Workload Characterization on a Production Hadoop Cluster: A Case Study on Taobao

Zujie Ren, Xianghua Xu, Jian Wan
School of Computer Science and Technology
Hangzhou Dianzi University
Hangzhou, China
Email:{renzj,xhxu,wanjian}@hdu.edu.cn

Weisong Shi
Department of Computer Science
Wayne State University
Detroit, USA
Email:weisong@wayne.edu

Min Zhou
Taobao, Inc.
Hangzhou, China
Email:zhouchen.zm@taobao.com

***Abstract*—MapReduce is becoming the state-of-the-art computing paradigm for processing large-scale datasets on a large cluster with tens or thousands of nodes. It has been widely used in various fields such as e-commerce, Web search, social networks, and scientific computation. Understanding the characteristics of MapReduce workloads is the key to achieving better configuration decisions and improving the system throughput. However, workload characterization of MapReduce, especially in a large-scale production environment, has not been well studied yet.**

To gain insight on MapReduce workloads, we collected a two-week workload trace from a 2,000-node Hadoop cluster at Taobao, which is the biggest online e-commerce enterprise in Asia, ranked 14th in the world as reported by Alexa. The workload trace covered 912,157 jobs, logged from Dec. 4 to Dec. 20, 2011. We characterized the workload at the granularity of job and task, respectively and concluded with a set of interesting observations. The results of workload characterization are representative and generally consistent with data platforms for e-commerce websites, which can help other researchers and engineers understand the performance and job characteristics of Hadoop in their production environments. In addition, we use these job analysis statistics to derive several implications for potential performance optimization solutions.

***Keywords*-workload characterization; Hadoop; MapReduce;**

I. Introduction

With the rapid growth of data volume in many enterprises, large-scale data processing becomes a challenging issue, attracting plenty of attention in both academic and industrial fields. The MapReduce framework [1], proposed by Google, provides a highly scalable solution for processing large-scale data. The fundamental concept of MapReduce is to distribute data among many nodes and process the data in a parallel manner. Hadoop, an open-source implementation of the MapReduce framework [2], can easily scale out to thousands of nodes and work with petabyte data. Due to its high scalability and performance, Hadoop has gained much popularity. High visibility organizations, such as Yahoo, Facebook, Twitter, and many research groups adopt Hadoop to run their data-intensive jobs.

Understanding the characteristics of MapReduce workloads is the key to achieving better configuration decisions and improving the system throughput. Hadoop operators can apply this knowledge to optimize the scheduling policies and to allocate resources more effectively under diverse workloads. However, as yet, workload characterization of MapReduce, especially in a large-scale production environment, has not been well studied yet.

To gain insight on MapReduce workloads, we collected a two-week workload trace from a 2,000-node Hadoop cluster at Taobao, Inc. This Hadoop cluster is named Yunti, which is an internal data platform for processing petabyte-level business data. The trace comprises over a two-week period of data, covering approximately one million jobs executed by 2,000 nodes. The trace was gathered by standard log tools in Hadoop. Jobs in Yunti are diverse, ranging from multi-hour traffic statistics analysis to several-second ad-hoc queries. Most of the jobs are automatically executed during the mid-night, and they should be completely finished within a pre-defined duration. These jobs are referred to as *periodic jobs*. Some jobs are submitted by the staff and run temporarily. These jobs are referred as to *temporary jobs*.

Our analysis reveals a range of workload characterization and provides some direct implications. We believe that workload characterization study is useful for helping Hadoop operators identify system bottleneck and figure out solutions for optimizing performance. A deep understanding of the characteristics of jobs running in Hadoop contributes in determining factors affecting jobs completion times, contributing to an optimal system configuration policy.

The contributions of this paper are listed as follows.

- We collect a two-week MapReduce workload trace from a 2,000-node production Hadoop cluster. The trace includes 912,157 jobs, which are representative and common in a data platform for an e-commerce website. This workload trace is a beneficial complement of a current public workload repository.
- We conduct a comprehensive analysis of the workload trace at the granularity of job and task, respectively. The main observations and their direct implications from this analysis are presented in Table I. These findings can help other researchers and engineers better understand the performance and job characteristics of Hadoop in their production environments.
- Based on the observational results, we derive several indirect implications for potential performance optimization solutions, including pre-scheduling periodic jobs,

dynamic resource allocations, job failure handling, the data skew issue, high availability improvement, and schedulability analysis.

The rest of this paper is organized as follows. Section II provides a brief introduction of MapReduce and Hadoop, and then gives an overview of the Yunti cluster. Section III discusses the summary of trace, including the information extracted and the format of the logs. A detailed analysis of these logs at the granularity of job and task is described in Section IV. Section V shows the resource utilization of the Yunti cluster in terms of CPU, memory, slot, and network. The indirect implications for performance optimization on Hadoop clusters are presented in Section VI. Section VII discusses related work and we conclude this paper in Section VIII.

II. Background

To facilitate the understanding of MapReduce workload analysis in this paper, this section first gives a brief overview of the MapReduce framework and its implementation - Hadoop, with an illustration about how a MapReduce job executes in Hadoop. For detailed information, we refer to [1], [2]. Then, we describe the architecture of the Yunti cluster and its implementation details.

A. Overview of MapReduce and Hadoop

MapReduce is a parallel and distributed framework proposed by Google for processing large datasets. The fundamental concept of MapReduce is to distribute data among many nodes and process the data in a parallel manner. A MapReduce job consists of two phases: map and reduce. In the map phase, input data is divided into independent chunks and each chunk is assigned to one of the compute nodes. Each chunk is processed by a map task in a completely parallel manner, yielding intermediate key-value pairs data. This intermediate data is the output of the map-phase, as well as the input for the reduce phase. In the reduce phase, the input data is aggregated, summarized, filtered, or combined in some way, forming the final output. Typically, both the input and the output of the job are stored in a distributed file system. The MapReduce processing pattern remains the same, while the specific functions defined for the map and reduce phase change to fit specific problems.

Hadoop is an open source implementation of the MapReduce framework. In Hadoop, MapReduce logical architecture follows a master-slaves model. The main components of MapReduce are comprised of a single master node called JobTracker, and many worker nodes called TaskTracker. JobTracker is responsible for accepting jobs from clients, decomposing the jobs into a set of tasks, monitoring and scheduling them, and re-executing the failed tasks. Each TaskTracker possesses a number of slots, and each slot can host a task belonging to any job. They receive tasks from the JobTracker and execute them. The JobTracker uses a heartbeat mechanism to monitor the status of TaskTrackers and the processes of running tasks. When some slots in a TaskTracker are idle, new tasks will be assigned at the next heartbeat.

Besides the component of MapReduce, Hadoop comprises another fundamental component: HDFS (Hadoop Distributed File System). HDFS is an implementation of the Google File System [3] used to store both the input data for map tasks and the output data of the reduce tasks. Similar to the MapReduce implementation, HDFS architecture also adopts a master-slaves model. HDFS consists of NameNode and DataNodes. NameNode is a master server that manages the file system namespace and regulates access to files by clients. DataNodes is a set of nodes responsible for data storage and management.

In HDFS, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode performs file system namespace operations including opening, closing, and renaming files and directories. It also determines the mapping of data blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file systems clients. The DataNodes also perform block creation, deletion, and replication as required under instructions from the NameNode.

Typically, TaskTrackers and DataNodes run on the same group of nodes. This property allows the job scheduler to effectively schedule tasks on the nodes where the data is already in place, thereby contributing to decrease data movements.

B. Architecture of the Yunti Cluster

The Yunti cluster is an internal data platform for processing petabyte-level business data mostly derived from the e-commerce website of ”www.taobao.com”. Up to December 2011, the Yunti cluster consists of over 2,000 heterogeneous nodes. The total volume of data stored in the Yunti has exceeded 25PB, and the data grow rapidly everyday, with the speed of 30TB per day. The goal of the Yunti cluster is to provide multi-user businesses with large-scale data analysis service for some online applications. Yunti is built on Hadoop, with some slight modifications. Figure 1 presents the overview of the Yunti cluster.

The data resident in the Yunti cluster is comprised of the replicas of online databases, logs of Web servers (e.g., Apache and ngnix), crawled pages, etc. The Yunti cluster employs some data synchronization tools such as DBSync and TimeTunnel, to maintain synchronization with the corresponding online server periodically. The synchronization cycles are optimized for different data sources, dependent on the synchronization cost and requirements of various applications.

Yunti is built based on Hadoop 0.19, and the job scheduler uses fair scheduler with a slight modification. Yunti assigns each user (internal staff) group a slot quota to control the slots' use. Each job can only use the slots belonging to its group. If there are no idle slots for a user group, the jobs in the group have to wait for idle slots. If the job has a high priority, the scheduler will kill some tasks and give the job some slots. In addition, the number of concurrent jobs in Yunti is also limited with a pre-defined upper bound.

The jobs executed on the Yunti cluster are diverse. Over fifty-six groups, including over five hundred users, submit

TABLE I
SUMMARY OF OBSERVATIONS AND IMPLICATIONS.

Observations	Implications	Sections
O1: Job arrival rate follows a fairly constant pattern.	Node scheduling mechanism based on the job arrival pattern will be feasible and useful for saving power.	IV-A1
O2: Slot preemptions caused by high-priority jobs are frequent.	Optimizing a preemption scheme should be a priority.	IV-A2
O3: The completion time of successful, failed, and killed jobs follow the log-normal distribution.	Jobs with diverse completion times bring extra challenges for job scheduling.	IV-A3
O4: 80% of jobs write less than 64MB data on HDFS.	Small files are a big problem in Hadoop. Improving the efficiency of small files access will be very beneficial.	IV-A4
O5: The task counts of jobs in Yunti are diverse. Small jobs constitute the majority, and medium and large jobs account for a considerable part.	Hadoop schedulers need to be improved for scheduling small jobs.	IV-B1
O6: Both map task and reduce task duration time follow a log-normal distribution.	Conventional simulation with uniform-duration tasks is unrealistic.	IV-B2
O7: The execution time inequity for map task is much worse than that of reduce tasks.	Data skew on the map phase is much worse due to some skewed map operations, such as with the table JOIN operations.	IV-B3
O8: The fair scheduler used in Hadoop does a very well on data locality.	To reduce data movement by further improving data locality will be not a satisfactory method.	IV-B4
O9: Data read and write workloads on HDFS are high. It might be beneficial to optimize the implementation of MapReduce on Hadoop.	Employing a distributed shared memory (DSM) system would be a good solution to decrease the read and write workloads on HDFS.	V-D
O10: Network transfer generated on shuffle operation and remote HDFS access causes high network load. Network will become a system bottleneck with the data volume growth.	Efficient data placement and scheduling policies for reducing network load are needed.	V-E

about sixty thousand jobs to the Yunti platform for everyday. These jobs include multi-hour collaborate filtering computations, as well as several-second ad-hoc queries. These jobs are originated by numerous applications, such as commodities recommendations, traffic statistics and advertise delivery system.

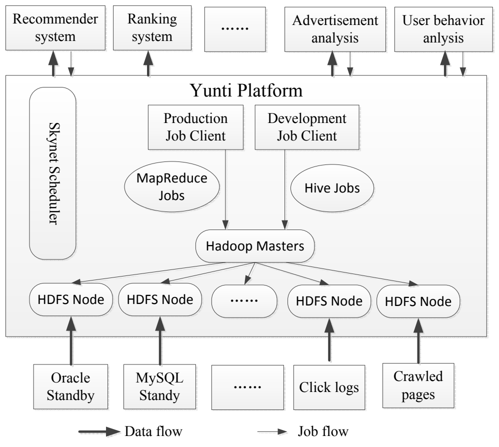


Fig. 1. Overview of the Yunti cluster.

III. Description of Trace

The trace was collected over a two-week period from Dec. 4 to Dec. 20, 2011. During this period, over 912,157 MapReduce jobs were generated by 572 different users belonging to fifty-six groups. Most of the jobs were submitted automatically and started execution at a consistent time each day. Table II gives an overview of the dataset. On average, each job consisted of 42 map tasks and 12 reduce tasks running on 19 nodes. Maximum nodes allocated to a job is 489. According to the job's final status, all jobs are separated into three groups:

- **Successful jobs**: those jobs executed successfully.

TABLE II
SUMMARY OF THE WORKLOAD TRACE.

Log Period	Dec 4, 2011 - Dec 20, 2011
Number of groups	56
Number of users	572
Number of jobs	912157
Successful jobs	902387
Failed jobs	5672
Killed jobs	3648
Average maps per job	42
Average reduces per job	12
Average nodes per job	19
Maximum nodes per job	489
Average job duration	350s
Maximum job duration	16h24m

- **Failed jobs**: those jobs aborted or canceled due to unhandled exceptions.
- **Killed jobs**: those jobs killed by the Yunti operators.

Among the 912,157 jobs, the number of successful job was 902,387 (about 99%), and the number of failed and killed jobs was 5,672 (0.6%) and 3,648 (0.4%), respectively. The trace related to job execution is collected by standard tools in Hadoop. However, raw trace data is enormous in volume, approximately 20GB. We extracted job characteristics using Taobao's internal monitoring tools and saved them in two relational tables: (1) Table JobStat: this table stores the job features. Each row represents a job and contains the following fields: JobID (a unique job identifier), job status (successful, failed or killed), job submission time, job launch time, job finish time, the number of map tasks, the number of reduce tasks, total duration of map tasks, total duration of reduce tasks, read/write bytes on HDFS, read/write bytes on local disks. (2)Table TaskStat, this table stores task features. Each row represents one task and contains the following fields: TaskID (a unique task identifier), JobID, task type (map or reduce), task status, task start time, and task finish time.

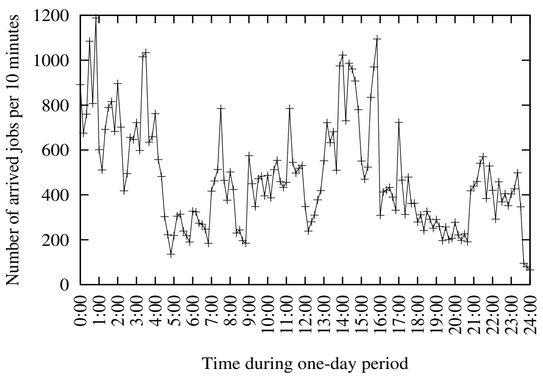


Fig. 2. Arriving jobs count for each 10-minute interval during one day.

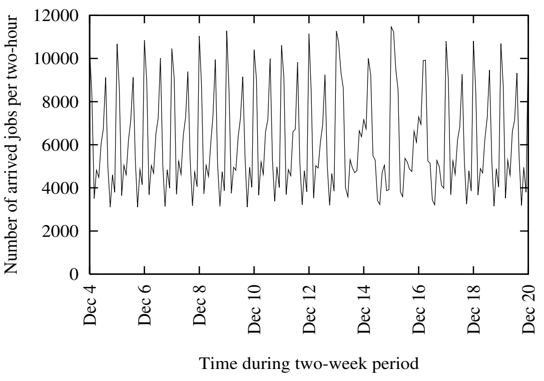


Fig. 3. Arriving jobs count for each two-hour interval during two weeks.

In addition to workload trace, we observed the resource utilization statistics of the Yunti cluster in terms of CPU, memory, slots, and storage during the same period. These statistics were collected by Ganglia [4] with 30-second sample intervals.

IV. Trace Analysis

In this section, we first describe job statistics, including job arrival pattern, job completion times, data size, and wave behavior. Then, we present task statistics, including task count, task duration, task equity, and data locality. Our empirical observations are italicized at the end of the corresponding paragraph.

A. Job Statistics

1) Job Arrival Rate: Figure 2 shows the number of arriving jobs per 10-minute interval during a one-day period. The workload reaches the first daily-peak from 1:00 am to 4:00 am, and reaches the second peak from 1:00 pm to 4:00 pm. The maximum number of arrived jobs within a 10-minute interval exceeds 1,100. The first workload peak is mainly formed by periodic jobs. All periodic jobs are pre-defined by application developers and submitted automatically. These jobs are arranged to be executed in the early morning of everyday. Most of the periodic jobs remain about the same, generating periodic reports. In addition, the second one is formed by temporary jobs. Temporary jobs are defined temporarily and submitted by application developers manually, just as ad-hoc queries. For most temporary jobs, these workloads are also referred to as MapReduce with Interactive Analysis (MIA) workloads [5]. During the work hours, many application developers will submit temporary jobs, forming the second workload peak. Note that most temporary jobs are executed only once.

Figure 3 depicts the arriving job count per two-hour intervals during two weeks. As shown in the figure, the job arrival pattern expressed every day is very similar, except for the bursty jobs arrival on Dec. 13 and 14. Bursty jobs were generated by a big promotion activity held on Dec. 12, especially for investigating the effect of the promotion activity. Therefore, some additional jobs were submitted in the subsequent days following a big promotion. **Observation 1.** *Job arrival rate follows a relatively-constant daily pattern. This observation implies that a resource scheduling mechanism based on the job arrival rate pattern will be feasible and beneficial for saving electrical power, i.e., some nodes work while the others sleep if the workload is low.*

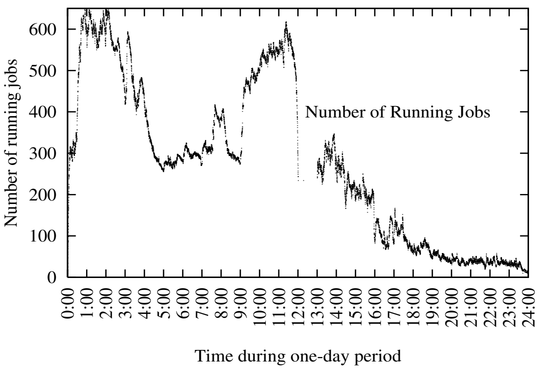


Fig. 4. Running jobs count, sampled per 5-second interval.

2) Jobs Count on Running Status: Figure 4 shows the changes of concurrently running jobs count loaded on the Yunti cluster during a one-day period. It was sampled per 5 seconds. The number of running jobs exceeds 600 at the peak time, and falls down to approximately 300 at 5:00 am. A couple of hours later, the second workload peak time appears at 9:00 am, when the company staff begins to work.

The feature of two workload peaks in number of running jobs is similar to the job arrival rates. However, if we compare Figure 2 and 4, we can obtain an interesting observation: the second peak of arriving jobs and running jobs count do not form at a same time interval. This observation can be explained by the difference of job duration time. According to Little's law [6] in the theory of queue, the long-term average number of running jobs is equal to the long-term average job arrival rate multiplied by the average job execution time. After detailed investigation, it is verified that temporary jobs arrived in the morning contains more long jobs than the jobs arrived in the afternoon, which confirms this observation.

Actually, the maximum count of concurrent running jobs has been limited to 300, but it exceeds the upper limit. That is

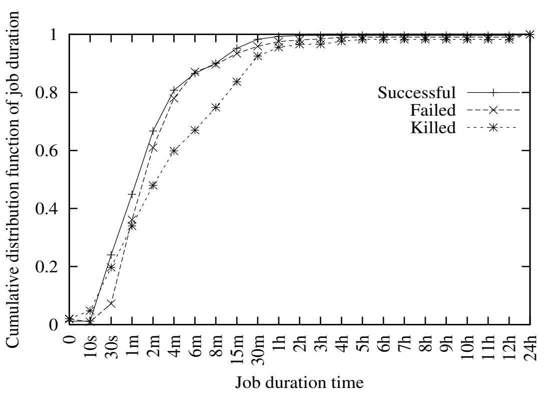


Fig. 5. Cumulative distribution job completion time during a two-week period.

because the high-priority jobs arrival will induce the scheduler to kill some tasks and preempt their occupied slots if there are not enough idle slots. Therefore, some running jobs may have to switch to be an awaiting-state. **Observation 2.** *As depicted in the Figure 4, we can infer that preemption is frequent, and optimizing a preemption scheme should be a priority.*

3) Job Completion Times: Figure 5 presents the cumulative distribution of completion time of successful jobs, failed jobs, and killed jobs during the two-week period. Failed jobs are jobs that were canceled due to task failure. Killed jobs are jobs that were killed by cluster operators due to slots preemption or some other reasons. The completion time of a job is calculated as the time scope from the first task starts until the last task ends.

Intuitively, the jobs completion time follows a long-tail distribution. More than 80% of jobs finished within 4 minutes. Maximum job completion time reaches 16 hours and 24 minutes. Failed jobs consume less time than killed jobs. Ninety percent of failed jobs were canceled within 8 minutes, while 90% of killed jobs were stopped within 25 minutes. When one task fails more than a pre-defined time (4 in default), the corresponding job will fail. Common causes of task failures include JVM failure, task execution time-out, and hardware faults, etc. Killed jobs are the jobs canceled by the Hadoop operators because these jobs are not well-designed. Most of the job failures are caused by failure times that exceed a pre-defined value. **Observation 3.** *After measuring the goodness of fit of the job completion times against the exponential, Weibull, and log-normal distributions, we find that the log-normal distribution is the best fitting distribution for the successful, failed, and killed jobs.* Successful jobs follows a log-normal distribution with $\mu = 1.55$ and $\delta = 0.42$.

4) Job Data Size: Job execution processes bring forth plenty of data read/write operations at various stages of MapReduce. The total bytes of data read/writes involve two parts, *HDFS read/write bytes* and *local read/write bytes*, which represent the total bytes of data read or written on HDFS and local disks, respectively. HDFS read/writes bytes involve input data read from HDFS and output data written on HDFS. The data access on HDFS is guided by NameNode and carried out by one of the HDFS DataNodes. Local read/write bytes involve the intermediate data generated in the MapReduce stages. The intermediate data are stored in local disks directly.

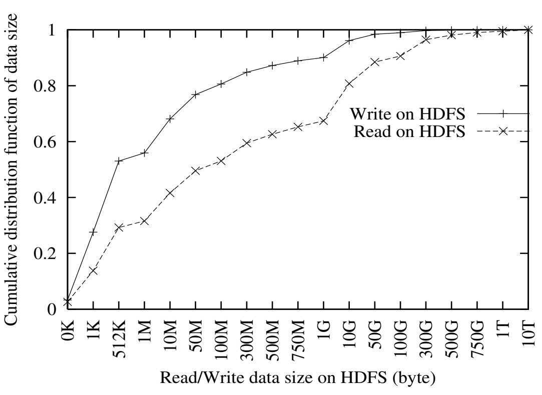


Fig. 6. Cumulative distribution of HDFS read/write bytes per job.

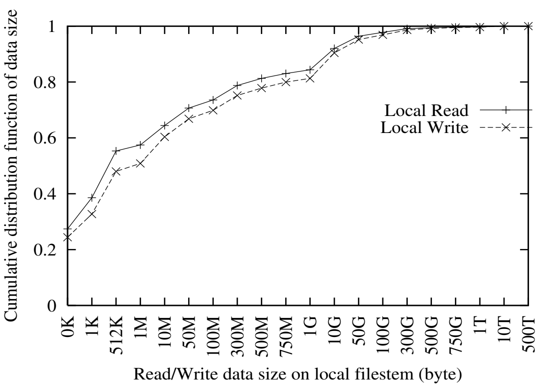


Fig. 7. Distribution of local read/write bytes per job.

Figures 6 and 7 show the cumulative distribution of HDFS and local read/write bytes per job. As shown in these two figures, 80% of jobs write under 64MB data to HDFS, and 50% of jobs read under 64MB data from HDFS. As expected, data read size is much larger than data write size on HDFS, because input data is much larger than output data for most jobs. While for IO on local disks, 80% of jobs read and write under 1G data on local disks, data write size is slightly more than data read size. This result is also expected because the intermediate data generated during the job execution process is stored on local disks. **Observation 4.** *Eighty percent of jobs write small files (#size <64MB) to HDFS. Some previous works reported that a large number of small files will depress the efficiency of NameNode [7], [8]. Therefore, efficiency for small files storage on HDFS is a significant and urgent issue to be solved.*

5) Job Waves: Job wave is defined as the times for allocating slots for a job [9]. In Hadoop, each map (or reduce) task is executed in a map (or reduce) slot. A slot is a unit of computing resources allocated for the corresponding task. A TaskTracker is configured to provide a fixed number of slots, and each slot can host one task independently. A common configuration for multi-core TaskTracker is to set two slots for each core.

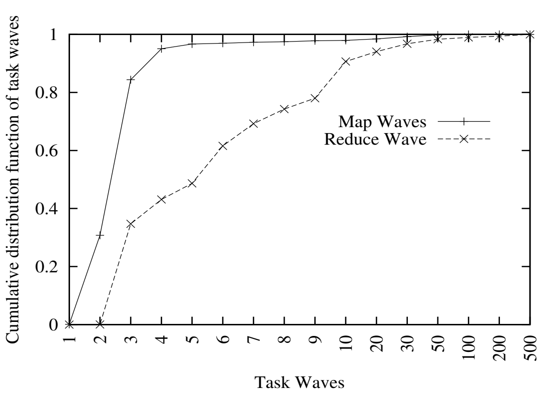


Fig. 8. Cumulative distribution of waves for tasks.

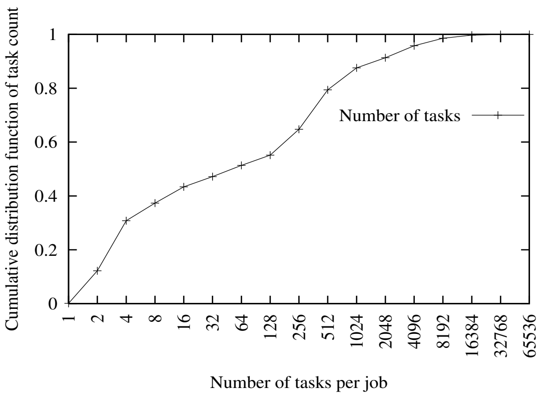


Fig. 9. Cumulative distribution of task count per job.

If the slots required by a job are more than the available idle slots, the job scheduler will first assign the available slots to let a portion of tasks be loaded, forming the first "wave" in the job execution. Then, the other tasks are scheduled when some slots are idle, forming the second and third waves. We refer to this property of slots being allocated by stages as *wave behavior* of job scheduling. From the perspective of an individual job, the number of waves is also dependent on the job size (the number of tasks in the job) and the shortage of available slots. Generally speaking, if the job size is large, the number of waves tends to be high; if the available slots are not abundant, the number of waves will be increased.

Figure 8 shows the cumulative distributions of the number of waves for map and reduce tasks in user jobs in our dataset. Given a job, the number of waves is calculated by collecting the start time of all tasks and counting the distinct start times. The number of distinct start times is the number of waves. It is observed that most jobs have relatively few map waves (more than 95% of jobs have under 5 waves of maps), while 4% of jobs have more than five waves for map tasks. Ninety-five percent of jobs have fewer than 50 waves for reduce tasks, and 80% of jobs have fewer than 10 waves. This result can be explained by the following reason: the starts of reduce tasks execution rely on map tasks being completed to a pre-defined degree, leading the start times of reduce tasks be much more dispersed.

B. Task Statistics

This subsection provides trace analysis at the granularity of task, including the cumulative distribution of task count per job, task duration, task equity, and data locality.

1) Task Count per Job: Figure 9 depicts the cumulative distribution of task count per job. Over 40% of jobs are divided into less than 10 tasks, while about 50% of jobs' task count ranges from 10 to 2,000. The largest job consists of 91,798 tasks, and the smallest job contains only one map task. Job distribution grouped by the size is also presented in Table III.

Small jobs pose a big challenge to Hadoop. Hadoop was originally designed for processing large jobs. It is reported that the data locality will be impacted for small jobs [10], [11]. **Observation 5.** *The task counts of jobs in Yunti are diverse. Small jobs account for the majority, and medium and large jobs are a considerable part of the jobs analyzed. Thus, Hadoop schedulers need to be improved for scheduling small jobs.*

TABLE III
JOB SIZE STATISTICS.

Task count within a job	number of jobs	Percentage
<10	370,062	40.57%
10-500	358,751	39.33%
500-2,000	109,733	12.03%
>2,000	73,611	8.07%

2) Task Duration: Figure 10 shows the cumulative distribution of the map and reduce task duration. More than 50% of tasks are executed for less than 10 seconds. The longest task lasts 20 minutes and 48s. **Observation 6.** *Both map task and reduce task duration time follow a log-normal distribution.* Map task duration follows a log-normal distribution with $\mu = 1.95$ and $\delta = 1.67$, and reduce task duration follows a log-normal distribution with $\mu = 3.52$ and $\delta = 1.56$.

To further summarize the task duration distribution, we classify all tasks into three groups according to their duration: second-tasks, minute-tasks and hour-tasks. Second-tasks mean those tasks executed for several seconds up to a fixed duration, i.e., 120 seconds. Second-tasks cover small task and temporal jobs. Minute-tasks include those tasks executed for more than 2 minutes but less than one hour. Hour-tasks refer to those

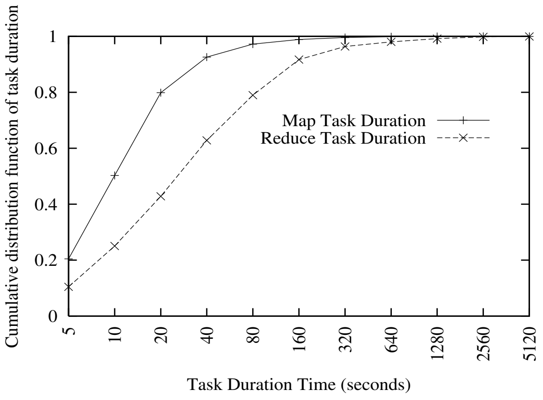


Fig. 10. Cumulative distribution of task durations of all jobs.

TABLE IV
TASK DURATION STATISTICS.

Task type	Quantity	Percentage
map second-tasks (0 - 120 seconds)	12954135	98%
map minute-tasks (2 minutes - 1 hour)	206234	1.5%
map hour-tasks (>1 hour)	1290	0.5%
reduce second-tasks (0 seconds -120 seconds)	1352878	87%
reduce minute-tasks (2 min - 1 hour)	184939	12%
reduce hour-tasks (>1 hour)	596	1%

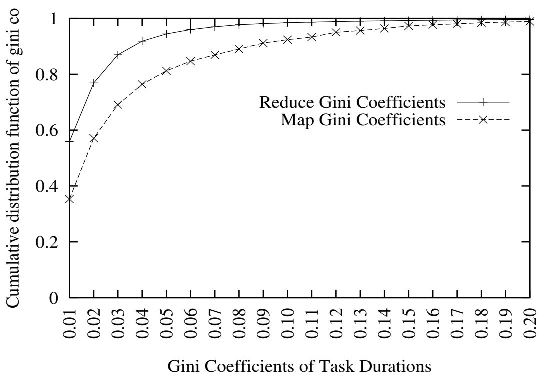


Fig. 11. Gini coefficients of task durations.

tasks with a duration of more than one hour. Table IV presents the amount of tasks in each task group.

3) Task Equity: Task equity measures the variation between task duration times. Ideally, the tasks belonging to the same job are executed in a parallel manner and consume a similar time duration. Task equity also indicates execution efficiency or parallelization degree.

Various methods have been proposed to measure the equity for a distribution. Gini coefficient [12] is the most commonly one. It is typically used to evaluate the income equality in a country. In our work, we use it to measure the equity of task duration. The Gini coefficient varies between 0.0 and 1.0. A value of 0 means all tasks have identical durations, while a higher value indicates that there is a greater disparity among the durations of tasks. A Gini coefficient of 1.0 indicates complete inequality.

Figure 11 shows the cumulative distribution of jobs with the given Gini coefficient for their map and reduce tasks, respectively. We observed that more than 95% of jobs with Gini coefficients of map task durations <0.15, and more than 95% of jobs with Gini coefficients of reduce task durations <0.05. **Observation 7.** *The task inequity problem for map tasks is much worse than the one for reduce tasks, which is beyond our conventional viewpoints. After in-depth analysis, we recognize that it is caused by skewed map operations, like table JOIN. The map tasks that perform the broadcasts do more I/O than the other ones.*

4) Data Locality: In order to reduce data movement, the fair scheduler in Hadoop uses a delay scheduling mechanism to optimize data locality of tasks (placing tasks on nodes that contain their input data). The job scheduler selects the task with data closest to the TaskTracker, trying to place the tasks on the same node if possible (Node-local), otherwise on the same rack (Rack-local), or finally, on a remote rack (Remote-rack). Table V shows the statistics of data locality for all tasks. 92.7% of tasks achieved node-local data locality. **Observation 8.** *The fair scheduler can achieve a high data locality. Therefore, to reduce data movement by further improving data locality is not necessary.*

TABLE V
DATA LOCALITY STATISTICS.

Locality level	Percentage
Node-local	92.7%
Rack-local	4.4%
Remote-rack	2.9%

V. RESOURCE UTILIZATION

In this section, we report a detailed resource utilization statistics of the Yunti cluster in terms of CPU utilization, memory usage, slots usage, network transfer, and data I/O during two weeks.

A. CPU Utilization

Figure 12(a) shows the average CPU utilization ratio for all nodes in the cluster during two weeks. The CPUs in all TaskTrackers are basically uniform. Each TaskTracker has 24 CPU and each CPU has 6 cores. The X-axis represents the day from 12/4/2011 to 12/20/2011, and the Y-axis represents the CPU utilization ratio. The red line on 70% for Y-axis, denotes the warning line of the CPU workload. The yellow line on 55% for the Y-axis, denotes the attention line of the CPU workload. Both the warning and attention lines are used to provide convenience for Hadoop operators to observe the cluster's status.

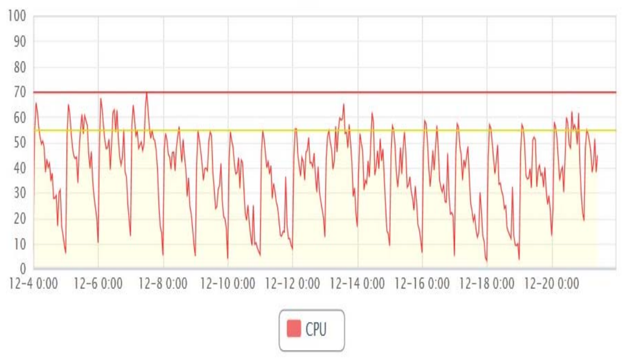
The curve in Figure 12(a) shows that the CPU utilization fluctuates periodically, reaching the peak during the interval between 1 am and 4 am everyday, and falling down in the daytime. The peak CPU utilization exceeds 90%, but it does not last very long. The trend of the CPU utilization ratio is highly correlated to the job arrival rate.

B. Memory Usage

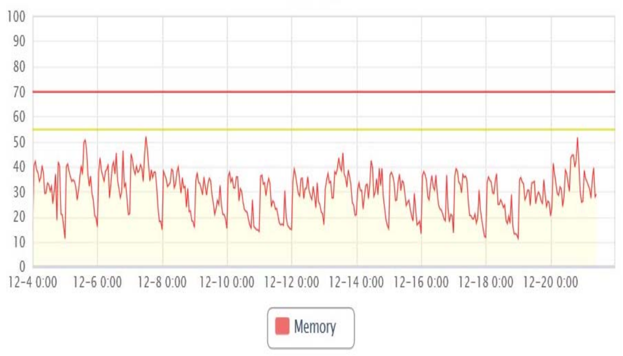
Figure 12(b) shows the memory usage for the cluster during two weeks. Y-axis represents the percentage of memory utilization. The memory size in each TaskTracker is 48G. Similar to Figure 12(a), the red line on 70% for the Y-axis, denotes the warning line of memory usage. The yellow line on 55% for the Y-axis, denotes the attention line of memory usage. The curve in Figure 12(b) also expresses to be periodic. The memory utilization fluctuates with the workload variation, and lies in the range between 20% and 40%. We can see that the memory is sufficient.

C. Slots Allocation

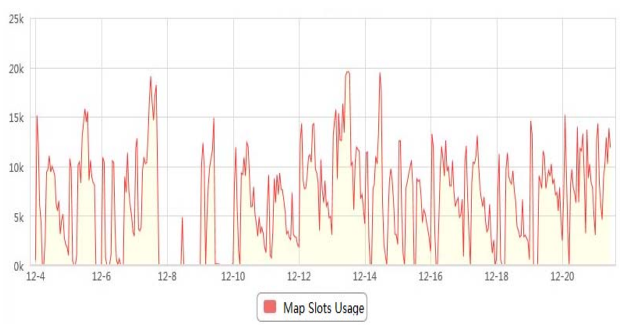
Figures 12(c) and 12(d) show the ratio of map slots usage and reduce slot usage, respectively. The Y-axis represents the number of slots occupied. As presented in these two figures, the slot usage curve also appears to be periodic. Both curves



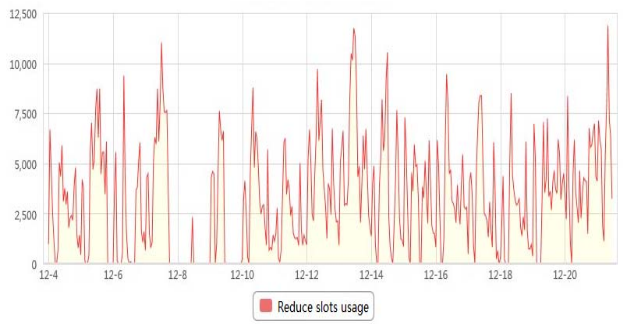
(a) CPU utilization.



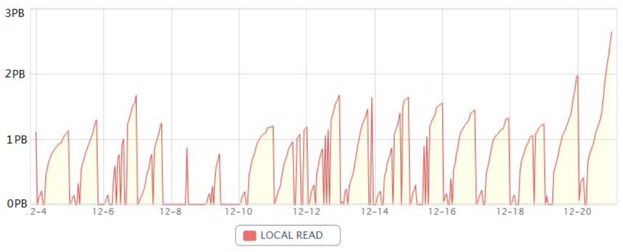
(b) Memory utilization.



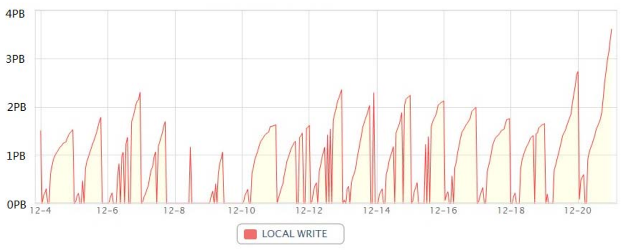
(c) Map slots utilization.



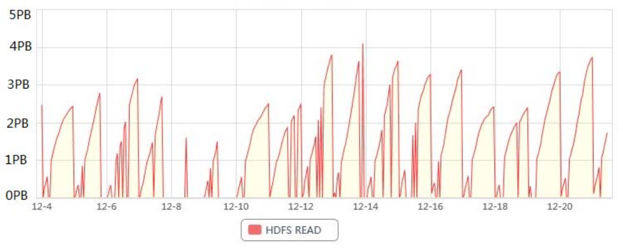
(d) Reduce slots utilization.



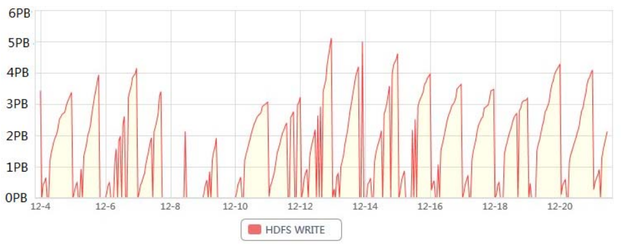
(e) Bytes read from local disks.



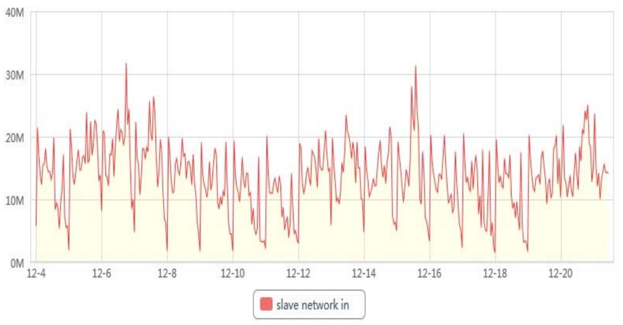
(f) Bytes written on local disks.



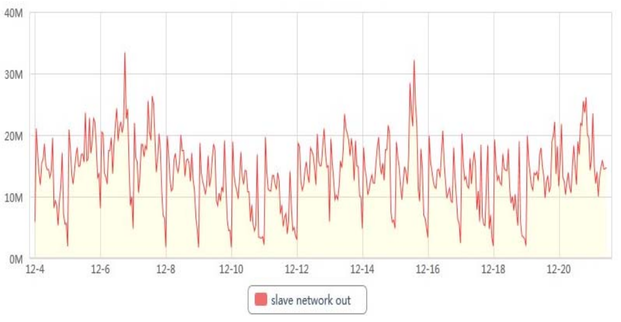
(g) Bytes read from HDFS.



(h) Bytes written on HDFS.



(i) In-flow network traffic.



(j) Out-flow network traffic.

Fig. 12. Resource utilization statistics during the two-week interval.

reach the summit in a short period everyday. The amount of map slots and reduce slots in the Yunti cluster are 19,643 and 15,273, respectively. The map slots usage is about 7,500 (38%), while the reduce slots usage is about 6,500 (43%). We can see that both kinds of these slots usage ratios do not reach a high level, the ratio of reduce slots usage is slightly higher than the one of map slots.

D. I/O Statistics

Figures 12(e) and 12(f) show the I/O statistics on local disks. Local disks read and write are mainly caused by accessing and storing intermediate data generated on MapReduce stages. We can observe that the local disks read and writes bytes reach about 1.5PB and 2PB per day, respectively. Since the Hadoop cluster consists of 2,000 nodes, data read and write speeds on each DataNode are 9MB/s and 12MB/s on average, respectively.

Figures 12(g) and 12(h) show the data I/O statistics on HDFS per day during two weeks. The Y-axis represents the amount of read/write bytes. During a job execution process, HDFS read is mainly generated by reading input data from HDFS, while HDFS write is mainly generated by writing output data to HDFS. It is observed that data reads and writes on HDFS reach about 3PB and 4PB per day, respectively. On average, data read and write speeds on each DataNode are 18MB/s and 25MB/s, respectively. Note that if a task gains node-level data locality, HDFS read is actually carried out on the local nodes. **Observation 9.** *Data read and write workloads on HDFS and local disks are high. It might be beneficial to optimize the implementation of MapReduce on Hadoop. Employing a distributed shared memory (DSM) system [13] would be a good solution to decrease the read/write workloads. The rapid growth of data volume requires scalable and efficient data management and scheduling strategies for Hadoop.*

E. Network Transfer

For each job execution, data is transferred or exchanged between the nodes of the Yunti cluster. The network traffic on a node contains in-flow data and out-flow data. The former means the data received from the other nodes, and the latter means the data sent to the other nodes. The network in-flow and out-flow are mainly generated in two cases: 1) data shuffle in the reduce stage; 2) remote HDFS access when tasks do not gain node-level data locality. Note that the values are collected cumulatively for each day.

Figures 12(i) and 12(j) show in-flow and out-flow network traffic per second during two weeks. The Y-axis represents the network IO speed (MB/s) for each node. The network IO consists of input read, shuffle, and output write. For most of the time, the in-flow and out-flow network traffic ranges from 10MB/s to 20MB/s. **Observation 10.** *We can see that the network load is rather busy. With the data volume grows and cluster scales, the network transfer will become a potential bottleneck of the whole system. Efficient data placement and scheduling policies for reducing network load are needed.*

TABLE VI
PERSON'S CORRELATION COEFFICIENTS BETWEEN RESOURCE UTILIZATION STATISTICS.

Comparison	Coefficient
CPU usage - Memory usage	0.763
Map slots - Reduce slots	0.812
In-flow network traffic - Out-flow network traffic	0.868
Bytes read from HDFS - Bytes read from local disks	0.902
Bytes written on HDFS - Bytes written on local disks	0.898

F. Summary of Resource Utilization

As expected, the resource utilization in terms of CPU, memory and bandwidth expresses periodic daily changes, which fits the job arrival rates change very well. Within an interval of one day, CPU, memory and bandwidth usage are bursty. Within an interval of one week, these statistics perform steady similar patterns. These results suggest that from a utilization perspective, there exist excellent opportunities for dynamic resource quota in the cluster.

The peak arrival pattern among the figures in Figure 12 seems to be very similar. To quantize the similarity, we conducted a Pearson's correlation analysis between these statistics of resource utilization and present the results in Table VI. As expected, these statistics exemplify high correlations with each other.

VI. IMPLICATIONS OF PERFORMANCE OPTIMIZATION

Based on the observations discussed in the above two sections, we derive some indirect implications for performance improvement and open issues. We discuss these implications in the following paragraphs.

A. Pre-scheduling Periodic Jobs

The trace analysis shows that over 80% of jobs are periodic jobs, which are loaded once everyday. Based on this observation, job scheduler could be optimized especially for these periodic jobs. Two aspects suggest that this improvement effort is feasible. From one aspect, periodic jobs require relative computing resources, which can be obtained from historical logs. From the other aspect, job scheduler is pluggable and easy to be replaced. It is practical to design a pre-scheduling model-based scheduler for Hadoop. The improved scheduler could reduce the times of preemption and optimize the sequence of job execution. To the best of our knowledge, there is little research about pre-scheduling periodic jobs.

B. Dynamic Resource Allocation

The trace analysis shows that the job arrival rate appears to follow a fairly constant pattern. From the historical job arrival log, it is feasible to identify the arrival pattern by using a data mining algorithm. Using the pattern, job scheduler can predict the slots usage for each group, thereby allowing for adjustment dynamically the slots usage upper bound for each group, leading to improve the slots utilization and system throughput.

C. Job Failure Handling

Hadoop is designed to be a fault-tolerant system. Jobs on Hadoop should be resilient to nodes failures. However, some job failures still come up from time to time. A job fails if the same task fails a pre-defined number of times (it is set to 4 by default). The trace analysis shows that failed jobs consume non-negligible computing resources. There were 0.6% failed jobs and 0.4% killed jobs in our trace. We observed that most failed jobs are caused by task failures, while most task failures are mainly caused by the exception of out-of-memory.

In order to reduce task failures, besides rewriting the programme code of failed jobs, it is necessary to devise a much more efficient memory allocation and management at the granularity of task. Reducing the possibility of exception of out-of-memory could save considerable computing resource. Until recently, most studies on the production trace of computer systems and clusters have focused on hardware failures [14], [15], while no prior work concerns the resource management in the context of failure handling.

D. Data Skew Issue

Data skew has been a fundamental problem since the beginning of the MapReduce framework, because a task doing too much work slows down the whole system. It is well-understood that the higher the task duration inequity is, the more time a job consumes, as the job finish time is dependent on the slowest task finish time. Based on preliminary analysis, we believe that the main reason for task duration inequity is data skew. Even well-written MapReduce jobs may also encounter a data skew issue. Dynamic data insertions and deletions also cause the data skew problem. The data repartition and redistribution algorithms implemented in HDFS can be used to solve the data skew issue. Despite extensive academia research and industry efforts, no effective skew handling mechanism has been implemented by Hadoop.

E. High Availability Improvement

Hadoop has employed some mechanism to improve its availability. For instance, creating and managing multiple data replicas across multiple nodes and racks. However, the high availability of master nodes, including JobTracker and NameNode, has not been achieved. The master nodes in Hadoop exist as a single point of failure.

Hadoop master nodes (JobTracker and NameNode) are apt to become the system performance bottleneck. Master nodes are centralized, which limits the scalability and availability greatly. Central masters are a bottleneck, and performance could be enhanced if they were distributed. Even if a single master is not a performance bottleneck given current cluster size, it could be the case that having more fine grained heartbeats from the same number of slave nodes will improve performance. A distributed master might contribute to achieve high availability. Numerous researchers propose various solutions for the JobTracker and NameNode high availability problem. However, thus far, few practical solutions have been applied to the large-scale cluster in real environments.

F. Schedulability Analysis

Recent trends of large-scale data analysis have moved beyond the batch-processing mode. MapReduce is challenged by various real-time data processing workloads. For instance, the workloads including online log processing, social network analysis, real-time web indexing, etc., require fast response times under high data rates.

Schedulability analysis is an efficient method to predict and verify whether a system can satisfy the time-constraints for real-time requests. Some previous works have been conducted on scheduling analysis, but they focused on multiprocessors [16] and virtualized platforms [17], based on relatively simple task models and mainly on computing resources. Compared with the previous work, schedulability analysis of MapReduce is much more challenging due to the complexity of MapReduce.

VII. Related Work

Workload characterization studies are useful for helping Hadoop operators identify system bottleneck and figure out solutions for optimizing performance. Many previous efforts have been accomplished in different areas, including network system [18], [19], storage system [20], [21], [22], Web server [23], [24], and HPC cluster [25]. Both network and storage subsystems are key components for the Hadoop system. Our trace analysis also covers workload statistics in terms of network and storage. This paper builds on these studies.

Several studies [26], [27], [28] have been conducted for workload analysis in grid environments and parallel computer systems. They proposed various methods for analyzing and modeling workload traces. However, the job characteristics and scheduling policies in grid are much different from the ones in a Hadoop system [29].

Mishra et al. [30] focused on workload characterization that includes behavior characteristics such as CPU and memory. The Yahoo Cloud Serving Benchmark [31] focused on characterizing the activity of database-like systems at the read/write level. Soila Kavulya et al. [9] conducted an analysis about the job characterization, such as job patterns and failure sources, based on the Hadoop logs from the M45 super-computing cluster. Their research work focused on predicting job completion time and found potential performance problems based on the historical data.

However, these studies do not analyze many of the workload characteristics discussed in this paper. For example, we report not only the job statistics but also task statistics. We also derive some direct implications on the observations of the MapReduce workload trace, and present resource utilization on a Hadoop cluster, which is very helpful,facilitating Hadoop operators to improve system performance.

Chen et al. [32] analyzed and compared two production MapReduce traces derived from Internet service companies, in order to develop a vocabulary for describing MapReduce workloads. The authors show that existing benchmarks fail to reproduce synthetic workloads that express such characteristics observed in real traces. These works are instructive to our work. Our study uses a similar statistical profile of the trace showing both behavior at the granularity of job and task. Our analysis of MapReduce workloads corroborates observations from previous studies of workload on the Hadoop system [33], [9].

Although our work is close to the study presented in [33], [9], this paper has two novel aspects: (1) the jobs analyzed in this paper are representative and common in data platform for an e-commerce website. We believe the trace is a beneficial complement of a current public workload repository. (2) In addition to workload analysis, we concentrate on the relation of workload analysis and the performance optimization method, and conclude some direct implications based on the analysis results. It is useful for guiding Hadoop operators to optimize performance.

VIII. Conclusion and future work

In this paper, we have presented the analysis of Hadoop trace derived from a 2,000-node production Hadoop cluster in Taobao, Inc. The trace covers the jobs execution logs over a two-week period, involving 912,157 jobs, which are representative and common in data platform for an e-commerce website. We conduct a comprehensive analysis of the workload trace at the granularity of job and task, respectively. Some main observations and their direct implications are concluded. These findings can help other researchers and engineers understand the performance and job characteristics of Hadoop in their production environments.

In the future, we plan to work on the implications derived from this work and integrate them into the Yunti cluster. In addition, we are currently working on a workload generator WaxElephant and Hadoop simulator Ankus based on the workload characteristics observed in this work. The simulator will be available at http://mist.cs.wayne.edu soon.

Acknowledgment

We would like to thank the Hadoop team at Taobao for practical experiences that guided this work. We are grateful to Jianying, Yunzheng, Wuwei, Tuhai, Zeyuan for their insightful suggestions. This research was supported by SSF of Hangzhou Dianzi University (No. KYS055611018), NSF of Zhejiang (No. LQ12F0200), NSF of China (No. 61202094), and the science and technology major project of Zhejiang (No. 2011C11038). Weisong Shi is in part supported by the Introduction of Innovative R&D team program of Guangdong Province (NO. 201001D0104726115), Hangzhou Dianzi University, and the NSF Career Award CCF-0643521.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.

[2] T. White, *Hadoop - The Definitive Guide*. O'Reilly, 2009.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SOSP*, vol. 37, no. 5, 2003, pp. 29–43.

[4] Ganglia. [Online]. Available: ganglia.sourceforge.net

[5] Y. Chen, S. Alspaugh, D. Borthakur, and R. H. Katz, "Energy efficiency for large-scale mapreduce workloads with significant interactive analysis," in *EuroSys*. ACM, 2012, pp. 43–56.

[6] J. D. C. Little, "A prof for the queuing formula L=hW," *Operations Research*, vol. 9, 1961.

[7] X. Liu, J. Han, Y. Zhong, C. Han, and X. He, "Implementing webGIS on hadoop: A case study of improving small file I/O performance on HDFS," in *CLUSTER*, 2009, pp. 1–8.

[8] G. Mackey, S. Sehrish, and J. Wang, "Improving metadata management for small files in HDFS," in *CLUSTER*, 2009, pp. 1–4.

[9] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *CCGRID*, 2010, pp. 94–103.

[10] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys*, 2010, pp. 265–278.

[11] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," in *CloudCom*, 2011, pp. 40–47.

[12] E. Andersen, *The Statistical Analysis of Categorical Data*. Springer, Berlin, 1992.

[13] S. Krishnamoorthy and A. Choudhary, "A scalable distributed shared memory architecture," *JPDC*, vol. 22, no. 3, pp. 547–554, 1994.

[14] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *DSN*, 2006, pp. 249–258.

[15] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *DSN*, 2004, p. 772.

[16] D. Liu and Y.-H. Lee, "Pfair scheduling of periodic tasks with allocation constraints on multiple processors," in *IPDPS*, 2004.

[17] J. Lee, A. Easwaran, and I. Shin, "LLF schedulability analysis on multiprocessor platforms," in *IEEE Real-Time Systems Symposium*, 2010, pp. 25–36.

[18] D. Ersoz, M. S. Yousif, and C. R. Das, "Characterizing network traffic in a cluster-based, multi-tier data center," in *ICDCS*, 2007, p. 59.

[19] V. Paxson, "Empirically derived analytic models of wide-area TCP connections," *IEEE/ACM Trans. Netw*, vol. 2, no. 4, pp. 316–336, 1994.

[20] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, and B. Schroeder, "An analysis of data corruption in the storage stack," *ACM Transactions on Storage*, vol. 4, no. 3, pp. 821–834, 2008.

[21] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: practical power-proportionality for data center storage," in *EuroSys*, 2011, pp. 169–182.

[22] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty, "File system workload analysis for large scale scientific computing applications," in *MSST*, 2004, pp. 139–152.

[23] E. Hernández-Orallo and J. Vila-Carbó, "Web server performance analysis using histogram workload models," *Computer Networks*, vol. 53, no. 15, pp. 2727–2739, 2009.

[24] W. Shi, Y. Wright, E. Collins, and V. Karamcheti, "Workload characterization of a personalized web site and its implications for dynamic content caching," in *WCW*, 2002, pp. 1–16.

[25] A. Iamnitchi, S. Doraimani, and G. Garzoglio, "Workload characterization in a high-energy data grid and impact on resource management," *Cluster Computing*, vol. 12, no. 2, pp. 153–173, 2009.

[26] K. Christodoulopoulos, V. Gkamas, and E. A. Varvarigos, "Statistical analysis and modeling of jobs in a grid environment," *J. Grid Comput*, vol. 6, no. 1, 2008.

[27] E. Medernach, "Workload analysis of a cluster in a grid environment," in *Job Scheduling Strategies for Parallel Processing*, 2005, pp. 36–61.

[28] B. Song, C. Ernemann, and R. Yahyapour, "User group-based workload analysis and modelling," in *CCGRID*, 2005, pp. 953–961.

[29] M. Zaharia, D. Borthakur, J. S. Sarma, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," Univ. of Calif., Berkeley, CA, Technical Report No. UCB/EECS-2009-55, Apr. 2009.

[30] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, J. M. Hellerstein, S. Chaudhuri, and M. Rosenblum, Eds., 2010, pp. 143–154.

[32] Y. Chen, A. Ganapathi, R. Griffith, and R. H. Katz, "The case for evaluating mapreduce performance using workload suites," in *MASCOTS*, 2011, pp. 390–399.

[33] Y. Chen, S. Alspaugh, and R. H. Katz, "Design insights for mapreduce from diverse production workloads," University of California, Berkeley, Tech. Rep. UCB/EECS-2012-17, 2012.