Energy savings and performance improvements with SSDs in the Hadoop Distributed File System

Ivanilton Polato

Text Submitted
to the
Institute of Mathematics and Statistics
of the
University of São Paulo
for the
Doctoral Degree in Science

Program: Computer Science
Advisor: Professor Fabio Kon

This research was supported by
Fundação Araucária (Projeto DINTER UTFPR/IME–USP),
CAPES, and CNPq

São Paulo, October, 2016

Energy savings and performance improvements with SSDs in the Hadoop Distributed File System

This version of the thesis contains the changes suggested by the Committee Members during the public defense of the original version of this work, which occurred in Aug 29th, 2016. A copy of the original version is available at the Institute of Mathematics and Statistics (IME) of the University of São Paulo (USP).

Committee Members:

- Prof. Dr. Fabio Kon (Advisor) – IME-USP
- Prof. Dr. Daniel Batista – IME-USP
- Prof. Dr. Denilson Barbosa – University of Alberta, Canada
- Prof. Dr. Fabio Costa – UFG
- Prof. Dr. Raphael Yokoingawa de Camargo – UFABC

Acknowledgements

First, I would like to thank my advisor Fabio Kon for his guidance and support to my research; when I needed advice, Fabio kept my focus on the objectives. Fabio, I know we can collaborate for a long time in future research.

I would like to thank Denilson Barbosa, which I consider my co-advisor, who welcomed me at University of Alberta. I had such a good time in Edmonton, and could learn from his research and suggestions. Denilson's support to provide the infrastructure to the development of this work was an enormous gesture of confidence. I also would like to thank Abram Hindle, who joined us and introduced me to the energy-aware computation world. Abram's analysis and insights were really helpful.

Thank you to all my colleagues during this journey, from our IME research groups, our friends from the Butantã's apartment (folks from the last subway train of the night), my work colleagues at UTFPR who always stood up for our CS department – DACOM.

This research received financial support from different foundations. I am thankful to Fundação Araucária (DINTER UTFPR/IME-USP), to the Emerging Leaders in the Americas Program (ELAP) from the Canadian Government, which granted me with a Sandwich scholarship, CAPES, and CNPq.

Finally, I really would like to thank my family. You are the main reason I am here today. Love you all! My wife and my kids, which understood the lonely road that I had to go and provided all the support I needed, always with a smile: thank you so much! Mom, Dad, and my sisters: love you!

Abstract

Energy issues gathered strong attention over the past decade, reaching IT data processing infrastructures. Now, they need to cope with such responsibility, adjusting existing platforms to reach acceptable performance while promoting energy consumption reduction. As the *de facto* platform for Big Data, Apache Hadoop has evolved significantly over the last years, with more than 60 releases bringing new features. By implementing the MapReduce programming paradigm and leveraging HDFS, its distributed file system, Hadoop has become a reliable and fault tolerant middleware for parallel and distributed computing over large datasets. Nevertheless, Hadoop may struggle under certain workloads, resulting in poor performance and high energy consumption. Users increasingly demand that high performance computing solutions address sustainability and limit energy consumption. In this thesis, we introduce $HDFS_H$, a hybrid storage mechanism for HDFS, which uses a combination of Hard Disks and Solid-State Disks to achieve higher performance while saving power in Hadoop computations. $HDFS_H$ brings, to the middleware, the best from HDs (affordable cost per GB and high storage capacity) and SSDs (high throughput and low energy consumption) in a configurable fashion, using dedicated storage zones for each storage device type. We implemented our mechanism as a block placement policy for HDFS, and assessed it over six recent releases of Hadoop with different architectural properties. Results indicate that our approach increases overall job performance while decreasing the energy consumption under most hybrid configurations evaluated. Our results also showed that, in many cases, storing only part of the data in SSDs results in significant energy savings and execution speedups.

Keywords: Hadoop, HDFS, Hybrid Storage, Energy Efficiency, Solid-State Disk, SSDs, Distributed File Systems, Parallel File Systems, Green Computing

Resumo

POLATO, I.. **Economia de energia e aumento de desempenho usando SSDs no Hadoop Distributed File System**. 2016. 87 f. Thesis (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

Ao longo da última década, questões energéticas atraíram forte atenção da sociedade, chegando às infraestruturas de TI para processamento de dados. Agora, essas infraestruturas devem se ajustar a essa responsabilidade, adequando plataformas existentes para alcançar desempenho aceitável enquanto promovem a redução no consumo de energia. Considerado um padrão para o processamento de Big Data, o Apache Hadoop tem evoluído significativamente ao longo dos últimos anos, com mais de 60 versões lançadas. Implementando o paradigma de programação MapReduce juntamente com o HDFS, seu sistema de arquivos distribuídos, o Hadoop tornou-se um middleware tolerante a falhas e confiável para a computação paralela e distribuída para grandes conjuntos de dados. No entanto, o Hadoop pode perder desempenho com determinadas cargas de trabalho, resultando em elevado consumo de energia. Cada vez mais, usuários exigem que a sustentabilidade e o consumo de energia controlado sejam parte intrínseca de soluções de computação de alto desempenho. Nesta tese, apresentamos o $HDFS_H$, um sistema de armazenamento híbrido para o HDFS, que usa uma combinação de discos rígidos e discos de estado sólido para alcançar maior desempenho, promovendo economia de energia em aplicações usando Hadoop. O $HDFS_H$ traz ao middleware o melhor dos HDs (custo acessível por GB e grande capacidade de armazenamento) e SSDs (alto desempenho e baixo consumo de energia) de forma configurável, usando zonas de armazenamento dedicadas para cada dispositivo de armazenamento. Implementamos nosso mecanismo como uma política de alocação de blocos para o HDFS e o avaliamos em seis versões recentes do Hadoop com diferentes arquiteturas de software. Os resultados indicam que nossa abordagem aumenta o desempenho geral das aplicações, enquanto diminui o consumo de energia na maioria das configurações híbridas avaliadas. Os resultados também mostram que, em muitos casos, armazenar apenas uma parte dos dados em SSDs resulta em economia significativa de energia e aumento na velocidade de execução.

Keywords: Hadoop, HDFS, Armazenamento Híbrido, Eficiência Energética, Discos de Estado Sólido, SSDs, Sistema de Arquivos Distribuído, Sistemas de Arquivos Paralelo, Computação Verde

Contents

Abbreviations vi

List of Figures vii

List of Tables ix

1 Introduction 1
- 1.1 Objectives . . . 3
- 1.2 Original Contributions . . . 3
- 1.3 Funding . . . 4
- 1.4 Publications . . . 4

2 Background 6
- 2.1 The Apache Hadoop Framework . . . 6
- 2.2 MapReduce . . . 8
- 2.3 HDFS . . . 9
- 2.4 YARN . . . 10
- 2.5 Storage Devices . . . 10
- 2.6 Green Computing . . . 13

3 Related Work 14
- 3.1 Hadoop Research . . . 14
- 3.2 Solid-State Drives on HDFS . . . 18
- 3.3 Energy and Green Computing Research . . . 19

4 Motivating and Grounding Experiments 21
- 4.1 Versions and Releases . . . 21
- 4.2 Benchmarks and Datasets Used . . . 23
- 4.3 Cluster Infrastructure . . . 24
- 4.4 Experiment Design and Methodology . . . 24
- 4.5 Results and Analysis . . . 26
  - 4.5.1 Job Makespan . . . 26
  - 4.5.2 Evidence of Changes in Hadoop's Performance . . . 29
  - 4.5.3 Energy Performance . . . 30
  - 4.5.4 Hadoop Source Code Analysis . . . 32
- 4.6 Final Considerations . . . 37

5 $HDFS_H$: a Hybrid File System **39**
5.1 $HDFS_H$ Storage Model . . . 39
5.1.1 Block Placement Policy . . . 41
5.1.2 Storage Cost Model . . . 42
5.2 Experimental Methodology and Datasets . . . 43
5.3 Energy Consumption and Performance Analysis . . . 44
5.3.1 I/O-Bound Benchmark Results . . . 45
5.3.2 Results for the CPU-Bound Benchmarks . . . 48
5.3.3 Using SSDs as Temporary Storage Space . . . 49
5.3.4 Performance and Speedup . . . 50
5.3.5 Cost Model Analysis . . . 52
5.4 Final Considerations . . . 55

6 Discussion **57**
6.1 Findings . . . 57
6.2 Threats to Validity . . . 60

7 Conclusions **62**
7.1 Original contributions . . . 62
7.2 Future Work . . . 64

Bibliography **67**

Acronyms

AFR	Annualized Failure Rate.
BSP	Bulk Synchronous Parallel.
FOB	Fresh Out of Box.
HDDs	Hard Disk Drives.
HDs	Hard Drives.
IOPS	Input/Output Operations Per Second.
MPI	Message Passing Interface.
MTBF	Mean Time Between Failures.
SNIA	Storage Networking Industry Association.
SSDs	Solid-State Drives.
TCO	Total Cost of Ownership.
YARN	Yet Another Resource Negotiator.

List of Figures

2.1 Hadoop Releases History . . . 7
2.2 Hadoop Architectural Modifications . . . 8
2.3 Distributed MapReduce Paradigm (DG08) . . . 9
2.4 Hadoop Daemons . . . 10
2.5 Storage Devices Performance Studies (Kim13) . . . 12
2.6 Storage Devices Performance Studies (Kim15) . . . 12

3.1 Taxonomy Hierarchical Organization. . . . 15

4.1 Hadoop Versions Genealogy Tree . . . 22
4.2 Cluster Infrastructure . . . 24
4.3 Hadoop Sort 10GB Job Makespan . . . 26
4.4 Hadoop Sort 48GB Job Makespan . . . 27
4.5 Hadoop Sort 256GB Job Makespan . . . 28
4.6 Hadoop WordCount Experiments Job Makespan . . . 28
4.7 Timeseries of Hadoop Mentions on StackOverflow . . . 29
4.8 Hadoop Sort Experiments Energy Consumption . . . 30
4.9 Hadoop Sort 256GB Energy Consumption . . . 31
4.10 Hadoop WordCount Experiments Energy Consumption . . . 31
4.11 Sort Benchmark Energy and Job Makespan Split by Phase . . . 35
4.12 WordCount Benchmark Energy and Job Makespan Split by Phase . . . 35

5.1 $HDFS_H$ 10GB Hadoop Sort Results . . . 45
5.2 $HDFS_H$ 48GB Hadoop Sort Results . . . 46
5.3 $HDFS_H$ 256GB Hadoop Sort Results . . . 46
5.4 $HDFS_H$ Normalized Comparison of Sort Benchmarks . . . 47
5.5 Energy Influence on Triple Block Replica (HD Configuration) . . . 48
5.6 Energy Consumption on Join Benchmark . . . 48
5.7 Energy Consumption on K-Means Benchmark . . . 49
5.8 Energy Consumption Rates Using *SSDzone* as Temporary Space . . . 50
5.9 Hadoop Performance: Multiple Releases over Configurations . . . 51
5.10 Hadoop Performance: Multiple Releases over Configurations . . . 51
5.11 $HDFS_H$ Cost Model Analysis: 256GB Sort Across 5 Configurations . . . 53
5.12 $HDFS_H$ Cost Model Analysis: 256GB Sort over Time . . . 53
5.13 $HDFS_H$ Cost Model Analysis: 48GB Sort over Time . . . 54

5.14 $HDFS_H$ Cost Model Analysis of $tmpSSD$: 256GB Sort 54

6.1 Hadoop Sort 256GB Job Makespan 57
6.2 Energy Consumption Results: Sort 48GB and 256GB 59

7.1 Energy Prediction Results on Multiple $HDFS_H$ Configurations 65

List of Tables

4.1 Hadoop releases used on the performance and energy consumption studies 23
4.2 Benchmarks and Dataset Sizes 23
4.3 Default Values for Configuration Files 25
4.4 Average Job Makespan Speedup Comparison (Times in s) 27
4.5 Average Energy Comparison (Energy values in kJ) 32
4.6 Pearson Correlation: Hadoop Size with Sort benchmarks 33
4.7 Correlation Summary: Size, Time, and Versions versus Energy 33
4.8 Number of Map and Reduce Tasks per Benchmark 34
4.9 Metrics in the top 60 models 36
4.10 CKJM-extended raw values 37

5.1 Definitions Used on the HDFS Hybrid Storage Model 40
5.2 Releases Used in the Experiments 43
5.3 Benchmarks and Dataset Sizes Used in the Experiments 43
5.4 Configurations Used in the Experiments 44
5.5 Sort Benchmarks: Average Energy Consumed (kJ) 47
5.6 Average Energy Consumed for the $tmpSSD$ Configuration (kJ) 50
5.7 Sort Benchmarks: Average Job Makespan (s) 52
5.8 Average Energy Reduction Percentage 55

7.1 Key Findings 62
7.2 Definitions Used on the Prediction Model 64

Chapter 1

Introduction

Two perspectives are relevant for big data analysis at the present: first, the 3 "Vs", Volume, Variety, and Velocity (Lan01; ZE11); second, hardware and software infrastructures capable of storing and processing all the collected data. Over the last years, the volume and speed of data creation consistently increased, consolidating "Big Data" (Whi12) as the reference to huge collections of datasets that can not be processed using traditional tools and individual computers. A recent study estimates that 90% of all data in the world was generated over the last two years (Bra13). The International Data Corporation (IDC) predicted that from 2005 to 2020, the digital universe will grow by a factor of 300, from 130 exabytes to 40,000 exabytes (VNOS12). The same study also predicted that the digital universe will roughly double every two years and the storage market will grow 55%. As a consequence, they expect that the discovery and analytics software market will grow 33% in the next years, which represents an 8 billion-dollar business.

Yet concerning the "Vs", data is being collected faster than ever. Take for instance the largest social network today, Facebook: in the first quarter of 2016, almost 1.1 billion users were online everyday, mostly on mobile platforms, with a growth of about 17% if compared to the last year (Fac16); Twitter has today more than 300 million active users every month (Twi16). Daily, Facebook handles more than 600TB of data while Twitter gathers around 12TB of data. At these rates, including the whole internet, users spontaneously generate petabytes of data daily.

As data generation continues to grow in a fast scale, the solution to store such volume was to increase the number of storage server facilities. This approach took advantage of hardware cost reduction, especially the reduction in the manufacturing cost of magnetic disks, also know as Hard Drives (HDs), a short form of Hard Disk Drives (HDDs). Companies are capable of storing data at a relatively low-cost/GB by creating new data centers. Roughly speaking, today the cost per GB of HDs is less than $0.05 (Smi12; Kom10), and this cost may be even lower when bundled with services from specialized providers. Alongside the price reduction, the number of data centers has grown consistently, increasing the availability of storage space. According to a recent report (Sta13), "*...70% of data center operators built a new site (data center) or renovated (expanded) a site in the past five years.*".

While it is easy and relatively cheap to store data, the bottleneck for IT companies is the information extraction process. The speed of data processing is currently much slower than the actual data collecting process. Thus, one of the largest technological challenges in software systems research is to provide mechanisms for manipulation and processing of large amounts of data in acceptable time. Web services and social media can produce together an impressive amount of data nowadays. These datasets may contain valuable information, and sometimes are not properly explored by existing systems, since they are stored in a non-structured manner, using different languages and formats, which, in many cases, are incompatible (Bak12; SAD+10). Yet, besides the proper exploration of datasets, the speed within which companies can obtain information is essential. IDC also states that, today, 23% of the digital universe is considered useful if tagged and analyzed, and from this total, only 1% has already been analyzed.

To reduce the gap between data collection and analysis, research in the last decade set the

focus on the data processing and information extraction side. Concerning the hardware, the use of clusters and grids was already a common solution to achieve higher computing power. In addition, clusters and grids have greatly benefitted from the price reduction and popularization of the x86 platform. As a consequence, x86 is the prevalent architecture on commercialized servers and personal computers, and the use of commodity hardware became a relatively low-cost way to obtain an infrastructure capable of performing large tasks that cannot be carried out by individual machines. On the software side, the main concepts of parallel and distributed computing, such as concurrency, synchronization, and fault tolerance, along with complexity abstraction, were properly implemented in several paradigms, frameworks and platforms, popularizing the use of clusters in such computing tasks by end users over the last years.

Nevertheless, data dependency and integrity, cluster load balancing, and task scheduling are still considered major concerns of parallel and distributed approaches. Adding the possibility of an almost certain machine failure, the use of these concepts becomes non-trivial to inexperienced programmers. Several frameworks were released to abstract these characteristics and provide high level solutions to end users (DPR+08; BEH+10; MAB+10; IBY+07). Some of them were built over well-known programming models, such as Message Passing Interface (MPI), Bulk Synchronous Parallel (BSP) and MapReduce. The latter is popular by its open source implementation Apache Hadoop, now considered the "De Facto" platform for parallel and distributed big data processing, used by major IT companies such as Amazon, Cloudera, Ebay, EMC[2], Facebook, Twitter, Yahoo!, and several others[1].

In this context of big data analysis supported by the expansion in the data center market, data storage and processing infrastructures encountered new challenges: energy consumption, power usage, and environmental impact. Focused on the information value, companies are concerned with the extraction process costs. It is necessary to increase the cost/benefit ratio: enhance the performance of the existing processes and infrastructures, reduce the carbon footprint of data centers by reducing the energy consumption rates, and design energy profiles that comply with governmental regulations. Our main motivation relies on an inevitable side-effect of the data analysis field expansion: the data centers increase in energy consumption. The number of data centers has consistently grown, increasing the availability of computing nodes and storage space, and demanding more power. Data centers maintenance costs and environmental impacts have consistently increased with the demand for more energy to power and cool them. In fact, energy accounts for 30% of the Total Cost of Ownership (TCO), a major and continuous cost for data centers (Ham10).

Recently, Green Computing (Mur08) and research on energy performance have focused on these issues, developing new technologies and solutions. On top of hardware approaches, Solid-State Drives (SSDs) are an interesting solution: they are fast with high performance rates and use far less power than HDs. SSDs are already present in storage service offers, even though not common yet. But there is a trend for the large-scale adoption of SSDs, supported by the constant price reduction per GB seen in the last five years. The cost per GB, which used to be around 20 times that of HDs, decreased, as for 2016, to around 5 times (Mea15; Mea16; All16). Merging these 3 factors – performance increase, energy consumption reduction and the recent cost drop – SSDs can provide unique enhancements to data analysis platforms. However, these characteristics must be properly incorporated into the existing software solutions, since most of the existing frameworks do not sufficiently support the coexistence of HDs and SSDs in the same environment, and therefore are not tailored to benefit from this possibility.

We propose in this thesis a hybrid storage approach for the Hadoop Distributed File System (HDFS), called $HDFS_H$, which seamlessly integrates both storage technologies – HDs and SSDs – to create a highly-efficient hybrid storage system. $HDFS_H$ addresses the coexistence of HDs and SSDs on the same storage space with the possibility of users setting the proportion of HD or SSD space needed considering performance, cost and energy consumption. Our hybrid storage model splits the file system into storage zones, wherein a block placement strategy directs file blocks to zones according to these predefined configurations. This enables the use of different storage configurations

[1] http://wiki.apache.org/hadoop/PoweredBy (Visited on 15/10/2016)

for different workloads, thereby achieving the desired tradeoff between performance and energy consumption. Our goal is to allow the user to determine the best configuration for the available infrastructure, by setting how much of each storage device should be used during MapReduce computations. Hadoop is one of the biggest projects from the Apache Foundation and rapidly evolved since the first releases with more than 50 releases over the last 5 years. The project evolution led to three development branches: Hadoop 1.x, which retained the original MapReduce characteristics: resource management and data processing in the same layer; Hadoop 0.23.x and 2.x, which introduced the Yet Another Resource Negotiator (YARN) resource manager, a new architectural layer, separating the MapReduce programming paradigm from the resource management.

1.1 Objectives

The main goal of this research is to develop a hybrid storage model for HDFS that can properly perceive the existing differences from storage devices – HDs and SSDs – on a Hadoop cluster, reducing the energy consumption while increasing the performance on MapReduce workloads. This objective is guided by the proposed research questions addressed by this thesis:

RQ1 : ***Are there benefits in adopting a hybrid storage approach using HDs and SSDs for Hadoop?***

RQ2 : ***Are there performance and energy consumption differences among the 3 Hadoop development branches?***

To answer these questions, we pursued the following objectives during our research:

- **Study and understand the history of Apache Hadoop through its record of versions and releases:** Hadoop is a large Apache project, which released more than 60 releases over the last decade. It is necessary to understand the evolution and the architectural and design choices made during the framework evolution to evaluate the impact on performance and energy consumption;
- **Study the performance of different versions and releases from different development branches:** the large number of releases on multiple branches justifies an investigation to discover potential performance differences between releases and specially for a comparison among different development branches;
- **Analyze the effects of hybrid storage on energy consumption:** the effect of SSDs on energy consumption reduction on Hadoop workloads is yet unknown although the use of such devices is already taking place on Hadoop clusters;
- **Analyze the framework performance under different storage configurations and workloads, mixing HDs and SSDs on a hybrid environment:** our approach must be properly assessed over different hybrid storage configurations to elucidate the differences in the use of HDs and SSDs on the platform;

1.2 Original Contributions

The key original contributions we achieved in our research are the following.

1. **A novel hybrid storage model for HDFS that takes into account the performance profiles of HDs and SSDs.** We developed, implemented, and tested a hybrid storage approach that seamlessly integrates HDs and SSDs into HDFS, creating separate storage zones for each type of device. $HDFS_H$ allows the setting of the amount of each storage zone used during MapReduce computations on Hadoop.

2. **An energy consumption profile for Hadoop under different workloads**. We assessed our approach over multiple Hadoop releases and discovered the energy consumption rates under I/O- and CPU-bound workloads for different datasets. Additionally, we detailed the energy consumption rates on the $HDFS_H$ when using different HD and SSD storage proportion in the tested workloads.

3. **A in-depth analysis of the Apache Hadoop framework**. We discovered that the three Hadoop branches and its versions perform differently under the same configurations and workloads. Some of the releases experienced performance problems, causing significant energy consumption increases during the experiments. We tracked these differences, classifying the releases according to their energetic performance.

4. **Time, cost, and energetic models for our approach**. We developed a cost model, allowing users to calculate the storage space cost. These models will allow users to have a total cost estimation of their computations on Hadoop clusters.

1.3 Funding

Our project was selected as one of the award recipients of the Emerging Leaders in the Americas Program (ELAP) 2013-2014 Award, granted by the Canadian Bureau for International Education (CBIE) on behalf of the Department of Foreign Affairs Trade and Development, Canada (DFATD), in the form of a scholarship for graduate students. This research received a valuable support from Professor Denilson Barbosa from the Computer Science Department at the University of Alberta, Canada. Professor Denilson co-supervised this research during five months at the University of Alberta (November/2013 — March/2014) providing the infrastructure for the experiments performed in this thesis. The infrastructure was used until the conclusion of our experiments, in the beginning of 2016, and the collaboration with the University of Alberta still continues.

1.4 Publications

This research produced relevant scientific publications on the topics investigated and the results achieved:

Paper 1: POLATO, I.; RÉ, R.; GOLDMAN, A.; KON, F. . "*A comprehensive view of Hadoop research – A systematic literature review*". Journal of Network and Computer Applications, v. 46, pp. 1-25, 2014.

We conducted a systematic literature review, following a previously designed protocol, to analyze Hadoop research within a timeframe. Starting with more than 1,500 papers including journals and conferences, we ended up deeply analyzing more than 100 papers in the final selection, developing a taxonomy of the Hadoop research from 2008 until 2013. This paper was used as an initial guideline to possible research topics regarding Hadoop, which later was directed to the hybrid storage approach.

Paper 2: GOLDMAN, A. ; KON, F. ; PEREIRA JUNIOR, F. ; POLATO, I. ; PEREIRA, R. F. . "*Capítulo 3: Apache Hadoop: Conceitos teóricos e práticos, evolução e novas possibilidades.*" In: SOUZA, A. F.; CESAR JUNIOR, R. M.; GALANTE, R;. (Org.). XXXI Jornadas de atualizações em informática. 1ed.Porto Alegre: SBC, 2012, pp. 88-136.

This book chapter was written as a basic tutorial for students starting in the Hadoop framework. It describes the Hadoop history presenting its main components with educational examples, from the installation and deployment modes to the development of a simple MapReduce application using Hadoop.

Paper 3: POLATO, I.; BARBOSA, D.; HINDLE, A.; KON, F. . "*Hadoop branching: Architectural impacts on energy and performance.*" In: 2015 Sixth International Green and Sustainable Computing Conference (IGSC), 2015, Las Vegas. 2015 Sixth International Green and Sustainable Computing Conference (IGSC). p. 1-4.

This short paper presents a summarized version of the grounding experiments performed on 12 initially selected Hadoop releases. We also presented a source code analysis conducted to explain the changes in the power consumption in different Hadoop releases due to architectural modifications with the insertion of the YARN resource manager.

Paper 4: POLATO, I.; BARBOSA, D.; HINDLE, A.; KON, F. . "*Hadoop energy consumption reduction with hybrid HDFS.*" In: the 31st Annual ACM Symposium, 2016, Pisa. Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16. New York: ACM Press, 2016. p. 406-411.

This is the paper that presents the main results achieved during our research. It illustrates the $HDFS_H$ research and the results achieved when using the hybrid storage space. We analyzed the changes in power consumption and performance increase concerning the approach over 6 Hadoop releases from the 3 development branches.

The remainder of this text is organized as follows. Chapter 2 presents the background concepts; Chapter 3 discusses related work; Chapter 4 introduces the grounding experiments which guided the modeling of $HDFS_H$; Chapter 5 presents the $HDFS_H$ hybrid storage model, development, evaluation, and results; Chapter 6 discusses our results, pointing out the key findings and threats to the validity of our approach; finally, Chapter 7 delineates our conclusions and future works that can spawn from this thesis.

Chapter 2

Background

Apache Hadoop is best known for being the most famous and open-source implementation of Google's MapReduce computing model and caught the attention of both the academic and industrial communities. The major reasons for its wide adoption are related to its scalability, fault tolerance, and the ability to perform parallel and distributed computations on commodity computing clusters. In this chapter, we present an overview of the background concepts that guided our studies on the Apache Hadoop and its subsystems MapReduce and HDFS.

2.1 The Apache Hadoop Framework

Parallel and distributed computing currently have a fundamental role in data processing and information extraction on large datasets. Over the last years, commodity hardware became part of clusters, as the x86 platform copes with the need for having a good cost/performance ratio, while decreasing maintenance costs. Alongside, several programming models and frameworks were developed to promote the use of grids and clusters to perform computations in a distributed and parallel way. We decided to investigate the Apache Hadoop framework, motivated by its wide adoption by the communities of developers, users, and researchers.

The MapReduce paradigm (GGL03; DG04), through its open source middleware implementation, Apache Hadoop, comprises a popular approach to processing large datasets. Hadoop is a relatively low-cost way to obtain an infrastructure capable of carrying out massive computations with a high level of parallelism due to its ability to achieve high computing power by allowing the use of commodity hardware – although not mandatory – in large-scale clusters. At the beginning it was exclusively based on the MapReduce paradigm, it uses a map function that generally filters and sorts the input data, and a reduce function, responsible for summarizing the results.

The Hadoop middleware framework has four major components: the Hadoop Common is the core system that provides support for the other three modules with common utilities; the *Hadoop Distributed File System* (HDFS), a block file storage, designed to reliably hold very large datasets using data replication (SKRC10); *MapReduce*, a system for parallel processing of large data sets using the homonymous programming paradigm; finally, the more recent component, the YARN resource manager and scheduler, which is included in Hadoop 0.23.x and 2.x releases.

Hadoop is now one of the largest projects from the Apache Foundation and rapidly evolved since the first releases: it had more than 70 releases since it was made a top-level Apache project, in January 2008. Figure 2.1 presents the Hadoop versions and its release history. In December 2011, Hadoop reached an important milestone: version 1.0.0 was released, bringing stability and popularity to the framework. At the same time, a new component named MapReduce 2 was being developed, which was later included in two of the three development branches. The "evolved" MapReduce paradigm was in fact part of an important component in the Hadoop project: the YARN resource manager. To accommodate such component, the Hadoop project went through an architectural modification, creating along the process, three development branches. The first branch – 1.x – contains the original Hadoop project, focused exclusively on the MapReduce paradigm. From

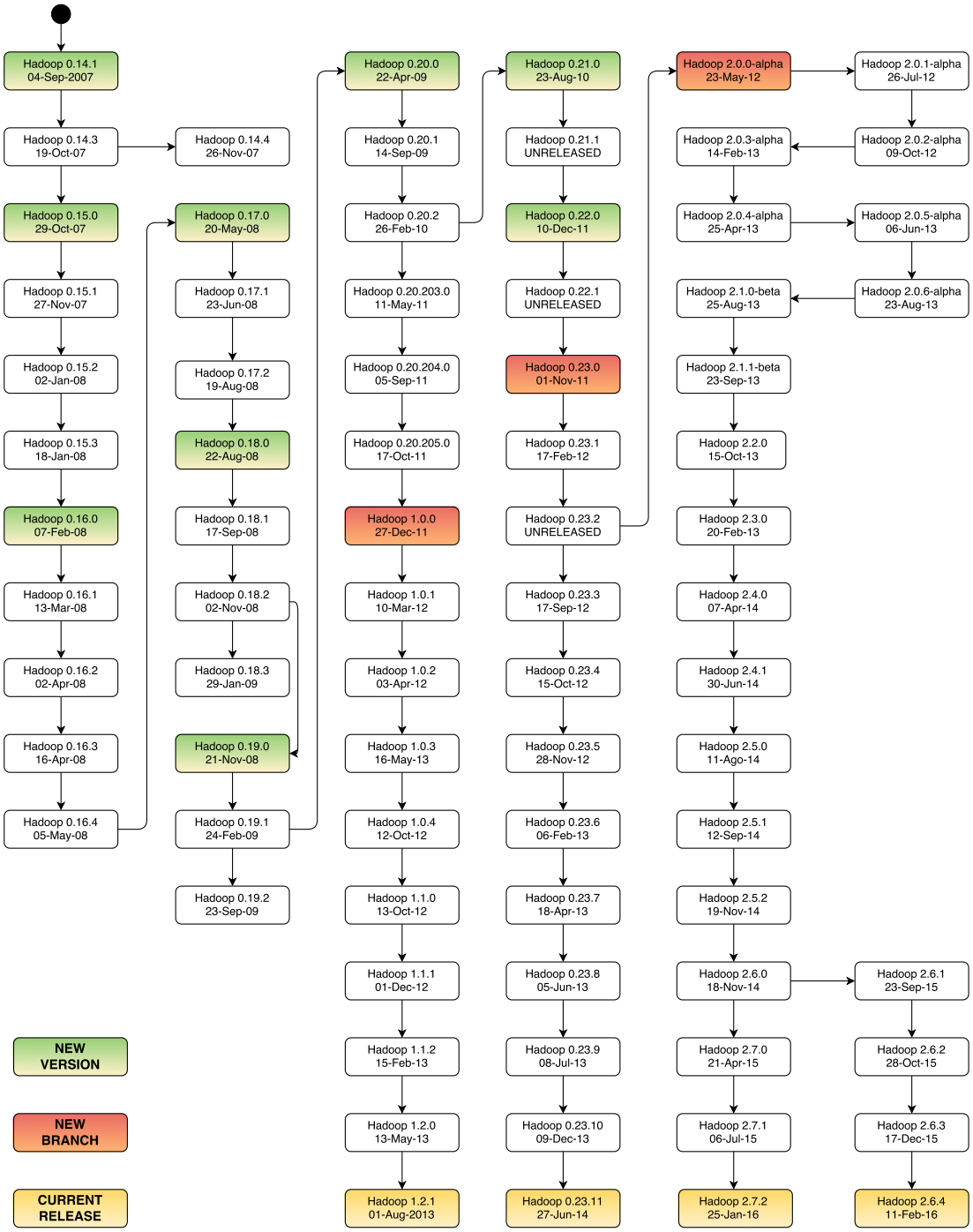


Figure 2.1: *Hadoop Releases History*

late 2011 on, the YARN component was shipped on the Hadoop 0.23.x releases. A few months later, branch 2.x was created. These two branches derived from branch 1.x, but suffered several architectural changes to receive the newly developed Yet Another Resource Negotiator (YARN), the result of a separation between the MapReduce processing engine and the resource management, previously implemented together. The major claimed innovation was the multi-tenancy concept, allowing several users to perform jobs on the same MapReduce cluster. Additionally, YARN was not tied only to MapReduce applications, allowing other processing models, such as message passing and interactive applications. Figure 2.2 shows the new layer introduced by the YARN development and the separation of the resource management and data processing modules.

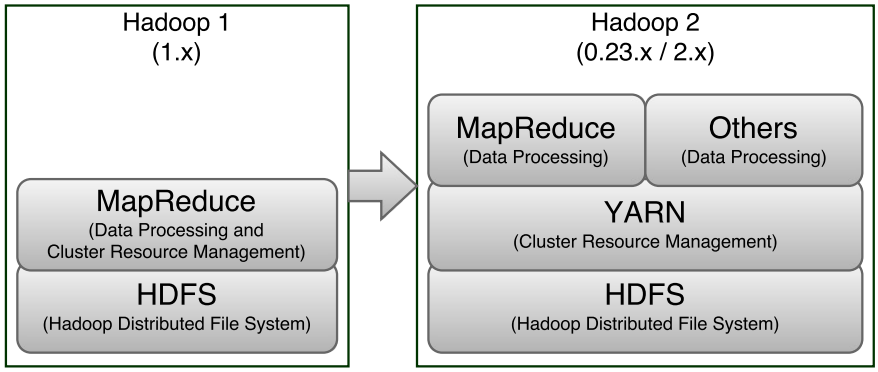


Figure 2.2: *Hadoop Architectural Modifications*

2.2 MapReduce

The MapReduce programming paradigm, now highly used in the context of Big Data, is not new. One of the first uses of this paradigm was on the LISP programming language. It relies basically on two functions, Map and Reduce. The first generates maps based on a given user-defined function, and the second groups Map outputs together to compute an answer. The paradigm is very useful when dealing with batch programs where data is manipulated in a sequential way. Recently the MapReduce paradigm attracted attention because of its applicability to parallel computing. Google's MapReduce composed initially of the GFS distributed filesystem (GGL03) and their implementation of MapReduce (DG04), brought to the fore the use of the simple and consolidated functions Map and Reduce in parallel and distributed computing using Java and C++ libraries. Additionally, the paradigm feeds the Reduce function with the Map function results. This enables parallelism since partitioned portions of data may be fed into different instances of Map tasks throughout the cluster. The results are gathered, used as inputs to the Reduce instances, and the computation is accomplished. The great novelty here is that the approach hides from users a lot of the complexity of parallelization and distribution. Users can focus on the functionality of their programs and the framework abstracts the complexity and controls the infrastructure.

Based on this novel approach, Doug Cutting, an employee of Yahoo! at the time, and Mike Cafarella, a professor at University of Michigan, developed Hadoop, later called the Apache Hadoop framework. It is an open source implementation of Google's MapReduce approach. It uses the same idea from Google: hiding complexity from users, allowing them to focus on programming the solution. Mostly known by its MapReduce implementation, Apache Hadoop also has an ecosystem composed of several applications, ranging from data warehousing to a data flow oriented programming language. The Apache Hadoop framework provides solutions to store, manipulate and extract information from Big Data in several ways. The framework has evolved over the last few years and promotes data integrity, replication, scalability, and failure recovery in a transparent and easy-to-use way. All these factors have made Apache Hadoop very popular both in academia and in industry.

The data processing strategy employed by Hadoop MapReduce relies on the same two primitive functions: Map and Reduce. Behind this simple abstraction is a single fixed data flow. A MapReduce job is divided into Map and Reduce tasks, and assigned to idle slots of workers according to these two stages. Thus, there are two types of workers, Mappers and Reducers. Each input data block will be processed by a Mapper, resulting in intermediate output, which is locally sorted, optionally combined from key-value pairs sharing the same key, and stored in local disks of the Mappers. The Reduce phase starts as soon as there are enough Map outputs to start a Reduce task. By default, when 5% of the Map tasks are complete, the scheduler assigns Reduce tasks to workers. The data transfer is performed by each Reducer that pulls and shuffles intermediate results using a one-to-one shuffling strategy. Reducers are responsible for reading intermediate results and merging them to group all values with the same keys. Subsequently, each Reducer applies Reduce to the intermediate values considering these keys to produce the final output that is stored in HDFS. Figure 2.3 presents

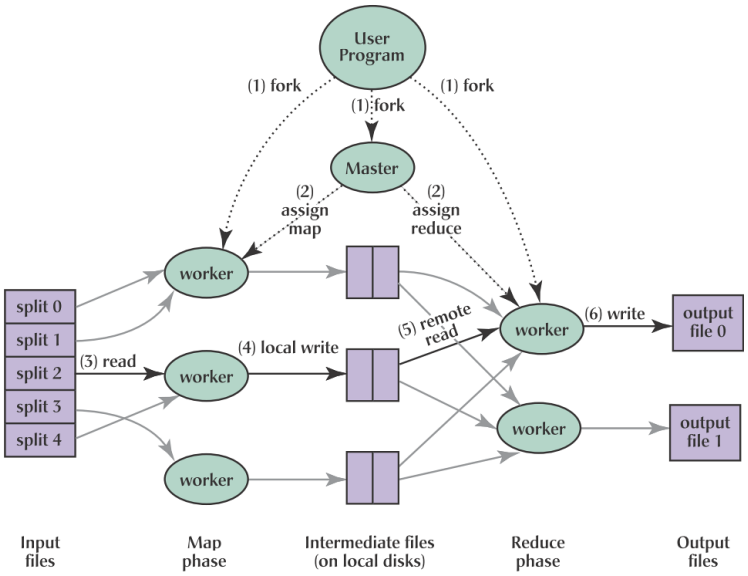


Figure 2.3: *Distributed MapReduce Paradigm (DG08)*

the original distributed MapReduce as conceived by its authors, later implemented on Hadoop.

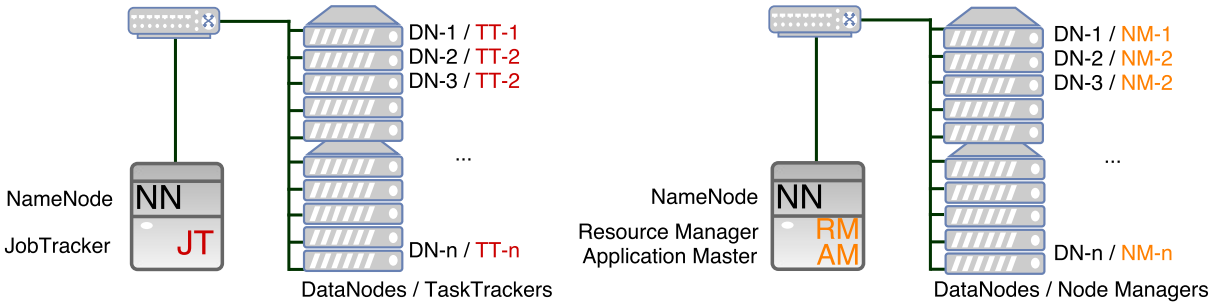
Both job and task control are in charge of the Hadoop daemons. Two daemons are responsible for the MapReduce job workflow in Hadoop. The master or head node of the cluster runs a daemon called JobTracker (JT), which is responsible for controlling the job metadata, status and job scheduling. Each slave node in the Hadoop cluster runs a TaskTracker (TT) daemon, which is responsible for executing the assigned job tasks (Maps or Reduces), reading from and writing data to the HDFS. Jobs are submitted to the JobTracker instance, which designates the TaskTracker workers and controls job scheduling and progress. The MapReduce data processing engine runs on top of the HDFS, generally reading file blocks on Map tasks and writing file blocks with the Reducers' results. These daemons are associated with other HDFS control daemons explained ahead.

2.3 HDFS

The Hadoop Distributed File System is the block storage layer that Hadoop uses to keep its files. Every other layer on Hadoop runs on top of it. HDFS was designed to hold very large datasets reliably using data replication (SKRC10). This allows HDFS to stream large amounts of data to user applications in a reasonable time. Its architecture is composed of two main entities: NameNode and DataNodes, which work in a master-slave fashion. NameNode is responsible for keeping metadata about what and where the files are stored in the file system. DataNodes are responsible for storing the data itself. HDFS works as a single-writer, multiple-reader file system. When a client opens a file for writing, it is granted a lease for the file and no other client can write to the file until the operation is completed. Additionally, after the file is closed, the bytes written cannot be altered or removed except that new data can be added to the file by reopening the file for append. This process works well on the Hadoop distributed multitask approach. Hadoop reads input blocks from HDFS for each job Map task, which are processed and stored locally on disks from the DataNodes. The intermediate results are pulled and distributed to the Reduce tasks. When the processing is complete the results are written back in the HDFS.

This file system works on top of the operating system's file system. Using configuration files and block placement policies, HDFS can place the blocks on the specific physical disks and directories, controlling where each block is stored and its copies replicated. HDFS also works with daemons on the head node and its workers. The NameNode (NN) daemon runs on the master node, managing the file system namespace, mapping the file blocks placement, and their replication, according to a property set on the configuration files. The default replication factor is 3, meaning that the original

block will have two more copies located on different machines of the cluster. Each worker node runs a DataNode (DN) daemon, which controls the locally stored blocks, providing the metadata about them. These daemons control the storage layer of Hadoop, providing services to the upper levels, such as the MapReduce engine. Figure 2.4a presents the logical organization of the Hadoop daemons. The head node runs both the NameNode and JobTracker daemons, while the workers run the DataNodes and the TaskTrackers daemons on releases 1.x. Notice the difference introduced by YARN, explained next.



(a) *Hadoop 1.x releases* **(b)** *Hadoop 0.23.x and 2.x releases*

Figure 2.4: *Hadoop Daemons*

2.4 YARN

With the introduction of YARN in Hadoop, two development branches were created. Hadoop 2.x releases were derived from the 0.23.2 release. The major difference is that some components were re-written to enable support for features such as HDFS Federation, which allows multiple, redundant NameNodes acting together. This is a solution to the single instance of NameNode, a well-known single point of failure on Hadoop, since with a NameNode failure, the whole file system becomes unavailable until its recovery. Releases 0.23.x does not include such feature.

The YARN resource manager was introduced by creating a new layer between the file system and the distributed data processing layer. On Hadoop 1.x releases, the MapReduce engine was responsible for job execution as well as resource allocation. From releases 0.23.x and 2.x on, YARN became responsible for resource management and controlling the applications submitted to Hadoop clusters. YARN brought the multi-tenancy possibility to the framework, a feature where multiple user accounts can submit jobs at the same time to a Hadoop cluster. Consequently, the MapReduce daemons JobTracker and TaskTracker were modified to accommodate such changes. On the head node, they are called Resource Manager (RM) and Application Master (AM). The Resource Manager component is responsible for controlling resource allocation for the cluster, since there is the possibility of parallel job submission from different users. The Application Master is responsible for the control of each application submitted to the cluster. For each MapReduce job submitted to a YARN Hadoop cluster, a new instance of the Application Master daemon is created. The worker nodes daemon were also modified accordingly and named Node Manager, which is the equivalent to the TaskTracker on releases 1.x. Figure 2.4b presents the daemons on YARN releases.

2.5 Storage Devices

Nowadays, a major part of the production clusters in data centers use regular magnetic HDs as their main storage devices. The main reason for this large-scale use is mainly the low price of these devices. Over the years, the cost/GB decreased constantly. Today, it is possible to buy a 4TB hard drive for less than two hundred dollars, which gives us the cost of less than $0.05/GB for the final consumer. If we add the cost of storage services, the price will not increase beyond $0.10/GB (Goo14).

Even though magnetic disks are inexpensive, this does not imply poor manufacturing quality. Actually, hard drive manufacturers have rigid quality control and assurance. Several studies concerning the durability and the analysis of failure metrics on hard drives were conducted over the last years. Most of these studies show that the Mean Time Between Failures (MTBF) and the Annualized Failure Rate (AFR) are three to four times lower than that specified by the manufacturers (SG07). Another study by (PWB07) pointed that the AFR for individual drives in a large population ranges from 1.7% for devices in the first year to 8.6% for three-year old devices. This means that hard drives will fail, and although this is unpredictable, the failure probability is low, less than 10% for a three-year old HD. The major problem occurs when we put together a very large number of drives into a storage server, and several servers into a data center. Hardware vendors can put more than 2,000 drives into one single storage server. With this in mind, failure is not a rare possibility. Hard drives will fail every day in a large data center, independently of their reliability.

As an alternative to the HDs, SSDs are becoming more popular every day. Despite the fact that SSDs are not new (Ren14), over the last five years, it was noticeable the increase on the use of SSDs. The first developments towards the modern SSDs date from the early research about storage systems in the 1950s. Late in the 1970s, IBM developed the first SSDs using semiconductors, but due prohibitive manufacturing cost, the idea was put aside. It was recently, in the late 1990s and beginning of the 2000s that these devices started to be developed and manufactured at scale, using flash-based memories. Recent developments in the last five years produced a wide variety of such devices using several communication interfaces such as Serial ATA (SATA) and PCI Express. Although the price of solid-state drives has decreased significantly over the last three years, the price is still high when compared to HDs. In 2014, the cost/GB of the SSDs sitted between $1.00 and $1.50, and there was expectations that it would decrease to less than $1.00/GB by the end of the year (Hac14). This would represent around 20x the cost/GB for the HDs. From this perspective, the cost on SSDs could still be considered prohibitive for general purpose use. But, only two years later, this scenario changed, where the SSD costs dropped substantially, and sit around 4 to 5 times the cost/GB of HDs (Mea15; All16), being expected to reach 3x the price of HDs cost/GB in 2017. This presents a new scenario, where the use of SSDs is quickly becoming common.

Although SSDs use modern technologies, completely different from HDs, they are still susceptible to failures. Few studies have been developed regarding SSDs failure rates, and the manufacturers do not clearly publish this information. Intel Corporation presented a study showing that SSDs have failure rates lower than 1% for first year devices (Int11). An ongoing research presents a major concern related to SSD failures due to power outages (Lei13). Most of the devices studied so far are not capable of surviving tests with multiple power outages, leading to a complete loss of the stored data. Finally, a study from 2011 points that with newer technologies, SSDs have a higher reliability than HDs, and recommends the use of hybrid environments, mixing HDs and SSDs (Tom11).

While there are still significant differences in the price of these devices, the performance differences of SSDs and HDs shows an even wider gap, but on the opposite side. While most of the SSDs available today are capable of reading random blocks at throughput rates between 250MB/s and 400MB/s, the fastest HDs have a read throughput of less than 170MB/s (Tom13). If we consider the read access times from HDs, including the rotational latency, HDs spinning at 7200RPM have response times in the 11ms range on average. Compared to the SSDs, which have response times around 0.5ms, there is a significant difference.

Another recent study developed by the Storage Networking Industry Association (SNIA) (Kim13), shows the comparison between HDs, SSHDs, and SSDs. The SSHDs are a class of devices that include a small NAND flash memory integrated (generally between 4 and 32GB) in the regular HD. These devices are also known as Hybrid HDs. Figure 2.5 reproduces one of the main tables of the document. The table presents three classes of devices – regular HDs and SSHDs, client SSDs, and enterprise SSDs – and their benchmarks in terms of Input/Output Operations Per Second (IOPS), throughput and response time. The author also separated the IOPS results into two sets: devices that receive a complete purge and have no write history, classified as Fresh Out of Box (FOB), and the devices that were not purged at all. The tests perform reading and writing operations, in

Summary Performance Data – HDD, SSHD, SSD									
Class	Type	FOB IOPS	IOPS (higher is better)			Throughput (larger is better)		Response Time (faster is better)	
Storage Device	Form Factor, Capacity, Cache	RND 4KiB 100% W	RND 4KiB 100% W	RND 4KiB 65:35 RW	RND 4KiB 100% R	SEQ 1024KiB 100% W	SEQ 1024KiB 100% R	RND 4KiB 100% W AVE	RND 4KiB 100% W MAX
HDD & SSHD									
7,200 RPM SATA Hybrid R30-4	2.5" SATA 500 GB WCD	125	147	150	135	97 MB/s	99 MB	15.55 msec	44.84 msec
15,000 RPM SAS HDD IN-1117	2.5" SAS 80 GB WCD	350	340	398	401	84 MB/s	90 MB/s	5.39 msec	97.28 msec
Client SSDs									
mSATA SSD R32-336	mSATA 32 GB WCD	18,000	838	1,318	52,793	79 MB/s	529 MB/s	1.39 msec	75.57 msec
SATA3 SSD IN8-1025	SATA3 256GB WCD	56,986	3,147	3,779	29,876	240 MB/s	400 MB/s	0.51 msec	1,218.45 msec
SATA3 SSD R30-5148	SATA3 256GB WCE	60,090	60,302	41,045	40,686	249 MB/s	386 MB/s	0.35 msec	17.83 msec

Figure 2.5: *Storage Devices Performance Studies (Kim13)*

random and sequential modes. From Figure 2.5, take for instance the client SSDs. These devices are developed for end-users, who do not have special high-performance requirements. For comparison purposes, we choose two devices that use regular SATA3 interfaces, which are common in commodity hardware nowadays. Comparing the 7200RPM HD on the table with the SATA3 SSD listed, we see that the SSD can make 20 times more writing IOPS than the HD, and more than 220 times more reading IOPS than the HD. In the mean case, using 65% random reads and 35% random write operations the SSD can perform up to 25x more IOPS than the HD. Concerning throughput rates, in the same case the SSD is more than 2x faster than the HD on writing operations, and around 4x faster on reading operations. Even if we compare the 15K RPM SAS HD, the performance differences are still significant.

2015 - Summary Performance Comparison by Storage Class											
Storage Class				IOPS FOB T4Q32	IOPS Steady State PTS-C T2Q16; PTS-E T4Q32; NVDIMM T128Q4			Bandwidth T1Q32; NVDIMM T32Q1		Response Time T1Q1	
	Category	Device Type	Capacity	RND 4KiB 100% W	RND 4KiB 100% W	RND 4KiB 65:35 RW	RND 4KiB 100% R	SEQ 128KiB 100% W	SEQ 128KiB 100% R	RND 4KiB 100% W Ave	RND 4KiB 100% W Max
HDD & SSHD - PTS-C v1.2											
1	SSHD	7,200 RPM 2.5" SATA Hybrid	500 GB	3,398	77	62	113	79 MB/s	76 MB/s	12.55 mSec	163.10 mSec
2	SAS HDD	15,000 RPM 2.5" SAS HDD	146 GB	652	133	248	514	147 MB/s	174 MB/s	2.24 mSec	24.54 mSec
CLIENT SSDs - PTS-C v1.2											
3	mSATA	mSATA 1.8" MLC	250 GB	63,127	5,927	13,545	79,423	300 MB/s	526 MB/s	0.13 mSec	11.86 mSec
4	M.2 x4 AHCI	M.2 x4 Gen 3 2280 MLC	128 GB	77,757	6,307	17,041	172,881	623 MB/s	1,895 MB/s	0.16 mSec	54.19 mSec
5	SATA	SATA6Gb/s 2.5" MLC	400 GB	52,723	34,406	50,646	61,178	428 MB/s	475 MB/s	0.082 mSec	3.13 mSec

Figure 2.6: *Storage Devices Performance Studies (Kim15)*

The same study was updated two years later, in 2015 (Figure 2.6). It shows the surprising evolution in the SSD industry. For some devices, the number of IOPS and the bandwidth increased greatly – specially in the client SSDs – accentuating the large performance differences between HDs and SSDs. We must point out that such performance gap between HDs and SSDs is a result of their conception and architecture: one is composed of magnetic disks with mechanical parts such as a motor and reading heads, while the other is a NAND flash-based system. For our research, this difference is important, but even more important is the systems adaptation to accommodate these

devices properly. Incorporating such different technologies, in a suitable way, achieving better use of each device is our challenge. The SSHDs devices, which combines in the same device both SSD and HD technologies were not tested with our approach, although they can improve performance by storing the most frequently accessed data in their NAND flash memory.

Besides the performance and cost/GB discussions, we also have to consider the power consumption of HDs and SSDs, and their impact on several aspects. First, the energy costs of powering and cooling data centers represent one of the major costs annually, reaching approximately 25 to 30% of the Total Cost of Ownership (TCO) of a data center (Ham10). While SSDs are still somewhat expensive in the cost/GB and have a limited storage space, they demand less power and have better performance than HDs. On average, an SSD needs about 10x less power than an HD on idle mode, and up to 3x times less power than HDs at maximum write throughput (Tom13). From this point of view, the use of SSDs can be more cost effective in terms of power in the long run. Yet, besides the lower power requirements, SSDs generate less heat than HDs, which are constantly spinning, even in idle mode. Thus, in addition, SSDs also help saving energy used by cooling systems, such as air conditioning.

Although solid-state drives are not the default storage devices yet, there is an increasing demand for the use of SSDs, balancing the overall costs, including initial cost/GB, performance, durability and energy consumed. There are several tools that can calculate the TCO for data centers under different scenarios using HDs and SSDs, but this is out of our scope. Our research intends to develop an appropriate environment to achieve the best performance when using SSDs on hybrid environments along with HDs, taking into account makespan performance and energy needs. To achieve our objective, we propose a set of modifications to the Hadoop Distributed File system to properly accommodate such different devices transparently to the end-user after the proper configurations.

2.6 Green Computing

Although not new, the term Green Computing became increasingly important over the last decade. Energy consumption is a timely and important topic: more than ever, data is easily gathered, stored, and processed. With the constant reduction on hardware cost, the number of data centers increased substantially in the recent past. Consequently, the energy consumption attributed to computing processes climbed rapidly. At the same time, energy costs have also increased due to its tightly coupled production using natural resources. All this context brought to light the Green Computing research in Computer Science. There is an actual need for efficient energy use for computing since its impact on TCO is the larger portion of the cost matrix of a data center today.

Murugesan (Mur08) defines Green Computing as "the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems – such as monitors, printers, storage devices, and networking and communications systems – efficiently and effectively with minimal or no impact on the environment." This definition is important because it sheds light on several research topics from software engineering to big data processing explored in the search for energetic efficiency. In the Green Computing research area, energy profiling represents an important topic: it investigates the behavior of software execution under different environments and workloads to generate an energy consumption profile of a given platform, software or infrastructure. Our work is heavily based on the energy profiling approach for the Hadoop platform, in search for increasing energy consumption efficiency in a hybrid storage environment while processing data using the MapReduce paradigm.

Chapter 3

Related Work

The Apache Hadoop framework became the most popular platform to process big data using the MapReduce paradigm. Supported by large companies such as Yahoo!, Facebook, Amazon, Cloudera, among others, Hadoop research was motivated by the need for new features and improvements of existing components. Our main research topic connects two Hadoop subjects: (1) Storage and Replication and (2) Energy Management. We developed practices involving different types of devices, changing the storage management layer in HDFS. Such modifications are mainly concerned with the energy consumption decrease as a consequence of SSD performance. We noticed that most of the research related to our work are isolated into these two subjects. On the one hand, Hadoop performance research increases performance through several means: component improvement, new features design, changing computation scalability; nevertheless, the results achieved are not concerned with the energetic impact on the platform. On the other hand, most of the energy-aware research on Hadoop is related to the holistic view of the infrastructure, concerned with data center zoning techniques, reducing energy consumption by shutting down racks or nodes, or even moving data into hot and cold storage zones. In the following sections, we analyze the Hadoop project in these related areas.

3.1 Hadoop Research

Hadoop had a fast evolution with significant changes over the last years. Such a rapidly growing project received attention from the research community. The result is the large corpus of Hadoop research produced since 2008. To investigate this scenario and have a comprehensive view of the project and its components, we conducted a systematic literature review focused on the main research on Apache Hadoop and the contributions developed by these works (PRGK14). One of the objectives of such literature review was the proposition of a taxonomy and classification of the related research literature into categories. Our taxonomy was based on the project structure proposed by the Apache Hadoop developers. As a result, the taxonomy grouped the analyzed studies into four major categories related to the framework's architecture: MapReduce, Storage, Ecosystem, Miscellaneous. The first category, MapReduce, includes studies that develop some solution involving the paradigm and its associated concepts, like scheduling, data flow, and resource allocation. The Data Storage & Manipulation category includes studies comprising HDFS, gathering studies involving storage, replication, indexing, random access, queries, research involving DBMS infrastructure, Cloud Computing, and Cloud Storage. The third category includes studies containing new approaches to the Hadoop Ecosystem, including papers related to both existing and new components created to cope with specific classes of problems. Finally, the last category received studies that did not fit well in the other three categories, including research involving GPGPU, cluster energy management, data security, and cryptography. Figure 3.1 shows the final organization of our classification effort. We were able to fit the selected papers into one or more correlated categories, which promoted a general view of the areas in the framework that received more contributions from 2008 to 2014. We investigated each selected paper in detail, and summarized the results into

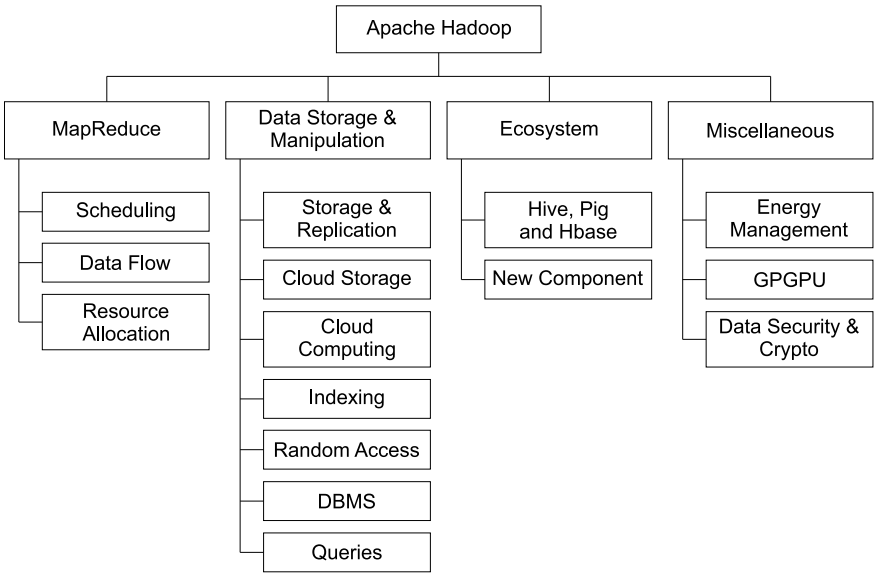


Figure 3.1: *Taxonomy Hierarchical Organization.*

sections of our systematic review. This was our initial effort to have an overview of the Hadoop research, which directed us into the storage and energy management Hadoop areas.

MapReduce: Scheduling, Data Locality and Resource Allocation

Hadoop performance (makespan reduction) is a well explored area, with most of the research concentrated on this topic. Scheduling is considered crucial to Hadoop performance. In this sense, the selected papers that were allocated in the scheduling, data flow, and resource allocation categories are mainly concerned with this issue. Some works propose multi-queue schedulers to improve performance (ZWM+12; TZHZ09; KKVR12). Other authors use different approaches to achieve it, such as the data locality aware schedulers (ZBSS+10; HLS11; ZFF+11; HS11; ZWYD12; HRS12; IJL+12; ZZF+11; YYH11; TZSC11), which are concerned with the correct allocation and placement of Map and Reduce tasks. Performance problems may also be tackled using historical data from cluster nodes, which allows, e.g., the speculative execution of MapReduce tasks (ZKJ+08; LWH11; RD11; CZG+10). Scheduling in heterogeneous clusters is also an important topic addressed by some studies (TZHZ09; ACRV12; KKVR12). Ultimately, some approaches develop mechanisms of reuse of intermediate data among MapReduce jobs using common datasets (KKVR12; NPM+10; SLT11).

Data locality as a performance issue has also been studied in Hadoop: He *et al.* (HLS11) propose a new scheduler with the premise that local Map tasks are always preferred over non-local Map tasks, no matter which job a task belongs to. Thus, it improves performance by avoiding data transfer in Map tasks, which may degrade job execution performance. Zhang *et al.* (ZZF+11) propose a scheduling method called next-k-node scheduling (NKS) that improves the data locality of Map tasks. The scheduler calculates a probability for each Map task in a job, generating low probabilities for tasks that do not have their input data not stored locally. In this way, it reserves tasks with lower probabilities to the nodes holding their input data, improving data locality. Partitioning skew – the case where the computational load is unbalanced among Map and/or Reduce tasks, generally causing performance degradation – is a problem that is present in some MapReduce applications. Hammoud *et al.* (HS11) present another approach discussing the data locality problem that deals specifically with Reduce tasks. The Reduce phase scheduling is modified to become aware of partitions, locations, and size, to decrease network traffic, increasing overall job performance; the work of Zhang *et al.* (ZWYD12) also deals with the Reduce tasks data locality by proposing a two-phase execution engine of Reduce tasks to cope with remote data access delays that may degrade system performance.

Regarding the Hadoop MapReduce data flow, several works improve performance (WQY+11; VBBE12; HWL11; IJL+10; KBHR12; VCC11; VCKC12; ZC11; LMA+10) or augment features to meet specific requirements (EER11; LAY11; BHBE12; ELcF10; GC12; LZZ12; BWR+11; ZGGW11). Hadoop data flow has already been changed in several ways: Wang *et al.* (WQY+11) propose an acceleration framework that overlaps the Shuffle, Merge, and Reduce phases, increasing the throughput of Hadoop and reducing CPU utilization; on the other hand, the proposal of Vernica *et al.* (VBBE12) focuses on the interaction of Mappers, introducing an asynchronous communication channel between Mappers, using a transactional distributed meta-data store (DMDS). In this way, Mappers can post metadata about their state and check the state of all other Mappers, allowing global optimizations during the execution; ultimately, Ho *et al.* (HWL11) and Ibrahim *et al.* (IJL+10) concentrate their efforts on improving performance by changing the data flow in the transition between Mappers and Reducers. Originally, Hadoop employs an all-to-all communication model between Mappers and Reducers. This strategy may result in saturation of network bandwidth during the shuffle phase. This problem is known as the Reducers Placement Problem (RPP).

These aspects are the main ones explored in the paradigm to improve performance in Hadoop jobs via MapReduce. We noticed that these solutions could improve significantly the performance of Hadoop, but only for applications that target specific research problems. Nevertheless, none of these solutions are concerned with the energy consumption performance of the framework. Therefore, our research set focus on the energy-aware performance of Hadoop, from the perspective of using hybrid storage devices within the HDFS. Furthermore we did not commit any changes in the MapReduce workflow.

HDFS: Data Storage and Manipulation

HDFS is the block storage layer that Hadoop uses to keep its files, designed to hold very large datasets reliably using data replication (SKRC10). This allows HDFS to stream large amounts of data to user applications in a reasonable time. HDFS has received several contributions that implement enhancements so that it can be used in different types of approaches in MapReduce computations.

We observed in our literature research that HDFS was modified to increase performance in several ways: tuning the I/O mode (JOSW10; ZYLL11; SRC10), solving data placement problems (XYR+10), or adapting it to deal with small files (JOSW10; DQZ+10) since HDFS was not originally designed to store such files. Some works replace the original HDFS with a more suitable solution for specific compatibility problems (MOT11; KVV11) or to support areas such as Cloud Computing (KVV11; GhJnBwY11). Differently from these solutions, our research is dedicated to promote performance increases by designing a mixed storage environment using SSDs and HDs.

A particular study that contributes in different ways to the framework was developed specifically to improve Hadoop's performance. The work of Jiang *et al.* (JOSW10) presents a MapReduce performance study, focusing on Hadoop's performance bottlenecks. Jiang *et al.* propose known alternative methods as solutions to tune MapReduce performance. They enhanced the way a reader retrieves data from the file system, the I/O mode, with a direct I/O support, which outperforms streaming I/O by 10%. The authors also proposed an optimization to HDFS to deal with small files; HDFS may have loss of performance when dealing with a large group of small files due to its strategy of keeping all metadata in the master node memory. This approach allows DataNodes to save some metadata of small files in their memory. Thus, the number of read and write requests received by the NameNode is reduced, improving performance when dealing with small files. Similarly, Shafer *et al.* (SRC10) analyze the HDFS performance bottlenecks under concurrent workloads. The authors claim that HDFS performance can be improved using application-level I/O scheduling while still preserve portability. Authors also explore solutions like pre-allocating file space on disk, adding pipelining and prefetching to both task scheduling and HDFS clients, and modifying or eliminating the local file system as a way to improve HDFS performance. But, since portability is a project premise in Hadoop, some of these changes may not be fully convenient, because of the portability reduction they may cause.

Because HDFS is designed to store and keep large files/datasets, Dong *et al.* (DQZ+10) propose a novel approach to improve the efficiency of storing and accessing small files on HDFS. In this approach, characteristics of file correlations and access locality of small files are considered for storing and accessing them. First, all correlated small files are merged into a larger file to reduce the metadata on NameNode. Second, a two-level prefetching mechanism is introduced to improve access efficiency. The approach is addressed to solve the small file problems of a specific resource sharing system, to store and share courseware mainly in the form of presentation files and video clips. The efficiency problem of storing and accessing small files on HDFS is caused mainly by high memory usage in the NameNode, which keeps all metadata in memory. Therefore, the greater the number of files, the larger the memory used. Performance is also affected by file correlations for data placement, and without a prefetching mechanism for reads, a considerable overhead may be generated.

Focusing on the same problem of correlated data placement, Eltabakh *et al.* (ETO+11) propose CoHadoop, a lightweight extension of Hadoop that allows applications to control where data are stored. CoHadoop addresses Hadoop's lack of ability to collocate related data on the same set of nodes, which can be used to improve the efficiency of many operations, including indexing, grouping, aggregation, columnar storage, and joins. The proposal extends HDFS to enable storage of related files at the file system level. The extension requires minimal changes to HDFS to be used, such as a new file property to identify related data files and to modify the data placement policy of HDFS to collocate all copies of those related files. Exploiting data placement on HDFS, Xie *et al.* (XYR+10) propose a new strategy to HDFS running on a heterogeneous cluster. The approach focuses on distributing a large dataset to the nodes according to the computing capacity of each one. Two algorithms were implemented and incorporated into HDFS. The first one initially distributes the file blocks from a dataset to the cluster nodes. The second data placement algorithm is used to solve data skew problems, reorganizing the file blocks distribution along the cluster. This algorithm is used in two specific situations. The first is when new nodes are added to the cluster. The second is when new data is appended to existing input files. In both cases, the file blocks are redistributed, since the initial data placement can be distorted.

Hadoop may work using a replacement file system, when changes in HDFS are not possible or easy to be made. Mikami *et al.* (MOT11) proposes the use of a new file system named GFarm. It is POSIX compliant and uses data locality, which makes it suitable to be used on Hadoop. GFarm can also be used to run MPI applications. Thus, the same data used on MapReduce applications can be used on different applications such as POSIX compliant and MPI applications. This would not be possible when using HDFS without generating extra copies of these data. To integrate GFarm into Hadoop, a plugin named Hadoop-GFarm was designed, implemented, and tested with MapReduce applications. Several other file systems also received attention from the community as an effort to promote alternatives to replace HDFS. Apache Cassandra (`http://cassandra.apache.org/`), Ceph ((MEMRB10)), GPFS from IBM (O'M11), Lustre (Rut11) and MapR (`http://www.mapr.com/`) have all been successfully deployed as HDFS replacements. Although this is a possibility, we decided to further investigate HDFS and modify it to accommodate our proposal.

Most of the analyzed works changed HDFS to increase performance by modifying the file system to cope with specific needs. Generally, the solutions emphasize restricted applications on specific classes of problems. Additionally, most of the approaches require a certain degree of modification on Hadoop source code. Few solutions (ETO+11; XYR+10) also propose block placement policies, besides the modifications, to achieve the results. On the extreme modification, HDFS was replaced to cope with performance issues on some applications. Yet, these approaches only targeted HDFS performance without considering energy consumption or using a combination of different storage devices, which are the main goals of our research.

3.2 Solid-State Drives on HDFS

The use of SSDs as a storage solution on clusters is such a recent trend in the industry that the first proposals for MapReduce clusters only recently appeared. Most of the research up to date tends to analyze whether MapReduce can benefit, in terms of performance, when deploying HDFS using SSDs. The work of Jeon *et al.* (JEMK13) analyzes the Flash Translation Layer (FTL) – the core engine for the SSDs – to understand the endurance implications of such technologies on Hadoop MapReduce workloads. As a result, the research presents the behavior of SSD for Hadoop-based workloads including wear-leveling details, garbage collection, translation and block/page mappings. The research of Kang *et al.* (KsKMP13) explores the benefits and limitations of in-storage processing on SSDs (the execution of applications on processors in the storage controller). This approach benefits from characteristics such as high performance on concurrent random writes, powerful processors, memory, and multiple I/O channels to flash memory provided by the SSDs, enabling in-storage processing with small hardware changes. The authors implemented a model (Smart SSD model) that uses an object-based protocol for low-level communication with the host, extending the Hadoop MapReduce framework to support a Smart SSD. Experiments show benefits such as increase of performance and reduction of total energy consumption.

Other researchers focus on incorporating SSDs into HDFS using caching mechanisms to achieve better performance. Zhao *et al.* proposed HyCache, an additional middleware layer between the distributed file system and the underlying local file systems, using SSDs to provide a file system with high throughput and low latency. MpCache (WJY14) is a SSD based hybrid storage system that caches both the input and localized data, dynamically tuning the cache space allocation between them. VENU (KIB14) is a dynamic data management tool for Hadoop that uses SSDs as a cache only for the workloads that can benefit from the use of SSD. From the same authors, HatS (KAB14) is a redesign of HDFS into a multi-tiered storage system, integrating heterogeneous storage technologies into Hadoop. An *et al.* (AKJ15) developed a mechanism that, given a block scheduled to be processed, loads the block into the SSD before a MapReduce I/O request for that block take place, therefore processing all blocks from SSD. Although all works presented performance improvements on Hadoop, they are focused on transparent data caching, differently from our hybrid design, which is also intended to increase performance, but considers SSDs as part of the resource allocation mechanism while targeting energy issues in the framework.

A few works also discuss SSDs' impact on Hadoop (MLK14; KC14; SK14), sometimes focusing on using SSDs as the sole storage device under HDFS, sometimes developing a single comparison between the use of SSDs and HDs. A hybrid approach using HDs and SSDs was investigated by Wu *et al.* (WLX+13). Differently from our approach, they developed a network bandwidth guide related to the HDFS replication factor and proposed a temporary data storage using both DRAM and SSDs. Another hybrid approach (TFL14) tested the use of SSDs and HDs under MapReduce, Hive and HBase workloads, but without the possibility of storage space customization, using all available space from HDs and SSDs in the hybrid configuration. Our approach tends to be more affordable, since we developed a hybrid file system that seamlessly couples the best from HDs (affordable cost per GB, high storage capacity, and, to some extent, endurance) and SSDs (high performance and low energy consumption rates) in a configurable fashion.

Recently, the Hadoop team of developers implemented an approach to incorporate hybrid storage into the HDFS. On the latest releases – 2.7.0 and 2.7.1 – the project made available the implementation of new storage policies. Two of the new policies make use of SSDs, but contrary to our approach, they only allow the use of all data on SSDs (All_SSD); or the use of one block replica stored in the SSDs, keeping the other replicas in the HDs (One_SSD). To the present moment, these newly implemented policies do not allow the controlled hybrid storage of blocks guaranteeing the uniform distribution of blocks over HDs and SSDs as we do in our approach. Also, there are no mentions of performance increase or energy consumption testing of such policies, which is the focus of our work.

3.3 Energy and Green Computing Research

Energy consumption has become a vital issue regarding computation costs. Over the past few years, Hadoop energetic research surfaced, presenting several works on the green computing area. Several papers discuss the cluster energy management problem generically. Regarding Apache Hadoop clusters, Li *et al.* (LAY11) propose an algorithm for maximizing the throughput of a rack of machines running a MapReduce workload, subject to a total power budget. The main idea is to optimize the trade-off between job completion time and power consumed. The novelty in the approach relies on the accounting for thermally-induced variations in machine power consumption. The algorithm minimizes job completion time (or equivalently, maximizes the computational capacity of a cluster) for any given power budget.

Lang *et al.* (LP10) investigated the processes of powering down MapReduce clusters during periods of low utilization. The authors examined a preexisting technique called "Covering Set" that keeps a fraction of the cluster nodes powered up during these low utilizations intervals. The research also proposed a new technique called the "All-In Strategy", which uses all the nodes in the cluster to accomplish the computations, and then power down all the nodes during low utilization phases. The same technique was used on the work of Yazd *et al.* (YVM12), which proposed a block placement policy for replicas in the HDFS. Although only tested on simulations, the authors claim that keeping only a subset of servers constantly on can boost the energy efficiency of Hadoop data centers.

Phan *et al.* (PIAB15) analyzed the impact of speculative execution on the performance and energy consumption of Hadoop clusters. The authors also found a strong correlation between time and energy consumption: straggling detection in Hadoop is not accurate and may lead to excessive unnecessary speculative execution, increasing the energy consumption of Hadoop clusters.

The work of Cheng *et al.* (CLJZ15) focuses on Hadoop task assignment on heterogeneous environments. The key finding is that not all job makespan reduction promotes energy consumption reduction. The authors proposed a heterogeneity-aware and adaptive task assignment approach that promotes energy efficiency of a Hadoop cluster without knowledge of the workload properties.

GreenHadoop (GLN+12) is an Apache Hadoop variant for data centers powered by photovoltaic solar (green energy) arrays and electrical grid (brown energy) as a backup. The objective is to investigate how to manage the computational workload to match the green energy supply in small/medium data centers running data-processing frameworks. However, scheduling the energy consumption of MapReduce jobs is challenging because they do not specify the number of servers to use, their run times, or their energy needs. Moreover, power-managing servers should guarantee that the data to be accessed by the jobs remain available. GreenHadoop seeks to maximize the green energy consumption of the MapReduce workload, or equivalently to minimize its brown energy consumption. GreenHadoop predicts the amount of solar energy that is likely to be available in the future, using historical data and weather forecasts. It also estimates the approximate energy needs for jobs using historical data. By using these predictions, GreenHadoop may then decide to delay some (low-priority) jobs to wait for available green energy, but always within their time bounds. If brown energy must be used to avoid bound violations, it schedules the jobs at times when brown energy is cheaper, while also managing the cost of peak brown power consumption. GreenHadoop controls energy usage by using its predictions and knowledge of the data required by the scheduled jobs. With this information, it defines how many and which servers to use and transitions other servers to low-power states to the extent possible.

Another approach to energy management in clusters is the GreenHDFS (KB10; KAEN11). Instead of dealing with the MapReduce component of Apache Hadoop, it deals with the HDFS. GreenHDFS partitions cluster servers into hot zones, used for frequently accessed files, and cold zones, for rarely used files. This approach enables energy saving by putting cold zone servers to sleep. To do so, a migration policy moves files between zones accordingly. Initially, this policy was reactive and used historical data to move files between zones. This approach was improved creating a predictive file zone placement, which defines the initial placement of a file, and then uses a predictive file migration policy. This approach uses supervised machine learning to train its file attribute component and to manage changes between zones.

GreenPipe (MWZL12), provides a specific solution for bioinformatics, but its main objective is related to energy consumption problems. GreenPipe is a MapReduce-enabled high-throughput workflow system for bioinformatics applications, which defines a XML based workflow and executes it on Hadoop. Workflow execution is divided in two modes. In the first one, called physical mode, the XML workflow is translated into MapReduce jobs and launched on a physical Hadoop cluster. The second mode, called virtual mode, works with virtualization, obtaining virtual machines from IaaS (Infrastructure as a Service) platforms and running Hadoop jobs in the VM cluster. Authors also address the optimizations of the planning and job priority, and energy efficiency by introducing a power-aware scheduling algorithm in the workflow engine. The scheduler tries to allocate VM resources based on the power consumed by the applications. It also tries to group VMs with similar energy traces in a physical machine, reducing the energy consumption without sacrificing application performance.

Still dealing with virtual environments, Feller *et al.* (FRM13) developed a Hadoop performance comparison between traditional and virtual clusters, analyzing the energetic impact of separating data from services. The results showed that Hadoop performance on physical clusters is significantly better than on virtual clusters, and that data separation is feasible, but greatly affects performance, and consequently energy consumption.

Green Computing is a strong motivation to achieve results that can reduce environmental issues, such as high energy consumption. But, instead of focusing on the whole infrastructure, we focused on the software responsible for controlling the storage space. Differently from other approaches that focus on Hadoop, we developed a hybrid environment that can increase performance while decreasing energy consumption on cluster nodes and data centers by using SSDs.

Software Energy Consumption

Our energetic Hadoop research was motivated by the green mining methodology (HWR+14; Hin12), the study of energy consumption over multiple versions of a software system, in the area of energy-aware mining of Mining Software Repositories research (Has08). Additionally, multi-version energy consumption analysis has been applied by Hindle et al. (HWR+14). The impact of software metrics, such as CKJM metrics (Spi05) and software churn (NB05) on software energy consumption was presented by Hindle (Hin13)

Chapter 4

Motivating and Grounding Experiments

In this chapter, we present concepts, research methodologies, and the results obtained in the Hadoop performance and energy consumption analysis. Over time, a large number of Hadoop features have been introduced and deprecated, resulting in three current development branches with several releases. We studied the impact of such feature evolution with regard to resource usage and adoption by practitioners. More precisely, we deployed several representative and judiciously chosen versions of Hadoop on the same cluster, ran a representative workload, and measured performance (job makespan – the time difference between the finish and start of Hadoop jobs) and resource usage (power). Our results help clarify the cost-benefit trade-offs of the various Hadoop versions.

More specifically, we are concerned with Hadoop performance and its impact on energy consumption, focusing on the changes between different releases and versions. Therefore, we mined the StackOverflow question/answer website to gather which versions of Hadoop are most frequently discussed. Using this measure as a proxy for feature adoption, and contrasting that with the corresponding cost-benefit trade-off, we arrived at some surprising findings. Most notably, our resource usage data shows that the YARN resource manager leads to *higher* energy consumption and *worse* performance, while our feature adoption data indicate that users rarely mention or discuss this.

YARN (Yet Another Resource Negotiator) (MVE+14) allows multiple data processing models besides MapReduce to co-exist in a single Hadoop cluster. It is touted as the "Data Operating System" and allows the deployment of sophisticated data analytics solutions that mix multiple data storage solutions. Nevertheless, from a computational point of view, previous versions of Hadoop that use a much simpler resource manager, are equally capable. However, as YARN is part of the latest branch (2.x), and there is a tendency for users to keep up with the latest version, our analysis indicates an unfortunate reality in which a completely unnecessary yet large waste of resources takes place.

To the best of our knowledge, up to this day, no studies targeting the framework's performance over multiple Hadoop releases were carried out. Hadoop performance results from several aspects. Some of them are related to the infrastructure, e.g., hardware, network, operating system; others, are related to Hadoop configurations and job characteristics. A poor performance will have direct impact on resource usage and, consequently, on job makespan. As a result, if more resources are allocated when running jobs, more energy is consumed. Thus, we searched for performance issues in different Hadoop versions through the analysis of the changes across releases. We related the impact of these changes on the cluster energy consumption. The results show that, having the same testbed (hardware, network and operating system), different versions and releases behave differently in terms of performance and energy consumed by Hadoop jobs.

4.1 Versions and Releases

To trace Hadoop history, we analyzed each one of the release logs from the project webpage, searching for the history of previous releases to build a genealogy tree. Figure 4.1 shows the Hadoop project genealogy tree that we built, from version 0.20.0 up to the latest releases, including the

release dates. We built the complete genealogy tree from scratch, starting from the first release available on the project page, 0.14.1, from September 4, 2007. Before version 0.20.0, all releases were made sequentially. Consequently, we decided to keep Figure 4.1 showing releases from version 0.20.x onwards. Although all releases are still listed on the page, the documentation for several releases is no longer available. To find the missing documentation, we used the Internet Archive[1], which crawls and save snapshots from Internet websites. Using this tool, we were able to find the missing documentation and fill in the gaps to build the complete tree.

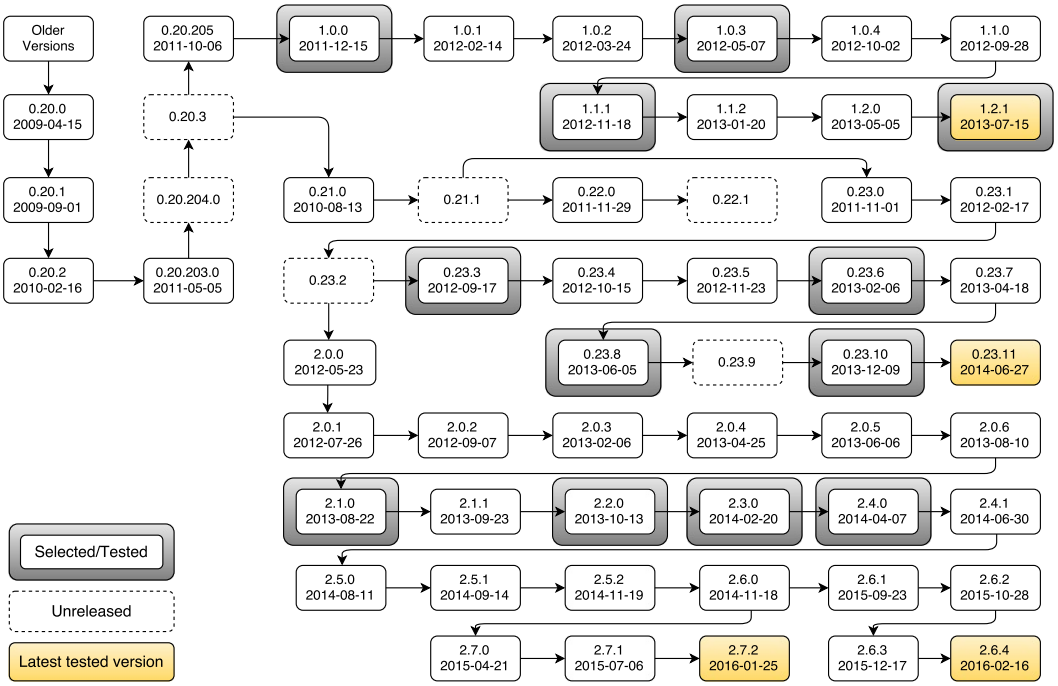


Figure 4.1: *Hadoop Versions Genealogy Tree*

According to our systematic review (PRGK14), most of the Hadoop research was carried out using old Hadoop versions ranging from 0.19.x to 0.22.x. In our conclusions, we noticed that the number of studies contributing to the project increased from 2010 to 2012, but decreased from 2013 on. This means that few studies contributed to the new releases of Hadoop, since they were launched in late 2011 (version 0.23.0 on November 01, 2011; version 1.0.0 on December 27, 2011) and in the second quarter of 2012 (version 2.0.0 on May 23, 2012). Additionally, versions 0.23.x and 2.x adopted a new architecture, introducing the YARN resource manager. Thus, we concluded that the instability of the new releases in the YARN branches and the inertia of adopting brand new Hadoop versions contributed to the low number of works using more recent versions at that time.

In these studies, we selected 12 Hadoop releases to compose our working set, since this research was intended to cover a certain range of releases from the Hadoop project. Our selection included releases from the 3 development branches. From the available releases, we selected the latest one of each branch at the time of experimentation (1.2.1, 0.23.10, and 2.4.0). To distinguish modifications inside the branches, we selected one intermediary version from each branch (1.1.1, 0.23.8, and 2.3.0). Furthermore, in the particular case of the performance studies presented in this chapter, we added two older releases from each branch to complete the set: 1.0.0, 1.0.3, 0.23.3, 0.23.6, 2.1.0, and 2.2.0. We can see the release distribution over time in Figure 4.1 where the selected releases are highlighted. Table 4.1 shows the set of selected versions and releases, with their respective release dates, and previous version.

[1]Internet Archive: http://archive.org/ (Visited on 15/10/2016)

Table 4.1: *Hadoop releases used on the performance and energy consumption studies*

Hadoop version	Release date
1.0.0	December 15, 2011
1.0.3	May 7, 2012
1.1.1	November 18, 2012
1.2.1	July 15, 2013
0.23.3	September 17, 2012
0.23.6	February 6, 2013
0.23.8	June 5, 2013
0.23.10	December 9, 2013
2.1.0	August 22, 2013
2.2.0	October 13, 2013
2.3.0	February 20, 2014
2.4.0	April 7, 2014

4.2 Benchmarks and Datasets Used

After the selection of releases, we chose the appropriate benchmarks for the performance evaluation. Hadoop comes with a large set of examples and tools that allow the benchmarking of clusters in several ways. Some of them are tools to benchmark the file system, such as DFSIO, a distributed I/O benchmarking tool. Others use MapReduce applications, such as WordCount, Sort, Terasort, and Join. These examples can be classified into I/O-bound or CPU-bound jobs. Since we are dealing with Hadoop performance and our main goals are directed to HDFS performance enhancements with the use of SSDs for storage, we focused our selection on I/O-bound benchmarks. From the default examples available, those dealing with sorting were suitable, including Sort and Terasort, since they are considered I/O-bound MapReduce applications. We also selected the WordCount benchmark to distinguish the behavior of our approach in one CPU-bound benchmark. To run the experiments, each benchmark uses a different dataset. We generated each of the datasets, storing them for later use and replication of the experiments. By doing this, we assure that every set of experiments uses the same input data, avoiding a possible bias in the results due to differences in the datasets.

Table 4.2: *Benchmarks and Dataset Sizes*

Benchmark	Dataset Size	Type
Sort	10GB 48GB 256GB	I/O-bound
WordCount	10GB 20GB 48GB	CPU-bound

For the Sort experiments, the datasets were generated using the *RandomWriter* job, which generates a predefined amount of randomized bytes. As a result of this job, we have a set of files with random keys that are used as input for the Sort jobs. The WordCount benchmark had its datasets created using generic text from web tools – also known as Lorem Ipsum generators. The text was continuously generated until the dataset files reached the desired sizes. Our final selection of benchmarks used in the experiments was composed of Sort and WordCount. The Terasort benchmark was discarded due to storage space limitation of our infrastructure. Table 4.2 presents our set of bench-

marks and their corresponding datasets size used for the experiments for the Hadoop performance investigation presented in this chapter.

4.3 Cluster Infrastructure

To perform the experiments, we assembled a commodity 9-node cluster. Each node had a quad-core processor (AMD A8-5600K; 3600 MHz), 8GB of RAM, a 1TB hard drive (Western Digital WD Black WD1002FAEX; 7200RPM; 64MB Cache; SATA 6.0Gb/s), and a 120GB solid-state drive (Intel SSDSC2BW120A4; SATA 6Gb/s; 20nm MLC). The network infrastructure used a gigabit switch. One of the nodes is designated to run exclusively the NameNode and JobTracker daemons, which also kept the history log from jobs. All the other eight nodes run exclusively DataNodes and TaskTracker daemons. Due to this configuration, our total storage space was approximately 9TB, including HDs and SSDs.

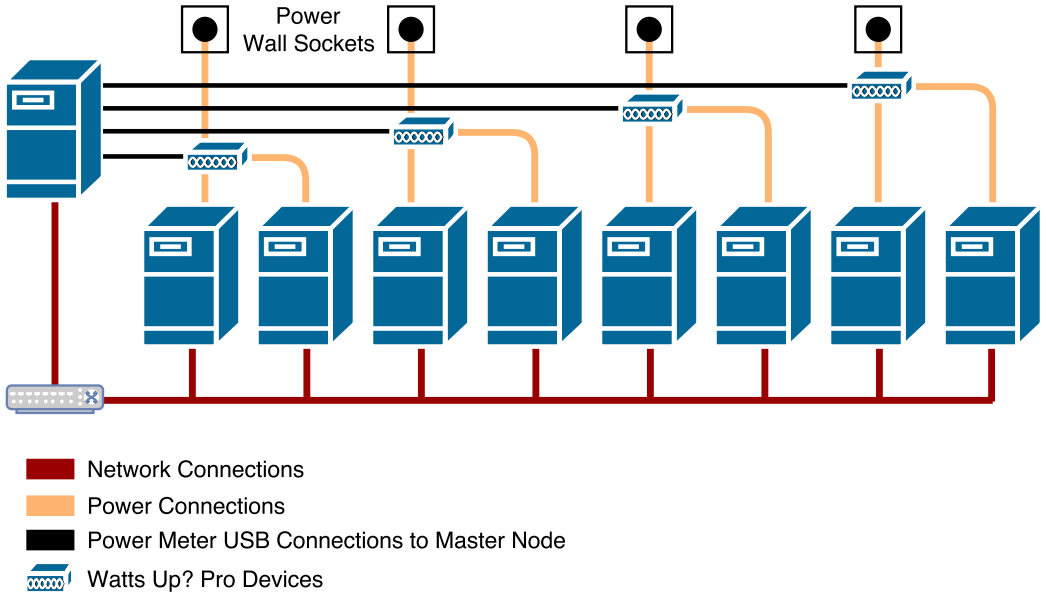


Figure 4.2: *Cluster Infrastructure*

To have appropriate energy consumption measurements, we instrumented the cluster with *Watts Up? Pro*[2], a hardware device that measures wall socket power use. It reports power measures per second containing watts, kWh, voltage, amps, power-factor, and other information. For our cluster, we have acquired four measurement devices, and connected the DataNodes in pairs on each one of the power devices. *Watts Up? Pro* reports its readings through a USB port. The USB ports of the four meters were connected to the master node of the cluster. Since our studies are directed to energy consumption on DataNodes, we decided not to perform measurements on the master node, which runs exclusively the NameNode and JobTracker daemons. This configuration is presented in Figure 4.2. Following the Hadoop premise to move computation to data, we focused on the energy consumption on the Hadoop DataNodes. Therefore, our experimentation was carried out on a cluster with energy measurement instrumentation exclusively on the DataNodes/TaskTrackers.

4.4 Experiment Design and Methodology

Our experiment methodology includes the test of several Hadoop releases using different workloads in the same testbed. For each experiment set, a Hadoop release was deployed individually in the cluster and the appropriate dataset for the experiment was sent to HDFS. Then, a set of

[2]Watts Up Meters: http://www.wattsupmeters.com/secure/products.php?pn=0 (Visited on 15/10/2016)

jobs was performed for the selected benchmark. If any of the jobs in a set of experiments did not record accordingly the power measurement reads, it was discarded. For each set of experiments, a minimum of 5 completed jobs (with complete power readings) had their data recorded. If necessary, additional jobs were deployed to complete the minimum required for our analysis. Thus, we guarantee that each pair release/benchmark had a minimum amount of data to assure equality during the analysis phase.

A set of calibration jobs was executed prior to the experiments. In this preparation phase, each of the selected Hadoop releases shown in Table 4.1 was pre-configured with default parameters, deployed, and tested individually in our cluster. These tests included a set of small jobs to verify any issues with the software and hardware infrastructure. We used 5GB-10GB datasets, performing Sort jobs with every tested release. Each Hadoop release was deployed using our default configuration – a set of configuration files to assure equality between releases during experimentation. Table 4.3 presents the main properties and the default values of the configuration files.

Table 4.3: *Default Values for Configuration Files*

Configuration File	Property Name	Value
hdfs-site.xml	`dfs.replication`	3
hdfs-site.xml	`dfs.block.size`	67108864
hdfs-site.xml	`io.sort.mb`	128
hdfs-site.xml	`io.sort.factor`	100
mapred-site.xml	`mapred.child.java.opts`	-Xmx768m
mapred-site.xml	`mapred.job.reuse.jvm.num.tasks`	-1

During a set of experiments, only the current benchmark dataset was stored in the HDFS. When changing between releases to run new experiments, there is incompatibility between different HDFS versions, which means that for every new set of experiments, a selected Hadoop release was deployed and the HDFS reformatted. Then, the dataset was sent to the file system and the experiment batch started. In this phase, experiments were designed to discover the existing differences among the tested releases. Therefore, we measured the performance and energy consumption on different storage devices using two different HDFS configuration scenarios: first, HDFS used only the HD space available in the DataNodes; second, the benchmarks ran using only the SSD space in the DataNodes.

To record the power meter readings in the master node, we adapted a python program based on the power meter manufacturer's software. Differently from the original manufacturer's C program, the Python approach demonstrated to be more efficient handling the USB/serial OS libraries, losing less readings from the meters. The Python version used to record the data can be downloaded from the BitBucket repository at http://bitbucket.org/ipolato/wattsup. The program polls the chosen meter through the USB port and receives the raw readings. The reading rate was set to once per second, and each meter logs the readings into separate files. During a job execution, we launched individual instances of the reading program for each power meter and kept them running in background until the end of the job. The power logging program was launched 5 seconds before the job started and was terminated 10 seconds after the job finished. This additional time at the end of the readings was intended to capture the final behavior of Hadoop during replication of the results in HDFS. Between each pair of successive jobs in a set of experiments, there was a wait interval of at least 90 seconds. This interval is used to bring the system and drives to their idle state and normalize the power consumption rate. Using this process, we can assure that the power readings at the beginning of each job are at the same levels. It is important to notice that, although these experiments are directed to performance studies, the energy measurements were also recorded, since the data was used during energy profiling – the process of extracting an energy consumption profile from the framework based on the experiment results. Additionally, we confirmed that the power meter and the measurement daemons do not affect cluster performance by running two sets

of experiments: one with the daemons running and another without. The job makespan results were the same during both sets of experiments.

4.5 Results and Analysis

Our experiments show large differences in performance and energy consumption among the multiple tested Hadoop releases. With the evolution in Hadoop branches, releases that included the YARN resource manager *performed worst* and *consumed more* energy when compared to the 1.x releases. In the following, we detail our results. At the top of each bar on the plots, we included a boxplot to show the distribution of the results on each experiment batch. It is clear that the results from the *SSD* configuration, in most cases, were more concise due to the hardware construction – SSDs use NAND flash memories. On the contrary, HDs are mechanical spinning disks, and we consider that their mechanical operation, which includes moving and rotating parts, causes more inconsistent times during I/O operations, which reflects on the data dispersion presented on most *HD* boxplots.

4.5.1 Job Makespan

Seeking for a hybrid approach, it was important to discover which differences are achieved in performance when using different storage devices (HDs vs SSDs) on Hadoop. SSDs are faster than HDs, which means that more read and write operations can be made per second. As an expected behavior, Hadoop jobs can improve performance by reducing the job makespan using SSDs. In this case, SSD use benefits more the I/O-intensive Hadoop jobs. CPU-intensive Hadoop jobs do not receive significant benefits since CPU-time dominates over I/O operations in the job makespan.

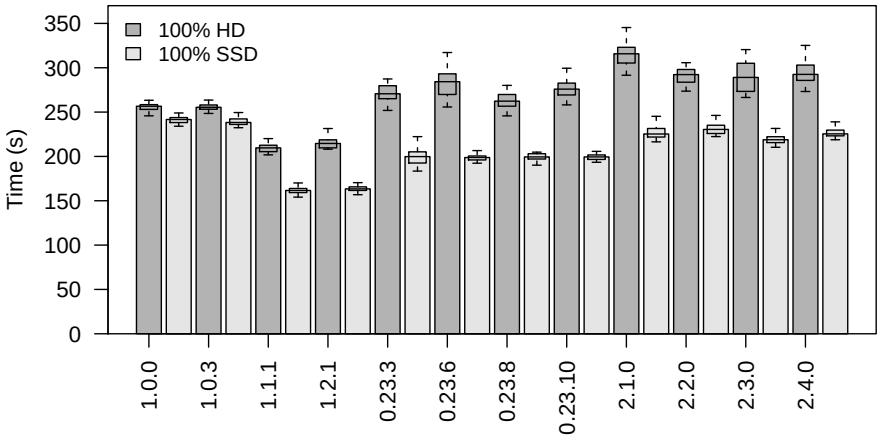


Figure 4.3: *Hadoop Sort 10GB Job Makespan*

Analyzing the results from the Sort benchmarks, we can observe the performance increases achieved when replacing HDs with SSDs in HDFS. Figure 4.3 shows the results of the 10GB Sort experiment on 12 tested Hadoop releases. We can observe that the SSD job makespan is lower than the HD job makespan for every release, as predicted. We also notice that the differences in releases 1.0.0 and 1.0.3 are smaller than for the other releases. From Figure 4.1 and Table 4.1, we can see that the latest releases from each branch tend to be the fastest ones (on average) within its branches. This means that they have more consolidated source code, receiving bug fixes and code improvement from previous releases. We also notice the small difference among the branches/releases, which is explained due to the relatively small size of the benchmark. This benchmark was particularly important for showing the performance differences between SSDs and HDs: while releases 1.0.0 and 1.0.3 reduced only 6% the job makespan using SSDs, all other tested releases reduced on average

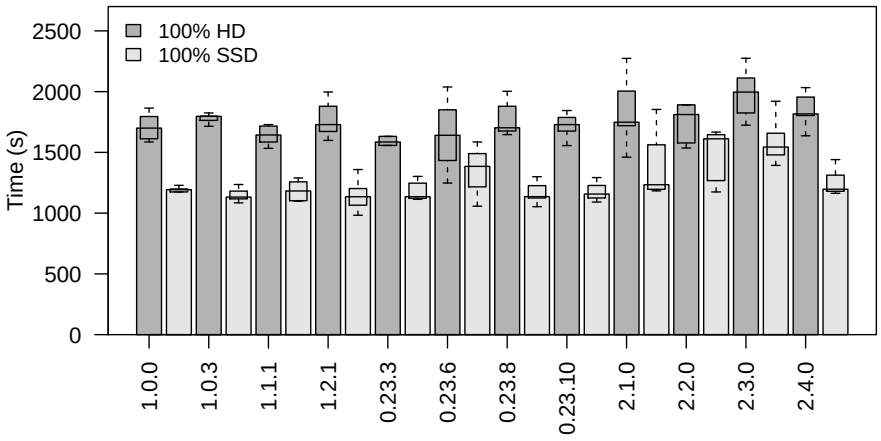


Figure 4.4: *Hadoop Sort 48GB Job Makespan*

25% the job makespan in the 10GB Sort, with release 0.23.6 achieving the biggest speedup with 30% of job makespan reduction.

Figure 4.4 presents the 48GB Sort results. This benchmark shows an increase in the differences between the HD and SSD use in Hadoop. The SSD use promoted significant performance increases in every branch and release tested. On average, SSDs proved to be 29% faster than HDs during the experiments. These results show that, as we increase the dataset size, performance enhancements rates also increase on average a little bit. We credit this to the fact that the 10GB Sort is a extremely short Hadoop job, and the instantiation times (job preparation, task resource allocation, and job cleaning) have a large weight on the job makespan. With the 48GB Sort this overhead time represents a smaller part in the job makespan. This explains the reason most releases had improved speedup rates when comparing the HD and SSD used. While a 10GB sort takes 4-5 minutes to complete, a 48GB Sort takes 29-30 minutes to run, and the 256GB (Figure 4.5) can take almost two hours to finish. Based on our data, we can also assume that the time component in a Hadoop job grows linearly following the increase in dataset size. Table 4.4 presents the job makespan results and the average speedups percentages.

Table 4.4: *Average Job Makespan Speedup Comparison (Times in s)*

Releases	10GB Sort			48GB Sort			256GB Sort		
	HD	*SSD*	Speedup	*HD*	*SSD*	Speedup	*HD*	*SSD*	Speedup
1.0.0	256	241	6%	1711	1159	32%	3937	3423	13%
1.0.3	256	239	7%	1779	1150	35%	4621	3488	25%
1.1.1	212	162	24%	1644	1184	28%	4761	3641	24%
1.2.1	216	163	24%	1763	1138	35%	4524	3517	22%
Branch Averages		14%			33%			21%	
0.23.3	271	201	26%	1713	1184	31%	7044	5969	15%
0.23.6	284	200	30%	1642	1353	18%	6229	5340	14%
0.23.8	264	200	24%	1767	1167	34%	6369	5284	17%
0.23.10	278	200	28%	1726	1175	32%	5994	5097	15%
Branch Averages		27%			29%			15%	
2.1.0	317	228	28%	1841	1406	24%	6184	5115	17%
2.2.0	292	232	21%	1840	1473	20%	6566	5852	11%
2.3.0	291	220	25%	1983	1589	20%	6695	5229	22%
2.4.0	296	227	23%	1849	1267	31%	6138	5541	10%
Branch Averages		24%			24%			15%	
Overall Averages		**22%**			**29%**			**17%**	

Analyzing the 256GB Sort experiments, we noticed a significant difference in the job makespan between releases from different branches. Although the overall speedup reduced if compared to the

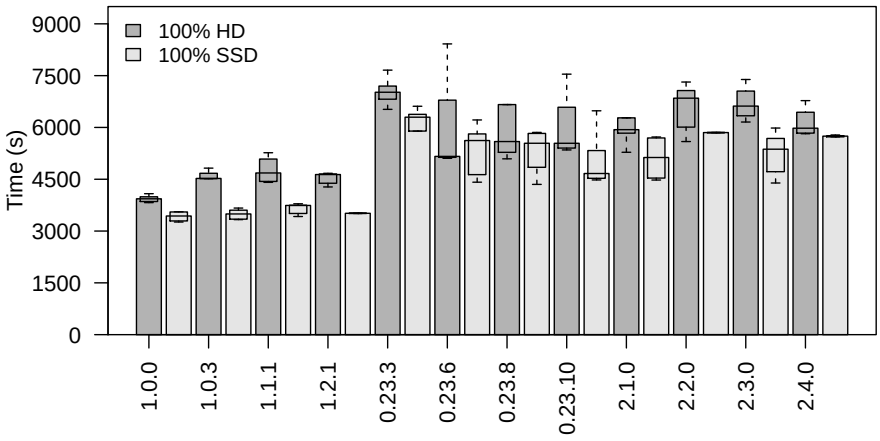
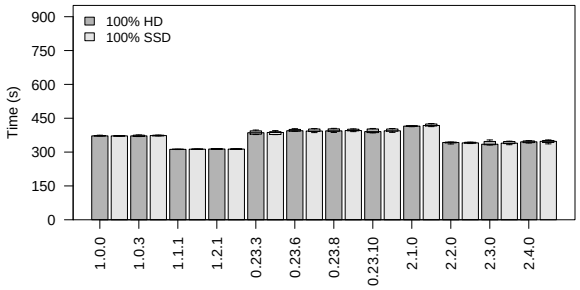


Figure 4.5: *Hadoop Sort 256GB Job Makespan*

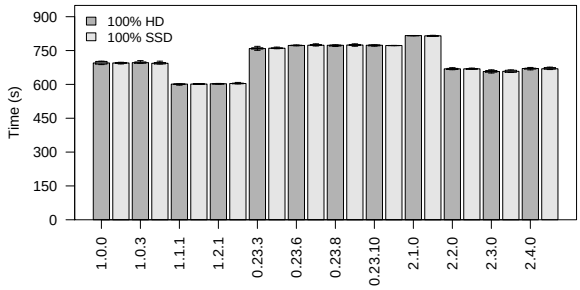
48GB Sort, releases 1.x performed faster when running experiments from SSDs than releases 0.23.x and 2.x, especially the most recent releases 1.1.1 and 1.2.1. But the most surprising result was that releases 1.x had better performances than 0.23.x and 2.x releases even when running experiments from HDs.

Finding 1: On average, 1.x releases were 30% faster than the other tested releases when running jobs with data on HDs, and 35% faster when running experiments using SSDs. This difference has a direct impact and it will be analyzed in the following sections.

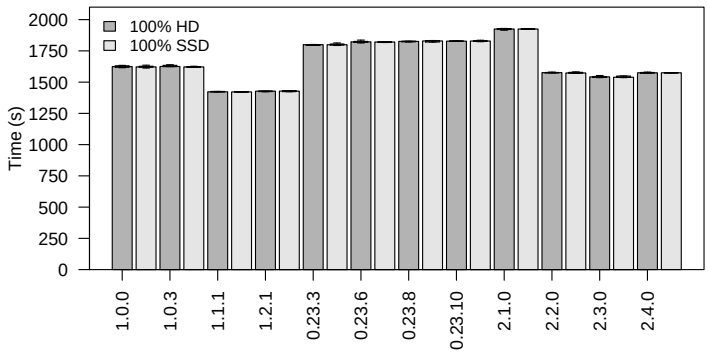
Regarding the WordCount experiments, they show that CPU-bound Hadoop jobs do not receive benefits from the SSD use, as mentioned before. However, these experiments corroborate the previous Sort results exposing the performance differences between the Hadoop development branches.



(a) *WordCount 10GB*



(b) *WordCount 20GB*



(c) *WordCount 48GB*

Figure 4.6: *Hadoop WordCount Experiments Job Makespan*

Figure 4.6 presents the achieved results. We can notice the evolution of releases, where the newest ones perform better than the older ones. The exception here is the 0.23.x branch which presented a constant behavior of its releases, with a slightly decrease in performance in the latest releases. The worst performance from Hadoop 2.1.0 can be explained since it is a beta release from the 2.x branch. Overall, in the three WordCount experiment sets, releases from branches 1.x and 2.x performed on average 10% faster than 0.23.x releases.

4.5.2 Evidence of Changes in Hadoop's Performance

Having demonstrated significant differences in Hadoop performance, we conducted an investigation to find out whether these issues were perceived or not by the community. It is a fact that Hadoop evolved rapidly between 2011 and 2012: architectural changes were introduced, bringing new components and consolidating flexibility into the framework. Behind the novelty of the 0.23.x and 2.x branch releases, there is a hidden controversial issue of software engineering: the performance versus flexibility dilemma. Is the loss in Hadoop's performance related to the architectural change necessary to implement YARN? Is it clear that the YARN resource manager has a direct negative impact on Hadoop's performance?

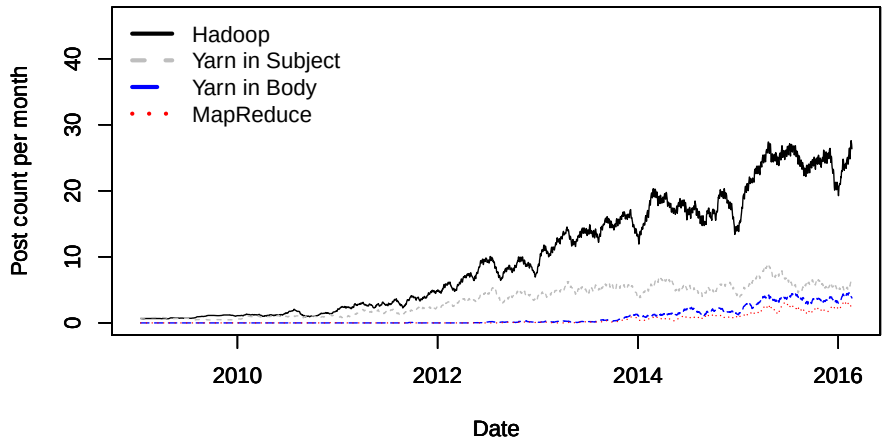


Figure 4.7: *Timeseries of Hadoop Mentions on StackOverflow*

Even though this issue can be confirmed by deploying releases from different branches, these questions remain virtually unanswered since the Hadoop community may have overlooked the impact of these modifications. The number of downloads and contributions to each branch release is evidence that there is no emphasis or concern regarding this degradation in performance. At present, browsing the source control history, it is evident that branch 2.x receives the majority of source code commits, with frequent release of new versions. The other two branches receive much less attention from Hadoop developers.

To determine if the community noticed this change in performance, we mined StackOverflow questions that asked about it. StackOverflow is a question and answers website that programmers use for general support. Sometimes, project-specific questions appear on the stackoverflow site. From StackOverflow, we retrieved the database dump[3] and parsed through all of the available questions tagged with `hadoop`, `yarn` and `mapreduce`. The mined dataset and scripts used during this analysis can be downloaded from our $HDFS_H$ GitHub repository[4].

Hadoop performance is a common topic on StackOverflow made by users when deploying a particular release. Most research regarding Hadoop performance promotes fine-tuning of configuration files, using the infrastructure characteristics to achieve the proposed objectives. Hardware, Operat-

[3]We used the "stackoverflow.com-Posts.7z" file downloaded from https://archive.org/details/stackexchange (Visited on 15/10/2016)

[4]Folder "StackingOverflow Mining" at http://github.com/ipolato/Hybrid_HDFS

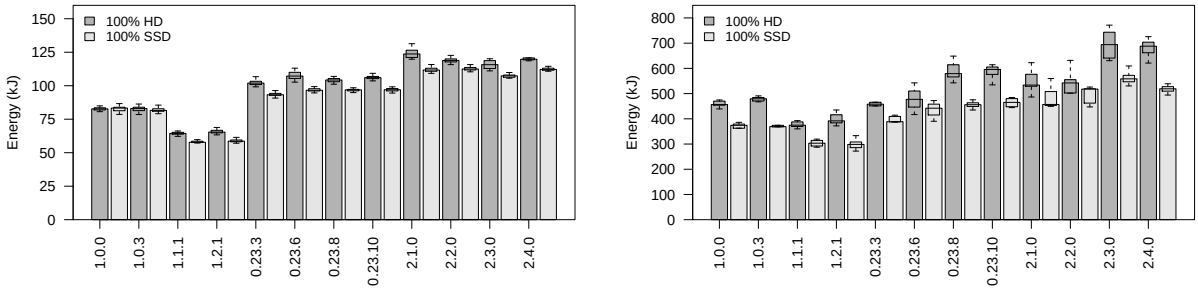
ing System, and Java characteristics are commonly addresses as the source of Hadoop performance problems, overlooking the internal behavior of framework components.

As we can observe in Figure 4.7, YARN is not an uncommon topic, especially on the subject of messages. Although the YARN performance was hinted, it was not specifically addressed by users. Questions regarding general Hadoop performance are more common, and in general, do not associate YARN performance issues with Hadoop performance. Therefore, we can conclude that most users are directed to download the most recent releases, unaware of the performance issues existing among the development branches. Furthermore, even though the 1.x branch is stable, it does not receive any more updates and source code commits, which force users to download newest versions, especially for security reasons.

Finding 2: Hadoop performance issues were identified by users, but for external reasons – Hardware, Operating Systems, Java – and not for changes in the Hadoop architecture during the evolution from 1.x to 0.23.x, and to 2.x releases. Additionally, releases 0.23.x and 2.x, which contain YARN, cannot run without such resource manager. Therefore, an unnecessary waste of resources occurs when using YARN releases to perform MapReduce jobs.

4.5.3 Energy Performance

After analyzing the Hadoop job makespan, we conducted an energy analysis of such results, seeking for the effects of the performance on the energy consumption. Starting with the 10GB Sort benchmark, Figure 4.8a shows the differences on energy consumption between each release using data stored in HDs and SSDs. As expected, the use of SSDs on Hadoop reduces the energy demand and decreases the overall job makespan. The differences among the three branches are easily identified. Releases 0.23.x.and 2.x consumed 30% more energy on average if compared to 1.x releases.



(a) *Sort 10GB* **(b)** *Sort 48GB*

Figure 4.8: *Hadoop Sort Experiments Energy Consumption*

Experiments with larger datasets also corroborate the results, as presented in Figures 4.8b and 4.9. The 48GB Sort showed that the energy consumption on 0.23.x and 2.x releases is still higher than releases 1.x, although proportionally lower when compared to the 10GB Sort.

The 256GB Sort results show the same pattern again (Figure 4.9), with releases 1.x saving considerably more when compared to the other branches. Table 4.5 presents the Sort results and comparison between individual releases and branches.

Finding 3: Overall, during the Sort experiments, we can observe that the use of 1.x releases promotes an energy consumption reduction ranging from 20% to 30% on average when comparing with the YARN releases.

The WordCount experiments showed a different view. Due to its CPU-bound nature, these experiments do not show any differences on energy consumption between the use of HDs and SSDs.

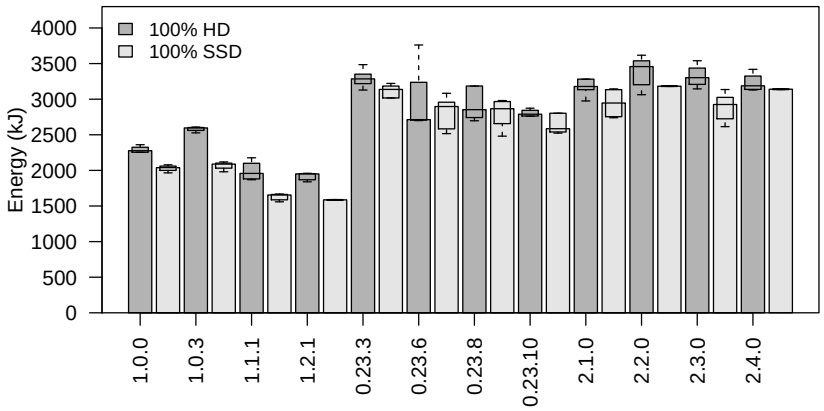
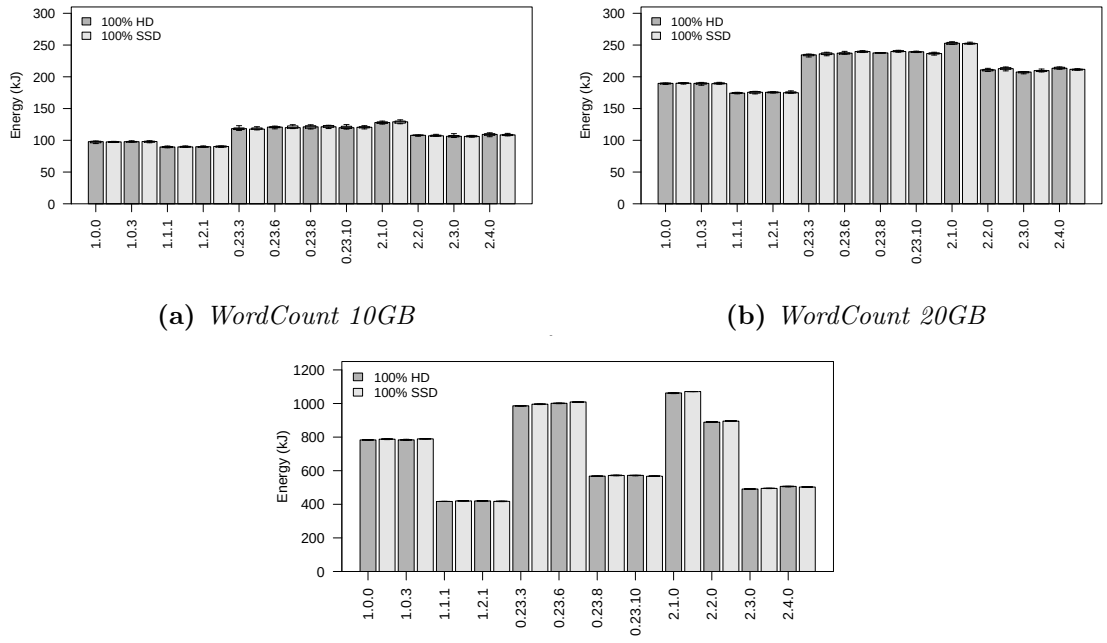


Figure 4.9: *Hadoop Sort 256GB Energy Consumption*

Figure 4.10 presents the WordCount results. The WordCount experiments with smaller datasets (10 and 20GB) present insignificant energy consumption differences among releases within the same branch. The 256GB WordCount experiments present the release evolution in the Hadoop project, where newer releases demand less power than older ones. We can clearly notice the reduction of energy consumption in every tested branch. The two initial releases from each branch performed poorly if compared to the latter releases, representing major energy consumptions during job execution. Comparing the latter releases of each branch, we can perceive a certain stability, showing no significant differences in energy consumption.

Back to the inter-branch comparison, the results show that the 1.x releases consume less energy than 0.23.x and 2.x releases, even on CPU-bound experiments. Releases 0.23.x used on average 30% more energy than 1.x releases, while 2.x releases used 21% more energy during experiments. These results follow the same pattern observed in the I/O-bound experiments with the Sort benchmark. Comparing the latest tested release from each branch, release 1.2.1 consumed on average 26% less energy than release 0.23.10 and consumed 18% less energy than Hadoop 2.4.0 on every WordCount experiment.



(a) *WordCount 10GB*

(b) *WordCount 20GB*

(c) *WordCount 48GB*

Figure 4.10: *Hadoop WordCount Experiments Energy Consumption*

Table 4.5: *Average Energy Comparison (Energy values in kJ)*

Releases	10GB Sort			48GB Sort			256GB Sort		
	HD	*SSD*	Reduction	*HD*	*SSD*	Reduction	*HD*	*SSD*	Reduction
1.0.0	83	83	0%	459	363	21%	2294	2030	12%
1.0.3	83	82	1%	479	372	22%	2577	2064	20%
1.1.1	65	58	10%	378	304	20%	1990	1626	18%
1.2.1	66	59	10%	398	298	25%	1914	1588	17%
Branch Averages	5%			22%			17%		
0.23.3	102	94	8%	477	398	17%	3294	3032	8%
0.23.6	107	97	10%	478	437	9%	3058	2807	8%
0.23.8	104	97	7%	588	456	22%	3101	2790	10%
0.23.10	106	97	9%	593	464	22%	2803	2736	2%
Branch Averages	9%			17%			7%		
2.1.0	124	112	10%	550	485	12%	3257	2944	10%
2.2.0	119	113	5%	547	495	10%	3376	3182	6%
2.3.0	116	107	7%	696	564	19%	3322	2885	13%
2.4.0	120	112	7%	682	521	24%	3231	3077	5%
Branch Averages	7%			16%			8%		

4.5.4 Hadoop Source Code Analysis

Following the energy consumption analysis, we conducted an investigation of the Hadoop source code, searching for correlation of several factors with the energy consumption. We performed three separate analyses during this process. First, an analysis of the software size versus job makespan and energy performance was made, since the software architecture has direct influence on performance, and consequently on power demands. Second, we analyzed the relation of the map and reduce tasks with power to explain the sources of energy consumption in a Hadoop job. Finally, we carried out an analysis of object-oriented metrics obtained using the CKJM-extended[5] tool to understand the framework evolution, correlating the results with software size, job makespan and energy consumption.

It is important to note that the investigation of the source of energy consumption in data centers is viable and needed, given that energy is one of their major costs and have significant environmental impact. In parallel and distributed computing, resource usage is the key to achieve a well-balanced trade-off between performance increase and energy savings. Furthermore, resource usage can be closely related to the software architecture, since the latter determines how computations will be accomplished. Therefore, changes in the software architecture during development — adding or removing features and components, modifications in classes, methods and attributes — directly affect energy consumption.

Hadoop fits exactly in this context. Over the last decade, Hadoop had more than 60 releases. From version 0.20.3, and over the course of two years, project decisions generated the three current development branches: 1.x, which is stable for now; 0.23.x and 2.x that maintains active development. Hadoop 1.x releases remained strictly correlated to MapReduce use, while 0.23.x and 2.x branches introduced critical architectural modifications, adding new components and changing the project architecture. These Hadoop releases became more flexible, allowing the use of different program paradigms, multitenancy and parallel instances of the framework running on the same infrastructure. Even though Hadoop is tightly linked to MapReduce and such architectural modifications not only caused losses in performance, but also increased greatly the energy consumption, Hadoop's energy consumption was never fully analyzed by the research community.

Size, Time, and Versions versus Energy

One confusing factor to any analysis of software and metrics is the size of the software – lines of source code – versus metrics and performance. *Lines of Code* (LOC) will correlate linearly with

[5]Extended CKJM Metrics: http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/ (Visited on 15/10/2016)

many features of the source code. For instance, if an "if" statement occurs with probability of 0.1 per line then roughly anything that correlates with LOC will correlate with "if" statements.

Looking for correlations between the source code and the energy consumption results, we subdivided **RQ2** into four questions :

RQ2.1 ***Does Hadoop size correlate with energy?***

RQ2.2 ***Does energy correlate with Hadoop execution time?***

RQ2.3 ***Does Hadoop size correlate with execution time?***

RQ2.4 ***If we order the Hadoop releases by release date, does the rank of the release (first, second, third, etc.) correlate with energy?***

To answer **RQ2.1**, we defined size as the number of Java lines of code counted by David Wheeler's SLOCCount[6] program for that version of Hadoop. We also defined a sort configuration as a Sort benchmark running on a specific storage, such as, Sort 10GB HD, Sort 10GB SSD, and so on. For all tests of all Sort benchmarks — 10, 48, 256GB, for both SSD and HD — for all tested Hadoop versions the Pearson correlation is 0.1496, a very weak to negligible correlation. The problem is that this measurement mixes benchmarks of different sizes. By isolating datasets and storage configurations, the values presented in Table 4.6 show a high Pearson correlation. Therefore, we argue that size does indeed correlate with energy, but if we control for other factors, such as version, we might find out that it is not as important.

Table 4.6: *Pearson Correlation: Hadoop Size with Sort benchmarks*

Storage	Sort 10GB	Sort 48GB	Sort 256GB
HD	0.9445	0.9656	0.9121
SSD	0.9502	0.9528	0.9146

On a single computer typically time and energy are related as energy is defined as the integration of power over time ($e = p \cdot t$). But we must consider that Hadoop tasks run on multiple computers. Thus, we analyzed a possible relation between job makespan and energy consumption in **RQ2.2**. Regardless of the sort task size or configuration, over all tests, a Pearson correlation of 0.9880 was achieved, indicating high linear correlation between time and energy, as expected. Thus, time acts as a high accuracy proxy for energy. As a result, we concluded that there is a relationship between energy and time and a strong relationship between size and energy.

As a third analysis, in **RQ2.3** we considered the Hadoop code size and the job makespan. Regardless of the sort configuration, a medium to high Pearson correlation of 0.7576 between Java LOC and execution time was achieved, indicating that code size and execution time are also related. Thus code size had a medium to high strength relationship depending on the sort configuration, with the lowest correlations belonging to the Sort 48GB experiments.

Table 4.7: *Correlation Summary: Size, Time, and Versions versus Energy*

Research Question	Correlations	Results	Correlation Value
RQ2.1	Hadoop Size (Java LOC) and Energy	High	> 0.91
RQ2.2	Energy and Job Makespan	High	≈ 0.98
RQ2.3	Hadoop Size (Java LOC) and Job Makespan	Medium to High	≈ 0.70
RQ2.4	Hadoop Version and Energy	Medium to High	≈ 0.65

If code size mattered, and indeed the code size increased over time, perhaps the release order, the version number, also matters, as pointed out in **RQ2.4**. Regardless of the sort configuration,

[6]David Wheeler's SLOCCount: http://www.dwheeler.com/sloccount/ (Visited on 15/10/2016)

a low Pearson correlation was measured. But if ordered according to the development branches — 1.0.0, 1.0.3, 1.1.1, 1.2.1, 0.23.3, 0.23.6, 0.23.8, 0.23.10, 2.1.0, 2.2.0, 2.3.0, 2.4.0 — we find a weak general correlation and a medium per sort correlation. Using this ordering, the Pearson correlation between version and size is 0.6675 in general. Thus we find that Hadoop versions can correlate heavily with energy consumption, but it depends on how the versions are ranked. Furthermore, the date of the Hadoop release is not correlated with energy consumption. This indicates that the branch of the code matters quite a bit, as we hinted from the experiments results. Table 4.7 presents the summary of our findings.

We found out that time and energy have a high correlation, while Hadoop version number, not release date, had a medium correlation with energy consumption. We also found that while size was strongly correlated with energy consumption, the version was much an equally strong predictor of energy consumption, as both are correlated with each other.

Finding 4: We argue that while LOC has a high correlation, it is the version and revisions to the code which truly matter, as shown by the results pointed out previously.

Maps & Reduces versus Power

Another important analysis is how the job tasks (maps & reduces) influence energy consumption. Although the number of map and reduce tasks can be explicitly set through configuration properties, this is not the regular behavior. The number of blocks used by the dataset in HDFS controls the number of map tasks in a Hadoop job. Since HDFS splits every file in a number of blocks, there is a direct relation between the dataset size and the number of map tasks to compute it. The block size, pre-configured in Hadoop, can also change this number: a small block size increases the number of map tasks; a large block size will generate a smaller number of map tasks. Both conditions may be undesired. With a small block size, the time to instantiate a map task will overcome the processing time, generating excessive I/O in the intermediate phases and severe performance loss. The second condition can also bring issues: a larger block will result in longer map tasks, that if unbalanced during execution, will slow the entire job down – long-tail jobs. In the same way, the number of reduce tasks also affects Hadoop: a high number of reduce tasks will decrease performance, since there will be data redistribution to accommodate the number of reduce tasks. The opposite is also true, a low number of reduce tasks will cause unnecessary data transfers between nodes finishing map tasks to gather the results to be computed. Although there is no ideal value for the number of reduce tasks, the block size and the number of files created in the output are considered. In our experiments, the number of map tasks was set by the dataset size divided by the default HDFS block size (64MB), and the number of reduce tasks was not set, being determined by the configuration files of the benchmark. The numbers are presented in Table 4.8.

Table 4.8: *Number of Map and Reduce Tasks per Benchmark*

Tasks	Datasets					
	Sort			WordCount		
Maps	160	768	4128	160	320	768
Reduces	48	48	48	1	1	1

Reduce tasks have a direct interaction with I/O operations because they are responsible for writing the results of computation to HDFS. Although Map tasks always read their splits from storage, read operations are not as costly as write operations. Both benchmarks have different interactions with the Reduce phase. The Sort benchmark uses the reduce phase to spill the ordered records to the file system. While map tasks are responsible for reading from disk and directly writing the data for the intermediate phase, reduce tasks will perform the sorting itself, pulling the records from the intermediate phase and writing the results to the HDFS. Therefore, reduce tasks in Sort jobs demand more energy than map tasks. Since this benchmark is purely I/O-bound, the energy

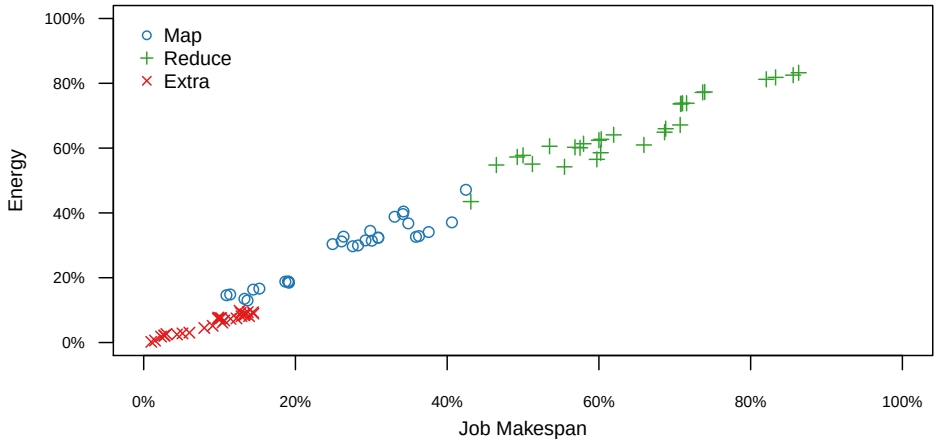


Figure 4.11: *Sort Benchmark Energy and Job Makespan Split by Phase*

consumption changes from release to release, especially among the tested branches, as presented in Figure 4.11.

Figure 4.11 depicts three job task types in terms of energy and makespan. Besides the Map and Reduce phases, the extra energy is the complementary energy spent by the framework before the beginning of the first map task (job preparation) and after the last reduce task (heavily associated with HDFS block replication). From our data, we concluded that 1.x releases use less than 20% of the total job energy for the Map phase, and less than 3% as complimentary energy; the 85% remaining is used during the Reduce phase. This changes considerably when compared with 0.23.x and 2.x releases, which uses on average 30% of the energy during the Map phase, 7% as extra energy, and approximately 62% of the energy for the Reduce phase.

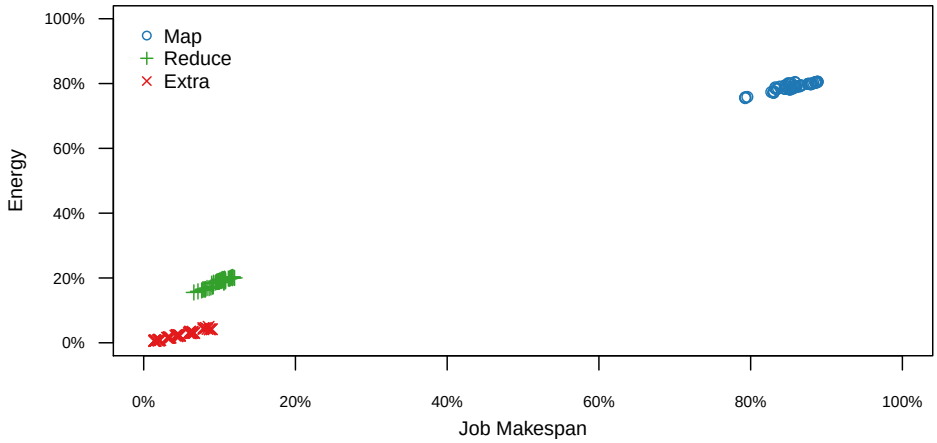


Figure 4.12: *WordCount Benchmark Energy and Job Makespan Split by Phase*

The results from the WordCount benchmark presented the large difference between CPU- and I/O-bound jobs in Hadoop. In this benchmark, the Map phase is the major responsible for the energy consumption, as a result of the word-by-word analysis from each input split and low weight of the I/O operations in the power demands of this particular benchmark. As a consequence, Figure 4.12 presents the results grouped more tightly. Even though releases use different amounts of energy during computation as previously presented, the energy consumption split among the phases is very close in every release/benchmark tested, as shown in Figure 4.12.

Hadoop CKJM-extended Metrics Analysis

As a final step in the Hadoop source code analysis we performed an analysis using the CKJM-extended suite (JS10), which measures a variety of object-oriented metrics for Java. This is a

worthwhile analysis since most of Hadoop is written in Java. These metrics were first proposed by *Chidamber and Kemerer* (CK94), and many of the CKJM metrics we investigate are the averages of the metrics over all classes.

These are some of the metrics measured as described by *Jureczko and Spinellis* (JS10):

- *Data access metric* (DAM) – a ratio of private and protected attributes to total attributes per class;
- *Afferent couplings* (Ca) – how many other classes use a specific class;
- *Number of children* (NOC) – the number of direct children of a class;
- *Weighted methods per class* (WMC) – the number of methods per class;
- *Number of public methods* (NPM);
- *Coupling between object classes* (CBO) – the number of classes coupled to a given class;
- *Lack of cohesion of methods* (LCOM and LCOM3) – counts methods that don't share attributes or fields;
- *Cohesion among methods of class* (CAM) – measures whether or not methods of a class share the same types of parameters or not; and finally;
- *Lines of Code* (Java LOC) – Lines of Code, previously calculated by SLOCCount.

For each Hadoop version, we ran the CKJM extended suite on its java jar files. The bytecode of each class was analyzed and metrics were calculated. Then, we averaged the metrics over the entire product. Once this was done, we could compare the benchmarked energy and time measurements against the extracted CKJM metrics. Each version of Hadoop had its own set of averaged CKJM metrics.

To investigate whether CKJM metrics were related to Hadoop performance, we produced thousands of linear models using data gathered with R scripts, which can be downloaded from our $HDFS_H$ GitHub repository at http://github.com/ipolato/Hybrid_HDFS. We evaluated these models, keeping only 506 of those that met a stringent criteria. This kind of multiple regression analysis is similar to ANOVA. The averaged CKJM metrics were then put into linear models of 2 to 4 independent variables (CKJM metrics) to model a dependent variable (energy use). Due to high linear correlation between many variables, models with 5 or more independent variables were not used. These models were produced per job (we consider here a job as the execution of an experiment on a dataset, e.g, Sort 10GB), as there is no hint in the source code as to the size of the job. All combinations of CKJM metrics, java LOC, and version (ranked from 1.x to 0.23.x to 2.x) were iterated over, but if they were not linearly independent (correlation of 0.75 or less) the model was not run. If a model was produced and all independent variables were not significant ($\alpha <= 0.05$) then the model was not kept.

The linear models per sort configuration were successful as the R^2 range was between 0.9035 and 0.9998 for the top 10 performing models of each configuration where all variables were significant. Table 4.9 shows the metrics present in at least 10 of the 60 top models, regardless of the configuration or dataset size. LOC, Ca, and WMC were the top 3 metrics. We expected the LOC metric to show up, as presented in subsection **Size, Time, and Versions versus Energy**. This demonstrates once more the power of size and version awareness. Other metrics such as WMN and Ca appeared in at least 50% of the models and have a considerable weight in our analysis.

Table 4.9: *Metrics in the top 60 models*

Metric	NPM	RFC	NOC	CBO	WMC	Ca	Java LOC
Count	10	11	11	13	16	30	47

Analyzing the metrics that were used to generate the models, we can observe a few behaviors that support the architectural changes' influences on the performance and energy consumption. Since we are correlating different versions of the same project, it is fair to make such comparisons as a means to explain the impact of source code changes in the energy consumption rates, which is directly related with Hadoop performance. Table 4.10 presents the raw CKJM-extended values of the main metrics that have a significant influence on the Hadoop behavior in this analysis.

Table 4.10: *CKJM-extended raw values*

Release	WMC	CBO	Ca	NPM	NOC	DAM	LCOM	LCOM3	Java LOC
1.0.0	6.883	11.610	5.699	4.278	0.276	0.367	48.579	1.016	343379
1.0.3	6.881	11.646	5.716	4.313	0.286	0.373	47.472	1.017	335042
1.1.1	6.914	11.805	5.830	4.257	0.292	0.379	51.777	1.013	351207
1.2.1	6.897	11.723	5.775	4.210	0.291	0.387	51.349	1.017	373825
0.23.3	10.045	13.798	6.485	7.195	0.306	0.363	234.458	1.074	532677
0.23.6	10.256	13.641	6.375	7.410	0.308	0.363	241.951	1.085	570643
0.23.8	10.244	13.629	6.370	7.397	0.308	0.364	240.399	1.082	514615
0.23.10	10.232	13.628	6.370	7.386	0.307	0.364	240.030	1.082	528641
2.1.0	11.927	14.520	6.321	8.720	0.259	0.367	260.697	1.119	668331
2.2.0	11.881	14.437	6.289	8.701	0.261	0.371	257.108	1.115	678536
2.3.0	11.981	14.491	6.290	8.752	0.257	0.371	258.995	1.108	738869
2.4.0	12.439	14.720	6.307	9.112	0.247	0.372	266.970	1.113	788555

Our data shows the direct influence of the WMC, CBO, LCOM, LCOM3, and Java LOC on the models. WMC increased greatly from 1.x releases to 2.x and 0.23.x releases. This is a direct consequence of the Java LOC metric, which nearly doubled if comparing YARN with 1.x releases. Additionally, the increase in source code size brought up another consequence presented by the LCOM and LCOM3 metrics, meaning that more classes have isolated methods that do not share common spaces, representing more instances running in parallel. This has also an influence on the CBO (Coupling between object classes) metric, present in almost 25% of the models. This indicates that there is a relationship between the general structure of the code and its final performance in terms of energy, with an influence of the version.

Finding 5: Version awareness plays a key factor in the Hadoop project, as a consequence of the architectural modifications designed to accommodate the YARN resource manager.

4.6 Final Considerations

In this Chapter, we presented our investigation of the Apache Hadoop project, its branches and releases, regarding energy consumption and performance under two storage scenarios. Hadoop is a powerful tool to process large datasets using mainly the MapReduce programming paradigm. The framework evolved through more than 60 releases and several development branches. We analyzed 4 releases of each of the 3 current branches: 1.x, 0.23.x, and 2.x. We successfully demonstrated throughout the experimentation that there is a significant difference in energy consumption between the framework development branches, mostly related to the modifications introduced during the architectural changes in the framework development.

The 1.x releases had better performance and better energy consumption than all the other tested releases. Releases 1.1.1 and 1.2.1, which did not incorporate the new resource manager of the framework, YARN, demonstrated to be the most energy efficient in the experiments. We acknowledge that the evolution of the framework brought desirable characteristics to the platform, such as the use of other programming paradigms and multitenancy. But we argue that the same evolution, through the development of the YARN resource manager, included in 0.23.x and 2.x releases, caused significant loss in performance, and consequently, increases in job energy consumption, as shown in Section 4.5.4.

In the experiments and analysis conducted, we established a strong correlation between version and release with energy consumption in Hadoop. Regarding storage devices, we tested the use of

HDs and SSDs on HDFS. As expected, the performance of jobs using SSDs overcame the HDs performance on I/O-bound jobs. On CPU-bound jobs there was no significant difference between experiments. While SSDs have an overall better performance and are more energy efficient, their use on HDFS can be prohibitive due to its cost. SSDs cost around $1.00 per GB (Hac14), while HDs' cost between $0.03 and $0.05 per GB (Mac14). At this cost per GB, SSDs price represent around 20 times or more the cost per GB of HDs. From this perspective, their use should match the expected performance increases and energy savings, and certainly used in a hybrid environment, mixed with HDs.

Chapter 5

$HDFS_H$: a Hybrid File System

In this chapter, we present our hybrid storage model for the Hadoop Distributed File System (HDFS), called $HDFS_H$, which seamlessly integrates both storage technologies – HDs and SSDs – to create a highly-efficient hybrid storage system. Our results indicate that, in most configurations, this approach promotes overall job performance increases, while decreasing energy consumption. Additionally, our hybrid storage model splits the file system into storage zones, wherein a block placement strategy directs file blocks to zones according to predefined rules. This enables the use of different storage configurations for different workloads, thereby achieving the desired tradeoff between performance and energy consumption. Our goal is to allow the user to determine the best configuration for the available infrastructure, by setting how much of each type of storage device should be used during MapReduce computations. The observations made during the experiments may also be used as a guide for users seeking to modify existing Hadoop clusters, or even put together a new cluster.

We consider that there is an increasing movement for the use of SSDs as a replacement for HDs. However, this is not likely to happen in the near future, since the capacities are still small when compared to HDs – the majority of SSDs are still around 400GB, while HDs are in the 2 to 4TB range. Additionally, cost must also be considered, although it has decreased greatly in the past few years. We expect to see a transition period in which both technologies will coexist. The more natural movement is the slow replacement of HDs and the addition of new SSDs to the existing storage servers, creating a hybrid storage environment. This is already a reality, since today it is possible to find storage providers offering categorized storage services at different price rates for HD and SSD storage space.

5.1 $HDFS_H$ Storage Model

To implement our model, we developed a hybrid storage approach for HDFS that leverages the different characteristics of HDs and SSDs connected to a Hadoop cluster. The key feature is the controlled use of SSDs to increase performance and reduce energy consumption. Yet, although these two characteristics of SSDs are outstanding, we must also consider their cost. In this context, our goal is to allow the user to determine the best configuration according to the available infrastructure, by setting how much of each storage device should be used during MapReduce computations. To describe our hybrid storage approach, the total HDFS space available must be expressed as the sum of available space in each device of the cluster DataNodes. We defined a formal model that guided the development of our approach. Most of the generic functions defined in this section are existing functions from HDFS. We formally defined them to promote a better visualization of our approach. In this model, we limited the storage devices in the cluster nodes to HDs and SSDs. Table 5.1 contains a glossary of symbols and functions used in our storage model.

First, we define the HDFS storage space as the sum of the available space on all DataNodes in the cluster. We wrote a generic function $SpaceAvailableInNode()$, which returns the free space that can be used by HDFS on a DataNode. Thus, the HDFS storage space is defined as:

Table 5.1: *Definitions Used on the HDFS Hybrid Storage Model*

Symbol	Definition
HD	Hard Drive
SSD	Solid-State Drive
DN	DataNode
d	Number of DataNodes in the cluster
DS	Dataset composed of multiple files
blockSize	HDFS default block size configured in `dfs.block.size`

$$HDFS = \sum_{i=1}^{d} SpaceAvailableInNode(DN_i) \tag{5.1}$$

Since we are modeling a hybrid environment, each DataNode may have different devices connected to it (HDs or SSDs), each with different amounts of available space. Thus, we define two storage space zones: *HDzone* is the sum of all HD space available for HDFS on DataNodes; and *SSDzone* is the counterpart for SSDs. From each DataNode, we can obtain the available space for each device category using the following equations.

$$HDspace(DN) = \sum_{i=1}^{z} SpaceAvailableInDevice(HD_i) \tag{5.2}$$

$$SSDspace(DN) = \sum_{i=1}^{w} SpaceAvailableInDevice(SSD_i) \tag{5.3}$$

where z and w are, respectively, the number of configured HDs and SSDs on DataNode DN. The total HD space will compose the *HDzone*, and similarly, the *SSDzone* will be composed of the sum of the SSD space available on each DataNode. We define them as:

$$HDzone = \sum_{i=1}^{d} HDspace(DN_i) \tag{5.4}$$

$$SSDzone = \sum_{i=1}^{d} SSDspace(DN_i) \tag{5.5}$$

where d is the number of DataNodes running in the cluster. Finally, we define the hybrid HDFS storage space for our model as the following:

$$HDFS_H = HDzone + SSDzone \tag{5.6}$$

This model captures the existing nuances between the different storage devices in the same file system. The NameNode middleware must be aware of the storage zones and the difference between the devices on each DataNode. Originally, the `dfs.data.dir` Hadoop configuration variable contains a comma separated list of the directories that HDFS can use. We extended this property to hold two lists, one for the SSD directories and the other for the HD directories. By using both lists to create HDFS storage space, the NameNode maintains a general view of the HDFS storage; it can also separately access the devices defined by the block placement policies.

5.1.1 Block Placement Policy

Following our HDFS hybrid storage model, we developed a block placement policy for $HDFS_H$. From version 1.x on, HDFS allows users to create their own pluggable block placement policies. For our purposes, a Block Placement Policy is an algorithm that specifies where a file's blocks will be stored on HDFS. These policies are controlled by the NameNode daemon, which manages the table of files, blocks, and locations.

The default HDFS block placement policy considers two main aspects: the rack map and available DataNodes. In the default scenario (3 replicas per file block), the default strategy is to randomly pick an available node and send the first copy of the block. This replica is then copied to a second randomly picked node. If possible, it will be placed on a different rack from the one that holds the first node. Finally, the third replica is copied from the second one and placed in the same rack, but on a different node. This HDFS block replication is explained by the developers as the Hadoop fault tolerance mechanism, which may avoid missing blocks of a file in case of shutdown of a node or an entire rack. Although this policy performs reasonably well, it does not consider, at any time, the existing differences between devices used to compose the available storage space on HDFS. If we connect an HD and an SSD into a DataNode of a Hadoop cluster and configure them in the *hdfs-site.xml* file, they will receive blocks to store independently of their characteristics. The configuration file *hdfs-site.xml* keeps configuration properties used by the NameNode to manage the file system, including the devices used. The `dfs.data.dir` property keeps a comma separated list of data directories used as storage locations. This configuration is made on each node of the cluster, and as a result, different DataNodes may use different devices and directories, and have different available space for the file system. The NameNode calculates the total available space as the sum of the free space on each DataNode, following the configurations for the devices/directories in the *hdfs-site.xml*.

The development of such a hybrid storage environment for Hadoop added more flexibility to HDFS. Since SSDs are faster and consume less power than HDs, given their installation on storage servers, we expect resulting performance gains and a decrease in energy consumption. In fact, this was presented in Chapter 4, where we presented the isolated benefits of the SSD use on Hadoop clusters. We seek now to present the benefits of a hybrid approach, mixing HDs and SSDs in HDFS. Our block placement policy sends a pre-configured percentage of the blocks to one of the designed storage zones, and the remainder of the blocks to the other. Let DS be a dataset that will be stored in $HDFS_H$. To know the number of blocks of DS, we must know the number of blocks of each file in DS. Modeling a generic function called *blocks* that returns the number of blocks used by a file according to the default pre-configured HDFS block size, we have:

$$blocks(file) = \lceil size(file)/blockSize \rceil \tag{5.7}$$

where the generic function *size* returns a given file's number of bytes. To obtain the total number of blocks needed to store DS into $HDFS_H$, we sum the individual number of blocks occupied by each file in DS:

$$blocksDS = \sum_{f=1}^{k} blocks(file_f) \tag{5.8}$$

where k is the number of files in DS.

Since we know in advance how many blocks DS will require in the file system and our policy controls the percentage of blocks sent to each storage zone, we express the input of DS into $HDFS_H$ as:

$$HDFS_H \leftarrow DS \begin{cases} SSDzone \leftarrow \lceil \rho.blocksDS \rceil \\ \\ HDzone \leftarrow \lfloor (1-\rho).blocksDS \rfloor \end{cases} \tag{5.9}$$

where ρ is the coefficient $(0 <= \rho <= 1)$ that determines how many blocks will be sent to the *SSDzone*. The complement of ρ determines the amount of blocks sent to the *HDzone*.

Our algorithm works by first knowing the number of blocks of a given dataset or file, which is calculated by HDFS. To store the blocks in one of the storage zones, we use one of the configured storage proportions. Consider for instance, that a user wants to store 25% of the blocks in the *SSDzone* and the 75% remaining blocks in the *HDzone*. The user configures this proportion in the HDFS configuration file and directs the blocks to the storage zones. This is accomplished in a round robin fashion, meaning that in this case, the first block will be stored in the *SSDzone*, and the following 3 blocks will be stored in the *HDzone*. When a block is stored in a specific zone, Hadoop is responsible for choosing the node in the cluster that receives the block and its replicas. Hadoop is in charge of this mainly because of the load balancing algorithms, which spreads the file blocks evenly across the whole cluster. Users are free to configure the storage proportion according to their needs. If for any reason, one of the storage zones is out of space, the blocks will be directed to the available space of the other zone. Therefore, to receive the benefits presented in our studies, users must consider the available space in each storage zone and configure Hadoop accordingly. All the source code and the releases tested using $HDFS_H$ can be downloaded from our repository at GitHub (http://github.com/ipolato/Hybrid_HDFS) under the folder "HDFS_H Hadoop Releases".

5.1.2 Storage Cost Model

Our storage model captures the essence of the proposed hybrid environment, mixing HDs and SSDs and thus supporting two important nuances on data processing: performance and energy consumption. A third concern nowadays is the cost of storage space. We now present our cost model, which allows the calculation of the storage costs based on the price/GB of each storage device using a given storage proportion. Our storage model splits the dataset blocks in HDFS into different storage zones using HD and SSD devices. Given that all files in $HDFS_H$ are stored either in the *HDzone* or in the *SSDzone*, the block proportion for each zone is complementary. Thus, considering a Hadoop job we have:

$$SSD_{prop} + HD_{prop} = 1 \tag{5.10}$$

where SSD_{prop} and HD_{prop} are the storage proportions defined for each zone by $HDFS_H$ policy.

Therefore, let DS be a dataset in $HDFS_H$. We can model the storage cost for a job as:

$$HDFS_H\, StorageCost = size(DS) \times (Cost_{SSD} + Cost_{HD}) \tag{5.11}$$

where:

$$\begin{aligned} Cost_{SSD} &= SSD_{prop} \times SSD_{cost/GB} \\ Cost_{HD} &= HD_{prop} \times HD_{cost/GB} \end{aligned} \tag{5.12}$$

using Equation 5.10 we have:

$$Cost_{HD} = (1 - SSD_{prop}) \times HD_{cost/GB} \tag{5.13}$$

Thus, we can estimate the storage cost for a Hadoop job using $HDFS_H$ by setting the SSD proportion, and the SSD and HD cost per GB. This model is useful when calculating the storage cost over time versus the energy cost and is applied and discussed during our analysis.

5.2 Experimental Methodology and Datasets

The studies from the previous chapter presented the differences in performance and energy consumption when using HDs and SSDs on Hadoop. We also presented the performance differences among the 3 Hadoop development branches. We now focus on the energy consumption and performance of the Hadoop releases using our hybrid approach. From the previous release set, we narrowed the studies to include two of the latest releases from each branch, since they performed better, as previously stated. We tested our hybrid approach on the Hadoop releases presented in Table 5.2. With this set we still can overview almost two years of the project's releases, including different versions and branches, representing different Hadoop middleware designs. Since Hadoop is constantly evolving and releasing new versions, we established a timeframe for this research, which was limited by the latest releases of April 2014.

Table 5.2: *Releases Used in the Experiments*

Hadoop Release	Date
1.1.1	2012-11-18
1.2.1	2013-07-15
0.23.8	2013-06-05
0.23.10	2013-12-09
2.3.0	2014-02-20
2.4.0	2014-04-07

With $HDFS_H$ and the block placement rules properly designed and the releases selected, we defined the set of benchmarks to test our approach. We used the same Sort I/O-bound benchmark, and included two extra benchmarks: Join and Mahout K-Means, provided in the HiBench (HHD+10) set, which is publicly available on GitHub. HiBench is a set of Hadoop tools and benchmarks, including machine learning and data analytics applications. We used an implementation of the K-Means clustering algorithm using the Mahout Library (Apa14) from HiBench in our experiments. To analyze our hybrid storage performance in different situations, our final selection of benchmarks is listed in Table 5.3, including the corresponding dataset size used in each experiment set.

Table 5.3: *Benchmarks and Dataset Sizes Used in the Experiments*

Benchmark	Dataset	Type
Sort	10GB 48GB 256GB	I/O-bound
Join	20GB	CPU-bound
K-Means Clustering	3×10^7 samples	CPU + I/O

Regarding the datasets, the Sort experiments used the previously generated datasets for the HD and SSD experiments. They were created using the *RandomWriter* job, which generates a predefined amount of random bytes. As a result of this job, a set of files with random keys serve as input for the Sort jobs. The Join benchmark performs a join between two datasets, in a database fashion. These experiments used datasets generated with DBGEN from the TPC-H benchmark (Cou02), which is widely used by the database community. Finally, we used an implementation of the K-Means clustering algorithm using the Mahout Library (Apa14) from HiBench in our experiments. K-means considers an euclidean space and attempts to group the existing elements into a predefined number of clusters. This benchmark is CPU-bound during the iteration phase and I/O-bound during the clustering phase. We chose our experimentation benchmarks based not only

on their I/O or CPU characteristics, but also on prior research analysis (HHD+10; PYXH14). Note that experiments using the HiBench benchmark set, have their datasets generated automatically. From the first run of these benchmarks, we cloned the dataset, saving it for reproducibility purposes. The other benchmarks had their datasets previously generated using external tools/jobs and saved accordingly.

To conduct these experiments we used the same methodology presented in Section 4.4: for each experiment set, a Hadoop release was deployed individually in the cluster and the appropriate dataset for the experiment was sent to HDFS. Then, a set of jobs was performed for the selected benchmark. If any of the jobs in a set of experiments did not record accordingly the power measurement reads, it was discarded. For each set of experiments, a minimum of 5 complete jobs (with complete power readings) had its data recorded. If necessary, additional jobs were deployed to complete the minimum required for our analysis. The infrastructure used in this experiment was the same one presented in Section 4.3.

Since our hybrid approach is quite flexible, we predefined a set of storage proportions to perform the experiments. Experiments using solely HDs and SSDs were performed once again with all the datasets, even though we already had the results from previous experiments. First, we kept all the data in the *HDzone*; next all the data was put in the *SSDzone*. We performed these experiments again to check whether our block placement policy could cause any effects on performance and/or energy consumption. The results showed that there is no difference between using the hybrid policy or configuring Hadoop manually to use only one storage type (HDs or SSDs). Concerning the hybrid part, our block placement policy was tested against three configurations, mixing HD and SSD storage space: one using 50% of HD and 50% of SSD space; another one using 80% of HD and 20% of SSD; the last one using 80% of the data on SSDs and the remaining 20% on HDs. Table 5.4 presents the storage proportions tested during the experiments.

Table 5.4: *Configurations Used in the Experiments*

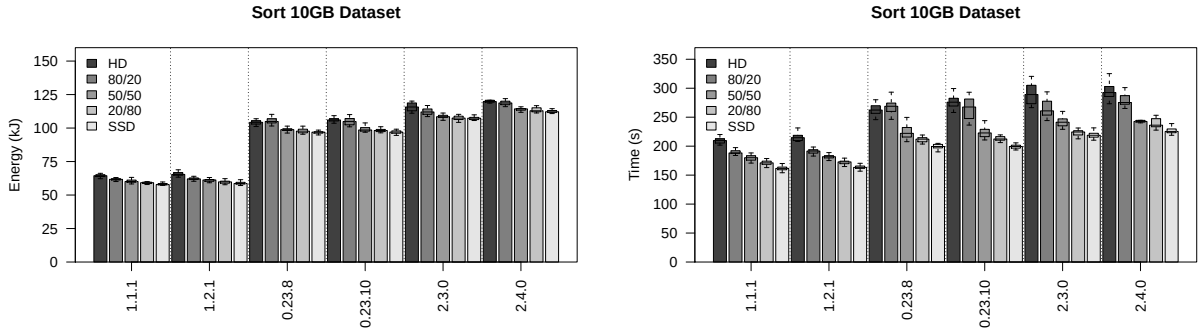
Data Percentage Configurations		
Configuration Short Name	*HDzone*	*SSDzone*
HD	100%	-
80/20	80%	20%
50/50	50%	50%
20/80	20%	80%
SSD	-	100%

5.3 Energy Consumption and Performance Analysis

The focus of $HDFS_H$ is to both increase performance and decrease energy consumption when using the hybrid environment. Regarding performance, we discovered a large difference among the several Hadoop releases. The performance issues were not related to the storage approach, but to architectural modifications introduced during the framework branching and evolution. Due to architectural changes in the middleware, Hadoop releases that included the YARN resource manager performed worst and consumed more energy when compared with the 1.x releases. Additionally, we proved that I/O-bound jobs benefit from the use of SSDs by producing lower results in job makespan for all the releases tested in the experiments. Consequently, we also demonstrated that energy consumption rates have decreased, firstly because SSDs are faster than HDs, and in this case, less energy is demanded; and secondly because SSDs demand less energy to work due to its conception. We now detail the results of our experiments using the hybrid approach.

5.3.1 I/O-Bound Benchmark Results

Starting with the 10GB Sort, the results are consistent with the previously observed behavior using only HDs and SSDs. Again, we can easily observe the differences among the three branches. Figure 5.1a shows the energy consumption results from all five storage configurations tested in the following order: *HD*, 80/20, 50/50, 20/80, and *SSD*. Whereas releases 0.23.x consumed on average 65% more power than releases 1.x, releases 2.x consumed on average 85% more power than the 1.x releases. The increase is partially explained by the performance loss: jobs running on 0.23.x releases were 27% slower than on 1.x releases. The same is observed in the 2.x branch, which was around 35% slower than 1.x releases, as Figure 5.1b illustrates. The significant difference in energy consumption can also be explained by the incremental addition of new features in recent Hadoop branches, as discussed previously. The YARN component brought flexibility to the framework, allowing other types of jobs to be executed in Hadoop, in addition to the original MapReduce. YARN also enabled the instantiation of multiple JobTrackers and NameNodes. Our experiments demonstrated that all this flexibility came at a price: loss in performance and, consequently, an increase in energy consumption when executing MapReduce jobs. From Figure 5.1, we can also observe that there are small differences between releases from the same branch, with small energy consumption variations.



(a) *Energy Consumption* **(b)** *Job Makespan*

Figure 5.1: *$HDFS_H$ 10GB Hadoop Sort Results*

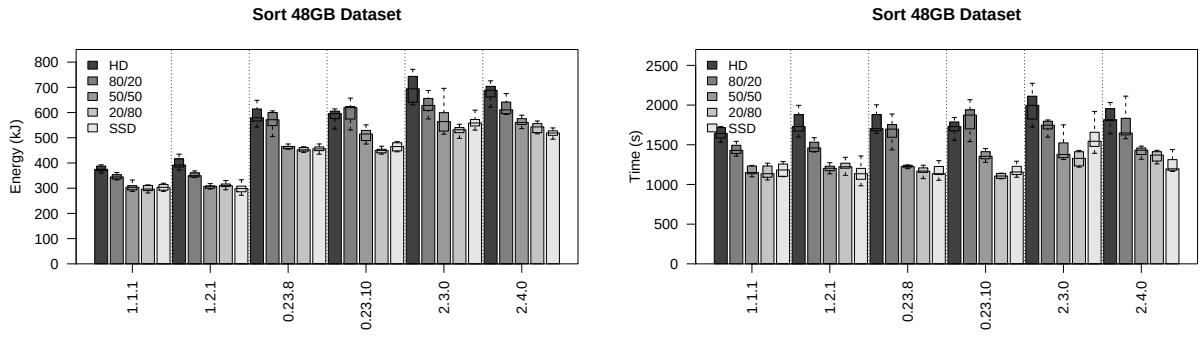
Further considering the 10GB Sort experiments, we can also observe the expected tendency in energy savings when moving data to the configurations that favor SSD use. We can also notice the non-linear response in both energy consumption and job makespan when inserting more SSDs storage space in the computation.

Finding 6: The middle 50/50 and 20/80 configurations tend to achieve results that are closer to the *SSD* configuration. This means that with half storage space coming from SSDs, we can achieve energy consumption rates closer to the use of an SSD-only HDFS.

Besides the decrease on energy consumption, this generates an inevitable and positive effect on the total storage space cost with the reduction of the average price/GB. In such case, an SSD-only HDFS would perform nearly the same as these two configurations, but would cost almost twice.

Next, we moved on to the experiments with larger datasets. Figure 5.2a shows the results for the 48GB Sort experiments. The results support the tendency toward energy savings when using SSDs. Analyzing these two initial experiments, we noticed that increasing the dataset size shifts the tendency of power saving toward the middle configurations: 50/50 and 20/80. This indicates that, by storing only a fraction of the data on SSDs with these specific hybrid configurations, we achieve a significant increase in performance and, consequently, a reduction in energy consumption. Our results indicates that, if all data is processed from the *SSDzone*, there is an average reduction of 20% in energy consumption. A similar reduction can also be observed in the 50/50 and 20/80 configurations in Figures 5.1 and 5.2, thus reinforcing the above finding.

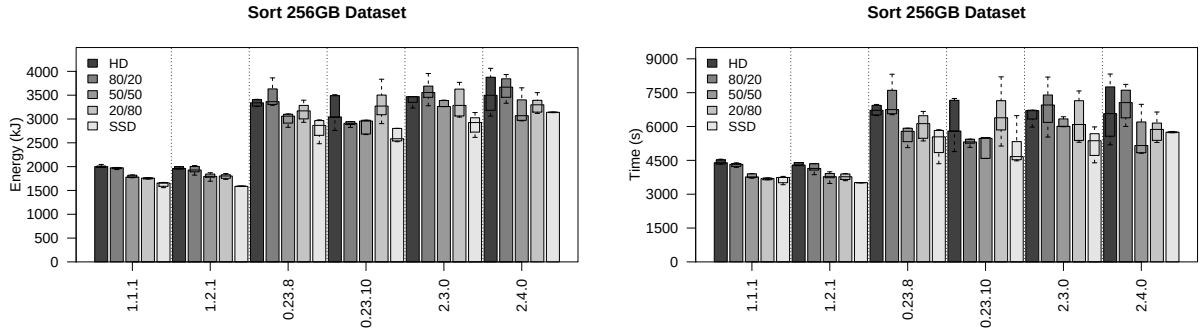
The results from the Sort benchmark using the 256GB dataset also corroborate the previous statements. With the increase in the dataset size, the energy consumption rates decrease in the



(a) *Energy Consumption* **(b)** *Job Makespan*

Figure 5.2: *$HDFS_H$ 48GB Hadoop Sort Results*

middle configurations, favoring the use of less SSD storage to achieve similar results to the *SSD* configuration. Figure 5.3 presents both the energy consumption and job makespan results from the experiments. The larger dataset experiment surfaced a new observation: most of the experiments using the 20/80 configurations performed worst or close to the *HD* configuration. This behavior was observed on YARN releases 0.23.8, 0.23.10, and 2.3.0 and is not present in releases 1.x. and in Hadoop 2.4.0. Even so, all tested releases had the same behavior on the 50/50 configuration.



(a) *Energy Consumption* **(b)** *Job Makespan*

Figure 5.3: *$HDFS_H$ 256GB Hadoop Sort Results*

This condition is attributed to the total SSD storage space. Due to this size limitation, the 20/80 Sort jobs move intermediary blocks to the *HD* portion of the configuration, degrading its performance. Analyzing the source code commits, we observed a few sets of commits regarding HDFS performance during the transition from Hadoop 2.3.0 to Hadoop 2.4.0. This is clearly observed in the results, analyzing the YARN branches.

Additionally, during the 256GB Sort experiments, we observed an issue regarding the *SSDzone* total space. Our DataNode infrastructure has eight 120GB SSDs, totaling 960GB of usable storage space. Using a replication factor of 3, a dataset triples its size in HDFS. Thus, in this setting the 256GB dataset uses 768GB, leaving less than 200GB of free space in the *SSDzone*. This does not allow the Sort Job execution, since it requires the equivalent of three times the dataset size of free space (at least 768GB) during its execution.

Therefore, the results presented in Figure 5.3 regarding the *SSD* configuration were achieved with the replication factor set to 1 on the HDFS configurations. These results are marked in Table 5.5, which presents all the energy consumption results from the Sort benchmarks and the energy consumption reduction promoted by each configuration when compared to the *HD* results. The impact of the replication factor was analyzed in further experiments presented at the end of this section.

To promote another comparison between different Sort benchmarks, we normalized the energy results using the averaged values of the observations for each tested configuration. For each pair

Table 5.5: *Sort Benchmarks: Average Energy Consumed (kJ)*

Dataset size	Release	HD	80/20	Reduction	50/50	Reduction	20/80	Reduction	SSD	Reduction
10GB	1.1.1	65	62	5%	60	8%	59	9%	58	11%
	1.2.1	66	62	6%	61	8%	60	9%	59	11%
	0.23.8	108	105	3%	99	8%	98	9%	97	10%
	0.23.10	106	105	1%	99	7%	98	8%	97	8%
	2.3.0	116	112	3%	109	6%	108	7%	107	8%
	2.4.0	120	119	1%	114	5%	113	6%	112	7%
48GB	1.1.1	378	348	8%	303	20%	299	21%	304	20%
	1.2.1	398	352	12%	306	23%	311	22%	298	25%
	0.23.8	588	566	4%	463	21%	454	23%	456	22%
	0.23.10	593	596	-1%	510	14%	449	24%	464	22%
	2.3.0	696	630	9%	579	17%	529	24%	564	19%
	2.4.0	682	631	7%	563	17%	540	21%	521	24%
256GB	1.1.1	2005	1967	2%	1795	10%	1750	13%	1626*	19%
	1.2.1	1972	1934	2%	1796	9%	1795	9%	1588*	19%
	0.23.8	3339	3469	-4%	3001	10%	3157	5%	2790*	16%
	0.23.10	3168	2971	6%	3000	5%	3320	-5%	2736*	14%
	2.3.0	3461	3587	-4%	3248	6%	3345	3%	2885*	17%
	2.4.0	3530	3650	-3%	3212	9%	3301	6%	3077*	13%

All reductions calculated based on the *HD* values. Negative values represent loss in performance.
* HDFS Replication Factor = 1

release–dataset, we divided each value by the maximum value of the group. In general, this value is associated with the *HD* configuration, as it demands more energy when running the experiments. This allowed the side-by-side comparison of the results from different experiments, since the results are in different scales. Figure 5.4 shows this panorama. We put together the experiments from the three Sort benchmarks (10GB, 48GB, and 256GB) in both versions from each release. This side-by-side comparison allows a better understanding of the $HDFS_H$ configurations benefits across datasets/releases. The 48GB Sort had the largest energy consumption reduction rates during the experiments. Overall, the 50/50, 20/80, and *SSD* configurations had similar results and strongly support our approach. Although the other two benchmarks also promote reductions on energy consumption, smaller rates were observed. In general, the 50/50 configuration had the closest results

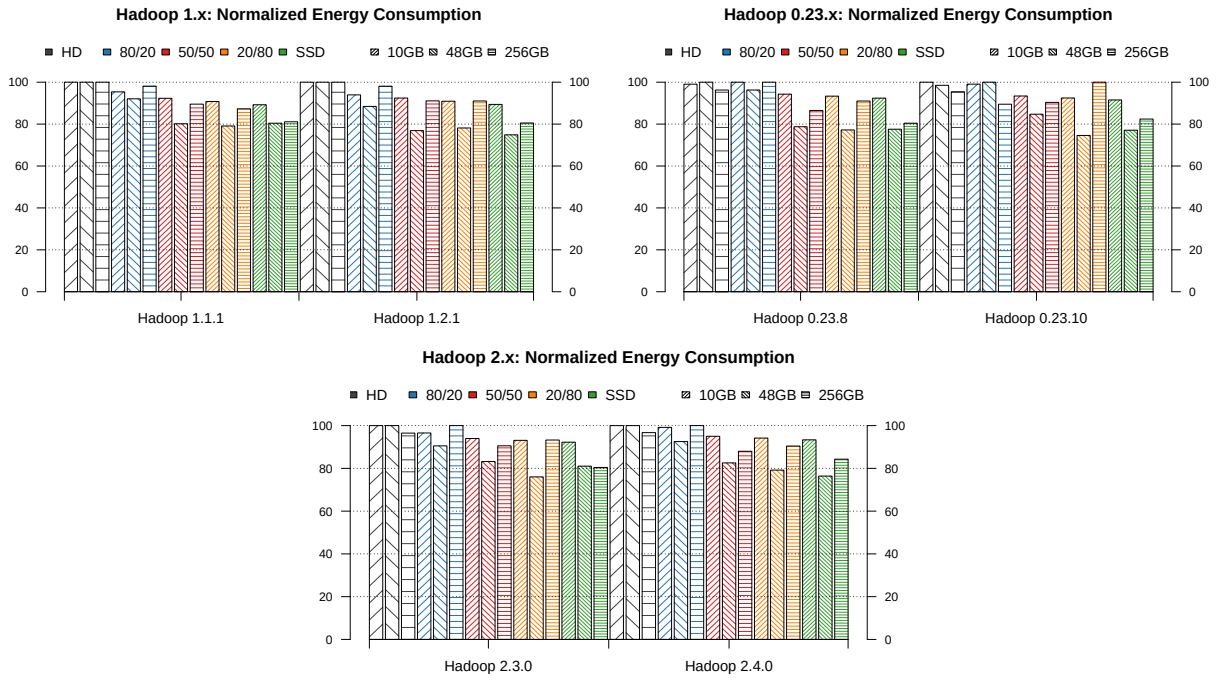


Figure 5.4: $HDFS_H$ *Normalized Comparison of Sort Benchmarks*

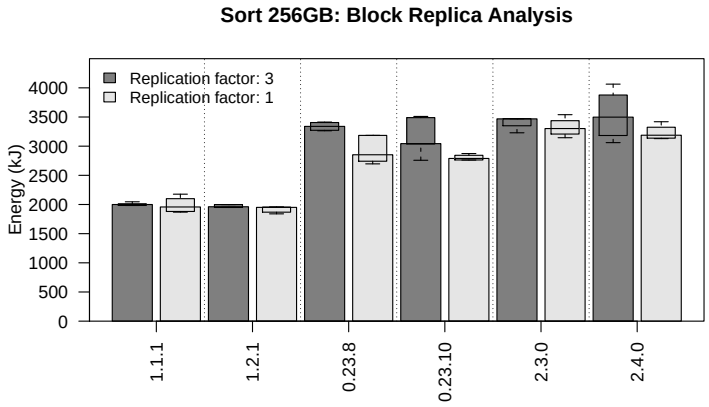


Figure 5.5: *Energy Influence on Triple Block Replica (HD Configuration)*

to the *SSD* energy performance, including all releases and Sort benchmarks.

Then, we investigated the issue raised by the *SSD* space limitation on our infrastructure to discover the effects of block replication on energy consumption. This set of experiments used the 256GB Sort dataset and ran using the *HD* configuration. The energy results are presented in Figure 5.5. Analyzing the results we can conclude that the triple block replication demands on average 1.5% more energy in 1.x releases, 9.5% in 0.23.x releases, and 6.2% in 2.x releases. Based on previous observations, in the configurations favoring the use of SSDs, this percentage is reduced, since SSDs demand far less energy than HDs.

5.3.2 Results for the CPU-Bound Benchmarks

$HDFS_H$ is directed to I/O-bound Hadoop jobs. We have confirmed that the use of isolated HDs or SSDs does not bring any benefit in the case of CPU-bound jobs. Even so, there was the need to investigate whether our hybrid approach could bring different results in terms of energy consumption to Hadoop. Therefore, we performed two sets of experiments using CPU-bound jobs. The differences between the hybrid HD and SSD configurations were insignificant. Figure 5.6 presents the results for the Join benchmark. Note that the differences are irrelevant between the configurations, as observed in the previous CPU-bound experiments using the WordCount application.

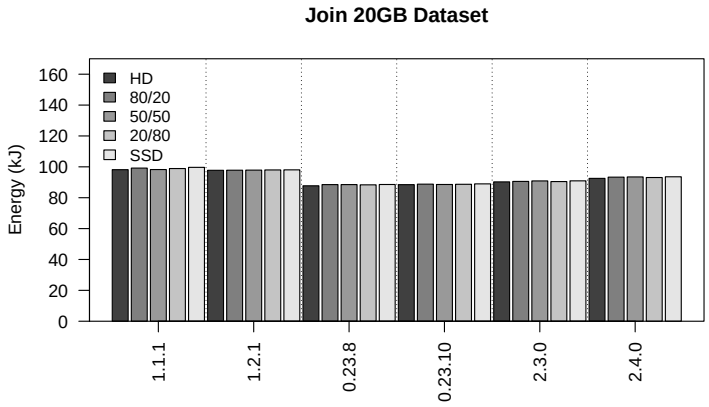


Figure 5.6: *Energy Consumption on Join Benchmark*

Mahout K-Means is a hybrid benchmark: it is CPU-bound in the iterations and I/O-bound during clustering. With our setup (3 iterations), 3/4 of the execution in this benchmark is CPU-bound, while the remaining 1/4 is I/O-bound. The CPU part of the execution dominated the

I/O-part as Figure 5.7 shows. There is no significant difference across the configurations, except for the differences among branches. We simplified the figure, presenting only the results for the two extreme configurations (HD and SSD).

Finding 7: Except for the inter-branch differences, there are no significant performance gains and energy consumption reductions when running CPU-bound jobs with our approach.

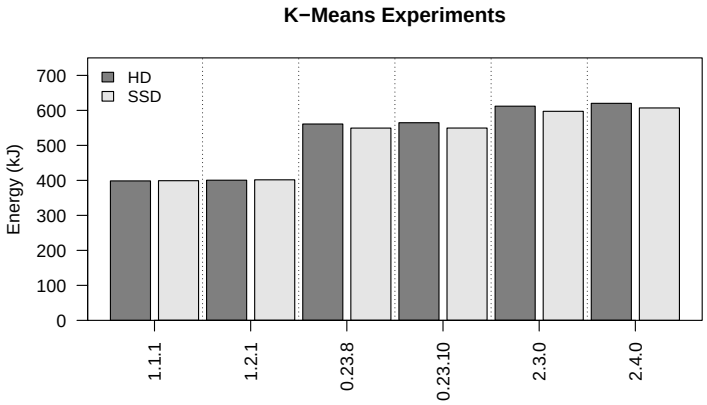


Figure 5.7: *Energy Consumption on K-Means Benchmark*

5.3.3 Using SSDs as Temporary Storage Space

From the beginning, all experiments used HDs as temporary space available during benchmarks, since this is the default Hadoop behavior unless manually specified. Following other approaches discussed in Chapter 3, which somehow incorporate SSD into Hadoop — most of them using SSDs as a cache-tier — we decided to experiment with the use of SSDs as temporary storage space. We performed a set of experiments using the *SSDzone* to store the temporary files generated during Hadoop jobs. We named this configuration *tmpSSD*. It stores the dataset file blocks in the *HDzone*, and all the temporary files generated during job execution in the *SSDzone*.

The results were promising, as presented in Figure 5.8, including the 48GB and 256GB Sort energy consumption rates. The smallest dataset (10GB) did present the same behavior, except for releases 1.x, which did not achieve major benefits, performing similarly to the SSD configuration. This is due to the dataset size, where the instantiation time dominates the execution time, and the data movement between storage zones degrades the overall job makespan and, consequently, the energy performance.

We observed a different behavior with the 0.23.x and 2.x releases, although they have the same software architecture. Releases from the 0.23.x branch did not achieve any energy benefits by using the *tmpSSD*. The opposite happened with the 2.x releases, which achieved better performance and consequently reduced their energy demands. The same pattern developed for the larger datasets in the 0.23.x and 2.x releases. The novelty was the 1.x branch results, which performed better with larger datasets. These results not only did performance improve, but there were significant energy savings when using the *SSDzone* as temporary space.

Finding 8: Releases 1.x and 2.x were significantly improved when using the *SSDzone* to store temporary files during the execution of Hadoop jobs. Such improvements were so significant that in most cases performance was even better than storing the dataset files in the *SSDzone*.

In contrast, releases from the 0.23.x branch did not perform well, and in some cases the results suggest that the use of this strategy may be prohibitive, especially with small datasets. Therefore,

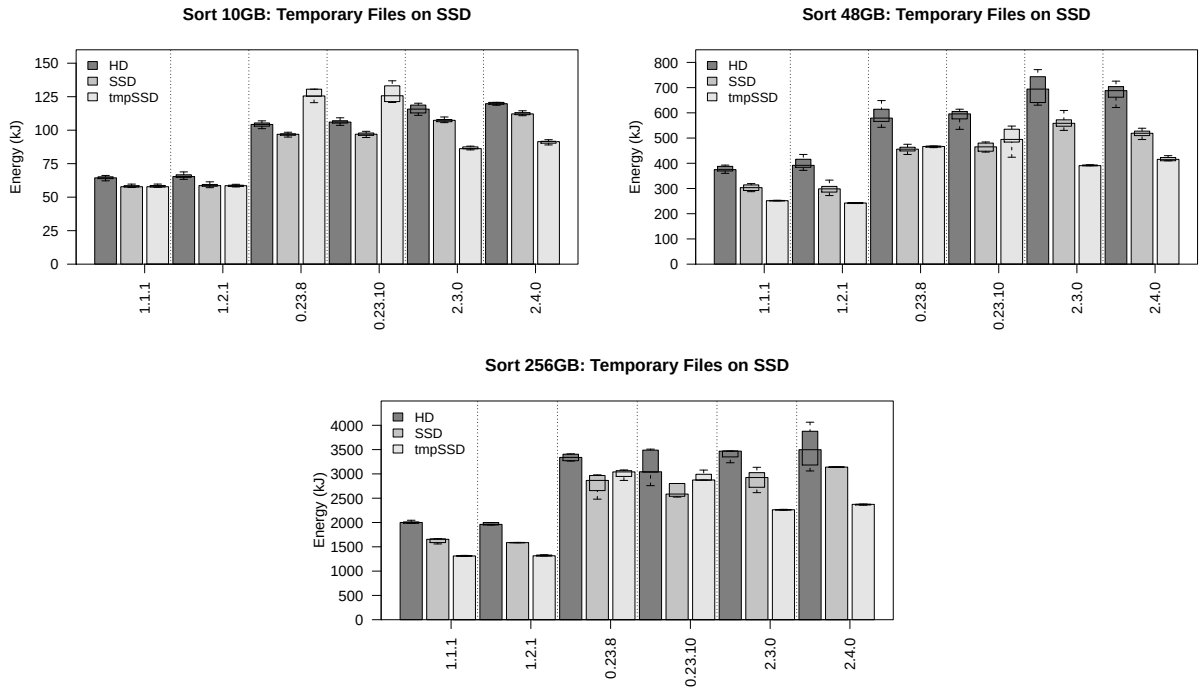


Figure 5.8: *Energy Consumption Rates Using SSDzone as Temporary Space*

using SSDs as part of a hybrid storage system offers two kinds of benefits for Hadoop computations: (I) primary storage in HDFS; and, (II) temporary storage space. In the latter case, changes in the behavior of temporary files allowed several releases to achieve better performance using the *tmpSSD* configuration. Table 5.6 presents the energy consumption rates achieved and the comparison with the *HD* and *SSD* configurations.

Table 5.6: *Average Energy Consumed for the tmpSSD Configuration (kJ)*

Dataset size	Release	*HD*	*SSD*	Reduction	*tmpSSD*	Reduction	Reduction from *SSD* to *tmpSSD*
10GB	1.1.1	65	58	11%	58	11%	0%
	1.2.1	66	59	11%	59	11%	0%
	0.23.8	104	97	7%	129	-24%	-33%
	0.23.10	106	97	8%	127	-20%	-31%
	2.3.0	116	107	8%	87	25%	19%
	2.4.0	120	112	7%	91	24%	19%
48GB	1.1.1	378	304	20%	252	33%	17%
	1.2.1	398	298	25%	242	39%	19%
	0.23.8	588	456	22%	466	21%	-2%
	0.23.10	593	464	22%	497	16%	-7%
	2.3.0	696	564	19%	391	44%	31%
	2.4.0	682	521	24%	418	39%	20%
256GB	1.1.1	2005	1626*	19%	1314	34%	19%
	1.2.1	1972	1588*	19%	1322	33%	17%
	0.23.8	3339	2790*	16%	3007	10%	-8%
	0.23.10	3168	2736*	14%	2938	7%	-7%
	2.3.0	3461	2885*	17%	2262	35%	22%
	2.4.0	3530	3077*	13%	2369	33%	23%

All reductions calculated based on the *HD* values. Negative values represent increase in energy consumption.
* HDFS Replication Factor = 1

5.3.4 Performance and Speedup

Regarding job makespan performance, we observed that storing more data in the SSDs enabled jobs to run faster, which was expected, since SSDs provide higher throughput. With the 10GB

dataset (Figure 5.9), the 80/20 configuration was on average 6% faster than the HD configuration, with only 20% of the data processed from the SSD; in the 50/50 configuration, jobs were on average 16% faster than the HD configuration; and, in the 20/80 configuration, 21% faster; finally, jobs running with the SSD-only configurations were on average 25% faster than purely running on HDs.

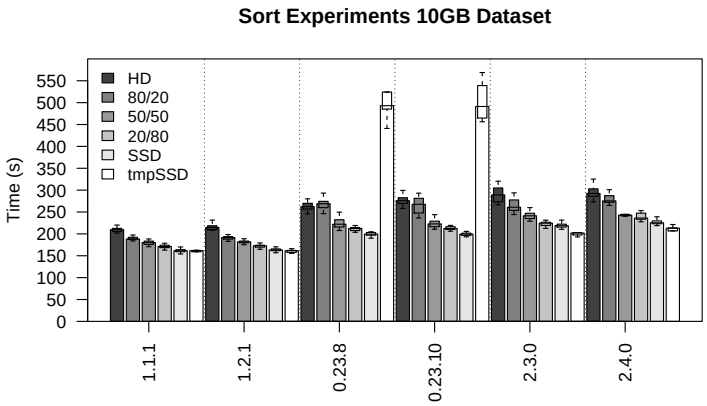


Figure 5.9: *Hadoop Performance: Multiple Releases over Configurations*

In the 48GB Sort, the observed speedups compared with the HD configuration were: 80/20, 8% faster; 50/50, 27% faster; 20/80, 32% faster; and SSD, 30% faster. In this particular case, we notice one hybrid configuration 20/80 achieving better results than the *SSD*. The average makespan for the Sort benchmarks can be seen in Figure 5.10. The 256GB Sort experiments presented almost the same panorama. In general, the 50/50, 20/80, and *SSD* were faster than the *HD* and 80/20 configurations. In this case, the *SSD* was the fastest configuration (19% faster), followed by the 50/50 (13%), putting the 20/80 in third with 9% of average speedup.

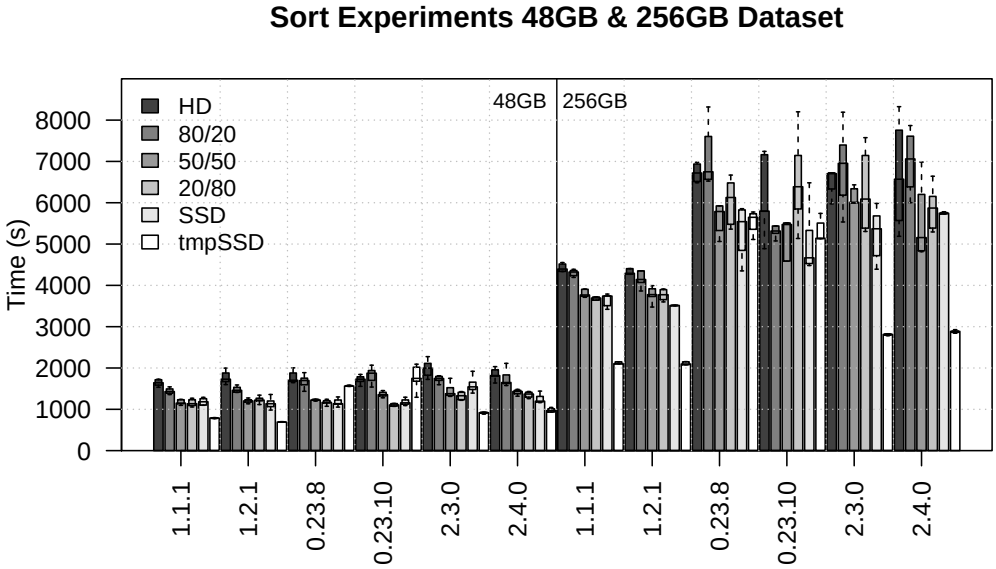


Figure 5.10: *Hadoop Performance: Multiple Releases over Configurations*

With the *tmpSSD* configuration, we achieved more promising results than in some *SSD* experiments. All experiments using the hybrid policy ran using HDs as temporary space. The *SSD* configuration stores the entire dataset on SSDs, and uses HDs as temporary space. The *tmpSSD* configuration does exactly the opposite, storing the dataset in the *HDzone*, and the temporary files

in the *SSDzone*. Thus, it is fair to compare these two experiment sets. Although the HD experiments tend to be much slower than the SSD ones, in this case the $tmpSSD$ outperforms in most cases the SSD experiments.

We notice that except for 0.23.x releases, the $tmpSSD$ configuration achieved major performance increases, especially on the larger datasets. Compared to the HD configuration, the $tmpSSD$ configuration performed on average 27% faster on the 10GB Sort, 53% faster on the 48GB Sort, and 54% faster on the 256GB Sort. Compared with the SSD results, $tmpSSD$ performed the same on the 10GB Sort, 34% faster on the 48GB Sort, and 44% faster on the 256GB Sort. Results from the 48GB and 256GB Sort experiments in 1.x and 2.x releases point to a speedup factor of more than twice when comparing the HD and $tmpSSD$ configurations, and on average more than 1.5 times when comparing the SSD and $tmpSSD$ configurations. It is worth to remind that, since not all energy consumed is derived from the job makespan, the same reduction rates observed in the job makespan cannot be applied to the energy consumption, as presented in the previous section. Table 5.7 presents all the performance data and the comparisons that were carried out.

Table 5.7: *Sort Benchmarks: Average Job Makespan* (s)

Dataset size	Release	HD	80/20	%	50/50	%	20/80	%	SSD	%	tmpSSD	%
10GB	1.1.1	212	190	10%	180	15%	171	19%	162	24%	160	25%
	1.2.1	216	191	12%	183	15%	172	20%	163	25%	161	25%
	0.23.8	264	270	-2%	225	15%	214	19%	200	24%	512	-94%
	0.23.10	278	266	4%	225	19%	214	23%	200	28%	502	-81%
	2.3.0	291	267	8%	242	17%	223	23%	220	24%	200	31%
	2.4.0	296	280	5%	245	17%	239	19%	227	23%	212	28%
48GB	1.1.1	1644	1454	12%	1191	28%	1155	30%	1184	28%	788	52%
	1.2.1	1763	1474	16%	1200	32%	1228	30%	1138	35%	693	61%
	0.23.8	1767	1670	5%	1227	31%	1172	34%	1167	34%	1566	11%
	0.23.10	1726	1795	-4%	1376	20%	1110	36%	1175	32%	1766	-2%
	2.3.0	1983	1727	13%	1465	26%	1325	33%	1589	20%	914	54%
	2.4.0	1849	1749	5%	1415	23%	1354	27%	1267	31%	979	47%
256GB	1.1.1	4428	4300	3%	3806	14%	3667	17%	3641	18%	2122	52%
	1.2.1	4318	4156	4%	3779	12%	3768	13%	3517	19%	2111	51%
	0.23.8	6726	7085	-5%	5606	17%	6023	10%	5284	21%	5545	18%
	0.23.10	6177	5530	10%	5597	9%	6543	-6%	5097	17%	5331	14%
	2.3.0	6696	6852	-2%	5974	11%	6264	6%	5229	22%	2810	58%
	2.4.0	6663	6998	-5%	5599	16%	5869	12%	5541	17%	2882	57%

The % columns refer to the speedup percentage calculated based on the HD values.
Negative values represent loss in performance.
* HDFS Replication Factor = 1

5.3.5 Cost Model Analysis

Besides energy and performance, our research is also concerned with storage cost, as this plays a strong role for corporations. For this first analysis, we assume that the $HD_{cost/GB}$ = \$0.05 and the $SSD_{cost/GB}$ = \$1.00 (20 times the $HD_{cost/GB}$). Such prices were practiced on average during 2014, when our first experiments were deployed. Figure 5.11 presents the results. We plotted the ratio between job makespan and job storage cost for the Sort jobs, using the tested releases from each branch. Note that the behavior of each individual release over the multiple configurations presents a pareto optimal configuration. The pareto optimal configurations[1] rely on resource tradeoffs. This concept was created and used by the Italian engineer and economist Vilfredo Pareto, which applied it in his studies of economy efficiency and income distribution. In our context, this means that we have to choose between the storage cost and the desired performance. This tradeoff is clear in Figure 5.11: the more SSDs are is used, the more expensive the storage becomes, but with less hours spent to perform the jobs, meaning that less energy is used. On the other hand, by using more HD space in the configurations, jobs take more hours to finish, demanding more energy, although the cost greatly decreases. This example can be used to identify which proportions are the best in the

[1] http://en.wikipedia.org/wiki/Pareto_efficiency (Visited on 15/10/2016)

tradeoff between cost and performance. On the bottom of the figure are the results from the HD configuration, while on the top are the SSD results. We can also notice the performance differences between releases, where the YARN releases performed worse with respect to job makespans than the 1.x releases.

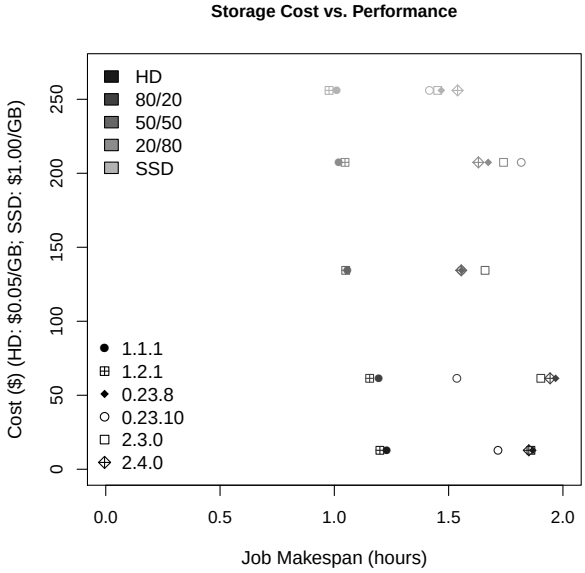


Figure 5.11: *$HDFS_H$ Cost Model Analysis: 256GB Sort Across 5 Configurations*

Most of the energy consumed by Hadoop comes from the job makespan. Therefore, it makes sense to lower the job makespan to reduce energy consumption. Additionally, the choice of storage device has also great impact on energy. While the cost affects this choice, the reduction in the cost/GB of SSDs raises a new trend. Analysts predict that the price of SSDs will equal that of HDs in a couple years (Mea15; All16). With our first studies dating from 2013/2014 and following the price reduction trend, we updated our cost analysis by decreasing the SSD cost/GB, using as a parameter the average prices on 2014 ($1.00/$GB$), 2015 ($0.50/GB), and the 2016($0.25/$GB$).

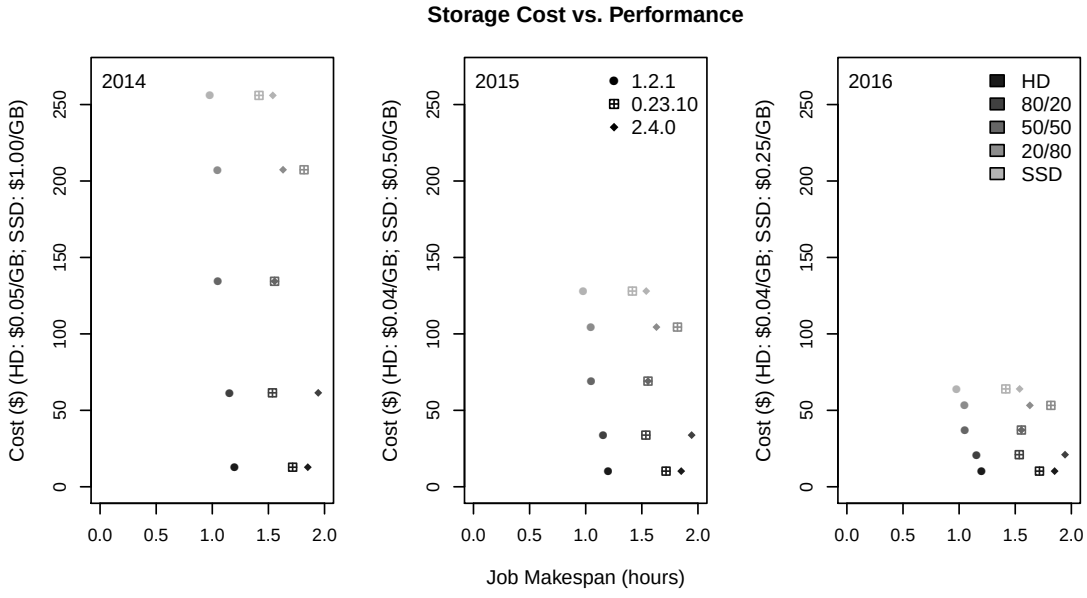


Figure 5.12: *$HDFS_H$ Cost Model Analysis: 256GB Sort over Time*

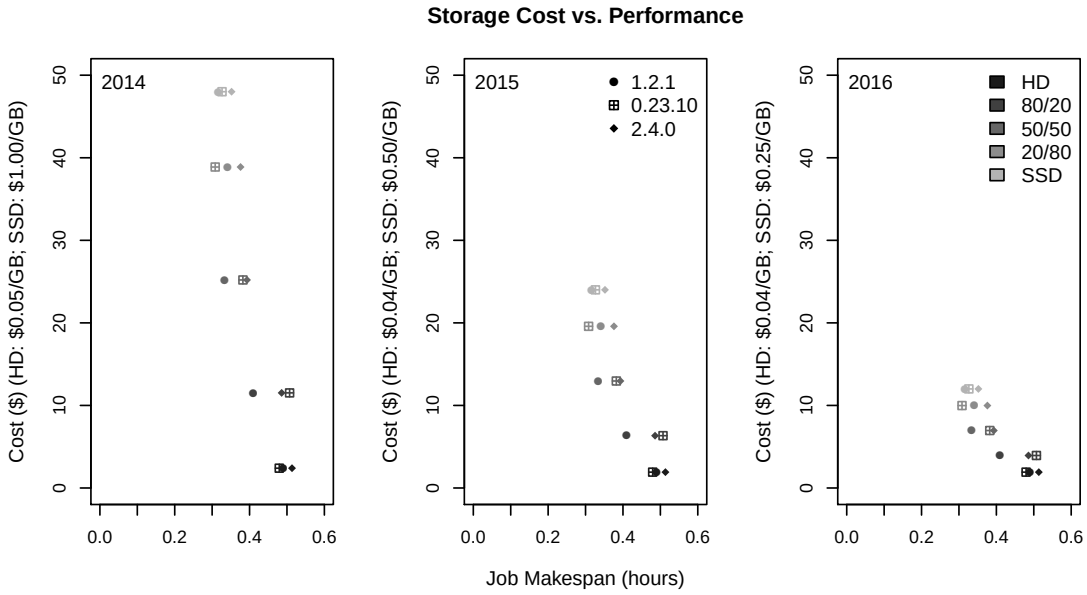


Figure 5.13: *$HDFS_H$ Cost Model Analysis: 48GB Sort over Time*

Figure 5.12 presents the results. The cost of HDs was also updated to $\$0.04/GB$. Since the behavior of the intra-branch releases is the same, we kept only the most recent studied release from each branch (1.2.1, 0.23.10, and 2.4.0).

The SSD price reduction brought an interesting fact: with about the same price from 2014 we are able in 2016 to introduce 100% of SSD storage space. This represented only 20% of the SSD storage in 2014. The same behavior and evolution can be observed in every release on the 48GB datasets tested in the sort experiments (Figure 5.13). In addition, we observe that, although there is a general behavior within the tested releases, each one has its particularities, which means that the optimal point varies from one release to another, specially amongst different branches.

We also applied our cost model to the *tmpSSD* configuration. The cost was calculated by adding up the cost/GB from HDs and SSDs, since the benchmark runs from the HDs and use the

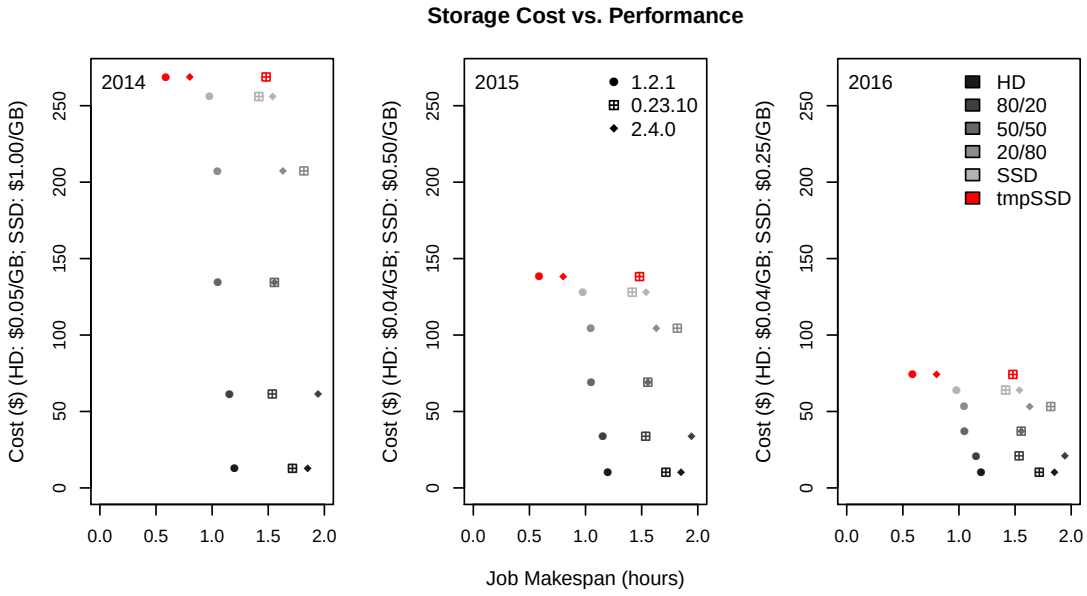


Figure 5.14: *$HDFS_H$ Cost Model Analysis of tmpSSD: 256GB Sort*

SSDs as temporary storage. Figure 5.14 shows the cost/performance ratio of the three latest tested releases from each branch over the hybrid configurations plus the *tmpSSD* results. The novelty presented is that, as for 2014, the cost of this approach was prohibitive for a couple reasons: first, because of the cost, which was more than processing data directly from SSDs; and second, because of the limitations at the time, e.g., SSD limit of write operations. But, as the technology evolved and the cost was reduced by approximately 75%, this seems to be a viable and feasible option for the near future. The results achieved were promising in terms of performance, and now they are also prominent in terms of cost/benefit. Additionally, as performance increased, less energy was consumed. Add to the mixture the future SSD technologies and this is a sustainable solution to introduce hybrid approaches on clusters.

5.4 Final Considerations

In this chapter, we presented our approach to transparently integrate HD and SSD technologies into $HDFS_H$. We designed a block placement policy that directs the file blocks to specific storage zones using storage proportions. The new policy is flexible enough to allow the configuration of the amount of data stored in each zone. We assessed our approach over multiple configurations, including three hybrid storage configurations: HD, 80/20, 50/50, 20/80, SSD, and $tmpSSD$. The main goals of our research were to promote energy consumption reduction and performance increase by using hybrid storage policies on Hadoop. Both objectives were achieved.

Regarding performance improvements, the use of SSDs was, by concept, a simple strategy to increase storage performance. However, the cost/GB had to be considered, as few years ago it was prohibitive for large storage systems. Other issues with solid-state drives were reliability (high fail rates) and durability, especially concerning the maximum number of writing operations a device can perform. In a short period of time, SSDs benefited from recent technological advances, promoting the overall cost reduction while increasing the average drive capacity, besides the reduction on fail rates. Yet to come in a short time, new technologies, such as 3D NAND Flash Memories (Int15), will enlighten even more this subject.

By incorporating SSDs into storage zones in a hybrid fashion, we promoted an overall performance increase as observed in the experiments. With only 20% of SSD storage space, there was an average increase of 5% in speed, regardless of the Hadoop version or dataset size. The results point to a speedup of 19% regardless of version or dataset when using 50% of the storage space from SSDs, while using only SSDs the speedup is around 25%. As demonstrated on Hadoop jobs, the larger the dataset, the more benefits we can achieve. Due to its architecture and the used programming languages, the instantiation times of JVM, memory and resource allocation become more evident if the jobs are shorter. Consequently, we achieved better results when running the 48GB and 256GB Sort experiments, if compared to the 10GB Sort. From the 48GB to the 256GB Sort there were no significant improvements, showing that the speedup rates tend to remain stable on larger datasets.

Concerning energy consumption, our block placement policy shows a reduction even when only a fraction of the data is stored in the modified HDFS. We showed that, with larger datasets, the reduction in energy demand can be significant, achieving up to a 20% savings under certain hybrid configurations. The 50/50 and 20/80 configurations proved to promote energy consumption reductions close to the SSD configuration in the I/O-bound experiments, as presented in Table 5.8.

Table 5.8: *Average Energy Reduction Percentage*

Sort Experiment	*HD*	80/20	50/50	20/80	*SSD*
10GB	*	3%	7%	8%	9%
48GB	*	7%	19%	22%	22%
256GB	*	0%	8%	5%	16%

**HD* values were used as base for comparison

The general use of $HDFS_H$ affords immediate benefits since it increases MapReduce jobs' performance and reduces energy consumption. In addition, we noticed that the energy reduction rates are smaller than the job makespan ones. The main reason is that there is not an exact correlation between job makespan and energy consumption, as most of the energy spent is determined by the performance and type of storage used. Yet, the branch/release and dataset size pair play a fundamental role in this case. We noticed that the YARN development brought an increase in energy consumption that is not directly correlated to job makespan. We observed experiments where the job makespan from YARN releases were close to 1.x releases, but the energy consumed was much higher. This has a direct relation with the architectural changes, especially the insertion of an extra layer – which Hadoop 0.23.x and 2.x cannot run without – controlling the resource usage. We acknowledge that YARN brought flexibility to the framework, but generated a significant performance loss. Since most of the energy consumed by Hadoop is associated with job makespan, branches 0.23.x and 2.x releases almost double the energy consumed to run the same jobs compared to 1.x releases. Users must focus on the real needs associated with Hadoop: flexibility or performance.

A remarkable result was the use of the $tmpSSD$ configuration. This particular configuration brought major performance increases in the experiments: on average it was 50% faster than the HD configuration and almost 40% faster than the SSD configuration. The SSD cost/GB might have been prohibitive a few years ago, and its use was destined to "special" and/or "hot" data. The cost reduction presented in the cost model analysis can be a game changer. The novelty of the cost of SSDs costs getting closer to that of HDs can bring new hybrid storage possibilities, such as the use of SSDs as temporary space ($tmpSSD$ configuration).

Chapter 6

Discussion

We implemented and tested a hybrid storage approach on 6 Hadoop releases from three different branches. In this chapter, we further discuss the key findings based on the observations we made. Our results indicate that, in most evaluated configurations, $HDFS_H$ promotes overall job performance increases while decreasing energy consumption.

6.1 Findings

One of the most important findings during the grounding experiments was the discovery of different behaviors of the Hadoop releases. The intra branch results are consistent with the general code evolution; newer releases generally receive bug fixes and code optimization. Thus, performance results are better, resulting in lower job makespan and, consequently, promoting energy consumption reduction in most cases. But when we consider the inter-branch comparison, the results presented a different scenario, with the newest releases that included YARN achieving worst performance results. Even more concerning, the energy consumption rates increased disproportionally to the performance increases, which means that severe overhead was introduced when changing the software architecture of such releases.

Chapter 4 presented such results and proposed answers to research question **RQ2**. Although our hybrid approach had not been tested yet at the time, we could notice the differences between the use of HDs and SSDs on Hadoop. As expected, I/O-bound jobs were the ones that benefited the most from the use of SSDs – they are capable of performing more IOPS than HDs, resulting in higher throughput; therefore they improve the general performance of a system. Hadoop inherited these

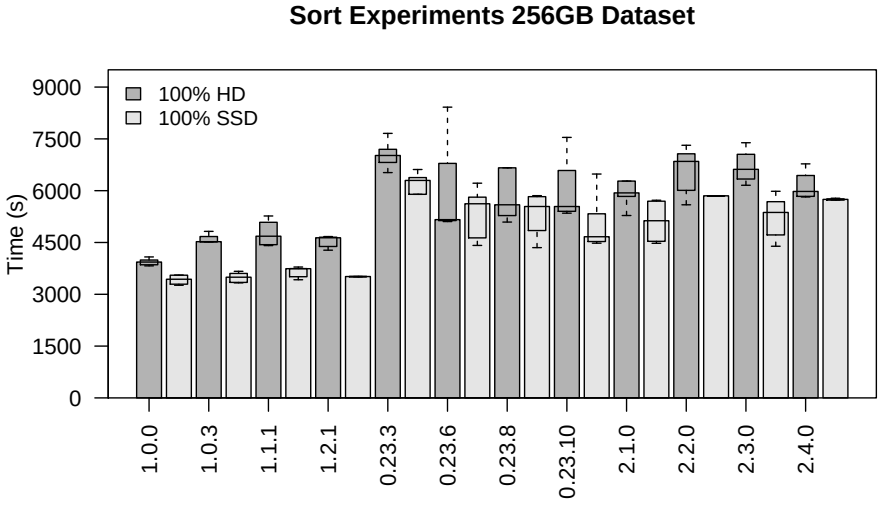


Figure 6.1: *Hadoop Sort 256GB Job Makespan*

benefits as well. Jobs running on Hadoop 1.x releases, specially 1.1.1 and 1.2.1, were 25% faster when using SSDs, regardless of dataset size. This speedup rate represented more than 16% of reduction on average energy consumption. Although releases 0.23.x and 2.x have performed on average the same as 1.x in terms of job makespan, the speedup on these releases represented only 12% of energy consumption reduction under the same circumstances. We also notice that releases 1.x improved performance on the 48GB and 256GB datasets, while the 0.23.x and 2.x releases performed better in the 48GB dataset, losing significant performance on the 256GB dataset. This had a direct effect on the power consumption of these releases on such experiments. This observation was the key finding of these experiments. There is a significant difference in energy consumption between branch 1.x and branches 0.23.x. and 2.x. The 256GB Sort experiment illustrated this difference on HDs and SSDs, as seen in Figure 6.1, presented in Chapter 4 and reproduced here again.

Another finding from these experiments is the variation on the results. We observed that results from 1.x releases were close together, regardless of HD or SSD use. Results from the other two branches were more scattered, with a larger variation. This can be observed in the boxplots of almost every figure depicting the performance or energy consumption results. We agree that this behavior would be more likely to happen in the experiments using HDs, which have multiple tracks and sectors, and some can have multiple disk plates. Also, the read and write operations are subjected to variating seek times and non-contiguous block storage which can particularly affect Hadoop experiments, since HDFS runs on top of the regular operating system. But we observed that this effect was much more evident on YARN releases.

If YARN severely affects the performance, we would expect the user community to perceive this issue. Even further, if the issue had been reported by users, there should be an effort by the developers to promote fixes and the framework optimization targeting performance improvements. But, a few problems contribute in this matter. First, there are no energy studies concerning Hadoop up to this date, at least, those connecting performance with energy consumption. Second, there is no track of the Hadoop performance by users: if a job doubles the makespan, users are not aware of the causes, since Hadoop deals with huge datasets, and this is not a top priority.

We confirmed this issue when mining StackOverflow in search of Hadoop performance questions. Seeking for a relation between YARN and performance issues, we notice that this is not perceived by users. Most of the questions regarding Hadoop's performance are related to the Hadoop configuration, showing that users do not know this issue. Additionally, Apache recommends the download of the newest releases – as for today, 2.7.2 is most recent stable release – directing users to releases that can produce poor performance results if compared to previous releases from other branches. Another problem here is that branch 1.x was discontinued, and the last release (1.2.1) did not receive any bug fixes and improvements in three years. Therefore, those seeking for an exclusive MapReduce platform are directed to YARN releases. In fact, this should not be an issue for casual users, but for large clusters constantly mining Big Data using MapReduce this could represent a significant waste of resources.

Searching for answers regarding performance issues present in the YARN releases, we conducted an analysis of the source code with CKJM-extended. After analyzing the source code of 12 Hadoop releases, the metrics pointed to an expected pattern of the modifications: as code grew, the architectural modifications introduced by YARN caused loss in performance, and greatly increased energy consumption. The new architectural layer and the obligation of running Hadoop using it were the apparent cause of this issue, since there is more communication overhead between the Hadoop layers. We were able to strengthen the answers for our second research question, **RQ2**, relating the loss in performance with the modifications on the Hadoop source code and architecture.

With such different behavior from Hadoop releases in the grounding experiments, we needed to discover what benefits the hybrid approach could bring to the framework. Our first research question, **RQ1**, is directly related to this topic. Yet, Hadoop's response to the mixed environment was uncertain. We implemented the hybrid policy in 6 Hadoop releases and performed the tests using 3 hybrid configurations: one favoring HDs (80/20), another using equal amounts of each (50/50), and the last using more SSDs (20/80). The findings from Chapter 5 showed the importance

of the mixed environment. Results of the hybrid experiments demonstrated that the proportions behavior is not linear. This key finding is the most important one in this thesis: in certain cases, with only 50% of the storage space using SSDs, we achieved results that are close to the use of 100% SSDs in HDFS. Again, as observed in Chapter 4, releases 1.x performed better and kept the previously discovered behavior, with releases 1.x producing better results than releases 2.x and 0.23.x. Figure 6.2 summarizes the results and shows the benefits of our approach in terms of energy consumption reduction.

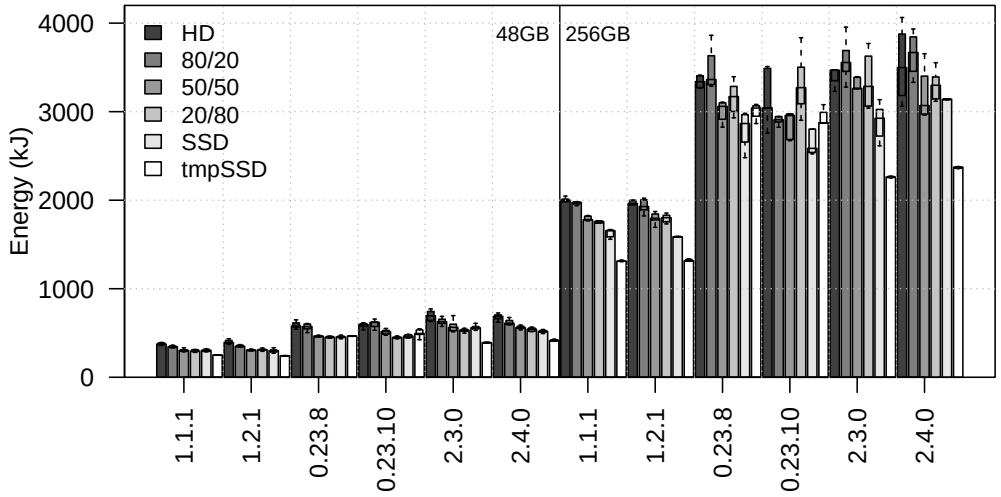


Figure 6.2: *Energy Consumption Results: Sort 48GB and 256GB*

We can notice another key finding of the thesis: the results achieved when using SSD devices as temporary storage space while using HDs as regular HDFS, particularly, an unexpected and original result, since it was not the initial focus of our investigation. The results were promising, especially because they improved performance beyond the speedups achieved with the use of SSDs only. These results show the substantial influence of temporary space on Hadoop computations. Hadoop writes all the intermediary results on disks, which is then sorted and shuffled among all nodes performing Reduce tasks. SSDs have a huge capacity of performing read and writing operations with high throughput, and therefore, improved the overall Hadoop performance, since only the final results of the computations were written in the HDs. The results were remarkable in releases 2.x, which presented poor energetic performances in the experiments. This configuration brought these releases closer to the 1.x releases, and presented a scenario of what could possibly be the real expected behavior of Hadoop 2.x releases after the architectural changes and implementation of new features.

Although Hadoop 0.23.x releases are close to the 2.x branch releases, they behaved differently. We can observe that branch 0.23 did not benefit at all from the *tmpSSD* experiments. In fact, performance was at the same level of the hybrid configurations. These releases have almost the same components, except for Hadoop High Availability. Nevertheless, we noticed that source code commit activity on Hadoop 2.x is much higher. Additionally, the number of releases from each branch is another clue on the Hadoop development focus: while branch 0.23 had only 1 release after 0.23.10, Hadoop 2.x had at least 12 releases in the same period. We can credit such performance differences to the lack of patches and fixes in the branch. However, our code analysis did not present substantial differences that could justify these effects and an in-depth analysis should be conducted targeting all the 0.23.x releases to a better understanding of the causes.

Using SSD as temporary storage space would be considered prohibitive a few years ago. At that time, SSDs had two major limitations: cost and maximum number of writing operations. Due to the cost per GB, which could reach 20 to 30 times the cost per GB of HDs, SSDs were not even considered a choice for temporary storage space. As cost reduced to the 2016 levels of 4 to 5 times the cost per unit of HDs, this approach should be considered. There is an expectation that

within 2 years the cost per GB of HDs and SSDs should be nearly the same. From this point of view, significant improvements could be achieved if, instead of replacing HDs, temporary space was introduced in Hadoop clusters using SSDs. This finding could bring energy consumption reductions allied with performance improvements.

The use of SSDs in Hadoop on both of the aforementioned situations – either using the Hybrid approach or the temporary space configuration – generates savings in the long term. But there are other factors that should be considered. First, the cost is, in our opinion, one of the most important factor. What would be the financial savings of adopting our hybrid approach? We conducted a study regarding our cost model and, as show in Section 5.3.5, SSDs have strongly benefited from the price reduction over the last 2 years. Additionally, energy consumption can be reduced with the introduction of SSDs in clusters, as we already demonstrated. On the $HDFS_H$ side, we can achieve approximately 15% of energy consumption reduction (averaging the 50/50, 20/80, and SSD configuration results) if comparing with HD-only storage for the 256GB experiments. This represents a reduction of 0.12kWh on energy consumption for our small 9-node infrastructure. This represents more than US$70 dollars/year of energy savings, considering an average cluster usage of 75% of the available time (running Hadoop jobs 18h/day, 270 days/year) and today's average energy price in the U.S. ($0.12/kWh)[1]. When we scale-up the results, the savings would became really prominent. Data centers run 24/7, and at a large Hadoop cluster, say 4,500 nodes on 40,000+ machines at Yahoo! (Asa14) or other companies such as Facebook, these savings completely justify the cost of introducing SSD space into clusters, which is already happening.

Even more surprising is the analysis of the $tmpSSD$ results. Regarding 1.x and 2.x releases, some of the experiments using this configuration achieved more than 30% of energy consumption reduction if compared to the HD results. If we consider today's average energy price on the U.S. ($0.12/kWh), the benefits of using our approach could save something close to 0.33kWh to run a 256GB Sort experiment, which takes approximately 1 hour to run. If we consider the same average cluster use of 75% of the time, this should be almost 200 dollars per year. With the price/GB around $0.35, the savings are enough to buy more than 500GB of SSD space. Given the SSD reduction price predictions for the next 2 years, users could buy even more storage space with the energy savings achieved, besides the benefits received from the performance increases.

Besides the financial benefits, we must consider the environmental benefits associated with the use of SSDs. First, they demand much less energy than HDs to work. But, a few other characteristics favors the reduction of environmental issues. Solid-state drives are smaller than HDs, and consequently, we can fit more SSDs in the same space used by HDs. SSDs also generate less heat in data centers. As a result, less energy is used to cool the whole infrastructure. Thus, we can have smaller data centers, with higher storage capacity, and demanding less energy for powering and cooling. At the same time, data centers can experience performance increases and overall cost reduction.

6.2 Threats to Validity

This thesis focuses on the effects of a hybrid storage approach applied to Hadoop. In our studies, external validity is threatened by the focus on the Hadoop project. What happens to Hadoop might not happen to other projects, thus we cannot safely generalize the findings beyond the Hadoop project. Although we decided to analyze the Hadoop framework, there are several other MapReduce frameworks available such as: Dryad (IBY+07), which is not a purely MapReduce framework, but uses the major concepts of MapReduce; DISCO (MTF11), another open source implementation of MapReduce in Erlang and Python; MARS (HFL+08), which runs on GPUs, implemented in C++; and SkyNet[2], an open source implementation of MapReduce in Ruby. We did not compare the Hadoop results with these, which may present different behaviors in other platforms. Additionally, we selected 12 Hadoop releases during our research, and only 6 of them were tested using our

[1] http://www.eia.gov/electricity/monthly/epm_table_grapher.cfm?t=epmt_5_6_a (Visited on 15/10/2016)
[2] http://skynet.rubyforge.org/ (Visited on 15/10/2016)

approach. Since new releases were launched after our selection, such releases may present different behaviour.

Furthermore, our workload range may represent a threat, since not all MapReduce jobs work in the same way during Map and Reduce phases. This threat is tempered by the focus on I/O- and CPU-bound workloads. Even though our experiments were limited to the Sort application in the I/O-bound benchmark class, we successfully established a relation between energy consumption and cost of I/O operations in Hadoop. The CPU-bound benchmarks (K-Means, Join, and WordCount) results showed that there is no significant benefit of using the approach, since CPU time dominates over I/O operations.

Finally, we tested our approach on a dedicated cluster, which may not reflect the behavior of Hadoop running on the Cloud, specially on virtual machines. In addition, external validity is threatened by our testing infrastructure, which had a limited HD and SSD storage space. While *HDzone* had $8 \times 1TB$ of available space, the *SSDzone* had $8 \times 120GB$, totaling less than 1TB of SSD space. Therefore, we did not test our approach on large-scale datasets. Although there is an effort to predict the Hadoop behavior on larger datasets, only real experiments can confirm the benefits of the hybrid approach in such scales.

Internal validity is also threatened by the narrow benchmark selection. Although most MapReduce jobs are either CPU-bound or I/O-bound, heterogeneous workloads and iterative MapReduce jobs can produce unexplored results, since our research was tested on single MapReduce jobs. This threat is reduced by the K-Means benchmark used to evaluate $HDFS_H$, which is an iterative hybrid benchmark (3/4 CPU-bound and 1/4 I/O-bound).

Chapter 7

Conclusions

Hadoop became a synonymous of Big Data processing using MapReduce over the last ten years. Yet, in these processing infrastructures, energy consumption poses a significant challenge for companies today. In this thesis, we presented $HDFS_H$, a hybrid storage system that seamlessly mixes SSDs and HDs into HDFS, promoting energy consumption reduction with performance improvements.

Our approach demonstrated to be a viable opportunity to enhance the performance of Hadoop clusters while decreasing their energy consumption. We introduced two mechanisms using SSDs: first, using a mixed environment and, second, using SSDs as temporary storage space. In both approaches, there are significant advantages, promoting, in some cases, a reduction of more than 20% on energy consumption. We summarize the key findings of this thesis in the Table 7.1.

Table 7.1: *Key Findings*

Finding 1:	On average, Hadoop 1.x releases were 30% faster than the other tested releases when running jobs with data on HDs, and 35% faster when running experiments using SSDs.
Finding 2:	Hadoop performance issues were already identified by users, but for external reasons – Hardware, Operating Systems, Java – and not for changes in the Hadoop architecture during the evolution from 1.x to 0.23.x, and to 2.x releases. Additionally, releases 0.23.x and 2.x, which contain YARN, cannot run without such resource manager. Therefore, an unnecessary waste of resources occurs when using these releases to perform MapReduce jobs.
Finding 3:	The I/O experiments showed that the use of 1.x releases promotes an energy consumption reduction ranging from 20% to 30% on average when comparing with the YARN releases.
Finding 4:	In our source code analysis, we discovered that LOC has a strong influence on Hadoop performance, but it is version number and revisions to the code that truly matter.
Finding 5:	Version awareness plays a key factor in the Hadoop project performance, as a consequence of the architectural modifications designed to accommodate the YARN resource manager.
Finding 6:	While using the $HDFS_H$, the middle 50/50 and 20/80 configurations tend to achieve results that are closer to the *SSD* configuration. This means that with half storage space coming from SSDs, we can achieve energy consumption rates closer to the use of an SSD-only HDFS.
Finding 7:	Except for the inter-branch differences, there are no significant performance gains and energy consumption reductions when running CPU-bound jobs with our approach.
Finding 8:	The performance and energy consumption of releases 1.x and 2.x were significantly improved when using the *SSDzone* to store temporary files during the execution of Hadoop jobs. Such improvements were so significant that in most cases performance was better than storing the entire dataset files in the *SSDzone*.

7.1 Original contributions

During the research, we achieved all the proposed objectives and found answers to our research questions. The original contributions of this thesis are the following.

Energy consumption reduction on most Hadoop releases. The hybrid use of HDs and SSDs reduced the energetic demands of Hadoop when running I/O-bound benchmarks. Energy consumption reduction was achieved under two scenarios:

I. When mixing the storage type, we achieve similar energetic rates for the 50/50, 20/80, and *SSD* configurations. The originality relies on the fact that the 50/50 experiments showed that it is possible to achieve performance improvements and energy saving patterns close to the *SSD* configuration. With a fraction of the SSD storage cost, similar results were achieved. We tested our approach on 6 Hadoop releases and every one promoted such reduction, some at lower rates, but most of them ranging from 10% to 20% of energy consumption reduction.

II. An alternative way of promoting performance increase was the SSD storage use as temporary HDFS space. This brought significant performance increases and presented even better energetic results compared to the hybrid approach. Furthermore, SSD price per GB is still considered high for end users and, although some studies point to a new direction, there is still the problem of SSD lifetime. Domestic users might not find this issue, but for data centers the life cycle of SSDs can still pose a problem depending on the usage rate.

This demonstrated that with our proposed approach, energy costs can be reduced, and as the price of SSDs is constantly decreasing, this option should be considered as a viable opportunity for the reduction of the TCO and of the environmental impact of data centers.

Performance increase achieved using $HDFS_H$. The origin of the energy consumption reduction is rooted on the performance increase brought by the use of SSDs in HDFS. The best benefits are achieved using only SSDs. But, as we seek for an alternative that can also be economically viable to the end user, the hybrid approach showed valuable results. Besides de major benefits of performance increases of 20% to 30%, the total storage cost was also considered, with the option of partial introduction of SSD space into HDFS. Additionally, as a bold alternative, the use of SSDs as temporary space brought performance increases of more than 50% under some experiments, which represents major enhancements for a Java-driven platform.

Development of energy profiles for Hadoop. Another original key contribution was the establishment of an energy profile for the conducted experiments. With future opportunities in mind, the energetic profiles will serve as a basis for comparison between different platforms, datasets, and even on energy predictions for experiments running on a Hadoop release. We also demonstrated that our results can serve as a basis for future developments in the Hadoop project, optimizing components to improve overall performance and reduce energy consumption, since this issue is almost unnoticed by users and developers.

With the results achieved, we also elucidated the proposed research questions. The solely use of SSDs in Hadoop would bring performance increases due to several factors, including their design using NAND flash memories and high throughput capacity, as already discussed. Therefore, **RQ1** was answered at two moments: first, showing that the hybrid approach is advantageous to Hadoop by increasing its overall performance while reducing the energy consumption on all the tested releases. Second, with the temporary SSD space which also brings significant improvements.

Regarding Hadoop performance, **RQ2** receives attention from all the experiments, since we noticed several differences among the 12 tested releases, both inside each branch and when comparing the branches themselves. Inside each branch, we noticed the effects that the evolution on source code can bring to a software. Most releases improved over time, making use of patches and fixes. But, some releases did not behave as expected during experiments under specific configurations, such as Hadoop 0.23.x releases.

The YARN resource manager was also tackled during our investigations, as it can be related – although not perceived by users and developers – to the performance decrease observed from releases 1.x on. It is interesting to note that the majority of Hadoop users are there for the MapReduce paradigm. YARN brought new features and desired flexibility for a few, but for the unaware MapReduce users, performance losses are a price unconsciously paid, especially because Hadoop

0.23.x and 2.x cannot run without YARN. With the results of this research, we expect to raise awareness regarding these issues with the Hadoop platform. A key point would be raise the awareness for the continued development of branch 1.x, with security patches and fixes, allowing a pure MapReduce platform to coexist with Hadoop YARN releases, leaving the final release selection to users.

7.2 Future Work

The results of this thesis present several future research opportunities. Energy prediction is an open research opportunity. We developed an initial energy model that, based on job makespan, dataset size, and a given $HDFS_H$ configuration, generates the energy predictions.

Table 7.2: *Definitions Used on the Prediction Model*

Symbol	Definition
$\hat{E}$	Predicted Energy
$\hat{T}$	Predicted Job Makespan
DS	Dataset Size
CFG	Storage Configuration on the $HDFS_H$

Considering Table 7.2, we designed a simple model using R to generate these predictions. First, we calculated the estimated job makespan ($\hat{T}$) using the dataset size (DS) and the chosen $HDFS_H$ configuration (CFG):

$$\hat{T} = c_0 + c_1 \cdot DS + c_2 \cdot CFG \tag{7.1}$$

With the predicted job time, we can calculate the energy prediction for the dataset:

$$\hat{E} = c_3 \cdot \hat{T} \tag{7.2}$$

where c_0 is the *intercept* and c_1, c_2, and c_3 are the coefficients of the variables generated during the predictions.

The idea is to use our results as training datasets and calculate the predictions for larger datasets. As an example, we created a dataset with all the experiments results, intentionally removing the 256GB Sort results. Then, we generated energy predictions for a 256 Sort job for the HD, 50/50, and SSD hybrid configurations. The results are shown in Figure 7.1. The predicted values show also a standard deviation interval, which was calculated based on the original values, according to the dispersion of our data. As we can see in Figure 7.1, with the simple model we used, most of the predictions were very accurate.

A refinement of this model and its prediction is the subject of ongoing research, which will include graduate students in the near future. Additionally, the model is based upon the results achieved in this thesis, and limited to the tested datasets as training sets. Additional experiments on larger datasets using $HDFS_H$ could improve the training sets, allowing more precise predictions.

Another research opportunity resulting from our findings is the further application of the proposed models. Additionally, as the Hadoop platform evolves, such models must be updated to reflect the evolution of the framework. This research should be linked with an update of the results achieved in this thesis, including the testing of new Hadoop releases, specially from branch 2.x, which had more than 12 releases after our experiments were finished. Further testing of these releases could improve our datasets, and influence the prediction model presented above.

An interesting investigation area for Hadoop is the energy profiling of Hadoop on virtual machines. Our research did not focus on virtualization for a few reasons. If we ran the experiments on a Cloud provider, e.g., Amazon Elastic Compute Cloud, we would not have complete access to energy consumption rates since VMs can run on different nodes or racks, most of them without the

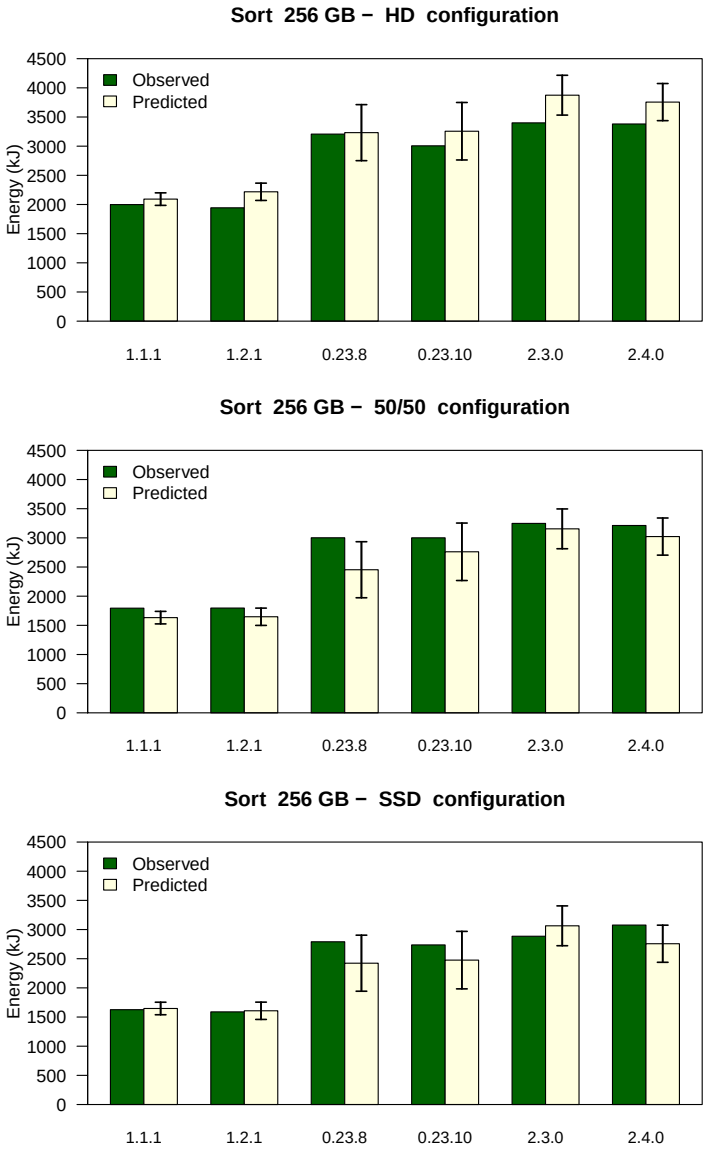


Figure 7.1: *Energy Prediction Results on Multiple $HDFS_H$ Configurations*

proper monitoring devices. Using our own infrastructure, we were able to physically monitor energy consumption on each node of the cluster. Moreover, if we focused in virtualization on our cluster, we would have had to change the energy monitoring equipment, since *WattsUp? Pro* is not capable of monitoring individual VMs. This would have generated restrictive costs for the benefits achieved. Therefore, the investigation of the Hadoop energy consumption on virtualization and simulators is an open research topic.

Finally, as cloud computing is well consolidated today, we need to start thinking of concepts such as "energy-aware services", which should be present at the top of Green Computing research themes. Virtualization and big data mining have been working together for a few years now, and new insights and research should certainly consider these topics with a focus on energy.

For the next decades, we expect to see data generation to increase exponentially. More important than gathering data will be the information extraction process. With plenty of data to analyze, it will be more important to select what data to keep and for how long it should be kept. With this velocity in generation, data already comes with an expiration date. Platforms will have to evolve to come up with solutions to such problems. Furthermore, energy consumption will be subject of

investigation and submitted to governmental regulations, since there is a general concern with the carbon footprint of the IT industry today.

During this 4-year research we saw hot computer science topics and technologies achieve maturity, such as cloud computing and solid-state drives. New topics are arising, using consolidated science and the lessons learned. At the top of the research agenda, the Internet of Things is one of the most interesting topics nowadays. It strongly relies on storage and processing facilities, corroborating even more the idea that Big Data is a reality and is here to stay. On top of it all, we consider the importance of green computing energetic research, destined to promote the energetic efficiency of the computing infrastructure.

Hopefully, in a world concerned with environmental issues, Big Data found an important allied on Green Computing, making use of its concepts to promote knowledge discovery for all mankind.

Bibliography

[ACRV12] Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan e T. N. Vijaykumar. Tarazu: optimizing MapReduce on heterogeneous clusters. *SIGARCH Computer Architecture News*, 40(1):61–74, 2012. Cited on pp. 15

[AKJ15] Jae Hoon An, Younghwan Kim e Kiman Jeon. Design and Implement of Pre-loading SSD Cache Data Using Split File on Hadoop MapReduce. In *Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems*, RACS, pages 457–460, New York, NY, USA, 2015. ACM. Cited on pp. 18

[All16] Darren Allan. The gap between SSD and hard disk prices is shrinking rapidly, 2016. Available at http://www.techradar.com/news/computing-components/storage/the-gap-between-ssd-and-hard-disk-prices-is-shrinking-rapidly-1316257 (Visited on 15/10/2016). Cited on pp. 2, 11, 53

[Apa14] Apache Mahout. Apache mahout, 2014. http://mahout.apache.org (Visited on 15/10/2016). Cited on pp. 43

[Asa14] Matt Asay. Why the world's largest hadoop installation may soon become the norm, 2014. Available at http://www.techrepublic.com/article/why-the-worlds-largest-hadoop-installation-may-soon-become-the-norm/ (Visited on 15/10/2016). Cited on pp. 60

[Bak12] K. Bakshi. Considerations for big data: Architecture and approach. In *Aerospace Conference*, pages 1–7. IEEE, march 2012. Cited on pp. 1

[BEH+10] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl e Daniel Warneke. Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st Symposium on Cloud Computing*, pages 119–130, New York, NY, USA, 2010. ACM. Cited on pp. 2

[BHBE12] Yingyi Bu, Bill Howe, Magdalena Balazinska e Michael D. Ernst. The HaLoop approach to large-scale iterative data analysis. *The VLDB Journal*, 21(2):169–190, Abril 2012. Cited on pp. 16

[Bra13] Petter Bae Brandtzæg. Big data - for better or worse, May 2013. Available at http://www.sintef.no/en/latest-news/big-data--for-better-or-worse/ (Visited on 15/10/2016). Cited on pp. 1

[BWR+11] Pramod Bhatotia, Alexander Wieder, Rodrigo Rodrigues, Umut A. Acar e Rafael Pasquin. Incoop: MapReduce for incremental computations. In *Proceedings of the 2nd Symposium on Cloud Computing*, volume 7, pages 1–14, New York, NY, USA, 2011. ACM. Cited on pp. 16

[CK94] Shyam R Chidamber e Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. Cited on pp. 36

[CLJZ15] D. Cheng, P. Lama, C. Jiang e X. Zhou. Towards energy efficiency in heterogeneous hadoop clusters by adaptive task assignment. In *IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, pages 359–368, June 2015. Cited on pp. 19

[Cou02] T. P. P. Council. TPC Benchmark H (Decision Support) Standard Specification, Jun 2002. http://www.tpc.org/tpch (Visited on 15/10/2016). Cited on pp. 43

[CZG+10] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng e Song Guo. SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In *10th International Conference on Computer and Information Technology*, pages 2736–2743. IEEE, 29 2010-july 1 2010. Cited on pp. 15

[DG04] Jeffrey Dean e Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th Conference on Operating Systems Design and Implementation*, volume 6, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association. Cited on pp. 6, 8

[DG08] Jeffrey Dean e Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Janeiro 2008. Cited on pp. vii, 9

[DPR+08] David J. DeWitt, Erik Paulson, Eric Robinson, Jeffrey Naughton, Joshua Royalty, Srinath Shankar e Andrew Krioukov. Clustera: an integrated computation and data management system. *Proceedings of the VLDB Endowment*, 1(1):28–41, Agosto 2008. Cited on pp. 2

[DQZ+10] Bo Dong, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li e Ying Li. A novel approach to improving the efficiency of storing and accessing small files on Hadoop: A case study by PowerPoint files. In *SCC'10*, pages 65–72. IEEE, july 2010. Cited on pp. 16, 17

[EER11] E. Elnikety, T. Elsayed e H.E. Ramadan. iHadoop: Asynchronous iterations for MapReduce. In *Third International Conference on Cloud Computing Technology and Science*, pages 81–90. IEEE, 29 2011-dec. 1 2011. Cited on pp. 16

[ELcF10] M. Elteir, Heshan Lin e Wu chun Feng. Enhancing MapReduce via asynchronous data processing. In *16th International Conference on Parallel and Distributed Systems*, pages 397–405. IEEE, dec. 2010. Cited on pp. 16

[ETO+11] Mohamed Y. Eltabakh, Yuanyuan Tian, Fatma Özcan, Rainer Gemulla, Aljoscha Krettek e John McPherson. CoHadoop: flexible data placement and its exploitation in Hadoop. *Proceedings of the VLDB Endowment*, 4(9):575–585, Junho 2011. Cited on pp. 17

[Fac16] Facebook. Facebook reports first quarter 2016 results, 2016. Available at http://investor.fb.com/investor-news/2016/default.aspx (Visited on 15/10/2016). Cited on pp. 1

[FRM13] E. Feller, L. Ramakrishnan e C. Morin. On the performance and energy efficiency of hadoop deployment models. In *IEEE International Conference on Big Data*, pages 131–136, Oct 2013. Cited on pp. 20

[GC12] R. Grover e M.J. Carey. Extending Map-Reduce for efficient predicate-based sampling. In *28th International Conference on Data Engineering*, pages 486–497. IEEE, april 2012. Cited on pp. 16

[GGL03] Sanjay Ghemawat, Howard Gobioff e Shun-Tak Leung. The Google File System. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003. Cited on pp. 6, 8

[GhJnBwY11] Song Guang-hua, Chuai Jun-na, Yang Bo-wei e Zheng Yao. QDFS: A quality-aware distributed file storage service based on HDFS. In *International Conference on Computer Science and Automation Engineering*, volume 2, pages 203–207. IEEE, june 2011. Cited on pp. 16

[GLN+12] Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres e Ricardo Bianchini. Greenhadoop: Leveraging green energy in data-processing frameworks. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 57–70, New York, NY, USA, 2012. ACM. Cited on pp. 19

[Goo14] Google Inc. Google cloud storage pricing, 2014. Available at http://cloud.google.com/pricing/ (Visited on 15/10/2016). Cited on pp. 10

[Hac14] Mark Hachman. SSD prices face uncertain future in 2014, 2014. Available at http://www.pcworld.com/article/2087480/ssd-prices-face-uncertain-future-in-2014.html (Visited on 15/10/2016). Cited on pp. 11, 38

[Ham10] James Hamilton. Overall data center costs, September 2010. Available at http://perspectives.mvdirona.com/2010/09/18/OverallDataCenterCosts.aspx (Visited on 15/10/2016). Cited on pp. 2, 13

[Has08] Ahmed E. Hassan. The Road Ahead for Mining Software Repositories. In *Proceedings of the Future of Software Maintenance (FoSM) at the 24th IEEE International Conference on Software Maintenance*, 2008. Cited on pp. 20

[HFL+08] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju e Tuyong Wang. Mars: A mapreduce framework on graphics processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 260–269, New York, NY, USA, 2008. ACM. Cited on pp. 60

[HHD+10] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie e Bo Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*, pages 41–51, March 2010. Cited on pp. 43, 44

[Hin12] Abram Hindle. Green Mining: A Methodology of Relating Software Change to Power Consumption. In *The 9th Working Conference on Mining Software Repositories (MSR)*, pages 78–87, 2012. Cited on pp. 20

[Hin13] Abram Hindle. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, pages 1–36, 2013. Cited on pp. 20

[HLS11] Chen He, Ying Lu e D. Swanson. Matchmaking: A new MapReduce scheduling technique. In *Third International Conference on Cloud Computing Technology and Science*, pages 40–47. IEEE, 29 2011-dec. 1 2011. Cited on pp. 15

[HRS12] M. Hammoud, M.S. Rehman e M.F. Sakr. Center-of-Gravity reduce task scheduling to lower MapReduce network traffic. In *International Conference on Cloud Computing*, pages 49–58. IEEE, june 2012. Cited on pp. 15

[HS11] M. Hammoud e M.F. Sakr. Locality-aware reduce task scheduling for MapReduce. In *Third International Conference on Cloud Computing Technology and Science*, pages 570–576. IEEE, 29 2011-dec. 1 2011. Cited on pp. 15

[HWL11] Li-Yung Ho, Jan-Jan Wu e Pangfeng Liu. Optimal algorithms for cross-rack communication optimization in MapReduce framework. In *International Conference on Cloud Computing*, pages 420–427. IEEE, july 2011. Cited on pp. 16

[HWR+14] Abram Hindle, Alex Wilson, Kent Rasmussen, Jed Barlow, Joshua Campbell e Stephen Romansky. GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework. In *The 11th Working Conference on Mining Software Repositories (MSR)*. ACM, 2014. Cited on pp. 20

[IBY+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell e Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, Março 2007. Cited on pp. 2, 60

[IJL+10] S. Ibrahim, Hai Jin, Lu Lu, Song Wu, Bingsheng He e Li Qi. LEEN: Locality/fairness-aware key partitioning for MapReduce in the cloud. In *Second International Conference on Cloud Computing Technology and Science*, pages 17–24, 30 2010-dec. 3 2010. Cited on pp. 16

[IJL+12] S. Ibrahim, Hai Jin, Lu Lu, Bingsheng He, G. Antoniu e Song Wu. Maestro: Replica-aware map scheduling for MapReduce. In *12th International Symposium on Cluster, Cloud and Grid Computing*, pages 435–442. IEEE/ACM, may 2012. Cited on pp. 15

[Int11] Intel IT. Solid-state drives and employee productivity, Jul 2011. Available at http://www.intel.com/content/dam/doc/technology-brief/intel-it-validating-reliability-of-intel-solid-state-drives-brief.pdf (Visited on 15/10/2016). Cited on pp. 11

[Int15] Intel Corporation. Micron and intel unveil new 3d nand flash memory, 2015. Available at https://newsroom.intel.com/news-releases/micron-and-intel-unveil-new-3d-nand-flash-memory/ (Visited on 15/10/2016). Cited on pp. 55

[JEMK13] Hyeran Jeon, Kaoutar El Maghraoui e Gokul B. Kandiraju. Investigating hybrid ssd ftl schemes for hadoop workloads. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, pages 20:1–20:10, New York, NY, USA, 2013. ACM. Cited on pp. 18

[JOSW10] Dawei Jiang, Beng Chin Ooi, Lei Shi e Sai Wu. The performance of MapReduce: an in-depth study. *Proceedings of the VLDB Endowment*, 3(1-2):472–483, Setembro 2010. Cited on pp. 16

[JS10] Marian Jureczko e Diomidis Spinellis. *Using Object-Oriented Design Metrics to Predict Software Defects*, volume Models and Methodology of System Dependability of *Monographs of System Dependability*, pages 69–81. Oficyna Wydawnicza Politechniki Wroclawskiej, Wroclaw, Poland, 2010. Cited on pp. 35, 36

[KAB14] K. R. Krish, A. Anwar e A. R. Butt. hats: A heterogeneity-aware tiered storage for hadoop. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 502–511, May 2014. Cited on pp. 18

[KAEN11] Rini T. Kaushik, Tarek Abdelzaher, Ryota Egashira e Klara Nahrstedt. Predictive data and energy management in greenhdfs. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pages 1–9, Washington, DC, USA, 2011. IEEE Computer Society. Cited on pp. 19

[KB10] Rini T. Kaushik e Milind Bhandarkar. Greenhdfs: Towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pages 1–9, Berkeley, CA, USA, 2010. USENIX Association. Cited on pp. 19

[KBHR12] YongChul Kwon, Magdalena Balazinska, Bill Howe e Jerome Rolia. SkewTune: mitigating skew in mapreduce applications. In *Proceedings of the International Conference on Management of Data*, pages 25–36, New York, NY, USA, 2012. ACM. Cited on pp. 16

[KC14] Karthik Kambatla e Yanpei Chen. The truth about mapreduce performance on ssds. In *28th Large Installation System Administration Conference (LISA14)*, pages 118–126, Seattle, WA, Novembro 2014. USENIX Association. Cited on pp. 18

[KIB14] K.R. Krish, M.S. Iqbal e A.R. Butt. VENU: Orchestrating SSDs in hadoop storage. In *IEEE International Conference on Big Data (Big Data)*, pages 207–212, Oct 2014. Cited on pp. 18

[Kim13] Eden Kim. SSD Performance - A Primer. An Introduction to Solid State Drive Performance, Evaluation and Test. Relatório técnico, Storage Networking Industry Association, August 2013. Available at http://www.snia.org/sites/default/files/SNIASSSI.SSDPerformance-APrimer2013.pdf (Visited on 15/10/2016). Cited on pp. vii, 11, 12

[Kim15] Eden Kim. Understanding SSD performance project. Relatório técnico, Storage Networking Industry Association, August 2015. Available at http://www.snia.org/forums/sssi/pts (Visited on 15/10/2016). Cited on pp. vii, 12

[KKVR12] K. Arun Kumar, Vamshi Krishna Konishetty, Kaladhar Voruganti e G. V. Prabhakara Rao. CASH: Context Aware Scheduler for Hadoop. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 52–61, New York, NY, USA, 2012. ACM. Cited on pp. 15

[Kom10] Matthew Komorowski. A history of storage cost, 2010. Available at http://www.mkomo.com/cost-per-gigabyte (Visited on 15/10/2016). Cited on pp. 1

[KsKMP13] Yangwook Kang, Yang suk Kee, E.L. Miller e Chanik Park. Enabling cost-effective data processing with smart ssd. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, pages 1–12, 2013. Cited on pp. 18

[KVV11] G. Kousiouris, G. Vafiadis e T. Varvarigou. A front-end, Hadoop-based data management service for efficient federated clouds. In *Third International Conference on Cloud Computing Technology and Science*, pages 511–516. IEEE, 29 2011-dec. 1 2011. Cited on pp. 16

[Lan01] Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Relatório técnico, META Group, February 2001. Available at http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf (Visited on 15/10/2016). Cited on pp. 1

[LAY11] Shen Li, T. Abdelzaher e Mindi Yuan. TAPA: Temperature aware power allocation in data center with Map-Reduce. In *International Green Computing Conference and Workshops*, pages 1–8, july 2011. Cited on pp. 16, 19

[Lei13] Luke Kenneth Casson Leighton. Analysis of ssd reliability during power-outages, December 2013. Available at http://lkcl.net/reports/ssd_analysis.html (Visited on 15/10/2016). Cited on pp. 11

[LMA+10] Heshan Lin, Xiaosong Ma, Jeremy Archuleta, Wu-chun Feng, Mark Gardner e Zhe Zhang. MOON: MapReduce On Opportunistic eNvironments. In *Proceedings of the 19th International Symposium on High Performance Distributed Computing*, pages 95–106, New York, NY, USA, 2010. ACM. Cited on pp. 16

[LP10] Willis Lang e Jignesh M. Patel. Energy management for MapReduce clusters. *Proceedings of the VLDB Endowment*, 3(1-2):129–139, 2010. Cited on pp. 19

[LWH11] Lei Lei, Tianyu Wo e Chunming Hu. CREST: Towards fast speculation of straggler tasks in MapReduce. In *8th International Conference on e-Business Engineering*, pages 311–316. IEEE, oct. 2011. Cited on pp. 15

[LZZ12] Nikolay Laptev, Kai Zeng e Carlo Zaniolo. Early accurate results for advanced analytics on MapReduce. *Proceedings of the VLDB Endowment*, 5(10):1028–1039, Junho 2012. Cited on pp. 16

[MAB+10] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser e Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the International Conference on Management of Data*, pages 135–146, New York, NY, USA, 2010. ACM. Cited on pp. 2

[Mac14] John C. MacCallum. Disk drive prices, 2014. Available at http://www.jcmit.com/diskprice.htm (Visited on 15/10/2016). Cited on pp. 38

[Mea15] Lucas Mearian. SSDs will be in more than 25% of new laptops this year, more than 40% by 2017, 2015. Available at http://www.computerworld.com/article/3010395/solid-state-drives/consumer-ssds-and-hard-drive-prices-are-nearing-parity.html (Visited on 15/10/2016). Cited on pp. 2, 11, 53

[Mea16] Lucas Mearian. SSD prices plummet again, close in on HDDs, 2016. Available at http://www.computerworld.com/article/3040694/data-storage/ssd-prices-plummet-again-close-in-on-hdds.html (Visited on 15/10/2016). Cited on pp. 2

[MEMRB10] Esteban Molina-Estolano, Carlos Maltzahn, Ben Reed e Scott A. Brandt. Haceph: Scalable metadata management for hadoop using ceph, 2010. Cited on pp. 17

[MLK14] Sangwhan Moon, Jaehwan Lee e Yang Suk Kee. Introducing SSDs to the hadoop mapreduce framework. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 272–279, June 2014. Cited on pp. 18

[MOT11] Shunsuke Mikami, Kazuki Ohta e Osamu Tatebe. Using the Gfarm File System as a POSIX compatible storage platform for Hadoop MapReduce applications. In *Proceedings of the 12th International Conference on Grid Computing*, pages 181–189, Washington, DC, USA, 2011. IEEE/ACM. Cited on pp. 16, 17

[MTF11] Prashanth Mundkur, Ville Tuulos e Jared Flatow. Disco: A computing platform for large-scale data analytics. In *Proceedings of the 10th ACM SIGPLAN Workshop on Erlang*, Erlang '11, pages 84–89, New York, NY, USA, 2011. ACM. Cited on pp. 60

[Mur08] S. Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10(1):24–33, Jan 2008. Cited on pp. 2, 13

[MVE+14] Arun C. Murthy, Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec e Jeff Markham. *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2.* Addison-Wesley Professional, 2014. Cited on pp. 21

[MWZL12] Yaokuan Mao, Wenjun Wu, Hui Zhang e Liang Luo. GreenPipe: A Hadoop based workflow system on energy-efficient clouds. In *26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 2211–2219. IEEE, may 2012. IEEE. Cited on pp. 20

[NB05] Nachiappan Nagappan e Thomas Ball. Use of relative code churn measures to predict system defect density. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005. Cited on pp. 20

[NPM+10] Tomasz Nykiel, Michalis Potamias, Chaitanya Mishra, George Kollios e Nick Koudas. MRShare: sharing across multiple queries in MapReduce. *Proceedings of the VLDB Endowment*, 3(1-2):494–505, Setembro 2010. Cited on pp. 15

[O'M11] Conor O'Mahony. Comparing hdfs and gpfs for hadoop, Nov 2011. Available at http://blog.intelligencecomputing.io/twitter/7653/repost-comparing-hdfs-and-gpfs-for-hadoop (Visited on 15/10/2016). Cited on pp. 17

[PIAB15] T. D. Phan, S. Ibrahim, G. Antoniu e L. Bougé. On understanding the energy impact of speculative execution in hadoop. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pages 396–403, Dec 2015. Cited on pp. 19

[PRGK14] Ivanilton Polato, Reginaldo Ré, Alfredo Goldman e Fabio Kon. A comprehensive view of Hadoop research - A systematic literature review. *Journal of Network and Computer Applications*, 46:1 – 25, 2014. Cited on pp. 14, 22

[PWB07] Eduardo Pinheiro, Wolf-Dietrich Weber e Luiz André Barroso. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, Berkeley, CA, USA, 2007. USENIX Association. Cited on pp. 11

[PYXH14] Fengfeng Pan, Yinliang Yue, Jin Xiong e Daxiang Hao. I/O characterization of big data workloads in data centers. In Jianfeng Zhan, Rui Han e Chuliang Weng, editors, *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, volume 8807 of *Lecture Notes in Computer Science*, pages 85–97. Springer International Publishing, 2014. Cited on pp. 44

[RD11] Aysan Rasooli e Douglas G. Down. An adaptive scheduling algorithm for dynamic heterogeneous Hadoop systems. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*, pages 30–44, Riverton, NJ, USA, 2011. IBM Corp. Cited on pp. 15

[Ren14] Thomas M. Rent. Origin of solid state drives, 2014. Available at http://www.storagereview.com/origin_solid_state_drives (Visited on 15/10/2016). Cited on pp. 11

[Rut11] Nathan Rutman. Map/reduce on lustre, February 2011. Available at http://www.xyratex.com/sites/default/files/Xyratex_white_paper_MapReduce_1-4.pdf (Visited on 15/10/2016). Cited on pp. 17

[SAD+10] Michael Stonebraker, Daniel Abadi, David J. DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo e Alexander Rasin. MapReduce and parallel DBMSs: friends or foes? *Communications of the ACM*, 53(1):64–71, Janeiro 2010. Cited on pp. 1

[SG07] Bianca Schroeder e Garth A. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, Berkeley, CA, USA, 2007. USENIX Association. Cited on pp. 11

[SK14] P. Saxena e P. Kumar. Performance evaluation of hdd and ssd on 10gige, ipoib amp; rdma-ib with hadoop cluster performance benchmarking system. In *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference* -, pages 30–35, Sept 2014. Cited on pp. 18

[SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia e Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies*, pages 1–10, Washington, DC, USA, 2010. IEEE. Cited on pp. 6, 9, 16

[SLT11] Lei Shi, Xiaohui Li e Kian-Lee Tan. S3: An efficient Shared Scan Scheduler on MapReduce framework. In *International Conference on Parallel Processing*, pages 325–334, sept. 2011. Cited on pp. 15

[Smi12] Ivan Smith. Cost of hard drive storage space, 2012. Available at http://ns1758.ca/winch/winchest.html (Visited on 15/10/2016). Cited on pp. 1

[Spi05] Diomidis Spinellis. Tool writing: A forgotten art? *IEEE Software*, 22(4):9–11, July/August 2005. Cited on pp. 20

[SRC10] J. Shafer, S. Rixner e A.L. Cox. The Hadoop Distributed Filesystem: Balancing portability and performance. In *International Symposium on Performance Analysis of Systems Software*, pages 122–133. IEEE, march 2010. IEEE. Cited on pp. 16

[Sta13] Matt Stansberry. Data Center Industry Survey 2013. Relatório técnico, Uptime Institute, 2013. Cited on pp. 1

[TFL14] W. Tan, L. Fong e Y. Liu. Effectiveness assessment of solid-state drive used in big data services. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 393–400, June 2014. Cited on pp. 18

[Tom11] Tom's Hardware. Investigation: Is your ssd more reliable than a hard drive?, 2011. Available at http://www.tomshardware.com/reviews/ssd-reliability-failure-rate,2923.html (Visited on 15/10/2016). Cited on pp. 11

[Tom13] Tom's Hardware. Performance charts hard drives and ssds, 2013. Available at http://www.tomshardware.com/charts/hard-drives-and-ssds,3.html (Visited on 15/10/2016). Cited on pp. 11, 13

[Twi16] Twitter Inc. Investor relations: Quarter results, 2016. Available at https://investor.twitterinc.com/results.cfm (Visited on 15/10/2016). Cited on pp. 1

[TZHZ09] Chao Tian, Haojie Zhou, Yongqiang He e Li Zha. A dynamic MapReduce scheduler for heterogeneous workloads. In *8th International Conference on Grid and Cooperative Computing*, pages 218–224, aug. 2009. Cited on pp. 15

[TZSC11] Yongcai Tao, Qing Zhang, Lei Shi e Pinhua Chen. Job scheduling optimization for multi-user MapReduce clusters. In *4th International Symposium on Parallel Architectures, Algorithms and Programming*, pages 213–217, dec. 2011. IEEE. Cited on pp. 15

[VBBE12] Rares Vernica, Andrey Balmin, Kevin S. Beyer e Vuk Ercegovac. Adaptive MapReduce using situation-aware mappers. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 420–431, New York, NY, USA, 2012. ACM. Cited on pp. 16

[VCC11] Abhishek Verma, Ludmila Cherkasova e Roy H. Campbell. ARIA: automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th International Conference on Autonomic Computing*, pages 235–244, New York, NY, USA, 2011. ACM. Cited on pp. 16

[VCKC12] A. Verma, L. Cherkasova, V.S. Kumar e R.H. Campbell. Deadline-based workload management for MapReduce environments: Pieces of the performance puzzle. In *Network Operations and Management Symposium*, pages 900–905. IEEE, 2012. Cited on pp. 16

[VNOS12] Dan Vesset, Ashish Nadkarni, Carl W. Olofson e David Schubmehl. Worldwide big data technology and services 2012-2016 forecast. Relatório técnico, IDC Corporate USA, 2012. Cited on pp. 1

[Whi12] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 3rd edição, 2012. Cited on pp. 1

[WJY14] Bo Wang, Jinlei Jiang e Guangwen Yang. mpCache: Accelerating mapreduce with hybrid storage system on many-core clusters. In Ching-Hsien Hsu, Xuanhua Shi e Valentina Salapura, editors, *Network and Parallel Computing*, volume 8707 of *Lecture Notes in Computer Science*, pages 220–233. Springer Berlin Heidelberg, 2014. Cited on pp. 18

[WLX+13] D. Wu, W. Luo, W. Xie, X. Ji, J. He e D. Wu. Understanding the impacts of solid-state storage on the hadoop performance. In *Advanced Cloud and Big Data (CBD), 2013 International Conference on*, pages 125–130, Dec 2013. Cited on pp. 18

[WQY+11] Yandong Wang, Xinyu Que, Weikuan Yu, Dror Goldenberg e Dhiraj Sehgal. Hadoop acceleration through network levitated merge. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, volume 57, pages 1–10, New York, NY, USA, 2011. ACM. Cited on pp. 16

[XYR+10] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares e Xiao Qin. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 1–9. IEEE, april 2010. IEEE. Cited on pp. 16, 17

[YVM12] S. A. Yazd, S. Venkatesan e N. Mittal. Energy efficient hadoop using mirrored data block replication policy. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 457–462, Oct 2012. Cited on pp. 19

[YYH11] Hsin-Han You, Chun-Chung Yang e Jiun-Long Huang. A load-aware scheduler for MapReduce framework in heterogeneous cloud environments. In *Proceedings of the Symposium on Applied Computing*, pages 127–132, New York, NY, USA, 2011. ACM. ACM. Cited on pp. 15

[ZBSS+10] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker e Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, pages 265–278, New York, NY, USA, 2010. ACM. Cited on pp. 15

[ZC11] Hao Zhu e Haopeng Chen. Adaptive failure detection via heartbeat under Hadoop. In *Asia-Pacific Services Computing Conference*, pages 231–238. IEEE, dec. 2011. Cited on pp. 16

[ZE11] P. Zikopoulos e C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. Mcgraw-hill, 2011. Cited on pp. 1

[ZFF+11] Xiaohong Zhang, Yuhong Feng, Shengzhong Feng, Jianping Fan e Zhong Ming. An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In *Proceedings of the International Conference on*

Cloud and Service Computing, pages 235–242, Washington, DC, USA, 2011. IEEE. Cited on pp. 15

[ZGGW11] Yanfeng Zhang, Qinxin Gao, Lixin Gao e Cuirong Wang. iMapReduce: A distributed computing framework for iterative computation. In *International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1112–1121. IEEE, may 2011. IEEE. Cited on pp. 16

[ZKJ+08] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz e Ion Stoica. Improving MapReduce performance in heterogeneous environments. In *Proceedings of the 8th Conference on Operating Systems Design and Implementation*, volume 8, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association. Cited on pp. 15

[ZWM+12] Yanrong Zhao, Weiping Wang, Dan Meng, YongChun Lv, Shubin Zhang e Jun Li. TDWS: A job scheduling algorithm based on MapReduce. In *7th International Conference on Networking, Architecture and Storage*, pages 313–319. IEEE, june 2012. Cited on pp. 15

[ZWYD12] Xiaohong Zhang, Guowei Wang, Zijing Yang e Yang Ding. A two-phase execution engine of reduce tasks in Hadoop MapReduce. In *International Conference on Systems and Informatics*, pages 858–864, may 2012. Cited on pp. 15

[ZYLL11] Jiaran Zhang, Xiaohui Yu, You Li e Liwei Lin. HadoopRsync. In *International Conference on Cloud and Service Computing*, pages 166–173, dec. 2011. Cited on pp. 16

[ZZF+11] Xiaohong Zhang, Zhiyong Zhong, Shengzhong Feng, B. Tu e Jianping Fan. Improving data locality of MapReduce by scheduling in homogeneous computing environments. In *9th International Symposium on Parallel and Distributed Processing with Applications*, pages 120–126. IEEE, may 2011. IEEE. Cited on pp. 15