The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis

Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang

Intel China Software Center, Shanghai, P.R.China, 200241

{shengsheng.huang, jie.huang, jason.dai, tao.xie, bo.huang}@intel.com

***Abstract*— The MapReduce model is becoming prominent for the large-scale data analysis in the cloud. In this paper, we present the benchmarking, evaluation and characterization of Hadoop, an open-source implementation of MapReduce. We first introduce HiBench, a new benchmark suite for Hadoop. It consists of a set of Hadoop programs, including both synthetic micro-benchmarks and real-world Hadoop applications. We then evaluate and characterize the Hadoop framework using HiBench, in terms of speed (i.e., job running time), throughput (i.e., the number of tasks completed per minute), HDFS bandwidth, system resource (e.g., CPU, memory and I/O) utilizations, and data access patterns.**

I. INTRODUCTION

The transition to cloud computing is a disruptive trend, where most users will perform their computing work by accessing services in the cloud through the clients. There are dramatic differences between delivering software as a service in the cloud for millions to use, versus distributing software as bits for millions to run on their PCs. First and foremost, services must be highly scalable, storing and processing an explosive size of data in the cloud. For instance, big websites can generate terabytes of raw log data every day. The sheer size of its data set has led to the emergence of new cloud infrastructures, characterized by the ability to scale to thousands of nodes, fault tolerance and relaxed consistency. In particular, the MapReduce model [1] proposed by Google provides a very attractive framework for large-scale data processing in the cloud.

At a high level, a MapReduce program essentially performs a group-by-aggregation in parallel over a cluster of nodes. In the first stage a *Map* function is applied in parallel to each partition of the input data, performing the grouping operations; and in the second stage, a *Reduce* function is applied in parallel to each group produced by the first stage, performing the final aggregation. The MapReduce model allows users to easily develop data analysis programs that can be scaled to thousands of nodes, without worrying about the details of parallelism.

Consequently, having moved beyond its origins in search indexing, the MapReduce model is becoming increasingly attractive to a broad spectrum of large-scale data analysis applications, including machine learning, financial analysis and simulation, bio-informatics research, and etc. In particular, its popular open source implementation, Hadoop [2], has been used by many companies (such as Yahoo and Facebook) in production for large-scale data analysis in the cloud. In addition, many new systems built on top of Hadoop (e.g., Pig [3], Hive [4], Mahout [5] and HBase [6]) have emerged and been used by a wide range of data analysis applications.

Therefore, it is essential to quantitatively evaluate and characterize the Hadoop framework through extensive benchmarking, so as to optimize the performance and total cost of ownership of Hadoop deployments, and to understand the tradeoffs of new computer system designs for the MapReduce-based data analysis using Hadoop. Unfortunately, existing Hadoop benchmark programs (e.g., GridMix [7] and the Hive performance benchmark [9]) cannot properly evaluate the Hadoop framework due to the limitations in their representativeness and diversity. For instance, Yahoo has resorted to the simplistic sorting programs [10] to evaluate their Hadoop clusters [11].

In this paper, we first propose *HiBench*, a new, realistic and comprehensive benchmark suite for Hadoop, which consists of a set of Hadoop programs including both synthetic micro-benchmarks and real-world applications. We then evaluate and characterize the Hadoop framework using HiBench, in terms of speed (i.e., job running time), throughput (i.e., the number of tasks completed per minute), the Hadoop distributed file system [12] (HDFS, an open source implementation of GFS [13]) bandwidth, system resource utilizations, as well as data access patterns.

The main contributions of this paper are:

- The design of HiBench, a new, realistic and comprehensive benchmark suite for Hadoop, which is essential for the community to properly evaluate and characterize the Hadoop framework.
- The quantitative characterization of different workloads in HiBench, including their data access patterns, system resource utilizations and HDFS bandwidth.
- The evaluation of different deployment (both hardware and software) choices using HiBench, in terms of speed, throughput and resource utilizations.

The rest of the paper is organized as follows. We first introduce the Hadoop and its existing benchmarks in section II, and then describe the details of the HiBench suite in section III. We present the experimental results in section IV, and finally we conclude the paper in section V.

II. RELATED WORK

In this section, we first present an overview of Hadoop, and

then describe existing Hadoop benchmark programs and discuss their limitations.

A. Overview of Hadoop

In Hadoop, the input data are first partitioned into splits, and then a distinct map task is launched to process each split. The intermediate map output (that is divided into several partitions) are then optionally combined (by the *Combiner*) and stored into a temporary file on the local file system. (However, if no reduce tasks are specified, the map task directly writes its output to HDFS as the final results.)

A distinct reduce task is launched to process each partition of the map outputs. The reduce task consists of three phases (namely, the shuffle, sort and reduce phases). The shuffle phase fetches the relevant partition of all map outputs, which are merged by the sort phase. Finally the reduce phase processes the partition and writes the final results to HDFS.

B. Existing Benchmarks

In practice, several benchmark programs are often used to evaluate the Hadoop framework. However, they are mainly micro level benchmarks measuring specific Hadoop properties.

1) Sort Programs: The sorting program has been pervasively accepted as an important performance indicator of MapReduce (e.g., it is used in the original MapReduce paper [1]), because sorting is an intrinsic behaviour of the MapReduce framework. For instance, the Hadoop Sort program [10] is often used as a convenient baseline benchmark for Hadoop, and is the primary benchmark for evaluating Hadoop optimizations in Yahoo [11]. In addition, both Yahoo and Google have used TeraSort [14] (a standard sort benchmark) to evaluate their MapReduce cluster [17] [18].

2) GridMix: GridMix [7] is a synthetic benchmark in the Hadoop distribution, which intends to model the data-access patterns of a Hadoop cluster by running a mix of Hadoop jobs (including 3-stage chained MapReduce job, large data sort, reference select, indirect read, and API text sort). In practice, the total running time of GidMix is almost always dominated by the large data sort job.

3) DFSIO: The DFSIO [19] is a file system benchmark in Hadoop that tests how HDFS handles a large number of tasks performing writes or reads simultaneously. It is implemented as a Hadoop job, in which each map task *i* just opens an HDFS file to write or read sequentially, and measures the data size (*Si*) and execution time (*Ti*) of that task. There is a single reduce task followed by a post-processing task, which aggregates the performance results of all the map tasks by computing the following two results (assuming there are totally *N* map tasks):

- Average I/O rate of each map task = $\sum_{N}(S_i / T_i)/N$
- Throughput of each map task = $\sum_{N} S_i / \sum_{N} T_i$

4) Hive Performance Benchmark: The Hive performance benchmark [9] is adapted from the benchmark first proposed by Pavlo et. al [8], which is used to compare the performance of Hadoop and parallel analytical databases. The benchmark consists of five programs, the first of which is the Grep program taken from the original MapReduce paper [1]; the other four analytical queries are designed to be representative of traditional structured data analysis workloads, including selection, aggregation, join and UDF aggregation jobs [8].

C. Discussions

Benchmarking is the quantitative foundation of any computer system research. For a benchmark suite to be relevant, its workloads should represent important applications of the target system, and be diverse enough to exhibit the range of behaviour of the target applications. Unfortunately, micro level Hadoop benchmarking programs as described above are limited in their representativeness and diversity, and consequently cannot properly evaluate the Hadoop framework.

In particular, though these synthetic kernels (such as Sort, Grep or GridMix) attempt to model the behaviour of real world Hadoop applications and clusters, they do not exhibit some important characteristics of real world Hadoop applications. First, as these kernels intend to simulate the data access patterns, they contain almost no computations in the map or reduce tasks, while many real world applications (e.g. machine learning [5]) have complex computations. Second, real world Hadoop applications may have data access patterns outside the original MapReduce model that is strictly followed by these kernels; for instance, the index system of Nutch [20] (a popular open source search engine) needs to read/write a lot of temporary files on local disks in its reduce tasks.

In addition, the workloads contained in the Hive performance benchmark can only represent traditional analytical database queries; in a sense, these workloads are more about what queries that Hadoop/MapReduce is not good at (e.g., random access using index and join using partitioning key), rather than evaluating Hadoop over a broad spectrum of large-scale data analysis.

On the other hand, the HiBench suite is a more realistic and comprehensive benchmark suite for Hadoop, including not only synthetic micro-benchmarks, but also real-world Hadoop applications representative of a wider range of large-scale data analysis (e.g., search indexing and machine learning). We plan to include more workloads representing additional data analysis applications (e.g., traditional structured data analysis such as those in the Hive performance benchmark) in the future version of HiBench.

III. THE HIBENCH SUITE

In this section we describe the HiBench suite, which consists of a set of Hadoop programs including both synthetic micro-benchmarks and real-world applications. Currently the benchmark suite contains eight workloads, classified into four categories, as shown in Table I. The first seven are directly

taken from their open source implementations, and the last one is an enhanced version of the DFSIO benchmark that we have extended to evaluate the aggregated bandwidth delivered by HDFS.

TABLE I
CONSTITUENT BENCHMARKS

Category	Workload
Micro Benchmarks	Sort WordCount TeraSort
Web Search	Nutch Indexing PageRank
Machine Learning	Bayesian Classification K-means Clustering
HDFS Benchmark	EnhancedDFSIO

A. Micro Benchmarks

The *Sort* [10], *WordCount* [21] and *TeraSort* [15] programs contained in the Hadoop distribution are three popular micro-benchmarks widely used in the community, and therefore are included in the HiBench suite. Both the Sort and WordCount programs are representative of a large subset of real-world MapReduce jobs – one class of programs transforming data from one representation to another, and another class extracting a small amount of interesting data from a large data set [1].

The Sort program relies on the Hadoop framework to sort the final results, and both its map and reduce functions are simply identity functions (that is, directly emitting the input key-value pairs as output). In the WordCount program, each map task simply emits *(word, 1)* for each word in the input, the combiner computes the partial sum of each word in a map task, and the reduce tasks simply compute the final sum for each word. In HiBench, the input data of Sort and WordCount are generated using the *RandomWriter* and *RandomTextWriter* programs contained in the Hadoop distribution respectively. The TeraSort program sorts 10 billion 100-byte records generated by the *TeraGen* program [16] contained in the Hadoop distribution.

B. Web Search

The *Nutch Indexing* and *PageRank* workloads are included in HiBench to evaluate Hadoop, because they are representative of one of the most significant uses of MapReduce (i.e., large-scale search indexing systems).

The Nutch Indexing workload is the indexing sub-system of Nutch [20], a popular open-source (Apache) search engine. We have used the crawler sub-system in Nutch to crawl an in-house Wikipedia mirror and generated about 2.4 million web pages as the input of this workload, which gathers the web links from input and converts the link information into inverted index files. The map function of NutchIndexing is simply an identity function, and the reduce function generates the inverted index files using the Lucene [22] plug-in.

The PageRank workload is taken from a test case in SmartFrog [23], an open source framework for managing distributed systems. It is an open source implementation of the page-rank algorithm [24], a link analysis algorithm used widely in web search engines that calculates the ranks of web pages according to the number of reference links. The PageRank workload consists of a chain of Hadoop jobs, among which several jobs are iterated until the converge condition is satisfied. We have used the Wikipedia page-to-page link database [25] as the input of this workload.

C. Machine Learning

The *Bayesian Classification* and *K-means Clustering* implementations contained in Mahout [5], an open-source (Apache) machine learning library built on top of Hadoop, are included in HiBench, because they are representative of one of another important uses of MapReduce (i.e., large-scale machine learning).

The Bayesian Classification workload implements the trainer part of Naive Bayesian [26] (a popular classification algorithm for knowledge discovery and data mining). It consists of four chained Hadoop jobs, which extract the terms using the *N-Gram* algorithm [27] from input web page text, calculate the *Tf-Idf* (term frequency–inverse document frequency) [28] for each term, and perform the weighting and normalization. The input of this benchmark is extracted from a subset of the Wikipedia dump [29]. The Wikipedia dump file is first split using the built-in *WikipediaXmlSplitter* in Mahout, and then prepared into text samples using the built-in *WikipediaDatasetCreator* in Mahout. The text samples are finally distributed into several files as the input of the benchmark.

The K-means Clustering workload implements K-means [30] (a well-known clustering algorithm for knowledge discovery and data mining). Its input is a set of samples, and each sample is represented as a numerical d-dimensional vector. The workload first computes the centroid of each cluster by running one Hadoop job iteratively, until different iterations converge or the maximum number of iterations (set by the user) is reached. After that, it runs a clustering job that assigns each sample to a cluster. We have developed a random data generator using statistic distributions to generate the workload input.

D. HDFS Benchmark

We have extended the *DFSIO* program contained in the Hadoop distribution to compute the aggregated bandwidth delivered by HDFS. As described in Section II, the original DFSIO program only computes the average I/O rate and throughput of each map task. Because DFSIO is tightly coupled to the Hadoop framework, it is not straightforward how to properly sum up the I/O rate or throughout of all the map tasks if some map tasks are delayed, re-tried or speculatively executed by the Hadoop framework.

The *Enhanced DFSIO* workload included in HiBench computes the aggregated bandwidth by first disabling the speculative execution of map tasks, and then sampling the

number of bytes read/written at fixed time intervals in each map task. Consequently, when a map task is complete, a series of samples is obtained, with each sample presented in the format of *(map id, timestamp, total bytes read/written).*

Assuming that the time on each server is synchronized (e.g., using a time server), during the reduce and post-processing stage, the samples of each map task are linear interpolated and re-sampled at a fixed plot rate, so as to align the time series between map tasks. The re-sampled points at the same timestamp of all map tasks are then summed up to compute the total number of bytes read/written by all the map tasks at that timestamp, and finally the aggregated throughput curve can be computed using that number (as illustrated in Fig. 1).

As shown in Fig. 1, the aggregated throughput curve has a warm-up period and a cool-down period where map tasks are launching up and shutting down respectively. Between these two periods, there is a steady period where the aggregated throughput values are stable across different time slots. Therefore, the Enhanced DFSIO workload computes the aggregated HDFS throughput by averaging the throughput value of each time slot in the steady period. In Enhanced DFSIO, when the number of concurrent map tasks at a time slot is above a specified percentage (e.g., 50% is used in our benchmarking) of the total map task slots in the Hadoop cluster, the slot is considered to be in the steady period.

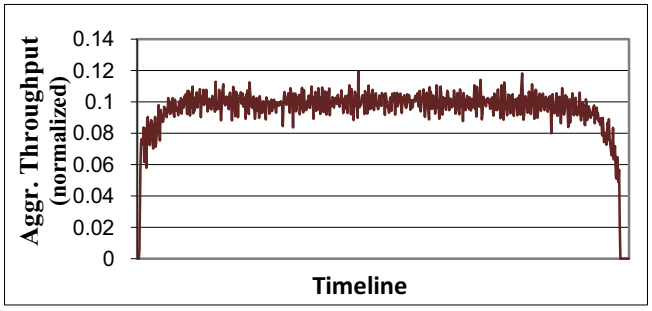


Fig. 1 Normalized aggregated throughput curve of Enhanced DFSIO (write)

IV. Evaluation and Characterization

In this section, we present the experimental results, which evaluate and characterize the Hadoop framework using HiBench, in terms of speed (i.e., job running time), throughput (i.e., the number of tasks completed per minute), aggregated HDFS bandwidth, system resource (e.g., CPU, memory and I/O) utilizations, and data access patterns. The baseline Hadoop cluster that we have used consists of one master (running JobTracker/NameNode) and several slaves (running TaskTracker/DataNode), all connected to a gigabit Ethernet switch. The detailed hardware and software configurations of the servers are shown in Table II.

TABLE II
Hardware And Software Configurations of Slaves

Processor	Dual-socket quad-core Intel® Xeon® X5500 processor
Hard disk drive	5 SATA 7200RPM HDDs (one for system and log files, and the other 4 for HDFS files)
Memory	16GB ECC DRAM
Network	1 Gigabit Ethernet NIC
Operation system	Redhat Enterprise Linux 5.2
JVM	jdk1.6.0_02
Hadoop version	Hadoop 0.19.1 with patch 5191[33]

A. Hadoop Workload Characterization

In this experiment, there are four slaves in the Hadoop

TABLE III
Workload Data Access Patterns

Workload	Job	Job/ Map Input	Map Output (Combiner Input)	Shuffle Data (Combiner Output)	Job/ Reduce Output
Sort	Sort	60G	60G	60G (no combiner)	60G
Word Count	WordCount	60G	85G	3.99G	1.6G
TeraSort	TeraSort	1T	139G [b]	139G (no combiner)	1T
Nutch Indexing	Nutch Indexing	8.4G [b]	26G	26G (no combiner)	6.3G
PageRank[a]	Dangling-Pages	1.21G	81.7K	672B	30B
	Update-Ranks	1.21G	5.3G	5.3G (no combiner)	1.21G
	SortRanks	1.21G	86M	86M (no combiner)	167M
Bayesian Classification-tion	Feature	1.8G	52G	39.7G	35G
	Tfldf	35G	26G	22.8G	13.3G
	Weight-Summer	13.3G	16.8G	8.7G	8.2G
	Theta-Normalizer	13.3G	5.6G	163K	8.6K
K-means Clustering [c]	Centroid-Computing	66G	67G	303K	4.6K
	Clustering	66G	66.3G	no combiner	no reducer

[a]PageRank runs 3 chained jobs iteratively. The data access patterns vary slightly between different iterations and the data here show one such iteration.
[b]The data size is the compressed result.
[c]We synthesized 160 million samples (each with 20 dimensions), to be clustered into 10 groups as input of K-Means Clustering workload.

TABLE IV
Workload Execution Time Ratio

Workload	Job	Avg. Map Task vs. Avg. Reduce Task Execution Time Ratio[a]	Map Stage vs. Reduce Stage Execution Time Ratio[b]
Sort	Sort	2.00%	33.36%
WordCount	WordCount	55.98%	94.67%
TeraSort	TeraSort	0.26%	70.71%
Nutch Index	Nutch Indexing	6.22%	31.16%
PageRank	DanglingPages	15.36%	124.12%
	UpdateRanks	49.90%	66.81%
	SortRanks	5.74%	49.50%
Bayesian Classicification	Feature	14.33%	57.93%
	Tfldf	7.86%	75.57%
	WeightSummer	19.32%	72.64%
	ThetaNormalizer	29.37%	87.75%
K-means Clustering	CentroidComputing	5.67%	102.58%
	Clustering	no reduce tasks	no reduce tasks

[a]Average Map Task execution time is defined as the average of execution time of all Map Tasks. Average Reduce Task is defined similarly.
[b]The Map Stage is defined as the period between when the first Map Task starts and when the last Map Task ends. The Reduce Stage is defined similarly.

cluster. Table III summarizes the data access patterns of MapReduce model for all workloads (except Enhanced DFSIO that has no relevant input and output) in the experiment, in terms of the sizes of job input, map output, shuffle data and job output; and Table IV summarizes the ratio of map execution time vs. reduce execution time of each workload. In addition, Fig. 2 – 9 shows the timeline-based CPU, memory and disk I/O utilizations of each workload. (Network I/O is never a bottleneck in the experiment, since the scale of the cluster used is small and all the nodes are arranged in one rack. Therefore the network utilization is not presented in this part.)

Since the Sort workload transforms data from one representation to another, the shuffle data and the job output of Sort are of the same size as the job input, as shown in Table III. Consequently, the Sort workload is mostly I/O bound, having moderate CPU utilization and heavy disk I/O, as shown in Fig. 2. In addition, considering the large amount of shuffle data, it is expected to have network I/O bottlenecks in the shuffle stage when running on large (multi-rack) clusters.

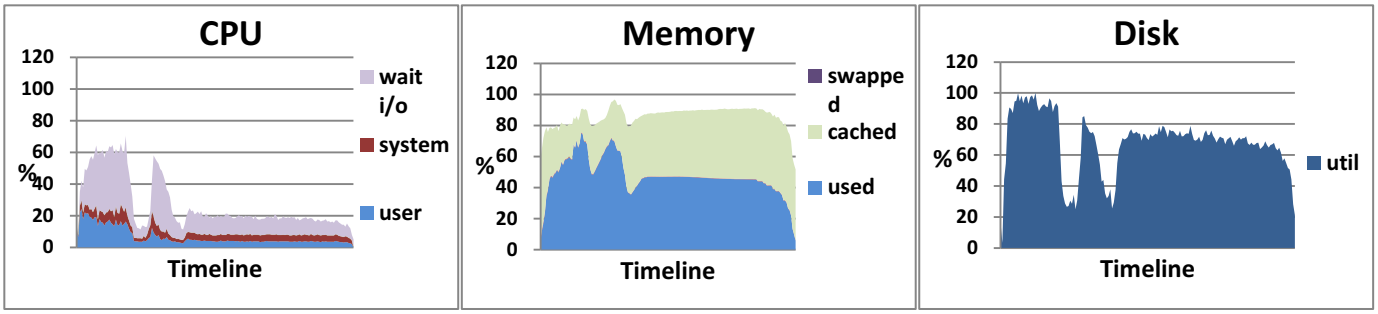


Fig. 2 System Utilization of Sort

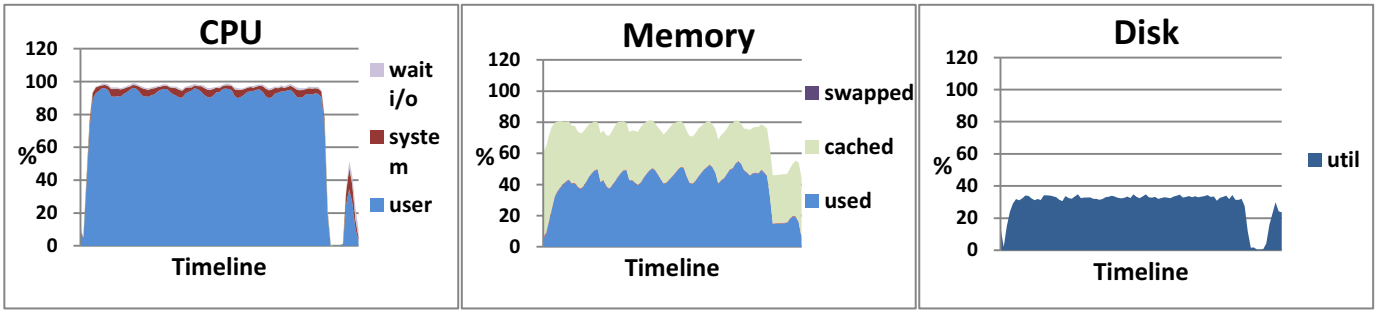


Fig. 3 System Utilization of WordCount

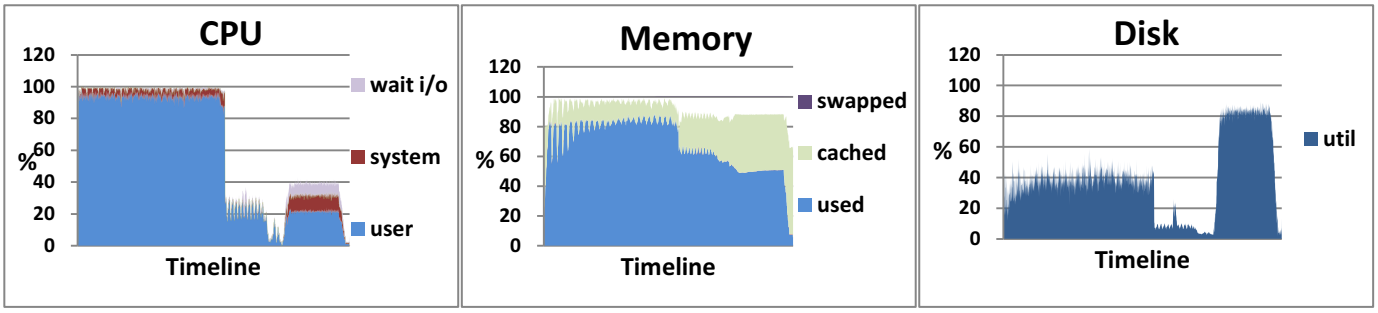


Fig. 4 System Utilization of TeraSort

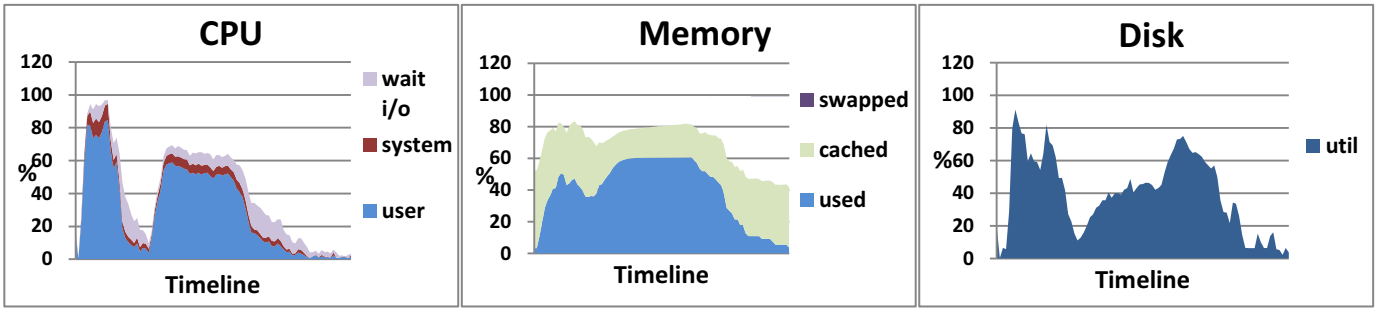


Fig. 5 System Utilization of Nutch Indexing

On the other hand, since the WordCount workload extracts a small amount of interesting data from a large data set, the shuffle data and the job output of WordCount are much smaller than the job input, as shown in Table III. Consequently, the WordCount workload is mostly CPU bound (especially during the map stage), having high CPU utilization, light disk/network I/O, as shown in Fig. 3. In addition, its behavior is expected to remain somewhat the same even on larger clusters.

In essence, the TeraSort workload is similar to Sort and therefore is I/O bound in nature. However, we have compressed its shuffle data (i.e., map output) in the experiment so as to minimize the disk and network I/O during shuffle, as shown in Table III. Consequently, TeraSort have very high CPU utilization and moderate disk I/O during the map stage and shuffle phases, and moderate CPU utilization and heavy disk I/O during the reduce phases, as shown in Fig. 4.

In the experiment, the NutchIndexing workload first decompresses the crawl data to about 26GB intermediate

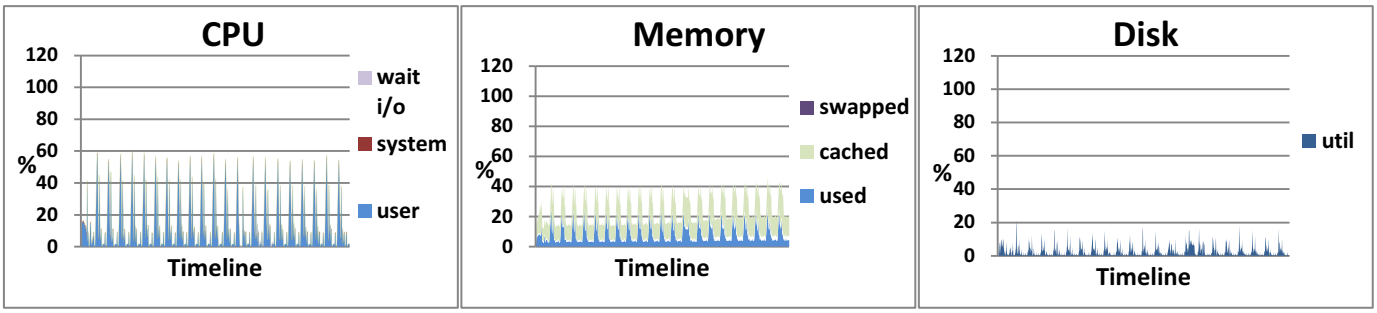


Fig. 6 System Utilization of PageRank

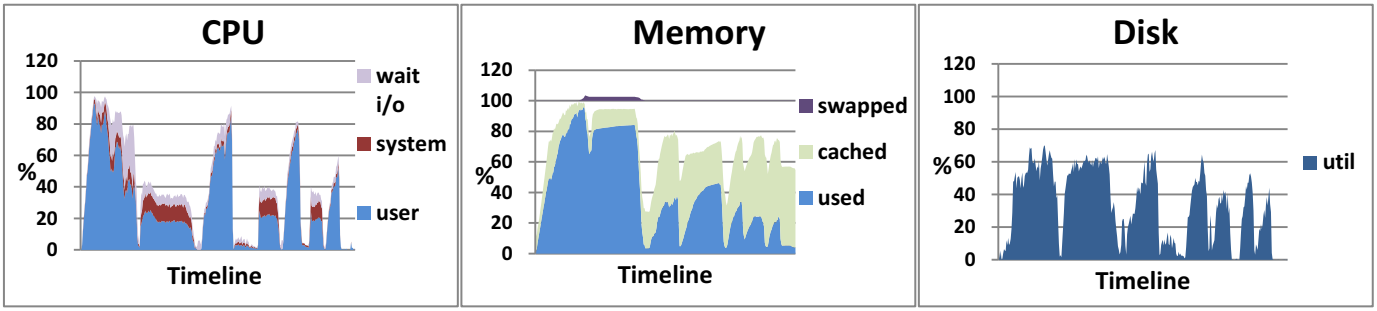


Fig. 7 System Utilization of Mahout Bayesian Classification

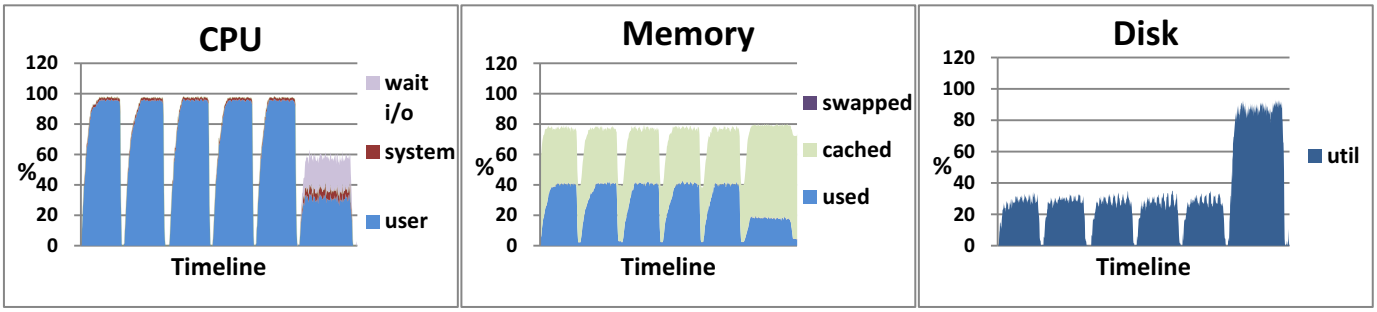


Fig. 8 System Utilization of Mahout K-Means Clustering

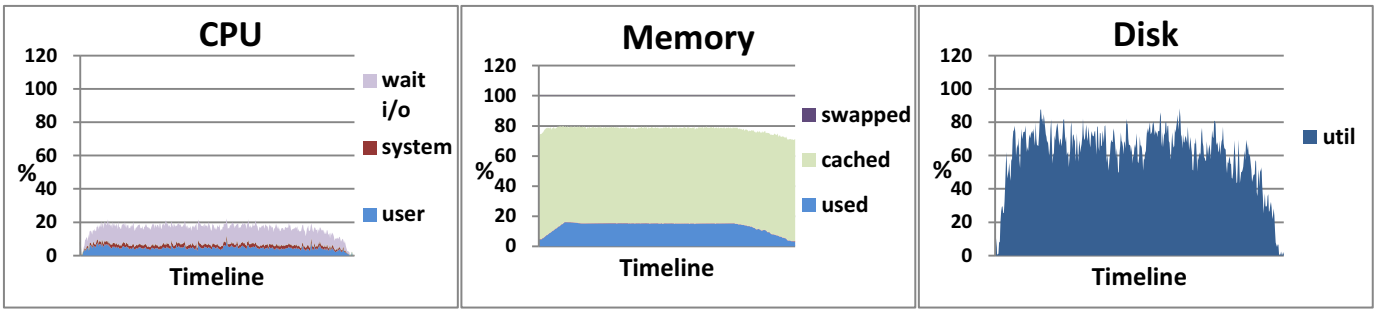


Fig. 9 System Utilization of DFSIO (read)

results in the map tasks, and then the reduce tasks convert them to about 6.3GB inverted index files. As shown in Fig. 5, NutchIndexing is CPU bound during map stage and more disk I/O bound (with about 60% CPU utilizations) in the reduce stage.

The PageRank workload spends most of its time on the iterations of several jobs, and these jobs are generally CPU bound, with low to medium disk I/O and memory utilizations, as shown in Fig. 6.

The Bayesian Classification workload contains four chained Hadoop jobs, and the first job is the most time consuming (taking about half of the total running time in the experiment). As shown in Fig. 7, the four jobs are all mostly disk I/O bound, except that the map tasks of the first job also have high (over 80%) CPU utilizations.

The K-means Clustering workload spends most of its time on job iterations for the centroid computation. In the experiment, we have set the max iteration limit to five so as to make sure that the test finishes in a reasonable time period. As shown in Fig. 8, the centroid computation in K-means Clustering is mostly CPU bound, because its combiner swallows most of the map outputs and consequently the shuffle data are relatively small, as shown in Table III. On the other hand, the clustering job is I/O bound, mostly due to the output to HDFS of the map tasks (as there are no reduce tasks in the clustering job).

Finally, since the Enhanced DFSIO workload only tests the HDFS throughput, it is completely disk I/O bound. Fig. 9 shows the resource utilizations of Enhanced DFSIO read workload (the utilizations of the write workload are similar).

As described in the above characterization, the Hadoop workload usually has very heavy disk and network I/O, unless the map output size is (or can be combined to be) very small (e.g., as in the case of WordCount and K-means Clustering). The best performance (total running time) of Hadoop workloads is usually obtained by accurately estimating the size of the map output, shuffle data and reduce input data, and properly allocating memory buffers to prevent multiple spilling (to disk) of those data [32]. In the experiment, we have tuned the workloads to keep these data in memory as much as possible, and consequently, as shown in Fig. 2 – 9, all of them have high memory consumptions (over 80% memory utilization), except PageRank that performs a lot of computations on a relatively small data set.

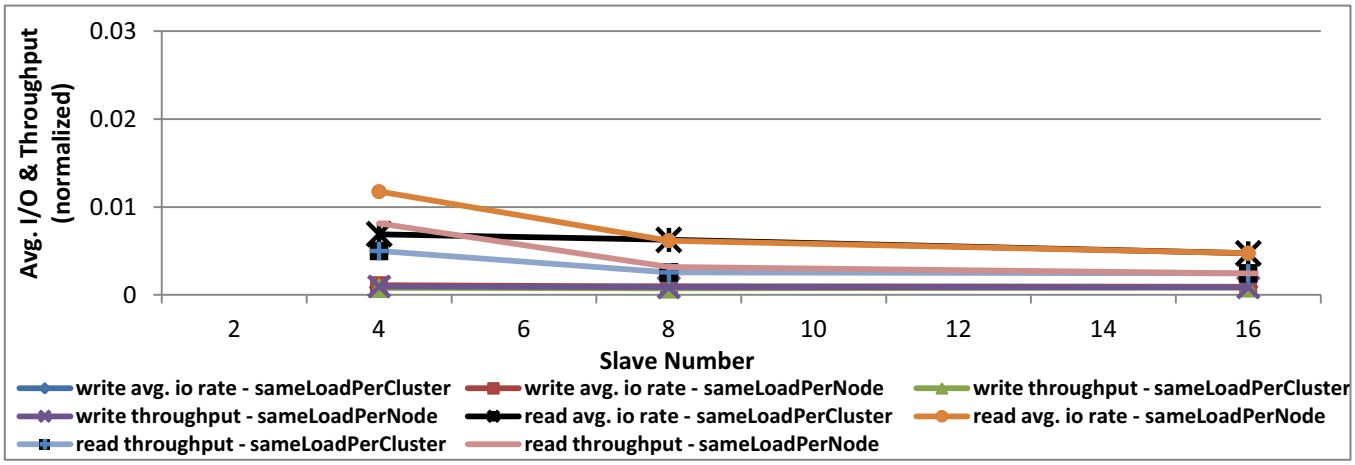


Fig. 10 DFSIO: normalized average I/O rate & throughput as function of slave number

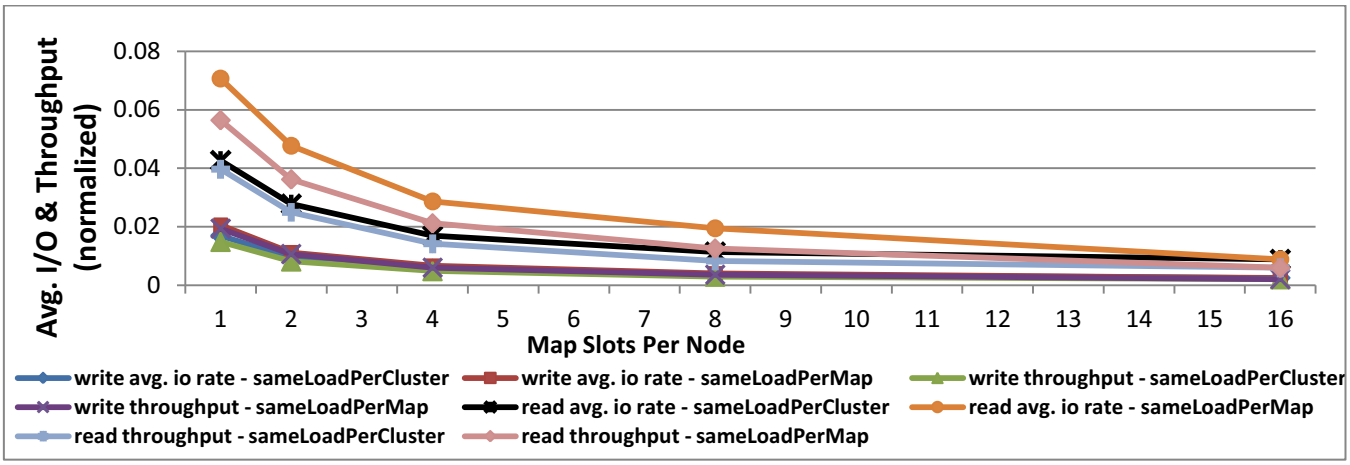


Fig. 11 DFSIO: normalized average I/O rate & throughput as function of map slots per node

TABLE V
TEST CONFIGURATION OF ENHANCED DFSIO BENCHMARKING

Workload Configuration	Size of Each File to Be Read/Written (MB)	Total Number of Files to Be Read/Written
Write – sameLoadPerCluster	360	256
Write – sameLoadPerNode	360	16 x number of slave nodes
Write – sameLoadPerMap	360	16 x number of map slots per node
Read - sameLoadPerCluster	720	256
Read - sameLoadPerNode	720	16 x number of slave nodes
Read - sameLoadPerMap	720	16 x number of map slots per node

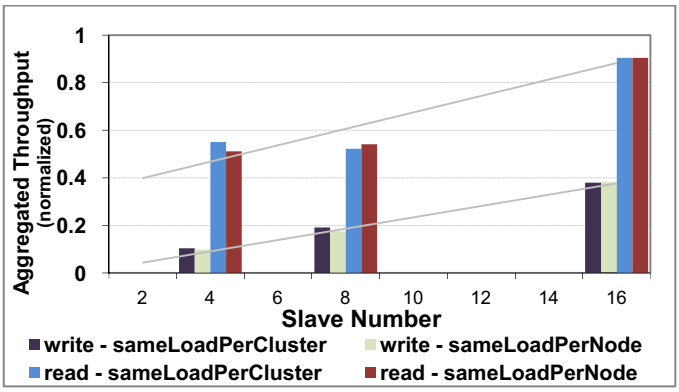


Fig. 12 Enhanced DFSIO: normalized aggregated throughputs as function of slave number

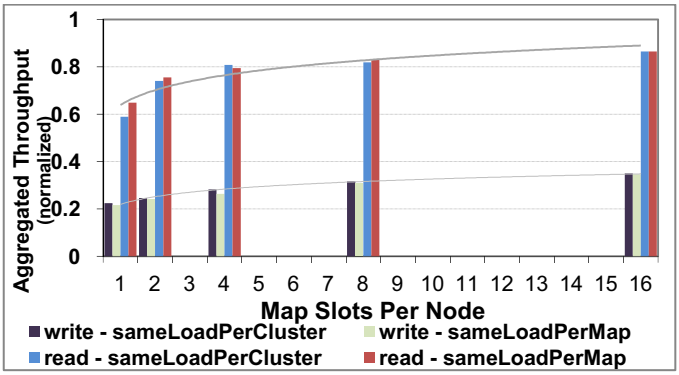


Fig. 13 Enhanced DFSIO: normalized aggregated throughputs as function of map slots per node

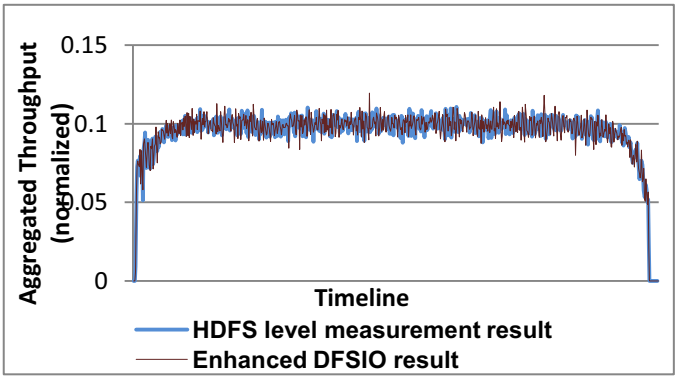


Fig. 14 Normalized aggregated throughput of Enhanced DFSIO vs. HDFS level measure result

B. HDFS Bandwidth

In this experiment, we examine the results of Enhanced DFSIO workload and the original DFSIO benchmark, using the *same-load-per-cluster*, *same-load-per-node* and *same-load-per-map* configurations (as shown in Table V), and for different numbers of slaves and different *map slots* (i.e. the maximum number of parallel map tasks that can run on each server). We have normalized the y-axis values of all the figures in this section (i.e. Fig. 10-14) as fraction of a certain value for clearer representation and easier comparison.

Fig. 10 and 11 show the average I/O rate & throughput of DFSIO for different numbers of slaves (with 4 *map slots* per node) in the cluster and different numbers of *map slots* per node (in a 16-slave cluster) respectively. Similarly, Fig. 12 and 13 show the aggregated throughput of Enhanced DFSIO for different numbers of slaves in the cluster and *map slots* per node respectively. As shown in Fig. 10 and 11, the DFSIO average I/O rate and throughput results remain the same or decrease slightly as the number of slaves or *map slots* increases; the aggregated throughput of HDFS actually increases as the number of slaves increases, as shown in Figure 12 and 13. Therefore, DFSIO is not appropriate for the evaluation of the aggregated bandwidth delivered by HDFS, which is the design goal of HDFS and GFS. As shown in Figure 12, the aggregated write throughput of HDFS scales almost linearly with the number of slaves in the experiment; on the other hand, the aggregated read throughput in the 4-slave cluster is about the same as or even slightly higher than that in the 8-slave cluster. The reason for that poor scalability of read is, since the HDFS replication factor is set to 3 in the experiment (i.e. each HDFS file block is replicated on 3 nodes), the possibility that a HDFS read request is served from the local node in the 4-slave cluster is much higher than that in the 8-node cluster. However, the benefit of locality quickly diminishes as the number of slaves in the cluster increases, and therefore it can be expected that the aggregated read throughput scales linearly with the number of slaves when the scale of cluster is large enough. In addition, as shown in Figure 13, the aggregated throughput will also increase with number of *map slots* in the 16-node cluster until the cluster is saturated.

Both workloads measure the number of bytes read/written at the application level; however, in HDFS write operations are buffered and performed asynchronously. Fig. 14 compares the aggregated throughput measured at the application level by Enhanced DFSIO and that measured at the HDFS level (i.e., when the bytes to be written are acknowledged by HDFS), and those two results are almost the same. Hence, the aggregated throughput measured at the application level by the Enhanced DFSIO can be used to properly evaluate the aggregated throughput of the HDFS cluster.

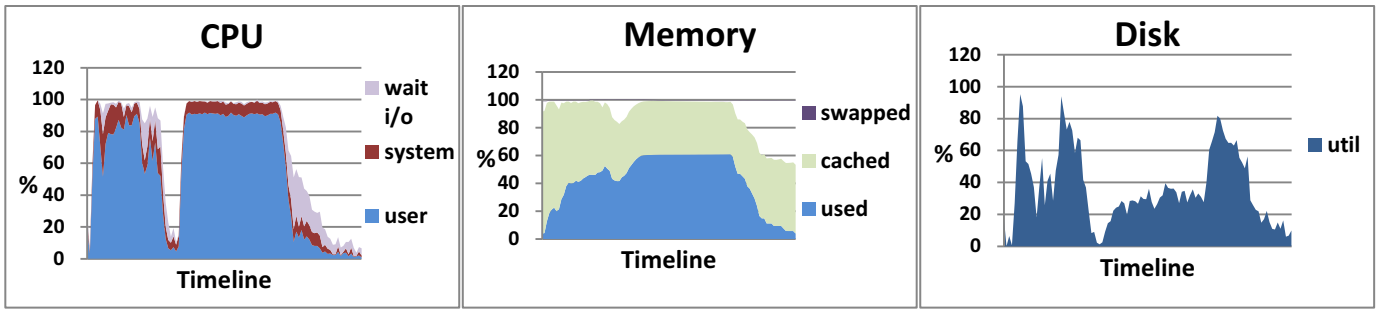


Fig. 17 NutchIndexing in the old cluster

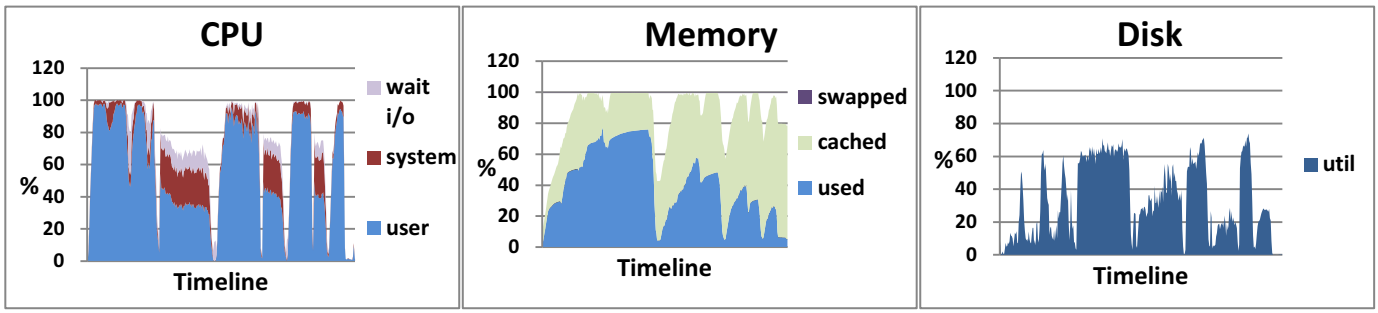


Fig. 18 Bayesian Classification in the old cluster

C. Evaluation of Hadoop Deployments

1) Comparison of processor platforms: In this experiment, we use HiBench to compare the baseline cluster with 4 slaves (referred to as the *current cluster*) and a 4-slave *old cluster* that is configured using the same configuration as shown in Table II, except that the slave uses the Xeon X5400 processor (older than the Xeon X5500 processor). Fig. 15 compares the speed (i.e., job running time) of the Sort and WordCount workloads in these two clusters, and Fig. 16 compares the throughput (i.e., the number of tasks completed per minute measured when the Hadoop cluster is at 100% utilization processing multiple Hadoop jobs).

In terms of speed, the current cluster has 48% and 66% speedup over the old cluster for Sort and WordCount respectively; on the other hand, in terms of throughput, the current cluster has 86% and 65% speedup over the old cluster for Sort and WordCount respectively. Therefore, both speed and throughput are important performance metrics in evaluating Hadoop deployments.

In addition, because the *current cluster* is more powerful than the *old cluster* in processors, the Nutch Indexing and Bayesian Classification workloads have different

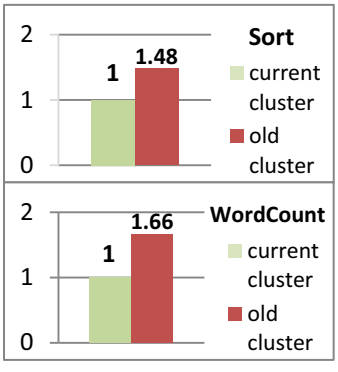


Fig. 15 Normalized Sort, WordCount job running time (lower is better)

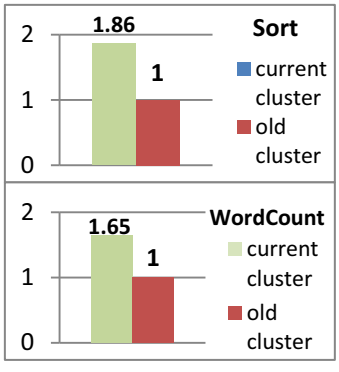
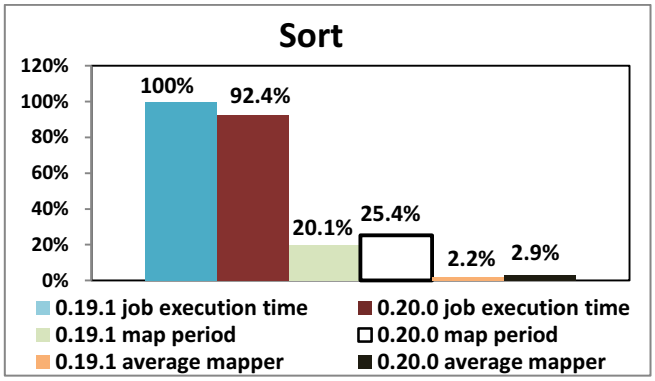


Fig. 16 Normalized Sort, WordCount throughput (higher is better)



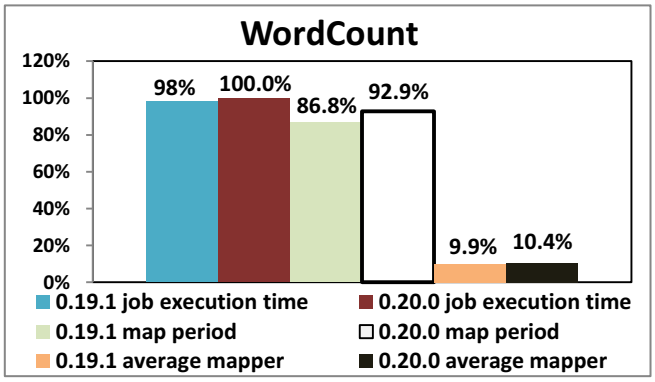


Fig. 19 Normalized job execution time & map period & average mapper comparison

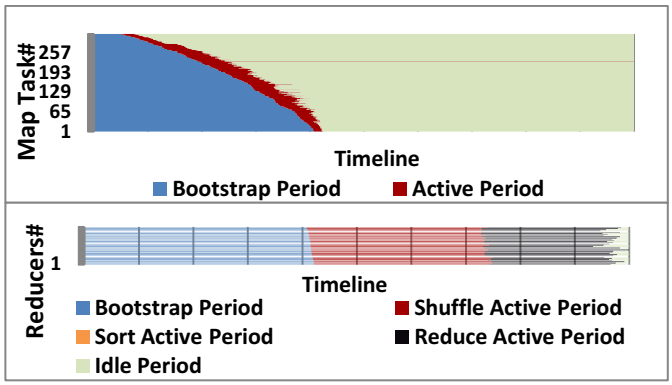


Fig. 20 Timeline-based MapReduce breakdown on Hadoop 0.19.1 for the 2nd job of the Bayesian Classification workload

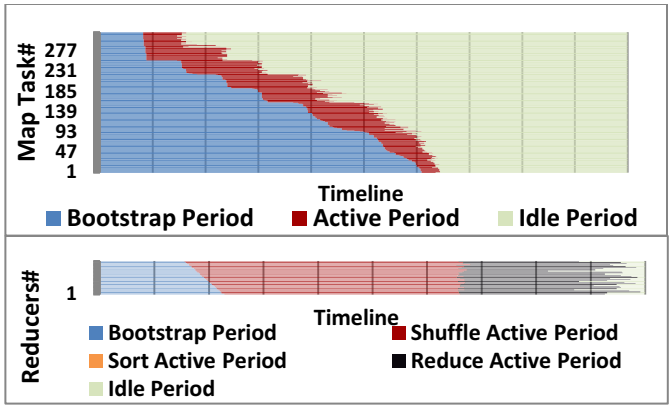


Fig. 21 Timeline-based MapReduce breakdown on Hadoop 0.20.0 for the 2nd job of the Bayesian Classification workload

characteristics on these two clusters. NutchIndexing, which is CPU bound during the map stage and more disk I/O bound during the reduce stage in the *current cluster* (as shown in Fig. 5), becomes completely CPU bound during both stages in the *old cluster* (as shown in Fig. 17). All the four jobs in Bayesian Classification are mostly disk I/O bound in the *current cluster* (as shown in Fig. 6), while become CPU bound during the map stage in the *old cluster* (as shown in Fig. 18).

2) Comparison of Hadoop versions: In this experiment, we use HiBench to compare Hadoop 0.19.1 and Hadoop 0.20.0 (using Yahoo! distribution of Hadoop [31]) in the baseline cluster with 4 slaves. Fig. 19 shows the running time of the Sort and WordCount workloads using Hadoop 0.19.1 and 0.20.0. As shown in Fig. 19, the Map stage in v0.20.0 is slower (and uses more memory) than v0.19.1, but the overall job running time is about the same or even slightly faster in v0.20.0. This is mostly due to the improved task scheduling (i.e., multi-task assignment) in v0.20.0.

Fig. 20 shows the timeline-based execution breakdown for the second Hadoop job in Bayesian Classification using Hadoop 0.19.1, in which most of reduce tasks cannot start until after all the map tasks are done. This is because in Hadoop 0.19.1, the JobTracker can only schedule one task in a heartbeat (every 5 seconds by default) to a TaskTracker and map tasks have higher priority. When there are a lot of map tasks that finish very fast (e.g., less than 5s), the scheduling of reduce tasks is significantly delayed. On the other hand, in Hadoop 0.20.0, the JobTracker can schedule multiple tasks in a heartbeat and this performance issue is no long there (as shown in Fig. 21).

V. Conclusion and Future Work

In this paper, we first introduce HiBench, a new, realistic and comprehensive benchmark suite for Hadoop, which consists of a set of Hadoop programs including both synthetic micro-benchmarks and real-world applications (e.g., web search and machine learning). We then use experimental results to evaluate and characterize the Hadoop framework using HiBench, in terms of speed (i.e., job running time), throughput (i.e., the number of tasks completed per minute), HDFS bandwidth, system resource (e.g., CPU, memory and I/O) utilizations, and data access patterns.

The HiBench suite is essential for the community to properly evaluate and characterize Hadoop, because its workloads not only represent a wide range of large-scale data analysis using Hadoop, but also exhibit very diverse behaviors in terms of data access patterns and resource utilizations, as shown by the experimental results.

We plan to continuously evolve the HiBench suite to represent an even broader spectrum of large-scale data analysis using the MapReduce model, such as financial analysis and bio-informatics research. In addition, we plan to quantitatively evaluate more Hadoop deployment choices using HiBench, including the scale of the cluster, the capacity of memory and disks, and the impacts of processor micro-architecture.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *USENIX OSDI*, December, 2004.
[2] Hadoop homepage. http://hadoop.apache.org/
[3] Pig homepage. http://hadoop.apache.org/pig/
[4] Hive homepage. http://hadoop.apache.org/hive/
[5] Mahout homepage. http://lucene.apache.org/mahout/
[6] HBase homepage. http://hadoop.apache.org/hbase/
[7] GridMix program. Available in Hadoop source distribution: src/benchmarks/gridmix.
[8] A. Pavlo, A. Rasin, S. Madden, M. Stonebraker, D. DeWitt, E. Paulson, L. Shrinivas, and D. J. Abadi. "A Comparison of Approaches to Large-Scale Data Analysis", *SIGMOD*, June, 2009
[9] Y. Jia and Z. Shao. "A Benchmark for Hive, PIG and Hadoop", Available: http://issues.apache.org/jira/browse/HIVE-396
[10] Sort program. Available in Hadoop source distribution: src/examples/org/apache/hadoop/examples/sort
[11] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. "Improving MapReduce Performance in Heterogeneous Environments", *OSDI'08*, December, 2008.
[12] HDFS homepage. http://hadoop.apache.org/hdfs/
[13] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," *19th ACM Symposium on Operating Systems Principles*, October, 2003.
[14] TeraSort. http://sortbenchmark.org/
[15] Hadoop TeraSort program. Available in Hadoop source distribution since 0.19 version: src/examples/org/apache/hadoop/examples/terasort
[16] TeraGen program. Available in Hadoop source distribution since 0.19 version: src/examples/org/apache/hadoop/examples/terasort/TeraGen
[17] O. O'Malley and A. C. Murthy, "Winning a 60 Second Dash with a Yellow Elephant". Available: http://sortbenchmark.org/Yahoo2009.pdf

[18] "Sorting 1PB with MapReduce", Available: http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html
[19] DFSIO program. Available in Hadoop source distribution: src/test/org/apache/hadoop/fs/TestDFSIO
[20] Nutch homepage. http://lucene.apache.org/nutch/
[21] WordCount program. Available in Hadoop source distribution: src/examples/org/apache/hadoop/ examples/WordCount
[22] Lucene homepage. http://lucene.apache.org
[23] SmarFrog project homepage. http://www.smartfrog.org
[24] P. Castagna, "Having fun with PageRank and MapReduce," Hadoop User Group UK talk. Available: http://static.last.fm/johan/huguk-20090414/paolo_castagna-pagerank.pdf
[25] H. Haselgrove, "Using the Wikipedia page-to-page link database," Available: http://users.on.net/~henry/home/wikipedia.htm
[26] Mahout Naïve Bayesian. http://cwiki.apache.org/MAHOUT/naivebayes.html
[27] N-Gram. http://en.wikipedia.org/wiki/N-gram
[28] Tf-Idf. http://en.wikipedia.org/wiki/Tf%E2%80%93idf
[29] Wikipedia Dump. http://en.wikipedia.org/wiki/index.php?curid=68321
[30] Mahout K-means. http://cwiki.apache.org/MAHOUT/k-means.html
[31] Yahoo! distribution of Hadoop. Available: http://developer.yahoo.com/hadoop/distribution/
[32] Nurcan Coskun, "Optimizing Hadoop Deployments", Hadoop World 2009 Presentation.
[33] Hadoop-5191. Available: http://issues.apache.org/jira/browse/HADOOP-5191.