5-2020

Improving Energy-Efficiency through Smart Data Placement in Hadoop Clusters

Ahmed Mostafa

COLUMBUS STATE UNIVERSITY

Improving Energy-Efficiency through Smart Data Placement in Hadoop Clusters

A THESIS SUBMITTED TO

THE D. ABBOTT TURNER COLLEGE OF BUSINESS

TSYS SCHOOL OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF

THE REQUIRMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

APPLIED COMPUTER SCEINCE

BY

AHMED MOSTAFA

COLUMBUS, GEORGIA

2020

ABSTRACT

Hadoop, a pioneering open source framework, has revolutionized the big data world because of its ability to process vast amounts of unstructured and semi-structured data. This ability makes Hadoop the 'go-to' technology for many industries that generate big data, thus it also aids in being cost effective, unlike other legacy systems. Hadoop MapReduce is used in large scale data parallel applications to process massive amounts of data across a cluster and is used for scheduling, processing, and executing jobs. Basically, MapReduce is the right hand of Hadoop, as its library is needed to process these large data sets. In this research thesis, this study proposes a smart framework model that profiles MapReduce tasks with the use of Machine Learning (ML) algorithms to effectively place the data in Hadoop clusters; activate only sufficient number of nodes to accomplish the data processing within the planned deadline time for the task. The model will ensure achieving energy efficiency by utilizing the minimum number of necessary nodes, with maximum utilization and least energy consumption to reduce the overall cost of operations in data centers that deploy the Hadoop clusters.

Keywords: Hadoop, ML, Energy-Aware, Big Data, MapReduce, HDFS

TO JULIA ‘CUMI’ BUNTING

In loving memory of my grandmother in law. You encouraged me to complete my Master’s degree and for me to never give up. If it were not for your kindness, understanding, and support I do not believe I would have gotten this far. I will always cherish our conversations on life, education, family, and our deep respect and love for one another’s beliefs.

ACKNOWLEDGEMENTS

To my advisor Dr. Yi Zhou and committee members Dr. Rania Hodhod and Dr. Shamim Khan, who have supported, advised me, and have become like a second family. I cannot thank you enough for all of your support.

To my wife and daughter who are the backbone of my strength.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1. Introduction

Big Data, we've all heard of it, living in a technology driven world, companies and organizations are constantly producing and collecting massive amounts of data. At such a rate, it is expected that by 2020 at least 35 zettabytes of data would be produced (Lublinsky B., et al. 2013 p. 1). With the rapid growth of different business sectors in the world, there becomes an increasing need for powerful data centers that are equipped with platforms like Hadoop clusters, which are capable of processing and communicating large scale of data. A study has been shown that data centers with a larger amount of servers can consume the power of megawatts in data processing and providing services in the Service Level Agreement (SLA), which in turn increases the electricity bill costs and can negatively affect business profitability (Qureshi, Weber, Balakrishnan, Guttag, & Maggs, 2009). Hadoop clusters with its open-source platform have proven to be a successful, efficient, and reliable business solution for data processing, i.e. *Facebook* & *Twitter* use Hadoop clusters in ML and data analytics operations ("Top 10 Industries using Big Data", 2016).

When Apache Hadoop came to the front, it was like a breath of fresh air. There was finally a solidified solution, which already proved to be successful in the commercial world. Being an open source project made it ever more popular and accessible. With the combination of Hadoop MapReduce, it allowed processing massive amounts of data shared on scalable clusters and performing convoluted data that wasn't possible to

analyze or index in the past. Nevertheless, things even gets better with the consideration of machine learning (ML) as a service in the Big Data scene, which will not only improve the way data is being processed, but will also allow for quick business decisions by understanding the patterns of the data itself. The ML algorithms that will be used in this proposal are the most used in Big Data and are from the family of supervised learning algorithms.

1.1 Brief history on Hadoop

Doug Cutting and Mike Cafarella started Hadoop in 2002, based off an Apache Nutch project that they were working on. The Apache Nutch project was based on building a search engine that can index billions of pages. During their research they found that building this search engine would cost half a million dollars, as well as thirty thousand dollars a month just in running costs, making this extremely expensive and not feasible at that moment. Realizing there was no way they could continue their project with the required costs, they began looking for a more cost-efficient solution in order to reduce the issue of storing and processing large data sets. Cutting and Cafarella came across a paper that Google released in 2003 on Google File System (GFS) which described how to store large datasets, according to Ghemawat, Gobioff, and Leung (2003). Realizing that this paper had half of what they needed to solve their issue, they carried on with their research. In 2004, Google yet again released a paper which provided the solution for what they were looking for in order to process large datasets, which is MapReduce (Dean

and Ghemawat, 2004). At this time, Google didn't actually implement GFS and MapReduce techniques, Cutting and Cafarella decided to try it out using both techniques GFS and MapReduce as an open source in their Apache Nutch project. With the project being open source, it would be able to reach more people.

After implementing both techniques, they found that Nutch was limited in clusters and needed a larger cluster to be able to run reliably, but unfortunately they were not able to do this on their own and needed to find a company that would be interested in their project and invest in it. That's when Cutting joined Yahoo! in 2006. He wanted to continue with his project to be open source and wanted to implement a dependable and scalable computing framework. Shortly after joining Yahoo!, he separated the distributed computing parts from Nutch and combined GFS and MapReduce and created Hadoop.

Yahoo! released Hadoop in 2008 as an open source with Apache Software Foundation (ASF), in which ASF tested successfully 4000 node cluster on Hadoop. Later in 2009, they were able to successfully sort a PetaByte of data under 17 hours which managed billions of searches and indexed millions of webpages using Hadoop. In the same year, Doug Cutting left Yahoo! and was employed by Cloudera. This gave him the ability to spread Hadoop to a larger array of industries, fulfilling his will of wanting to share Hadoop with the world.

In 2011, ASF released Apache Hadoop version 1.0 while version 2.0.6 became available with the inclusion of Apache Hadoop YARN in 2003, and the most recent version was released in December 2017, that is version 3.0 (White, 2015, p. 32 & 33).

1.2 Research Motivation

In the real world, it is likely common that a data center with 1000-rack consumes 10MW of power annum (Manzanares, Qin, Ruan, & Yin, 2011), which poses a burden on the budget in a way that can affect the overall business profitability. Therefore, there is a need for an efficient approach that can address the power consumption in Hadoop clusters since it is the most used framework in data centers, and this is what motivates this study. There are three pivotal factors that motivate this study:

- The high demand of cost minimization of the overall data center's operations where Hadoop clusters are deployed.
- The need for a resilient framework that efficiently reduces the energy consumption in the high workload Hadoop clusters.
- The significance of deploying a model that is able to predict the least necessary resources, which maximizes the node utilization and minimizes energy consumption.

1.3 Our Contribution

Unlike the traditional model-based data placement solutions, this research is adopting ML algorithms as an intelligent solution for data placement in Hadoop clusters.

- The research introduces a novel approach to build an energy-aware MapReduce framework that aims to reduce the energy consumption of data processing in data centers, which in turn should reduce the total cost of operations in these data centers.
- Furthermore, the reduction in the data processing cost exhibits a good business opportunity for data centers on the cloud by allowing them to give better offers for the servers' tenants, which can be appealing to more tenants and, accordingly, increase the profitability of the data center.

1.4 Challenges

Processing multiple workloads in terms of the operation type and the data size on Hadoop cluster is the preliminary phase in this thesis. Hadoop benchmarking comes in the place and play the main role for providing the necessary functionalities to test the performance of Hadoop cluster and/or studying the hardware resources needed for processing different workloads on the cluster. In general, Hadoop benchmarks are developed for more general test purposes and not for specific needs in terms of the size and the type of data being processed on the cluster, which proposed some challenges

related to the benchmark configuration. For example, in this study, in order to execute Wordcount MapReduce job for different workloads on Hadoop cluster, we had to generate different data size by replicating a certain amount of data several times and store the repeated data in a file for execution.

The Apache Software Foundation has released Hadoop in many releases such as (2.6.5, 2.7.2, 2.7.7, 2.8.4 …etc.), each release has slight configuration differences and different stability than the other ones. Some Hadoop releases have low stability, while some releases have better stability. This likely happens due to the bugs that potentially come with the open source software releases. From our experience, Hadoop official configuration does not really work as expected due to the different hardware resources with different configurations, in addition to the different operating systems configuration where Hadoop is installed. This posed a challenge for our study in terms of choosing the right operating system and the right Hadoop release that will optimally be configured for the available hardware resources in order to reach our study goals.

1.5 Thesis Organization

In Chapter 2 we talk about the Hadoop system architecture, its components and the contribution of each component to the Hadoop system performance, then we will demonstrate a brief definition for supervised and unsupervised ML algorithms. Related research and studies are discussed in Chapter 3. The Hadoop benchmarking (HiBench) is explained in Chapter 4, along with an illustration for the experimental hardware and

software setup. Chapter 5 discusses our research approach, in particular the data collection and how it was used by the ML model, data preparation pipeline, and the training of the ML model. Chapter 6 includes the experimental results, analysis of the ML models' performance, recommendation for the energy-aware Hadoop system architecture, our research conclusion and future work, and system scalability.

Chapter 2. Background

2.1 Hadoop System Architecture

Hadoop is an open source framework that is used to manage, store and process massive amounts of data running on large scalable clustered systems in a relatively short period of time. Hadoop provides a reliable, scalable, and fault tolerant system. It also offers a cost-effective way to store colossal amounts data without having to commit more processing power, thus, having the ability to scale only when needed. Hadoop systems is the heart of the Big Data ecosystem used in data mining, predictive analytics, and ML. With the ability to handle unstructured, structured, and semi-structured data, Hadoop can analyze, process, and distribute data. Accordingly, it becomes appealing to an array of industries in the big data realm (Lublinsky et al., 2013, p. 4). There are 3 main components of Hadoop as shown in Figure 2.1.1.

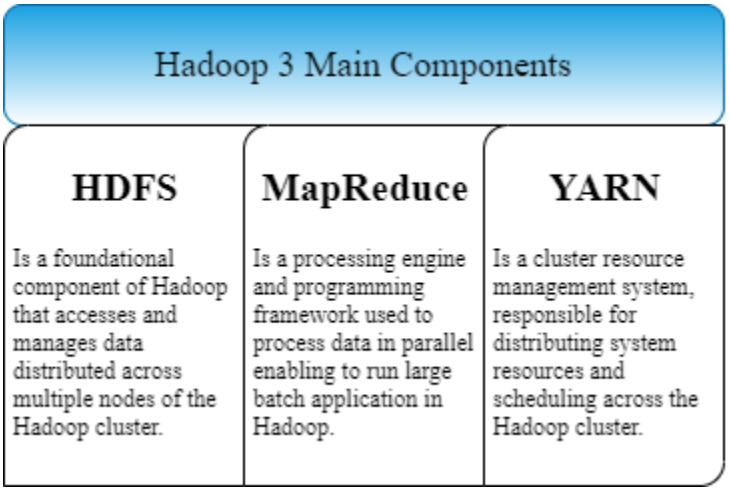


Figure 2.1.1: Hadoop's Main Components

2.1.1 HDFS

HDFS is a foundational component to Hadoop and a foundation for other tools. HDFS was designed to provide storage for exceptionally large files, i.e. petabytes and above. Data is written once but read multiple times, a process known as 'streaming data access pattern'. HDFS runs on commodity hardware (easily accessible and inexpensive) making HDFS much more affordable and easier to use in comparison to other file systems. HDFS splits files into blocks and sends them to numerous nodes across the Hadoop cluster (Lublinsky et al., 2013, p.20&21), thus, HDFS is a block-structured file system, see Figure 2.1.1.1. Master/Slave nodes (NameNode and DataNode) are what form HDFS cluster:

- NameNode (Master Node) manages the file system namespace. Stores all metadata of the filesystem across the cluster by which it is stored in the main memory. Metadata is designed to be compressed. NameNode manages the file system namespace and knows where all the DataNodes block files are located (Lublinsky et al., 2013, p.20&21). The functions of HDFS NameNode executes file system namespace operations, i.e., renaming, opening, and closing directories and files. Maintains and manages the DataNodes as well as mapping blocks of a file to DataNodes. NameNode maintains all locations of every block of a file, as well as the replication factor of all the blocks. Receiving heartbeat and block report from

the DataNodes, thus, ensuring that the DataNode is alive. In the event that DataNodes fail, a NameNode will select a new DataNode for new replicas.

- DataNode (Slave Node) are the workers (slave) of the filesystem, they store and bring back any blocks when ordered to do so and report back to the NameNode (Lublinsky et al., 2013, p.20&21). The functions of HDFS DataNode are to serve the client write/read requests and receive instructions from the NameNode to perform block creation, deletion, and replication, as well as submits block reports to NameNode which contains the list of blocks. The health of HDFS is reported from NameNode, as DataNode sends NameNode a heartbeat.

Blocks in HDFS architecture are files that are split into block-size pieces called block, by default are the size of the block are 128 Mb, but the block size can be configured based on requirements. As an example, consider a file size that is 612 Mb, HDFS creates four blocks that, by default, will be of size 128 Mb and then one block will be the size 100 Mb. On the other hand, a file size that is only 3 Mb will only use 3 Mb of the disk space, thus allowing this small sized file to not occupy the full block size space in the disk.

Blocks Replication Management in HDFS consists of storing replicas of a block on numerous DataNodes that are based on a replication factor. The number of replicas to be replicated for blocks of a file is called the replication factor in the HDFS architecture. As an example, if the replication factor is 2, then two replicas of the block

will be stored on different DataNodes, thus allowing the block to be accessible from another DataNode that contains the replica in the event if one of the data blocks fails. Say that we want to store a file of 128 Mb and the replication factor is 2. Thus (2*128=256) 256 Mb of disk space will be used for a file as two copies of the block will be stored ("Hadoop HDFS Architecture Explanation and Assumptions" 2020).

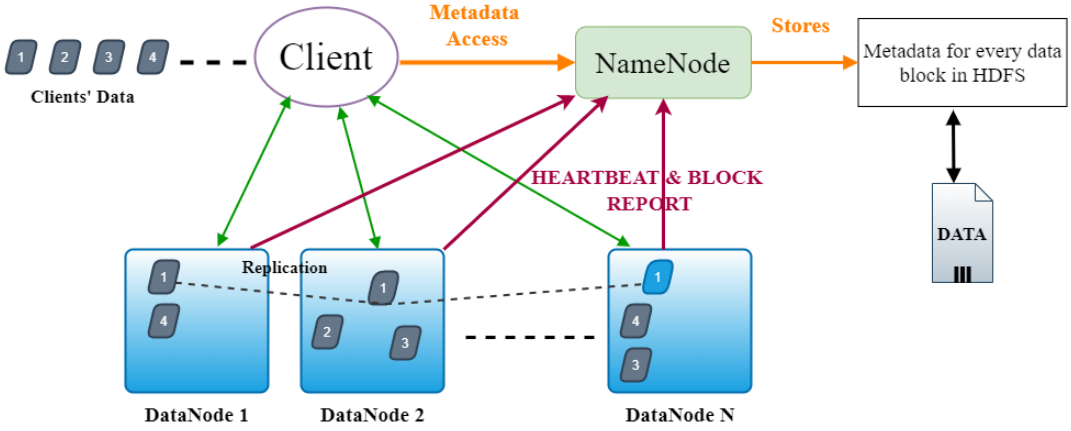


Figure 2.1.1.1: HDFS Architecture

2.1.2. MapReduce

Google invented MapReduce in 2004, which was suitable for parallel data processing in a distributed computing environment. MapReduce was designed to run on commodity hardware in order to solve large data computational issues and problems. MapReduce is a framework in which data splitting, data distribution in the cluster, data parallel processing, execution synchronization, and fault tolerance is automatically

managed (Agarwal and Khanam, 2015). MapReduce framework is typically composed of two tasks: Map Task & Reduce Task.

- **Map Task (*Mapper*)**: Takes the input data in the form of key value pairs and then generates the output in the form of key value pairs (Agarwal and Khanam, 2015). Below are the various phases of the Map Task.

  RecordReader: Converts the input split into records, the data is then parsed into records, but does not parse itself. The data is given to the mapper function in key value pairs.

  Map: A user defined function that processes from the RecordReader the key value pair, producing multiple or zero intermediate key value pairs. The key value pair is determined by the mapper function. Usually the key is the data that the reducer function performs the grouping operation. The value is the combined data that gets the final result in the reducer function.

  Combiner: An optional function used as a localized reducer that groups the data in in the Map phase. In many scenarios, aggregating the intermediated data from the mapper decreases the amount of data that is needed to move over the network and provides ultimate performance gain without any disadvantages. However, the combiner is not always guaranteed to execute.

  Partitioner: Takes from the Mapper the intermediate key value pairs, then splits them into shards, allowing one shard per reducer. The Partitioner by default

retrieves the hash code of the key, and evenly distributes the keys on the reducers by performing modulo operations by the number of reducers (key.hashcode()%number of reducers). This provides that the key with the same value from different mappers will ultimately end at the same reducer. From each map task the partitioned data is written onto the local file system, awaiting there for the reducer to pull it ("Hadoop Architecture in Detail – HDFS, Yarn, & MapReduce" 2019).

- **Reduce Task (*Reducer*)**: Takes the input of key and list of value pairs then generates the output as key value pairs. The output in this phase is the final output (Agarwal and Khanam, 2015). Below are various phases of the Reduce Task.

  Shuffle & Sort: The first step for the reducer is shuffle and sort, which downloads to the machine where the reducer is running the data written by the Partitioner. This step then sorts pieces of the individual data to a large data list, collecting the equivalent keys, by doing so, allowing the framework to make it easier to iterate in the reduce task. Although this phase is not customizable, the framework automatically handles everything while ensuring that the developer has complete control on how the keys are grouped and sorted through a comparator object.

  Reducer: Performs the reduce function once per key grouping. The framework hands the iterator object and function key that contains all values belonging to the key. The Reducer can be written to filter and combine data in multiple ways. When

the reduce function finishes it gives an OutputFormat of either zero or more key value pairs. The Reduce function, similar to the Map function, is different from job to job.

OutoutFormat: The final phase in the reduce task includes taking the key value pair from the Reducer and writing it in a file by the recordwriter. Separating the key and value by default are separated by tab with each record by a newline character. The final data will be written to HDFS ("Hadoop Architecture in Detail – HDFS, Yarn, & MapReduce" 2019).

A *job* in the MapReduce model is an application that is to be executed. An example of the MapReduce model is shown in Figure 2.1.2.1. The mapper and reducer jointly create a Hadoop job. It's worth mentioning that the mapper is a compulsory part of the job and the reducer is noncompulsory, the user is still responsible for implementing the logic that will give the desired output for his own task (Lee, Hsieh, Hsieh, & Hsiao, 2014). In MapReduce, there are two daemons to process executing jobs: JobTracker and TaskTracker.

- **JobTracker** is in charge of all the jobs scheduling and task dispersion.
- **TaskTracker** is the worker and must execute all tasks given and return the results to the JobTracker (Lee, Hsieh, Hsieh, & Hsiao, 2014).

JobTracker and TaskTracker communicate with one another using a heartbeat message in which these heartbeats tell the JobTracker that the TaskTracker is still alive and the TaskTracker is able to signify what time it would be ready to run a new task (Lublinsky et al., 2013, p. 68).

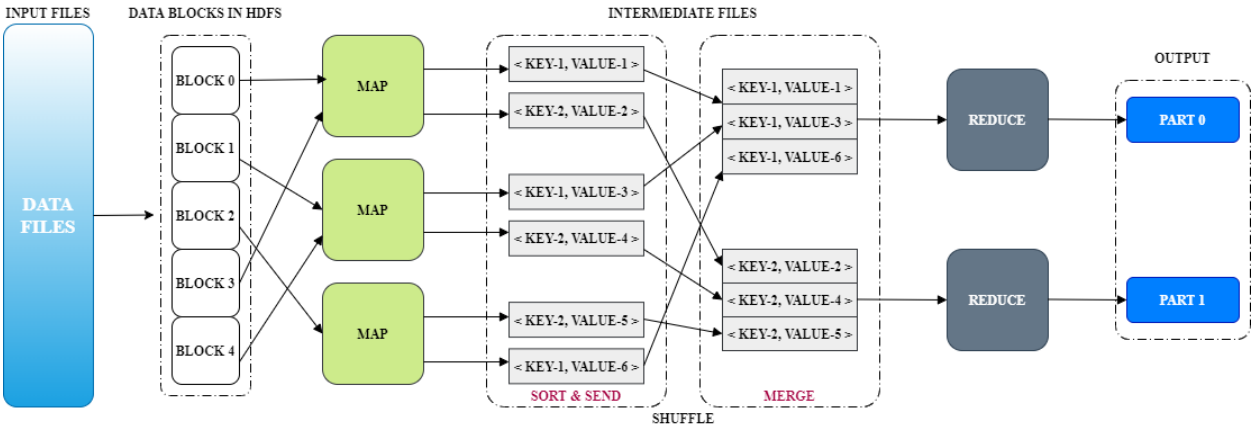


Figure 2.1.2.1: MapReduce Job Flow

2.1.3. YARN

YARN stands for, Yet Another Resource Negotiator, was introduced in Apache Hadoop 2 in 2013 as a new cluster resource management system. Although designed to improve MapReduce implementation, it is also capable to sustain other distributed computing paradigms (White, 2015, p.97). It is important note that YARN does not totally replace MapReduce but can be used alongside it. By introducing YARN, it took away the complete reliance on MapReduce and opened the door for Hadoop to run applications on other engines such as Apache Spark, Apache Kafka, Apache Flink as well as Apache Storm.

YARN separates the job scheduling and resource managements into two separate daemons, basically separating the functionality of MapReduce's JobTracker. There are two long-running daemons, *one* called the Resource Manager (RM), which comprises of an Application Manager and Scheduler and manages the resources across the cluster, and *the second one* called the Application Master (AM), which caters to the support of specific applications. What the AM does once it runs depends completely on the application itself, whereas the application could be either a single job which is common in jobs done in MapReduce or multiple jobs as a Directed Acyclic Graph (DAG). It is important to note that YARN does not alter the MapReduce programming model or its API's but makes a way for a different resource model in carrying out MapReduce jobs. Most MapReduce applications will work as they are, but most likely need to be recompiled.

The architecture of YARN in Figure 2.1.3.1 shows the client program submitting an application with all the specifications needed for the AM, thus all the information needed must be provided to the RM so the RM then finds a node manager in order to launch the initial container. What the AM does once it runs depends completely on the application itself; within the container it is running it could run a computation itself, and return the results to the client, or it could send a request to the RM for more containers, so that it can run a distributed computation. A container may use a UNIX process (Lublinsky et al., 2013, p.450-451).

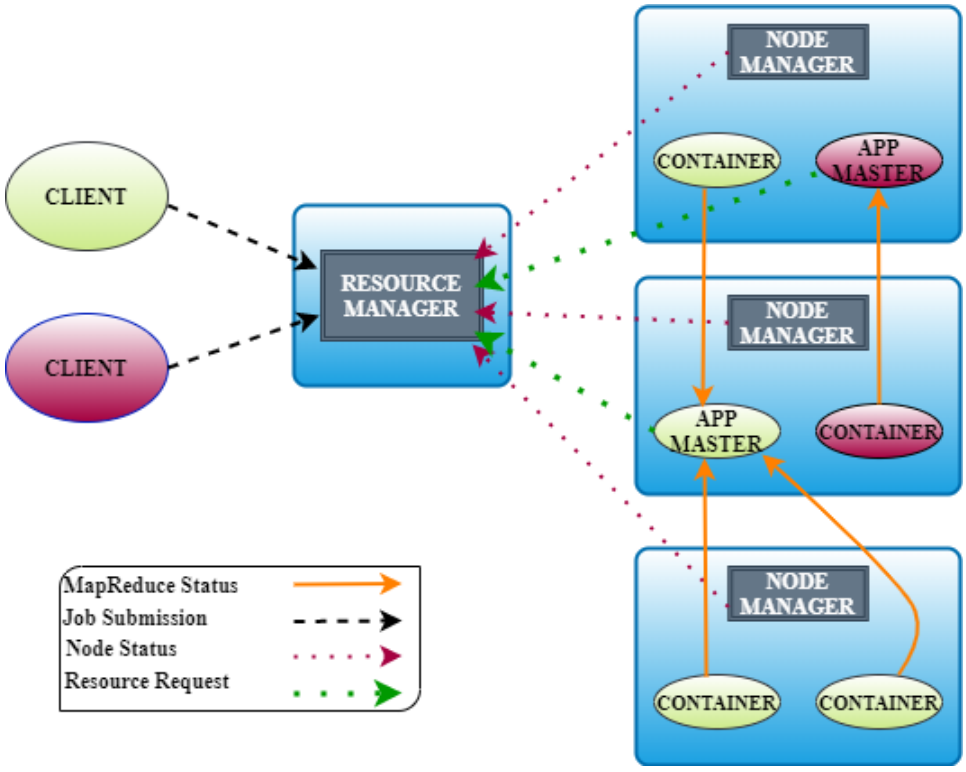


Figure 2.1.3.1: YARN Architecture

Although there are three main components in Hadoop as shown in Figure 2.1.1, there are more core components to Hadoops ecosystem which is ever growing. Other components are HBase, ZooKeeper, Oozie, Pig, and Hive, just to mention a few. Hadoop has the ability to store large amounts of data, is flexible, cost effective, has high computational power, and linear scaling. Hadoop has become the superpower in the Big Data Industry.

2.2. Machine Learning (ML)

ML is a subarea of Artificial Intelligence, which focuses on algorithms that are designed to give computer systems and software applications the ability to learn automatically and improve without being programmed to do so. The main objective of

ML is to make it available for computers to learn without the need of assistance from humans. With that said, the learning process often consists of data or observations, in which ML looks for patterns in the data or the information that has been observed in order to make better decisions or more accurate decisions.

ML algorithms are generally categorized as supervised and unsupervised learning classification, but in Big Data, Supervised Learning are typically the go-to algorithms used. However, unsupervised learning is also used. Below I will explain what supervised learning and unsupervised learning are and the algorithms commonly used in Big Data.

Supervised Learning is when you train or teach the machine using a learning algorithm from training datasets. The algorithm makes predictions from the datasets given and is corrected, once the learning stops, that's when the algorithm has reached an optimal level of performance (Brownlee, 2016).

- *Logistic Regression*: A categorical algorithm which is used to appoint observations in a distinct batch of classes. The achievement depends much on the size of training data ("Logic Regression", 2017).
- *Multiple Linear Regression*: A regression algorithm which attempts to observe more than one independent variables and one dependent variable by finding an optimal fitted linear equation that describes the observed data ("Multiple Linear Regression", 1997-1998).

- *Naïve Bayes*: Based on Bayes Theorem and is a compilation of probabilistic classification algorithms. It is scalable, does not require large training datasets (Soni, 2018).
- *Random Forest*: A collection of decision trees. Works well with large datasets but should use caution when creating too deep of a tree as it could cause overfitting (Donges, 2019).
- *K-Nearest Neighbor (KNN)*: Used for both regression and classification predictive issues. It can be computationally challenging in both test and training phases, as they correlate all training samples when classifying all test samples (Apruzzese, Colajanni, Ferretti, Guido, & Marchetti, 2018).
- *Support Vector Machines (SVM)*: Non-probabilistic classifier, defined as a separating hyperplane. This particular algorithm is not very scalable and is best used as a binary classifier (Patel, 2017).

Unsupervised Learning: Data is not labeled or classified; thus, its main goal is to infer a function from unlabeled data in order to describe a hidden structure (Dua S. and Du, X. 2011, pp. 31).

- *Clustering*: Divides and then regroups data points into groups that are related and more similar. Although there are hundreds of clustering algorithms, two of the most commonly used clustering algorithms are K-Mean clustering and Hierarchical clustering (Kaushik, 2016).

Chapter 3. Related Work

Many studies have been conducted to improve Hadoop clusters' energy efficiency; numerous algorithms and platforms have been developed in order to minimize the amount of power consumed in data centers. In this thesis, I will address related work of the energy consumption techniques from three conceptual points of views: scheduling and allocating resources in the cloud and data centers, efficient energy utilization in data centers, and different approaches of scheduling MapReduce jobs.

3.1 Scheduling and allocating resources in the cloud and data centers

Estimating the resources needed for computations in data centers and the cloud can be an effective method for cost reduction, probabilistic models have been built for task scheduling in cloud computing by using Erlang stochastic (Hacker, Mahadik 2011). The study shows that modeling the probability of resources needed, and task waiting queue based on different workloads can help in estimating the clusters' size and the amount of spare resources needed, which would possibly control the cost of needed resources and therefore effectively reduce the cost of energy consumed in data centers.

According to Tian and Chen (2011), by modeling MapReduce processing components, data centers' resource provisioning can be optimized, and therefore jobs processing cost can be minimized. Tian and Chen have proposed a function that helps to reduce the jobs' financial cost, the function models the relationship between input data,

resources needed, i.e. slots for Map and Reduce tasks and the job complexity, where the function parameters can be utilized based on the requested job.

Palanisamy, Singh, and Liu (2015) have presented a MapReduce model for data processing in the cloud. Their model automates cluster configuration in the cloud based on the job deadline and the MapReduce profile of the reuqested job, which can globaly optimize the resource utilization in the cloud. According to Palanisamy et al. (2015), the model significantly reduces data center resources cost by 80% for processing workloads such as Facebook workloads.

A Microsoft research has been conducted by (Jalaparti, Ballani, Costa, Karagiannis, & Rowstron 2012) on making the data center service providers more efficient in the cloud. Based on the customer's MapReduce job complexity and the customer's cost constraints, the system's model predicts multiple tuples of resources as computetional resources and network bandwidths. Then, the resource tuple choice is made according to the customer's job desired completion time, the existance of resources that yields the cheapest cost for the cloud service provider, and the ensurance of resource avaliability for future customers.

3.2 Efficient energy utilization in data centers

Kaushik, Bhandarkar, and Nahrstedt (2010) have simulated an approach of classifying Yahoo's Hadoop cluster servers into two categories, hot and cold categories.

Hot category classifies the servers that currently have data that's being accessed, and cold category classifies servers that are in a sleeping mode. This classification is based on data processing classification in terms of performance requirements, cost, SLA, and power characterstics, which in turn affects the data placment in HDFS in the cluster. The researchers' simulation show that their approach can reduce the power consumption in Yahoo Hadoop cluster by 24% annually.

Goiri et. al. (2012) have proposed a novel Hadoop framework (GreenHadoop) that aimed to reduce the On-Grid energy consumption in data centers by relying more on the solar power as an alternative renewable energy. Their proposed framework schedules MapReduce jobs in a way that allows the maximum amount of solar energy to be used to complete the jobs within its deadline constraint, and if the On-Grid power has to be used in order to meet the jobs' deadline constrain, then the framwork schedules these jobs in the time where the On-Grid energy consumption is the cheapest.

Wirtz and Ge (2011) have conducted an experiment on Hadoop MapReduce tasks to improve energy efficiency in data centers. Their energy reduction approach is based on two techniques: changing the amount of concurrently working nodes, and adusting the scaling of the CPU's frequency and voltage, where both techniques are based on the MapReduce jobs computational characteristic.

In another study on reducing the energy consumption in Hadoop clusters, Lang and Patel (2010) have performed an experimental comparison between two exterme

approaches on different MapReduce workloads. The first approach is based on powering up a few number of nodes when the cluster is underutilization, the second approach is based on using all clusters' nodes for processing a MapReduce workload and then shutting down every single node in the cluster. The second exterme approach has been proven to be more effective in improving the energy efficiency in Hadoop cluster according to (Lang & Patel, 2010).

3.3 Different approaches of scheduling MapReduce jobs

Sandholm and Lai (2009) have presented a resource allocation system that improves the scheduling process of MapReduce jobs. The system achieves its goal in three ways: the user-assigned and regulated priorities for different service levels to jobs, allocating cluster's resources is adujsted dynamically to satisfy job phases, automatic detection and elimination of the bottlenecks during the job processing life time.

Wang, Shen, Yu, Nie, and Kou (2013) have proposed a scheduling technique that improves system throughput in job-intensive environments. Their schedular algorithm analyzes the MapReduce job requirements, and satisfies four main factors that can improve system throughput, the factors are: data processing locality should be maintained at its highest ratio, choosing the nonlocal processing that keeps the system throughput high, keeping stored data on the cluster nodes as balanced as possible to avoid poor network performance, and making use of all the cluster computing resources

by lessening the amount of idle nodes. This schedular algorithm improves system throughput on the expenses of the energy consumption.

Verma, Cherkasova, and Campbell (2011) have proposed a non traditional MapReduce framework for controlling the cluster resource allocation towards achieving applications performance objectives. In their approach, firstly, they profile the MapReduce job based on its performance characteristics during the map and reduce phases. Secondly, they built a model that is able to estimate the necessary cluster resources needed to complete the MapReduce job based on the job profile and a given job deadline for completion. Finally, they implement a job schedular in Hadoop that orders the MapReduce jobs and determines the amount of needed resources, to ensure meeting jobs completion time requirement.

Kurazumi, Tsumura, Saito, and Matsuo (2012) study weren't concerned about the energy efficiency in Hadoop cluster, but they focused on improving the node's CPU efficiency for the I/O bounded jobs, instead. Their approach of improving the CPU performance is to dynamically detect the I/O waiting times during the MapReduce jobs execution and schedule more tasks to the CPU processing slots during these times to shorten jobs execution time.

Chapter 4. Benchmarks & Experimental Setup

4.1 HiBench

HiBench is considered a big data benchmark suite for the Hadoop framework and is the most commonly used application in MapReduce jobs. The benchmarks used comprehensively classify big data Hadoop framework in terms of system resource utilization, throughput, and speed. The benchmarks used in this research for unstructured data include Micro benchmarks, i.e., WordCount, Sort, and TeraSort. For semi-structured data included Web Search benchmark, i.e., PageRank. For Machine Learning benchmarks I used K-Means (Huang S., Huang J., Yan, Lan, Jinquan, 2010).

4.1.1 Micro Benchmark

- ***WordCount*** is a CPU bound process. WordCount benchmark reads the input text file that calculates how many times each word occurs. Using the RandomTextWriter program found in Hadoop, the input data is created by executing the script for the workload. This job takes away a small amount of information from data of a larger source (Huang S., Huang J., Yan, Lan, Jinquan, 2010).
- ***Sort*** is an I/O bound process. Sort benchmark as its name suggests sorts, sorting the input text file by key. Using the RandomTextWriter program found in Hadoop,

the input data is created by executing the script for the workload. This program uses map or reduce to run the job where the tasks write large series of unsorted words without interaction between the tasks. Based on key, the output of the key value pairs in map phase get sorted and shuffled and then is reduced again based on key. During the shuffle and merge stages of the MapReduce model the data is automatically sorted (Huang S., Huang J., Yan, Lan, Jinquan, 2010).

- ***TeraSort*** is both a CPU bound process (during map phase) and I/O bound process (during reduce phase). Similarly, like *Sort benchmark* it sorts by key the input text file, however, TeraSort has the ability to sort and distribute equal loads to all nodes during the process and uses either map or reduce to sort the final order of samples input data. Using the TeraGen program found in Hadoop, which uses either map or reduce to create data, the input data is created by executing the script for the workload, and by default has the ability to produce billions of byte records (Huang S., Huang J., Yan, Lan, Jinquan, 2010).

4.1.2 Web Search Benchmarks

PageRank is a CPU bound process. It measures the quality and importance of a website as well as calculates the number of these websites and links. The implementation of the PageRank algorithm is used in MapReduce for large scale search indexing. The workload comprises of multiple Hadoop MapReduce jobs, and are

iterated until conditions of coverage are satisfied (Huang S., Huang J., Yan, Lan, Jinquan, 2010).

4.1.3 Machine Learning Benchmarks

K-Means Clustering is both CPU bound (during iteration) and I/O bound (during clustering) process. K-means is a widely used clustering algorithm in machine learning. This clustering algorithm can be used in Hadoop by executing the Hadoop job iteratively until the desired number of iterations have met the specified limit, then allowing the clustering job to run and assigns each sample to a cluster. Each sample is defined as a numerical d-dimensional vector. The workload input is created based on a statistic distribution using a random data generator. (Huang S., Huang J., Yan, Lan, Jinquan, 2010).

4.2 Experimental Setup

One of the most Hadoop's traction features is its capability to run on commodity hardware, particularly when processing batch jobs overnight for reports or actionable information production. Unlike batch jobs production environment, processing real-time jobs on Hadoop cluster require very high hardware specifications such as large memory size i.e. 512 GB. In this study our goal is developing energy-aware Hadoop cluster framework for processing batch jobs, therefore, we conducted the experiment on commodity hardware.

4.2.1 Hardware and Prerequisites Installation

We setup 7-nodes Hadoop cluster (1 NameNode and 6 DataNodes), Table 4.2.1 below depicts the hardware specifications of the cluster's nodes.

Table 4.2.1: Cluster's Nodes Hardware Specifications

Node	Specifications
NameNode	2.4GHz CPU (4 cores), 8 GB Memory, 228 GB HDD
DataNode 1	2.4GHz CPU (4 cores), 4 GB Memory, 228 GB HDD
DataNode 2	2.4GHz CPU (4 cores), 8 GB Memory, 228 GB HDD
DataNode 3	2.4GHz CPU (4 cores), 8 GB Memory, 228 GB HDD
DataNode 4	2.5GHz CPU (4 cores), 8 GB Memory, 457 GB HDD
DataNode 5	2.3GHz CPU (4 cores), 16 GB Memory, 468 GB HDD
DataNode 6	2.3GHz CPU (4 cores), 16 GB Memory, 468 GB HDD

Each node is equipped with Ubuntu 16.04 Operating System. We connected the nodes to power meters to enable measuring the power consumption in KWh for every workload processed. We started Hadoop cluster environment setup by installing and configuring all the prerequisites software tools and packages on each node, such as Java OpenJDK 1.8.0_252, psutil 5.7.0 (Cross-platform lib for process and system monitoring in Python)….etc.

4.2.2 Multi-Node Hadoop Cluster Setup & Configuration

After the prerequisites, we setup multi-node Hadoop cluster 7-nodes by installing Apache Hadoop 2.7.2 distribution on each node. We configured 1 node as a

master (NameNode) node which is responsible of managing the file system namespace and regulates clients file access. We configured 6 nodes as DataNodes (slaves) which are responsible of storing actual business data in blocks, managing these data blocks based on the NameNode demand, and respond to the read/write requests from the client's file system.

There are four XML files which include the main Hadoop cluster configuration, these files are:

- core-site.xml: Contains configuration for the core Hadoop functionalities that are essential to MapReduce and HDFS such as I/O settings.
- hdfs-site.xml: Contains configuration for the NameNode, secondary NameNode, DataNodes, and the HDFS daemons settings.
- mapred-site.xml: Contains configuration settings for MapReduce daemons such as Map tasks and Reduce tasks (note that job tracker and task tracker are deprecated properties in Hadoop v2.7.2).
- yarn-site.xml: Contains configuration settings for NodeManagers and ResourceManagers.

We have configured Hadoop cluster with a high efficient and maximum resource utilization goal in mind. The details of the configuration settings for the above four files are presented in Appendix A.

Chapter 5. Research Methodology

This study aims to profile MapReduce tasks with the use of ML algorithms to effectively place the data in an energy-aware Hadoop clusters; activate only sufficient number of nodes to accomplish the data processing efficiency by utilizing the minimum necessary nodes, with maximum utilization, and least energy consumption to reduce the overall cost of operations in data centers that deploy the Hadoop clusters. We have used HiBench benchmark for profiling MapReduce workloads. The benchmarks are micro benchmarks (Terasort, Sort, Wordcount), web search benchmark (Pagerank), and the machine learning benchmark (K-means clustering algorithm). Our research objective was inferenced through the below two phases:

5.1 Phase 1: Resource Utilization measurements

The procedure started by installing Hadoop cluster (7-nodes): one master (NameNode) node and 6 slave (DataNode) nodes. The HiBench benchmarks were configured to generate the desired workload data size that corresponds to each test. In order to measure the power consumption with each workload processed on the cluster, we used dedicated power meters which were always connected to the master and slave nodes.

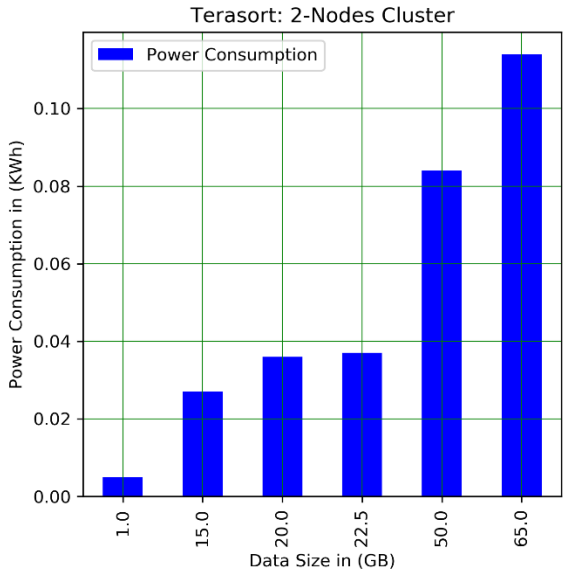
To study the minimum necessary hardware resources needed for processing MapReduce workloads with the lowest possible power consumption, we characterized

different MapReduce workloads. The strategy that we followed was observing the energy consumption at different workloads and different number of cluster's nodes. Besides, observing the power consumption of the cluster's nodes we also observed the nodes hardware resources utilization such as CPU utilization, memory utilization, and storage utilization. Our experiment was carried out on two types of operations; I/O bound (Terasort, Sort) in which the major job's time to complete is spent on waiting for I/O operation to be completed, and CPU bound (Wordcount, Pagerank, Kmean) in which the major job's time to complete is spent on waiting for operations using the CPU to be completed.

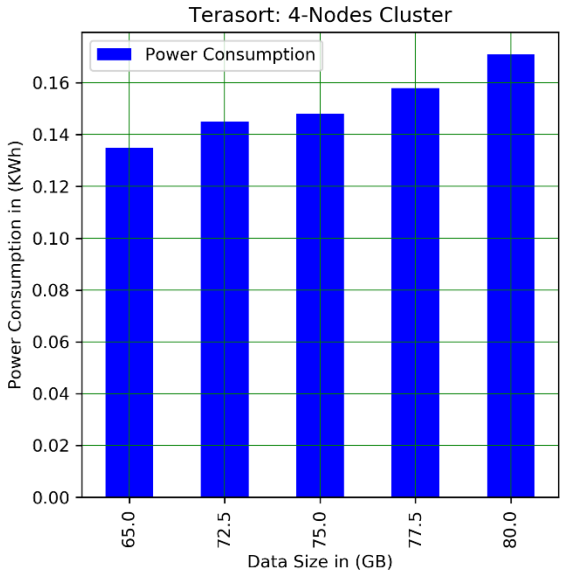
5.1.1 Terasort

The limitation of the I/O bound jobs such as Terasort is the cluster's storage capacity. Due to the DataNodes storage capacity, in our experiment we could only process up to 65 GB data size on the 2-nodes Hadoop cluster which is NameNode equipped with 228 GB Hard Disk Drive (HDD) and one DataNode that was equipped with 228 GB (HDD). Figure 5.1.1.1 shows the power consumption of processing different Terasort workloads at different number of Hadoop cluster's nodes. Using the HiBench Terasort benchmark configuration we generated different workloads by changing the data size parameter in the Terasort benchmark configuration file i.e. data size parameter = 100000000 for generating 10 GB data size, 1000000000 for generating

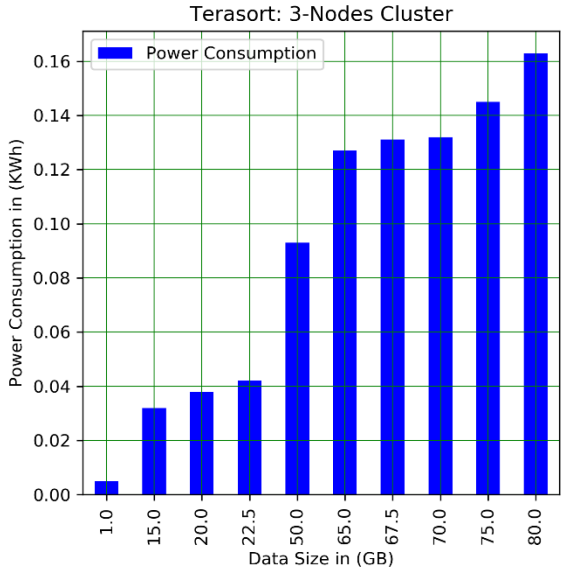
100 GB data size ...etc. then we processed the generated workloads on different numbers of Hadoop cluster's nodes.



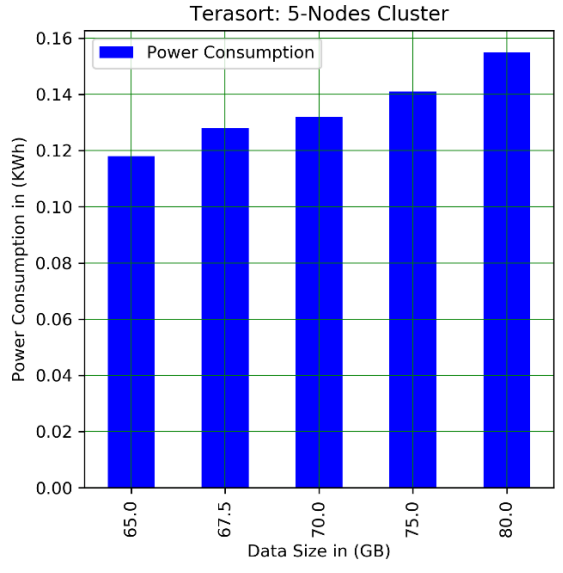
(a)



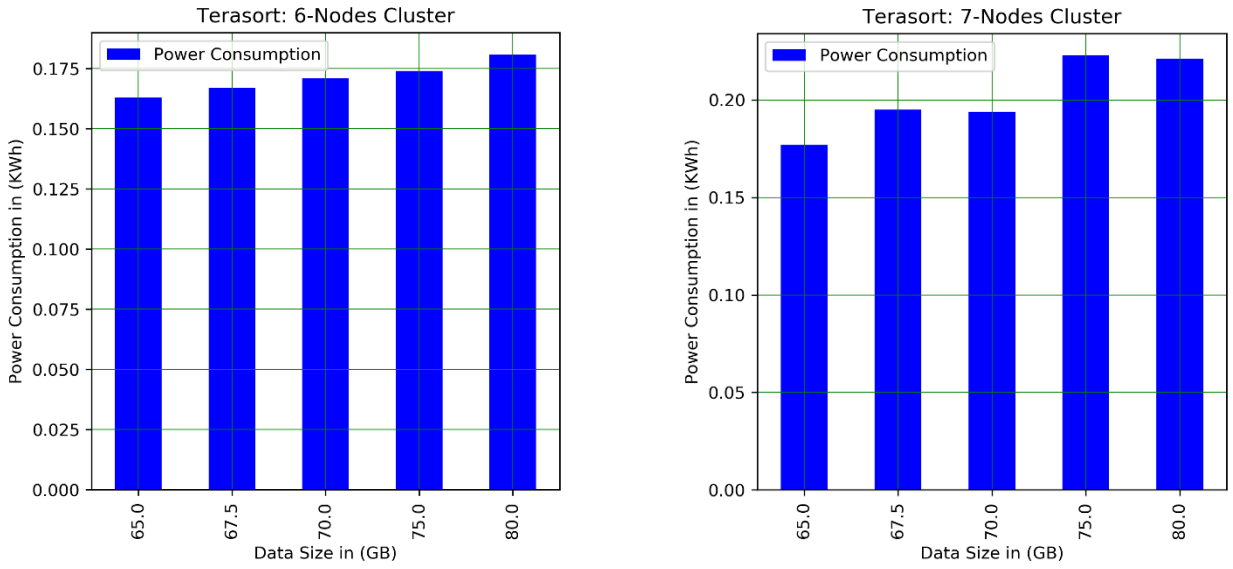
(c)



(b)



(d)



(e) (f)

Figure 5.1.1.1: Power Consumption vs Terasort Workloads on Hadoop Cluster

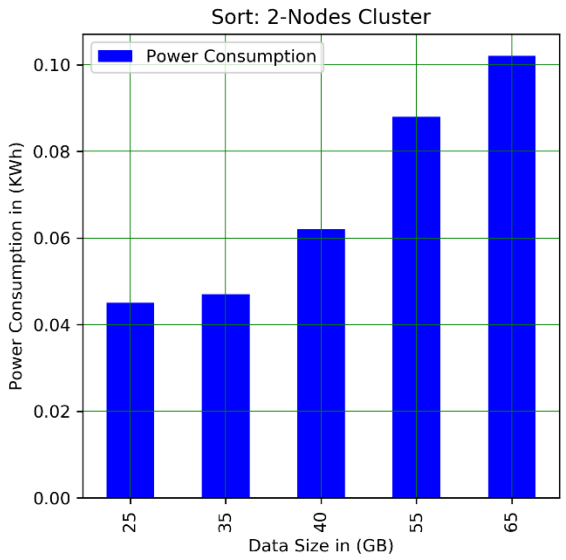
For example, the power consumption of processing Terasort workloads on 5 nodes Hadoop cluster is shown in Figure 5.1.1.1 (d) , the vertical axis represents the cluster's total power consumption in KWh for executing different Terasort MapReduce workloads at (65.0, 67.5, 70.0, 75,.0, 80.0) Giga bytes, which are represented on the horizontal axis.

The complete experimental results for all the Terasort workloads and the cluster's resource utilization is shown in Appendix B.

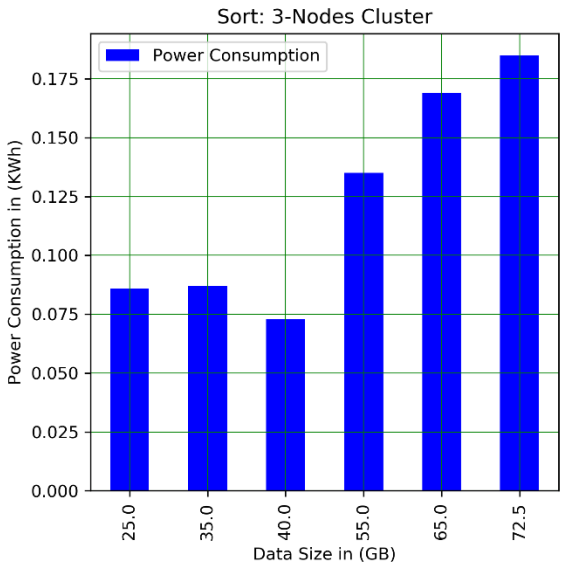
5.1.2 Sort

In Hadoop cluster, Sort is an I/O bound job, so as mentioned before the limitation in this type of jobs is the cluster's storage capacity. In our experiment we could not

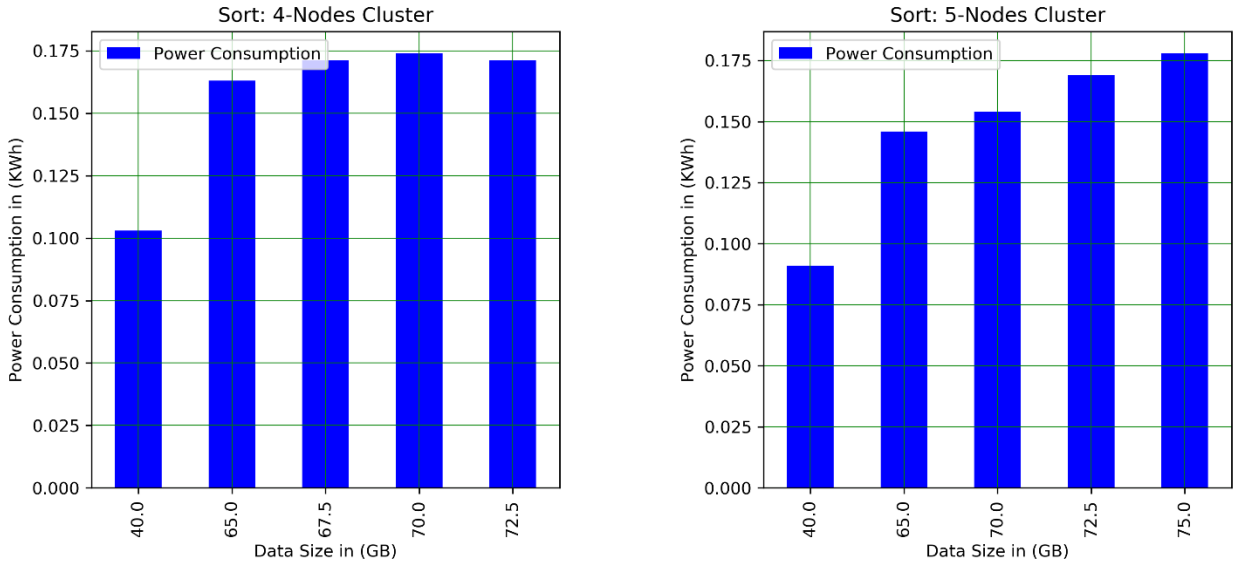
process more than 65 GB data size on 2-node Hadoop cluster due to the storage capacity limitation, where processing 67.5 GB on the 2-nodes cluster has failed to complete. Similarly, 3-nodes (one NameNode and two DataNodes) Hadoop cluster could process only up to 72.5 GB sort workloads, the 75 GB sort workload has failed to complete on 3-nodes Hadoop cluster. Figure 5.1.2.1 shows the power consumption of processing different Sort workloads at different number of Hadoop cluster's nodes. Using the HiBench Sort benchmark configuration we generated different dataset size in a similar manner as what we did in the Terasort benchmark, then we processed it on different numbers of Hadoop cluster's nodes.



(a)



(b)



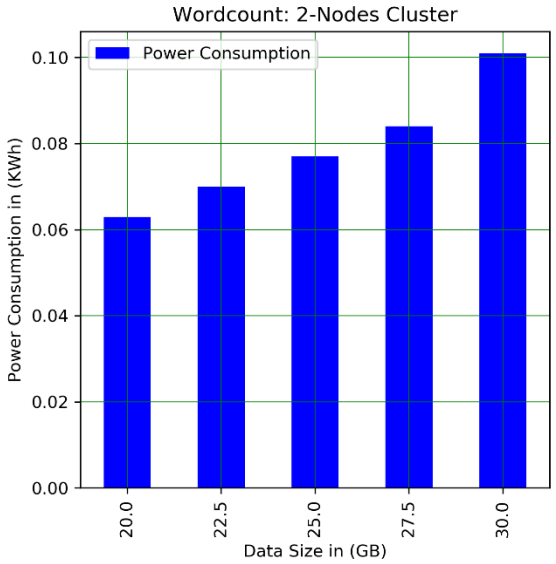
(c) (d)

Figure 5.1.2.1: Power Consumption vs Sort Workloads on Hadoop Cluster

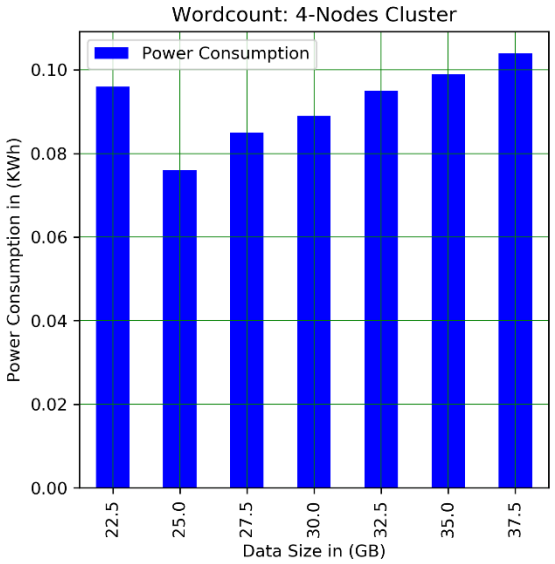
The complete experimental results for all the Sort workloads and the cluster's resource utilization is shown in Appendix B.

5.1.3 Wordcount

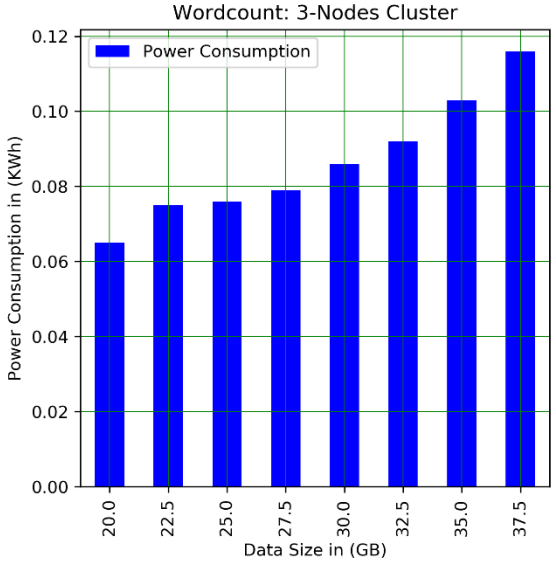
Unlike I/O bound jobs, CPU bound jobs such as Wordcount MapReduce job, the limitations is the CPU utilization. During the experiment we have not experienced having average CPU utilization over 90%, however we have observed an average CPU utilization of 88.60% while processing 35 GB Wordcount workload on 3-nodes (one NameNode and 2 DataNodes) Hadoop cluster, during a 1 hour and 5 minutes period of time. Figure 5.1.3.1 shows the power consumption of processing different Wordcount workloads at different number of Hadoop cluster's nodes.


Wordcount: 2-Nodes Cluster
Power Consumption
Power Consumption in (KWh)
0.00
0.02
0.04
0.06
0.08
0.10
20.0
22.5
25.0
27.5
30.0
Data Size in (GB)

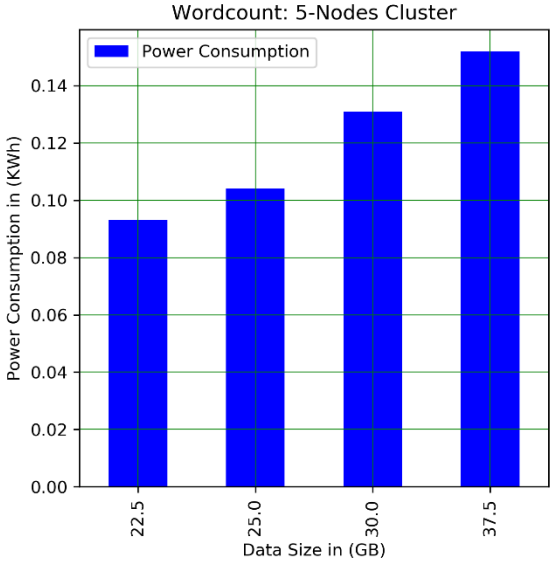
(a)


Wordcount: 4-Nodes Cluster
Power Consumption
Power Consumption in (KWh)
0.00
0.02
0.04
0.06
0.08
0.10
22.5
25.0
27.5
30.0
32.5
35.0
37.5
Data Size in (GB)

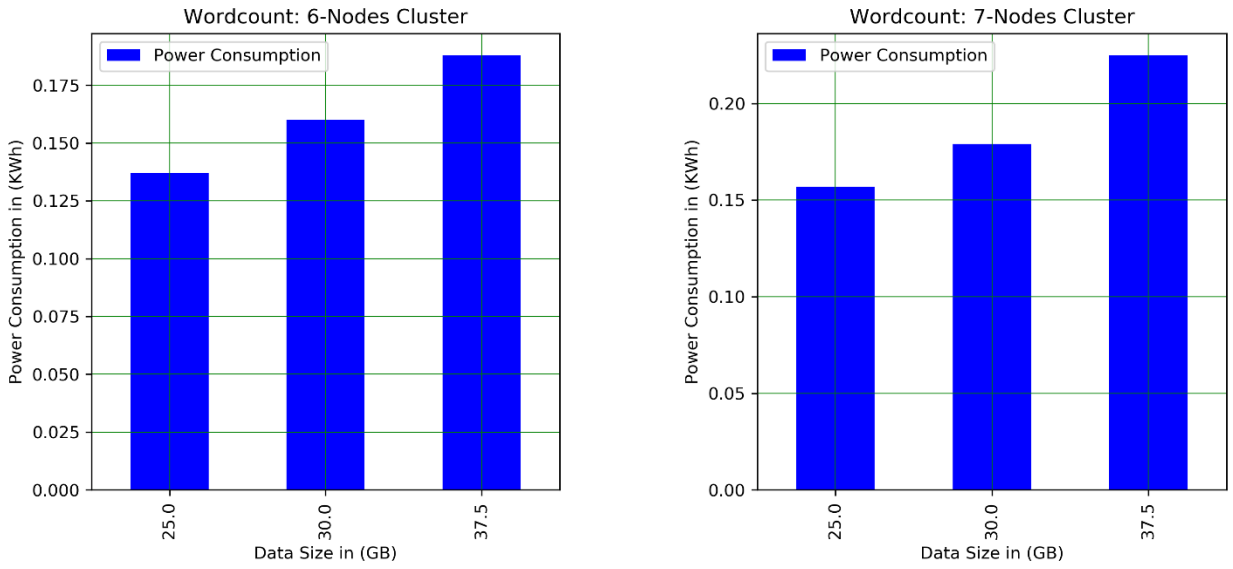
(c)


Wordcount: 3-Nodes Cluster
Power Consumption
Power Consumption in (KWh)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
Data Size in (GB)

(b)


Wordcount: 5-Nodes Cluster
Power Consumption
Power Consumption in (KWh)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
22.5
25.0
30.0
37.5
Data Size in (GB)

(d)



(e) (f)

Figure 5.1.3.1: Power Consumption vs Wordcount Workloads on Hadoop Cluster

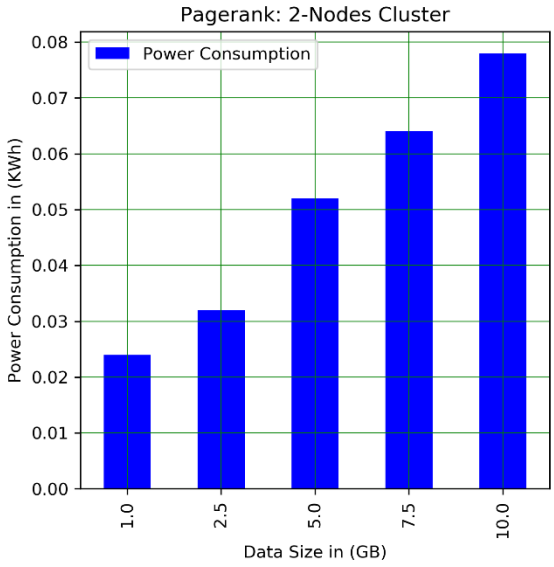
Due to some implementation and configuration limitations of the HiBench Wordcount benchmark version that we used, we had implemented a Python script that repeats a 5.5 MB text data to generate different data size, then we submitted the generated workloads to the Hadoop cluster to apply Wordcount benchmark on it at different number of the cluster's nodes.

The complete experimental results for all the Wordcount workloads and the cluster's resource utilization is shown in Appendix B.

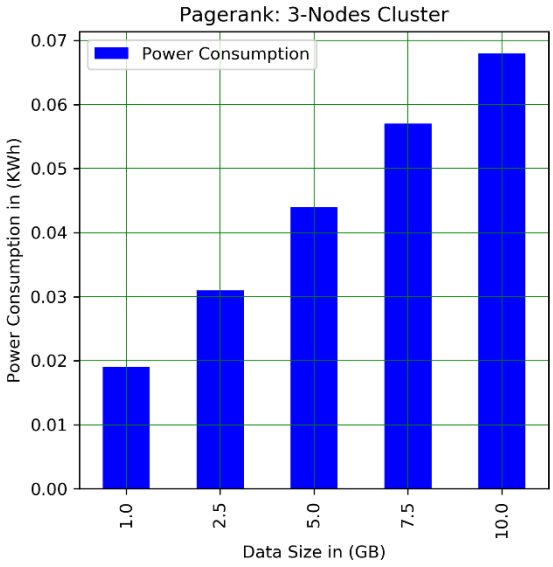
5.1.4 Pagerank

Pagerank is a web search benchmark where its operations are CPU bound. This type of operations spend the majority of its execution time waiting for the CPU

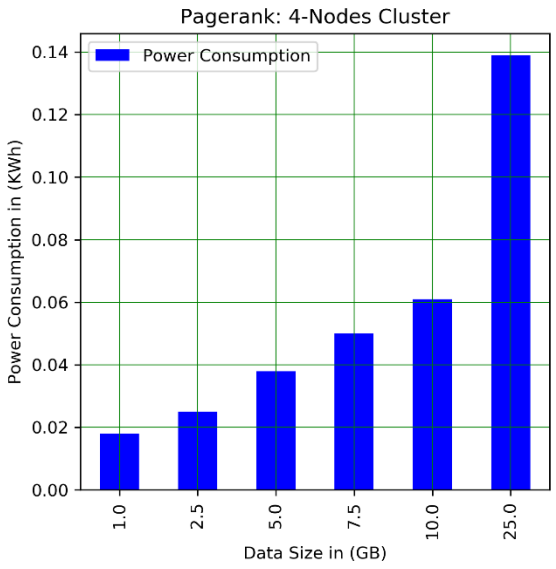
resource, which results in a high average CPU utilization. In our experiment with pagerank workloads, the highest record of the average CPU utilization was 75.98% while processing a 5 GB (5000000 pages) workload on a 3-nodes (one NameNode and two DataNodes) Hadoop cluster, during a 29 minutes period of time, as shown in the fully detailed results' tables in Appendix B. Figure 5.1.4.1 shows the power consumption of processing different Pagerank workloads at different number of Hadoop cluster's nodes.



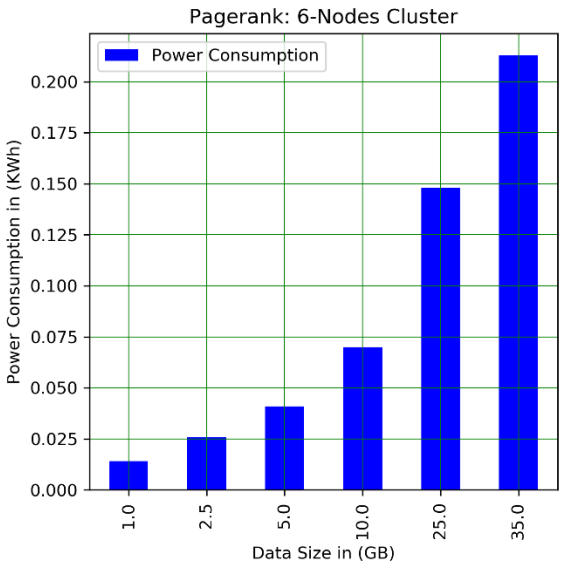
(a)



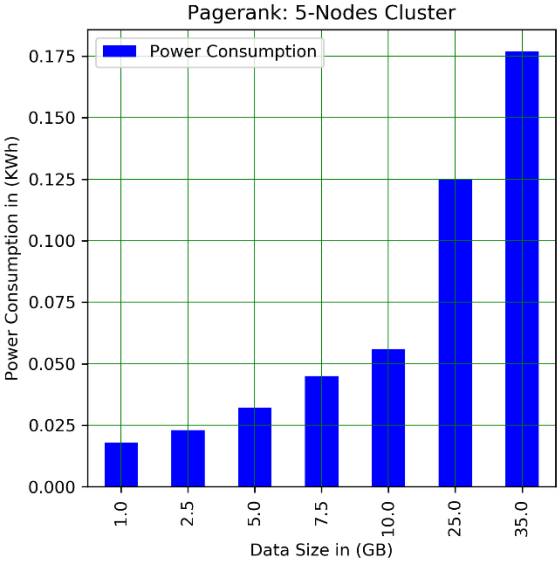
(b)



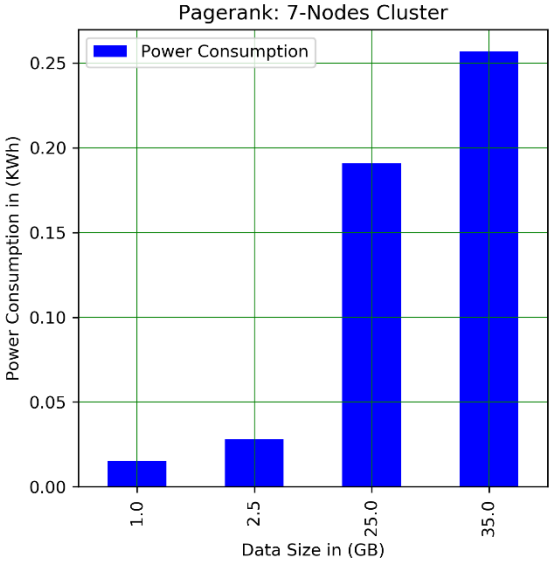
(c)



(e)



(d)



(f)

Figure 5.1.4.1: Power Consumption vs Pagerank Workloads on Hadoop Cluster

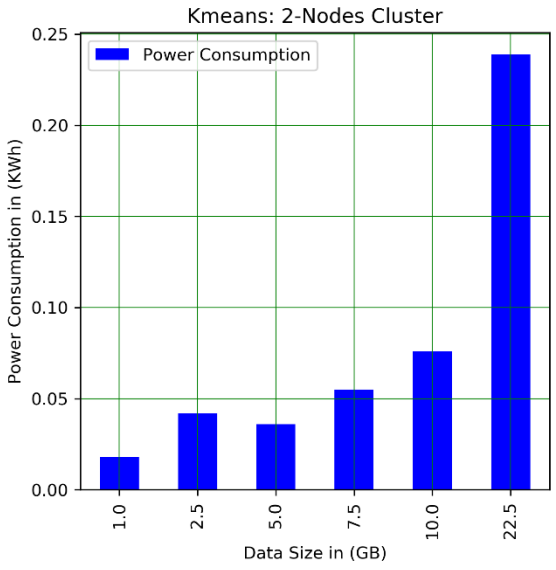
Using the HiBench Pagerank benchmark configuration we generated different workload size by using different number of pages i.e. 35000000 pages for generating a

35 GB workload, 50000000 pages for generating a 50 GB workload….etc. while we processed the generated workloads on different numbers of Hadoop cluster's nodes.

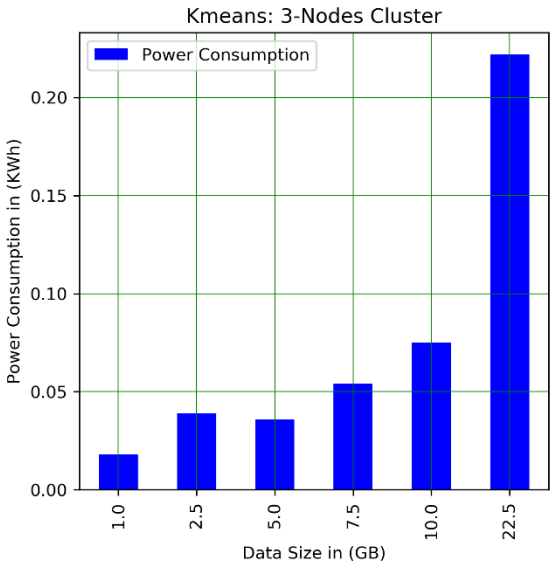
The complete experimental results for all the Pagerank workloads and the cluster's resource utilization is shown in Appendix B.

5.1.5 Kmeans

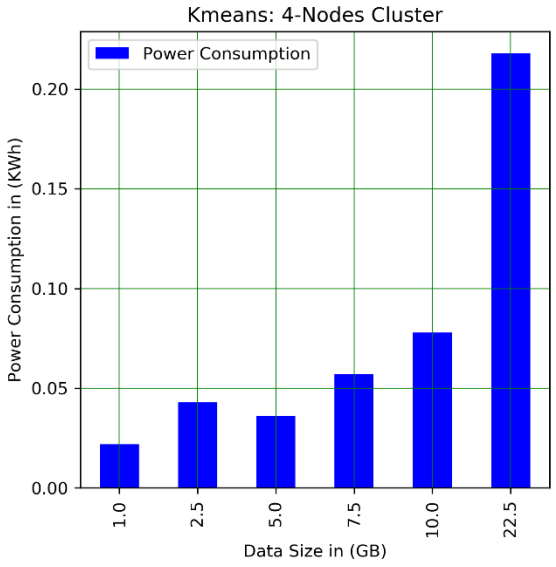
Kmeans, a machine learning benchmark which is another CPU bound benchmark. In this experiment we observed an average CPU utilization of 91.19%, while processing a 17.5 GB workload (16 clusters, number of samples 30000000, and 6000000 samples per input file) on 3-nodes (one NameNode and two dataNodes) Hadoop cluster, during a 1 hour and 38 minutes period of time. Figure 5.1.5.1 shows the power consumption of processing different Kmeans workloads on Hadoop cluster.



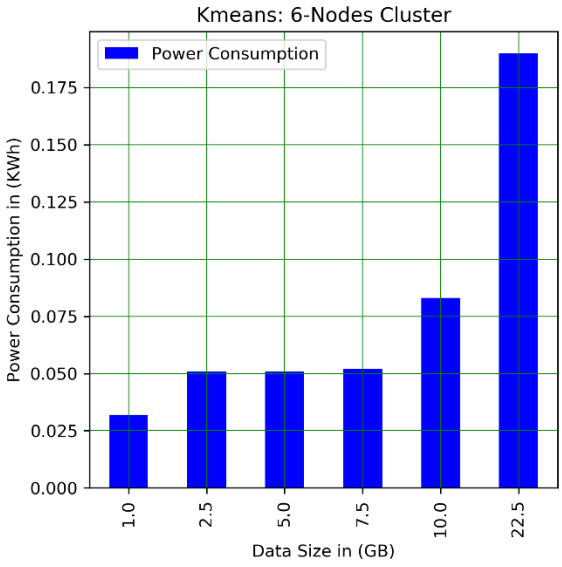
(a)



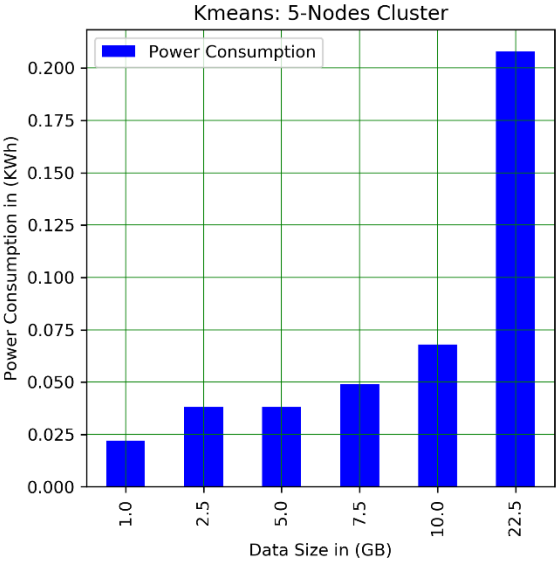
(b)



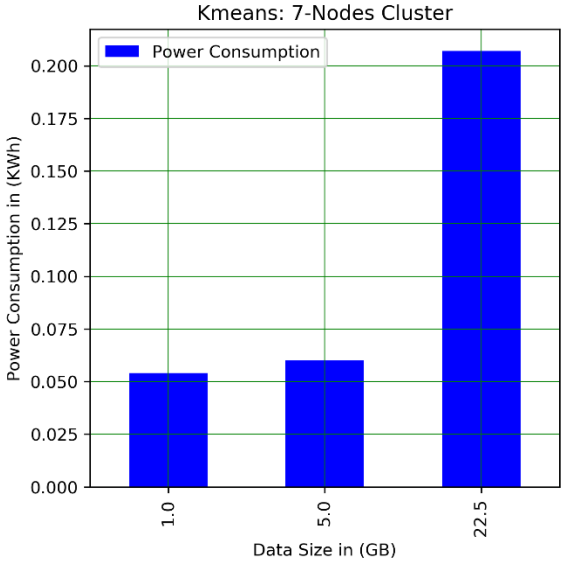
(c)



(e)



(d)



(f)

Figure 5.1.5.1: Power Consumption vs Kmeans Workloads on Hadoop Cluster

Using the HiBench Kmeans benchmark configuration we generated different workload size by changing the number of clusters at a fixed number of samples of 30000000 and a fixed number of samples per input file of 6000000. For example to

generate a 20 GB workload we set number of clusters to 18, number of samples 30000000, and the number of samples per input file to 6000000, and so on and so forth for each workload, we just change the number of clusters. Then we processed the generated workloads on different numbers of Hadoop cluster's nodes.

The complete experimental results for all the Kmeans workloads and the cluster's resource utilization are shown in Appendix B.

Note that in our study the workload processing time was not a concern, as we assumed batch job processing where the execution time is not a significant factor. Our main goal of observation was finding out the minimum power consumption for processing a certain workload (data size) on a certain number of nodes with achieving maximum resource utilization, regardless the execution time that is taken.

5.2 Phase 2: Prediction Model Implementation

The collected data from phase 1 was used to train three supervised ML models which are; logistic regression, random forest classifier with 100 estimators, and Support Vector Machine (SVM) classifier with a kernel of 5^{th} order polynomial function, wherein we compared the models according to each their prediction accuracy scores. As we mentioned before that our study goal is to find out the number of nodes that consume the minimum amount of power to execute a certain workload in Hadoop cluster. The collected data which is used to train the ML models is defined with two features; the

workload type i.e. Wordcount, Sort…etc., the workload size i.e. 65 GB, 70 GB…etc., and one label which is the hardware resources needed for consuming the minimum amount of power to process the workload i.e. as a result of our experiment, the ML model would predict cluster resources of (4 x 2.4GHz CPUs (16 cores) + 1 x 2.5GHz CPU (4 cores)), 36 GB of memory, and storage space of 1369 GB for processing a 80 GB Terasort workload. More on ML model and the experiment result analysis will be discussed in chapter 6.

5.2.1 Collecting Training Data

We categorized the hardware resources in our Hadoop cluster into 6 categories as shown in Table 5.2.1.1:

Table 5.2.1.1: Hardware Resources Categories in Hadoop Cluster

Resources Category	**CPU (GHz)**	**Memory (GB)**	**Storage (GB)**
1	2 x 2.4 CPUs (8 cores)	16	456
2	3 x 2.4 CPUs (12 cores)	20	684
3	4 x 2.4 CPUs (16 cores)	28	912
4	4 x 2.4 CPUs (16 cores) + 1 x 2.5 CPU (4 cores)	36	1369
5	4 x 2.4 CPUs (16 cores) + 1 x 2.5 CPU (4 cores) + 2.3 CPU (4 cores)	52	1837
6	4 x 2.4 CPUs (16 cores) + 1 x 2.5 CPU (4 cores) + 2 x 2.3 CPUs (8 cores)	68	2305

Based on our experimental results from phase 1 and our study goal, we collected 5 datasets, each dataset represents the experimental results of one of the five benchmarks Terasort, Sort, Wordcount, Pagerank and Kmeans. The 5 datasets include

124 data record, each data record has two features (Operation type, Data size) and a label (Resource category).

Unlike the traditional Hadoop cluster benchmarking studies, our experiment approach is to process different workloads (within different data size ranges) on different number of Hadoop cluster nodes (2-nodes to 7-nodes), measure the power consumption accompanied with each workload execution. Based on power consumption observations we were able to figure out an optimal number of Hadoop cluster nodes which consume the minimum amount of power to complete a certain workload job execution.

Considering the aforementioned, we did not have to process and observe the same workloads within a certain data size range on each of the Hadoop cluster's nodes category (Table 5.2.1.1 shows the Hadoop cluster nodes categories). Hence, once we observe that the power consumption for processing workloads in a certain data size range on a certain cluster's nodes category, starts to show increase in power consumption, than the power consumption of processing the same workloads or a few of it within the same data size range on a lower cluster's nodes category. Then we infer that we do not have to observe processing any workload within this data size range on a higher cluster's nodes category, because the result would be more increase of the power consumption for any workload within this data size range, which is unnecessary for our experimental objectives. Therefore, we can notice in Figure 5.2.1.1 that the

workload experimental frequency is not uniformly distributed on the workload sizes in our study, regardless of the operation type.

Let's take the Sort workloads as an example to explain our experiment approach, when we look at Appendix B the Sort Workload Characterization section, we can notice that the power consumption for processing workloads in the data size range from 25 GB to 65 GB on 2-nodes (1 NameNode and 1 DataNode) Hadoop cluster is lower than processing the same workloads on 3-nodes cluster. The power consumption result of processing the workloads on 2-nodes cluster was then suffices to infer that there is no need for us to process the entire workloads range from 25 GB to 65 GB on 4-nodes cluster, since the power consumption would increase, as we can see the power consumption of processing 40 GB and 65 GB workloads on the 4-nodes cluster. Furthermore, we can infer that the workloads within data size range below 25 GB will consume lower power when it is processed on 2-nodes cluster. Figure 5.2.1.2 depicts the difference in power consumption of processing workloads in the data size range from 25 GB to 65 GB on 2-nodes and 3-nodes Hadoop cluster as we explained before. The same approach applied on workloads from 67.5 GB to 72.5 GB. For example, the power consumption of processing 70 GB on 5-nodes cluster is lower than the power consumption of processing 70 GB on 4-nodes cluster, and is lower than the power consumption of processing 72.5 GB on 3-nodes cluster, the same for the workload 72.5 GB, therefore, we infer that an energy-aware Hadoop framework would process the

workloads from 67.5 to 72.5 GB on 5-nodes Hadoop cluster, and no need of processing the workloads 67.5 GB and 70 GB on 3-nodes cluster, and as we mentioned before that 2-nodes Hadoop cluster failed to process any workload that is above 65 GB due to the storage capacity limitations. Figure 5.2.1.3 shows the power consumption comparisons of processing both workloads 70 GB and 72.5 GB on 4-nodes and 5-nodes Hadoop cluster. In addition, the power consumption of processing workload of 72.5 GB on 6-nodes cluster was much higher than processing the same workload on 3, 4, 5, and 6-nodes cluster, hence, it was unnecessary to experiment processing other workloads on the 6-nodes and 7-nodes cluster. Figure 5.2.1.4 shows the power consumption of processing a 72.5 GB workload on 3, 4, 5, and 6-nodes cluster.

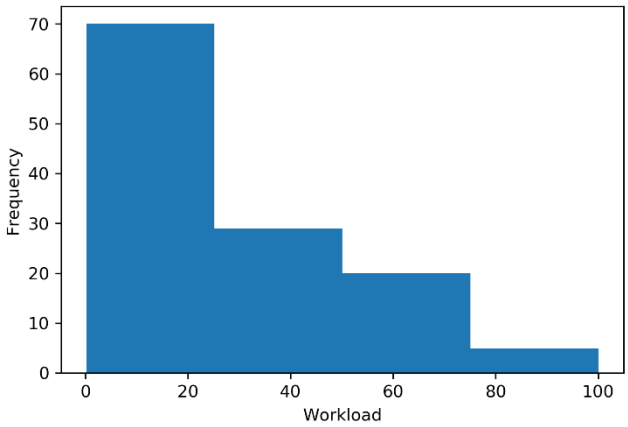


Figure 5.2.1.1: Workload Size Frequency in the Study

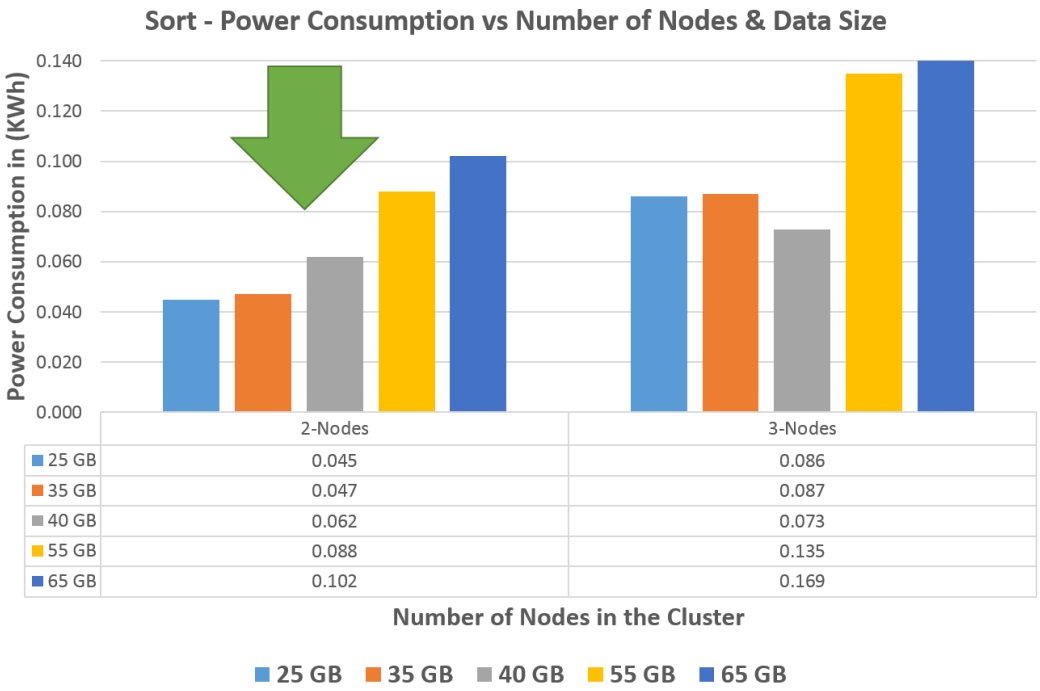


Figure 5.2.1.2: Processing 25 GB – 65 GB on 2 & 3-Nodes Hadoop Cluster

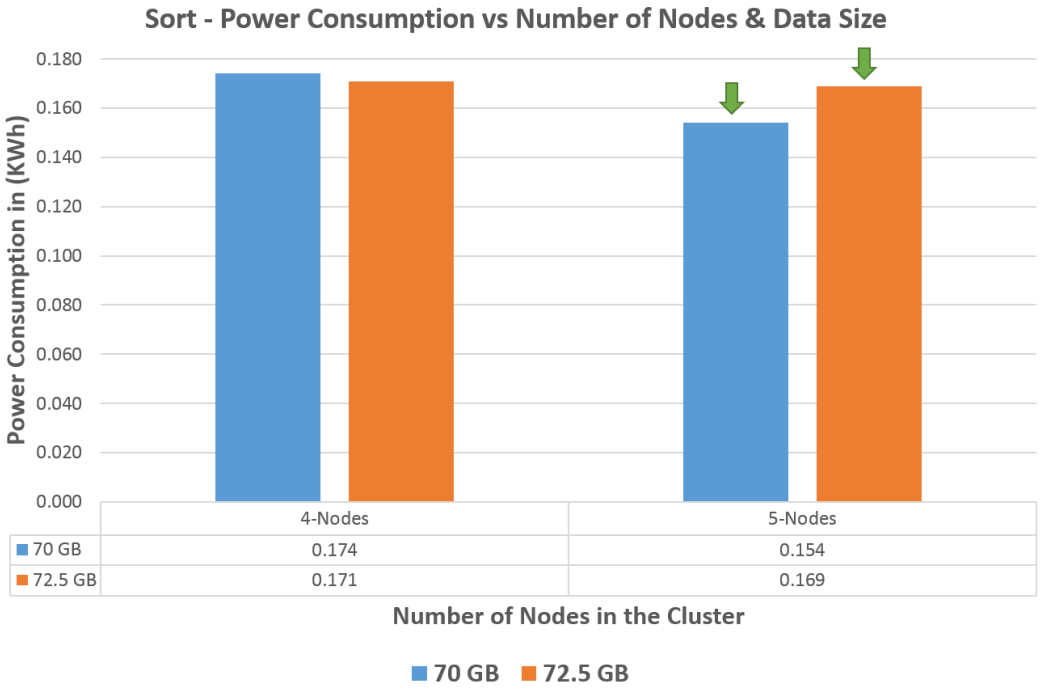


Figure 5.2.1.3: Processing 70 GB & 72.5 GB on 4 & 5-Nodes Hadoop Cluster

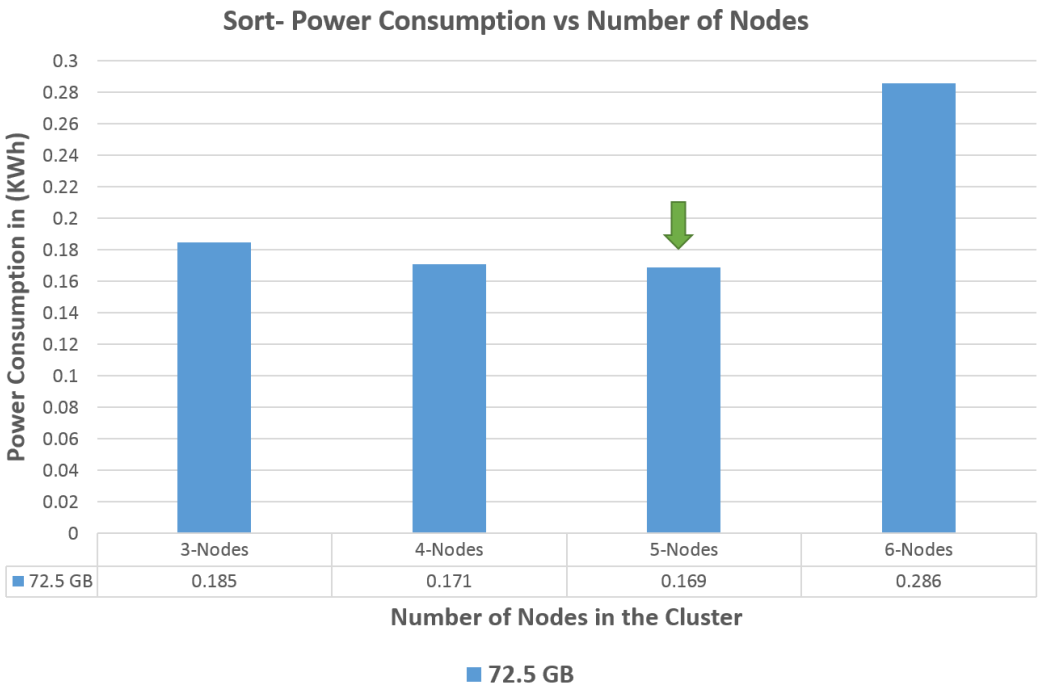


Figure 5.2.1.4: Processing 72.5 GB on 3, 4, 5, & 6-Nodes Hadoop Cluster

The same experimental approach was applied on the other four benchmarks (Terasort, Wordcount, Pagerank, and Kmeans) in order to collect the dataset which was used later to train the ML model. A sample of the collected dataset during the experiment is shown in Table 5.2.1.2 below:

Table 5.2.1.2: A Sample of the Training Dataset

Workload Type	Data Size (GB)	Resource Category
Kmeans	0.75	1
Terasort	0.5	1
Terasort	67.5	4
Terasort	57.5	1
Terasort	60	1
Pagerank	35	4
Terasort	55	1
Pagerank	2.5	4

Sort	1	1
Terasort	85	4
Sort	72.5	4
wordcount	12.5	1
Sort	50	1
wordcount	37.5	3
Sort	2.5	1
wordcount	30	2
Terasort	50	1
Terasort	1	1
Terasort	80	4
Pagerank	0.5	5
Kmeans	0.75	1
Terasort	0.5	1
Terasort	67.5	4
Terasort	57.5	1
Terasort	60	1

Figure 5.2.1.5 shows a scattering plot for the workload data sizes distribution against the Hadoop cluster hardware categories, regardless of the workload type. Figure 5.2.1.6 shows a descriptive statistics of the dataset, wherein the second column from the left concludes statistics about the [Data Size (GB)] column in Table 5.2.1.2, and the third column concludes statistics about the [Resource Category] column in Table 5.2.1.2.

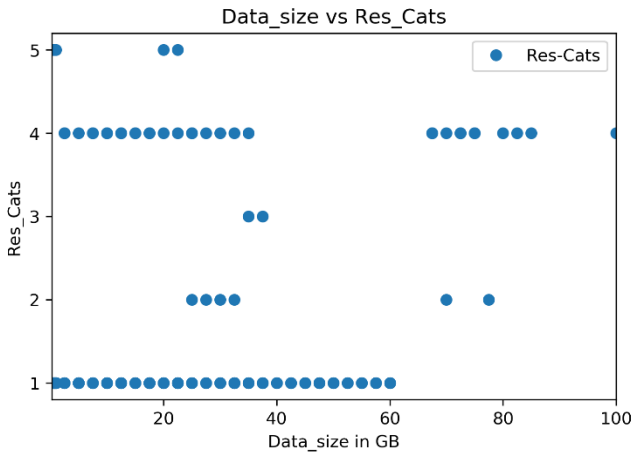


Figure 5.2.1.5: The workload data sizes distribution against the cluster resource categories

	DS-(GB)	Res-Cats
count	124.000000	124.000000
mean	26.713710	2.048387
std	23.737677	1.458671
min	0.250000	1.000000
25%	7.500000	1.000000
50%	22.500000	1.000000
75%	38.125000	4.000000
max	100.000000	5.000000

Figure 5.2.1.6: Descriptive Statistics of the Dataset

To illustrate the approach of our energy-aware Hadoop cluster, when we look at Table 5.2.1.1, Table 5.2.1.2 and Appendix B under the (Wordcount Workload Characteization) section, for the cluster to process a 30 GB Wordcount workload, the framework will decommission (disconnect a node from the cluster and do not process any tasks on it) or power off 4-DataNodes from the 7-nodes cluster, and process the 30 GB Wordcount workload on a 3-nodes (1 NameNode and 2 DataNodes) Hadoop

cluster, wherein the process will consume 0.086 KWh of power, which represents a reduction in the power consumption by at most 51.96% than processing this Wordcount workload on the entire 7-nodes Hadoop cluster in the experiment, and reducing power consumption by at least 3.37% than processing this workload on 4-nodes Hadoop cluster. More on the result analysis will be discussed in Chapter 6.

5.2.2 Data Preprocessing and Model Training

In order to obtain high accuracy of a ML model, dataset in the study has to get through a pipeline of preparation processes, starting from loading the dataset until training the model. The steps of preparation the dataset are shown below:

- *Loading the dataset:* We loaded our experimental data from "Data.xlsx" spreadsheet using Python 3 *pandas* library into jupyter Notebook.
- *Extracting the features and the target values:* We have two features in the dataset; 'Workload Type' and 'Data Size (GB)', we loaded their values in a variable and we loaded the target 'Resource Category' values into another variable for further data processing.
- *One-hot encode data:* In order to use the categorical feature 'Workload Type' for training the ML model, we encoded (binary variables representation) this feature using the get_dummies() method from pandas. Figure 5.2.2.1 below shows the first 5 rows in the dataset after the one-hot data encoding.

	DS-(GB)	Res-Cats	Oper_Kmeans	Oper_Pagerank	Oper_Sort	Oper_Terasort	Oper_wordcount
0	0.75	1	1	0	0	0	0
1	0.50	1	0	0	0	1	0
2	67.50	4	0	0	0	1	0
3	57.50	1	0	0	0	1	0
4	60.00	1	0	0	0	1	0

Figure 5.2.2.1: One-hot Data Encoded

- *Convert data into arrays*: We converted the features and the label data into arrays using numpy python library, to prepare for data splitting.
- *Splitting the dataset*: We split the dataset (124 data records) into 85% for training the model, and 15% of the dataset for testing the trained model on unseen data and evaluate the model accuracy.
- *Standardized scaler*: In order to ensure that there will not be feature data with high order of magnitude that will dominate the ML estimator, we rescaled the features (training and testing features) using standardized scaler, as a method to avoid the high variation in the data magnitudes.
- *ML model training*: We instantiated three ML models; logistic regression, random forest classifier, and support vector machine classifier. We fitted the three models with the same training dataset.
- *ML model prediction evaluation*: The three ML models were evaluated against the same test dataset, where we generated the confusion matrix and classification report for each model, for us to compare and decide which one is more suitable

for our energy-aware Hadoop framework. The SVM and the random forest classifiers performed the same with a higher degree of accuracy than the logistic regression model. We will discuss more about the ML models testing results in chapter 6.

Chapter 6. Results Analysis and Conclusion

6.1 Workload Profiling Analysis

Our experiment observations show that as we increased the number of the cluster's nodes i.e. to 6-nodes or 7-nodes Hadoop cluster, when processing I/O bound jobs such as Terasort and Sort; the power consumption increased significantly.

6.1.1 Terasort Workload Profiles

In this section, we are going to demonstrate the Hadoop cluster's power consumption at different Terasort workloads, and explain our approach of profiling these workloads. Figure 6.1.1.1 shows that the Terasort workloads in the range from 1 GB to 65 GB consume less power when we process them on 2-nodes cluster, based on Table 5.2.1.1, 2-nodes cluster is a resource category 1, therefore, we state that the Terasort workloads in the range from 1 GB to 65 GB require resource category 1. In addition, Figure 6.1.1.2 shows that the power consumption of processing a 65 GB Terasort workload on 2-nodes cluster (category 1) is lesser than processing the same workload on the other cluster resource categories (refer to Table 5.2.1.1). Similarly, Figure 6.1.1.3 shows that the profile of the 75 GB and 80 GB Terasort workloads would be category 4, processing 75 GB & 80 GB Terasort workloads consume less power on 5-nodes cluster than processing them on the other cluster resource categories. We can find the rest of the Tearsort workload profiles in Appendix B.

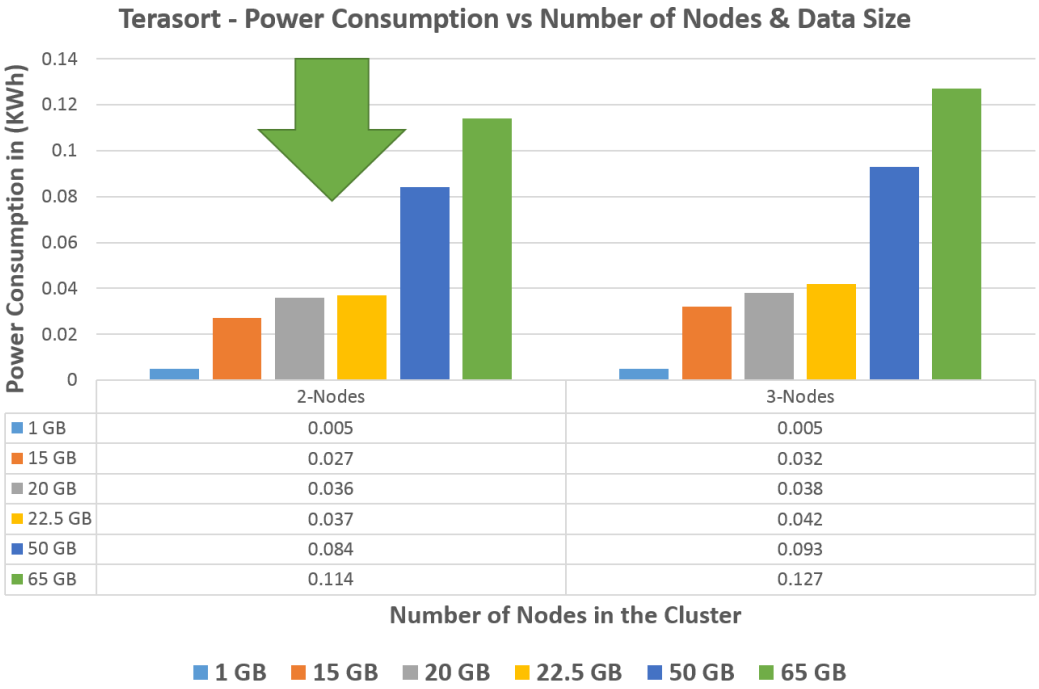


Figure 6.1.1.1: Resources required to Process 1 GB – 65 GB Terasort Workload

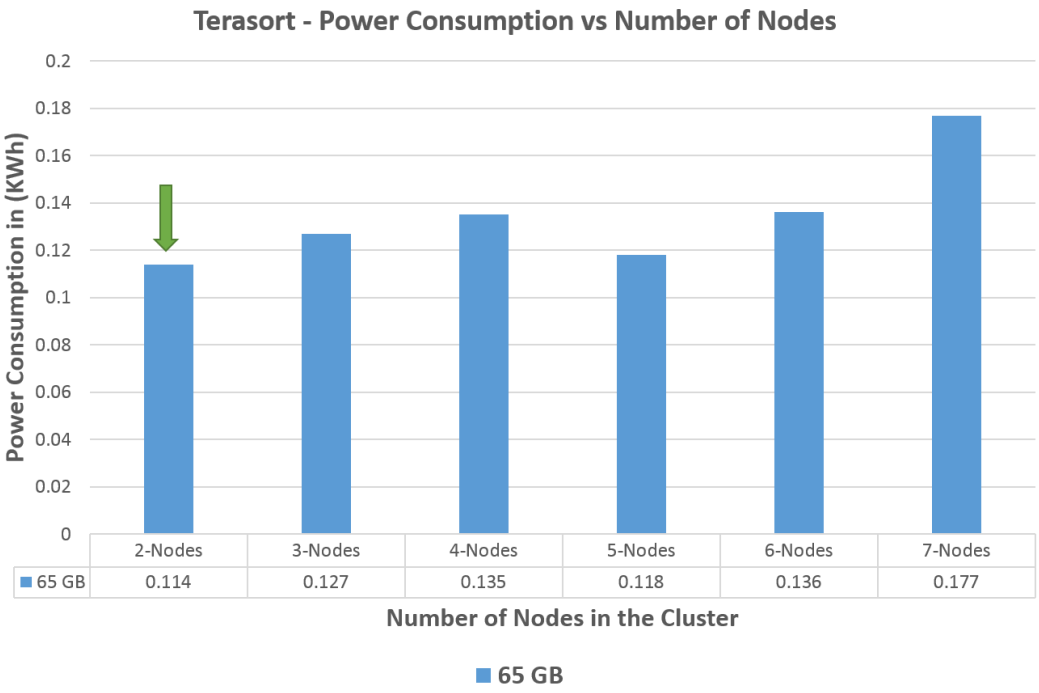


Figure 6.1.1.2: Resources required to Process 65 GB Terasort workload in our Experiment

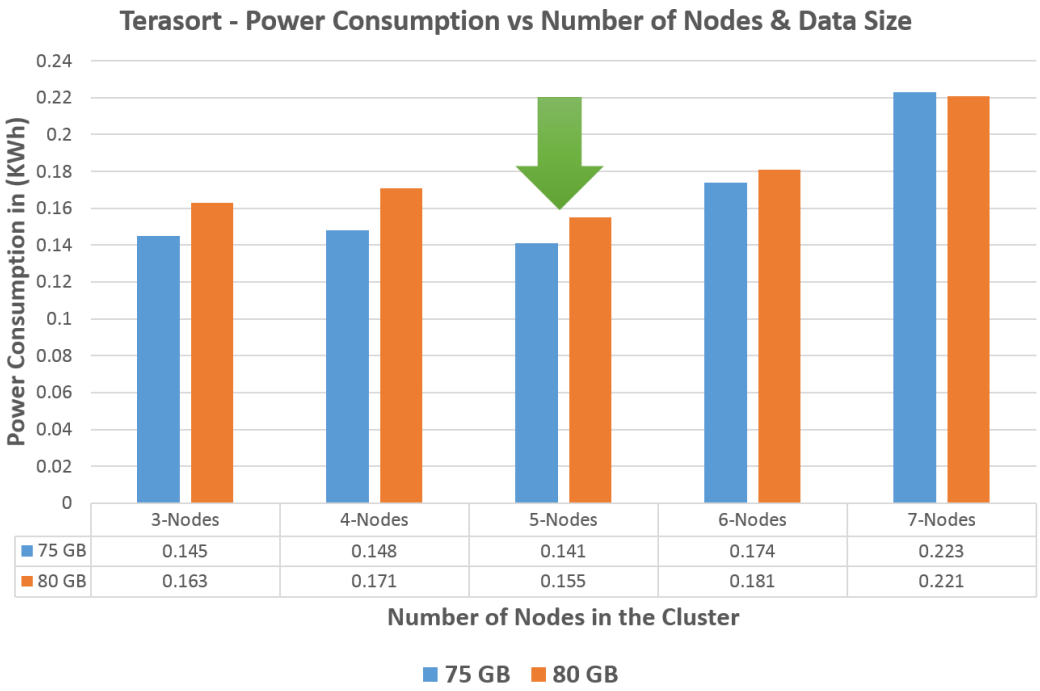


Figure 6.1.1.3: Resources required to Process 75 GB & 80 GB Terasort Workload

6.1.2 Sort Workload Profiles

In this section, we are going to demonstrate the Hadoop cluster's power consumption at different Sort workloads. In chapter 5, Figure 5.2.1.2 shows that the 25 GB – 65 GB Sort workload's profile is the category 1 cluster resource (refer to Table 5.2.1.1). Figure 6.1.2.1 below shows that the power consumption of processing 40 GB & 65 GB Sort workloads on 2-nodes cluster (category 1) is lesser than processing the same workloads on the other cluster resource categories. In chapter 5, Figure 5.2.1.3 shows that the profile of 70 GB & 72.5 GB Sort workloads is category 4 which is 5-nodes cluster. We can find the rest of the Sort workload profiles in Appendix B.

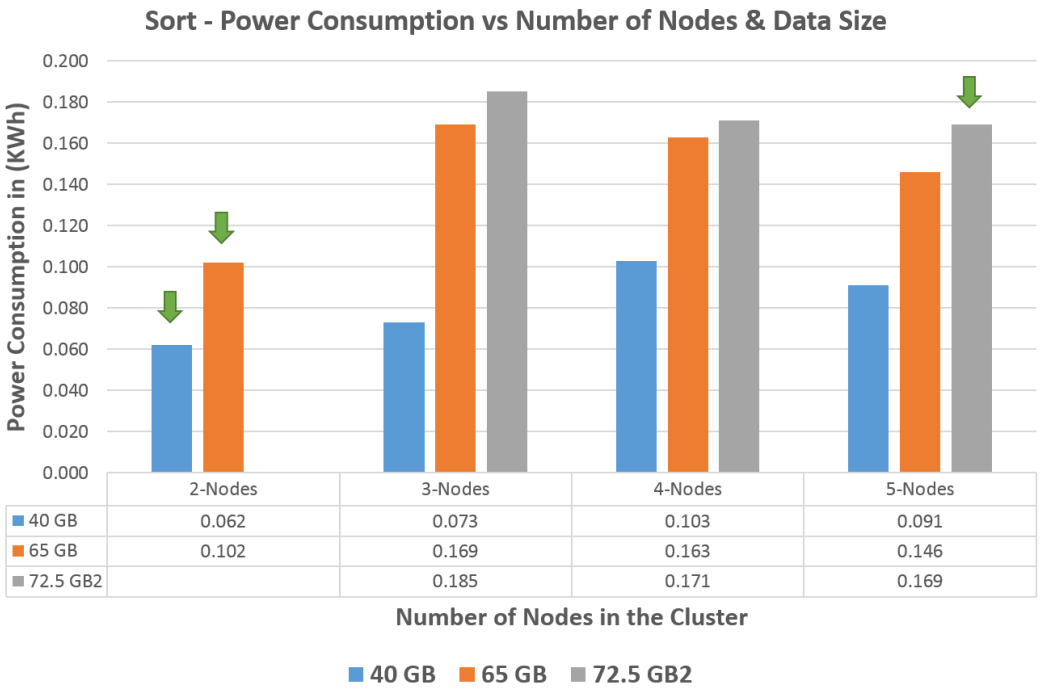


Figure 6.1.2.1: Resources required to Process 40 GB – 72.5 GB Sort workload

6.1.3 Wordcount Workload Profiles

In this section, we are going to demonstrate the Hadoop cluster's power consumption at different Wordcount workloads. Figure 6.1.3.1 shows that the Wordcount workloads 20 GB & 22.5 GB consume less power when we process them on 2-nodes cluster which is a resource category 1 (refer to Table 5.2.1.1). The Wordcount workloads 25 GB, 27.5 GB, and 30 GB consume less power when we process them on 3-nodes cluster which means that they have a category 2 resource profile. Figure 6.1.3.2 shows that processing a 37.5 GB Wordcount workload on 4-nodes cluster consume less power, therefore, the Wordcount workload 37.5 GB has a category 3 resource profile. We can find the rest of the Wordcount workload profiles in Appendix B.

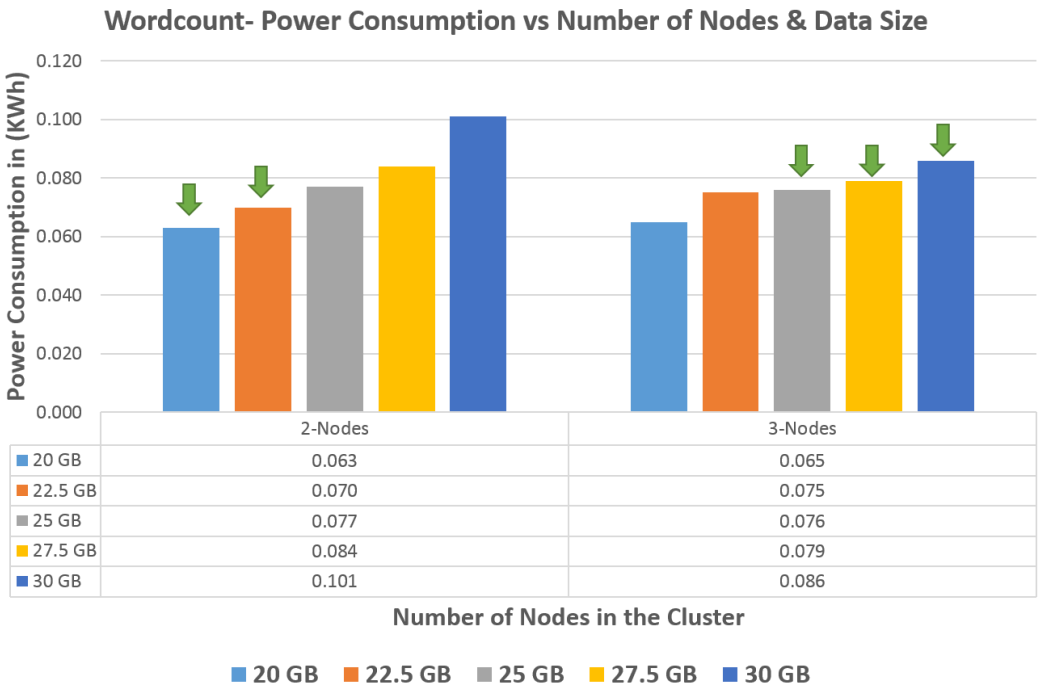


Figure 6.1.3.1: Resources required to Process 20 GB – 30 GB Wordcount workload

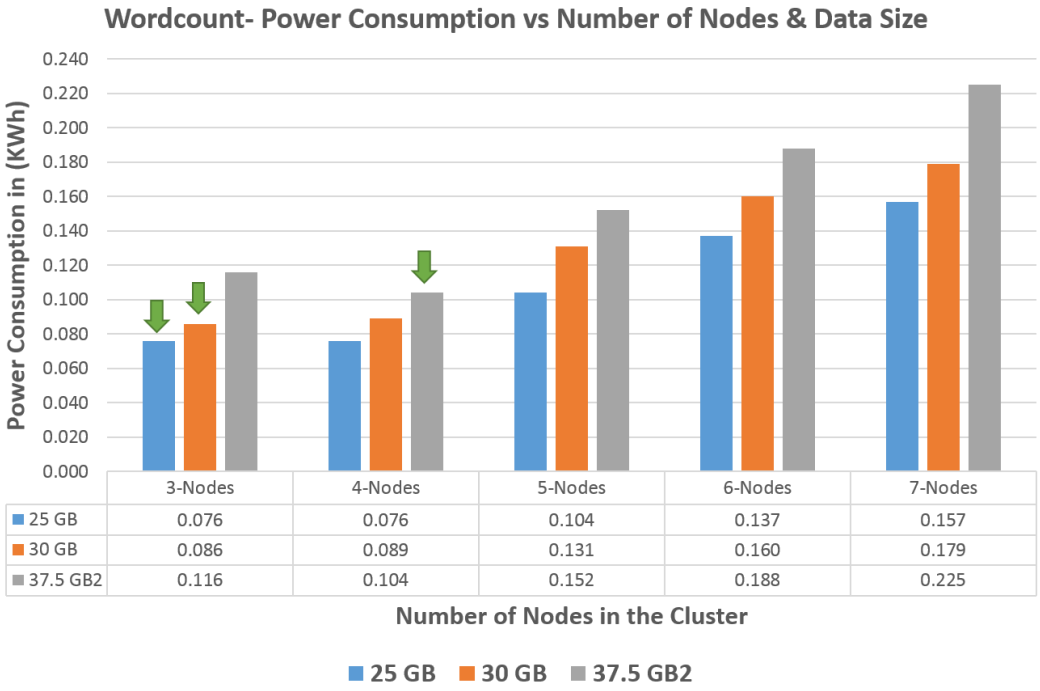


Figure 6.1.3.2: Resources required to Process 25 GB – 37.5 GB Wordcount workload

6.1.4 Pagerank Workload Profiles

In this section, we are going to demonstrate the Hadoop cluster's power consumption at different Pagerank workloads. Figure 6.1.4.1 shows that 1 GB Pagerank workload consumes less power it is processed on 6-nodes cluster which means that 1 GB Pagerank workload has a category 5 resource profile (refer to Table 5.2.1.1), similarly, the 2.5 Pagerank workload has a category 4 resource profile. Figure 6.1.4.2

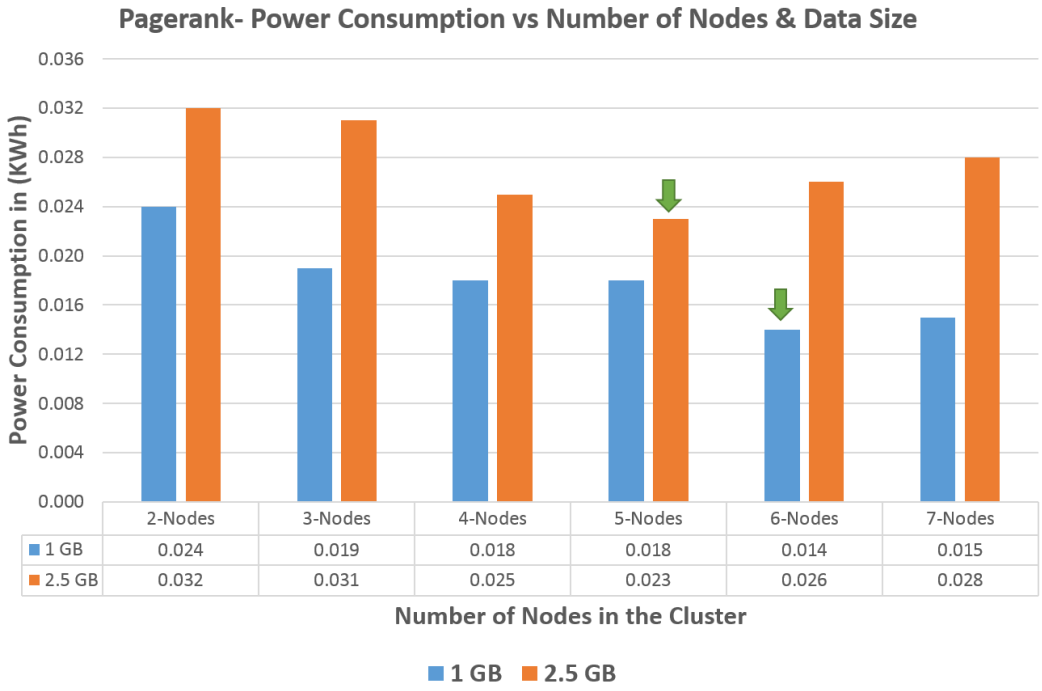


Figure 6.1.4.1: Resources required to Process 1 GB & 2.5 GB Pagerank workload

Figure 6.1.4.2 and Figure 6.1.4.3 show that 5 GB, 10 GB, 25 GB, and 35 GB consume less power when they are processed on a 5-nodes cluster, which means that they have a category 4 resource profile. We can find the rest of the Pagerank workload profiles in Appendix B.

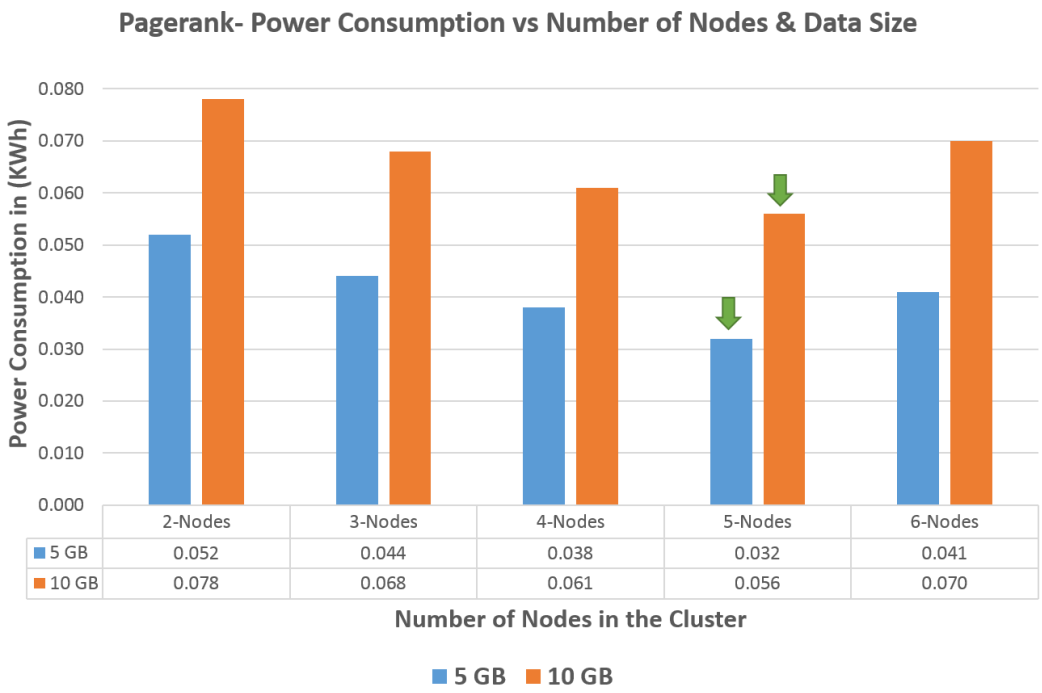


Figure 6.1.4.2: Resources required to Process 5 GB & 10 GB Pagerank workload

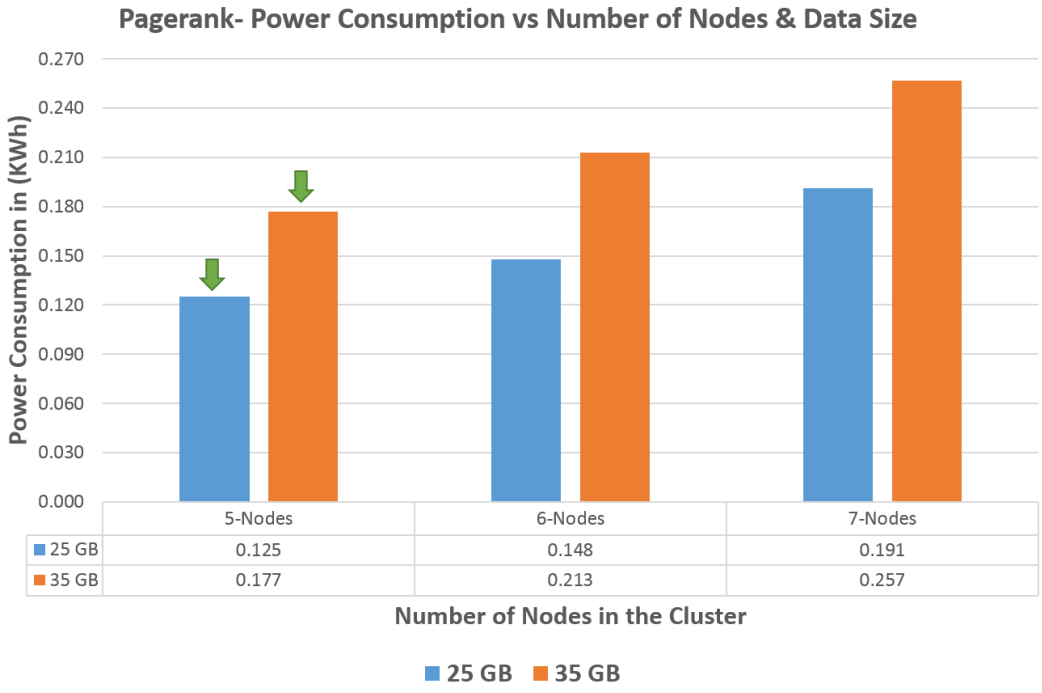


Figure 6.1.4.3: Resources required to Process 25 GB & 35 GB Pagerank workload

6.1.5 Kmeans Workload Profiles

In this section, we are going to demonstrate the Hadoop cluster's power consumption at different Kmeans workloads. Figure 6.1.5.1 shows that 1 GB and 5 GB Kmeans workloads consume less power when they are processed on 2-nodes cluster, which means that they have a category 1 resource profile (refer to Table 5.2.1.1), also, the Kmeans workload of 22.5 GB consume less power when it is processed on 6-nodes cluster, which means that 22.5 GB Kmeans workload has a category 5 resource profile. We can find the rest of the Kmeans workload profiles in Appendix B.

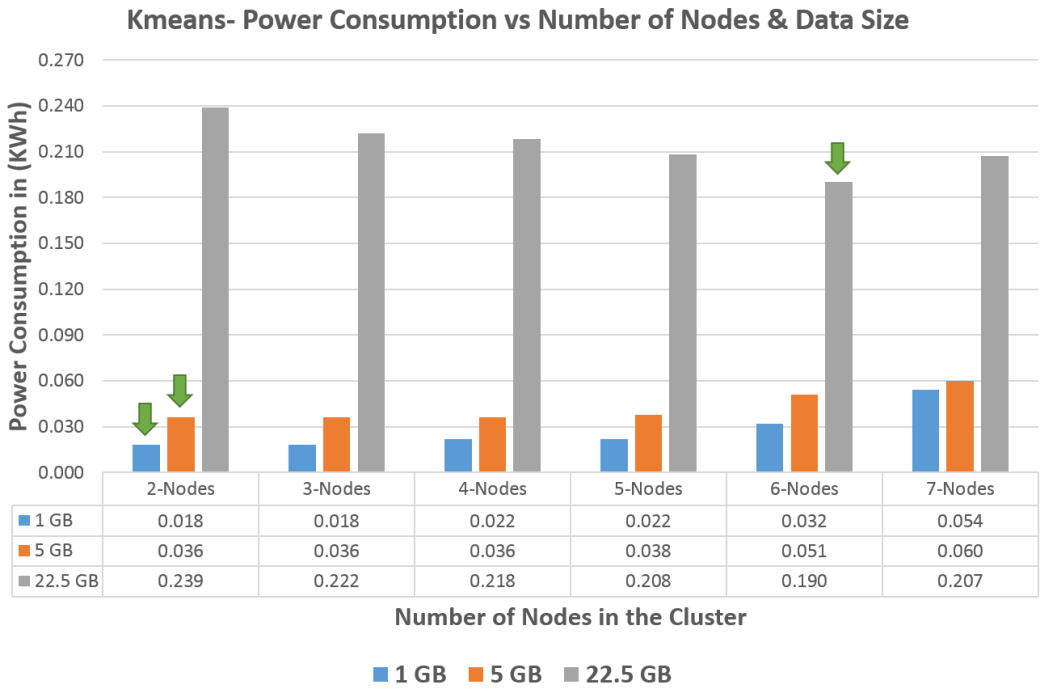


Figure 6.1.5.1: Resources required to Process 1 GB – 22.5 GB Kmeans workload

6.1.6 Key Observations

In our study, it has been proven that while processing a MapReduce job in Hadoop cluster, despite of the workload size and type, the NameNode consumes the lowest amount of power in the cluster to complete the job, see our experiment observations in Appendix B.

Scaling up Hadoop cluster size (commissioning more DataNodes to the cluster) to process a MapReduce job, does not always lead to an increase of the power consumption i.e. in our experimental setup, processing a 1 GB Pagerank workload on 6-nodes Hadoop cluster (1 NameNode and 5 DataNodes) would consume a power of 0.014 KWh, which represents approximately a 41.67% reduction in the cluster power consumption than processing the same workload on only 2-nodes (1 NameNode and 1 DataNode) of the cluster, as it would consume a power of 0.024 KWh.

6.2 Machine Learning Models Evaluation

As we mentioned in chapter 5, among the three ML models that we compared, the logistic regression was the model that had the lowest prediction accuracy score of 89.47% on the testing data. Figure 6.2.1 shows the confusion matrix which describes the performance of the logistic regression classifier model on the testing dataset. The same testing dataset which is 19 data samples (15% split from the experimental collected data as we mentioned in chapter 5) was used to test the three ML models.

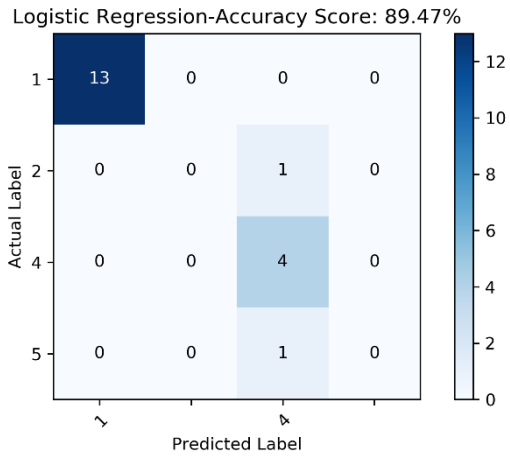


Figure 6.2.1: Confusion Matrix of the Logistic Regression Model

The left vertical axis represents the true labels in the testing dataset and the horizontal axis represents the predicted labels. As we can see in Figure 6.2.1 there are 2 samples of the testing data were incorrectly predicted which are: the actual label 5 was predicted as label 4 once, and the actual label 2 was predicted as label 4 once, whereas the actual label 1 was correctly predicted, 13 times, and the actual label 4 was correctly predicted, 4 times. Hence, the model accuracy score on the testing data is calculated as 17 correctly predicted labels out of 19 data samples equals to 89.47%.

From the confusion matrix in Figure 6.2.1, the actual label 4 was predicted 6 times, 2 of these predictions were false and 4 predictions were true, therefore, the prediction precision of label 4 is $\frac{4}{6} \times 100 \approx 0.67\%$ as shown in the classification report (precision, recall, and f1-score) of the logistic regression model in Figure 6.2.2 below.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	13
2	0.00	0.00	0.00	1
4	0.67	1.00	0.80	4
5	0.00	0.00	0.00	1
accuracy			0.89	19
macro avg	0.42	0.50	0.45	19
weighted avg	0.82	0.89	0.85	19

Figure 6.2.2: Classification Report of the Logistic Regression Model

Another way of visualizing the model's performance is shown in Figure 6.2.3, as it depicts the actual testing labels vs the predicted labels.

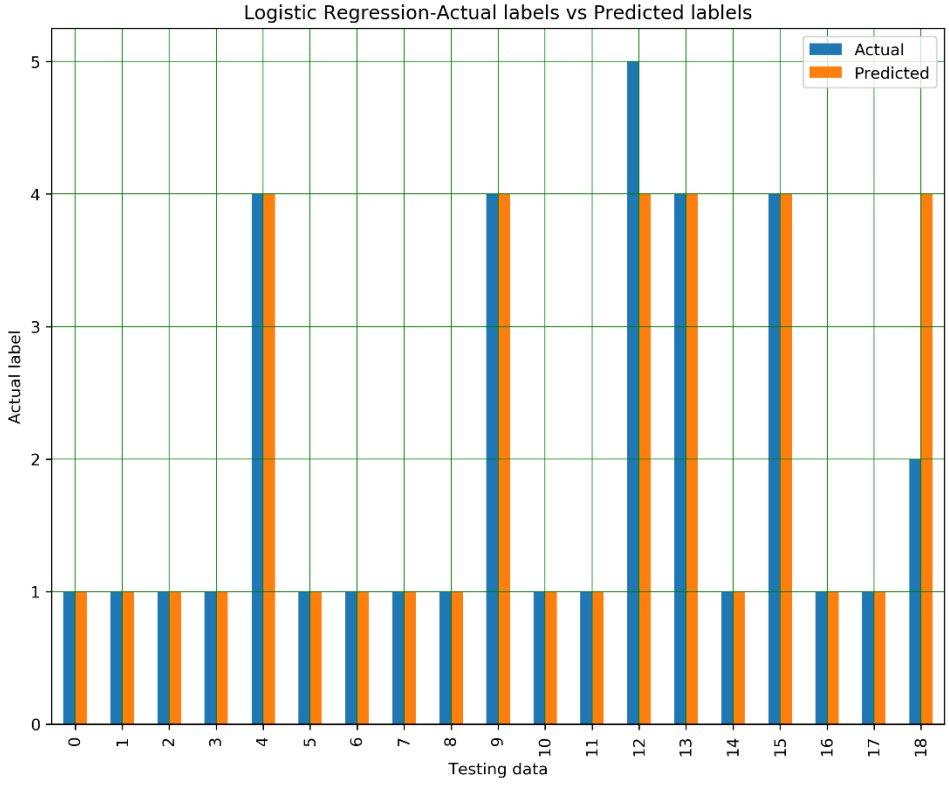


Figure 6.2.3: Predicting the Testing Data in the Logistic Regression Model

The SVM classifier performed exactly the same as the random forest classifier on the testing data by tuning its polynomial kernel function's degree, wherein both classifier's

accuracy scored is 94.74%. The random forest classifier' accuracy score 94.74% was obtained by our initial random forest classifier which was with 100 estimators. We experienced changing the number of the estimators by step of 100 to 1000 estimators, however, the random forest classifier accuracy did not change, as shown in Figure 6.2.4 below.

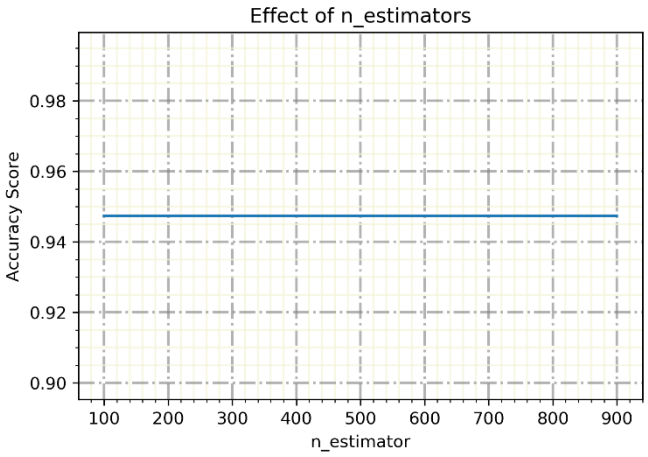


Figure 6.2.4: Random Forest Classifier Accuracy Score with Different Estimators Value

On the other hand, the SVM classifier accuracy score was increased linearly from 84.21% to 94.74% by changing the classifier polynomial kernel function's degree from 2 to 5 degrees, as shown in Figure 6.2.5 below.

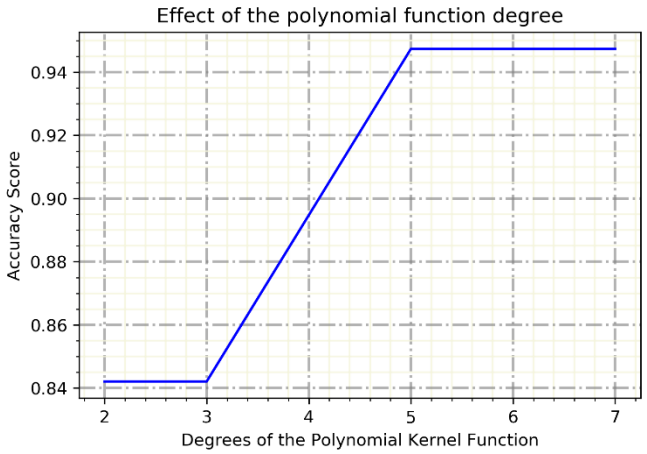


Figure 6.2.5: Effect of the Polynomial Kernel Function degrees on the SVM Classifier Acc. Score

The SVM classifier accuracy score with polynomial kernel function at degree 5 was the highest among the other kernel functions, as the accuracy score was 89.47%, 84.21%, and 78.95% with *linear* kernel function, *rbf* kernel function, and *sigmoid* kernel function, respectively as shown in Figure 6.2.6.

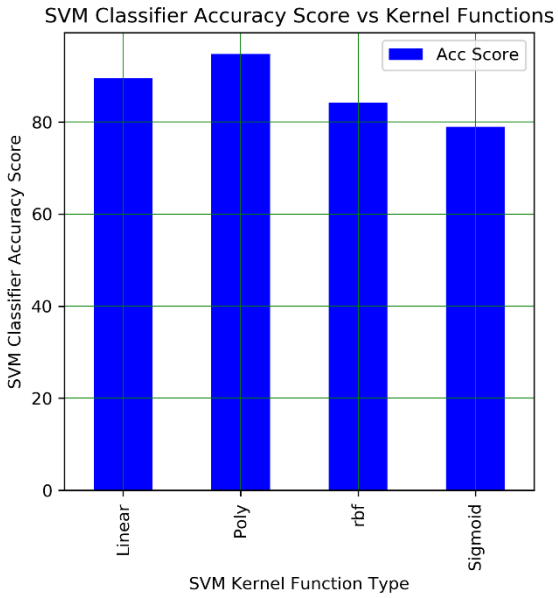


Figure 6.2.6: SVM Classifier Accuracy Score against Different Kernel Function Types

Since the random forest classifier and the SVM classifier (with polynomial kernel function at degree 5) have the same performance, then we are going to demonstrate the confusion matrix and the classification report of the random forest classifier and the same would apply to the SVM classifier performance results.

Figure 6.2.7 shows that there is 1 sample of the testing data was incorrectly predicted which is: the actual label 2 was predicted as label 4 once, whereas the actual label 1 was correctly predicted, 13 times, the actual label 4 was correctly predicted, 4

times, and the actual label 5 was correctly predicted once. Hence, the model accuracy score on the testing data is calculated as 18 correctly predicted labels out of 19 data samples equals to 94.74%.

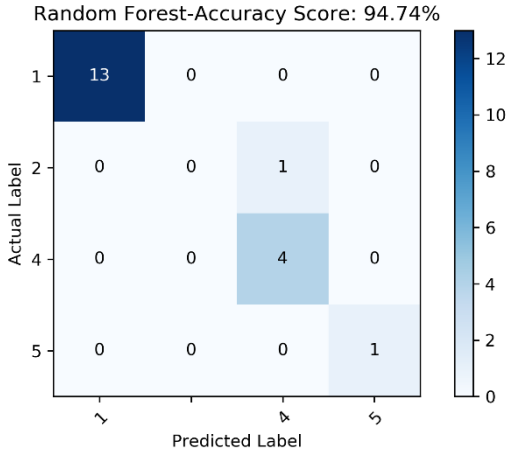


Figure 6.2.7: Confusion Matrix of the Random Forest Classifier Model

From the confusion matrix in Figure 6.2.7, the actual label 4 was predicted 5 times, 1 of these predictions was false and 4 predictions were true, therefore, the prediction precision of label 4 is $\frac{4}{5} \times 100 = 0.80\%$ as shown in the classification report (precision, recall, and f1-score) of the random forest classifier model in Figure 6.2.8 below.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	13
2	0.00	0.00	0.00	1
4	0.80	1.00	0.89	4
5	1.00	1.00	1.00	1
accuracy			0.95	19
macro avg	0.70	0.75	0.72	19
weighted avg	0.91	0.95	0.92	19

Figure 6.2.8: Classification Report of the Random Forest Classifier Model

Another way of visualizing the model's performance is shown in Figure 6.2.9, as it depicts the actual testing labels vs the predicted labels.

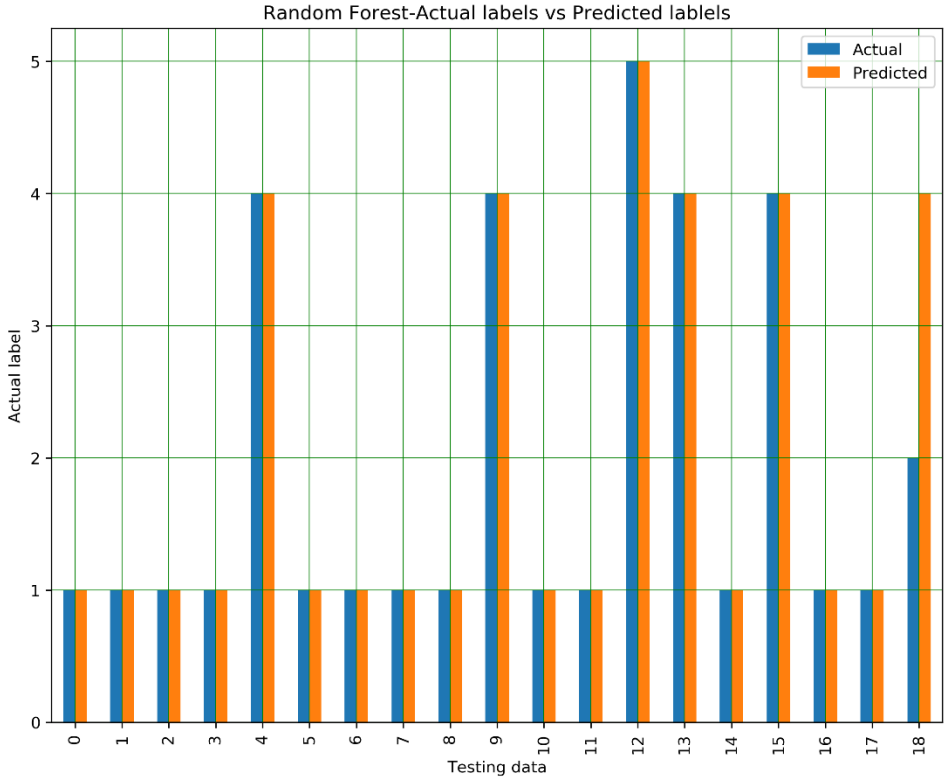


Figure 6.2.9: Predicting the Testing Data in the Random Forest Classifier Model

Since we have two ML models that perform the same on our testing dataset, we can use either model. Therefore, we decided to go further with the **Random Forest Classifier** as the prediction model in our energy-aware Hadoop cluster framework.

6.3 Data Block Replications Impact

As illustrated in Chapter 2, one of the most significant features of Hadoop HDFS is that it has a high machine failure tolerance. HDFS achieves the machine fail tolerance by splitting the input data into blocks and replicates these data blocks into the cluster's DataNodes with a replication factor i.e. 1, 3…etc. as if one machine fails or its connection with the NameNode gets broken or disrupted, the data is still accessible from the other machines.

Data replication through the network of connected Hadoop cluster nodes consumes a good amount of power during the workload processing. The replication factor in Hadoop configuration tells the HDFS how many replicas of the same data block will be placed in the cluster's nodes. In our study, and throughout the entire experiment we have set the replication factor to 3 replicas, therefore all our observations of power consumption were based on using 3 replicas.

In order to study the impact of the replication factor on the power consumption, we have changed the replication factor to 1 replica, then we tested this new replication factor on the processing of two types of workloads, Sort workload which is an I/O

bound operation, and Pagerank workload which is a CPU bound operation. We processed a 65 GB Sort workload on 2, 3, 4, and 5-nodes Hadoop cluster, and we processed a 10 GB Pagerank workload on 2, 3, 4, and 5-nodes Hadoop cluster.

Figure 6.3.1 depicts the significant drop of the power consumption while using replication factor of 1. The power consumption of processing 65 GB Sort workload on 2-nodes has been reduced by 7.84%, power consumption has been reduced by 40.24% when processing the same workload on 3-nodes, power consumption has been reduced by 33.74% when processing the same workload on 4-nodes, and power consumption has been reduced by 38.36% when processing the same workload on 5-bodes Hadoop cluster. As we can notice, in our energy-aware Hadoop framework, and if our Hadoop cluster is comprised of only 5-nodes (1 NameNode and 4 DataNodes), so, in this environment if we are using 1 replica while process a 65 GB Sort workload, then the framework will process this workload on the 5-nodes Hadoop cluster which will save at least 4.26% KWh of power, however, if we are using replication factor 3, then the framework will process the 65 GB Sort workload on 2-nodes (1 NameNode and 1 DataNode) Hadoop cluster which will save at least 30.17% KWh of power.

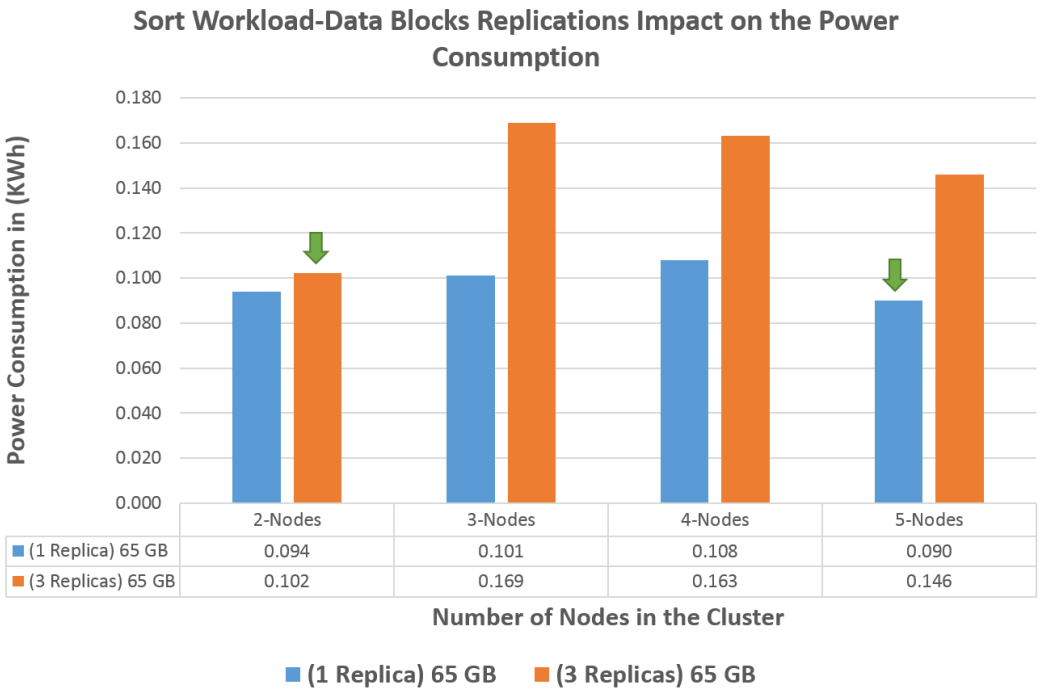


Figure 6.3.1: Sort Workload-Data Block Replications Impact on Power Consumption

Using the same assumption that our Hadoop cluster is only 5-nodes (1 NameNode and 4 DataNodes) Figure 6.3.2 shows that in our energy-aware Hadoop framework, 10 GB Pagerank workload will be processed on 5-nodes in both cases of the replication factors, as we will save at least 11.86% KWh of power while using replication factor 1, and we will save at least 8.20% KWh of power while using replication factor 3. We can notice that power consumption has been reduced by 5.19% when processing 10 GB Pagerank workload on 2-nodes Hadoop cluster with replication factor 1, the power consumption has been reduced by 7.35% when processing the same workload on 3-nodes with replication factor 1, the power consumption has been reduced by 3.28% when processing the same workload on 4-nodes with replication

factor 1, and the power consumption has been reduced by 7.14% when processing the same workload on 5-nodes with replication factor 1.

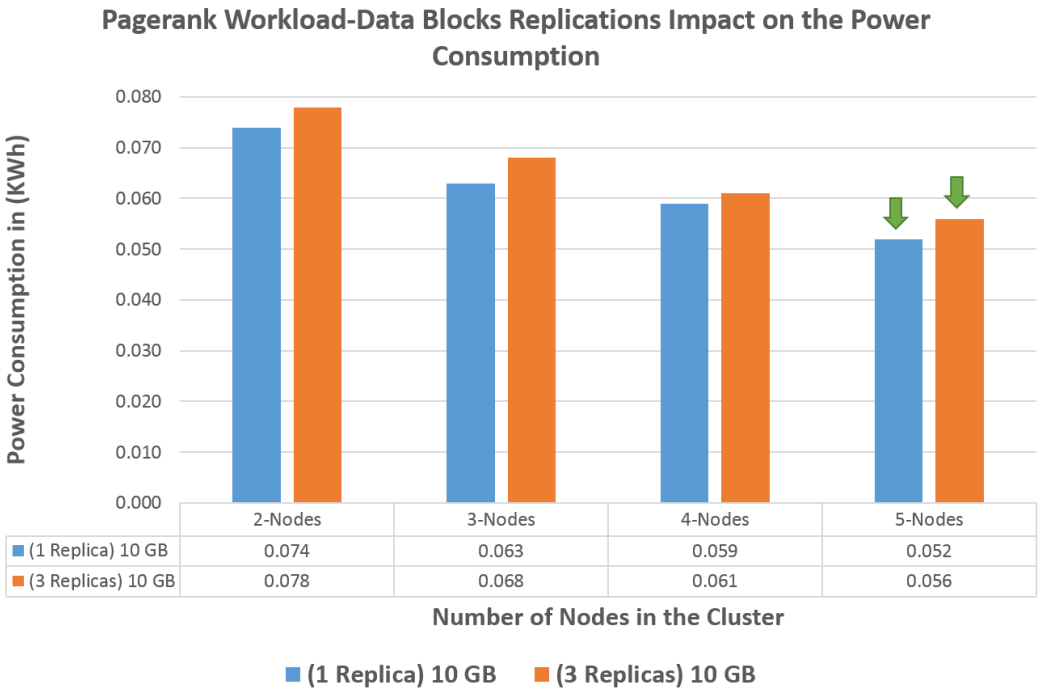


Figure 6.3.2: Pagerank Workload-Data Block Replications Impact on Power Consumption

6.4 Energy-Aware Hadoop System Architecture

When a client node submits a job to Hadoop NameNode, by default Hadoop framework will split the input data into data blocks, replicates the blocks in the cluster's nodes based on replication factor, and then uses the cluster nodes resources to process and complete the job and stores the output file(s) in the HDFS, where the client can access it.

Figure 6.4.1 shows the default Hadoop framework architecture, the resource manager node decides upon the resources i.e. CPU, network resources, memory, disk space…etc. for each DataNode (refer to chapter 2 for more details).

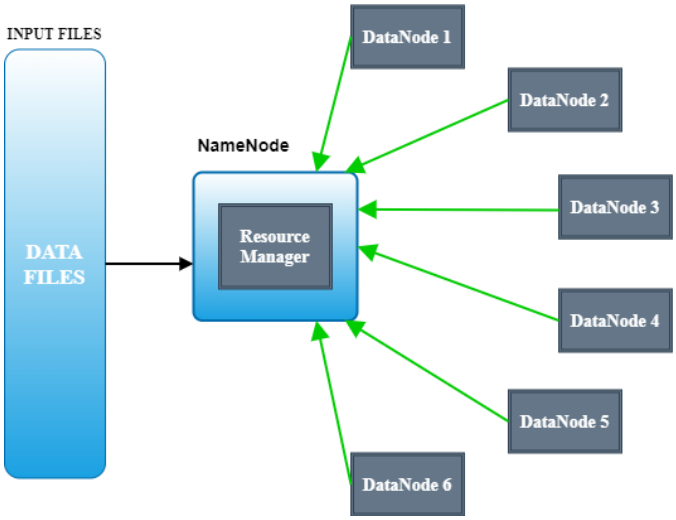


Figure 6.4.1: The Default Hadoop Cluster Framework before Integrating our Intelligent Module

In the above architecture, the resource manager node maintains a live connection with all the DataNodes in the cluster in order to manage the job execution, and the resource provisioning decision does not take in the consideration the amount of power that will be consumed to execute the job, therefore, it is highly likely that a certain extra unnecessary amount of power will be consumed with each job execution process.

On the contrary, our proposed energy-aware Hadoop framework does take in the consideration the minimum amount of power that is needed to execute a job, and so the cluster rescales up or down based on the minimum number of nodes that are required to complete the job. Figure 6.4.2 shows our proposed energy-aware Hadoop

framework, where the NameNode is equipped with a ML-based module that assists the resource manager in managing the cluster's resources.

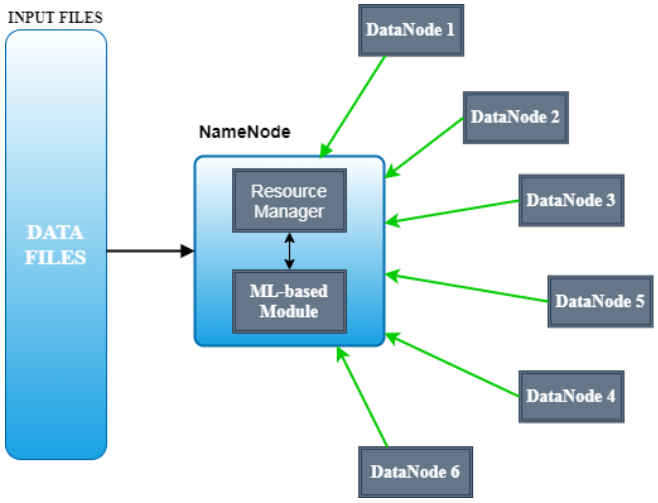


Figure 6.4.2: Energy-Aware Hadoop Cluster Framework Equipped with our ML-based Module

In the proposed framework, upon the NameNode (Master Node) receiving a job, based on the job profile (characteristics) the ML-based module will predict the minimum necessary cluster resources that are required to execute the job, based on the prediction number of DataNodes will be either decommissioned from the cluster or commissioned to the cluster to complete the job execution in an energy-aware environment.

Example: Consider submitting a 37.5 GB Wordcount workload to Hadoop cluster framework that is shown in Figure 6.4.2, based on our experimental observation (see Appendix B) a 37.5 GB Wordcount workload would be processed on a 4-nodes Hadoop cluster (1 NameNode and 3 DataNodes). With the workload profile and the assistance of the ML-based module in the NameNode, the resource requirements (4-nodes) will be

predicted. Figure 6.4.3 shows the expected system behavior with the ML-based module assistance.

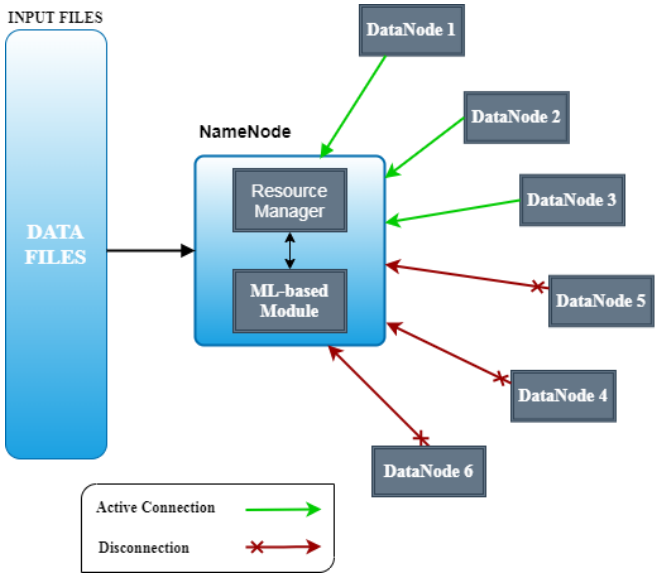


Figure 6.4.3: Example of Decommissioning 3 DataNodes based on the ML-based Module

Therefore, out of the 7-nodes in the cluster 3 Datanodes will be decommissioned (which means disconnected from the cluster, or put in standby mode, or completely powered off) from the cluster, and the 37.5 GB Wordcount workload will be processed on 4-nodes cluster. Processing the workload on 4-nodes (the power consumption is 0.104 KWh) instead of 7-nodes cluster (the power consumption is 0.225 KWh) would save about 53.78% of the operation's power consumption.

6.5 Conclusion

In this study, we have proved that enterprise datacenters can potentially increase their business profitability by decreasing the operating costs when adopting intelligent

solutions in production. The energy cost in datacenters while processing batch jobs can significantly be decreased, by reducing the operations power consumption through our proposed smart data placement solution in Hadoop clusters. Our results analysis showed that by augmenting the traditional Hadoop framework with our ML-based module which makes predictions based on the workload profile, the power consumption of processing workloads can be reduced by more than 50% in some cases.

In addition, one of the most valuable observations in our study is that by decreasing the data blocks replication factor in Hadoop cluster, the power consumption can be reduced significantly. Such feature can be added to our ML-based module based on task requirements and business need.

6.6 Future Work and Scalability

Developing an energy-aware and auto-scale framework solution for Hadoop cluster can be one of the most promising continuation to our current study, where we can replace the manual commissioning/decommissioning technique of DataNodes by an intelligent framework that is able to facilitate Hadoop cluster scalability. In the production environment this auto-scale framework solution can be implemented in one of two ways:

1- A standalone smart module that takes the decision by the leverage of ML algorithm on the required DataNodes to be connected with the NameNode in

Hadoop cluster, decommission the unneeded DataNodes, then place the data in the HDFS as a preparation step for processing in Hadoop cluster.

2- An intelligent module integrated in Hadoop source code as a novel energy-aware Hadoop distribution. In this framework, Hadoop delegates its preliminary phase of the resource management to the integrated module, which uses ML algorithm to energy-aware rescaling the cluster, then place the data in the HDFS for processing.

Bibliography

Lublinsky, B., Smith, K. T., & Yakubovich, A. (2013). *Professional Hadoop Solutions*. Indianapolis, IN: John Wiley & Sons

Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., & Maggs, B. (2009). Cutting the electricity bill for Internet-Scale systems. *SIGCOMM'09*. doi: 10.1145/1592568.1592584

Top 10 industries using Big Data and 121 companies who hire Hadoop developers. (2016). Retrieved from https://www.dezyre.com/article/top-10-industries-using-big-data-and-121-companies-who-hire-hadoop-developers/69

Ghemawat, S., Gobioff, H., & Leung, S-T., (2003). The Google File System. *Google*. Retrieved from https://storage.googleapis.com/pub-tools-public-publication-data/pdf/035fc972c796d33122033a0614bc94cff1527999.pdf

Dean, J. & Ghemawat, S., (2004). MapReduce: Simplified Data Processing on Large Clusters. *Google*. Retrieved from https://storage.googleapis.com/pub-tools-public-publication-data/pdf/16cb30b4b92fd4989b8619a61752a2387c6dd474.pdf

Rouse, M., (2019). Guide to big data analytics tools, trends, and best practices. *Tech Target, Search Data Management*, Retrieved from https://searchdatamanagement.techtarget.com/definition/Hadoop

White, T. (2015). *Hadoop: The Definitive Guide.* Sebastopol, CA: O'Reilly Media Inc. 2015-03-19T19:44:25Z

Agarwal, S. & Khanam Z., (2015). MapReduce: A Survey Paper on Recent Expansion. *International Journal of Advanced Computer Science and Applications*, *6*(8)

Lee C-W, Hsieh K-Y, Hsieh S-Y, & Hsiao H-C, (2014). A Dynamic Data Placement Strategy for Hadoop in Heterogeneous Environments. *El Sevier Big Data Research*, *1*, 14 – 22. doi: 10.1016/j.bdr.2014.07.002

Brownlee J. Supervised and Unsupervised Machine Learning Algorithms (2016) *Machine Learning Algorithms*. Retrieved from URL https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Logic Regression. (2017). Retrieved from URL: https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html

Multiple Linear Regression. (1997-1998). Retrieved from URL: http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm

Soni, D. Introduction to Naïve Bayes Classification. (2018). Retrieved from URL https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54

Donges, N. A complete Guide to the Random Forest Algorithm. (2019). Retrieved from URL https://builtin.com/data-science/random-forest-algorithm

Apruzzese, G., Colajanni, M., Ferretti, L., Guido, A., Marchetti, M. (2018). On the effectiveness of machine and deep learning for cyber security. *10th International conference on cyber conflic (CyCon).* doi: 10.23919/CYCON.2018.8405026

Patel, S. Chapter 2: SVM (Support Vector Machine) – Theory. (2017). Retreived from URL https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

Dua S. and Du, X., (2011). *Data Mining and Machine Learning in Cyber Security.* Boca Raton, FL: Auerbach Publications. ISBN: 13:978-1-4398-3943-0

Kaushik S. An Introduction to Clustering and different methods of clustering. (2016). Retrieved from URL: https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/

Hacker, T. J., & Mahadik, K. (2011). Flexible resource allocation for reliable virtual cluster computing systems. *International Conference for High Performance Computing, Networking, Storage and Analysis*. doi: 10.1145/2063384.2063448

Tian, F., & Chen, K. (2011). Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds. *IEEE 4th International Conference on Cloud Computing*. doi: 10.1109/CLOUD.2011.14

Palanisamy, B., Singh, A., & Liu, L. (2015). Cost effective resource provisioning for MapReduce in a cloud. *IEEE Transactions on Parallel and Distributed Systems*, *26*(5), 1265 – 1279. doi: 10.1109/TPDS.2014.2320498

Jalaparti, V., Ballani, H., Costa, P., Karagiannis, T., & Rowstron, A. (2012). Bazaar: Enabling predictable performance in datacenters. *Microsoft Res., Cambridge, U.K., Tech. Rep*. MSR-TR-2012-38, 2012

Kaushik, R. T., Bhandarkar, M., & Nahrstedt, K. (2010). Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System. *IEEE Second International Conference on Cloud Computing Technology and Science*. doi: 10.1109/CloudCom.2010.109

Goiri, I., Le, K., Nguyen, T. D., Guitart, J., Torres, J., & Bianchini, R. (2012). GreenHadoop: leveraging green energy in data-processing frameworks. *Proceeding EuroSys '12 Proceedings of the 7th ACM european conference on Computer Systems*, 57 – 70. doi: 10.1145/2168836.2168843

Wirtz, T., & Ge, R. (2011). Improving MapReduce energy efficiency for computation intensive workloads. *2011 International Green Computing Conference and Workshops*. doi: 10.1109/IGCC.2011.6008564

Lang, W., & Patel, J. M. (2010). Energy management for MapReduce Clusters. *Proceedings of the VLDB Endowment*, *3*(1-2), 129 – 139. doi: 10.14778/1920841.1920862

Sandholm, T., Lai, K. (2009). MapReduce optimization using regulated dynamic prioritization. *SIGMETRICS '09 Proceedings of the eleventh international joint*

conference on Measurement and modeling of computer systems, *37*(1), 299 – 310. doi: 10.1145/2492101.1555384

Wang, X., Shen, D., Yu, G., Nie, T., & Kou Y. (2013). A Throughput Driven Task Scheduler for Improving MapReduce Performance in Job Intensive Environments. *IEEE International Congress on Big Data*. doi: 10.1109/BigData.Congress.2013.36

Verma, A., Cherkasova, L., & Campbell, R. H. (2011). ARIA: automatic resource inference and allocation for mapreduce environments. *ICAC '11 Proceedings of the 8th ACM international conference on Autonomic computing*. 235 – 244. doi: 10.1145/1998582.1998637

Kurazumi, S., Tsumura, T., Saito, S., Matsuo, H. (2012). Dynamic Processing Slots Schedulaing for I/O Intensive Jobs of Hadoop MapReduce. *IEEE Third International Conference on Networking and Computing*. doi: 10.1109/ICNC.2012.53

Manzanares, A., Qin, X., Ruan, X., & Yin, S. (2011). Pre-bud: Prefetching for energy-efficient parallel i/o systems with buffer disks. *ACM Transactions on Storage (TOS)*, *7*(1), 3. doi: 10.1145/1970343.1970346

Nayak, R., (2018). Hadoop Performance Evaluation by Benchmarking and Stress Testing with TeraSort and TestDFSIO. Retrieved from https://medium.com/ymedialabs-innovation/hadoop-performance-evaluation-by-benchmarking-and-stress-testing-with-terasort-and-testdfsio-444b22c77db2

Huang S., Huang J., Liu Y., Yi L., and Dai J., (2010) "HiBench: A Representative and Comprehensive Hadoop Benchmark Suite," Intel Asia-Pacific Research and Development Ltd., Shanghai, P.R. China, 200241.

Hadoop Architecture in Detail – HDFS, YARN & MapReduce (2019). Retrieved from URL: https://data-flair.training/blogs/hadoop-architecture/

Hadoop HDFS Architecture Explanation and Assumptions (2020) Retrieved from URL: https://data-flair.training/blogs/hadoop-hdfs-architecture/

Appendix A

The Hadoop cluster main configuration settings in this study are included in the following four files:

core-site.xml configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
    <name>fs.default.name</name>
    <value>hdfs://master:9000</value>
    <description>The name of the default file system.
        A URI whose scheme and authority determine the FileSystem
        implementation. The uri's scheme determines the config
        property (fs.SCHEME.impl) naming the FileSystem
        implementation class. The uri's authority is used to determine
        the host, etc. for a filesystem.
    </description>
</property>

<property>
    <name>io.file.buffer.size</name>
    <value>131072</value>
</property>

</configuration>
```

hdfs-site.xml configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
    <name>dfs.replication</name>
    <value>3</value>
</property>

<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/yarn_data/hdfs/namenode</value>
</property>

<property>
    <name>dfs.blocksize</name>
    <value>536870912</value>
</property>

</configuration>
```

As we can see in the above configuration that the replication factor in our expeiment is set to 3, and the data blocksize is set to 512 MB

mapred-site.xml configurations:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>

<property>
    <name>mapreduce.jobhistory.address</name>
    <value>master:10020</value><!-- pointing jobhistory to master -->
    <description>Host and port for job History Server (default 0.0.0.0:10020)</description>
</property>

<property>
        <name>mapreduce.map.memory.mb</name>
        <value>1536</value>
</property>

<property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>3072</value>
</property>
<property>
     <name>mapreduce.map.cpu.vcores</name>
     <value>4</value>
</property>

<property>
        <name>mapreduce.reduce.cpu.vcores</name>
        <value>4</value>
</property>

<property>
        <name>mapreduce.task.io.sort.mb</name>
        <value>100</value>
</property>

<property>
        <name>mapreduce.task.io.sort.factor</name>
        <value>10</value>
</property>

<property>
        <name>mapreduce.reduce.shuffle.parallelcopies</name>
        <value>50</value>
</property>
```

```
<property>
        <name>mapreduce.map.sort.spill.percent</name>
        <value>0.80</value>
</property>

<property>
        <name>mapreduce.job.running.map.limit</name>
        <value>0</value>
</property>

<property>
        <name>mapreduce.job.running.reduce.limit</name>
        <value>0</value>
</property>

</configuration>
```

yarn-site.xml configuration:

```
<?xml version="1.0"?>

<configuration>

<!-- Site specific YARN configuration properties -->

<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>

<property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>

<property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>master:8025</value>
</property>

<property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>master:8030</value>
</property>
```

```
<property>
    <name>yarn.resourcemanager.address</name>
    <value>master:8050</value>
</property>

<property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>2048</value>
</property>

<property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>7168</value>
</property>

<property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>4</value>
</property>

<property>
        <name>yarn.scheduler.minimum-allocation-vcores</name>
        <value>1</value>
</property>

<property>
        <name>yarn.scheduler.maximum-allocation-vcores</name>
        <value>4</value>
</property>

</configuration>
```

Appendix B

The below tables depict the study observations. The table's column [Data (GB)] represents the workload size in Giga Byte, the column [CPU] represents the average CPU utilization of the node while processing the corresponding data size that is shown in the table, the [Mem] column represents the average memory utilization, the [HDD] column represents the average storage utilization of the node while processing the corresponding workload, the [Exec Time] column represents the job execution time and the [Total Power] column represents the entire cluster's total power consumption in (KWh) to complete the MapReduce job.

Note

The highlighted cells in the table indicate that X workload should be processed by Y number of nodes in order to consume the lowest amount of power in Hadoop cluster, which concludes our study goal i.e. the optimal number of cluster's nodes for processing a 65 GB Terasort workload with the lowest amount of power consumption is 2-nodes, and so on so forth for all the workloads shown in the tables.

Terasort workload characterization

- 2-nodes Hadoop cluster

	NameNode			DataNode				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power

0.25	4.61	23.52	8.00	43.72	23.75	3.00	0:05	0.004
0.5	4.11	23.45	8.00	43.55	23.99	3.00	0:06	0.004
0.75	3.95	23.47	8.00	44.62	25.13	3.00	0:06	0.005
1	3.68	23.68	8.00	44.62	26.44	3.00	0:06	0.005
2.5	3.11	23.74	8.00	50.49	26.58	6.00	0:09	0.008
5	2.29	23.83	8.00	48.57	27.75	9.00	0:13	0.012
7.5	1.92	23.57	8.00	49.20	27.17	9.00	0:18	0.014
10	1.71	24.14	8.00	48.49	29.25	16.00	0:23	0.020
12.5	1.52	24.37	8.00	44.95	27.00	13.00	0:28	0.025
15	1.41	24.11	8.00	44.52	27.99	17.00	0:34	0.027
17.5	1.28	24.38	8.00	42.71	27.40	24.00	0:40	0.032
20	1.24	24.78	8.00	42.92	27.26	22.00	0:45	0.036
22.5	1.24	24.20	8.00	50.92	27.78	26.00	0:44	0.037
50	0.86	25.40	8.00	36.24	28.72	59.00	1:54	0.084
65	0.79	26.07	36.00	34.02	24.14	88.00	2:33	0.114

- 3-nodes Hadoop cluster

	NameNode			**Datanode 1**			**DataNode 2**				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
0.25	7.18	23.23	8.00	34.54	38.42	2.00	50.60	20.26	3.00	0:03	0.004
0.5	6.42	23.02	8.00	31.90	38.12	3.00	50.32	20.42	3.00	0:03	0.006
0.75	6.44	22.64	8.00	34.54	39.25	3.00	56.75	21.18	3.00	0:04	0.005
1	6.25	23.63	8.00	43.11	40.90	3.00	45.91	20.15	3.00	0:03	0.005
15	1.60	24.14	8.00	25.81	47.51	12.00	33.11	26.90	17.00	0:28	0.032
17.5	1.62	24.38	8.00	34.19	50.08	15.00	30.18	28.79	13.00	0:29	0.035
20	1.56	24.75	8.00	37.54	48.72	16.00	34.04	29.35	16.00	0:29	0.038
22.5	1.53	24.30	8.00	34.96	46.09	17.00	38.19	27.24	23.00	0:31	0.042
50	1.11	25.22	8.00	27.24	46.75	34.00	32.88	26.45	49.00	1:12	0.093
62.5	0.87	25.85	36.00	18.78	46.38	39.00	28.30	22.35	48.00	1:54	0.128
65	0.88	25.68	36.00	21.63	37.94	40.00	27.66	22.76	50.00	1:53	0.127
67.5	0.89	25.32	36.00	25.58	37.33	49.00	25.90	21.35	45.00	1:54	0.131
70	0.89	24.73	36.00	24.48	37.75	51.00	24.60	21.46	48.00	1:58	0.132
72.5	0.86	25.43	36.00	19.16	36.61	51.00	27.65	22.51	57.00	2:07	0.141
75	0.84	25.18	36.00	20.24	37.48	66.00	25.82	21.89	52.00	2:10	0.145
77.5	0.86	25.89	36.00	19.81	36.02	45.00	31.47	22.89	62.00	2:07	0.147
80	0.82	25.43	36.00	15.64	37.05	55.00	27.74	21.60	61.00	2:32	0.163
82.5	0.80	25.61	36.00	20.51	46.78	53.00	25.43	22.08	60.00	2:27	0.163

- 4-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
65	0.98	25.36	36.00	22.26	46.72	46.00	21.53	22.48	48.00	29.87	21.58	36.00	1:28	0.135
72.5	1.02	25.59	36.00	19.45	35.45	43.00	23.91	22.33	51.00	25.40	21.40	43.00	1:35	0.145
75	0.98	25.36	36.00	20.48	34.47	46.00	26.38	22.47	47.00	26.91	21.86	46.00	1:32	0.148
77.5	0.98	25.45	36.00	20.06	34.89	45.00	28.48	21.40	52.00	24.54	23.78	49.00	1:40	0.158
80	0.90	25.49	36.00	15.54	45.26	51.00	22.23	21.21	47.00	27.56	21.34	56.00	1:52	0.171
82.5	0.92	25.40	36.00	21.12	40.62	48.00	24.05	21.03	50.00	24.52	21.34	53.00	1:52	0.173
85	0.95	25.41	36.00	16.61	36.83	56.00	26.27	21.36	60.00	24.19	20.56	51.00	1:54	0.173
100	0.75	25.96	35.00	15.13	36.74	63.00	19.13	20.63	58.00	26.38	20.39	70.00	2:19	0.209

- 5-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
62.5	1.17	25.24	36.00	18.79	46.41	30.00	22.11	20.37	34.00	26.87	20.78	30.00
65	1.18	24.97	36.00	22.44	43.76	31.00	18.23	21.31	30.00	26.08	20.59	34.00
67.5	1.16	24.89	36.00	21.64	44.99	36.00	21.68	21.88	31.00	22.10	19.80	34.00
70	1.12	25.23	36.00	17.53	34.21	36.00	21.43	20.95	35.00	23.94	20.89	30.00
72.5	1.13	25.26	36.00	15.36	33.89	34.00	20.12	22.15	35.00	21.88	19.36	31.00
75	1.10	25.28	36.00	20.76	34.00	39.00	21.47	21.35	38.00	28.82	20.90	36.00
77.5	1.10	24.96	36.00	17.90	36.90	40.00	23.89	21.56	33.00	25.68	21.42	38.00
80	1.03	25.14	36.00	18.07	36.82	41.00	23.44	22.74	35.00	20.78	19.97	41.00
82.5	1.08	25.60	36.00	22.33	43.58	44.00	27.86	21.46	43.00	23.34	24.00	37.00
85	1.04	25.35	36.00	19.61	45.11	37.00	21.32	21.99	38.00	23.82	19.68	34.00
100	0.85	25.54	35.00	14.48	37.03	43.00	20.13	22.40	44.00	23.86	19.47	49.00

	DataNode 4				
Data (GB)	CPU	Mem	HDD	Exec Time	Total Power
62.5	16.60	20.25	15.00	1:05	0.118
65	21.67	20.81	17.00	1:03	0.118
67.5	17.73	20.71	17.00	1:09	0.128
70	18.02	19.96	19.00	1:11	0.132
72.5	23.16	20.50	20.00	1:12	0.130
75	16.74	21.39	18.00	1:14	0.141

77.5	13.41	24.33	21.00	1:22	0.150
80	17.05	20.31	18.00	1:26	0.155
82.5	17.40	19.28	19.00	1:18	0.154
85	18.03	19.62	25.00	1:30	0.166
100	13.28	18.69	25.00	1:53	0.201

- 6-nodes Hadoop cluster

	NameNode			**DataNode 1**			**DataNode 2**			**DataNode 3**		
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**
65	0.93	25.30	35.00	16.58	35.11	31.00	18.75	25.80	24.00	19.22	21.96	30.00
67.5	0.91	25.35	35.00	16.88	33.40	30.00	19.71	20.48	27.00	18.00	19.39	25.00
70	0.89	26.23	35.00	16.27	34.30	27.00	20.98	21.26	29.00	20.07	21.27	26.00
75	0.89	26.17	35.00	15.64	33.41	31.00	19.03	20.39	28.00	16.72	19.56	31.00
80	0.89	25.45	35.00	15.87	35.77	31.00	18.72	21.24	31.00	16.60	18.92	28.00
100	0.77	25.68	35.00	13.98	33.35	32.00	16.25	20.03	35.00	18.34	18.66	37.00

	DataNode 4			**DataNode 5**				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
65	13.08	20.55	12.00	17.66	16.12	12.00	1:32	0.163
67.5	11.88	19.66	12.00	27.54	18.66	15.00	1:30	0.167
70	13.45	19.86	14.00	18.26	17.06	12.00	1:34	0.171
75	13.75	20.38	14.00	28.35	16.00	16.00	1:35	0.174
80	14.92	19.12	16.00	30.95	16.49	17.00	1:38	0.181
100	10.98	17.80	20.00	27.75	16.23	22.00	2:13	0.235

- 7-nodes Hadoop cluster

	NameNode			**DataNode 1**			**DataNode 2**			**DataNode 3**		
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**
65	1.94	26.33	35.00	15.66	32.77	26.00	14.52	21.31	22.00	16.88	20.06	22.00
67.5	1.95	27.13	35.00	11.96	32.75	24.00	14.32	21.22	21.00	16.51	19.75	21.00
70	0.88	25.83	35.00	13.50	32.64	27.00	13.96	19.89	24.00	16.65	20.28	23.00
75	0.86	26.49	35.00	12.11	32.75	26.00	16.47	20.56	26.00	15.51	18.68	24.00
80	0.80	27.39	35.00	14.60	31.23	32.00	15.07	20.31	25.00	14.22	18.63	22.00

	DataNode 4			DataNode 5			DataNode 6				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
65	11.09	20.51	10.00	19.15	18.82	12.00	22.18	20.73	10.00	1:36	0.177
67.5	16.51	19.75	21.00	19.23	19.44	11.00	19.23	19.44	11.00	1:48	0.195
70	11.58	18.85	9.00	16.70	16.62	11.00	23.28	16.48	12.00	1:45	0.194
75	12.92	18.67	13.00	15.86	15.95	13.00	16.22	17.33	11.00	1:59	0.223
80	13.24	21.37	14.00	15.48	16.79	11.00	20.48	16.20	14.00	2:01	0.221

Sort workload characterization

- 2-nodes Hadoop cluster

	NameNode			DataNode				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
25	2.14	24.06	11.00	23.64	28.57	25.00	1:01	0.045
35	3.45	24.68	16.00	27.53	27.05	39.00	1:05	0.047
40	2.25	24.75	35.00	24.00	23.28	48.00	1:29	0.062
45	3.50	24.99	29.00	23.85	26.13	53.00	1:30	0.070
55	3.57	25.25	29.00	22.50	25.94	57.00	1:53	0.088
60	2.42	24.99	29.00	22.15	26.90	60.00	2:05	0.087
62.5	2.48	25.55	29.00	21.65	26.66	61.00	2:13	0.092
65	2.49	25.09	29.00	21.09	28.56	74.00	2:22	0.102

- 3-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
25	1.84	24.28	11.00	16.08	43.69	34.00	17.76	25.91	34.00	1:14	0.086
35	3.34	25.09	16	17.52	42.36	43	21.13	25.04	50	1:19	0.087
40	2.20	24.05	35	18.13	36.30	39	22.37	22.64	40	1:09	0.073
55	3.01	25.13	29	12.24	43.72	50	17.14	24.88	52	1:58	0.135
65	1.83	25.94	29	11.74	42.50	60.00	15.04	25.75	68.00	2:38	0.169
72.5	1.90	25.45	29	10.40	43.02	66.00	14.49	25.05	81.00	2:49	0.185

- 4-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
40	1.53	24.24	35.00	14.84	35.75	42.00	17.87	21.00	41.00	17.83	20.97	40.00	1:16	0.103
65	1.94	25.34	35.00	12.51	35.09	62.00	15.58	21.41	63.00	15.64	19.49	62.00	2:01	0.163
67.5	2.02	24.84	38.00	12.63	33.96	65.00	14.74	21.91	65.00	14.62	19.85	64.00	2:09	0.171
70	2.04	24.77	38.00	13.35	32.90	66.00	14.54	20.16	67.00	15.49	19.57	67.00	2:11	0.174
72.5	2.27	24.58	38.00	13.60	33.85	69.00	14.57	20.98	70.00	15.98	19.74	69.00	2:06	0.171

- 5-nodes Hadoop cluster

`	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
40	1.23	24.12	38.00	17.23	34.91	37.00	20.81	21.65	32.00	19.42	20.66	29.00
65	2.13	23.97	38.00	13.15	32.61	52.00	15.03	22.26	49.00	16.29	21.42	46.00
70	2.19	24.66	38.00	14.38	34.04	49.00	15.80	21.68	54.00	15.90	19.31	51.00
72.5	2.11	24.52	38.00	13.47	34.88	50.00	13.83	20.67	55.00	13.88	19.73	53.00
75	2.18	25.14	35.00	12.10	34.11	60.00	13.53	20.07	56.00	14.58	18.92	52.00

	DataNode 4				
Data (GB)	CPU	Mem	HDD	Exec Time	Total Power
40	14.83	19.82	16.00	0:52	0.091
65	12.26	21.21	24.00	1:29	0.146
70	13.28	18.39	25.00	1:34	0.154
72.5	11.41	19.29	26.00	1:44	0.169
75	11.23	18.79	27.00	1:51	0.178

- 6-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
72.5	1.71	31.83	35.00	10.48	30.79	50.00	13.14	18.91	46.00	10.45	17.43	41.00

	DataNode 4			DataNode 5				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
72.5	8.37	17.57	22.00	13.39	15.26	20.00	3:03	0.286

Wordcount workload characterization

- 2-nodes Hadoop cluster

	NameNode			DataNode				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
20	2.16	23.21	17.00	77.63	31.54	11.00	1:05	0.063
22.5	2.01	23.51	18.00	75.33	29.82	13.00	1:14	0.070
25	2.08	23.67	41.00	76.32	25.50	14.00	1:21	0.077
27.5	2.05	23.35	19.00	76.80	27.30	15.00	1:28	0.084
30	2.02	23.47	21.00	77.90	28.96	16.00	1:35	0.101

- 3-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
20	3.23	23.36	17.00	59.82	41.58	11.00	70.43	27.89	11.00	0:39	0.065
22.5	3.27	24.06	18.00	44.52	44.92	12.00	83.09	27.12	13.00	0:44	0.075
25	3.03	22.90	19.00	39.15	43.48	14.00	84.94	27.80	14.00	0:50	0.076
27.5	2.92	23.74	42.00	71.19	40.47	15.00	67.96	21.58	15.00	0:49	0.079
30	3.20	22.89	43.00	71.03	39.48	16.00	62.31	20.42	16.00	0:55	0.086
32.5	2.82	23.05	44.00	64.96	43.75	17.00	76.08	25.44	18.00	0:55	0.092
35	2.85	23.26	35.00	88.60	49.27	18.00	43.76	21.05	19.00	1:05	0.103
37.5	2.90	23.92	36.00	42.38	43.97	19.00	84.46	23.35	20.00	1:13	0.116

- 4-nodes Hadoop cluster

	NameNode	DataNode 1	DataNode 2	DataNode 3		

Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
22.5	4.35	23.84	45.00	33.90	28.04	13.00	33.18	17.15	14.00	36.49	18.11	13.00	0:58	0.096
25	3.90	23.22	41.00	73.20	36.21	14.00	39.41	24.08	14.00	56.77	20.20	14.00	0:37	0.076
27.5	3.80	22.89	42.00	34.76	40.55	15.00	59.94	21.12	15.00	78.02	22.36	15.00	0:40	0.085
30	3.77	23.84	43.00	63.86	44.01	16.00	34.70	20.07	16.00	76.26	22.09	16.00	0:43	0.089
32.5	3.86	23.58	34.00	81.43	48.30	17.00	64.99	21.68	18.00	39.51	21.27	17.00	0:44	0.095
35	4.01	23.32	35.00	62.83	43.98	18.00	58.27	20.76	19.00	58.97	21.49	18.00	0:46	0.099
37.5	4.10	24.09	36.00	70.60	48.04	19.00	58.90	20.55	20.00	65.20	20.19	19.00	0:47	0.104

- 5-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
22.5	4.67	23.81	45.00	33.63	30.29	11.00	23.62	17.58	11.00	14.06	15.46	11.00
25	4.66	23.95	46.00	22.35	27.13	11.00	33.10	18.01	11.00	24.57	18.29	11.00
30	3.48	24.08	48.00	25.15	25.91	13.00	36.94	17.48	14.00	28.93	17.61	12.00
37.5	3.62	23.22	52.00	25.32	25.86	15.00	19.33	16.77	17.00	34.64	16.73	15.00

	DataNode 4				
Data (GB)	CPU	Mem	HDD	Exec Time	Total Power
22.5	34.85	16.82	5.00	0:54	0.093
25	27.57	17.10	6.00	0:58	0.104
30	16.91	15.63	7.00	1:12	0.131
37.5	27.93	15.53	8.00	1:24	0.152

- 6-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
25	3.53	24.33	46.00	13.17	25.04	10.00	14.66	16.23	10.00	12.97	16.40	7.00
30	2.52	24.38	48.00	20.88	25.61	11.00	14.58	16.19	12.00	10.01	15.16	11.00
37.5	2.71	24.69	52.00	15.75	24.48	13.00	18.64	16.70	15.00	15.51	15.01	13.00

	DataNode 4			DataNode 5				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
25	14.63	14.86	5.00	26.00	14.50	4.00	1:29	0.137
30	10.57	15.75	5.00	18.81	14.64	5.00	1:43	0.160
37.5	20.53	17.93	6.00	10.14	14.43	6.00	1:54	0.188

- 7-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
25	2.40	22.84	46.00	12.60	23.64	9.00	13.20	17.54	9.00	12.61	16.59	9.00
30	2.49	24.21	48.00	15.24	24.73	7.00	11.95	17.11	13.00	11.44	14.73	10.00
37.5	2.41	24.24	52.00	14.51	23.15	11.00	7.74	15.16	11.00	10.58	14.61	11.00

	DataNode 4			DataNode 5			DataNode 6				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
25	8.88	14.12	4.00	9.61	14.26	4.00	9.61	14.12	4.00	1:34	0.157
30	16.17	14.92	5.00	8.33	14.19	4.00	7.26	14.19	4.00	1:47	0.179
37.5	15.54	14.56	7.00	7.77	14.21	4.00	11.01	14.51	5.00	2:16	0.225

Pagerank workload characterization

- 2-nodes Hadoop cluster

	NameNode			DataNode				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
1	1.71	24.44	30.00	63.67	23.94	4.00	0:25	0.024
2.5	1.46	24.79	30.00	60.98	25.04	6.00	0:37	0.032
5	1.19	24.77	30.00	56.22	25.72	8.00	1:00	0.052
7.5	1.04	25.36	30.00	56.07	26.00	9.00	1:13	0.064
10	1.05	24.81	30.00	55.14	25.76	9.00	1:26	0.078

- 3-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
1	2.80	23.94	30.00	54.39	43.13	4.00	78.82	23.59	4.00	0:13	0.019
2.5	2.38	24.90	30.00	56.46	42.99	4.00	73.69	26.54	7.00	0:18	0.031
5	1.95	25.40	30.00	48.08	44.39	5.00	75.98	24.65	7.00	0:29	0.044
7.5	1.74	24.94	30.00	48.08	42.73	6.00	75.47	24.87	10.00	0:35	0.057
10	1.51	24.78	30.00	70.43	42.05	7.00	52.02	27.26	7.00	0:41	0.068

- 4-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
1	3.90	24.65	30.00	62.34	32.88	4.00	74.48	20.35	4.00	64.22	25.08	4.00	0:09	0.018
2.5	3.24	25.07	30.00	48.95	36.99	5.00	74.81	22.16	6.00	72.02	25.14	5.00	0:11	0.025
5	2.49	25.28	30.00	68.53	36.27	6.00	47.36	22.76	8.00	75.90	27.10	8.00	0:17	0.038
7.5	2.31	25.23	30.00	69.55	37.24	6.00	63.33	22.81	7.00	54.83	28.75	6.00	0:23	0.050
10	1.88	25.36	30.00	62.04	36.53	13.00	52.41	21.68	7.00	68.54	26.50	7.00	0:28	0.061
25	1.21	25.25	30.00	42.53	36.50	29.00	61.09	22.57	17.00	61.34	21.06	14.00	1:07	0.139

- 5-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
1	4.95	24.84	30.00	68.04	30.90	3.00	56.11	19.12	4.00	64.85	23.21	3.00
2.5	4.16	25.07	30.00	71.09	35.29	5.00	62.05	22.08	5.00	56.27	25.01	4.00
5	3.23	25.08	30.00	55.66	36.46	7.00	57.75	20.93	7.00	66.76	26.45	4.00
7.5	2.85	24.37	30.00	67.78	35.27	6.00	51.94	21.67	8.00	51.73	21.23	5.00
10	2.41	24.85	30.00	64.43	37.18	11.00	50.48	23.15	7.00	53.77	21.27	5.00
25	1.56	24.95	30.00	45.32	34.71	22.00	59.33	23.28	11.00	64.10	22.04	11.00
35	2.62	24.84	35.00	40.92	34.48	29.00	58.68	23.24	16.00	53.22	20.19	15.00

	DataNode 4		

Data (GB)	CPU	Mem	HDD	Exec Time	Total Power
1	60.41	18.68	1.00	0:07	0.018
2.5	57.48	20.28	1.00	0:08	0.023
5	66.71	22.15	2.00	0:12	0.032
7.5	59.20	20.23	2.00	0:17	0.045
10	58.28	20.85	2.00	0:21	0.056
25	45.79	21.51	5.00	0:48	0.125
35	43.89	19.78	7.00	1:10	0.177

- 6-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
1	5.37	25.27	35.00	51.00	29.18	4.00	43.01	18.73	4.00	45.64	18.48	3.00
2.5	3.66	25.49	35.00	45.46	33.81	5.00	43.16	19.96	5.00	46.35	19.63	5.00
5	2.64	25.72	35.00	40.47	36.22	6.00	45.35	22.40	6.00	44.63	20.80	6.00
10	1.87	24.93	35.00	35.80	35.28	6.00	41.84	22.40	7.00	36.31	21.97	6.00
25	1.30	26.06	35.00	38.25	36.50	12.00	38.90	20.64	10.00	42.09	20.13	9.00
35	2.42	24.16	35.00	33.94	32.45	17.00	35.72	19.30	14.00	33.31	18.88	13.00

	DataNode 4			DataNode 5				
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	Exec Time	Total Power
1	52.39	17.77	1.00	44.72	15.68	1.00	0:06	0.014
2.5	40.37	18.22	2.00	30.75	16.74	2.00	0:10	0.026
5	34.31	20.26	2.00	23.40	17.08	2.00	0:17	0.041
10	32.27	19.39	2.00	27.36	17.47	2.00	0:31	0.070
25	29.32	19.20	5.00	26.59	16.59	4.00	1:05	0.148
35	26.15	18.60	6.00	30.14	16.98	5.00	1:39	0.213

- 7-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD	CPU	Mem	HDD
1	6.90	25.21	35.00	46.85	32.05	4.00	47.22	19.37	4.00	47.10	17.30	3.00
2.5	4.73	25.93	35.00	36.01	31.59	5.00	34.12	20.09	5.00	42.15	18.12	4.00

25	2.14	25.65	35.00	19.22	32.40	16.00	27.55	19.44	18.00	28.03	19.12	9.00
35	2.30	25.35	35.00	26.20	33.35	15.00	32.08	20.97	17.00	25.87	19.83	11.00

	DataNode 4			DataNode 5			DataNode 6				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
1	44.06	17.19	1.00	37.23	15.49	1.00	38.56	16.28	1.00	0:05	0.015
2.5	33.33	17.72	2.00	28.35	16.49	2.00	30.93	15.68	1.00	0:11	0.028
25	16.73	17.75	4.00	23.93	16.77	3.00	27.46	15.98	4.00	1:30	0.191
35	20.75	18.29	5.00	22.39	16.24	5.00	24.16	15.93	1.00	1:58	0.257

Kmeans workload characterization

- 2-nodes Hadoop cluster

	NameNode			DataNode				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
1	1.89	24.73	30.00	71.88	26.92	8.00	0:18	0.018
2.5	1.27	27.55	30.00	78.27	25.03	9.00	0:42	0.042
5	1.31	24.65	30.00	81.11	24.55	8.00	0:38	0.036
7.5	1.11	25.24	30.00	79.51	24.67	10.00	0:56	0.055
10	0.99	25.57	30.00	83.73	24.75	12.00	1:15	0.076
22.5	1.48	30.9	8.00	84.56	28.65	24.00	3:43	0.239

- 3-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
1	2.62	24.75	30.00	47.19	35.54	8.00	56.98	23.87	8.00	0:12	0.018
2.5	1.78	24.99	30.00	68.81	38.07	9.00	66.66	26.28	9.00	0:26	0.039
5	1.91	25.13	30.00	72.08	43.46	8.00	70.61	23.12	8.00	0:22	0.036
7.5	1.51	25.69	30.00	78.14	39.98	10.00	65.38	28.06	10.00	0:36	0.054
10	1.43	25.77	30.00	61.24	48.63	12.00	77.61	29.42	12.00	0:48	0.075
12.5	2.15	25.76	30.00	79.79	40.69	15.00	71.24	24.39	16.00	1:03	0.104
15	2.07	25.96	30.00	74.94	41.49	17.00	79.48	22.2	18.00	1:15	0.128

17.5	1.01	26.26	30.00	91.19	50.79	19.00	57.92	27.82	20.00	1:38	0.160
20	0.86	26.58	30.00	85.20	50.13	21.00	65.62	27.9	22.00	1:53	0.187
22.5	2.26	38.87	8.00	85.09	50.37	24.00	70.93	29.13	24.00	2:06	0.222

- 4-nodes Hadoop cluster

	NameNode			**DataNode 1**			**DataNode 2**			**DataNode 3**				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
1	2.94	26.10	30.00	44.28	37.70	8.00	60.78	23.05	8.00	41.73	20.69	8.00	0:11	0.022
2.5	1.98	27.64	30.00	57.87	47.60	9.00	57.32	22.53	9.00	65.12	23.51	9.00	0:21	0.043
5	2.25	25.32	30.00	62.15	47.65	8.00	61.65	23.25	8.00	69.35	23.84	8.00	0:17	0.036
7.5	1.83	25.54	30.00	69.14	49.40	10.00	59.98	22.88	11.00	68.47	22.45	10.00	0:26	0.057
10	1.55	25.80	30.00	67.70	47.78	12.00	66.68	23.69	12.00	65.37	22.25	12.00	0:35	0.078
12.5	2.40	26.06	30.00	61.54	41.29	15.00	79.55	25.22	16.00	72.62	23.60	15.00	0:46	0.105
15	1.19	26.39	30.00	69.08	39.34	17.00	73.19	23.10	18.00	74.07	20.91	17.00	0:55	0.127
17.5	1.10	25.96	30.00	72.64	50.51	19.00	69.83	27.85	20.00	71.03	22.33	19.00	1:09	0.157
20	1.06	26.87	30.00	73.44	49.63	22.00	75.60	27.66	22.00	78.12	23.90	22.00	1:18	0.182
22.5	1.32	36.86	30.00	74.27	50.29	24.00	66.26	27.29	24.00	71.98	23.52	24.00	1:37	0.218

- 5-nodes Hadoop cluster

	NameNode			**DataNode 1**			**DataNode 2**			**DataNode 3**		
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**
1	3.41	25.83	30.00	54.61	36.43	7.00	44.17	20.90	7.00	54.61	21.87	7.00
2.5	2.52	25.02	30.00	55.81	35.81	7.00	62.32	26.05	8.00	64.02	23.43	8.00
5	2.58	25.28	30.00	66.59	38.82	8.00	57.39	21.50	7.00	60.54	21.37	6.00
7.5	2.42	25.45	30.00	63.21	38.21	8.00	67.33	25.96	9.00	61.60	22.44	7.00
10	1.93	25.78	30.00	63.89	39.27	10.00	65.97	23.39	9.00	69.30	23.50	11.00
12.5	2.72	25.80	30.00	72.05	39.35	12.00	65.21	23.54	13.00	68.46	22.03	12.00
15	1.56	25.99	30.00	66.60	41.31	14.00	70.93	24.25	13.00	70.35	22.73	13.00
17.5	1.35	26.16	30.00	66.90	38.82	16.00	69.92	23.83	16.00	69.08	22.47	15.00
20	1.76	37.55	30.00	64.20	49.03	17.00	70.55	27.48	17.00	71.29	25.16	16.00
22.5	2.45	33.90	30.00	62.93	45.91	19.00	73.76	29.45	20.00	72.19	25.64	17.00

25	1.12	26.47	30.00	80.04	41.12	21.00	69.72	23.17	21.00	74.16	21.70	20.00

	DataNode 4				
Data (GB)	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
1	39.20	18.80	3.00	0:09	0.022
2.5	51.81	21.85	3.00	0:16	0.038
5	55.38	23.36	3.00	0:15	0.038
7.5	58.12	23.47	4.00	0:19	0.049
10	53.00	21.96	4.00	0:27	0.068
12.5	72.19	22.55	5.00	0:34	0.094
15	69.93	22.17	7.00	0:41	0.113
17.5	69.96	21.64	7.00	0:50	0.139
20	67.87	27.76	8.00	0:59	0.179
22.5	65.08	26.24	8.00	1:09	0.208
25	73.78	21.66	10.00	1:18	0.224

- 6-nodes Hadoop cluster

	NameNode			DataNode 1			DataNode 2			DataNode 3		
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**
1	2.15	25.09	35.00	24.58	31.70	7.00	26.71	19.40	7.00	8.62	15.51	6.00
2.5	2.18	25.60	35.00	40.16	32.36	7.00	42.70	22.11	9.00	29.18	19.50	7.00
5	1.79	27.92	35.00	33.90	30.87	6.00	29.22	18.79	8.00	17.51	16.84	5.00
7.5	1.95	25.77	35.00	38.56	36.05	8.00	32.19	19.97	9.00	31.32	18.25	8.00
10	1.63	24.99	35.00	41.22	34.49	9.00	44.35	20.87	12.00	40.06	21.36	9.00
12.5	1.47	26.28	35.00	47.48	36.04	11.00	44.92	21.21	12.00	43.97	22.70	11.00
15	1.29	25.71	35.00	41.68	33.76	12.00	46.95	21.34	13.00	50.12	21.22	11.00
17.5	1.29	26.10	35.00	49.13	33.40	13.00	53.35	23.62	14.00	53.83	21.44	13.00
20	1.28	26.53	35.00	52.04	36.09	13.00	54.85	21.67	15.00	55.59	20.27	16.00
22.5	1.26	27.23	35.00	60.06	36.35	17.00	54.25	23.37	17.00	62.70	22.51	16.00
25	1.17	27.21	35.00	53.30	36.31	18.00	66.22	22.35	20.00	59.12	21.10	16.00

	DataNode 4			DataNode 5				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
1	20.76	16.31	3.00	20.20	15.15	2.00	0:18	0.032
2.5	31.64	19.76	4.00	32.12	15.89	2.00	0:23	0.051
5	33.67	18.83	3.00	28.58	16.34	3.00	0:26	0.051

7.5	38.41	19.24	4.00	26.31	16.23	3.00	0:25	0.052
10	36.20	18.57	5.00	35.00	16.65	4.00	0:36	0.083
12.5	36.29	19.00	5.00	38.21	16.69	5.00	0:49	0.116
15	38.91	19.49	5.00	41.60	16.34	6.00	0:58	0.140
17.5	47.56	19.95	7.00	46.78	17.66	6.00	0:58	0.147
20	50.79	20.82	7.00	48.16	17.07	6.00	1:04	0.168
22.5	53.84	21.41	7.00	48.99	17.18	7.00	1:10	0.190
25	53.52	20.81	8.00	52.67	17.10	8.00	1:24	0.229

- 7-nodes Hadoop cluster

	NameNode			**DataNode 1**			**DataNode 2**			**DataNode 3**		
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**
1	2.91	25.69	35.00	28.92	30.91	7.00	21.96	18.26	7.00	16.66	17.82	6.00
5	2.92	25.81	35.00	22.42	27.11	6.00	27.12	18.40	7.00	23.11	17.94	5.00
22.5	1.23	26.46	35.00	47.92	34.26	13.00	50.18	22.00	14.00	45.10	21.10	13.00

	DataNode 4			**DataNode 5**			**DataNode 6**				
Data (GB)	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**CPU**	**Mem**	**HDD**	**Exec Time**	**Total Power**
1	14.97	19.26	3.00	18.46	16.63	2.00	17.14	15.25	2.00	0:27	0.054
5	17.77	17.08	3.00	19.42	15.81	2.00	38.90	17.20	3.00	0:28	0.060
22.5	43.89	19.38	7.00	43.42	16.67	7.00	44.05	18.44	5.00	1:18	0.207