



UNIVERSITEIT VAN AMSTERDAM



---

# Evaluating Energy Consumption of Distributed and Non-Distributed File Systems

Comparative Study

Huseyin Ediz Yildirim

*supervised by*

*Dhr. Dr. Adam Belloum*

*August 27, 2019*

---

## **Contact Information :**

Author :

Huseyin Ediz Yildirim

e-mail : edzyldrm@gmail.com

Supervisors :

Dhr. Dr. A.S.Z. (Adam) Belloum

Faculty of Science

Informatics Institute

e-mail : A.S.Z.Belloum@uva.nl

Dr. A.M. (Ana) Oprescu

Faculty of Science

Informatics Institute

e-mail : A.M.Oprescu@uva.nl

Universiteit van Amsterdam  
Faculty of Science  
Science Park 904  
1098 XH Amsterdam  
Tel. Reception: +31 (0)20525 8626

Vrije Universiteit Amsterdam  
Faculty of Science  
De Boelelaan 1085  
1081 HV Amsterdam  
Tel. Board: +31 (0)20598 7500

## Abstract

Big Data and Cloud Computing technologies are spread over all of the corporate fields and also daily life with health care, social media, tv services and so on. The energy consumption of those services, with the hosts, data centers, computers in the network and all peripherals is even above a large country's annual consumption. Since data centres are all around the world with similar hardware and software, with this volume of consumption and variety of structural elements they are considered with their potential at energy saving because a small change of code may have a huge result in energy consumption. Until recent studies, there were not a challenge of making distributed file systems energy efficient or using them efficiently. The most of the journey of a file happens inside the same rack in which it is used and this is a challenge to overcome in the field since a distributed file stays distributed through all of the journey. To open up this challenge and see the opportunities we evaluated energy consumption and execution time of some types of work loads in Distributed and Host file systems using PowerAPI, a software monitoring tool and derived some ideas around the results.

**Objectives** are to investigate and evaluate different file systems on same hardware and under similar work loads with different types measurements and reports.

**Methods** for this thesis are to use a power monitoring tool Power API for measurements and to use natural work loads with common libraries and computer jobs to trace the changes in systems and their effects on execution time and energy consumption.

**Results** show that some configurational choices and some work loads with different file sizes show remarkable efficiency possibilities (from 11% to 64% for some cases) with high loss on execution time to consider for future work.

**Conclusions** can be made considering the factors of the experiments. File sizes, work load density and block size of distributed file system are both related to energy consumption and under some test cases, there are possibilities to achieve a better energy management for the same work loads.

**Key words:** Review Study, Energy Efficiency, Distributed File Systems, Cloud, Hadoop, ext4, Data Storage, Hybrid, Comparative, Green IT.

## 0.1 ACKNOWLEDGEMENTS

First of all, I would like to thank my parents for their endless support and encouragement for whatever I do in my life. I also want to express my love to my best friend Can, who was the most exclusive roommate, friend and listener ever. You will always be missed and remembered.

I must thank to Dr. Adam Belloum for his patience, support and guidance through the whole process. And I want to thank Dr. Ana Oprescu for insights, comments and guidance through the results of the thesis.

I would like to express my gratitude to the Republic of Turkey - Ministry of National Education for making this Master's and further studies possible with the financial support they exhibit with the scholarship, without which the whole study was not possible for me.

And finally I want to thank my dear friend Emilios Voma for his support from beginning to the end and all of my friends who helped me directly or indirectly for my master studies.

### 0.1.1 LIST OF ABBREVIATIONS

Table 0.1

US	United States
UNCCC	United Nations Framework Convention on Climate Change
IT	Information Technology
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
AWS EC2	Amazon Web Services Elastic Compute Cloud
GCE	Google Compute Engine
IoT	Internet of Things
DFS	Distributed File System
OS	Operating System
HDFS	Hadoop Distributed File System
CPU	Central Processing Unit
API	Application Programming Interface
I/O	Input / Output
VM	Virtual Machine
PM	Physical Machine
POSIX	Portable Operating System Interface for Unix
CLI	Command Line Interface
GUI	Graphical User Interface
DHT	Distributed Hash Table
P2P	Peer to Peer
EMRSA	Energy-Aware MapReduce Scheduling Algorithm
OPT	Optimal
MSPAN	Minimized Makespan
M2M	Machine to Machine
E-EON	Energy Efficiently Optimized Network
EEE	Energy Efficient Ethernet
AQM	Active Queue Management
ECN	Explicit Congestion Notification
SPEC sfs	Standard Performance Evaluation Corporation Server File Suite
SRC	Source
TAR	Target
MB	MegaByte
GB	GigaByte
TB	TeraByte
HDD	Hard Drive Disk
GPU	Graphical Processing Unit

# CONTENTS

0.1	Acknowledgements . . . . .	ii
0.1.1	List of Abbreviations . . . . .	iii
<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.1.1	Cloud . . . . .	2
	Software as a Service (SaaS) . . . . .	3
	Platform as a Service(PaaS) . . . . .	3
	Infrastructure as a Service(IaaS) . . . . .	3
1.1.2	Distributed File Systems . . . . .	3
1.1.3	Apache Hadoop . . . . .	4
	NameNode . . . . .	5
	DataNode . . . . .	5
	Secondary NameNode . . . . .	5
	Job Tracker . . . . .	5
	Task Tracker . . . . .	5
1.1.4	Ext4 FileSystem . . . . .	5
1.1.5	Energy Consumption Measurements . . . . .	6
1.2	Motivation and Objectives . . . . .	6
1.3	Research Questions - Hypotheses . . . . .	7
<b>2</b>	<b>Related Work</b>	<b>8</b>
<b>3</b>	<b>Procedure</b>	<b>14</b>
3.1	Methods Used . . . . .	14
3.1.1	PowerAPI . . . . .	16
3.2	Test Environment . . . . .	17
3.3	Test Structure . . . . .	17
3.4	Energy Consumption . . . . .	20
3.5	Pre-Test Procedure . . . . .	21
3.5.1	Hadoop Installation . . . . .	21
3.5.2	PowerAPI Setup . . . . .	27
3.5.3	Test Generation . . . . .	27
<b>4</b>	<b>Analysis and Results</b>	<b>29</b>
4.1	Energy Consumption of HDFS and Host File System . . . . .	29
4.1.1	Mean Values . . . . .	29
4.1.2	Median Values . . . . .	30
4.1.3	Standard Deviation of Values . . . . .	31
4.1.4	Total Energy Consumption and Execution Time . . . . .	31
	Total Energy Consumption . . . . .	31
	Execution Time . . . . .	32
4.2	Energy Consumption and Execution Time of Migration Between File Systems . . . . .	33

4.3	Energy Consumption and Execution Time for Test Scenarios . . . . .	34
<b>5</b>	<b>Conclusion and Future Work</b>	<b>35</b>
5.1	Analysis of Graphical Data . . . . .	35
5.2	Analysis of Test Runs . . . . .	36
5.3	Conclusions and Analysis of TestBed . . . . .	36
5.4	Future Work . . . . .	37
	<b>References</b>	<b>38</b>
<b>6</b>	<b>Appendix &amp; Notes</b>	<b>42</b>

## LIST OF FIGURES

1.1	Evolution of Cloud . . . . .	2
1.2	Hadoop Architecture . . . . .	5
1.3	Cisco - Global data center traffic estimation by destination in 2021 . . . . .	7
2.1	a) Mirrored Data Block Replication b) Covering Subset Method . . . . .	9
2.2	CPU Utilization - Power Usage . . . . .	10
2.3	Energy Consumption . . . . .	11
2.4	Fraction of Green Energy Cost and Grid Electric Cost of Prioritized Jobs . . . . .	11
2.5	Global devices and connections growth . . . . .	12
3.1	Non-Live VM Migration . . . . .	15
3.2	wordCount Process Flow Diagram . . . . .	16
3.3	PowerAPI Architecture . . . . .	16
3.4	Test 1 Activity Diagram . . . . .	18
3.5	Distribution of VM sizes based on AutoSupport Data . . . . .	18
3.6	Test 2 Activity Diagram . . . . .	19
3.7	Java Version Check . . . . .	22
3.8	SSH to localhost on Ubuntu . . . . .	23
3.9	Hadoop Web Interface . . . . .	26
4.1	Mean Values of Energy Consumption(mW) . . . . .	29
4.2	Median Values of Energy Consumption(mW) . . . . .	30
4.3	Total Energy Consumption of Tests . . . . .	32
4.4	Total Execution Time of Test Cases . . . . .	33



# 1 INTRODUCTION

This document is a thesis report of a comparative study over energy consumption of a distributed file system and a non-distributed file system under similar workloads. Within the document we will question the firmness of using distributed file system in a cluster environment and evaluate the effects of using the host file system instead, with gains and losses. In general the thesis aims to insight researchers to consider host file system as an existing alternative for some cases and be knowing of the relationship between file sizes, application types and Distributed File System's configuration. Chapter 1 is giving background and rationale of the study with preface explanations and definitions with motivation and purpose of the thesis, it is followed by related work in the field in Chapter 2. Chapter 3 explains about the test phase, structure of environment, methods and procedure. Chapter 4 explains the results of the tests and analyses those results with justifying information. Chapter 5 gives the conclusions of the thesis and some proposals of future work.

## 1.1 BACKGROUND

The cloud computing field has grown over the years and the whole concept of cloud computing has been consolidated with numerous studies, researches and applications. Cloud computing paradigm also evolved through years and widened the perspective of the field from a computer network to a general enterprise value. Starting with simple secure backup purposes, it arrived to grid computing and cloud has evolved to mailing apps, stock handling jobs, text messaging applications and now it is used worldwide to handle country-sized economics, stock market transactions, huge social media platforms, broad scientific researches from space explorations to genetic simulations and it can be controlled by almost any device with an internet connection(Figure 1.1) [1]. With the overgrowing demand on this way of processing power, other methodologies were also involved to achieve the ability to use more of the power. One of the paradigms arose with the linked computing is big data concept, which enables the processing power to reach more data sources at the same time through networks and overcome the bottleneck which occurs with lots of computing power but less of data to be able to handle and process. Big data brings Distributed Storage to life, which makes the processing unit and the user independent from the locality of data source and enables implementation of higher skills on analysing, processing, storing and securing the data.

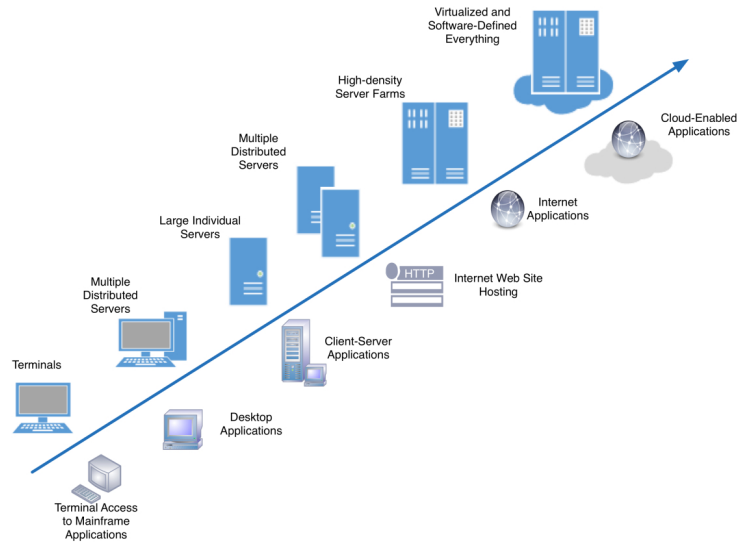


Figure 1.1: Evolution of Cloud

Corporation business, social media, healthcare, science, space explorations, television services, mobile networks; in less words everything is running or being kept in data centers now. Although a small partition of them are powered by Green energy sources such as wind or sun, these sources are not fully available and not enough to power all the infrastructure. Parallel to the progress in the field, usage of computer infrastructure in data centers and demand on energy have a major growth. Power usage of data centers in the United States (US) was estimated as 1.8% of all consumption in 2014 and it is 7% of global energy consumption in 2017, yet it is forecasted that it will be around 13% in 2030. [2] Danilak from Forbes reported that US data centers needs 34 big coal-power plants and global usage of data centers is more than the United Kingdom's entire consumption. [3] According to the United Nations Framework Convention on Climate Change (UNCCC) CO<sub>2</sub> emission rates of information and communications technology (ICT) devices are arising and now it is 2%, equal to the aviation sector. [4] Next to its generalized potential as a problem, there is more to see in the details. While all this energy seems to be going for computation or storage, IT equipment needs cooling, lighting, monitoring and power distribution. With the most efficient scenario, those supportive and complementary equipment take 24% of the total consumption and in average it is 40% . [2]

### 1.1.1 CLOUD

Cloud computing is a technological model for enabling usage of large and shared computing resources over a computer network. It allows users to access a big, flexible, scalable hardware resource with advanced security. It can be accessed, monitored, scaled and managed through the Internet, can be asked and reshaped on users' demand or by user-provided software. Nowadays a big part of the cloud resources are casted by automated software considering estimation of probable need of the cloud software. There have been different approaches

to provide usage of cloud through years, which are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

**SOFTWARE AS A SERVICE (SAAS)** Software as a Service is a computer technology to deliver software as a service through an online distribution. SaaS capacitates users to pay for software subscription and pay for what they use instead of buying the software with a license. [5] Some examples of SaaS are Google Apps, ZenDesk, BigCommerce and Dropbox.

**PLATFORM AS A SERVICE(PAAS)** Platform as a Service (PaaS) is defined by Boniface et al. as the provision of a development platform and environment providing services and storage, hosted in the cloud.[6] PaaS providers provide an environment with all necessary infrastructure to enable developing web applications for developers, saving them from the trouble of knowing the hardware and software behind it. PaaS developers construct and deploy their application on the platform without installing tools, frameworks to run the web application. By this approach, different developers use the same development platform and toolkits to deliver the application to users. Some examples of PaaS are Apprenda, Pivotal, and Red Hat Openshift.

**INFRASTRUCTURE AS A SERVICE(IAAS)** Infrastructure as a Service is a computer solution in which providers bring usage of configured and virtualized hardware to enable users to setup their virtual machines on a user-defined hardware without any server and hardware management. Developers get a subscription for hardware such as memory, processor, storage and pay as much as they use. Some of the examples of IaaS are Amazon Web Services Elastic Compute Cloud (AWS EC2), Digital Ocean and Google Compute Engine (GCE).

### 1.1.2 DISTRIBUTED FILE SYSTEMS

Distributed File System (DFS) as a concept of technology refers to a set of data unit scattered over a network and acts to users as it is kept in a single data storage unit. Data can be accessed and used by multiple users from different locations and with different privileges. DFSs have a broad domain range to be used, such as weather forecasting, scientific purposes, researches in medicine, machine learning, IoT applications, statistical jobs and more. [7]

There are different approaches to achieve the yields of distributed storage and applications. These are architectural choices to ensure better scalability, security, flexibility, maintainability, sustainability, and transparency. A distributed file system can run on a separate Operating System (OS) or there can be a layer of code to manage interaction with other systems. DFS can have a centralized form in which the system is managed by a central node with metadata but the data itself is still distributed over a network. On the other hand DFS can have a fully distributed form in which system is ran by different servers with metadata and other servers just to keep the data or it can be a parallel system in which there are parallelized servers with mirrored data that run synchronously. [7] [8]

### 1.1.3 APACHE HADOOP

Hadoop Distributed File System (HDFS) is a software project licensed under Apache License 2.0 by Apache Software Foundation. It is a software framework for storing data in its distributed file system and for running applications on the data using its own frameworks and services. It is built to run on commodity hardware and it provides thousands of nodes that work together. First released in 01-April-2006 with version 0.1.0 and the most current version by April 2019 is 3.2.0. [9] Hadoop is used by many companies and one of the biggest users, Yahoo!, runs more than 100.000 CPUs in more than 40.000 computers with 4500 nodes in a single cluster[10]. Yahoo! keeps doing scaling tests and development of Hadoop at the same time. As Shvachko et al. expressed, the details of Hadoop architecture and its fundamentals are shown in the Table 1.1 [11]

HDFS	Hadoop Distributed File System
MapReduce	Computation Framework
HBase	Table Service
Pig	Dataflow Language and Parallel Execution Framework
Hive	Data Warehouse Infrastructure
ZooKeeper	Distributed Coordination Service
Chukwa	Management Data Collection System
Avro	Data Serialization System

Table 1.1: Hadoop Constituents

Hadoop is usually considered as two parts, HDFS and MapReduce. HDFS stores the data while MapReduce runs processes on it. Some components of HDFS and their mechanisms within the HDFS architecture (Figure 1.2) were defined below [12]. Most of them are conceptually common in also other big data frameworks; hence some of them are unique to Hadoop.

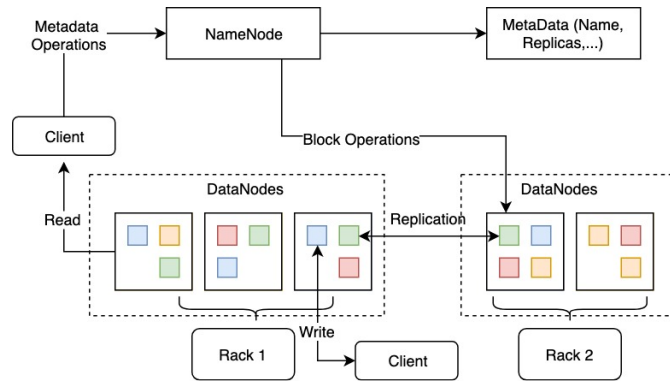


Figure 1.2: Hadoop Architecture

`NAMENODE` is a master service of HDFS, which is responsible for the management of the file system and the data. Management services differ from tracking replication and location of blocks to stashing I/O logs of the entire system and to keeping the metadata of the entire data in the cluster. Blocks are fragments of the data (by default the block size is 128 MB)[13], that NameNode utilizes in order to create copies and replicas on the Data Nodes.

`DATANODE` is a slave service of HDFS, which is responsible for keeping the data stored, writing and reading data controlled by a client through the NameNode. DataNodes send notifications (heartbeats) every 3 seconds (by default) to the NameNode to announce that it is up and running. If NameNode doesn't receive heartbeats from a DataNode for 2 minutes (by default) it is then considered as dead and NameNode takes another possible replica of the data from another DataNode.

`SECONDARY NAMENODE` is a master service of HDFS, which carries out the checkpoints of the metadata of HDFS to present it to NameNode when asked.

`JOB TRACKER` is a master service of HDFS. It collects the requests from the client and informs the NameNode about the needed metadata for MapReduce jobs.

`TASK TRACKER` is a slave service of HDFS, which gets orders of tasks from the Job Tracker and apply them on the file on DataNodes. It is also known as the Mapper. [11] [14]

#### 1.1.4 EXT4 FILESYSTEM

Ext4 (Extended) is file system which was built as a development of Minix File System to overcome some sizing and verification issues. Minix has been released as the file system of Minix Operating System in 1987 and Ext was released 5 years later with one of the first releases of Linux in 1992. The first improvements were on metadata system, and it was upgraded to Ext2 in a short time.

Ext2 was an update to deal with file size problem of that times and its data loss issues in abnormal shutdowns. Small voltage problems were solved with Ext3's journal function, a pre-log of files which records the changes that will be made to the file in advance. Ext4 is the latest version of Ext file system which was primarily made for an improved performance, reliability and capacity. It also has a new way of putting time stamps, including nanoseconds and two high-order bits to overcome the Year 2038 problem, until 2446. Ext4 as it is named, uses real extends instead of blocks which define a start point and an end point on the hard drive, in order to make it possible to reserve a bigger space for a single file by using the same amount of space in mapping [15] [16].

#### 1.1.5 ENERGY CONSUMPTION MEASUREMENTS

Energy consumption measurements can be done using external hardware such as voltmeters or other measurement tools or using software with formulas using the output of sensors on hardware. PowerAPI is an operating system (OS) level system monitoring toolkit which uses hardware sensors to build software-related power measurements. PowerAPI supports multiple types of sensors and by standing between OS and applications, it has a wide range of usage from unit testing to runtime app-specific research. It has also been used as a tool to standardize power monitoring and control at exascale computing [17] [18] [19] [20] [21] [22].

#### 1.2 MOTIVATION AND OBJECTIVES

Cisco Global Cloud Index 2016 report depicts, above 70% of the data traffic goes inside the same cluster and when the racks are considered, above 90% of the traffic goes inside the same rack, mostly in the same hard drive[23](Figure 1.3). Generally those drives are hard-connected to each other via a serial bus or direct network attachments. Distributed file systems, naturally, built to let multiple users to have multiple access on files and they are built on top of non-distributed, conventional (host) file systems. Some parts of workloads that include only a simple process inside the rack don't have to be executed in a distributed environment. These apply for migration of data and metadata, applications and Virtual Machines and replication of data in which data is replicated inside the same cluster as a mirror of the original file for further usage. (Figure 3.1) Moreover, most of the analyses for scheduling, task and hardware allocation and built-in trackers of DFSs (job trackers and task trackers in Hadoop) are executed by the server admin itself and the results are bound over network to have a central analysis. These analyses are made by sorting and defining on different sizes of files using text (or downgraded to text) data [24]. Distributed file systems in general, spend most of the CPU power and electricity to handle heavy Input/Output loads, which makes them less efficient when it works with lots of small volume of data instead of singular big files [25]. For Hadoop and other Distributed File Systems which use blocked data structure, it means more than usual since big files are also divided into small blocks. Considering most of the distributed file systems are not built-in at the OS level and need to have a handful of services to keep running; the unnecessary use of distributed file systems in this sense might result in leakage in energy consumption and execution time for some types of computer jobs.

We gave the priority to two pinpoints in this motivation, first migration of files inside and between those file systems, second analyse type of cpu loads inside the same rack therefore we set our experiments to simulate such scenarios.

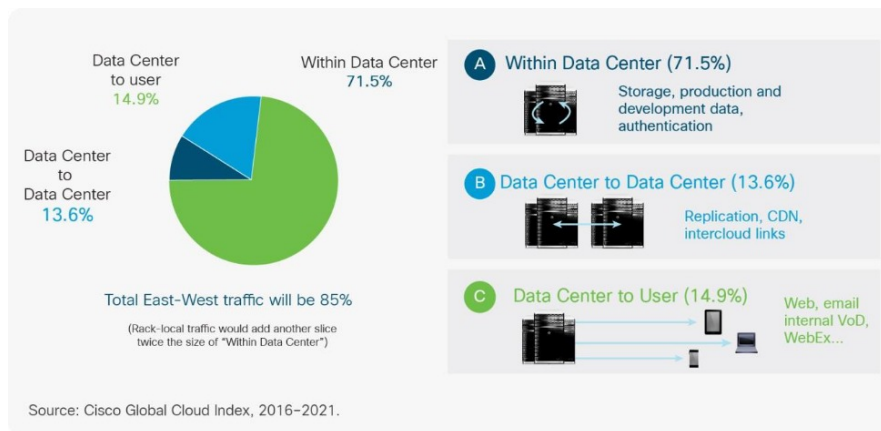


Figure 1.3: Cisco - Global data center traffic estimation by destination in 2021

As another potential of energy misuse, we included I/O related computer jobs such as wordCount, which includes reading, sorting and comparing different numbers of text data. (Figure 3.2) We also ran our experiments on physical machines instead of VMs, to achieve better measurements of energy consumption and not to have a leakage in performance, since it's found that physical machines have significantly better performance than virtual machines[26] [27] and energy measurements on VMs are not as feasible as it is on PMs. [28]

Objectives in this thesis are;

1. Detailed study on a commonly used, open-source Distributed File Systems Architecture (Hadoop Distributed File System)
2. Evaluating the energy consumption of Hadoop Distributed File System and an ext4 non-distributed file system under migration and analysis type of workloads
3. Examining the relationship between energy consumption, execution time, processed file size, and file count.

### 1.3 RESEARCH QUESTIONS - HYPOTHESES

Considering the motivational factors above, our research question mainly focuses on the evaluation of Hadoop Distributed File System against non-distributed ext4 host file system. So, we came up with;

"Does architectural type of the file system affect the execution time and the energy consumption of data replication and data analysis processes inside the cluster?"

Furthermore, We formed our hypotheses as follows;

---

$H_1$  = The architectural type of file system (Distributed - Non Distributed) has an effect on

energy consumption of the data replication and wordCount processes inside the cluster.

---

$H_2$  = The architectural type of file system (Distributed - Non Distributed) has an effect on executing time of data migration and wordCount processes inside the cluster.

---

---

$H_{0_1}$  = The architectural type of file system has no effect on energy consumption of the data migration and wordCount processes inside the cluster.

---

$H_{0_2}$  = The architectural type of file system has no effect on execution time of data migration and wordCount processes inside the cluster.

---

## 2 RELATED WORK

In this section, we have discussed previous research studies related to energy consumption on cloud-related systems and some comparative studies on distributed file systems. Earlier studies were usually focused on different important aspects of distributed storage such as performance, throughput, security, and reliability.

Bai et al. studied several distributed file systems conceptually and compared them to each other considering some core features [29]. They compared Hadoop v0.21.0, Moose FS v1.6.19 and Lustre v1.8.4. Their study showed that Hadoop and MooseFS are easier to install than Lustre FS. Lustre doesn't support automatic data balance and it has no data replication factor in it while Hadoop has a default of 2 (configurable) and MooseFS also has 1 replica of a file by default hence they all achieve data redundancy by their own specifications. All three DFSs don't support single point of failure. MooseFS and Lustre support snapshots to be used for safety or migrational purposes. All three DFSs support scaling out and configurable data blocks. Hadoop, MooseFS, and Lustre can be accessed via a Command Line Interface (CLI) and Portable Operating System Interface for Unix (POSIX) while Hadoop and MooseFS also support Web Graphical User Interface (GUI).

Shirinbab et al. evaluated different distributed storage systems considering read/write/delete performances and the recovery time of the system when a node goes down [30]. They picked two unstructured storage systems, Compuverde Unstructured and Openstack's Swift and two structured storage systems, Compuverde Structured, and Gluster FS. Two of them use Distributed Hash Tables (DHT) to keep track of how data is distributed and the other two use multicasting to access the stored data. They installed those systems on the same hardware setup. The authors also observed that the architectural decision of using DHT or multicasting has different negative factors. DHT causes processing load while multicasting causes network overload. As performance, the authors achieved that Compuverde outperforms the other two when the number of client requests increases. For replication and recovering also, Compuverde acts faster than the other two systems. It is also observed that during the self-healing tests of systems, Gluster didn't exceed 50% even for high workloads, which may refer to a domestic bottleneck in it.

Zhou et al. proposed a Virtualized Hybrid Distributed File System for large scale data storage solutions. It combines the fault tolerance ability of Google File System and manageabil-



ity, scalability and throughput advantages of Peer to Peer (P2P) network. [31] They aimed to attenuate required space and infrastructure for a cloud system that uses distributed file systems and lower the cost of data transfer using P2P network method. For security, they used layered virtualization between file owner and storage service provider. Their results show that a distributed file system combined with a generic P2P network can help enterprises and individuals to save energy, storage space, and cost during file transfers.

Yazd et al. proposed a new method for data replication in Hadoop systems, aiming to have a better energy efficiency[32]. Their study is called Mirrored Data Block Replication Policy, is based on Covering Subset method, in which storage nodes with active/necessary data in a cluster is kept active, while the nodes that includes replicas of that data are passive but Covering Subset method doesn't include a policy for data replication method and they are randomly replicated in nodes. Hence authors proposed mirrored replication model, in which passive nodes of replicas will be created as mirrored copies of active nodes (Figure 2.1). Thus when new similar tasks arrive and a new data node is necessary, data demand can be fulfilled with a minimum number of re-activated data nodes. They simulated both Covering Subset and Mirrored Data Replication methods on Matlab, with a number of data blocks between 3000 to 5000 and no common data block between tasks. The simulation showed that while the number of nodes increases, efficiency of the proposed method also follows the increasing trend. Because the likeliness of sharing data blocks on similar tasks is less in Covering Subset than Mirrored Data Block Replication. With a higher number of resources it is expected to have better results with the method.

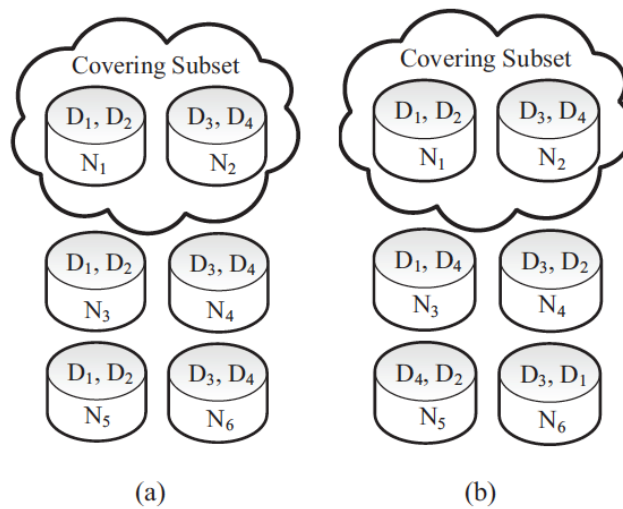


Figure 2.1: a) Mirrored Data Block Replication b) Covering Subset Method

Cloud providers apply various methods of Load Balancing to achieve better distribution of work, in order to avoid overloading of some of the resources. The rest of the resources that reside in low level of utilization may consume a noteworthy amount of power without doing any intensive tasks. Task scheduling and load balancing can be done either dynamically or

statically. Both contribute to better resource utilization, lower response time or less energy consumption. Wirtz et al. experimented MapReduce workloads under different scenarios with different benchmarking tools; a Matrix Multiplication app, CloudBurst and Integer Sort with varying numbers of CPU cores and processor frequencies[33]. To be able to measure the direct cost of an execution, they used induced power instead of overall system usage, which also considers the idle power as 0; the contrary of which has been shown with different studies and reports. Duan et al. [34] and Barroso et al. [35] concluded that energy usage of a CPU is not proportional to its utilization level. They also observed that the most energy-efficient way for the CPUs to work, is to use the highest level of utilization(Figure 2.2). Duan et al. have also showed that having a CPU in the sleeping state for a shorter time than its wake up latency, causes a negative energy efficiency, because the energy consumption for waking up is higher than the sum of its idle status. Although the results of Wirtz’s experiments are not directly proportional to the workloads and cores, they show that with an astute scheduling it is highly possible to achieve better energy efficiency. They also excluded the energy consumption of Name Node, which has a potential to have a difference in energy consumption of such infrastructure under the tests.

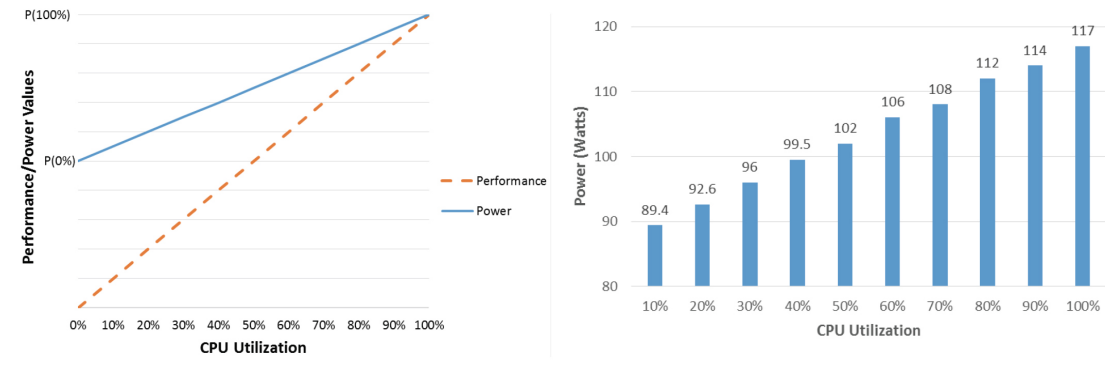


Figure 2.2: CPU Utilization - Power Usage

Mashayekhy et al. proposed an algorithm named Energy-Aware MapReduce Scheduling Algorithm (EMRSA) on Hadoop, declaring it is the first task placement algorithm designed to minimize the energy consumption. Authors modeled the problem of task scheduling on MapReduce jobs to come up with a greedy algorithm. Then they compared their results to two different algorithms, OPT as an optimal solution with minimized energy consumption algorithm and an MSPAN as optimal solution with minimized makespan of the job. [36] Their results show that EMRSA has 40% and OPT has 49% less energy usage than MSPAN for small-scale workloads between 20 to 30 Map and Reduce jobs. They claim that OPT has a lower energy usage but it also has a slow execution time, and for big data jobs it is not practical because of its long runtimes. For large-scale workloads with between 160-200 Map tasks and 200-500 Reduce tasks, EMRSA compared to greedy MSPAN has an average of 32% less energy usage with an opportunity to achieve better results when Reduce tasks are less than Map tasks. (See Figure 2.3 and Table 2.1)

Workload	Map Count	Reduce Count
I	160	200
II	300	200
III	250	400
IV	200	500

Table 2.1: Workloads

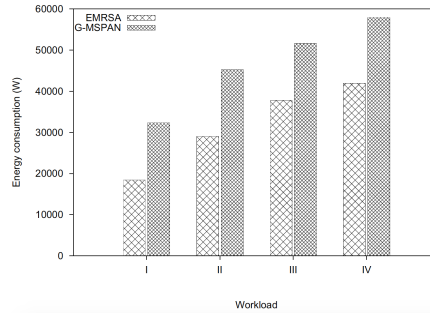


Figure 2.3: Energy Consumption

Goiri et al. presented GreenHadoop [37], a new bargain to be used in cloud environments working with fully or partially renewable energy sources. Their study aims to make the cloud infrastructure work with renewable energy as much as possible and to do so the authors developed a new framework for Hadoop clusters which predicts upcoming energy usage of workload and the amount of solar energy which will be available, then distributes the tasks based on those predictions. They also prioritize jobs and set low priority jobs to run in "green hours" with renewable energy sources. When the data center needs to run with electrical grid power, their framework selects the times when grid energy is cheap. Their results show that up to %31 of GreenHadoop systems' can be obtained from renewable sources and it can save up to %39 of the energy cost compared to base Hadoop systems. (Figure 2.4

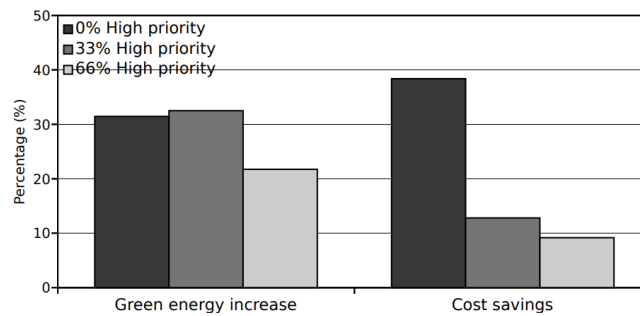


Figure 2.4: Fraction of Green Energy Cost and Grid Electric Cost of Prioritized Jobs

Next to the opportunities to achieve better energy-aware distributed storage system by improving task scheduling, VM allocation, data replication, etc. for large clusters there is another bottleneck, network energy consumption. Cisco depicts that global IP traffic is already doubled in the last 2 years and it will be increasing with the same speed within the near future [23]. While overall network usage is increasing, machine-to-machine (M2M) connection is also having a bigger proportion, the estimated percentage of M2M connections will go from 34% in 2017 to 51% in 2022(Figure 2.5). Over 3/4 of the global network usage is between Data Centers, which points out where to look for the solutions for a global effect. Network usage

in Hadoop, for example, is 12 % of overall consumption on a full load and it is not proportional to network utilization level, meaning that its proportion on a low-utilization system is higher than this. Fischer e Silva proposed a set of techniques for energy-efficiently optimized networks for Hadoop systems and similar distributed storage solutions, called E-EON [38]. The author experimented different usages of Energy Efficient Ethernet (EEE) and Packet Coalescing and combining them with existing technologies on most network devices such as Active Queue Management (AQM) and Explicit Congestion Notification (ECN) to achieve less on/off overheads on ethernet equipment. The author obtained promising results on real experiments and simulations, with combined techniques it is possible to achieve 70% less energy consumption on network devices with better cluster throughput and lower latency, with no loss of performance. Hence this research also depends on special hardware, the author also suggests network equipment vendors to implement configurable Packet Coalescing and network administrators to implement suggested techniques based on their own system's latency requirements.

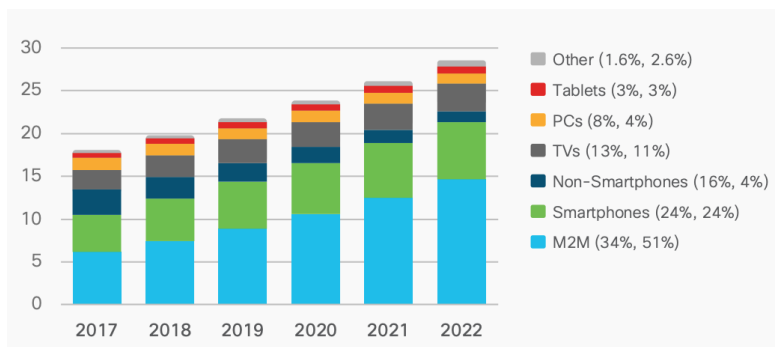


Figure 2.5: Global devices and connections growth

To evaluate different installation conditions of distributed file systems and their effects to performance and energy consumption, Feller et al. evaluated different Hadoop models; traditional centralized computing and data services and a separated model in which data and computing services were separated [26]. They obtained that virtualized and separated forms of the cluster could be useful for flexibility hence the impact is not considerably big. Data locality doesn't also have a large impact on such environments with high-level virtualization and with advanced networks. They achieved in their experiments that performance on physical cluster is significantly better than it is on virtualized clusters and overall power consumption is heavily related to application type and characteristics.

Malik et al. studied the characterization of Hadoop Map/Reduce workloads and their impact on energy efficiency and performance. They also analysed configuration parameters of Hadoop and experimented them with different settings on similar infrastructure and workloads besides including the role of the processing unit type. They did their experiments with both high-frequency/high-performance processors and low-frequency/energy-efficient processors for similar workloads and examined the results [39]. Their results show that the increasing the number of mapping jobs with an increasing number of cores cause a better en-

ergy efficiency hence this efficiency of Hadoop is also highly dependent on its initial configuration, especially on block size, which was realized that it is not optimal for most of the cases. Their study also showed that with the increasing number of mapper jobs, CPU frequency gets less important, since mappers can run simultaneously on the same cores, so using low-frequency processors for mappers increases energy efficiency. Another result from their study is that input data size for computing related jobs is not as important as it is for I/O related jobs. Shortly, this study showed that Hadoop needs a special configuration for each workload. Intense mapping workloads can be run by low-frequency processors and even in the case when the number of processors is limited, Hadoop can still be configured to be energy-efficient by fine-tuning the processors and Hadoop's block size.

Other than those, there are studies which aim to evaluate and compare different distributed file systems considering energy consumption to help to have a more energy-aware choice in certain jobs. Kolli et al. evaluated different distributed storage systems, namely GlusterFS and Compuverde FS, and compared them considering execution time and energy consumption on several workloads. PowerAPI [22] was used for energy evaluation and SPECsfs 2014 was used for generating synthetic workload on Gluster and Compuverde distributed file systems. It was observed that Compuverde consumes more energy than Gluster on a similar workload. With the increasing workloads Compuverde kept consuming more energy than Gluster. Authors assumed that it is caused by the extra features and a higher number of process count of Compuverde, which don't exist in Gluster FS.

Bourdon et al. compared energy consumption of different programming languages for the same algorithms and their impact on the overall system and achieved that interpreted languages consume more energy than compiled ones[40]. They also experimented further in a different study, to achieve the effects of the choice of programming language, algorithm and platform on the energy consumption, since most of the cloud providers use interpreted languages in the server side. They used PowerAPI [22] tool for the measurements of CPU, network, etc.

Besides these studies on the field, there are more to achieve better energy management on cloud base systems. Jalali et al. studied Fog Computing and came up with some scenarios [41] with Internet of Things (IoT) applications in which Fog Computing can be energy-efficient. Duan et al. [34] used dynamic prediction algorithms to foresee idle intervals and achieve energy consumption. Colmant et al. studied power estimation algorithms on VMs and came up with a middleware called BitWatts, an extensive development of PowerAPI. Their experiments showed that BitWatts can make consistent predictions with convenient power models of hardware[42]. Hai Zhu et al. proposed a new algorithm to dynamically optimize a cloud-based system to meet the SLA. They derived two new optimizations, one is on VM deployment considering idle intervals and one dynamic voltage power adjustment to reduce the energy usage of an application. They achieved promising results in some of the experiments, comparing their system to the generic ones[43]. Cheng et al. proposed an energy efficiency task assignment algorithm using previous work on Dynamic Voltage and Frequency Scaling on heterogeneous Hadoop Clusters. They run experiments on a Hadoop Cluster with a different type of hardware. They compared their results to different scheduler algorithms, named Fair Scheduler[44] and Tarazu with an energy saving of 23 % and 17% [45] respectively.

## 3 PROCEDURE

To simulate an in-cluster work of data replication and analysis jobs, we designed our experiments to run on two identical Hadoop clusters. To make it closer to real life scenarios, instead of using benchmark tools to create a synthetic workload, our tasks are to copy various sizes and numbers of files to and from both file systems, create various numbers of directories and doing analyses on various sizes of text files both on host file system and Hadoop Distributed File System. During the experiments, we measured energy consumption of each task. We also examined some scenarios to cover a whole process, such as comparing the energy consumption and execution time of a process on distributed file system against its equivalent by adding up energy consumption and execution time of locating the files to host file system, executing the job and relocating them back to the distributed file system.

### 3.1 METHODS USED

Migration of a Virtual Machine can be handled in quite a few ways, hence all consist of copying of many files; creating a mirror of the VM image and then transferring that image file to another cluster. A non-live migration, referred also as static, cold or dead is the relocation of a VM while it is off. Firstly, files are mirrored inside the same disk and then copied to the target location(Figure 3.1). Live migration is relocating the VM while it is up and running. In this procedure, states and memory pages are also mirrored inside the disk and then copied to the target location[46]. These states are saved in chunks of small files for each stateful variable. Relocation of a VM is a migration with consideration of local files, this time various numbers of small and big files are first mirrored inside the cluster and then, that mirror is sent to another storage area. For migration in general, the control of the task (application), data to process or both can be moved to a target location[47]. [48]

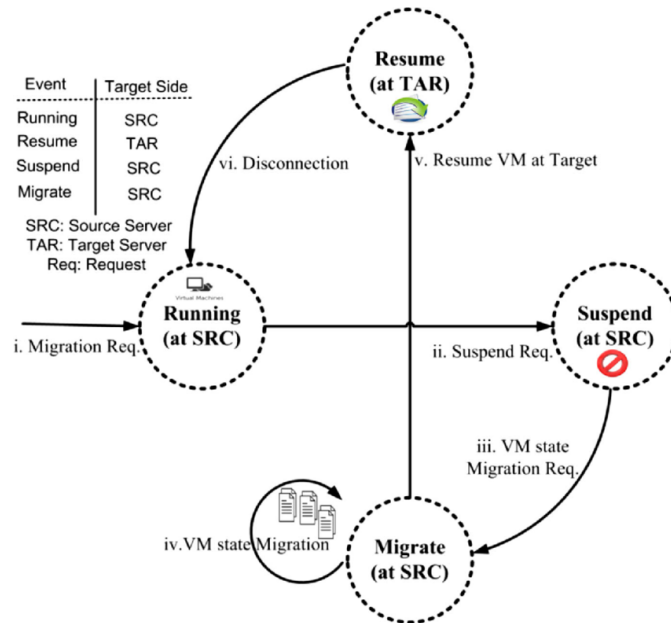


Figure 3.1: Non-Live VM Migration

We aimed to evaluate the in-cluster jobs considering their energy consumption and execution time values, to decide if some of the jobs with distributed file structures can be executed in a non-distributed environment to save energy. These include pre-migration phase of data and VM images, preparation and migration of HDFS snapshots which are mostly small files transferred in a distributed environment, migration of the cluster to a new rack, etc. To simulate real life alike workloads, we created two test structures, a data migration process and an analysis process. We ran our experiments with data migration on the host file system using Apache Commons IO, a Java I/O library capsulation created by Apache Software Foundation including the most common functionality in an easy to use library [49], which has been developed since 2013. And for the analysis phase of our tests, we used wordCount which has also categorized to be one of the most common jobs which run on Hadoop configurations [39] and other distributed file systems, in which input files are taken in to memory by line, each line is sent to individual mapper instances, mapped with key / value pairs, sorted considering their identicalities, reduced into key / value pairs and printed the output with numbers of instances. (Figure 3.2) We executed our experiments on different sizes and numbers of files, from 5 MB to 48 GB singular files for migration and from 5 MB to 1 GB of unstructured text files for analysis. We used the same file size and count on host file system to not to discard the effects of the blocked structure of Hadoop DFS.

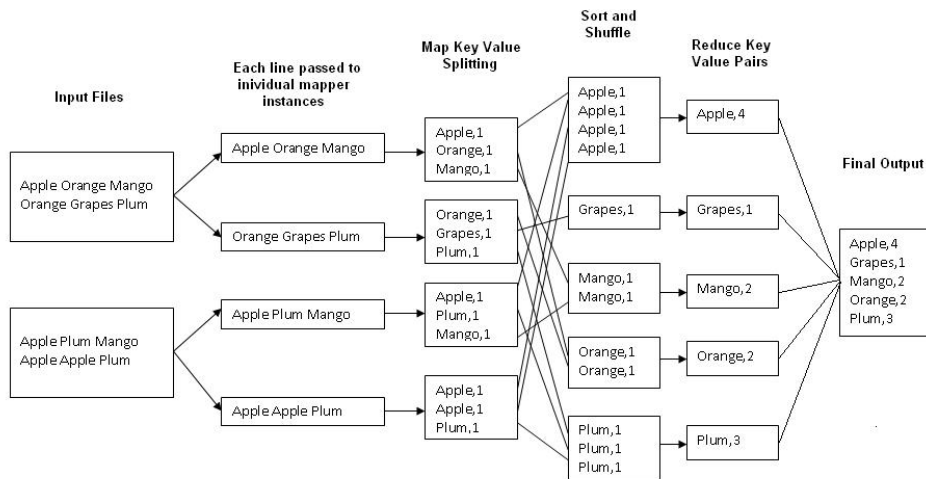


Figure 3.2: wordCount Process Flow Diagram

### 3.1.1 POWERAPI

For energy measurements, we used Power Application Programming Interface (PowerAPI), an OS level system monitoring toolkit to build software-related power measurements. PowerAPI supports a wide range of sensors including physical meters, processor interfaces, hardware counters, and OS counters and stands between OS and Applications (Figure 3.3). [22] It is also used in a wide range of tools and researches; mostly for unit testing, monitoring, creating estimations for dynamic scheduling. [18] [19] [20] [21] [22] PowerAPI takes the raw sensor data of hardware below OS and formulates that raw data using OS-free equations for utilization and configuration of CPU and other components (HDD, GPU, Network Cards, etc). It follows the applications to see what type of utilizations they are activating and formulates an output energy measurement for a given frequency in milliseconds.

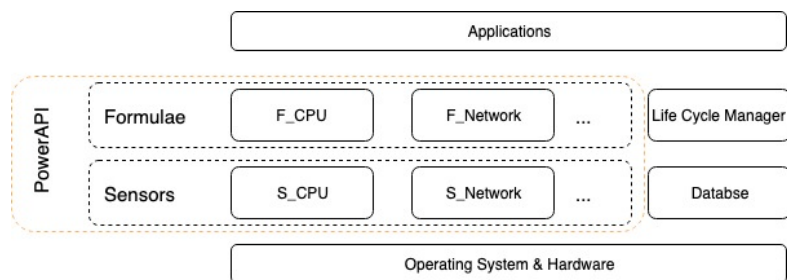


Figure 3.3: PowerAPI Architecture



### 3.2 TEST ENVIRONMENT

We prepared an environment of two Hadoop Clusters (Hadoop v2.9.2) on Linux Ubuntu 18.04 with two data nodes each and run the experiments on a single node and also on both of them to achieve closer results to real-life scenarios. Since Hadoop promises to run smoothly on commodity hardware, we used a commodity high-performance CPU for the experiments. Our cluster runs on ;

I	Intel I7 6th Generation CPU
II	8 x 4 GB DDR4 RAM
III	4 x 1 TB 7200 RPM Western Digital SATA HDD (Sata III)
IV	MSI x99A Workstation Motherboard
V	Killer Network E2400 High-Performance Network Card
VI	Dell RAID Controller H700

Table 3.1: Experimental Setup

### 3.3 TEST STRUCTURE

Our tests were structured to achieve a close scenario to daily jobs on a cluster with a distributed file system and focused on two phases; copying files (Test 1) and analysing files (Test 2) to be able to analyse the necessity of using distributed file system architecture considering execution time and overall energy consumption. We ran our experiments to migrate files inside both HDFS and host file system and also between file systems to take switching file systems for some workloads or file sizes in action and also to consider executing some of the workloads on the non-distributed structure and carry it back to its place. We used Python 3.6 and Java 11 for scripting the mirroring behaviour and Java 11 for analysing behaviour. For reading files we used wordCount with MapReduce job to simulate an exclusive analysis of a file with I/O for every word and we ran copying jobs both inside and between file systems, to simulate copying phases of inside cluster jobs. During the experiments, we recorded energy consumption measurements of each task, starting-finish timestamps, and process ids. Since Hadoop runs its own services independent from the OS it is running on, idle status and ready-to-go status of Hadoop were subtracted from related sums of the results. For both wordCount and Copying jobs, we ran it on different file sizes to cover more possible scenarios. To be able to cover the most of possible scenarios with less tolerance, we ran the test multiple times for small, medium and large-sized files. (Table : 3.2, Table : 3.3) Test 1 takes a file, creates a new folder in target directory copies to that folder while PowerAPI measures the energy consumption of the whole process with its own arguments(Figure 3.4).

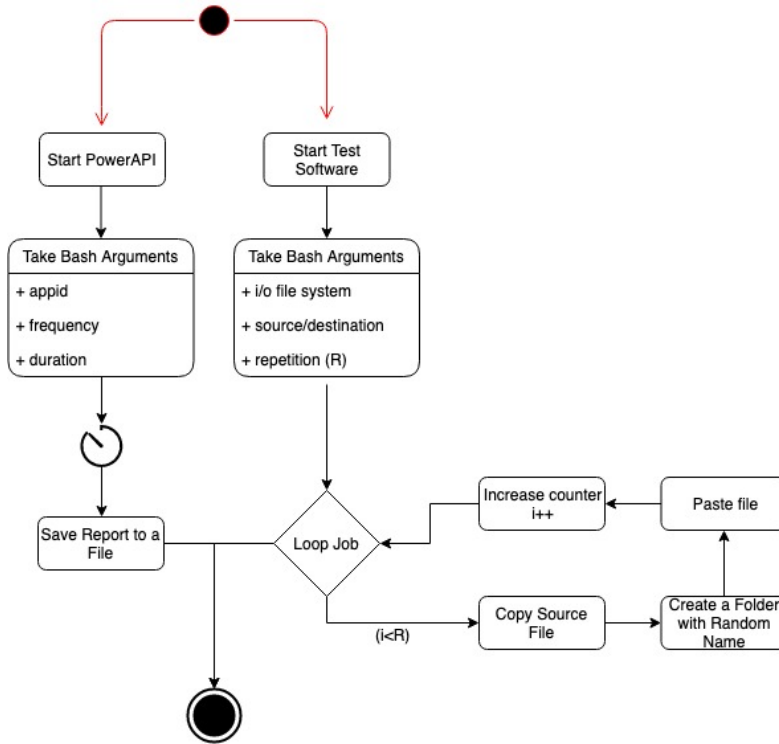


Figure 3.4: Test 1 Activity Diagram

The largest file size we used was a VM image of 48 GB, which is one of the most common sizes for a VM migration based on collected data over 400,000 different virtual machines. (Figure 3.5 [50]).

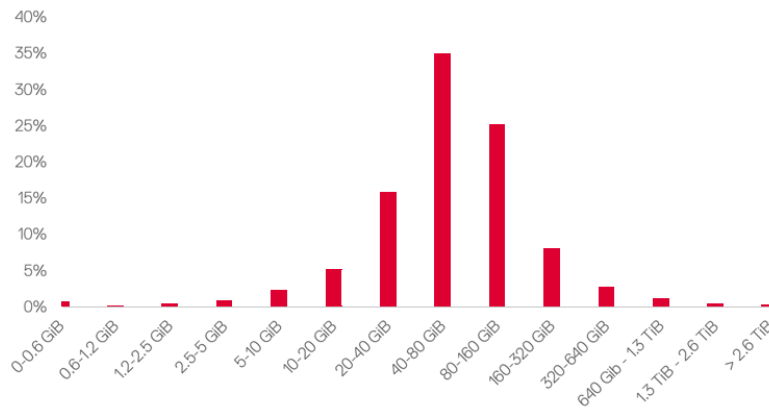


Figure 3.5: Distribution of VM sizes based on AutoSupport Data

Test 2 is covering an analysis of a file, which covers most of the in-cluster jobs of a Hadoop server, including arranging of memory pages, most of the in-cluster data loads, analysing the

data before and during migration. Hadoop runs on a MapReduce framework and it's one of the most common types of workloads on a Hadoop Cluster and also common for different cloud systems. We ran Test 2 also on different text files with various sizes, from 5 MB to 1 GB on both HDFS and Host file system. During the experiments, we measured the energy consumption using PowerAPI. (Figure 3.6)

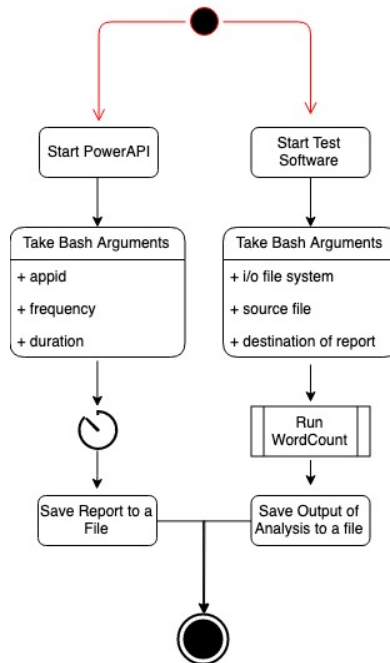


Figure 3.6: Test 2 Activity Diagram

PowerAPI is not triggered inside the test software but from outside. Pseudo codes for the

test software and PowerAPI script are as follows(Code 1, Code 2).

---

**Algorithm 1:** Test Case 1 & 2 Pseudo Code

---

```

Load config (I/O, directories, files, repetition count, Task);
Load tasks [Copy, Move, Delete, Create, Rename, wordCount];
Load definitions;
Load Task.Select;
for  $i = 0; i < repeat.count; i = i + 1$  do
    print start messages;
    print initial commands;
    run Task.Select
    if Error then
        | Quit job;
        | Print Error Message;
    end
    Print completion message for repetition  $i$ 
end
Print complete message for test with total count.

```

---



---

**Algorithm 2:** PowerAPI Script Pseudo Code

---

```

Load config (Frequency, Duration, Appid, Formula, Output.File, );
Load libraries;
for  $i = Dur; i > 0; Countdown\ i = i - Freq$  do
    print app id;
    print cpu id ;
    print epoch timestamp;
    if Error then
        | Quit job;
        | Print Error Message;
    end
    Print energy consumption (mW);
end
Print complete message;
Shut Down PowerAPI

```

---

### 3.4 ENERGY CONSUMPTION

- Mean of Energy Consumption The mean of energy consumption here is the power used by the process during its execution time. It is calculated by adding up the power consumption values and dividing them by the number of measurements.

The mathematical formula for the mean of energy consumption is;

$$Mean, \bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{\sum_{x_1}^{x_n}}{n}$$

where  $x_1, x_2, x_n$  are the energy consumption measurements and  $n$  is the total number of measurements.

- **Median of Energy Consumption** The median of energy consumption values here is the value in the middle of the array of measurements. In our experiments, it shows the middle point energy consumption of an application with repetitions and depicts the general behaviour of the file system but it needs to be analysed with standard deviation to make real conclusions. During the measurements, PowerAPI follows for the Linux kernel for the active cores, microprocessors and the triggering process on the OS side. If the process is not active when PowerAPI gets the value, it basically gets a value of 0 mW, so median value is very important for the measurements with PowerAPI because those 0 mW measurements reduce the obtained mean value.

The mathematical formula for the median value is;  
If the total number of the measurements are odd :

$$Median = \left(\frac{n+1}{2}\right)^{th} term$$

If the total number of the measurements are even :

$$Median = \frac{\left(\frac{n}{2}\right)^{th} term + \left(\frac{n}{2} + 1\right)^{th} term}{2}$$

- **Standard Deviation of Energy Consumption** Standard deviation is a measure which shows the spread of the data from the mean value. The standard deviation of energy consumption shows how much the samples differ from the mean value of all the measurement gathered for each process/test case. This shows the fluctuation of power used by CPU and can be useful for the measurements of utilization levels.

The mathematical formula for the standard deviation is:

$$StandardDeviations = \sqrt{\frac{\sum(x_i - \bar{x})}{n-1}}$$

Where  $x_i$  is each power measurement gathered,  $\bar{x}$  is the mean of power measurements and  $n$  is the total number of measurements in the dataset. Standard deviation tells us about the energy-consuming behaviour of an application on file systems and the effects of other factors.

## 3.5 PRE-TEST PROCEDURE

### 3.5.1 HADOOP INSTALLATION

Hadoop needs to run on an OS, like most of the other distributed file system frameworks. We used Ubuntu 18.04 for the installation. Hadoop follows a through installation for each

cluster. Numbers, IPs, Ports can be configured in Hadoop's configuration files. Because of the nature of Hadoop's commodity hardware and usage, a new cluster can be added to a network anytime by editing and sourcing the files.

From now on, most of the commands will be run in the terminal and codes with % will be run with admin rights and codes with \$ will be run without.

To be able to install Hadoop, we need to have Java installed in our machine. First we remove existing java installation, to have a fresh start for our cluster. (This step is not a must.)

```
1 apt purge openjdk*
```

We install java from the existing library with

```
1 % add-apt-repository -y ppa:webupd8team/java
2 % apt update
3 % apt install -y oracle-java8-installer
```

After installation occurs, we can check our java installation with

```
1 $ java -version
```

If we see our Java distribution version, we have installed Java perfectly (Figure : 3.7)

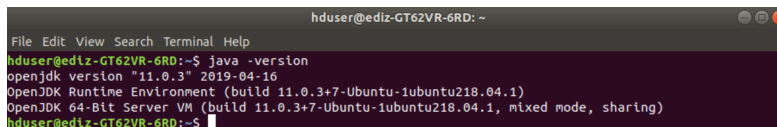
A terminal window screenshot showing the command 'java -version' and its output. The output indicates that OpenJDK version 11.0.3 is installed, which is a 64-bit server VM. The terminal title is 'hduser@ediz-GT62VR-6RD: ~' and the prompt is 'hduser@ediz-GT62VR-6RD:~\$'.

Figure 3.7: Java Version Check

After Java installation, we should create a JAVA\_HOME variable to make Hadoop be able to find Java executables. To do this, we should edit /etc/profile and then update the file with

```
1 % echo "export JAVA_HOME=/usr" >> /etc/profile
2 % source /etc/profile
```

After this step we need to disable IPv6, since Hadoop only supports IPv4. It's done by editing the file /etc/sysctl.conf in Ubuntu. We need to append the following part to the file and reboot the system for it to take effect.

Open the file with

```
1 % nano /etc/sysctl.conf
```

Append this to the file

```
1 net.ipv6.conf.all.disable_ipv6 = 1
2 net.ipv6.conf.default.disable_ipv6 = 1
3 net.ipv6.conf.lo.disable_ipv6 = 1
```

Then reboot the system. To use hadoop with a privileged profile, we'll create a new user with a new usergroup. Users from other clusters will share the same group. To create a usergroup named hadoopgroup with a user in it,

```
1 % addgroup hadoopgroup
2 % adduser -ingroup hadoopgroup hadoopuser
```

Now we will install SSH and configure it to enable accessing the machine without a prompt.

```
1 % apt install ssh
2 % systemctl enable ssh
3 % systemctl start ssh
```

To configure the new user with the SSH, we will switch to that user with

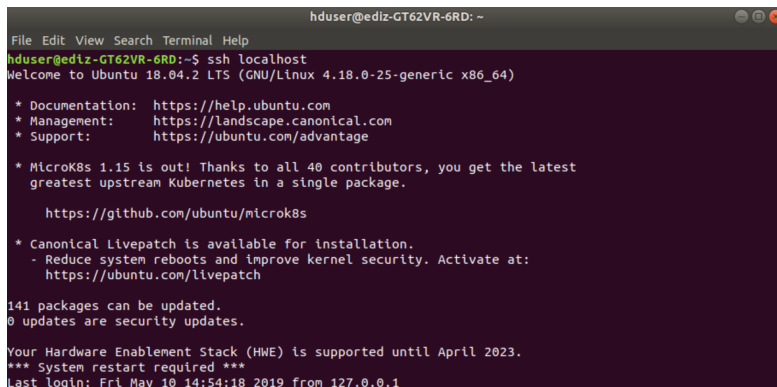
```
1 % su - hadoopuser
```

Now we will create a SSH key and authorize the user.

```
1 $ ssh-keygen -t rsa -P ""
2 $ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
3 $ chmod 600 ~/.ssh/authorized_keys
4 $ ssh-copy-id -i ~/.ssh/id_rsa.pub localhost
```

To check connection, this command should work without a prompt for a password and show an output of OS (Figure : 3.8)

```
1 $ssh localhost
```



```
hduser@ediz-GT62VR-6RD: ~
File Edit View Search Terminal Help
hduser@ediz-GT62VR-6RD:~$ ssh localhost
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.18.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * MicroK8s 1.15 is out! Thanks to all 40 contributors, you get the latest
   greatest upstream Kubernetes in a single package.
   https://github.com/ubuntu/microk8s

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
   https://ubuntu.com/livepatch

141 packages can be updated.
0 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
*** System restart required ***
Last login: Fri May 10 14:54:18 2019 from 127.0.0.1
```

Figure 3.8: SSH to localhost on Ubuntu

Now we can proceed to installation of Hadoop. After switching back to the administrator account we will download the version from Hadoop's website and locate to the encapsulating folder and extract the compressed files.

```
1 %su - administrator
2 % wget http://it.apache.contactlab.it/hadoop/common/hadoop-2.9.2/hadoop
   -2.9.2.tar.gz
3 % tar xzf hadoop-2.9.2.tar.gz
4 /usr/local/hadoop
```

Then we will move the extracted files to /usr/local/ directory of Ubuntu and then link the directory to /usr/local/hadoop for cleaner installation and management.

```
1 % mv hadoop-2.9.2 /usr/local
2 % ln -sf /usr/local/hadoop-2.9.2/
```

We need to set chown variables for hadoopuser user while we are in the administrator account.

```
1 % chown -R hadoopuser:hadoopgroup /usr/local/hadoop-2.9.2/
```

Now we will configure new environment variables for Hadoop by appending Hadoop configuration to some files. First, to `/.bashrc` file append these.

```
1 # Hadoop config
2 export HADOOP_PREFIX=/usr/local/hadoop
3 export HADOOP_HOME=/usr/local/hadoop
4 export HADOOP_MAPRED_HOME=${HADOOP_HOME}
5 export HADOOP_COMMON_HOME=${HADOOP_HOME}
6 export HADOOP_HDFS_HOME=${HADOOP_HOME}
7 export YARN_HOME=${HADOOP_HOME}
8 export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
9 # Native path
10 export HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
11 export HADOOP_OPTS="-Djava.library.path=${HADOOP_PREFIX}/lib/native"
12 # Java path
13 export JAVA_HOME="/usr"
14 # OS path
15 export PATH=$PATH:$HADOOP
```

And then to the `hadoop-env.sh` file in `/usr/local/hadoop/etc/hadoop/` we need to append JAVA variable for Hadoop.

```
1 export JAVA_HOME="/usr"
```

Now we can proceed to edit Hadoop's configuration files located in the `/usr/local/hadoop/etc/hadoop` folder. Files are

- `core-site.xml`
- `hdfs-site.xml`
- `mapred-site.xml` (`mapred-site.xml.template` in some versions)
- `yarn-site.xml`

For `core-site.xml` we need to append

```
1 <configuration>
2 <property>
3   <name>fs.default.name</name>
4   <value>hdfs://localhost:9000</value>
5 </property>
6 </configuration>
```

For `hdfs-site.xml` we need to append

```
1 <configuration>
2 <property>
3   <name>dfs.replication</name>
4   <value>1</value>
5 </property>
6
7 <property>
```



```

8 <name>dfs.name.dir</name>
9 <value>file:/usr/local/hadoop/hadoopdata/hdfs/namenode</value>
10 </property>
11
12 <property>
13 <name>dfs.data.dir</name>
14 <value>file:/usr/local/hadoop/hadoopdata/hdfs/datanode</value>
15 </property>
16 </configuration>

```

For mapred-site.xml we need to append

```

1 <configuration>
2 <property>
3 <name>mapreduce.framework.name</name>
4 <value>yarn</value>
5 </property>
6 </configuration>

```

For yarn-site.xml we need to append

```

1 <configuration>
2 <property>
3 <name>yarn.nodemanager.aux-services</name>
4 <value>mapreduce_shuffle</value>
5 </property>
6 </configuration>

```

After these steps, Hadoop should be accessible via terminal. We will start with formatting our first namenode, which is responsible for file system, tracking, replication and formation of blocks with the following command. <sup>1</sup>

```

1 $ hdfs namenode -format

```

This command should give a crowded output and we should check for this line to make sure it successfully formatted the namenode.

```

1 INFO common.Storage: Storage directory /usr/local/hadoop/hadoopdata/hdfs/
  namenode has been successfully formatted.

```

After this, we can start hadoop services by running

```

1 $ start-dfs.sh
2 $ start-yarn.sh

```

To check the situation of the Hadoop services anytime, we can use the following command followed by an output like follows.

```

1 $ jps
1 hadoopuser@farlands ~ $ jps
2 26899 Jps
3 26216 SecondaryNameNode

```

<sup>1</sup>Command for Hadoop DFS is hdfs in version 2.9.2, it may change for newer or older versions. Please check documentation for your release on Apache Hadoop's website. [9]

```

4 25912 NameNode
5 26041 DataNode
6 26378 ResourceManager
7 26494 NodeManager

```

To stop hadoop services for further configuration, we can use the following commands.

```

1 $ stop-dfs.sh
2 $ stop-yarn.sh

```

After starting Hadoop services, we can use Hadoop's web interface via the link "http://localhost:50070/" in a web browser (Figure 3.9). It shows the number of alive and dead nodes with data consumption of each, individual Http addresses of nodes, the whole filesystem and folder architecture, full log for services and situations of snapshots, maintenance, migration, etc.

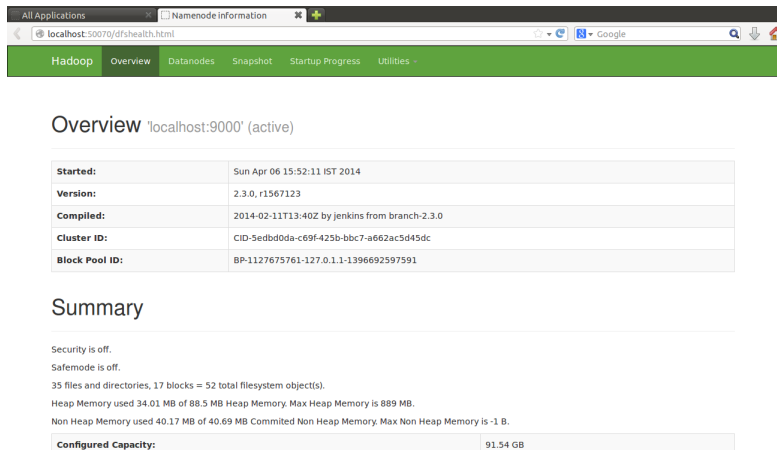


Figure 3.9: Hadoop Web Interface

An example of a full log from Hadoop Web UI is,

```

1 /*****
2 STARTUP_MSG: Starting DataNode
3 STARTUP_MSG:   host = ediz-GT62VR-6RD/127.0.1.1
4 STARTUP_MSG:   args = []
5 STARTUP_MSG:   version = 2.9.2
6 STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop
   -2.9.2/share/hadoop/common/lib/jettison-1.1.jar:/usr/local/hadoop
   -2.9.2/share/hadoop/common/lib/jetty-util-6.1.26.jar:/usr/local/hadoop
   -2.9.2/share/hadoop/common/lib/apacheds-i18n-2.0.0-M15.jar:/usr/local/
   hadoop-2.9.2/share/hadoop/common/lib/jcip-annotations-1.0-1.jar:/usr/
   local/hadoop-2.9.2/share/hadoop/common/lib/log4j-1.2.17.jar:/usr/local
   /hadoop-2.9.2/share/hadoop/common/lib/jsr305-3.0.0.jar:/usr/local/
   hadoop-2.9.2/share/hadoop/common/lib/commons-io-2.4.jar:/usr/local/
   hadoop-2.9.2/share/hadoop/common/lib/netty-3.6.2.Final.jar ...

```

### 3.5.2 POWERAPI SETUP

PowerAPI can be downloaded from the links on GitHub, the compiling method may differ on different platforms. You can find links and basics on the website [22]. For example, to compile PowerAPI using Maven, following command should be used in it's parent folder.

```
1 cd $powerapi_directory
2 mvn install
```

By default it comes with all modules installed, pom.xml file is for further configuration. After compiling the code, we can run PowerAPI from it's parent folder which includes build files and bin folder with the following example command.

```
1 ./bin/powerapi modules procfs-cpu-simple monitor --frequency 1000 --apps
   java --agg max --console duration 30
```

Basic arguments are;

- `-procfs-cpu-simple monitor` : formulated sensor monitor of cpus
- `-frequency` is frequency of measurements in milliseconds
- `-apps` (or `-pids`) is the apps we focus on
- `-agg` is the measurements we are having (Min, Max, Average, Standard Deviation, Coefficient of Variation)
- `-console duration` : duration of measurements in seconds

An example output for measuring the power consumption of Java apps looks like follows.

```
1 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316594148;targets
   =java;devices=cpu;power=462.26415094339626 mW
2 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316595155;targets
   =java;devices=cpu;power=0.0 mW
3 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316596165;targets
   =java;devices=cpu;power=9464.547677261613 mW
4 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316597155;targets
   =java;devices=cpu;power=18611.182519280206 mW
5 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316598166;targets
   =java;devices=cpu;power=9430.188679245284 mW
6 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316599165;targets
   =java;devices=cpu;power=3244.010088272383 mW
7 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316600166;targets
   =java;devices=cpu;power=1484.8484848484848 mW
8 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316601165;targets
   =java;devices=cpu;power=3074.144486692015 mW
9 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316602165;targets
   =java;devices=cpu;power=0.0 mW
10 muid=284af255-0cea-47a8-a36b-7e2208843d86;timestamp=1557316603165;targets
    =java;devices=cpu;power=4881.693648816937 mW
11 PowerAPI is shutting down ...
```

### 3.5.3 TEST GENERATION

Our test scenarios were made to cover a set of scenarios, mainly to examine if some in-cluster workloads could be done outside the distributed file structure, if distributed structure

is not necessary or it's not a time restricted job. This covers the prediction algorithms, most of built-in and 3rd party tasks which somehow involve data multiplication, pre-migration mirrors of large and small files, replication of data and metadata, snapshot migrations for scheduling and monitoring the clusters, other data transfers inside the rack without a need of a distributed file structure to prepare the data to be used distributively by relocating it to the related cluster. We stopped all of the background services to prevent from having CPU or applications busy by exterior factors.

File Size (Cat)	Count
5 MB (Small)	100
20 MB (Small)	100
100 MB (Small)	100
512 MB (Medium)	50
1 GB (Medium)	50
2 GB (Medium)	30
48 GB (Large)	1

Table 3.2: Test 1 Cases

File Size (Cat)	Count
5 MB (Small)	100
20 MB (Small)	100
100 MB (Small)	100
512 MB (Medium)	50
1 GB (Medium)	50
2 GB (Medium)	30

Table 3.3: Test 2 Cases

## 4 ANALYSIS AND RESULTS

### 4.1 ENERGY CONSUMPTION OF HDFS AND HOST FILE SYSTEM

#### 4.1.1 MEAN VALUES

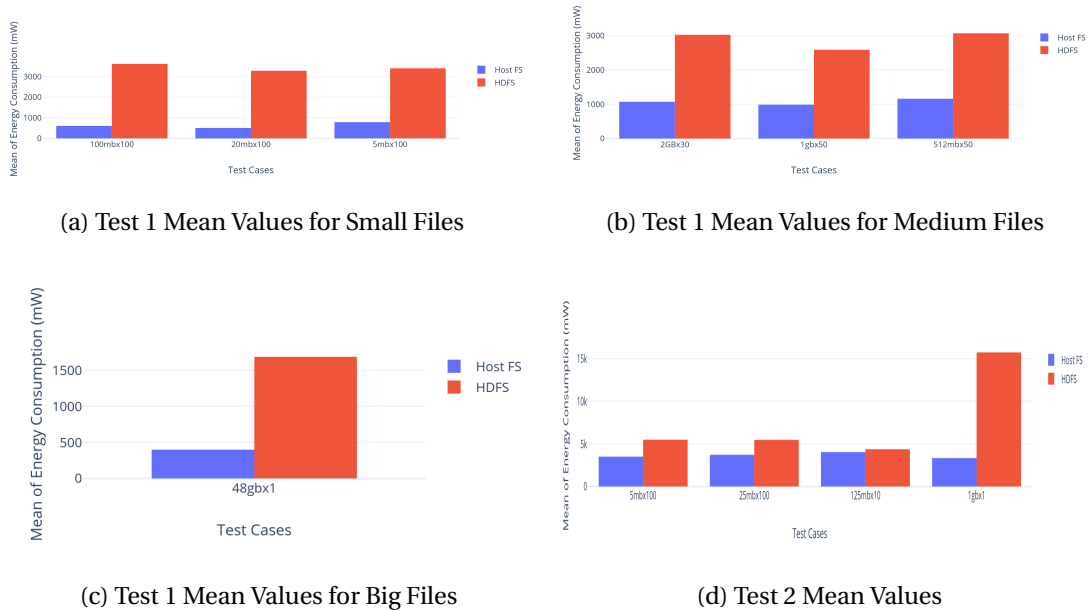


Figure 4.1: Mean Values of Energy Consumption(mW)

Figure 4.1 shows the mean values for Test 1 (4.1a, 4.1b 4.1c) and Test 2 (4.1d). It is expected to have an apparent difference as we see in the means of energy consumption since HDFS uses distributed file structure with additional services and utilities and Host FS does not. By looking at the mean values and the changes depending on file sizes, we can conclude that the difference between mean values decrease when the file size gets bigger than Hadoop's block size(128MB). Mean values of HDFS also decreases when the file size is bigger hence the difference of host file system between small and big files are smaller. When we consider the CPU load, too (4.1d), on the contrary of Test 1, we see that when the file size is bigger than Hadoop's block size, the mean of energy consumption increases. Since PowerAPI provides measurements based on formulations and some of the values are lower than exact but summed to the next value, we need to check Medians, Standard Deviations and Total Consumption values, too.

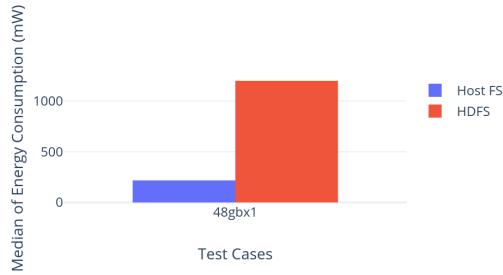
### 4.1.2 MEDIAN VALUES



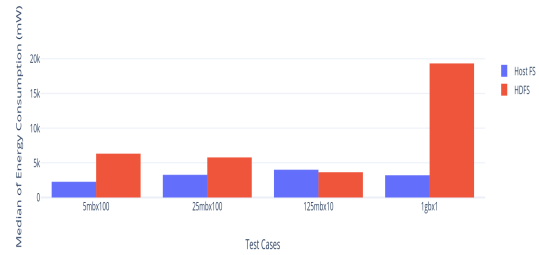
(a) Test 1 Median Values for Small Files



(b) Test 1 Median Values for Medium Files



(c) Test 1 Median Values for Big Files



(d) Test 2 Median Values

Figure 4.2: Median Values of Energy Consumption(mW)

Figure 4.2 shows the Median values of Test 1 (4.2a, 4.2b 4.2c) and Test 2 (4.2d) Here also we see similar patterns with mean values, meaning that values are regularly distributed. Since PowerAPI doesn't do a rounding but it adds the irregular measurement to the next one, this is still a good distribution of values. It should have been affected by that we turned off all of the system processes during the tests and our test structure were assigned to the tests, only. Still, by looking at the values we don't see the decrease at the power consumption of Hadoop depending on the block size and file sizes. So the difference in mean values could be caused by total execution time.

### 4.1.3 STANDARD DEVIATION OF VALUES

Test Case	St-Dev in Host FS Test 1	St-Dev in HDFS Test 1
5 MB (x100)	315.3	3166.0
20 MB (x100)	297.3	3179.4
100 MB (x100)	300.0	3792.5
512 MB (x50)	200.0	2680.7
1 GB (x50)	140.4	2322.5
2 GB (x30)	128.8	1778.6
48 GB (x1)	307.8	1162.6

Table 4.1: Standard Deviation Values of Test 1 (mW)

Test Case	St-Dev in Host FS Test 2	St-Dev in HDFS Test 2
5 MB (x100)	2543.0	3274.6
25 MB (x100)	1157.7	2994.6
125 MB (x10)	726.2	2105.2
1 GB (x1)	450.3	8124.5

Table 4.2: Standard Deviation Values of Test 2 (mW)

As we see in the standard deviations, they almost show the maximum values of each measurement, since PowerAPI doesn't provide any roundings and only uses sensor values for the measurements and formulations and a measure using PowerAPI includes very low values for utilization moments of processes and 0.0w values for pre-utilization moments of CPU, which is expected as it is and makes it harder to have conclusions using only standard deviations. Nevertheless, we can conclude that HDFS triggers new processes for each file and stops the process when the migration is done even if there are more files to migrate which also coincides with previous work about Hadoop I/O management [26], while it's apparently not the case for Host FS.

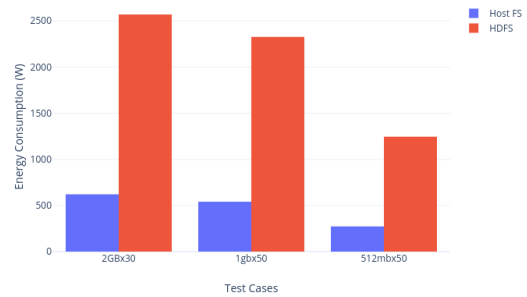
### 4.1.4 TOTAL ENERGY CONSUMPTION AND EXECUTION TIME

#### TOTAL ENERGY CONSUMPTION

Figures below (4.3) show the total energy consumption of an executed task. These values show that difference between distributed and non-distributed file systems and the potential of leakage in energy is high. To be able to make a comment about the energy consumption of each system, we also have to consider the migration of such files between distributed and non-distributed file systems and also the overall execution times for certain jobs.



(a) Test 1 :Total Energy Consumption for Small Files (W)



(b) Test 1 :Total Energy Consumption for Medium Files (W)



(c) Test 1 :Total Energy Consumption for Big Files (W)



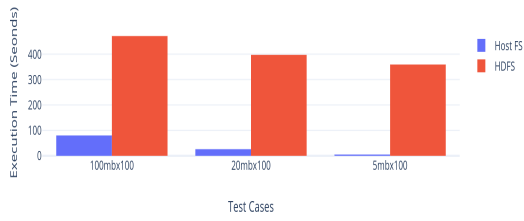
(d) Test 2 :Total Energy Consumption

Figure 4.3: Total Energy Consumption of Tests

#### EXECUTION TIME

Figures below(4.4) show the total execution time of test tasks. As it is in the energy consumption, some tasks take more time to execute on HDFS than Host FS, except the wordCount on 1 GB text file. Even if HDFS consumes more energy than Host FS, it takes less time to process the same file under the distributed structure. This shows as a nature of Hadoop Distributed File System is faster when running MapReduce jobs on large files. As we also have seen in the energy consumptions, when it gets above the block size of Hadoop, it consumes more energy but it takes relatively less time to execute.

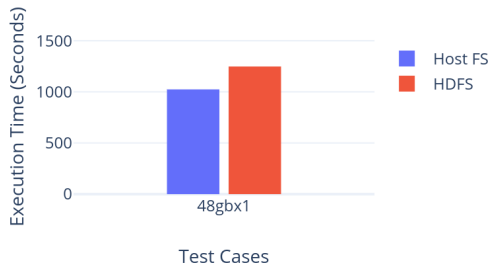




(a) Test 1 : Execution Time for Small Files (W)



(b) Test 1 : Execution Time for Medium Files (W)



(c) Test 1 : Execution Time for Big Files (W)



(d) Test 2 : Execution Time

Figure 4.4: Total Execution Time of Test Cases

## 4.2 ENERGY CONSUMPTION AND EXECUTION TIME OF MIGRATION BETWEEN FILE SYSTEMS

To be able to compare whole scenarios to each other, we measured the energy consumption of migrating files from HDFS to Host FS and from Host FS to HDFS. Table 4.3 shows the execution times for migration of file from Host FS to HDFS and from HDFS to Host FS. Using these and the total execution time and energy consumption values, we can now come up with a table of whole scenarios including overall energy consumption and execution time so we can decide which cases are more energy-efficient than its alternative.

Test Case	Host FS to HDFS	HDFS to Host FS
5 MB (x100)	663 W (317s)	758 W (325s)
20 MB (x100)	828 W (351s)	793 W (334s)
100 MB (x100)	1072 W (399s)	1046 W (386s)
512 MB (x50)	2061 W (359s)	1774 W (380s)
1 GB (x50)	3722 W (592s)	1556 W (592s)
2 GB (x30)	6339 W (730s)	4072 W (783s)
48 GB (x1)	1671 W (1081s)	992 W (1197s)

Table 4.3: Energy Consumption and Execution Time of Migration Between Filesystems

### 4.3 ENERGY CONSUMPTION AND EXECUTION TIME FOR TEST SCENARIOS

Now by adding up the values measured from tests, it is possible to form scenarios with real measurements. Those scenarios are formed to compare if using non-distributed infrastructure is an energy efficiency alternative for some tasks which don't have to be run under distributed structure and don't have a strict time to be executed, such as workflow analysis on HDFS, evaluating load balancers, estimation processes and any kind of file migration between hard drives. Tables 4.4 and 4.5 show the total execution time of a task only inside HDFS and routed via a Host FS. For copying files to another location inside the rack, we pulled the files from HDFS, relocated the files and pushed them back to HDFS. For analysing type loads, we pulled the files from HDFS, ran the jobs and pushed the files back to HDFS.

Test 1 Case	Energy Difference (=(HDFS - Routed))(W)	Time Difference (=(HDFS - Routed))(sec)
5 MB (x100)	986 W	-323 s
20 MB (x100)	863 W	-415 s
100 MB (x100)	802 W	-474 s
512 MB (x50)	-952 W	-569 s
1 GB (x50)	-1350 W	-832 s
2 GB (x30)	-1398 W	-1242 s
48 GB (x1)	-965 W	-2054 s

Table 4.4: Test 1 Scenarios' Energy and Time Differences

Test 2 Case	Energy Difference (=(HDFS - Routed))(W)	Time Difference (=(HDFS - Routed))(sec)
5 MB (x100)	7053 W	1228 s
25 MB (x100)	7545 W	1241 s
125 MB (x10)	-1435 W	-628 s
1 GB (x1)	-500 W	-845 s

Table 4.5: Test 2 Scenarios Energy Consumption and Time Differences

## 5 CONCLUSION AND FUTURE WORK

As expected, Hadoop Distributed File System's functionality, security and extra services cause consuming more energy than the non-distributed ext4 formatted file system under the same circumstances, for most of the cases. For some types of copying and analysing jobs, it was observed that routing the files to Host FS and migrating them over Host FS can be energy saving. It is also dependent on file count for some test cases, meaning that the larger number and closer to real-life scenarios can save a better proportion of energy.

Test 1 showed that when the file sizes are smaller than Hadoop's block size (Test 1 case 1,2 and 3) there is a possibility to save energy using a route via the Host FS. The most efficient scenario was when the file size is just under the Hadoop's configured block size (128mb vs 125mb) with 27% energy saving. When the file sizes get bigger than the block size, HDFS starts to operate more efficiently than Host FS. This, by the way, doesn't apply for the execution time for Test 1 cases. When the file sizes get bigger, the differences in execution times don't change behaviour, continues to accumulate depending on file size and file count.

Test 2 also showed similarly to Test 1 that when the file sizes are smaller than Hadoop's block size (Test case 1 and 2), a routing over Host FS can cause energy savings on analyse type of jobs. The difference here is when the file size gets close to Hadoop's block size, Hadoop starts to become more energy-efficient than other cases. Even if we process more data, it takes less time and less energy on Hadoop (Test 2:3 vs Test 2:4). Differences in execution times are also similar to Test 1 with almost the same behaviour in cases where file size is close to Hadoop's block size (125mb to 128mb). Hadoop performs faster when the block size meets the file size. Quite different than its behaviour in energy consumption and similar to the behaviour in Test 1, Hadoop executed the tasks much faster when the file sizes was bigger.

The experiments shows that if there is not a necessity for the distributed environment for a data migration job, for volumes under or equal to the block size and different numbers of files and process types, it is suggested to use a non-distributed file structure if the execution time is not a concern. While these are the tests inside the same rack, this can apply for mostly data analysis for dynamic scaling of hardware and software, pre-migration mirroring of files and conjugated file verification types of workloads most of which don't have a strict deadline.

The hardware we used in our tests are testing environment and even Hadoop promises to work smoothly on commodity hardware, the results may be different from the production environment. We used all the configuration as default, including block size of 128 MB.

### 5.1 ANALYSIS OF GRAPHICAL DATA

In the Analysis and Results section, we visualized a number of graphs and tables of certain measurements, such as Means (Figure : 4.1), Medians (Figure : 4.2), Standard Deviations (Table : 4.1, 4.2) and aggregated values of those certain measurements (Figure 4.3, 4.4). Firstly, from the graphical representation of mean and median values, we see that HDFS's energy consumption trend is higher than Host FS. This obviously is caused by Hadoop's additional functionalities and services since Host FS is an embedded system of an OS and it runs inside the OS kernel, so it is already there when you have an OS running. By analysing the mean

and median values, we can come up with the similar conclusion about file sizes and its effects to the energy consumption and execution time; in the mean and median values of Test 2:4, we see that Hadoop is consuming more energy than it usually does with other file sizes. Having high median values in HDFS measurements instead of moderate values shows that PowerAPI's measurements are not far from sensor values but have some low measurements of low utilization levels of CPUs and they are recovered by PowerAPI within its formulations. Standard Deviation values show less information about the test cases but more about the testbed. PowerAPI gets low values and high values in a row to correct the formulation of sensor values, it causes to have a larger interval in measurements and a larger standard deviation. Different than other measurements, the standard deviation is higher than usual inside Test 2; when the file size is small on Host FS. Every iteration in Host FS is started by a new readFile operation and continued by a wordCount analysis process. This difference in standard deviation can be caused by low consumption of readFile operation on small files and a higher consumption analysing job as it is similar to larger file sizes.

## 5.2 ANALYSIS OF TEST RUNS

We formed our test runs to cover a large diameter of common scenarios in real life. To be able to achieve this, we used different file sizes with different repetitions for each workload. This helped us to see the effects of file sizes, configurations and repetition counts and the effects on execution time and energy consumption. Working with different numbers of test runs also helped us to separate focus points from others for a better analysis, furthermore, we lowered the error margin by having a large pool of observations.

## 5.3 CONCLUSIONS AND ANALYSIS OF TESTBED

We used one of the most common and up to date CPUs for the time of the test with a integrated hub energy sensors on. We ran everything inside the same set of hard drives, motherboards, and CPUs to achieve a better measurement with less external factors. We didn't use any Virtual Machines but Physical Machines to have less leakage at the performance([26])and to have more accurate measurements ([28]). We used a relatively moderate and default values for Hadoop's configuration, 128 MB of block size as the most important factor for our tests. PowerAPI has been set to have a frequency of 1000 ms during the tests with a margin of error from 0.5% up to 3%. We didn't design our tests to go through a network protocol, except for cluster communication and it is also known that under some circumstances network peripherals are also important factors in energy consumption. We also didn't load the CPUs with other jobs, which is good for testing but not good for real-life scenarios since most of the clusters will be running with high utilization under virtualized environments. Moreover, we didn't push the workloads at once to fully utilize the CPU's and since they are known to be more efficient under full utilization, test results may differ with weaker processors or higher utilization levels.

## 5.4 FUTURE WORK

For the future work using the knowledge we achieved from this thesis, we can suggest working on hybrid file system protocols of file system allocation for cloud usage inside the rack, which takes also Host File Systems into consideration with a suggestion of a condition of turning the Distributed File System off or to a low utilization state during the phase. Hybrid file systems are already under consideration for researchers and some companies hence most of the time, energy consumption is not one of the concerns and Host FS is not considered as an alternative for such workloads. For analysis type of jobs in which chunks of small volume data are processed such as prediction algorithms, task prioritizing for scheduling and for migration of lots of types such as images with layers, snapshots of file/folder structures, data in multiple pieces, backup migration and data replication type of jobs in which big volumes of data are migrated inside the cluster, our test results can be used as a guide to decide on the route of the files.

As we see in the relationship between the file size and HDFS block size, further research can be made for investigating the effects of the configuration of a Distributed File System to the execution time and energy consumption. A hybrid configuration or hybrid Data Nodes with varying configuration settings can be experimented to have better insight for making decisions for cloud managers.

Since we didn't include a migration over a network in our experiments, a potential future work can be done by using a network and comparing DFS's behaviour to a Host FS, since DFSs are all dependent on a Host File System even if they are different DFSs. Moreover, as we mentioned earlier, P2P systems are known by their relevance to the nature of non-distributed file systems and their potential of energy efficiency in the networks. Thus a route allocation of files considering P2P networks for migration of files may have a potential of energy saving in this matter. PowerAPI has other metrics, which are used to measure and compare the effects of other factors such as parallel CPUs and some are better in results with fractions, some of them can also be used to measure behaviour under similar tests and can be compared to the results we achieved.

## REFERENCES

- [1] J. M. B, "The Evolution To Cloud Computing (How Did We Get Here?)," *The Enterprise Cloud Blog*, 2015.
- [2] M. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," *Energies*, vol. 10, p. 1470, 09 2017.
- [3] R. Danilak, "Why energy is a big and rapidly growing problem for data centers," 2017. Forbes Technology Council Community Voice.
- [4] U. United Nations Framework Convention on Climate Change, "Ict sector helping to tackle climate change," 2016.
- [5] A. B. Nassif and M. A. M. Capretz, "Moving from saas applications towards soa services," in *2010 6th World Congress on Services*, pp. 187–188, July 2010.
- [6] M. Boniface, B. Nasser, J. Papay, S. C. Phillips, A. Servin, X. Yang, Z. Zlatev, S. V. Gogouvitis, G. Katsaros, K. Konstanteli, G. Kousiouris, A. Menychtas, and D. Kyriazis, "Platform-as-a-service architecture for real-time quality of service management in clouds," in *2010 Fifth International Conference on Internet and Web Applications and Services*, pp. 155–160, May 2010.
- [7] B. Depardon, G. Le Mahec, and C. Séguin, "Analysis of Six Distributed File Systems," research report, Université de Picardie Jules Verne, Feb. 2013.
- [8] E. Levy and A. Silberschatz, "Distributed file systems: Concepts and examples," *ACM Comput. Surv.*, vol. 22, pp. 321–374, Dec. 1990.
- [9] A. S. Foundation, "Hadoop releases," 2006.
- [10] A. H. Community, "Powered by apache hadoop," 2007.
- [11] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10, May 2010.
- [12] A. Ben Ayed, M. Ben Halima, and A. Alimi, "Mapreduce based text detection in big data natural scene videos," vol. 53, 08 2015.
- [13] Apache Software Foundation, "Apache Hadoop - Default Configuration File," 2019. Version 3.0, [Online; accessed on 04.05.2019 via <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml>].
- [14] Wikipedia, "Apache hadoop — Wikipedia, the free encyclopedia," 2008-2019. [Online; accessed 03.05.2019].
- [15] David Both, "An introduction to Linux's EXT4 filesystem," 2017. [Online, accessed on 02.04.2019] via <https://opensource.com/article/17/5/introduction-ext4-filesystem>.
- [16] David Both, "An introduction to Linux filesystems," 2016. <https://opensource.com/life/16/10/introduction-linux-filesystems>, [Online accessed on 02.04.2019].
- [17] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. T. Pedretti, and J. H. Laros III,

- “Standardizing power monitoring and control at exascale,” *Computer*, vol. 49, pp. 38–46, Oct 2016.
- [18] A. Nouredine, R. Rouvoy, and L. Seinturier, “Unit Testing of Energy Consumption of Software Libraries,” in *Symposium On Applied Computing*, (Gyeongju, South Korea), pp. 1200–1205, Mar. 2014.
- [19] A. Nouredine, R. Rouvoy, and L. Seinturier, “Monitoring Energy Hotspots in Software,” *Journal of Automated Software Engineering*, vol. 22, pp. 291–332, Sept. 2015.
- [20] M. Colmant, P. Felber, R. Rouvoy, and L. Seinturier, “WattsKit: Software-Defined Power Monitoring of Distributed Systems,” in *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (F. Capello, G. Fox, and J. Garcia-Blas, eds.), Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), (Madrid, Spain), p. 10, IEEE, May 2017.
- [21] A. Nouredine, A. Bourdon, R. Rouvoy, and L. Seinturier, “Runtime Monitoring of Software Energy Hotspots,” in *ASE - The 27th IEEE/ACM International Conference on Automated Software Engineering - 2012*, (Essen, Germany), pp. 160–169, Sept. 2012.
- [22] U. o. L. Institut national de recherche en informatique et en automatique (INRIA), “Powerapi,” 2018. [Online; accessed on 02.05.2019 via <http://powerapi.org>].
- [23] CISCO, “Cisco visual networking index: Forecast and trends, 2017 - 2022 white paper,” tech. rep., 2018(updated February 27, 2019).
- [24] R. Abu Khurma, H. Harahsheh, and A. Sharieh, “Task scheduling algorithm in cloud computing based on modified round robin algorithm,” *Journal of Theoretical and Applied Information Technology*, vol. 96, pp. 5869–5888, 09 2018.
- [25] C. Kaewkasi and W. Srisuruk, “A study of big data processing constraints on a low-power hadoop cluster,” in *2014 International Computer Science and Engineering Conference (ICSEC)*, pp. 267–272, July 2014.
- [26] E. Feller, L. Ramakrishnan, and C. Morin, “On the performance and energy efficiency of hadoop deployment models,” in *2013 IEEE International Conference on Big Data*, pp. 131–136, Oct 2013.
- [27] E. Feller, L. Ramakrishnan, and C. Morin, “Performance and energy efficiency of big data applications in cloud environments: A hadoop case study,” *Journal of Parallel and Distributed Computing*, vol. 79-80, pp. 80 – 89, 2015. Special Issue on Scalable Systems for Big Data Management and Analytics.
- [28] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, “Virtual machine power metering and provisioning,” in *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC ’10, (New York, NY, USA), pp. 39–50, ACM, 2010.
- [29] S. Bai and H. Wu, “The performance study on several distributed file systems,” in *2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 226–229, Oct 2011.
- [30] S. Shirinbab, L. Lundberg, and D. Erman, “Performance evaluation of distributed storage systems for cloud computing,” *I. J. Comput. Appl.*, vol. 20, pp. 195–207, 2013.
- [31] X. Zhou and L. He, “A virtualized hybrid distributed file system,” in *2013 International*

- Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 202–205, Oct 2013.
- [32] S. A. Yazd, S. Venkatesan, and N. Mittal, “Energy efficient hadoop using mirrored data block replication policy,” in *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pp. 457–462, Oct 2012.
- [33] T. Wirtz and R. Ge, “Improving mapreduce energy efficiency for computation intensive workloads,” in *2011 International Green Computing Conference and Workshops*, pp. 1–8, July 2011.
- [34] L. Duan, D. Zhan, and J. Hohnerlein, “Optimizing cloud data center energy efficiency via dynamic prediction of cpu idle intervals,” in *2015 IEEE 8th International Conference on Cloud Computing*, pp. 985–988, June 2015.
- [35] L. A. Barroso and U. Holzle, “The case for energy-proportional computing,” *Computer*, vol. 40, pp. 33–37, Dec 2007.
- [36] L. Mashayekhy, M. M. Nejad, D. Grosu, D. Lu, and W. Shi, “Energy-aware scheduling of mapreduce jobs,” in *2014 IEEE International Congress on Big Data*, pp. 32–39, June 2014.
- [37] I. n. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, “Greenhadoop: Leveraging green energy in data-processing frameworks,” in *Proceedings of the 7th ACM European Conference on Computer Systems, EuroSys ’12*, (New York, NY, USA), pp. 57–70, ACM, 2012.
- [38] R. Fischer e Silva and P. Carpenter, *E-EON: Energy-Efficient and Optimized Networks for Hadoop*. PhD thesis, 05 2018.
- [39] M. Malik, K. Neshatpour, S. Rafatirad, and H. Homayoun, “Hadoop workloads characterization for performance and energy efficiency optimizations on microservers,” *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, pp. 355–368, July 2018.
- [40] A. Nouredine, A. Bourdon, R. Rouvoy, and L. Seinturier, “A preliminary study of the impact of software engineering on greenit,” in *2012 First International Workshop on Green and Sustainable Software (GREENS)*, pp. 21–27, June 2012.
- [41] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, “Fog computing may help to save energy in cloud computing,” *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 1728–1739, May 2016.
- [42] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe, “Process-level power estimation in vm-based systems,” in *EuroSys*, 2015.
- [43] H. Zhu, X. Wang, and H. Wang, “A new model for energy consumption optimization under cloud computing and its genetic algorithm,” in *2014 Tenth International Conference on Computational Intelligence and Security*, pp. 7–11, Nov 2014.
- [44] Apache Software Foundation, “Hadoop: Fair scheduler (v2.7.4),” 2017. [Online; accessed on 05.05.2019] via <https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/FairScheduler.html>.
- [45] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, “Tarazu: Optimizing mapreduce on heterogeneous clusters,” *SIGARCH Comput. Archit. News*, vol. 40, pp. 61–74, Mar. 2012.



- [46] R. Ahmad, A. Gani, S. h. Ab hamid, M. Shiraz, F. Xia, and S. Madani, "Virtual machine migration in cloud data centers: A review, taxonomy, and open research issues," *The Journal of Supercomputing*, vol. 71, 07 2015.
- [47] P. Kaur and A. Rani, "Virtual machine migration in cloud computing," *International Journal of Grid and Distributed Computing*, vol. 8, pp. 337–342, 10 2015.
- [48] L. Zhu, J. Chen, Q. He, D. Huang, and S. Wu, "Itc-lm: A smart iteration-termination criterion based live virtual machine migration," in *Network and Parallel Computing* (C.-H. Hsu, X. Li, X. Shi, and R. Zheng, eds.), (Berlin, Heidelberg), pp. 118–129, Springer Berlin Heidelberg, 2013.
- [49] Apache Commons IO, "Apache Software Foundation," 2012-2019. [Online; accessed on 15.05.2019] via <https://commons.apache.org/proper/commons-io/index.html>.
- [50] Tintri by DataDirect Networks, "Data Dive: VM Sizes in the Real World," 2016. [Online; accessed on 08.05.2019] via <https://www.tintri.com/blog/2016/05/data-dive-vm-sizes-real-world>.

## 6 APPENDIX & NOTES

We used R Studio(V1.1.453) for statistical jobs, Plotly Python Library for graphical presentation, Microsoft Excel (V16.28 (19081202)) and Microsoft Power Query (V2.59.5135.201) for clustering and structuring raw outputs, Draw.io for UMLs and other figures, Gitlab for Git service, Google Drive for cloud service and Overleaf for creating the report with Latex.

Since we used a simplified type (IEEE Tran) of references for a clearer visual representation, printed format may not be as satisfying as raw format for researchers. A raw format of bibliography file and full version of code used in this thesis with working example can be found at:

GitLab Link 

```
1 https://gitlab.com/edzyldrm/ms-thesis/
```

Here are some snippets from the code and the procedure :

```
1 ### Definitions
2
3 def __init__(self, repeat_count):
4     self.hdfs = None
5     self.host_fs = None
6     self.repeat_count = repeat_count
7     self.hdfs_source_base = hdfs_source_dir
8     self.hdfs_dest_base = hdfs_dest_dir
9     self.fs_source_base = fs_source_dir
10    self.fs_dest_base = fs_dest_dir
11    self.output_dir = '/output_' + random_string(8)
12    self.tasks = []

1 ### Test Case 1 Function
2
3 def copy_command(self):
4     if self.source_fs == 'hdfs' or self.destination_fs == 'hdfs':
5         cmd_list = ['hdfs', 'dfs']
6         if self.source_fs == 'hdfs' and self.destination_fs == 'hdfs'
7         :
8             cmd_list.extend(['-cp', '-f'])
9             cmd_list.append(self.hdfs_source_name)
10            elif self.source_fs == 'host' and self.destination_fs == '
11            hdfs':
12                cmd_list.extend(['-put', '-f'])
13                cmd_list.append(self.fs_source_name)
14                cmd_list.append(self.hdfs_destination_name)
15            else:
16                cmd_list = ['cp', '-r', self.fs_source_name, self.
17                fs_destination_name]
18
19            return [], cmd_list
```

```
1 ## Test Case 2 Function
2
3 def wordcount_command(self):
4     cmd_list = ['hadoop', 'jar', 'libs/wc.jar', 'WordCount',
5                 self.hdfs_source_name, self.hdfs_destination_name]
6     return [], cmd_list
7
8 def wordcount_file_command(self):
9     cmd_list = ['java', '-jar', 'libs/wordcountfile.jar',
10                self.fs_source_name]
11    return [], cmd_list
```

```

1  ### Run Tasks
2  for i in range(self.repeat_count):
3      print("*****")
4      print("## Round({}/{})".format(i + 1, self.repeat_count))
5      print("Preparing for tasks")
6      init_commands = [['hdfs', 'dfs', '-mkdir', self.hdfs_dest_base +
7          self.output_dir],
8          ['mkdir', self.fs_dest_base + self.output_dir]]
9      for cmd in init_commands:
10         (ret, out, err) = self.run_cmd('init', cmd)
11         if out:
12             print(out.decode('utf-8'))
13         if err:
14             print(err.decode('utf-8'))
15     print("*****")
16     for task in self.tasks:
17         fs_task = FsTask(task, self.hdfs_source_base, self.
18             hdfs_dest_base, self.fs_source_base,
19             self.fs_dest_base, self.output_dir)
20         print("\tTask: {}".format(fs_task.name))
21         init_commands, task_commands = fs_task.commands()
22         if init_commands:
23             (ret, out, err) = self.run_cmd('init', init_commands)
24             if out:
25                 print(out.decode('utf-8'))
26             if err:
27                 print(err.decode('utf-8'))
28         if task_commands:
29             try:
30                 (ret, out, err) = self.run_cmd('task', task_commands)
31                 if out:
32                     print(out.decode('utf-8'))
33                 if err:
34                     print(err.decode('utf-8'))
35             except FileNotFoundError as e:
36                 print(e)
37         print("*****")
38         print("### Finishing for tasks")
39         init_commands = [['hdfs', 'dfs', '-rm', '-R', self.hdfs_dest_base
40             + self.output_dir],
41             ['rm', '-rf', self.fs_dest_base + self.
42             output_dir]]
43         for cmd in init_commands:
44             (ret, out, err) = self.run_cmd('clean', cmd)
45             if out:
46                 print(out.decode('utf-8'))
47             if err:
48                 print(err.decode('utf-8'))
49         print("*****")

```

```
1 ## example json file
2
3 [
4   {
5     "name": "Do Map/Reduce job on the selected text file on hdfs .",
6     "command": "mapReduce",
7     "source": "hdfs:filea.txt",
8     "destination": "hdfs:output",
9     "regex": "'property'"
10  }
11 ]
```