Characterizing the Efficiency of Data Deduplication for Big Data Storage Management

Ruijin Zhou, Ming Liu, Tao Li
University of Florida
Gainesville, Florida, USA
{zhourj, mingliu} @ufl.edu, taoli@ece.ufl.edu

***Abstract*—The demand for data storage and processing is increasing at a rapid speed in the big data era. Such a tremendous amount of data pushes the limit on storage capacity and on the storage network. A significant portion of the dataset in big data workloads is redundant. As a result, deduplication technology, which removes replicas, becomes an attractive solution to save disk space and traffic in a big data environment. However, the overhead of extra CPU computation (hash indexing) and IO latency introduced by deduplication should be considered. Therefore, the net effect of using deduplication for big data workloads needs to be examined. To this end, we characterize the redundancy of typical big data workloads to justify the need for deduplication. We analyze and characterize the performance and energy impact brought by deduplication under various big data environments. In our experiments, we identify three sources of redundancy in big data workloads: 1) deploying more nodes, 2) expanding the dataset, and 3) using replication mechanisms. We elaborate on the advantages and disadvantages of different deduplication layers, locations, and granularities. In addition, we uncover the relation between energy overhead and the degree of redundancy. Furthermore, we investigate the deduplication efficiency in an SSD environment for big data workloads.**

Keywords: Big Data, Deduplication, Storage Management

I. INTRODUCTION

More than 2.5 quintillion bytes of data are generated every day. 90% of the total data has been created just in the past few years alone. To contain such a massive amount of data, storage continues to grow at an explosive rate (52% per year) [1][14]. By the end of 2015, the size of the total generated data will surpass 7.9 zettabytes (ZB). This number is expected to reach 35 ZB in 2020, which has proven to be too conservative [2][15].

The big data era is here and it is more than just a matter of data size. Big data also represents the increasing complexity of the data handling process. Instead of sitting in one archive storage, data is now typically distributed among different locations. Worse, 80% of generated data is claimed to be unstructured [3]. Hadoop [8] is emerging as the most popular big data platform to manage and structure data for further analysis. However, the replication mechanism of Hadoop generates a lot of duplicate data, which occupy both disk space and network bandwidth. Moreover, Hadoop is increasingly being built on top of a virtualization layer, which further yields redundancy within and across virtual machine disk images [26][27][28]. Redundancy in big data workloads becomes even higher in the case of 1) deploying more virtual machines (VMs) and 2) expanding the active data set.

Deduplication [22] is being widely used to reduce cost and save space in data centers. This technology eliminates redundancy by removing data blocks with identical content. Deduplication is not only used in backups and archives, but also increasingly adopted in primary workloads [4][5]. The benefits of deduplication are 1) saving disk space, which further leads to saving money on buying storage devices and 2) reducing IO traffic, which results in higher IO throughput.

Cloud vendors pay more attention to deduplication because there are many replicas in the data outsourced to the cloud (e.g. 75% of data are redundant based on a survey [6]). The redundancy is even higher for the big data workloads on Hadoop platform. As a result, more companies are researching deduplication technology, and it has become a common practice in cloud storage environments.

Deduplication seems to be a suitable solution for data explosion in the big data era by 1) slowing down the data expansion speed by eliminating redundant data, and 2) relieving pressure on disk bandwidth by removing redundant IO accesses. However, deduplication also introduces overhead to the system. For example, hash indexing needs be performed for every IO request to identify duplicates, which results in longer IO response time. In addition, extra CPU power is required to calculate the hash values in each IO request, which leads to higher energy consumption. Since the volume of IO requests is enormous and increasing in big data workloads, the overall performance and energy efficiency under different deduplication configurations is worthwhile to be studied thoroughly. Towards this end, we first justify the need for deduplication by measuring the deduplication ratio (input/output size) of typical big data workloads. We characterize the performance impact under various deduplication configurations, namely deduplication layer (global versus local), deduplication location (metadata versus data), and deduplication granularity. In addition, we study the energy impact for workloads with different IO behaviors and degrees of redundancy. Moreover, we consider an emerging storage medium (solid state drive or SSD) in our storage environment.

The following are the key observations we made:

- Big data workloads have redundancy. On average, 44% of the active data set in our big data workloads is redundant. Deploying an additional VM yields 97%

more redundant data. Using a replication mechanism in the Hadoop distributed file system (HDFS) introduces 19% redundant data on average. The data node contains 25% more redundant data than the name node.

- Deduplication helps workloads utilize more disk throughput (3X higher when deduplication is on in our experiment), which leads to, at most, a 45% performance improvement. However, due to the overhead of hash indexing, performance can degrade by 161% for some benchmarks in an extreme case.
- Extra hash computation on the CPU leads to additional power consumption (around 10%), which results in energy overhead (7%). However, for the benchmarks with a high level of redundancy, the overall energy can be saved (by 43% in our experiment) because deduplication reduces workload execution time. There is a strong correlation between energy impact and the degree of redundancy.
- In a hybrid SSD/HDD environment, deduplication can improve the system performance (by up to 17% in our experiment) if the SSD ratio is tuned properly. However, in a pure SSD environment, deduplication costs performance and energy overhead (about 5% and 6% respectively).

After further investigating the experimental results and configuring deduplication in different ways, we derive and highlight the following insights and design implications for data deduplication in a big data storage environment:

- The Performance impact of deduplication is determined by both the benefit of avoiding redundant disk traffic and the overhead of hash indexing.
- Local deduplication can leverage parallelism to hide hashing overhead and maintain data availability. Nevertheless, it cannot remove all redundant disk accesses. Global deduplication can eliminate redundant disk accesses. However, it has high hashing overhead and reduces data availability.
- Deduplication on the name node (metadata) has less hashing overhead than deduplication on the data node because it has less data. However, it also saves less disk traffic. Deduplication on data nodes (data) removes more redundant disk accesses but has high hashing overhead. Deduplication on metadata yields better performance if the degree of redundancy in the data set is low.
- Fine-grained deduplication avoids more disk traffic but leads to higher hashing overhead than coarse-grained deduplication. Coupling fine granularity with local deduplication can exploit more redundant disk accesses and hide hash overhead. Using coarse granularity in global deduplication can help lower hashing overhead.
- Since SSD has higher IO performance than HDD, deduplication yields less benefit of reducing disk traffic in SSD than in HDD. In a hybrid SSD and HDD storage environment, only if the SSD ratio is low can deduplication improve performance.

The rest of this paper is organized as follows: Section II describes the background and motivation for this work. Section III explains the evaluation methodology. Section IV presents and analyzes the experimental results. Section V discusses deduplication efficiency in an SSD storage environment. Section VI concludes this paper.

II. Backgrounds and Motivation

Big data is defined as any attribute that challenges the constraints of a system's capability or business need [7]. There are a lot of big data examples: Amazon runs the world's three largest Linux databases (7.8TB, 18.5TB, and 24.7TB) to contain millions of back-end operations and queries from more than half a million third-party sellers and Walmart's transaction database contains more than 2.5 petabytes of data. As the volume of data grows from terabytes (TB) to zettabytes (ZB), the pressure on the storage capacity becomes higher [16]. Companies, either cloud providers or data center owners, have to pay additional money to purchase extra hard drives to contain such big data. At the age when data is exploding, the cost of storage stands out in a company's budget.

What is worse, big data is more than just a matter of data size. It also involves the way we deal with the data. IBM claims that 80% of data captured today is unstructured, including climate information gathered by sensors, posts on social media websites, cell phone GPS signals, and online transaction records. In order to manage and structure those data, Hadoop [8] is emerging as a core platform for big data applications. Hadoop implements a computational paradigm named MapReduce, where data and application are divided into small fragments and distributed among different nodes. For the purpose of data availability, Hadoop Distributed File System (HDFS) has the attribute of keeping multiple copies of data blocks, which makes the big data even bigger. Worse, each node in Hadoop is usually a virtual machine with a virtual disk image, which has high redundancy. The data for big data workloads is big not only because the useful and identical data is big, but also because the redundancy is very high. A large amount of redundant data costs undesired disk traffic and greedy demand for storage capacity.

Deduplication technology [22] has been researched for about a decade now and widely adopted in backup and archive storage. By comparing the hash value of data blocks, it identifies and removes duplicated data blocks so that the storage space is saved. Recently, deduplication has started to emerge in primary storage systems, which also have a significant amount of redundant data [4]. In primary storage systems, hash computation and comparison are performed in each IO request. Once the data is identified as redundant, the IO request will not be issued. Instead, a pointer is created to

the actual data. By doing so, IO traffic is reduced and IO performance is improved [19][20].

There are two types of deduplication: inline and offline. Several studies [4][9][10] shed light on the advantages and disadvantages of these two schemes. The offline scheme, which performs deduplication after the data is saved on hard drives, is used to save disk space on archival storage. The inline scheme performs deduplication before the data is issued so that both disk traffic and space are saved. In this paper, we use inline deduplication. Additionally, where to implement deduplication is also debatable. It can be implemented at the HDFS level or virtualization level. Since the redundant data within the VM disk images cannot be detected by HDFS, we believe it is better to apply deduplication at the virtualization level. Currently, ZFS [11] and Opendedup [12] support deduplication functionality for VM disk images. In this paper, we select ZFS as our inline deduplication tool.

Using deduplication in the big data storage system seems to be attractive because: 1) a deduplication scheme can help reduce a large amount of redundant data generated by big data workloads; 2) disk IO traffic can be significantly reduced so that the whole system can gain much better performance.

However, deduplication also introduces overhead to the system, namely, 1) extra time for hash indexing in each IO request, and 2) extra power consumption for hash value computation in each IO request. More IO requests result in higher overhead in both performance and power. Performance impact is determined by both the benefit of saving disk traffic and the overhead of hash indexing. Energy impact is determined by power overhead of hash computation and workload execution time.

Workloads have various IO behaviors and different degrees of redundancy, which will largely affect their sensitivity on disk traffic and hash indexing. Once the benefit of reducing disk IO traffic and the overhead of hash indexing changes, workload performance becomes less predictable, which further leads to undetermined energy consumption. Apart from saving space, the efficiency of deduplication for big data storage management is still largely unknown. To reveal the performance and energy impact introduced by deduplication, this paper thoroughly characterizes the efficiency of deduplication under various big data workloads with different configurations.

III. Evaluation Methodology

A. Experimental Setup

We evaluate the workload performance and CPU overhead on a Dell PowerEdge rack, which is equipped with four 1U PowerEdge R610 servers and two 2U PowerEdge R710 servers. Each server is assembled with two 2.4GHz quad-core Intel Xeon processors, 24GB of RAM and one 160GB local hard drive. The 2U servers, which have six additional 1TB hard drives and four 240GB SSDs, are presented as the storage server to hold the large dataset. All servers are connected to an in-rack private switch (Dell PowerConnect 6224) with 1Gb/s bandwidth. The overview of our experimental setup is shown in Figure 1. We install Xen 4.2.1 with Linux kernel 3.0.0-12 to support virtualization. We deploy ZFS on Linux [11] to support deduplication functionality. Software iSCSI is setup to expose disks in the storage server to the compute server. We boot up a maximum of 10 VMs for Hadoop slave nodes. For each VM, we assign 2 vCPUs, 8GB RAM, 40GB local hard drives and one 500GB ZFS volume as the data drive. Inside the virtual machine, we use Ubuntu 12.04 as the guest operating system and Hadoop 1.0.4 as our big data platform. To avoid memory exceptions from Hadoop, we increase the Java heap size to 2GB. In addition, to avoid background jobs in the operating system, we use a ZFS volume as the data disk to hold data generated by Hadoop. Therefore, the deduplication will only apply to Hadoop workloads. To measure the energy and power consumption, we connect a *Watts up?* [21] meter to the power supply to obtain the power trace during workload execution.

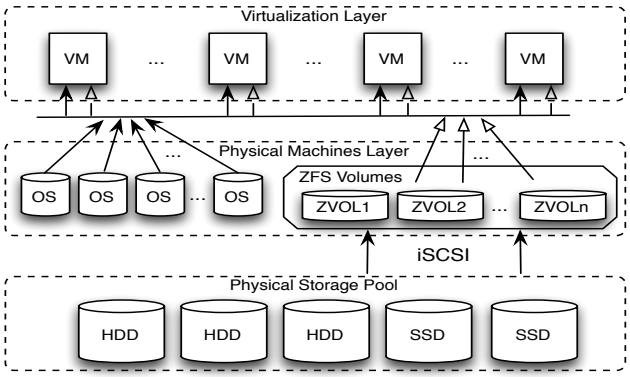


Figure 1. Overview of experimental setup

B. Workload Execution Scenarios

On top of the Hadoop platform, we run several typical big data applications including web search, machine learning, analytical query, and sorting. HiBench benchmark suite [13], which is provided by Intel, includes a wide range of Hadoop applications, namely *PageRank, Nutch Indexing, Bayesian Classification (Bayes), Kmeans Clustering, Hive, Sort, Terasort, and Enhanced DFSIO*. The workloads are summarized in Table 1. By tuning the parameters, we change total data size from 0.5TB to 5TB, within which the active dataset size ranges from 1GB to 100GB. Note that *Enhanced DFSIO* is an IO micro benchmark that reports IO throughput. For other benchmarks, we use execution time to indicate the performance.

Table 2 summarizes all workload and deduplication configurations we considered in this paper. From a workload perspective, big data means the data is very large and growing. Therefore, we consider the following three types of data growth: 1) Deploying more VMs in the Hadoop platform, which usually happens when more computation

power is needed. 2) Expanding the active dataset, which usually occurs when more input data is gathered. 3) Increasing the replicas in HDFS, which is usually used to guarantee higher data availability. We also consider compression technology [17], which slows down the speed of data growth. From a deduplication point of view, we consider three aspects: deduplication layer (global versus local), deduplication location (metadata versus data), and deduplication granularity. Global deduplication means that all VM disk images are assigned to one ZFS pool. Thus, all redundant data blocks can be observed by the deduplication mechanism. Local deduplication means each VM disk image is assigned to one ZFS pool. Deduplication for each VM is handled separately by different ZFS processes. For metadata deduplication, we only apply deduplication on the name node, while we apply deduplication on data nodes for data deduplication. The deduplication granularity is also tunable by changing the record size in ZFS. In our experiments, we use 1K to 128K as our granularities. We also consider SSDs in our storage environment. We assign more virtual machine disk images on to the SSD to increase the ratio of SSDs in the Hadoop platform.

Table 1. Workload summary

Category	Benchmarks	Tunable Parameters
Web Search	*PageRank*	Pages
	Nutchindexing	Pages
Machine Learning	*Bayesian Classification*	Pages, class
	Kmeans Clustering	Samples, samples per input file, dimensions
Analytical Query	*Hive*	Universities, pages
Micro-benchmarks	*Sort*	Data size
	Terasort	Data size
	Enhanced DFSIO	Number of read files, read file size, number of write files, write file size

Table 2. Execution scenarios

Workload Configurations	
Scenarios	Description
Deploy more VMs	Deploy more hadoop nodes
Expand active data set	Expand input dataset
Replicas	Increase/decrease replicas in HDFS
Compression	Turn on/off compression in HiBench
Deduplication Configurations	
Global/Local Deduplication	Global: all VMs in one ZFS pool Local: one VM in one ZFS pool
Metadata/Data Deduplication	Metadata: deduplication on name node Data: deduplication on data nodes
Granularity	Change record size in ZFS (1K –128K)
SSD Ratio	Change SSD ratio in storage pool

C. Metrics

The deduplication ratio is defined as the ratio between the sizes of data before and after deduplication is applied. A larger value of the deduplication ratio indicates a higher level of redundancy. ZFS computes the deduplication ratio online while a workload is running. Thus, in this case, deduplication ratio is equal to the total bytes that are issued to the ZFS pool divided by the total bytes that are actually issued to the disks. In this paper, we use the deduplication ratio reported by ZFS.

Throughput is defined as the size of data that can be delivered within a certain period. By using deduplication, the replicas on the network will be removed, which is expected to increase available disk bandwidth for the workloads. So, we use throughput reported by *Enhanced DFSIO* to measure the traffic difference between deduplication when it is on and off.

Program execution time is used to measure the performance of big data workloads. Although IO behavior is the focus in this paper, CPU and memory are also important in big data workloads. Thus, it is fair to use program execution time to indicate the overall performance. By observing and comparing the execution time before and after using deduplication, we can see how much impact deduplication introduces to the workloads.

CPU overhead is obtained by monitoring CPU usage of the ZFS process during workload execution. Note that ZFS has other functions (e.g. data transaction management, IO pipeline, etc.) to perform besides deduplication. Therefore, CPU usage for ZFS is non-zero even when the deduplication function is turned off.

The power consumption trace is gathered by the *Watts up?* meter, which measures both voltage and current on the power cord. *Watts up?* computes the power based on these two data series. We further multiply the value of average power consumption by the total execution time to obtain the average energy consumption for workloads.

IV. Efficiency Analysis for Deduplication in Big Data Environment

A. Redundancy Analysis

Three behaviors in big data workloads will inevitably yield redundancy: 1) deploying more nodes, 2) increasing the dataset size, and 3) using replication for availability.

As the need for computation increases, we have to deploy more nodes to provide more computation power. Since each node is a virtual machine, the number of virtual disk images increases when we increase the number of nodes. Within each disk image, there is a large percentage of redundant data: empty data blocks and similar operating system files. Figure 2 shows the deduplication ratio prior to running all Hadoop benchmarks The deduplication ratio increases from 70X to 220X when we deploy more VMs. Even within a single virtual image, there is 97% redundant data (due to empty blocks and similar files), which yields a 73X deduplication ratio. Therefore, redundancy becomes higher as more VMs are deployed in the Hadoop environment. In this case, applying deduplication can save a lot of space.

Another attribute of big data workloads is their large and increasing volume of dataset. Figure 3 shows the

deduplication ratio of the dataset of our benchmarks on a single node. As can be seen, different benchmarks have different deduplication ratios, which means the levels of redundancy are different. This is true in the real world. Some applications (e.g. *Kmeans* and *Sort*) have high redundancy and the degree of redundancy becomes higher as the size of the dataset increases; some applications have low redundancy and they always generate different data as the dataset increases, which is similar to *Terasort* in our benchmark suite. Therefore, the correlation between dataset size and the level of redundancy is not strong. However, on average, 44% of the active dataset in big data workloads is redundant.

Some well-designed workloads incorporate a compression technique directly in their code with the goal of sacrificing a certain degree of performance to save disk space. In this case, the deduplication ratio will be near 1. Here, we intentionally add compression in the *Kmeans* benchmark to see how compression affects the deduplication ratio. Figure 4 shows that the deduplication ratio on a compressed dataset is 50% less than that on an uncompressed dataset. The reasons are 1) compression breaks the pattern of data layout, which lowers the possibility of duplicates; 2) compression makes the dataset denser than before and eliminates zero bits, which is a major source of duplicates.

Although compression technique can lower the possibility of duplicated data blocks, Hadoop, as well as other distributed big data platforms, will inevitable generate a large amount of duplicates because of replication schemes. To guarantee the availability of data blocks, Hadoop duplicates data blocks and distributes them to different data nodes. The default number of replicas is three, which can also be changed by the user to adapt to their specific situation. The higher the number of the replicas, the better the availability. In Figure 5, we increase the number of replicas from 1 to 6. We find that as the number of replicas increases, the deduplication ratio will increase across all different input datasets. On average, adding one more replica introduces 19% redundant data as shown in Figure 5. In the default setting, every data block in HDFS will be copied 3 times and distributed to different nodes. The correlation between the number of replicas and the degree of redundancy is very strong. Therefore, applying deduplication to Hadoop workloads can save a lot of redundant disk traffic and space, although the availability issue should be taken care of.

By further investigating into the Hadoop structure, we find that Hadoop uses the name node to store all the metadata (indexing files) and data nodes to store actual data blocks. Therefore, in Figure 6, we apply deduplication separately on the name node and data nodes. As can be seen, in most cases data nodes have the larger amount of duplicates. On average, data nodes have 25% more redundant data than the name node. Considering the replicas in HDFS, when the dataset keeps increasing, the duplicates in data nodes will increase at a much faster speed than the name node. However, in some benchmarks (e.g. *Terasort, and Bayes*), the name node has an equal or even larger deduplication ratio than the data nodes. Since the name node stores metadata, which impacts the IO performance, the idea of using deduplication on the name node to reduce disk access and improve performance is worthwhile to be studied. From the disk space point of view, data nodes are better places to apply deduplication than the name node.

To summarize, big data workloads have a high level of redundancy originating from: 1) deploying more nodes, 2) increasing the number of replicas, and 3) increasing the size of the input dataset, which might also result in high redundancy for some workloads. For the Hadoop platform, the data nodes have more redundancy than the name node. Since redundancy is inevitable in big data workloads, the deduplication technique is valuable for a big data storage environment.

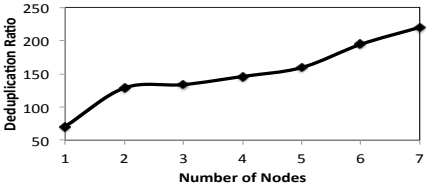


Figure 2. Deduplication ratio when deploying more nodes

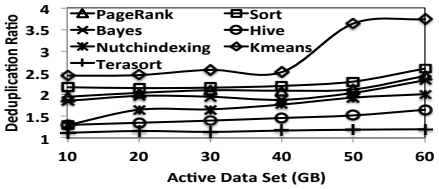


Figure 3. Deduplication ratio when expanding active dataset

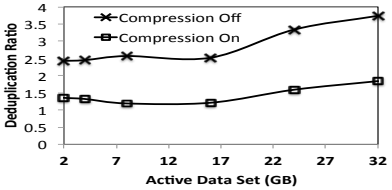


Figure 4. Deduplication ratio when compression is on/off (on *Kmeans*)

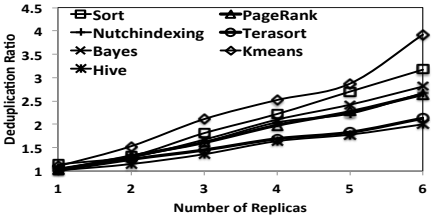


Figure 5. Deduplication ratio when increasing the number of replicas

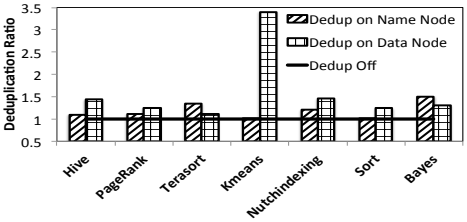


Figure 6. Deduplication ratio when deduplication on name node/data node

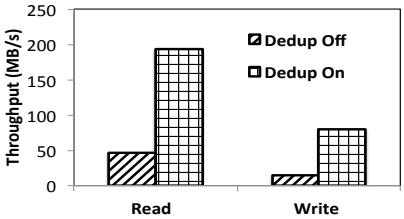


Figure 7. Throughput when deduplication is on/off (reported by *Enhanced DFSIO*)

B. Performance Impact

The performance impact introduced by deduplication is determined by two factors: the benefit of reducing redundant disk accesses and the overhead of adding extra CPU cycles in each IO request. Using deduplication, redundant IO requests do not need to be issued to hard drives. Instead, those IO requests just need to update hash indexes and pointers in memory, which is much faster than accessing hard drives. If the workload keeps issuing IO requests with identical contents, only one IO request ideally needs to be issued to hard drives. In Figure 7, we use the IO micro benchmark *Enhanced DFSIO* to measure the IO throughput before and after deduplication is applied. As can be seen, the read and write throughput with deduplication are 4 and 3.5 times higher than those without deduplication respectively. From a disk access point of view, deduplication can improve performance by avoiding redundant IO requests. However, to identify redundant IO requests, extra CPU cycles are needed to compute the hash value of data blocks and go through the hash table to find the address of the data blocks with the same content. This extra process occurs in each IO request, which makes the IO response time longer. If the volume of IO requests is large, which is usually true for big data workloads, this overhead will be significant. From a single IO request point of view, deduplication can decrease the performance by slowing down the IO handling process. Therefore, the performance impact depends on which one plays the dominant role: the benefit of removing redundant disk access or the overhead of prolonging each single IO request.

On the one hand, to maximize the benefit of avoiding disk access, we need to identify as many duplicates as possible. On the other hand, to minimize the overhead of long IO requests, we can either use an advanced hash indexing algorithm or reduce the amount of IO requests that are hashed. To achieve this, we consider three different aspects: 1) deduplication layer, 2) deduplication location, and 3) deduplication granularity.

Deduplication layer: Figure 8 shows the concept of different deduplication layers: local and global. As shown in Figure 8 (a), local deduplication means deduplication is only applied within a single VM. To achieve this, different VMs should be assigned to separate ZFS pools. Thus, the deduplication mechanism can only detect replicas within a single node. Redundant IO requests across different VMs are still issued to hard drives. On the other hand, Figure 8 (b) illustrates the global deduplication, which means the deduplication technique is applied across different VMs. To set this up, all virtual machine disk images should be assigned to the same ZFS pool. As a result, every IO request from Hadoop workloads will go through the same deduplication "filter". Therefore, the duplicated IO requests across VMs can be detected.

Local deduplication and global deduplication have their advantages and disadvantages. Local deduplication cannot remove replicas across different VMs. So, it preserves the replication mechanism but only handles redundancy within single VM. Global deduplication removes all possible replicas within and across VMs. So, it achieves minimal disk accesses at the cost of lowering data availability.

Due to the page limitation, Figure 9–12 show the execution time of 4 typical benchmarks. We identify the following four trends. Similar trends holds for the rest of the benchmarks.

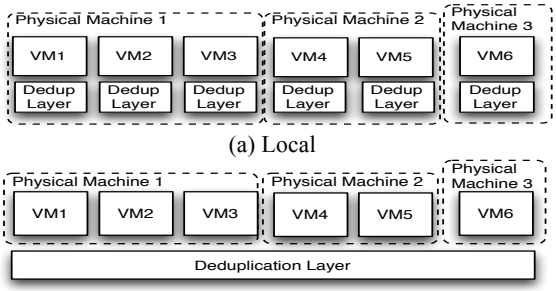


Figure 8. Different deduplication layers

i) Local deduplication yields performance overhead when expanding active dataset size

Figure 9 shows the execution time of big data workloads when local deduplication is turned on and off under the situation of expanding the active dataset. First, as the size of the active dataset increases, the execution time of the big data application increases due to the increasing amount of processing data. If we turn on the local deduplication, as shown in Figure 9 (a–d), the workload execution time becomes longer. Therefore, in this situation, the overhead of hash computation and indexing takes the dominant role.

Further looking into each benchmark, we find that when the active dataset expands, performance overhead becomes bigger for some benchmarks (e.g. *Page Rank, Terasort*) but not for the others. As Figure 3 shows, redundant data for *Page Rank* and *Terasort* does not increase as the active dataset increases. Thus, as the volume of IO requests increases, the increase of hash indexing and computation overhead overwhelms the decrease of disk accesses. Therefore, performance overhead becomes even bigger for these types of benchmarks.

ii) Local deduplication leverages parallelism to hide hashing overhead when deploying more VMs

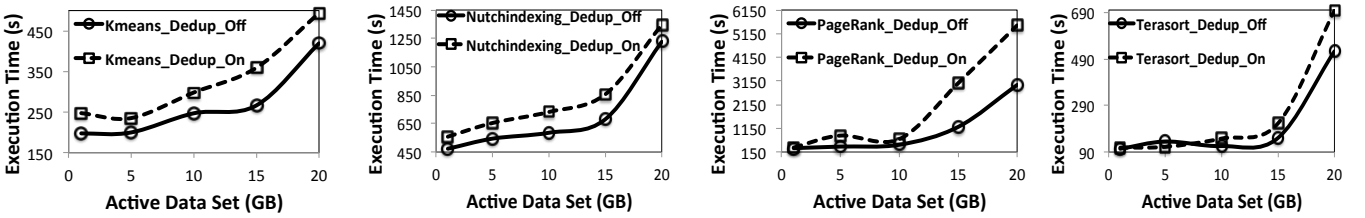
Figure 10 shows the execution time when local deduplication is applied under the situation of deploying more VMs. Since more computation power (CPUs) is deployed as the number of nodes increases, the execution time becomes less. If local deduplication is turned on, the performance degrades in most cases, which means the overhead plays the dominant role. However, this is not the case for the benchmarks *Kmeans* and *Nutchindexing*. Since deduplication for different nodes is handled in separated processes, a higher percentage of IO requests can be handled in parallel when more nodes are used. Although the amount of hash indexing and computation does not change, more of them are performed in parallel when more VMs are deployed. Therefore, the hash indexing and computation overhead decreases. If the number of nodes is large enough, the performance with deduplication can be better than without deduplication.

iii) Global deduplication can detects replicas both within and across VMs

Figure 11 shows the execution time when global deduplication is applied under the situation of expanding active data set. Since global deduplication can remove all redundant disk accesses, the benefit of reducing disk IO accesses is larger than in local deduplication. And the hash indexing overhead remains the same because the number of IO requests is fixed on a specific workload. Furthermore, global deduplication can detect redundant data generated by the replication mechanism in HDFS because the ZFS pool with a global view can see all the replicas across nodes. So, global deduplication can yield better performance than local deduplication. Therefore, in most cases (e.g. *Kmeans, PageRank*, and part of *Nutchindexing*), turning on deduplication can yield shorter execution time. As the size of the active dataset increases, the advantage of using deduplication becomes more visible. Since *Terasort* does not have enough redundancy, the saving on disk accesses cannot amortize the overhead of hash indexing and computation, which leads to lower performance when deduplication is on.

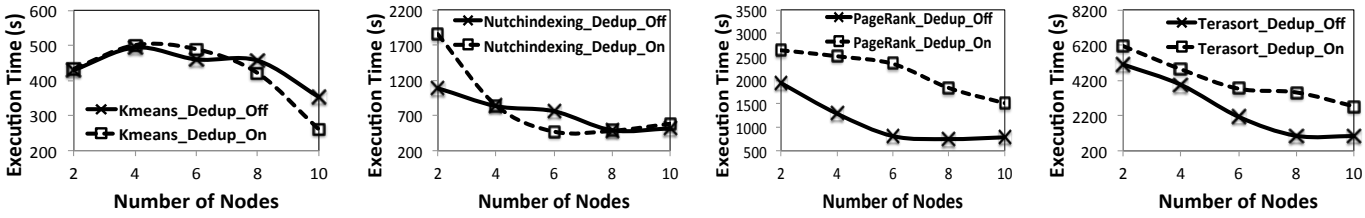
iv) Global deduplication cannot leverage parallelism when deploying more VMs

Figure 12 shows the execution time when global deduplication is used under the situation of deploying more VMs. As more nodes are deployed, the total execution time decreases due to more computation units. However, unlike local deduplication, global deduplication cannot parallelize hash computation and indexing as more VMs are deployed. Therefore, unless there is abundant redundancy, such as in *Kmeans*, the performance will not be better (e.g. *Nutchindexing, PageRank*, and *Terasort*).



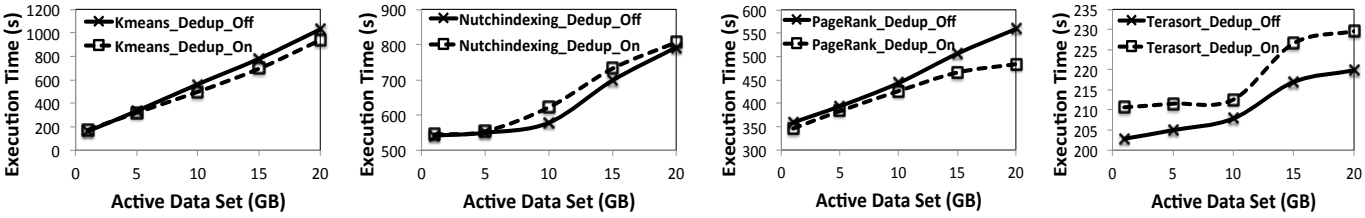
(a) *Kmeans* (b) *Nutchindexing* (c) *Page Rank* (d) *Terasort*

Figure 9. Execution time when active dataset expands and local deduplication is on/off



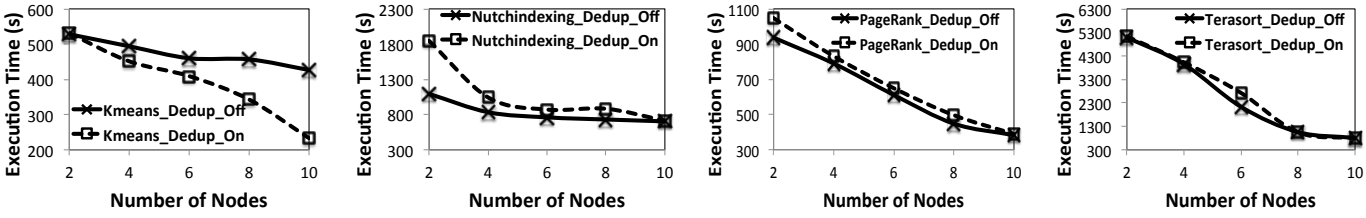
(a) *Kmeans* (b) *Nutchindexing* (c) *Page Rank* (d) *Terasort*

Figure 10. Execution time when more VMs are deployed and local deduplication is on/off



(a) *Kmeans* (b) *Nutchindexing* (c) *Page Rank* (d) *Terasort*

Figure 11. Execution time when active dataset expands and global deduplication is on/off



(a) *Kmeans* (b) *Nutchindexing* (c) *Page Rank* (d) *Terasort*

Figure 12. Execution time when more VMs are deployed and global deduplication is on/off

In summary, local deduplication cannot fully remove all the replicas, which leads to a negative performance impact when the active dataset increases. However, local deduplication can leverage the parallelism for hash computation and indexing when more nodes are deployed, which makes performance slightly better. Besides, local deduplication maintains the data availability. Global deduplication, on the other hand, eliminates all possible redundant IO requests, which leads to a positive performance impact when active dataset size expands. However, it cannot leverage parallelism for hashing when more nodes are deployed. Moreover, it decreases data availability.

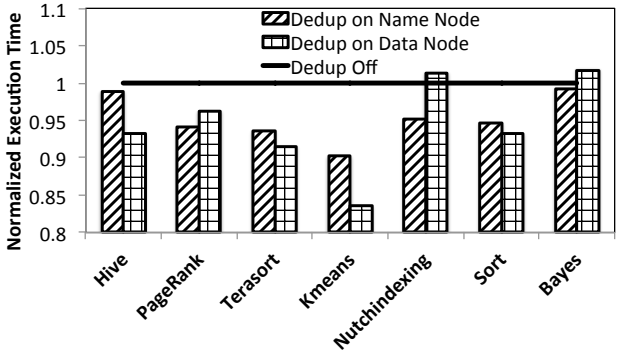


Figure 13. Execution time when deduplication is applied to name node or data node

Name node vs. Data node deduplication: The Hadoop platform divides VMs into a name node and data nodes. The name node is mainly used to hold metadata, while data nodes store actual data blocks. As we discussed in the previous subsection (Figure 6), the redundancy of the data nodes is typically larger than that on the name node. Besides, the replication mechanism largely affects the redundancy among data nodes. So, applying deduplication on data nodes saves more disk traffic but incurs more hash computation. On the other hand, metadata is important for IO performance and the size is usually smaller than actual data. Therefore, deduplication on the name node incurs less overhead of hash indexing than deduplciation on data node because of this smaller data size. However, because the volume of data being deduplicated is smaller for name nodes than for data nodes, the resulting reduction in IO traffic is less as compared to deduplication on data nodes.. As shown in Figure 13, for some benchmarks (e.g. *PageRank* and *Nutchindexing*), deduplication on metadata can save more execution time than deduplication on actual data. This is because these benchmarks do not have enough redundancy in their actual data. Instead, deduplication on metadata saves a lot of indexing and lookup time. For the benchmarks with enough redundancy (e.g. *Sort* and *Kmeans*), deduplication on data nodes saves much more traffic and gains better performance improvement than deduplication on the name node. Therefore, deduplication is better on data nodes if the number of duplicates on the dataset is large enough. Otherwise, deduplication on the name node could be an alternative option.

In ZFS, **deduplication granularity** can be changed from 1K to 128K. A small record size means deduplication occurs in fine grain, while a large record size means deduplication occurs in a coarse grain. Fine-grained deduplication can remove more duplicates and save more disk traffic than coarse-grained deduplication. However fine-grained deduplication uses more CPU cycles to perform hash indexing than the time coarse-grained deduplication takes.

Figure 14 shows the workload execution time when deduplication granularity varies. In some cases, as the deduplication granularity increases, the execution time becomes shorter since the benefit from reducing disk access overwhelms the extra hash indexing time. However, on some benchmarks (*Kmeans, Nutchindexing, and Sort*), when deduplication granularity is too small, the performance will become worse because the overhead of hash indexing becomes so high that the saving on disk accesses cannot offset it. As discussed earlier, local deduplication can be configured to hide the high hashing overhead of fine-grained deduplication, while global deduplication can be coupled with coarse-grained deduplication to remove more redundant disk accesses but still maintain low hashing overhead.

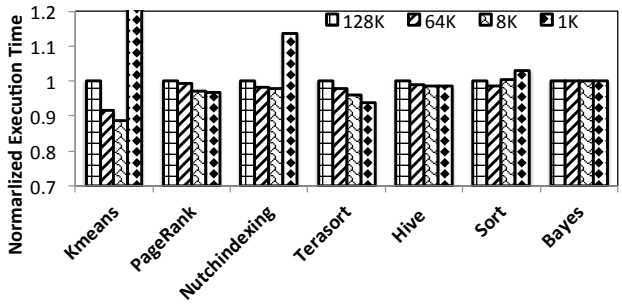
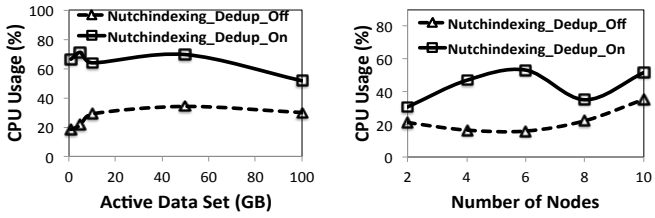


Figure 14. Normalized execution time when the deduplication granularity decreases (128K is baseline)

C. Energy Overhead Analysis

The power impact introduced by deduplication is determined by two factors: 1) extra CPU power consumption on hash computation and indexing and 2) power saving on reducing IO issued to hard drives. However, CPU is one of the largest power consumers in computer systems. Any behaviors involved in intense CPU computation will bump up power consumption. Power consumed by IO access is less significant compared to CPU. Adding hash computation in each IO request, deduplication turns every IO into extra CPU computation, which brings more power overhead. The more IO requests there are, the extra power consumption there is.



(a) Expanding active data set (b) Deploying more VMs

Figure 15. CPU usage of ZFS process (on *Nutchindexing*)

To investigate the CPU overhead of hash computation, we monitor the CPU usage for ZFS processes when deduplication is on and off.

Figure 15 (a) shows an example of the average CPU usage of ZFS processes when active dataset expands on *Nutchindexing*. As can be seen, CPU usage increases at the beginning when the dataset is small. Then it remains stable when the dataset is large. Turning on deduplication bumps up CPU usage by 37% on average.

Figure 15 (b) shows the average CPU usage of ZFS on the same case when more nodes are deployed. As the number of nodes increases, the CPU usage becomes slightly higher with some fluctuation. On average, deduplication introduces 22% CPU overhead. The other workloads follow a similar trend.

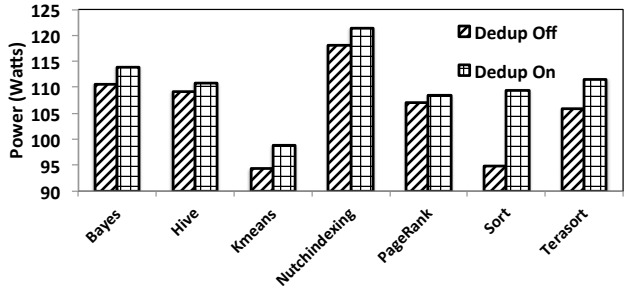
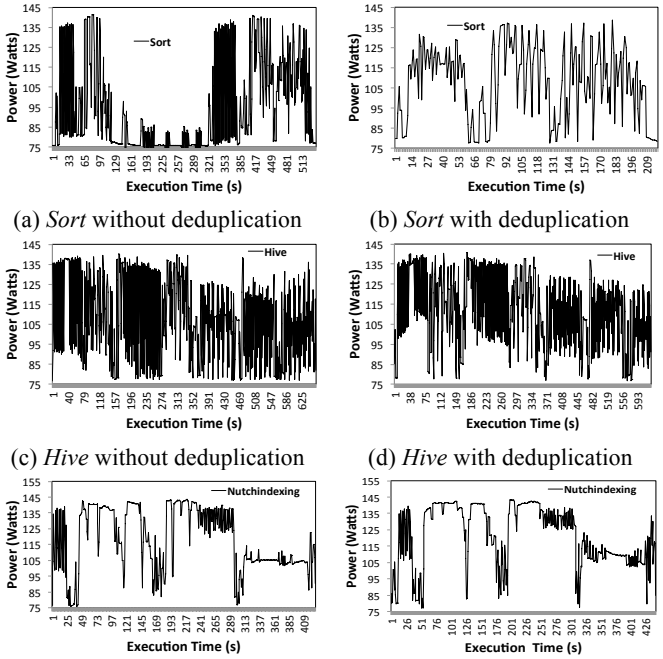


Figure 16. Power consumption when deduplication is on/off



(a) *Sort* without deduplication (b) *Sort* with deduplication

(c) *Hive* without deduplication (d) *Hive* with deduplication

(e) *Nutchindexing* without deduplication (f) *Nutchindexing* with deduplication

Figure 17. Power trace for different big data workloads

In Figure 16, we compare power consumption when deduplication is on and off. Power overhead is 10% on average when deduplication is applied.

Further looking into different benchmark types, we show the power traces during the execution time in Figure 17. Note that the power spike indicates intensive CPU computation while the power valley means waiting for IO response. The first type of benchmark has a very long IO waiting time, which is shown as Figure 17 (a). When deduplication is on, the long IO burst turn into long CPU computation (Figure 17 (b)), which generates a lot of power consumption. However, the execution time is much shorter because a lot of redundant disk accesses are avoided.

In the second type of benchmark, CPU and IO occur one after the other. Thus, the power trace exhibits significant fluctuation, which is shown as Figure 17 (c). When deduplication is applied to this type of workload, the power consumption (Figure 17 (d)) is almost the same as when deduplication is off. However, the execution time is shorter due to the removal of duplicates.

The third type of benchmarks involves a large amount of CPU computation with few IO requests (Figure 17 (e)). In this case, the power consumption will have little difference due to a small amount of hash computation. However, the execution time is slightly longer.

Energy is determined by power consumption and execution time. Although deduplication brings power overhead, the energy consumption can be less if workload execution time is short enough. The execution time, as we discussed earlier, depends on the level of redundancy and how deduplication is configured. Here, we assume the deduplication configuration is fixed. Then, the redundancy determines the execution time.

Figure 18 shows the total energy consumption for our benchmarks. For workloads with a high level of redundancy (e.g. *Kmeans*, and *Sort*), the overall energy consumption with deduplication is lower than it is without deduplication. However, for workloads with a low level of redundancy (e.g. *Terasort,* and *PageRank*), the total energy consumption with deduplication is higher than it is without deduplication. Therefore, it is wise to apply deduplication according to the level of redundancy of workloads.

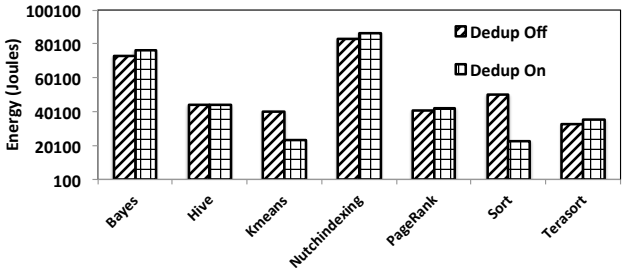
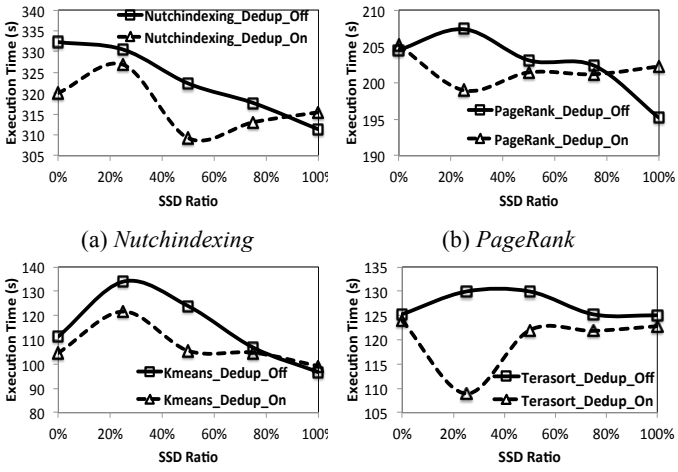


Figure 18. Energy consumption when deduplication is on/off

V. SSD STORAGE ENVIRONMENT

Recently, SSD has become more popular in data centers because of its high IO performance and low energy consumption. In our experiments, we find that SSD has 4 times higher IOPS than HDD while SSD consumes only 15% as much power as HDD. However, SSD has less capacity than traditional hard drives. Using deduplication to reduce the size of the dataset can help SSD contain more critical data, especially in big data environments. In this section, we discuss the efficiency of deduplication for an SSD environment.

Figure 19 shows the execution time with and without deduplication when we increase the ratio of SSDs in our storage environment. Since SSD has better IO performance, the IO bottleneck is gradually removed when more Hadoop nodes are assigned to the SSD. Therefore, the execution time decreases when the SSD ratio increases, as shown in Figure 19. However, the performance impact by deduplication varies in different SSD ratios. When the SSD ratio is low, turning on deduplication has better performance (at most, a 17% improvement in *Terasort*) than turning it off. In the case of a high SSD ratio, the performance improvement reduces when deduplication is on. For some benchmarks (e.g. *PageRank*), deduplication can even decrease the performance when the SSD ratio reaches 100%. This is because the benefit of reducing disk access is larger in HDD than in SSD. Since reducing disk IO does not have enough benefits in SSD, the overhead of hash indexing and computation becomes the dominant part in performance. In summary, as the SSD ratio becomes higher in the Hadoop platform, deduplication brings less performance benefit, and sometimes even reduces the performance.



(a) *Nutchindexing* (b) *PageRank*

(c) *Kmeans* (d) *Terasort*

Figure 19. Performance when SSD ratio varies

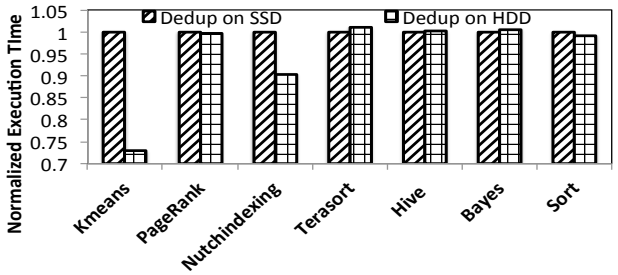


Figure 20. Normalized execution time when performing deduplication on SSD and HDD (SSD is the baseline)

To further verify this conclusion, we place 50% of the nodes on SSD and the rest on HDD. We first enable deduplication on the nodes with HDD but not on the nodes with SSD. Then, we disable the deduplication on nodes with HDD and enable it on the nodes with SSD. The results are shown in Figure 20. As can be seen, deduplication on HDD gains more performance improvement than on SSD. The performance improvement is strongly correlated with the degree of redundancy. Since SSD removes the IO traffic bottleneck, IO performance in the SSD environment is not as critical as in the HDD environment. Therefore, deduplication cannot gain enough benefit of saving IO traffic. Instead, it adds extra hash indexing overhead. In summary, using deduplication can squeeze more nodes onto SSD at the cost of performance degradation.

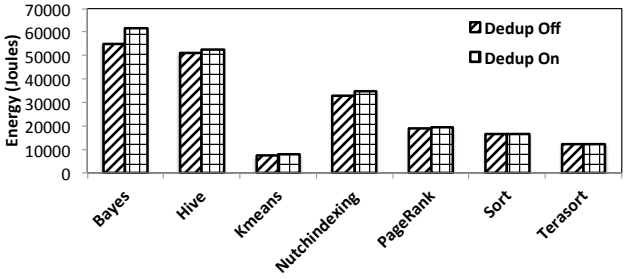


Figure 21. Energy consumption when deduplication is on/off

Figure 21 shows the energy impact brought by deduplication in the SSD environment. Compared to Figure 18, which shows the energy in the HDD environment, we find that the overall energy consumption is lower in the SSD environment. However, in the SSD environment, turning on deduplication does not reduce enough workload execution time. Considering the power overhead, the total energy consumption is higher when deduplication is applied. Therefore turning on deduplication will result in energy overhead, unless the level of redundancy is high enough that execution time can be reduced by deduplication,

VI. Related Work

Deduplication technology has been intensively studied in the past. Among those, two papers [10, 19] mainly focus on offline deduplication for backup storage. Recently, inline deduplication [4, 5, 17, 20], which targets primary workloads, has gained more interest. Those works primarily focus on performance improvement and space saving. Costa et al. [23] discussed the energy and performance tradeoff for deduplication. There are also several works on dataset characterization for deduplication: Park et al. [24] proposed three new metrics for analyzing the dataset of backup application. Tarasov et al. [18] proposed a new model for generating a realistic dataset for deduplication analysis.

In terms of Hadoop platform, two papers [16, 25] mainly focus on MapReduce computation optimization and HDFS file distribution characterization.

Our work differs from these sources in that 1) we consider using deduplication for big data workloads, which have unique attributes such as distributed computation, a large and growing dataset, and the replication mechanism; 2) we explore different deduplication layers, locations and granularity; 3) we characterize deduplication efficiency from both a performance and an energy perspective; and 4) we consider an emerging storage medium, such as SSD.

VII. Conclusions

We characterize the efficiency of using deduplication for big data storage management. We find three sources of redundancy in big data workloads: 1) deploying more VMs, 2) expanding the dataset, and 3) using the replication mechanism in HDFS. We further analyze the performance impact from two aspects: the benefit of avoiding redundant disk accesses and the overhead of hashing. Local deduplication can leverage parallelism to reduce the overhead of hashing but cannot exploit the maximum redundancy. Global deduplication can eliminate all redundant disk accesses but cannot hide the hash overhead. Deduplication on the name node is better if the level of redundancy is low in the active dataset. Fine-grained deduplication is not always better, especially when the dataset is big. Although deduplication brings extra power consumption, the energy consumption can be less if the level of redundancy is high enough to reduce execution time. The degree of redundancy is an important workload characteristic that has impact on the efficiency of deduplication in big data environment. The benefit of deduplication diminishes when more SSDs are deployed into the Hadoop system. In a pure SSD environment, deduplication can help squeeze more VMs at the cost of performance and energy overhead.

Acknowledgement

We thank Julie Brodeur and our shepherd for giving valuable advices on improving this paper. This work is supported in part by NSF grants 1320100, 1117261, 1017000, 0937869, 0916384, 0845721(CAREER), 0834288, 0811611, 0720476, by SRC grants 2008-HJ-1798, 2007-RJ-1651G, by Microsoft Research Trustworthy Computing, Safe and Scalable Multi-core Computing Awards, by NASA/Florida Space Grant Consortium FSREGP Award 16296041-Y4, and by three IBM Faculty Awards.

References

[1] Chris Eaton, Dirk Deroos, Tom Deutsch, George Lapis, and Paul Zikopoulos, Understanding Big Data, McGraw-Hill Companies, 2012

[2] Patricia Florissi, EMC And Big Data, http://cloudappsnews.com

[3] CWADN, http://www.computerweekly.com/

[4] Kiran Srinivasan, Tim Bisson, et al., iDedup: Latency-aware Inline Data Deduplication for Primary Storage, FAST, 2012

[5] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system, USENIX ATC, 2011

[6] Arkady Zaslavsky, Charith Perera, Dimitrios Georgakopoulos, Sensing as a Service and Big Data, arXiv: 1301.0159

[7] Analytics: The real-world use of big data, Executive Report, IBM Global Business Services Business Analytics and Optimization, 2012

[8] Hadoop, http://hadoop.apache.org

[9] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble, Sparse indexing: Large scale, inline deduplication using sampling and locality, FAST, 2009

[10] Atish Kathpal, Matthew John, Gaurav Makkar, Distributed Duplicate Detection in Post-Process Data De-duplication, HiPC, 2011

[11] ZFS on Linux, http://zfsonlinux.org

[12] OpenDedup, http://opendedup.org

[13] Shengsheng Huang, Jie Huang, et al., The HiBench benchmark suite: Characterization of the MapReduce-based data analysis, ICDEW, 2010

[14] Managing Meta Data for the Business: Reducing IT Redundancy, http://www.eiminstitute.org

[15] D.A. Reed, D.B. Gannon, and J.R. Larus, "Imagining the Future: Thoughts on Computing," Computer, vol. 45, 2012

[16] Xixian Han, Jianzhong Li, Efficient Skyline Computation on Big Data, IEEE Transactions on Knowledge and Data Engineering, 2012

[17] Cornel Constantinescu, Joseph Glider and David Chambliss, Mixing Deduplication and Compression on Active Data Sets, DCC, 2011

[18] Vasily Tarasov, et al., Generating Realistic Datasets for Deduplication Analysis, USENEX ATC, 2012

[19] Giridhar Appaji Nag Yasa, P. C. Nagesh, Space Savings and Design Considerations in Variable Length Deduplication, ACM SIGOPS, 2012

[20] R. Koller and R. Rangaswami. I/O deduplication: Utilizing content similarity to improve I/O performance, FAST, 2010

[21] Wattsup Meter, https://www.wattsupmeters.com

[22] Deduplication, http://en.wikipedia.org/wiki/Data_deduplication

[23] L. B. Costa, S, Al-Kiswany, et al., Assessing data deduplication trade-offs from an energy and performance perspective, IGCC, 2011

[24] Nohhyun Park, David, Lilja, Characterizing Datasets for Data Deduplication in Backup Applications, IISWC, 2011

[25] Cristina, et al., A Storage-Centric Analysis of MapReduce Workloads: File Popularity, Temporal Locality and Arrival Patterns, IISWC, 2012

[26] Keren Jin, Ethan L. Miller, The Effectiveness of Deduplication on Virtual Machine Disk Images, SYSTOR 2009

[27] K. R. Jayaram, et al., An Empirical Analysis of Similarity in Virtual Machine Images, Middleware 2011

[28] Anthony Liguori, Eric Van Hensbergen, Experiences with Content Addressable Storage and Virtual Disks, WIOV 2008