes, 10000 smart meter files per query

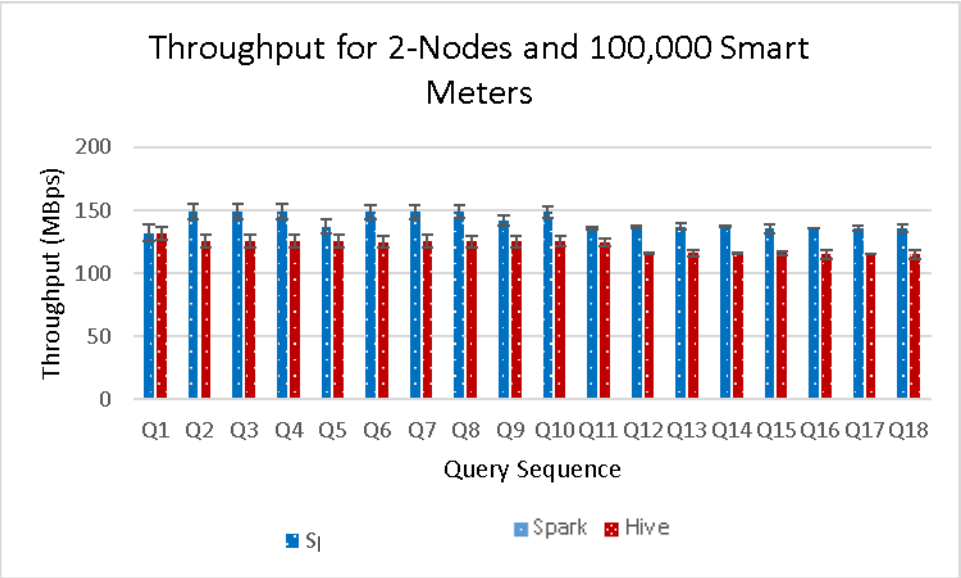

Figure A.2.11: Mean Throughput for Spark and Hive across 2 Nodes, 100000 smart meter files per query

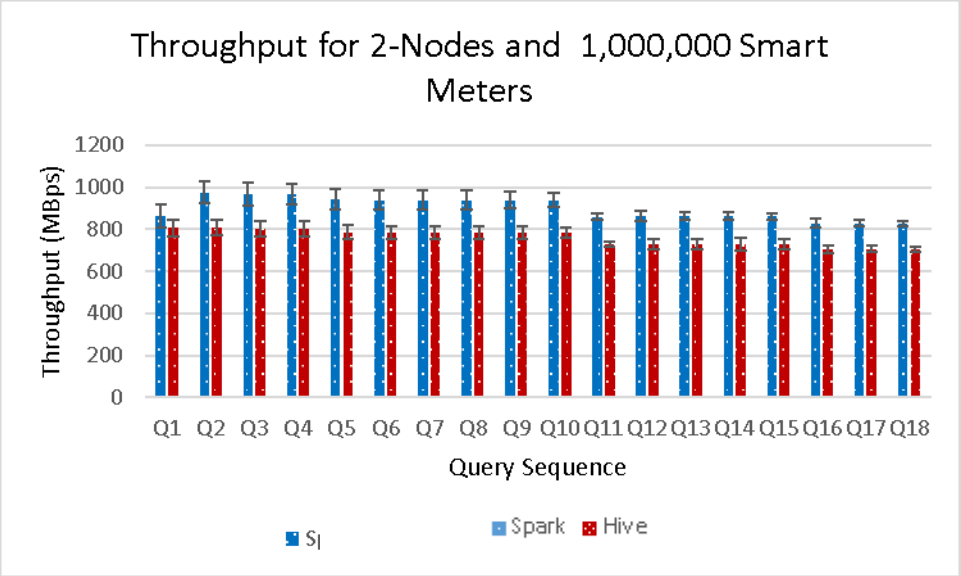

Figure A.2.12: Mean Throughput for Spark and Hive across 2 Nodes, 1,000,000 smart meter files per query

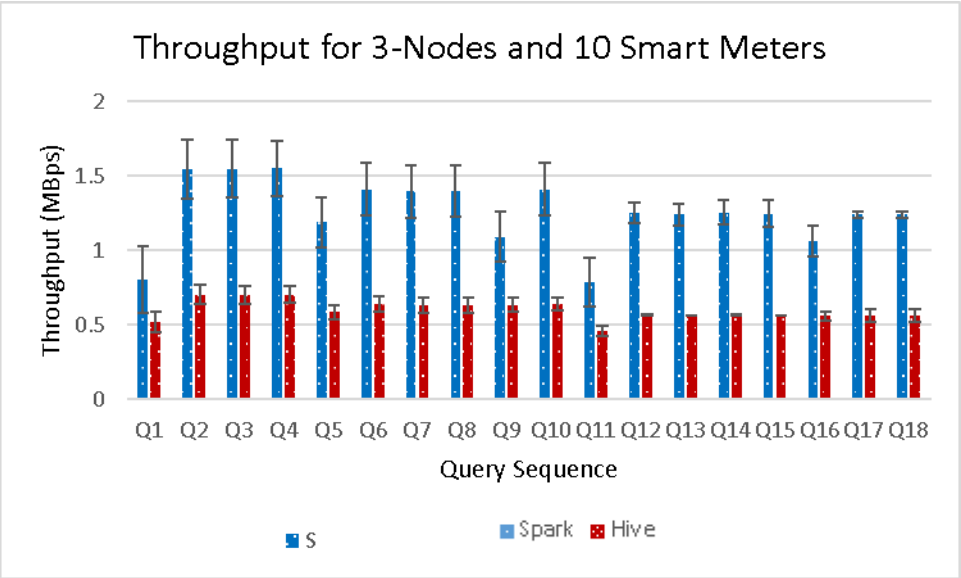

Figure A.2.13: Mean Throughput for Spark and Hive across 3 Nodes, 10 smart meter files per query

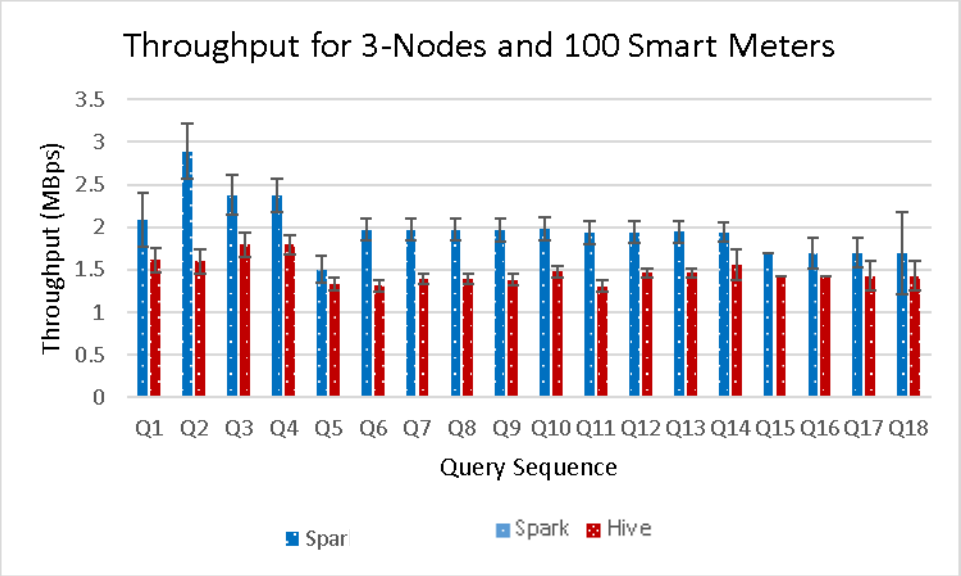

Figure A.2.14: Mean Throughput for Spark and Hive across 3 Nodes, 100 smart meter files per query

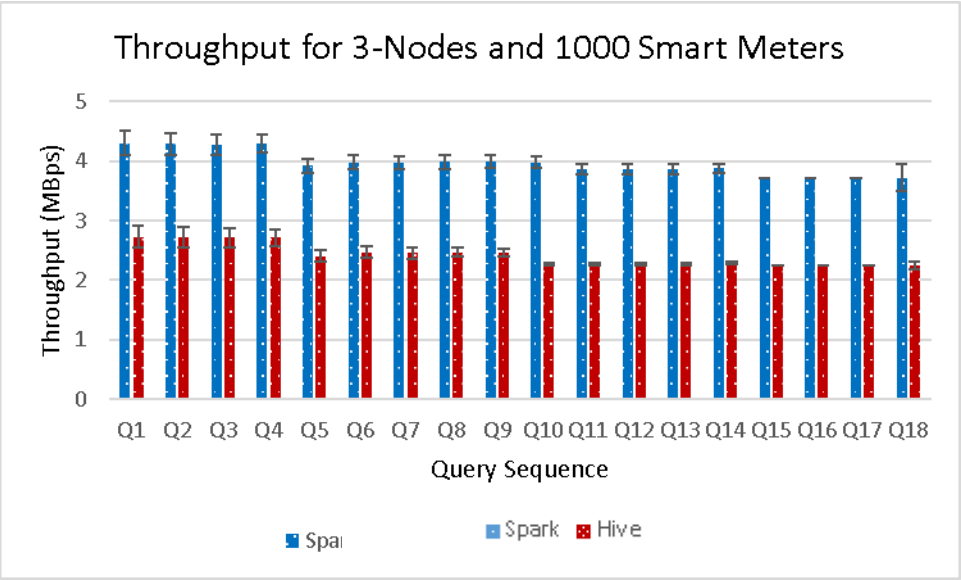

Figure A.2.15: Mean Throughput for Spark and Hive across 3 Nodes, 1000 smart meter files per query

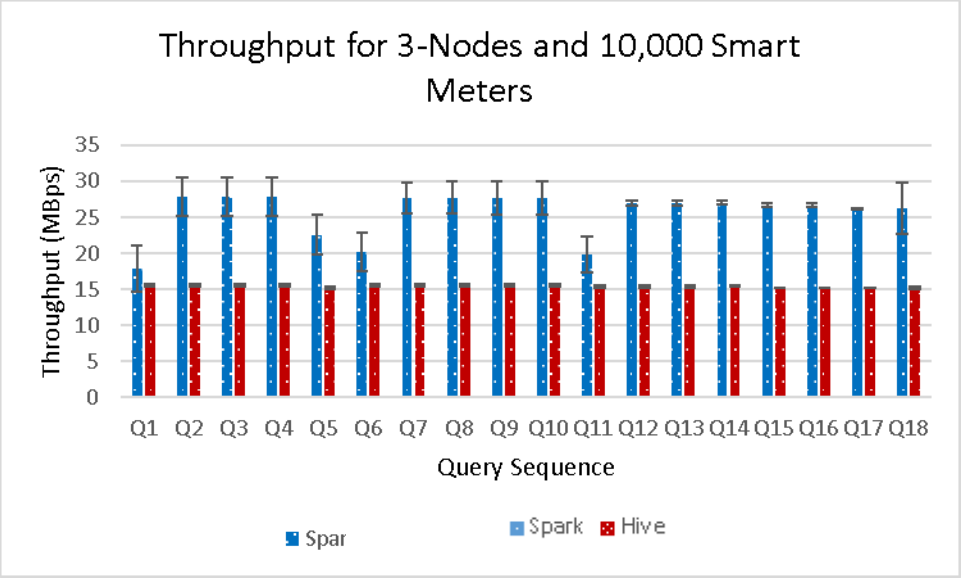

Figure A.2.16: Mean Throughput for Spark and Hive across 3 Nodes, 10000 smart meter files per query

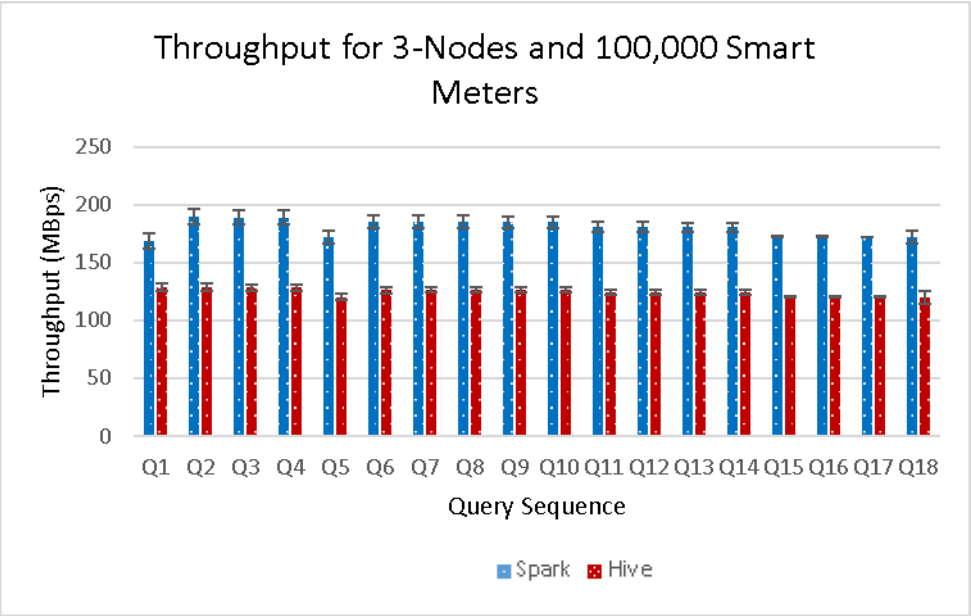

Figure A.2.17: Mean Throughput for Spark and Hive across 3 Nodes, 100000 smart meter files per query

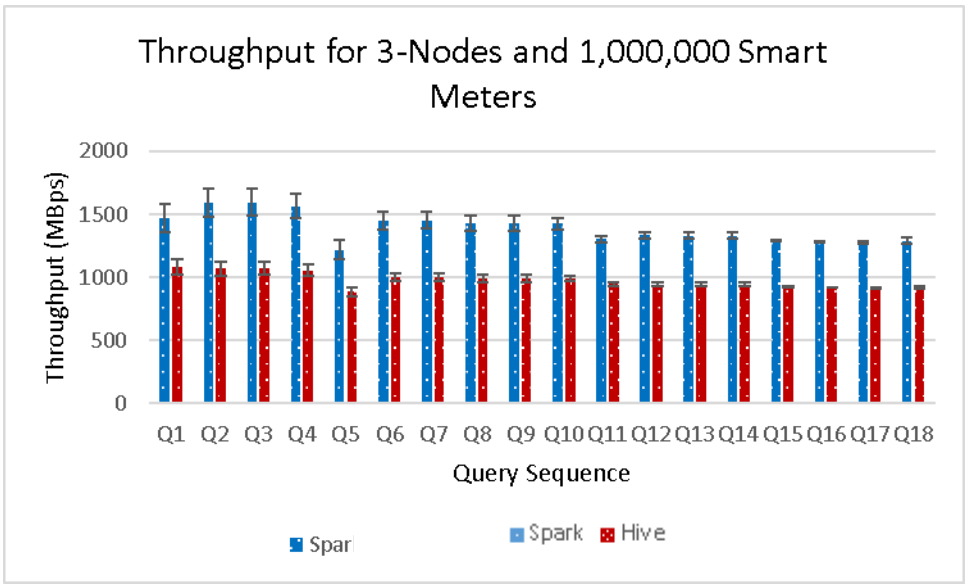

Figure A.2.18: Mean Throughput for Spark and Hive across 3 Nodes, 1,000,000 smart meter files per query

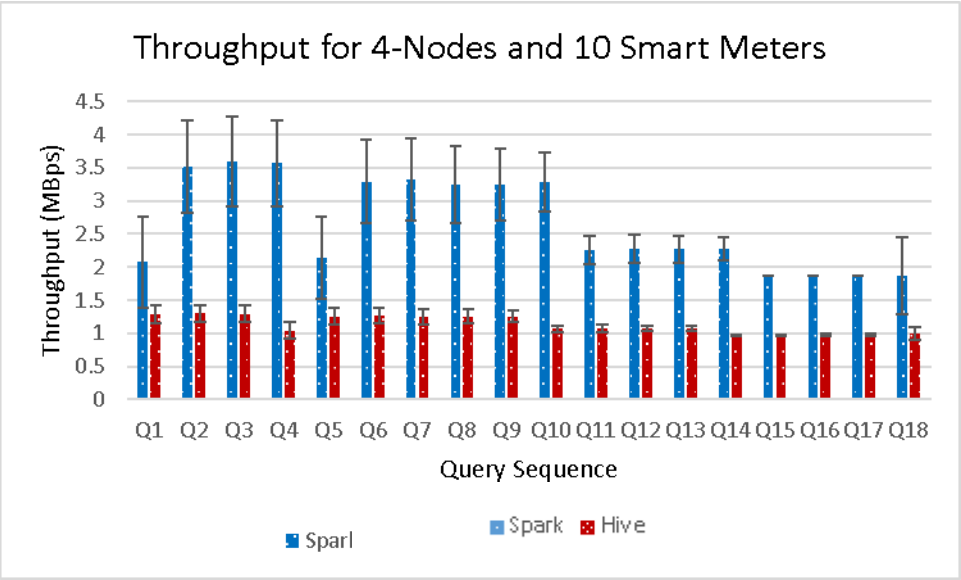

Figure A.2.19: Mean Throughput for Spark and Hive across 4 Nodes, 10 smart meter files per query

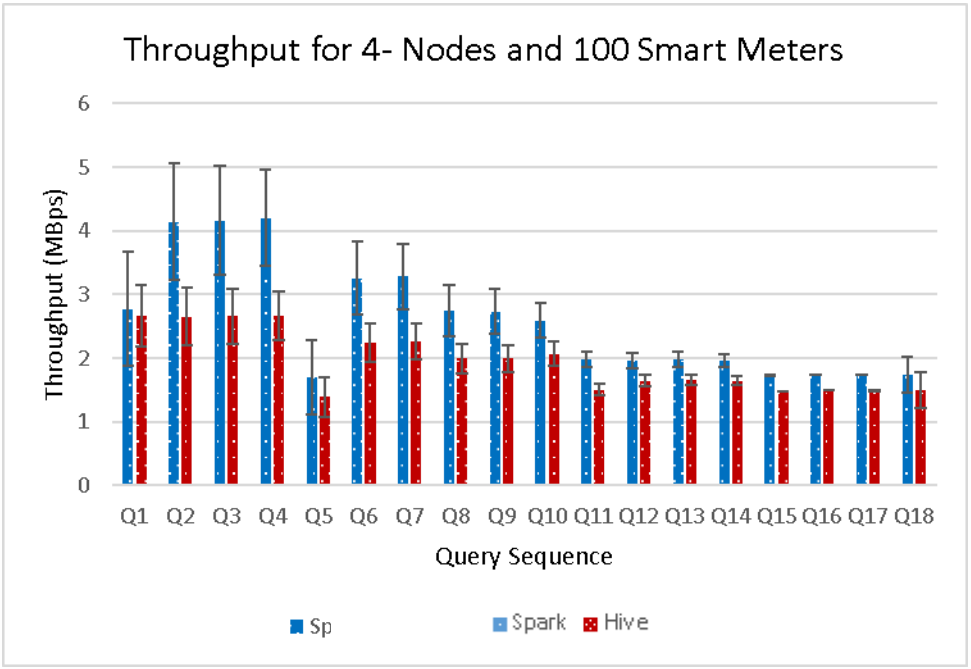

Figure A.2.20: Mean Throughput for Spark and Hive across 4 Nodes, 100 smart meter files per query

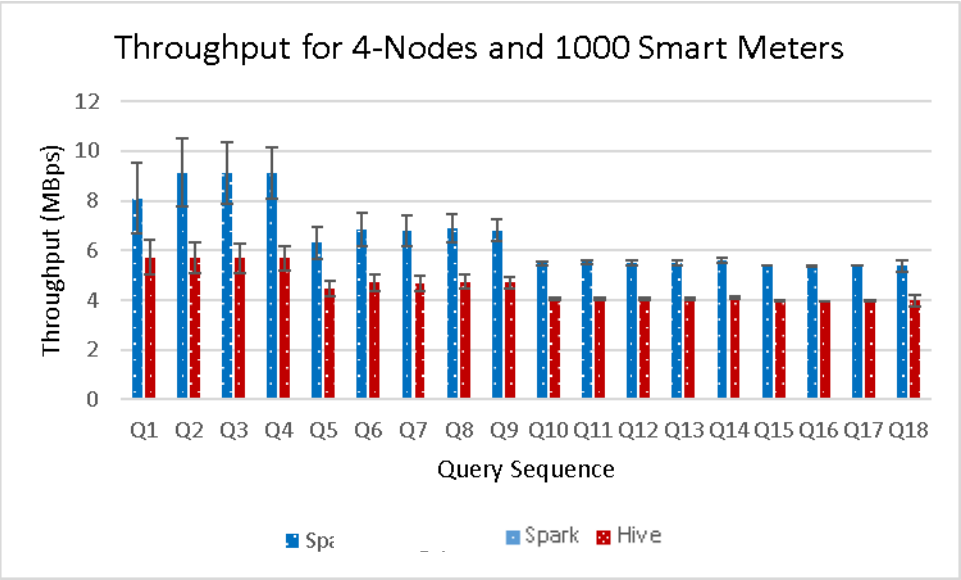

Figure A.2.21: Mean Throughput for Spark and Hive across 4 Nodes, 1000 smart meter files per query

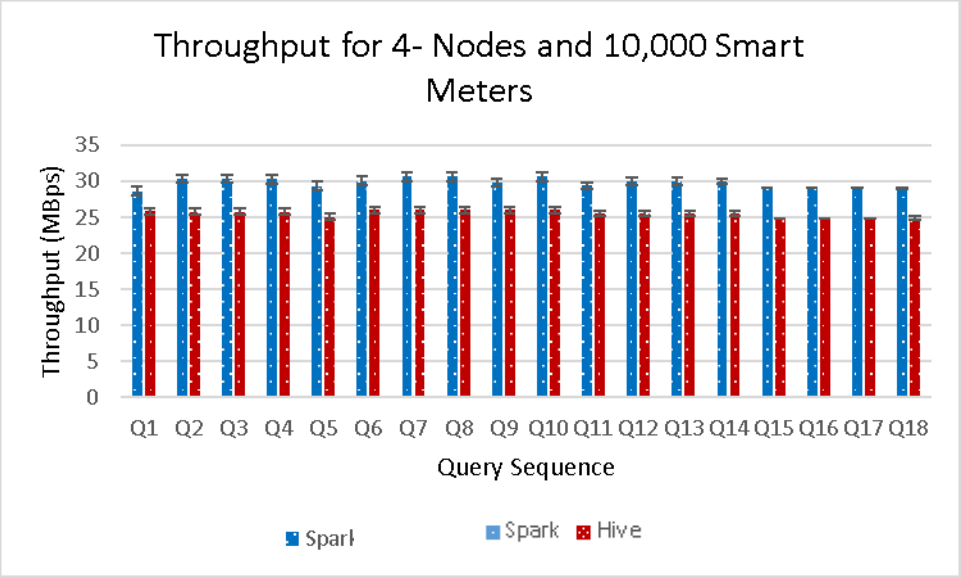

Figure A.2.22: Mean Throughput for Spark and Hive across 4 Nodes, 10000 smart meter files per query

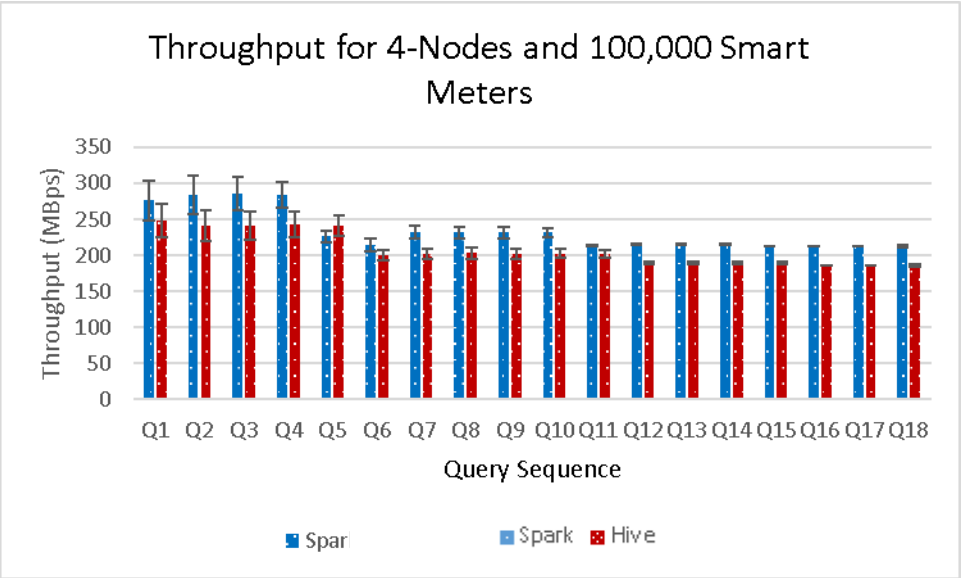

Figure A.2.23: Mean Throughput for Spark and Hive across 4 Nodes, 100000 smart meter files per query

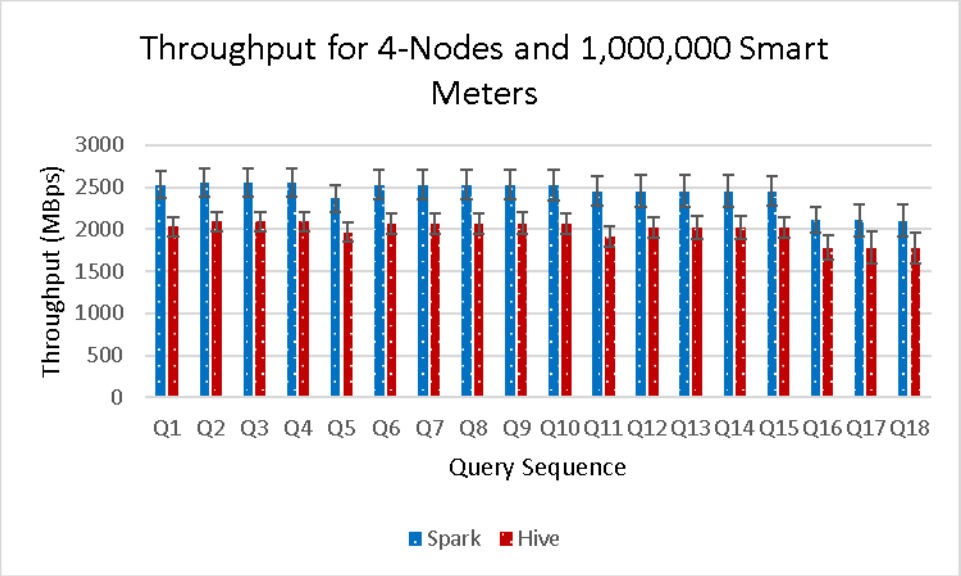

Figure A.2.24: Mean Throughput for Spark and Hive across 4 Nodes, 1,000,000 smart meter files per query

Vita

Ragini Gupta was born in 1993, in New Delhi, India. She received her primary, secondary and high school education in India. She received her B.Sc. degree in Computer Engineering from the American University of Sharjah in 2016.

In January 2016, she joined the master's program of Computer Engineering at the American University of Sharjah as a graduate teaching assistant. During the course of her master's study, she co-authored two conference papers and one journal paper. Her research interests are Big Data, Internet of Things, Cyber Physical Systems, and Real Time Embedded Systems.