Hybrid HDFS: decreasing energy consumption and speeding up Hadoop using SSDs

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 being to address sustainability and limit power consumption. In this paper, 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 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 the Hadoop project, representing different designs of the Hadoop middleware. 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.

Hybrid HDFS: Decreasing Energy Consumption and Speeding up Hadoop using SSDs

Ivanilton Polato
Dept. of Computer Science
Federal University of Technology – Paraná
Campo Mourão, PR, Brazil
ipolato@utfpr.edu.br

Fabio Kon
Dept. of Computer Science
University of São Paulo
São Paulo, SP, Brazil
fabio.kon@ime.usp.br

Denilson Barbosa and Abram Hindle
Dept. of Computer Science
University of Alberta
Edmonton, AB, Canada
{denilson,abram.hindle}@ualberta.ca

ABSTRACT

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 being to address sustainability and limit power consumption. In this paper, 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 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 the Hadoop project, representing different designs of the Hadoop middleware. 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.

1. INTRODUCTION

Nowadays, two perspectives are relevant for big data analysis: the 3 "Vs": volume, variety, and velocity [21, 36]; and hardware and software infrastructures capable of processing all the collected data. These processing infrastructures now encounter new performance challenges: energy consumption, power usage, and environmental impact. Over the last years, the volume and speed of data creation consistently increased. A recent study estimates that 90% of all data in the world was generated over the last two years [3]. 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 [31]. 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% until 2016, which represents an 8 billion-dollar business [31].

In terms of infrastructure, the storage server market has benefited from continually decreasing magnetic spinning hard disk prices and higher performance solid state drives. Companies can store data at a relatively low cost per GB and create new data centers at affordable prices. Over the last three years, the cost for hard disks has ranged between $0.03 and $0.05 per GB [22], with specialized providers offering ever lower costs for hard disks bundled with services. SSDs are faster, but cost around 20 times or more per GB.

From this perspective, the cost of SSDs could still be considered prohibitive for general storage purposes. However, they can provide unique performance enhancements to data analysis by reducing energy consumption if they receive adequate support from processing platforms. In recent years, a number of frameworks have been released to support parallel and distributed computing, and provide high level solutions to end-users [2, 6, 14, 23]. Some of these frameworks were built over well-known programming models, such as MPI and MapReduce. Nevertheless, these frameworks do not sufficiently acknowledge the coexistence of HDs and SSDs in the same environment, and therefore are not tailored to benefit from this possibility.

Within this context, this paper presents a hybrid storage model for the Hadoop Distributed File System (HDFS), called $HDFS_H$, which seamless 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 the 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 storage device should be used during MapReduce computations. The observations made during experiments may also be used as a guide for users seeking to modify existing Hadoop clusters, or even put together a new cluster. The key original contributions of this paper are:

- **A novel hybrid storage model for HDFS that takes into account the performance profiles of HDs and SSDs.** Since SSDs are faster and use less power than HDs but are much more expensive, these characteristics must be adequately treated for different workloads.
- **An HD- and SSD-aware block placement policy that is optimized for heterogeneous storage.** This policy's rules were designed to take advantage of the differences between HDs and SSDs by accommodating a pre-defined percentage of the total number of blocks in the SSDs, and the remainder in the HDs. Additionally, we also present the results of an experiment in directing temporary files to a specific storage zone.
- **Evaluation of the technique over multiple versions of Apache Hadoop.** Our research showed that each Hadoop branch presents a unique energy consumption profile; we detail these results and show how our system behaves in each situation.

Our motivation relies on an inevitable side-effect of the data analysis field's expansion: 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 [12].

Despite the fact that it may be easy and relatively cheap to store data today, the information extraction process remains a bottleneck for IT companies. A large body of research concerns how to perform computations on large scale datasets within acceptable time limits. As a result, great effort is put into frameworks that help developers leverage numerous available computing resources, providing support for data integrity, replication, load balancing, task scheduling, scalability, and failure recovery.

The MapReduce paradigm [5, 8], through its open source middleware implementation, Apache Hadoop, comprises such an approach to processing large datasets. The map function generally filters and sorts the input data, while the reduce functions are responsible for summarizing the results. Popular for its ability to achieve high computing power by using commodity hardware in large-scale clusters, Hadoop is a relatively low-cost way to obtain an infrastructure capable of carrying out massive computations with a high level of parallelism. The Hadoop middleware framework has three major components: the *Hadoop Distributed File System* (HDFS), a block file storage, designed to reliably hold very large datasets using data replication [30]; *MapReduce*, a system for parallel processing of large data sets using the homonymous programming paradigm; and the more recent component, the *YARN* scheduler and resource manager, which is included in Hadoop 0.23.0 and 2.0.0.

Using MapReduce over HDFS, the middleware has rapidly evolved over the last years, promoting transparent data integrity, replication, scalability, and failure recovery features that have made Apache Hadoop very popular both in academia and industry. The platform popularization over the last five years has benefitted from several concepts and technologies that consolidated in the same period, including, e.g., the use of Cloud Computing to achieve scalability, availability, and flexibility in data processing [27].

Despite Hadoop's popularity, it still struggles to properly incorporate certain features. For example, Hadoop typically lacks features that support heterogeneous environments. In its storage layer, Hadoop uniformly treats diverse storage devices. Thus, even it supports the use of SSDs, the HDFS does not acknowledge any differences between HDs and SSDs.

The remainder of this paper is organized as follows. Section 2 presents an overview of Apache Hadoop, including its history and components. Section 3 introduces our hybrid storage, the block placement models, and a cost model. Section 4 delineates the experimentation methodology and the infrastructure we used for testing. Section 5 presents and analyzes the results. Section 6 discusses related work. Finally, Section 7 presents our conclusions and future research directions.

2. APACHE HADOOP AND HDFS

The MapReduce programming model, now highly used in the Big Data context, is not new. One of its first uses was on the LISP programming language, which later inspired Google's approach [5, pp. 1]. Google's MapReduce was initially composed of the GFS distributed filesystem [8] and an implementation of MapReduce [5]. Hadoop was developed based on this parallel approach, using the same idea from Google's implementation: hiding complexity from users and thereby allowing them to focus on programming the paradigm's two primitive functions, Map and Reduce. To move computation towards data, Hadoop uses the HDFS file system. HDFS [30] is a block direct storage system capable of storing, managing, and streaming large amounts of data in a reasonable time to user applications. As mentioned earlier, HDFS lacks differentiation of the different storage devices attached to a Hadoop cluster node; consequently, it cannot properly exploit the features provided by such devices to customizably increase job performance or decrease a cluster's energy consumption. Our approach tackles this specific issue, creating storage zones according to the device types connected to the cluster nodes. To the best of our knowledge, this is a novel approach to represent and manage storage space for Hadoop's MapReduce computations.

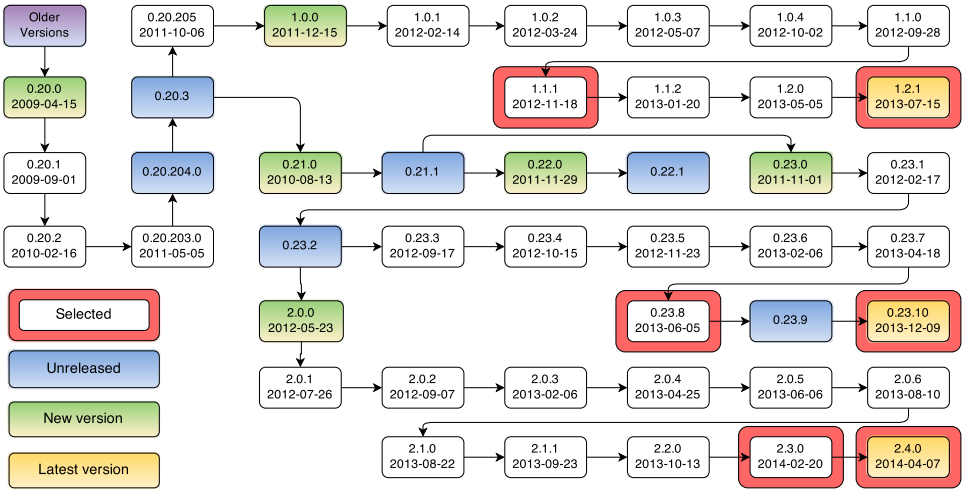

Figure 1: Hadoop Versions Genealogy Tree

2.1 Brief Hadoop History

Throughout its history, Hadoop experienced more than 60 releases in several development branches. As of now, Hadoop has three main development branches: 1.x, 0.23.x and 2.x. Our research focused on each of these branches' recent releases. We observed that most of the Hadoop R&D works were developed using older versions ranging from 0.19.x to 0.22.x [27]. We also observed that the number of studies contributing to the project increased from 2010 to 2012, but decreased in 2013, meaning that fewer studies focused on Hadoop's new releases, since they were launched in late 2011 (version 0.23.0 on 2011-11-01; version 1.0.0 on 2011-12-27) and in the second quarter of 2012 (version 2.0.0 on 2012-05-23) [11]. Additionally, versions 0.23.x and 2.x adopted a new architecture, introducing the YARN resource manager. Thus, we suspect that the new releases' instability in the YARN branches may have directly contributed to the low number of studies using such versions. We built a genealogy tree from the Hadoop project using release logs from each project branch and its releases. Figure 1 shows a part of the Hadoop project genealogy tree, from version 0.20.0 up to the latest releases, including the release dates.

The versions we selected for our experiments include the Hadoop project's three current branches: 1.x, 0.23.x and 2.x. The 0.23.x and 2.x branches include the YARN resource manager. The difference between them is that 2.x releases include the High Available NameNode for HDFS – an effort focused on the automatic failover of the NameNode – whereas the 0.23.x releases exclude such a feature. The 1.x Hadoop releases do not include the YARN feature and have the limitation that they are more tightly coupled to the MapReduce paradigm and mostly designed to run batch jobs, making them less flexible than the other two branches. Table 1 lists the selected group of releases. With these selections we can overview almost two years of the project's releases, including different versions and branches, representing different Hadoop middleware designs.

Table 1: 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

3. HDFS HYBRID STORAGE

We developed a hybrid storage approach for HDFS that leverages the different characteristics of HDs and SSDs connected to a Hadoop cluster. This approach's key feature is the controlled use of SSDs to increase performance and reduce energy consumption. Yet, although these two SSDs' characteristics are outstanding, we must also consider their price. The cost per GB of the SSDs sits between $1.00 and $1.50, with expectations it will decrease to less than $1.00 per GB in the near future [10]. Even reaching this cost per GB, it still represents around 20 times or more the cost per GB of HDs. From this perspective, the cost of SSDs could still be considered prohibitive for general use as the single storage option. Recently, a newly developed hybrid storage device, known as SSHD, combines in the same device both SSD and HD technologies. Generally, these devices can improve performance by storing the most frequently accessed data in their NAND flash memory.

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. The observations made during experiments may also be used as a guide for users seeking to modify existing Hadoop clusters, or even put together a new cluster. 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. In our model, we limited the storage devices in the cluster nodes to HDs and SSDs, excluding any SSHDs,

since they do not increase performance on read/write operations compared to SSDs. Table 2 contains a glossary of symbols and functions used in our storage model.

Table 2: Definitions used on the HDFS Hybrid Storage Model

Symbol	Definition
HD	Hard Disk
SSD	Solid-State Disk
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`

First, we define the HDFS storage space as the sum of the available spaces on all DataNodes in the cluster. We use 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:

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

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

$$HDspace(DN) = \sum_{i=1}^{z} SpaceAvailableInDevice(HD_i) \quad (2)$$

$$SSDspace(DN) = \sum_{i=1}^{w} SpaceAvailableInDevice(SSD_i) \quad (3)$$

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:

$$HDzone = \sum_{i=1}^{d} HDspace(DN_i) \quad (4)$$

$$SSDzone = \sum_{i=1}^{d} SSDspace(DN_i) \quad (5)$$

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:

$$HDFS_H = HDzone + SSDzone \quad (6)$$

Our storage model aims to capture 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 NameNode maintains a general view of the HDFS storage; and also can separately access the devices from the block placement policies.

3.1 Block Placement Policies

Following our HDFS hybrid storage model, we created the 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 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 energy consumption decreases. Our first block placement policy can send 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. Using 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:

$$blocks(file) = \lceil size(file)/blockSize \rceil \quad (7)$$

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

$$blocksDS = \sum_{f=1}^{k} blocks(file_f) \quad (8)$$

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:

$$HDFS_H \leftarrow DS \begin{cases} SSDzone \leftarrow \lceil \rho.blocksDS \rceil \\ HDzone \leftarrow \lfloor (1-\rho).blocksDS \rfloor \end{cases} \quad (9)$$

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*.

In all of the experiments, we configured the *HDzone* to hold the temporary files during job execution. To explore additional possibilities, we performed experiments using the *SSDzone* to store Hadoop temporary files. These experiments aim to show added benefits of including SSD storage space into a Hadoop cluster.

3.2 Cost Model

Our storage model splits the dataset blocks in the HDFS into different storage zones using HD and SSD devices. Given that all files in the $HDFS_H$ are stored either in the *HDzone* or in the *SSDzone*, the block proportion for each zone is complimentary. Thus, considering a Hadoop job we have:

$$SSD_{prop} + HD_{prop} = 1 \quad (10)$$

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

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

$$HDFS_H\, StorageCost = size(DS) \times (SSD_C + HD_C) \quad (11)$$

where:

$$\begin{aligned} SSD_C &= SSD_{prop} \times SSD_{costPerGB} \\ HD_C &= HD_{prop} \times HD_{costPerGB} \end{aligned} \quad (12)$$

using Equation 10 we have:

$$HD_C = (1 - SSD_{prop}) \times HD_{costPerGB} \quad (13)$$

Thus, we can estimate the storage cost for a Hadoop job using $HDFS_H$ by setting the SSD proportion, and the SSDs' and HDs' cost per GB.

4. EXPERIMENTAL METHODOLOGY AND INFRASTRUCTURE

With $HDFS_H$ and the block placement rules properly designed, we then selected the Hadoop releases and benchmarks for the experiments. Hadoop comes with a large set of examples and tools that allow benchmarking in several ways. Some of them use MapReduce applications, such as Word Count, Sort, Terasort, and Join. Another relevant benchmark set is HiBench [13], which is publicly available on GitHub and developed by Intel, and includes machine learning and data analytics benchmarks, as well as the Hadoop project's stock benchmarks. To cover an array of different situations, our final selection of benchmarks includes Sort and Join from Hadoop and the Mahout K-Means clustering from HiBench. Table 3 presents our set of benchmarks and their corresponding dataset size. Although our approach focuses on storage, we also performed experiments using CPU-bound benchmarks to analyze the behavior of the hybrid storage under these workloads.

Table 3: Benchmarks and Dataset Sizes used in the experiments

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

Each benchmark uses a specific dataset that was generated and stored for experimental reuse and replication. For the Sort experiments, the datasets were generated 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 [4], which is widely used by the database community. Finally, we used an implementation of the K-Means clustering algorithm using the Mahout Library [1] from HiBench in our experiments. K-means considers a 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 [13, 26].

Table 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%

Our testing infrastructure limited dataset sizes. Since our cluster has eight DataNodes with two disks – one HD (1TB) and one SSD (120GB) – the *SSDzone* total size, which was 960GB, limited our datasets. Additionally, after each executions the Sort benchmark doubles the used file system space. Therefore, the 256GB dataset benchmarks cannot be performed only using the SSDs, and the experiments were run both on the HD only and hybrid configurations. For each benchmark, we ran a batch job as follows: for small datasets, a 30-job batch; and for medium and large ones, a 5-job batch. Each selected Hadoop release ran one batch for each benchmark/dataset pair. The 10GB and 48GB Sort experiments, the Join, and the K-Means experiments ran with a cluster configuration using 4 nodes, whereas the 256GB Sort ran in the full cluster configuration. We used this approach to check the framework's behavior under different

configurations, and to check the cluster's energy consumption. To record the power measurements during the experiments, we deployed an additional daemon on the head node gathering the data from each power meter, and stored the power readings from each node in separate files. We configured this daemon to record one reading every second from each power meter.

To illustrate the use of our block placement policy, we set five predefined proportions to split the data into the storage zones. The first one keeps all the data in the *HDzone*, and does not use the *SSDzone*. The three intermediate ones vary the amount of data stored in each zone, as Table 4 shows. The last one uses only the *SSDzone* to store the data.

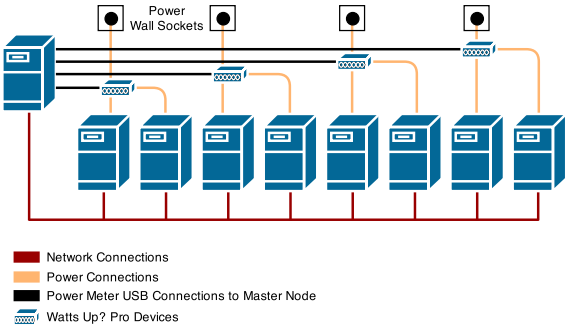

Figure 2: Cluster Infrastructure

The testing infrastructure is a 9-node commodity cluster. Each node has a quad-core processor (AMD A8-5600K; 3600 MHz), 8GB of RAM, a 1TB hard disk (Western Digital WD Black WD1002FAEX; 7200RPM; 64MB Cache; SATA 6.0Gb/s), and a 120GB solid-state disk (Intel SSDSC2BW120A4; SATA 6Gb/s; 20nm MLC). All nodes run the Red Hat (4.4.7-4) GNU/Linux operating system. One of the nodes (head) exclusively runs the Hadoop NameNode and JobTracker daemons, which also keeps the job history log. The other eight nodes run Hadoop DataNodes and TaskTracker daemons. For the energy consumption measurements, we instrumented the cluster with *Watts Up? Pro*, a hardware device that measures wall socket power use and reports power measures every second containing watts, kWh, voltage, amps,

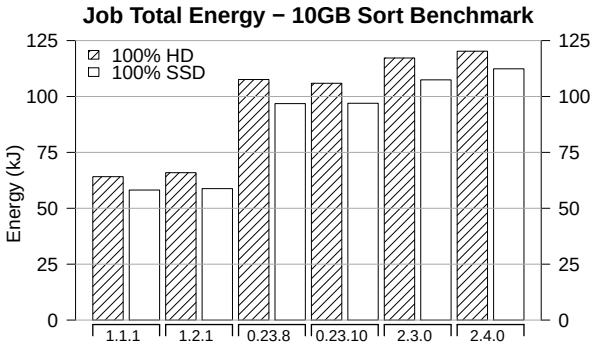

Figure 3: Average Energy Consumption

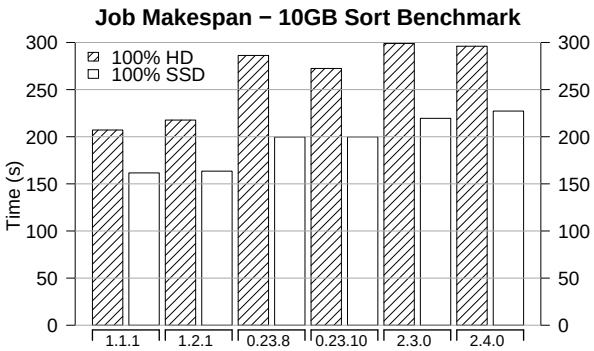

Figure 4: Average Job Makespan

power-factor, and other information. We connected the DataNodes in pairs on each one of the four power meter devices. Since we were concerned with power measurements on hybrid storage, we did not instrument the head node, since it is responsible for coordinating Hadoop and does not store any of the HDFS block files. We present the results and analysis next. Figure 2 illustrates our hardware infrastructure.

5. EXPERIMENTAL RESULTS AND ANALYSIS

We were concerned with the energy consumption impact of hybrid storage system. Our first finding was the large difference among the multiple Hadoop releases. Due to architectural changes in the middleware, Hadoop 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 experimental results. In terms of performance, we consider makespan as the time difference between the start and finish of Hadoop jobs.

5.1 I/O-Bound Benchmark Results

Starting with the 10GB Sort benchmark, Figure 3 shows the differences between two $HDFS_H$ configurations: *HD* and *SSD*. The differences among the three branches are easily identified. 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 1.1.1 and 1.2.1 releases.

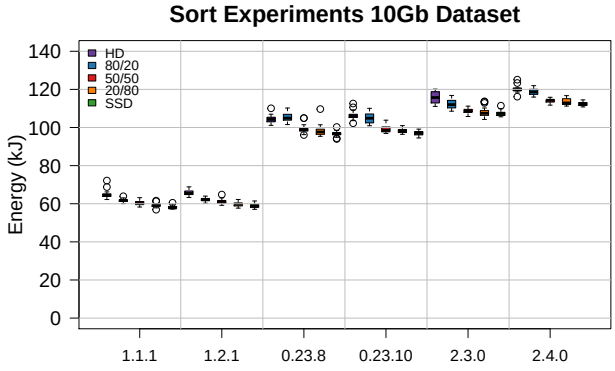

Figure 5: Energy Consumption: Sort 10GB

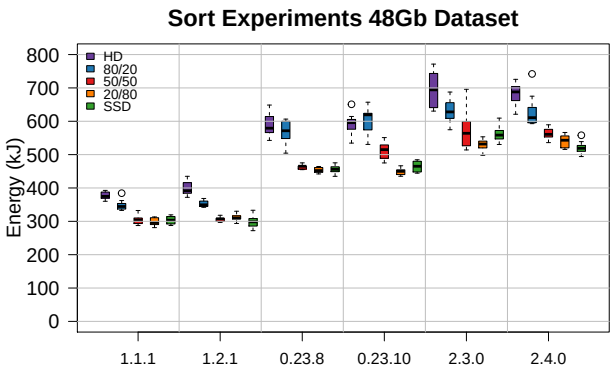

Figure 6: Energy Consumption: Sort 48GB

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 4 illustrates. The significant difference in energy consumption can also be explained by recent Hadoop branches' increase in the number of included features. 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 Figures 3 and 4, we can also observe that there are small differences between releases from the same branch, with small energy consumption variations.

Further considering the 10GB Sort experiments, Figure 5 presents the results for all the releases using the five configurations we tested, and in the following order: *HD*, 80/20, 50/50, 20/80, and *SSD*. We can also observe, in Figure 5, the expected tendency in energy savings when moving data to the configurations that favor SSD use.

Table 5: Sort Benchmarks: Energy consumed (kJ)

Dataset size	Release	HD	80/20	50/50	20/80	SSD
10GB	1.1.1	64	62	60	59	58
	1.2.1	66	62	61	60	59
	0.23.8	108	105	99	98	97
	0.23.10	106	105	99	98	97
	2.3.0	117	112	109	108	107
	2.4.0	120	119	114	113	112
48GB	1.1.1	378	348	303	299	304
	1.2.1	398	352	306	311	298
	0.23.8	588	566	463	454	456
	0.23.10	593	596	510	449	464
	2.3.0	696	630	579	529	564
	2.4.0	682	631	563	540	521
256GB	1.1.1	2005	1967	1795	1750	1626*
	1.2.1	1972	1934	1796	1795	1588*
	0.23.8	3339	3469	3001	3157	2790*
	0.23.10	3168	2971	3000	3320	2736*
	2.3.0	3461	3587	3248	3345	2885*
	2.4.0	3530	3650	3212	3301	3077*

* HDFS Replication Factor = 1

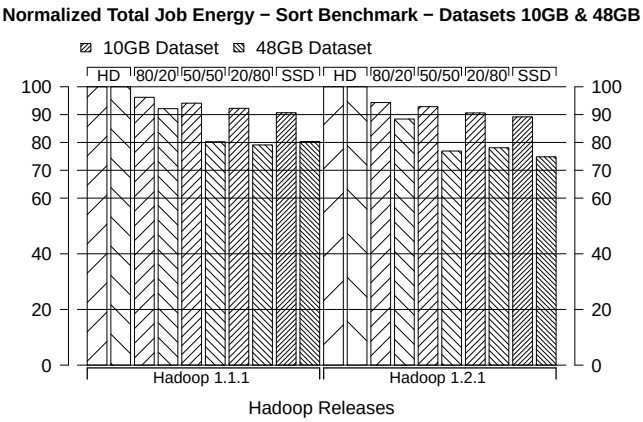

Figure 7: Energy Consumption: 1.x Releases

Next, we moved on to the experiments with larger datasets. Figure 6 shows the results for the 48GB Sort experiments. The results support the tendency toward energy savings when using SSD. 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. Additionally, a similar reduction can also be observed in the 50/50 and 20/80 configurations in Figures 5 and 6. The energy consumption results from the Sort benchmarks are listed in Table 5.

To promote a better data visualization, we normalized the energy results using the previously presented values – the average of the observations for each tested configuration. Therefore, for a pair release–dataset, we divided each value by the maximum value of the group. Generally this value is associated with the HD configuration, since it demands more power when running experiments. This allowed the side-by-side comparison of the results from different dataset

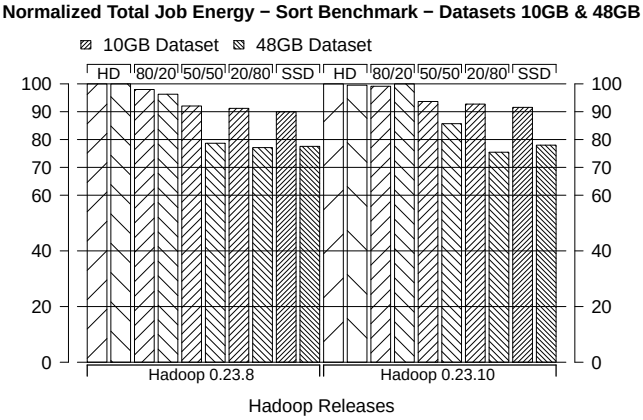

Figure 8: Energy Consumption: 0.23.x Releases

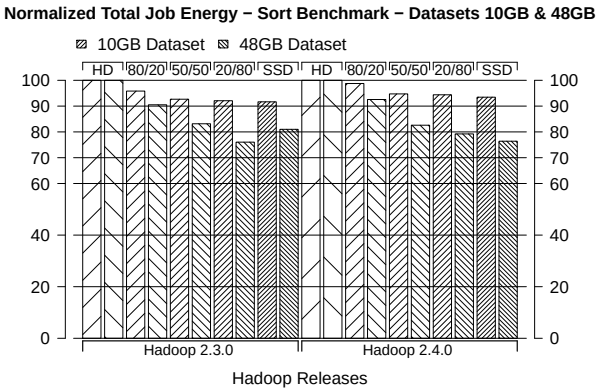

Figure 9: Energy Consumption: 2.x Releases

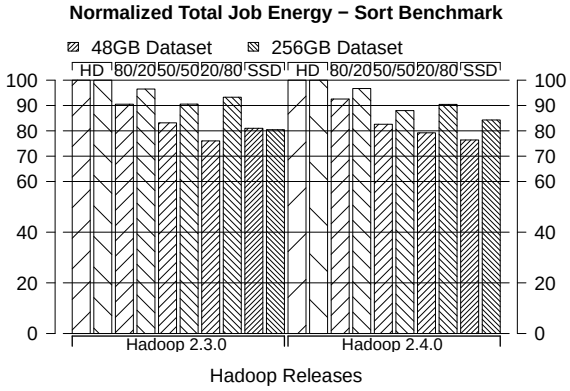

Figure 11: Energy Consumption: 2.x Releases

experiments, since their results are in different scales. Figures 7, 8, and 9 shows this panorama. We put together the experiments from the two initial Sort benchmarks (10GB and 48GB) in both versions from each release. Surprisingly, for some releases the use of 50/50 configurations result in energy consumption close to the power consumed when exclusively processing data from SSD. This can be clearly observed in releases 1.1.1, 1.2.1, 0.23.8, and 2.3.0. Even in the other two releases, the energy consumption rates are closer to the SSD values than to the HDs ones.

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 middle configurations, favoring the use of less SSD storage to achieve results similar to the SSD-only configurations. Table 5 presents these results. The power consumption reduction when using more SSD storage is clearly noticeable. During the 256GB Sort experiments, we observed an issue regarding the *SSDzone* total space. Our DataNode infrastructure has eight 120GB SSD disks, totaling 960GB of useful storage space. Using a replication factor of 3, a dataset triples its size in the 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 256GB Sort experiments ran without block replication in the $HDFS_H$ for the SSD configuration with this particular dataset. These results appears in Table 5.

We also compared the results of the 256GB Sort with the 48GB Sort in normalized graphs. Figures 10 and 11 indicate that we can achieve relatively higher performance and lower energy consumption by increasing the dataset size.

To further investigate the effects of block replication on energy consumption, we ran experiments with the 256GB Sort dataset on the HD configurations, presented in Figure 12. The results indicate that the triple block replication demands on average 1.5% more energy in the 1.x brach, 9.5% more energy in 0.23.x releases, and 6.2% on branch 2.x. Following our previous results, this means that in the configurations favoring the SSD storage, this percentage should be greatly reduced, since SSDs consumes far less energy than HDs.

5.2 CPU-Bound Benchmarks Results

To evaluate whether our approach favors I/O-bound jobs only, we performed two sets of experiments using CPU-bound jobs. The differences between the hybrid HD and SSD configurations were insignificant, thus we present only the results for the two extreme configurations using the Join

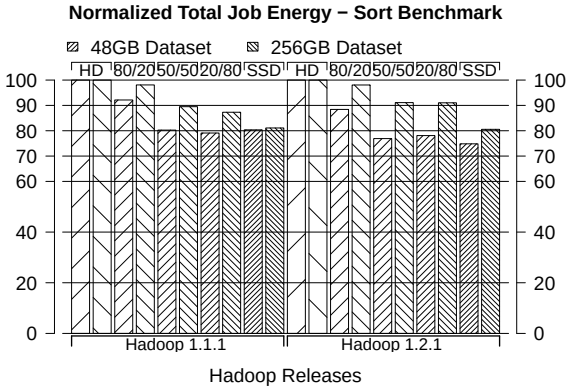

Figure 10: Energy Consumption: 1.x Releases

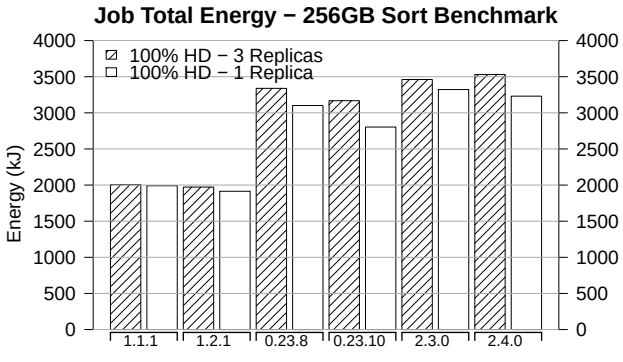

Figure 12: Impact of Triple Block Replica on Energy

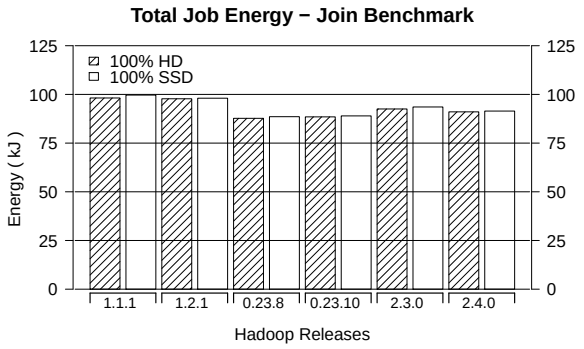

Figure 13: Energy Consumption, Join Benchmark

and K-Means benchmarks. Except for the differences among branches, the Join benchmark did not present any significant differences, as seen in Figure 13.

The Mahout K-Means is a hybrid benchmark that is CPU-bound in the iterations, and I/O-bound in clustering. With our setup (3 iterations), 3/4 of the execution in this benchmark was CPU-bound, while the rest was I/O-bound. As can be seen in Figure 14, again, there is no significant difference across the configurations, except for the differences among branches.

We thus conclude that, except for the difference among branches and releases, there are no significant gains in terms of performance and energy consumption when running CPU-bound jobs with our approach.

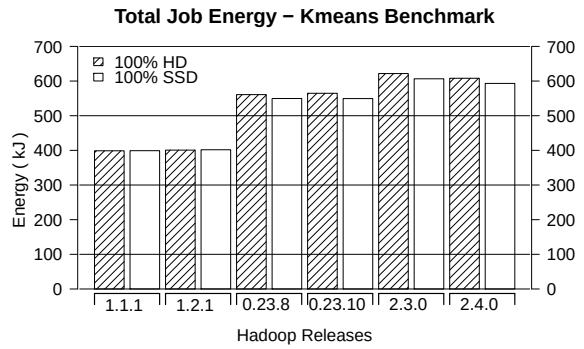

Figure 14: Energy Consumption K-Means Benchmark

5.3 Using SSD as Temporary Storage Space

Following the experiment's results, 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*. Figures 15 and 16 present the results. We notice that for the 10GB dataset there is no significant difference in the 1.x releases. The results are similar to the *SSD* configuration, which can be explained by the dataset size. But

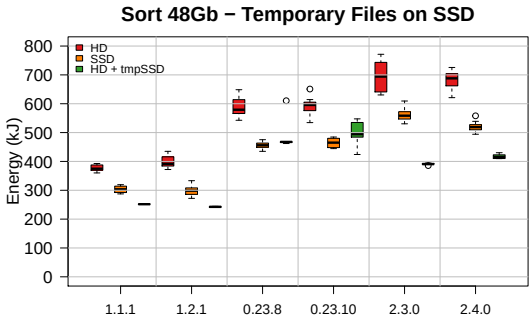

Figure 15: Energy using *SSDzone* as temporary

we observed a different behavior from 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 bigger datasets in the 0.23.x and 2.x releases. The novelty was the 1.x branch results, which performed better with bigger datasets. This means that not only did performance improve, but there were significant energy savings when using the *SSDzone* as temporary space.

As a general conclusion here, it is worth mentioning that both 1.x and 2.x releases significantly improved when using the *SSDzone* to store temporary files during Hadoop jobs, such that in most cases the performance was even better than storing the dataset files in the *SSDzone*. By contrast, releases from 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, using SSD 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 with them on *SSD*.

5.4 Speedup

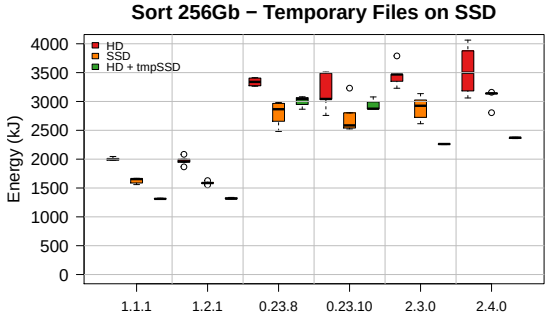

Figure 16: Energy using *SSDzone* as temporary

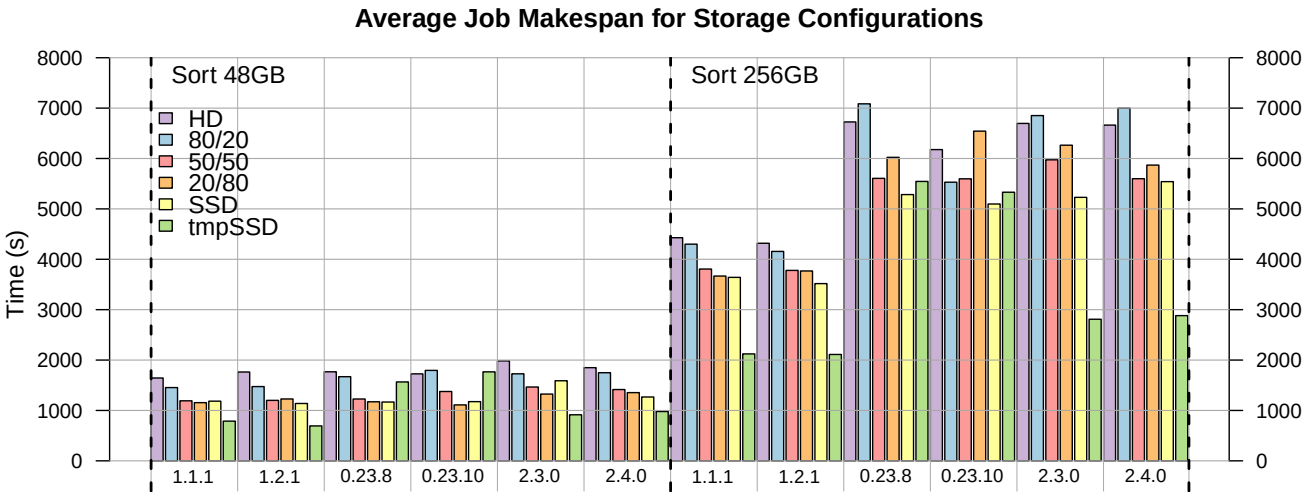

Figure 17: Hadoop Performance: Multiple Releases over Configurations

Regarding job makespan performance, we observed that storing more data in the SSDs enabled the jobs to run faster, which was expected, since SSDs provide higher throughput. The novelty here is the non-linear behavior of the job makespan as we increase the dataset size. With the 10GB dataset, the 80/20 configuration was on average 7% faster than the HD configuration, with only 20% of the data processed from the SSD; in the 50/50 configuration, jobs were on average 17% faster than the HD configuration; and, in the 20/80 configuration, 22% faster; finally, jobs running with the SSD-only configurations were on average 26% faster than purely running on HD. In the 48GB Sort, the observed speedups compared to the HD configuration were: 80/20, 8% faster; 50/50, 27% faster; 20/80, 31% faster; and SSD, 30% faster. The average makespan from the Sort benchmarks can be seen in Figure 17.

5.5 Cost Model Analysis

With the *SSDzone* as a temporary storage, we achieved more promising results than some SSD-only experiments. Figure 17 also presents these experiment's results. All experiments from the hybrid policy ran using the HD as temporary space. Thus, the *SSD* configuration stores the entire dataset on SSD disks, 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 SSDs. Thus, it is fair to compare these two experiment sets. Although the HD experiments tend to be much slower than the SSD, in this case the *HDzone* using SSD as temporary space in most cases outperforms the *SSDzone* experiments. Results from the 48GB and 256GB Sort experiments in 1.x and 2.x releases point to a speedup factor of more than 2 times when comparing *HD* versus *tmpSSD* configurations, and on average more than 1.5 times when comparing *SSD* versus *tmpSSD* configurations.

For the storage cost analysis, we assume that the $HD_{cost/GB} =$ \$0.05 and the $SSD_{cost/GB} =$ \$1.00 (20 times the $HD_{cost/GB}$). Following we plotted the ratio between job makespan and job storage cost for the sort jobs, using the latest tested release from each branch. We can observe in Figure 18 that, for each release, there is a pareto optimal configuration. This example can be used to identify which proportions are the best in the tradeoff between cost and performance. The more SSD is used, the more expensive the cost is, but with less hours spent to perform the job, less energy is used. On the other hand, by using more HD in the configurations, jobs take more hours to finish, demanding more energy, but the cost decreases greatly. This behavior can be observed in every release on the 48GB and 256GB datasets tested in the sort experiments. Finally we can 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.

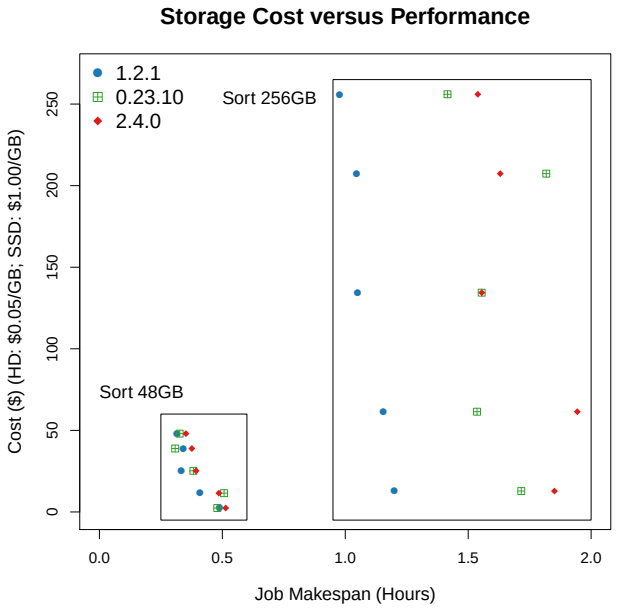

Figure 18: $HDFS_H$ storage cost versus performance

6. RELATED WORK

In our comprehensive literature research [27], we observed that HDFS has been modified to increase its performance in multiple ways: tuning the I/O mode [16,29,34], solving data placement problems [33], and adapting it to support small files processing [7, 16], since HDFS was not originally designed for such purposes. Some works replaced the original HDFS with a more suitable solution for specific compatibility problems [19, 24], or to support areas such as Cloud Computing [9, 19]. Yet, these approaches only targeted the HDFS' performance increases without considering energy consumption or using a combination of different storage devices.

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. Jeon et al. [15] analyze the Flash Translation Layer (FTL) – the SSD core engine – to understand the endurance implications of such technologies on Hadoop MapReduce workloads. Kang et al. [18] explore the benefits and limitations of in-storage processing on SSDs (the execution of applications on processors in the storage controller). Other researchers focus on incorporating SSDs into HDFS using caching mechanisms to achieve better performance [20,28,32,35]. A few works also discuss SSDs' impact on Hadoop [17,25], sometimes focusing on using SSDs as the sole storage device under HDFS. 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.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our approach to seamlessly integrating HD and SSD technologies into $HDFS_H$. Our block placement policy shows an energy consumption 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 general use of $HDFS_H$ affords immediate benefits since it increases MapReduce jobs' performance and reduces energy consumption. Yet, for now, users must manually define these configurations. In future work, we intend to investigate autonomic heuristics that would analyze the workflow to automatically and dynamically configure $HDFS_H$ for optimal performance and energy consumption.

Regarding Hadoop branches and their significant performance and energy consumption differences, 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. In the case of the latter option, we strongly recommend selecting Hadoop releases from 1.x branch, and applying the compatible HDFS security patches.

Additionally, we observed that the *tmpSSD* configuration can bring benefits when used with the 1.x and 2.x branches. This configuration can increase speedup rates to outperform the exclusive use of SSD on certain releases and benchmarks. Four out of 6 tested releases showed performance enhancements using this configuration on our larger datasets. Therefore, the use of SSD as an energy saver and performance enhancer for MapReduce computations ought to be seen as a viable alternative to existing Hadoop clusters, since Hadoop does not need to be modified to allow tmpSSD configurations. We recommend that Hadoop deployment consider using SSDs for temporary space.

Finally, we are also concerned with the Hadoop middleware architectural changes. We are conducting a code repository data mining experiment to investigate how source-code modifications across Hadoop releases have impacted performance and, consequently, energy consumption. Additionally, our results may show how much of the power being used relates to job makespan, correlating the evolution of the framework and its architectural changes with performance and energy consumption.

Acknowledgment

This work was funded by Fundação Araucária, CNPq-Brazil, and by the Emerging Leaders in the Americas Program (ELAP) from the Government of Canada. Abram Hindle is supported by an NSERC Discovery Grant.

8. REFERENCES

[1] Apache Mahout. Apache Mahout, 2014.

[2] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. 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.

[3] P. B. Brandtzæg. Big data - for better or worse, May 2013.

[4] T. P. P. Council. TPC Benchmark H (Decision Support) Standard Specification, Jun 2002.

[5] J. Dean and S. 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.

[6] D. J. DeWitt, E. Paulson, E. Robinson, J. Naughton, J. Royalty, S. Shankar, and A. Krioukov. Clustera: an integrated computation and data management system. *Proceedings of the VLDB Endowment*, 1(1):28–41, Aug. 2008.

[7] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, and Y. Li. A novel approach to improving the efficiency of storing and accessing small files on Hadoop: A case study by PowerPoint files. In *International Conference on Services Computing*, pages 65–72. IEEE, july 2010.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[9] S. Guang-hua, C. Jun-na, Y. Bo-wei, and Z. 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.

[10] M. Hachman. SSD prices face uncertain future in 2014, 2014.

[11] A. Hadoop. Apache hadoop releases. http://hadoop.apache.org/releases.html, 2015.

[12] J. Hamilton. Overall data center costs, September 2010.

[13] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51, March 2010.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, Mar. 2007.

[15] H. Jeon, K. El Maghraoui, and G. 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.

[16] D. Jiang, B. C. Ooi, L. Shi, and S. Wu. The performance of MapReduce: an in-depth study. *Proceedings of the VLDB Endowment*, 3(1-2):472–483, Sept. 2010.

[17] K. Kambatla and Y. Chen. The truth about mapreduce performance on ssds. In *28th Large Installation System Administration Conference (LISA14)*, pages 118–126, Seattle, WA, Nov. 2014. USENIX Association.

[18] Y. Kang, Y. suk Kee, E. Miller, and C. 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.

[19] G. Kousiouris, G. Vafiadis, and 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.

[20] K. Krish, M. Iqbal, and A. Butt. Venu: Orchestrating ssds in hadoop storage. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 207–212, Oct 2014.

[21] D. Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001.

[22] J. C. MacCallum. Disk drive prices. http://www.jcmit.com/diskprice.htm, 2014.

[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. 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.

[24] S. Mikami, K. Ohta, and O. 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.

[25] S. Moon, J. Lee, and Y. S. Kee. Introducing ssds to the hadoop mapreduce framework. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 272–279, June 2014.

[26] F. Pan, Y. Yue, J. Xiong, and D. Hao. I/O characterization of big data workloads in data centers. In J. Zhan, R. Han, and C. 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.

[27] I. Polato, R. Ré, A. Goldman, and F. Kon. A comprehensive view of Hadoop research - A systematic literature review. *Journal of Network and Computer Applications*, 46:1 – 25, 2014.

[28] M. Roger, Y. Xu, and M. Zhao. Bigcache for big-data systems. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 189–194, Oct 2014.

[29] J. Shafer, S. Rixner, and A. 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.

[30] K. Shvachko, H. Kuang, S. Radia, and R. 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.

[31] D. Vesset, A. Nadkarni, C. W. Olofson, and D. Schubmehl. Worldwide big data technology and services 2012-2016 forecast. Technical report, IDC Corporate USA, 2012.

[32] B. Wang, J. Jiang, and G. Yang. mpCache: Accelerating mapreduce with hybrid storage system on many-core clusters. In C.-H. Hsu, X. Shi, and V. Salapura, editors, *Network and Parallel Computing*, volume 8707 of *Lecture Notes in Computer Science*, pages 220–233. Springer Berlin Heidelberg, 2014.

[33] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. 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.

[34] J. Zhang, X. Yu, Y. Li, and L. Lin. HadoopRsync. In *International Conference on Cloud and Service Computing*, pages 166–173, dec. 2011.

[35] D. Zhao and I. Raicu. Hycache: A user-level caching middleware for distributed file systems. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1997–2006, May 2013.

[36] P. Zikopoulos and C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. Mcgraw-hill, 2011.