Workload Analysis, Implications and Optimization on a Production Hadoop Cluster: A Case Study on Taobao

Zujie Ren, Jian Wan, *Member, IEEE,* Weisong Shi, *Senior Member, IEEE,* Xianghua Xu, *Member, IEEE,* and Min Zhou

Abstract—Understanding the characteristics of MapReduce workloads in a Hadoop cluster is the key to making optimal configuration decisions and improving the system efficiency and throughput. However, workload analysis on a Hadoop cluster, especially in a large-scale e-commerce production environment, has not been well studied yet. In this paper, we performed a comprehensive workload analysis using the trace collected from a 2,000-node Hadoop cluster at Taobao, which is the biggest online e-commerce enterprise in Asia, ranked 10^{th} in the world as reported by Alexa. The results of the workload analysis are representative and generally consistent with the data warehouses for e-commerce web sites, which can help researchers and engineers understand the workload characteristics of Hadoop in their production environments. Based on the observations and implications derived from the trace, we designed a workload generator *Ankus*, to expedite the performance evaluation and debugging of new mechanisms. Ankus supports synthesizing an e-commerce style MapReduce workload at a low cost. Furthermore, we proposed and implemented a job scheduling algorithm *Fair4S*, which is designed to be biased towards small jobs. Small jobs account for the majority of the workload and most of them require instant and interactive responses, which is an important phenomenon at production Hadoop systems. The inefficiency of Hadoop fair scheduler for handling small jobs motivates us to design the Fair4S, which introduces pool weights and extends job priorities to guarantee the rapid responses for small jobs. Experimental evaluation verified the Fair4S accelerates the average waiting times of small jobs by a factor of 7 compared with the fair scheduler.

Index Terms—Hadoop, MapReduce, workload analysis, workload synthesis, job scheduler

1 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 the 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 applications.

Our work was originally motivated by the need for improving the system performance and resource utilization of a Hadoop cluster in Taobao. To meet this end, we solve the problem in two ways. In one aspect, the Hadoop system involves hundreds of configuration parameters. It will be beneficial, but challenging, for Hadoop operators to make optimal configuration decisions. In another aspect, as tens of thousands of MapReduce jobs are scheduled and executed everyday, the job scheduling algorithm plays a critical role in the whole system performance. Optimizing the job scheduling efficiency is also essential work.

Our work starts with the analysis of MapReduce workload in the Hadoop cluster. We believe that understanding the characteristics of MapReduce workloads is the key to obtain better configuration decisions. Hadoop operators can apply the results of the workload analysis to optimize the scheduling policies and to allocate resources more effectively under diverse workloads. However, as of yet, workload analysis of MapReduce, especially in a large-scale e-commerce production environment, has not been well studied.

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. Our analysis reveals a range of workload characteristics and provides some direct implications. We believe that workload analysis study is useful for helping Hadoop operators identify system bottlenecks

- *Zujie Ren, Jian Wan and Xianghua Xu are with the School of Computer Science and Technology, Hangzhou Dianzi University, China E-mail: {renzj,wanjian,xhxu}@hdu.edu.cn*
- *Weisong Shi is with the Department of Computer Science, Wayne State University, Detroit, USA. E-mail: weisong@wayne.edu*
- *Min Zhou is with Taobao, Inc. Hangzhou, Zhejiang Province, China Email:zhouchen.zm@taobao.com*

and figure out solutions for optimizing performance. A deep understanding of the characteristics of jobs running in Hadoop contributes to determining the factors affecting jobs execution efficiency.

Due to the system complexity and scale, an intuition-based analysis is often inadequate to explicitly capture the characteristics of a workload trace. The problem is even worse if the workload is diverse. The first challenge of MapReduce workload characterization arises in identifying key factors and parameters that might help systems designers and practitioners of large-scale MapReduce systems. The second challenge is to figure out appropriate statistical models for profiling various characteristics, which give insights of open research problems for future system design, implementation and optimization. Finally, an even greater challenge is to deeply understand the workload characteristics, so as to design practical tools and derive new knowledge about the MapReduce systems.

To address these challenges, we conducted a comprehensive workload analysis using statistical models. Based on the results of the workload analysis, we designed an effective workload generator *Ankus*, to synthesize an e-commerce style workload for expediting the performance evaluation. Furthermore, we proposed and implemented a job scheduling algorithm *Fair4S*, which is designed to be biased towards small jobs. Fair4S introduces pool weights and extends job priorities to guarantee the rapid response for small jobs. More specifically, the contributions of this paper are listed as follows.

- We undertook a comprehensive analysis based on a production workload trace, which is collected from a 2,000-node production Hadoop cluster during two weeks. The trace includes 912,157 jobs, which are representative and common in data analysis platforms of e-commerce web sites. The main observations and their direct implications derived from the trace analysis are listed in Table 1. These findings can help researchers and engineers better understand the performance and job characteristics of Hadoop in their production environments.
- We designed and implemented a workload generator called *Ankus* based on the models derived from the workload analysis. Ankus supports two mechanisms for generating workloads, one is to replay a historical workload traces, or scale the workload intensity which follows similar statistical distribution observed in the real-world workload trace; the other is to synthesize configurable workloads to meet various user demands. Ankus facilitates the evaluation of job schedulers and debugging on a Hadoop cluster.
- We proposed and implemented a job scheduler called Fair4S, to optimize the completion time of small jobs. According to the results of trace analysis, we found that small jobs account for the majority and most of them require instant and interactive responses. Unfortunately, fair scheduler [3], the default and most commonly used scheduler of Hadoop, fails to support interactive responses for small jobs. The same observation is true on Facebook [4]. To address this problem, Fair4S introduces pool weights and extends job priorities to guarantee the rapid response for small jobs. It is verified that Fair4S accelerates the average waiting times by a factor of 7 compared with fair scheduler for small jobs.

The rest of this paper is organized as follows. Section 2 provides a brief introduction of MapReduce and Hadoop, and then gives an overview of the Yunti cluster. Section 2.2 discusses the summary of the trace, including the information extracted and the format of the logs. A detailed analysis of these logs at the granularity of jobs and tasks is described in Section 3. Section 4 shows the resource utilization of the Yunti cluster in terms of disk IO and network transfer. Section 5 describes the workload generator Ankus and Section 6 presents the job scheduling aiming at optimizing the execution efficiency of small jobs. The evaluation results on Ankus and Fair4S are given in Section 7. Section 8 discusses the related work and this paper is concluded in Section 9.

2 Background

To facilitate the understanding of MapReduce workload analysis in this paper, we describe the architecture of the Yunti cluster [5] and its implementation details. Then we present the summary of trace, including the information extracted and the format of the logs.

2.1 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 web site of "www.taobao.com". Up to December 2011, the Yunti cluster consists of over 2,000 heterogeneous nodes[1]. The total volume of data stored in the Yunti has exceeded 25PB, and the data volume grows 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 0.19, with some slight modifications. Figure 1 presents the overview of the Yunti cluster.

The data resident in the Yunti cluster comprises the replicas of online databases, logs of Web servers (e.g., Apache and Ngnix), crawled pages, and so on. The Yunti cluster maintains 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.

The jobs executed on the Yunti cluster are diverse. Over fifty-six groups, including over five hundred users, submit 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 originate from numerous applications, such as commodities recommendations, traffic statistics and advertise delivery system.

1. Note that the scale of the Yunti grows every week. All statistics about the Yunti cluster given in this paper was collected on Dec.12, 2011.

TABLE 1: Summary of observations and implications.

Observations	Implications	Sections
O1: Job arrival rate follows a fairly constant pattern. The arrival times of jobs follow a Poisson distribution.	Node scheduling mechanism based on the job arrival pattern will be feasible and useful for saving power.	3.1.1
O2: Slot preemptions caused by high-priority jobs are frequent.	Optimizing a preemption scheme should be a priority.	3.1.2
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.	3.1.3
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.	3.1.4
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.	3.2.1
O6: Both map task and reduce task duration time follow a log-normal distribution.	Conventional simulation with uniform-duration tasks is unrealistic.	3.2.2
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.	3.2.3
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.	3.2.4
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 system would be a good solution to decrease the read and write workloads on HDFS.	4.1
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.	4.2

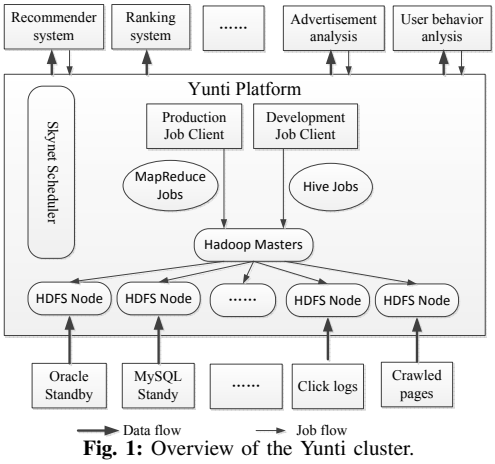


Fig. 1: Overview of the Yunti cluster.

TABLE 2: Summary of the workload trace.

Log Period	4/10-20/10, 2011
Number of groups	56
Number of users	572
Number of jobs	91,2157
Successful jobs	90,2837 (90%)
Failed jobs	5,672 (0.6%)
Killed jobs	3,648 (0.4%)
Maximum maps per job	91,081
Average maps per job	42
Minimum maps per job	1
Standard deviation of maps	2677.3
Maximum reduces per job	28,107
Average reduces per job	12
Minimum reduces per job	1
Standard deviation of maps	291.3
Maximum job duration	59040s
Average job duration	35s
Minimum job duration	1s
Standard deviation of job durations	237.7

2.2 Overview of the 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 submitted by 572 different users belonging to fifty-six groups. Most of the jobs are daily periodic, which are submitted automatically and loaded at a constant time every day. Table 2 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 was 489. According to the job's final status, all jobs are separated into three groups:

- **Successful jobs**: those jobs executed successfully.
- **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 902,837 (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,

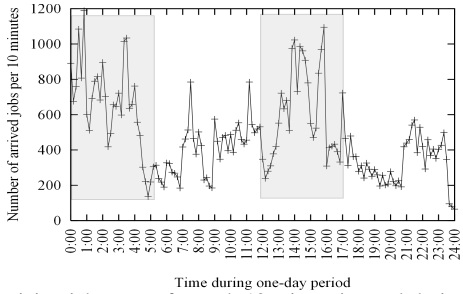


Fig. 2: Arriving jobs count for each 10-minute interval during one day.

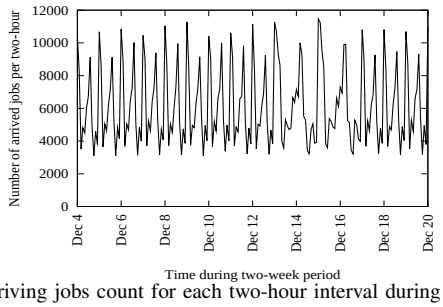


Fig. 3: Arriving jobs count for each two-hour interval during two weeks.

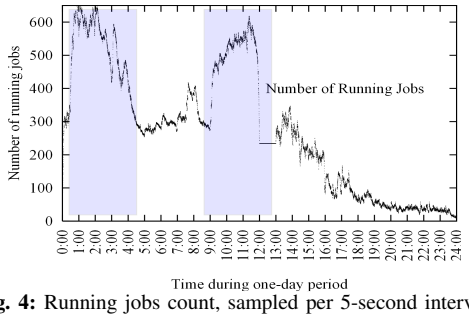


Fig. 4: Running jobs count, sampled per 5-second interval.

task start time, and task finish time.

In addition to the workload trace, resource utilization statistics of the Yunti cluster in terms of network transfer, and data I/O were collected by Ganglia [6]. Details on resource utilization are presented in Section 4.

3 Workload Traces Analysis

In this section, we first describe job statistics, including job arrival pattern, job completion times and data size. 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.

3.1 Job Statistics

3.1.1 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 p.m. to 4 p.m. The maximum number of arrived jobs within a 10-minute interval exceeds 1,100. The first workload peak is mainly formed by periodic jobs, which are pre-defined by application developers and loaded at a constant time everyday. 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.

After curve fitting, the sequence of arrival times of these jobs is composed by two Poisson random processes. Assuming a time unit is one minute, the values of λ of Poisson distribution for periodic and temporary jobs are 76 and 42, respectively. For most temporary jobs, these workloads are also referred to as MapReduce with Interactive Analysis (MIA) workloads [7]. 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. The arrival times of jobs follow a Poisson distribution. 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.*

3.1.2 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 do not appear at a same time. This observation can be explained by the difference of job duration time. According to Little's law [8] 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 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 in an awaiting-state. **Observation 2.** *As depicted in the Figure 4, we can infer*

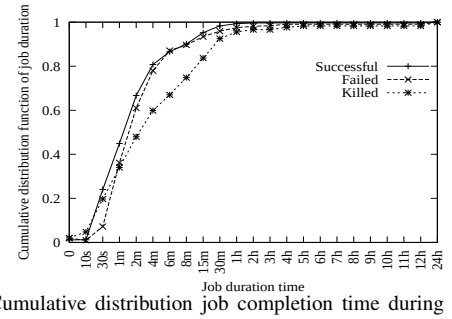


Fig. 5: Cumulative distribution job completion time during a two-week period.

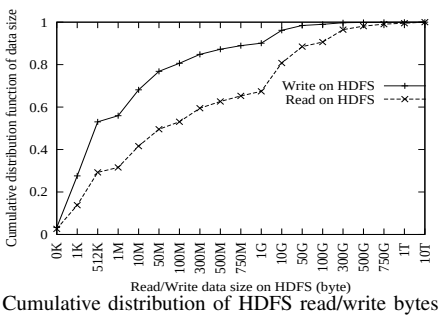


Fig. 6: Cumulative distribution of HDFS read/write bytes per job.

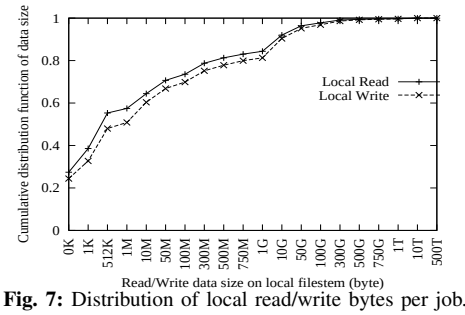


Fig. 7: Distribution of local read/write bytes per job.

that preemption is frequent, and optimizing a preemption scheme should be a priority.

3.1.3 Job Completion Times

Figure 5 presents the cumulative distribution function (CDF) 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 Hadoop 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. The 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, and so on. 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. We employed the Levenberg-Marquardt algorithm [9] to fit the job completion times to the Weibull, exponential and log-normal distributions. The goodness of fit between the original distribution and the reference distributions is examined by the Kolmogorov-Smirnov test[10], which calculates the maximal distance between the (empirical) CDF of the original distribution and the reference distributions. Kolmogorov-Smirnov test is a commonly used technique to compare two distributions, which is also adopted in several literatures[11], [12]. **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.* A log-normal distribution $LN(4.32, 1.31)$ fits the CDF of successful jobs duration with a Kolmogorov-Smirnov (*KS* for short) value of 0.05, where $LN(x, \mu, \delta) = \frac{1}{x\delta\sqrt{2\pi}} e^{-\frac{(\ln x-\mu)^2}{2\delta^2}}$ is the log-normal distribution with mean μ and variance δ.

3.1.4 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 is stored in local disks directly.

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 1GB 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 [13], [14]. Therefore, efficiency for small files storage on HDFS is a significant and urgent issue to be solved.*

3.2 Task Statistics

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

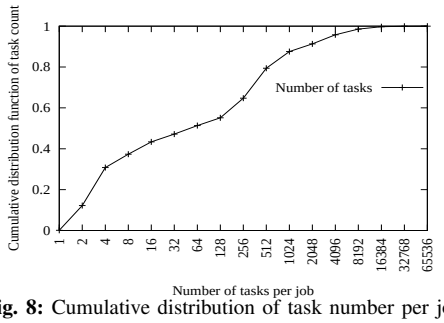


Fig. 8: Cumulative distribution of task number per job.

TABLE 3: 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%

3.2.1 Task Number

Figure 8 depicts the cumulative distribution of task number 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 3.

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 [15], [16]. **Observation 5.** *The task counts of jobs in the 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.* In the latter part of this paper (Section 6), we will describe an improved fair scheduler for optimizing the small jobs execution.

3.2.2 Task Duration

Figure 9 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.* For map task duration, $LN(2.31, 0.87)$ fits with a *KS* value of 0.02; for reduce task duration, $LN(3.24, 1.37)$ fits with a *KS* value of 0.01.

To further summarize the task duration distribution, we

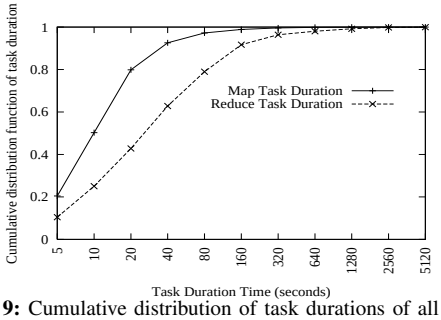


Fig. 9: Cumulative distribution of task durations of all jobs.

TABLE 4: Task duration statistics.

Task type	Quantity	Percentage
map second-tasks (<120s)	12954135	98%
map minute-tasks (2m-1h)	206234	1.5%
map hour-tasks (>1h)	1290	0.5%
reduce second-tasks (<120s)	1352878	87%
reduce minute-tasks (2m-1h)	184939	12%
reduce hour-tasks (>1h)	596	1%

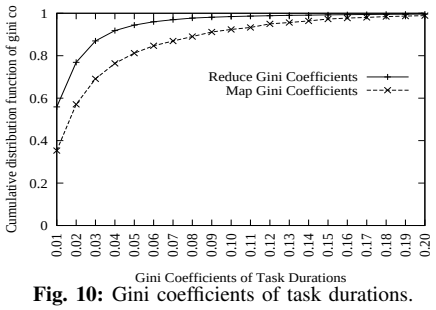


Fig. 10: Gini coefficients of task durations.

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 tasks with a duration of more than one hour. Table 4 presents the amount of tasks in each task group.

3.2.3 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 [17] 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 and 1. 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 indicates complete inequality.

Figure 10 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.*

TABLE 5: Data locality Statistics.

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

3.2.4 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 5 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.*

4 Resource Utilization

In this section, we report a detailed resource utilization statistics of the Yunti cluster in terms of network transfer, and data I/O during the same period. All statistics are collected once over 30 seconds. For the statistics in terms of CPU utilization, memory usage, please refer to our previous work[5].

4.1 I/O Statistics

Figures 11(a) and 11(b) show the I/O statistics on local disks. Local disks read and write are mainly caused by accessing intermediate data generated on the map stage. 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 11(c) and 11(d) 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 caused 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 system [18] 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.*

4.2 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 11(e) and 11(f) 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.** *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.*

5 Ankus: E-Commerce Workload Synthesization

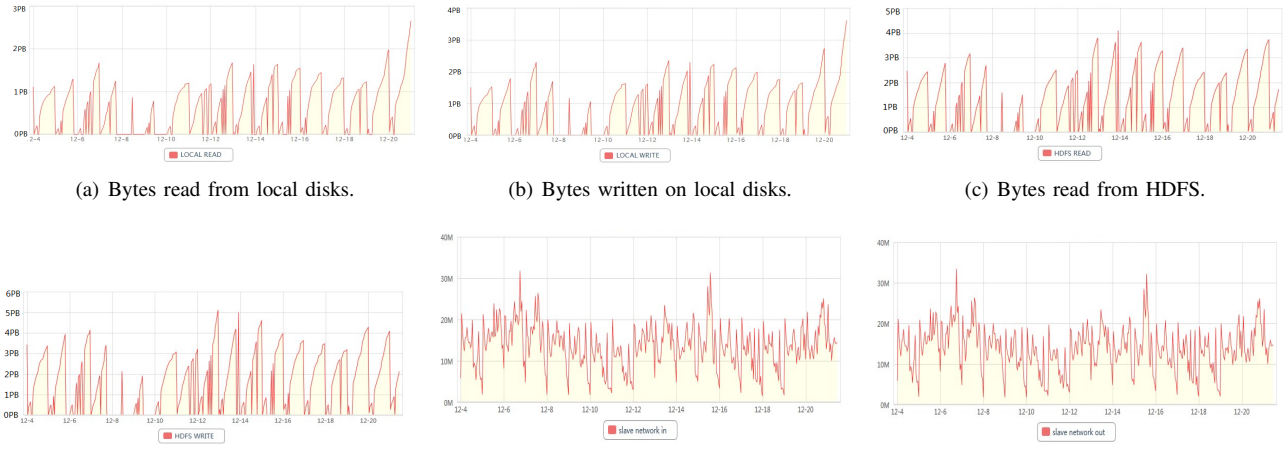
In a production environment, a workload generator is needed to synthesize a representative workload for a particular use. The generated workload is used to evaluate the performance of Hadoop for a specific configuration. However, the increased complexity and diversity of the workload impose difficulties for the development of workload generator.

The results of the workload trace analysis in Section 3 provide significant information for synthesizing representative workloads in e-commerce data warehouses. Based on the models derived from the trace analysis, we design a workload generator Ankus to support flexible and configurable e-commerce workload synthesization.

5.1 Overview of Ankus

The functionality of Ankus relies on a workload trace. For each job in the workload trace, it covers a lot of information about the job, such as job submission time, map/reduce tasks count, bytes and records read/written by each task and allocated memory for each task. Given a workload trace, Ankus distills the statistical distributions and models of the trace, and employs these information to generate a job sequence that simulates a real-world workload. Ankus can be easily generalized to be used in other application environments by only replacing the workload trace for bootstrapping Ankus.

Ankus provides two mechanisms for generating workloads. The first one is to merely replay a sample of the workload trace, or scale the workload intensity which follows similar statistical distribution derived from in the workload trace. This mechanism generates a comparable workload that mimics real workloads. The second one is to synthesize a groups of jobs, which follows specific



(a) Bytes read from local disks. (b) Bytes written on local disks. (c) Bytes read from HDFS.

(d) Bytes written on HDFS. (e) In-flow network traffic. (f) Out-flow network traffic.

Fig. 11: Resource utilization statistics during the two-week interval.

statistics configured by users. The second manner is often used by Hadoop operators to synthesize diverse job types and mixtures for performance evaluation. No matter which mechanism is used by Ankus, the jobs generated by Ankus are injected to JobTracker with a Poisson random process.

As the first mechanism is easy to understand, we mainly focus on the implementation of the second mechanism. In order to synthesize representative jobs, identifying the job groups and find a small set of representative jobs in the workload is requisite. Therefore, we model each individual MapReduce job and then cluster the jobs into several groups. The representative jobs are formed by the cores of all job clusters.

5.2 Job Modeling

Based on the results described in Section 3, we find that an individual MapReduce job can be modeled by many features (dimensions). Among these features, we selected six discriminative features for modeling each MapReduce jobs, including the running time, the input data size, the output data size, local I/O size, running time, map task duration and reduce task duration. Map (reduce) duration is defined as the sum of every map (reduce) task execution time. For example, if a job contains two map tasks and each task execution consumes 10 seconds, the map duration of this job is 20 seconds. Formally, a MapReduce job is modeled by a tuple as $Job_i = \{I_i, O_i, LIO_i, JD_i, MD_i, RD_i\}$, where the meanings of these symbols are listed in the Table 6. All values in each dimension are normalized into a uniform range [0,1].

5.3 Job Clustering

As each job is described by a tuple of features, K-means algorithm [19] is a good solution to determine the natural clusters of jobs. The jobs belonging to a same cluster are aggregated and the core of the cluster is regarded as a single 'representative job'.

However, the job cluster number K may not be easy to be determined. The optimal value is dependent on the diversity of jobs. If the workload is diverse, a higher value for k will be much more accurate. Otherwise, a small value for K is enough. Motivated by the work in [20], we employ a heuristical method for finding the optimal value by incrementing k until the improvement of cluster quality is below than a predefined threshold.

TABLE 6: Symbols and description.

Symbols	Description
Job_i	The i-th MapReduce job
I_i	The input data size of the i-th job
O_i	The output data size of the i-th job
LIO_i	The local read and write of i-th job
JD_i	The running time of the i-th job
MD_i	The sum of each map task duration time
RD_i	The sum of each reduce task duration time

5.4 Job Mixture

In Ankus, the collection of the workload is described by two features.

- Job arrival pattern, which is characterized by number of submitted job per minute. We use a list of $<T_k$,$J_i>$to describe a job submission pattern. Each entry of $<T_k$, $J_i>$represents the i-th job arrives at the time of T_i.
- Share of representative jobs. For synthesized workload, the share of representative jobs can be consistent with historical workload, or be set by users.

In summary, workload generation is performed by the following steps. First, the total number of jobs in the trace is calculated and the job arrival entries list is derived. Then, a heuristic K-means clustering is employed to group all jobs and capture the representative jobs. Finally, the jobs are mixed based on share given by users. According to the job arrival patterns derived from the workload analysis (Section 3.1.1), the arrival of all jobs is simulated by a Poisson random process. The parameter λ is configurable for users. Adjusting the parameters of jobs share and poisson distribution allows us to tune the workload characteristics.

6 Fair4S: Job Scheduler for Small Jobs

As presented in the observation *O5* (Section 3.2.1), small jobs constitute the majority of the workload and most of small jobs require instant responses. The same observation is also valid in the Hadoop cluster of Facebook [4], which is a good example of large-scale social media networks. We believe that this observation is a common case. Therefore, if the scheduler is designed biased towards small jobs, the execution efficiency of small jobs will be improved effectively.

6.1 Revisit of Fair Scheduler

Fair scheduler (FAIR in short) was originally designed by Facebook and subsequently released to the Hadoop community. FAIR is the default scheduler used in Hadoop. The rationale of FAIR is to give every job a fair share of slots over time. The jobs are assigned to pools, where each pool is configured with a minimum share of slots. Idle slots are shared among jobs and assigned to the job with the highest slot deficit.

In the early stage, the Yunti cluster directly employed FAIR [3] to allocate the slots because FAIR achieves high performance and supports multi-user clusters. However, after several months of system running, it is observed that FAIR is not optimal for scheduling small jobs within a miscellaneous workload. The goal of FAIR is to assure the fairness among all jobs. FAIR always reassigns idle slots to the pool with the highest slot deficits. However, small jobs usually require fewer slots, leading to the slot deficits of small jobs are lower than the other jobs. Therefore, small jobs are more likely to suffer from long waits than the other jobs. The users of Yunti submitting small jobs, including application developers, data analysts and project managers from different departments in Taobao, will complain about the long-waits.

With new workloads which feature short and interactive jobs are emerging, small jobs are becoming pervasive. Many small jobs are interactive and online analysis, which requires instant and interactive response. It is quite necessary to design a job scheduler biased towards on small jobs. To meet this goal, the source of FAIR's weakness needs further analysis.

The weakness of FAIR is caused by the policies of slot allocation and re-assignment. In one aspect, setting a minimum share of slots does contribute to maintain fairness, but the response time of jobs are uncontrollable. We believe that specifying a maximum share, rather that a minimum share for each pool contributes to guarantee the efficiency of jobs that require short response times. In the other aspect, the job priority in FAIR only exists four types, not differential enough for massive and diverse jobs. In FAIR, the priority of a job is determined by job-types (*production* or *experimental*) and job-priority (*VERY_LOW*, *LOW*, *NORMAL*, *HIGH*, *VERY_HIGH*). This priority configuration makes it hard to satisfy the requirements for scheduling massive jobs in production environments.

6.2 Features of Fair4S

We introduce four techniques to address the weakness sources of FAIR and implement these techniques in Fair4S. These techniques are described as follows.

1) **Setting Slots Quota for Pools**. All jobs are divided into several pools. Each job belongs to one of these pools. In FAIR, each group is configured with a minimum share to guarantee that it will receive its minimum share if the sum of the minimum shares of all pools is lower than the slots capacity [3]. While in Fair4S, each pool is configured with a maximum slot occupancy. All jobs belonging to an identical pool share the slots quota, and the number of slots used by these jobs at a time is limited to the maximum slots occupancy of their pool. The slot occupancy upper limit of user groups makes the slots assignment more flexible and adjustable, and ensures the slots occupancy isolation across different user groups. Even if some slots are occupied by some large jobs, the influence is only limited to the local pool inside.
2) **Setting Slot Quota for Individual Users**. In Fair4S, each user is configured with a maximum slots occupance. Given a user, no matter how many jobs he/she submits, the total number of occupied slots will not exceed the quota. This constraint on individual user avoids that a user submit too many jobs and these jobs occupy too many slots.
3) **Assigning Slots based on Pool Weight**. In FAIR, priority is given to the pool with the highest deficit when idle slots are allocated. While for Fair4S, each pool is configured with a weight. All pools which wait for more slots form a queue of pools. Given a pool, the occurrence times in the queue is linear to the weight of the pool. Therefore, a pool with a high weight will be allocated with more slots. As the pool weight is configurable, the pool weight-based slot assignment policy decreases small jobs' waiting time (for slots) effectively.
4) **Extending Job Priorities**. Compared with FAIR, Fair4S introduces an extensive and quantified priority for each job. The job priority is described by an integral number ranged from 0 to 1000. Generally, within a pool, a job with a higher priority can preempt the slots used by another job with a lower priority. An quantified job priority contributes to differentiate the priorities of small jobs in different user-groups.

6.3 Procedure of Slots Allocation

The slots allocation is performed by two steps, which can be described as follows:

1) The first step is to allocate slots to job pools. Each job pool is configured with two parameters of maximum slots quota and pool weight. In any case, the count of slots allocated to a job pool would not exceed its maximum slots quota. If slots requirement for one

Algorithm 1 Procedure of Fair4S algorithm.

Input:
A list of job pools, each of which contains a set of jobs
Output:
Results of task scheduling and slots allocation
Parameters:
P_i: the i-th job pool
$PWeight_i$: the weight of the job pool P_i
$PQuota_i$: the quota of the job pool P_i
J_k^i: the k-th job in the job pool P_i
$JPriority_k$: the priority of job J_k^i
$PQueue$: a queue of job pools
$OSlotNum_i$: the number of slots occupied by P_i

//Configuration Initialization
for each P_i **do**
  Initializing $PWeight_i$ and $PQuota_i$
  for each J_k^i in P_i **do**
    Initializing $JPriority_k$
//Slots Allocation
repeat
  Forming $PQueue$ based on $PWeight$ values. The number of occurrence for P_i is linear to its weight $PWeight_i$
  //The 1st step: allocating idle slots to $PQueue$
  repeat *//Round-Robbin Scheduling*
    for each P_i in $PQueue$ **do**
      if $OSlotNum_i < PQuota_i$ **then**
        Allocating one slot to P_i
        $OSlotNum_i \leftarrow OSlotNum_i + 1$
  until No slot is idle or $PQueue$ is empty
  //The 2nd step: allocating idle slots to jobs
  for each P_i in $PQueue$ **do**
    for each J_k^i in P_i **do**
      Calculating $JPriority_k$
    Sorting a job queue according to $JPriority$
    repeat
      Allocating slots to the job with the highest $JPriority$
    until No slot is idle or P_i is empty
until $PQueue$ is empty

job pool varies, the maximum slots quota can be manually adjusted by Hadoop operators.

If a job pool requests more slots, the scheduler firstly judges whether the slots occupance of the pool will exceed the quota. If not, the pool will be appended with the queue and wait for slot allocation. The scheduler allocates the slots by weighted round-robbin algorithm. Probabilistically, a pool with high allocation weight will be more likely to be allocated with slots.

2) The second step is to allocate slots to individual jobs. Each job is configured with a parameter of job priority, which is a value between 0 and 1000. The job priority and deficit are normalized and combined into a weight of the job. Within a job pool, idle slots are allocated to the jobs with the highest weight.

$$W_i = \alpha \cdot NP_i + \beta \cdot ND_i \qquad where \qquad \alpha + \beta = 1$$

where NP_i denotes the priority of the job i, and ND_i denotes the deficits of the job i. Both values are normalized into the interval of [0,1] by dividing the maximum values across all jobs.

To clarify the description of Fair4S, we present the pseudo-code of Fair4S in Algorithm 1.

7 Evaluation

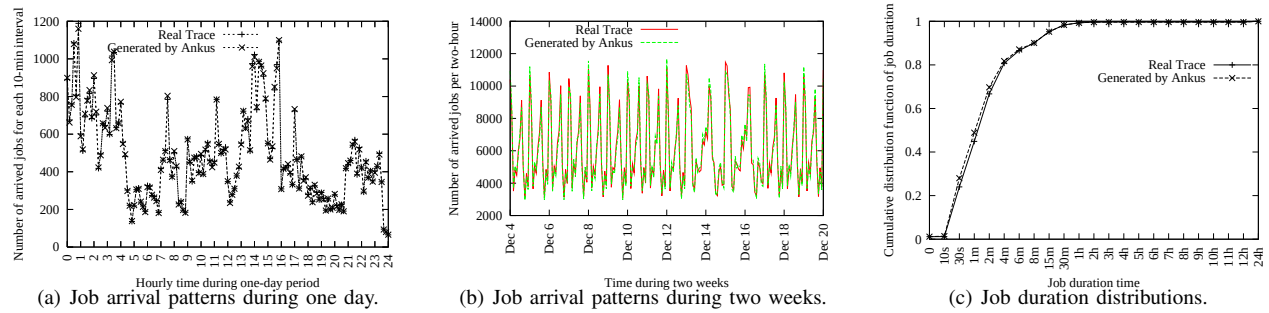
In this section, we first conducted a group of experiments to validate the representativeness of workload generated by Ankus, and then evaluate the performance of Fair4S.

7.1 Evaluation of Ankus

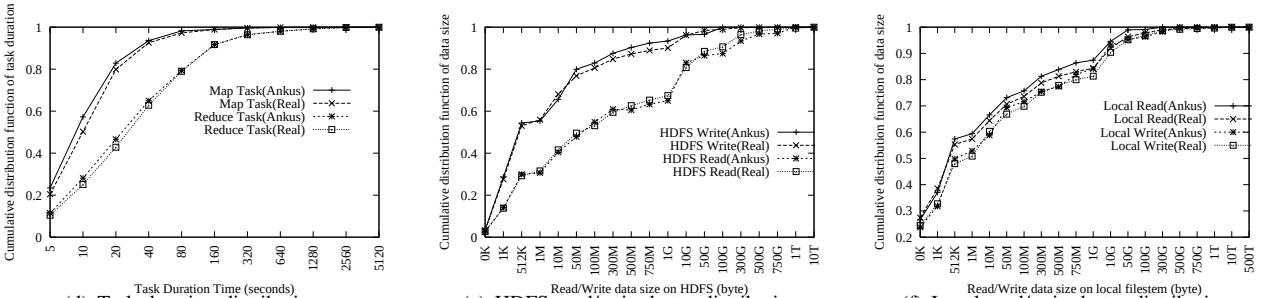
To validate the representativeness of the workload generator, we need to measure the similarity between the generated workload and real-world trace. Unfortunately, there is no existing metric for measuring the similarity of two workloads yet. In this study, we conducted a comparison between the generated workload and real-world trace in terms of job arrival rate and duration time. The purpose of comparison is to confirm the synthesized workload generated by Ankus expresses realistic characteristics.

Figures 12(a) and 12(b) shows the comparison results in terms of job arrival rate. To investigate these statistical properties, we firstly generate a day-long workload using Ankus, which contains 64,238 jobs. Ankus samples the historical traces for every 10 minutes. All of the jobs are submitted to a Hadoop cluster for performance testing. Then, we use Ankus to replay all the job arrivals observed in the trace collected from Dec. 4 to Dec. 20. As shown in these two figures, the workload generated by Ankus does introduce a degree of statistical variation, but it is relatively small. That is to say, Ankus supports producing the realistic workload with representative characteristics.

Figures 12(c) and 12(d) depict the CDF of job duration and task duration for 10,000 jobs generated by Ankus. The 10,000 jobs are generated based the collected trace described in Section 2.2. We did not generate a entire day's workload because it is not allowed to impose too much extra workload on the production Hadoop cluster. All generated jobs are submitted to Yunti, which executes those jobs as "real" jobs. Ideally, the average of duration of the jobs and tasks are independent with the generation of jobs. JobTracker and TaskTrackers are unaware of the source of the jobs. Therefore, the distribution of job duration and task duration should keep unchanged. As we expected, Figures 12(c) and 12(d) have confirmed the inference. These figures show the small deviation of distribution statistics between the real-world trace (the curve labeled by *real*) and the generated workload (the curve label by *Ankus*). To quantitatively measure the deviation between these two curves, we use Kolmogorov-Smirnov (KS) test method and the corresponding KS values are presented in Table 7.



(a) Job arrival patterns during one day. (b) Job arrival patterns during two weeks. (c) Job duration distributions.



(d) Task duration distributions. (e) HDFS read/write bytes distribution. (f) Local read/write bytes distribution.

Fig. 12: Characterization comparison of the real workload and the synthetic workload.

TABLE 7: The quantitative comparison results using Kolmogorov-Smirnov test.

CDF of distributions	KS value
Job Duration (Figure 12(c))	0.04
Map Task Duration (Figure 12(d))	0.07
Reduce Task Duration (Figure 12(d))	0.04
HDFS Read Bytes (Figure 12(e))	0.03
HDFS Write Bytes (Figure 12(e))	0.04
Local Read Bytes (Figure 12(f))	0.032
Local Read Bytes (Figure 12(f))	0.028

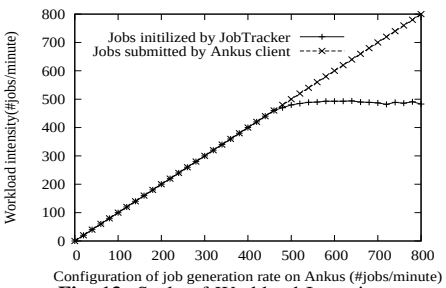


Fig. 13: Scale of Workload Intensity.

Figures 12(e) and 12(f) present the CDF of read/write bytes on HDFS and local disks for the 10,000 jobs. We can observe the is a small deviation between the real jobs and generated jobs (Table 7). This variation can be explained as follows: some jobs in the workload are date-related, and the data in HDFS grows everyday. Given a job, if it is executed repeatedly at different times, the data involved by the job will change, thus the read/write bytes from HDFS/local disks would change as well.

Figure 13 depicts the scalability of workload intensity generated by Ankus. The X-axis represents the value of job generation rate configured in Ankus, and the Y-axis represents the workload intensity, which is measured by the number of arrived jobs per minute. Ankus samples a set of jobs from the collected trace and submits the jobs to the JobTracker of Yunti with a specific rate. Meanwhile, both the count of jobs initialized by JobTracker and the count of jobs generated by Ankus are observed. As shown in Figure 13, the number of generated by Ankus increases linearly, but the number of initialized by the JobTracker becomes convergent when the JobTracker is overloaded. JobTracker consumes a great deal of effort on job initialization and tracking the heartbeats messages, so it is more susceptible to become a bottleneck compared with the Hadoop clients. The results presented in the figure illustrated Ankus could provide high enough scalability. Moreover, the scalability of Ankus can be further increased by adding more clients for submitting jobs.

7.2 Evaluation of Fair4S

In this subsection, we evaluate the scheduling performance of Fair4S. As FAIR is the default scheduler and most commonly used scheduler of Hadoop, and FIFO achieves a high performance in terms of batch response time [3], we selected FIFO and FAIR as references to compare.

7.2.1 Extending Ankus with a MapReduce Simulator

Fair4S has been implemented to the Yunti since April, 2011. Before that time, Yunti employed FAIR [3] as the scheduling algorithm. To measure the improvement achieved by Fair4S, a comprehensive performance comparison between Fair4S and FAIR is necessary. However, it is infeasible to obtain the accurate comparison results on the production systems[2]. To facilitate the performance evaluation, we ex-

2. The reason is two-fold: it is not allowed to switch the scheduler on a production environment; the historical statistics of the Yunti using FAIR is not appropriate for reference because the workload, cluster and data volume have scaled up by multiple times.

tend Ankus with a component of a MapReduce simulator which mimics a Hadoop cluster with thousands of nodes based on a small number of physical nodes.

The input of the MapReduce simulator involves the topology of a cluster, cluster scale, network transfer speed, node capacities, and so on. Each node is modeled by a set of configurable parameters related to various resources. The output of the simulator is a log for describing how the workload executed under a specified Hadoop configuration.

The MapReduce simulator simulates the computing nodes and Map/Reduce tasks, and models them by a set of features affecting the job execution time. The map task and reduce task are simulated by two Java objects of *SimMapTask* and *SimReduceTask*, respectively. Both the *SimMapTask* and *SimReduceTask* objects are driven by a discrete-event engine. For example, to model the process of copy input chunks to local disks from HDFS, a *SimMapTask* adds an event into the event engine, and when the event gets timeout, the event engine would inform the *SimMapTask* that the input data is ready. MapReduce simulator also provides the simulation of HDFS, which is modeled by three parameters in Ankus: replication level, block size and data locality strategy.

7.2.2 Evaluation of Fair4S

We conducted a group of experiments to evaluate the effectiveness of Fair4S in comparison with FIFO and FAIR using the MapReduce simulator. Different scheduling algorithms are incorporated in the simulator component of Ankus, and then the workload generated by Ankus are loaded to drive the simulator.

Figure 14(a) depicts the workload execution process 15 minutes at the interval of 0 a.m. to 4 a.m. We used Ankus to replay the workload trace collected from the Yunti. At this interval, most of submitted jobs are large jobs, which are executed periodically everyday. As shown in Figure 14(a), Fair4S and FIFO achieved similar workload execution efficiency.

Figure 14(b) shows the throughput per 15 minutes for both Fair4S and FIFO. In this comparison, only large jobs were generated and executed. It is observed that performance achieved by Fair4S is comparable to the one of FIFO.

Figure 14(c) presents the workload execution process per one minute. The small jobs were selected from the temporary jobs submitted on the daytime. Then, the small jobs were synthesized and submitted to the Yunti. We record the start time and finish time for each job, and concluded that Fair4S achieved a remarkable improvements for scheduling small jobs compared with FAIR.

Figure 14(d) shows the cumulative distribution of waiting time for small jobs. Using Fair4S, over 80 percent of small jobs wait less than 4 seconds, and the median job waiting time is about 1.3 second. While using FAIR, only about 30 percent of small jobs wait less than 4 seconds, and the median job waiting time is about 9 second. Therefore, the scheduling efficiency achieved by Fair4S was improved by a factor of 7.

8 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 systems [21], [22], storage systems [23], [24], Web servers [25], [26], and HPC clusters [27]. 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.

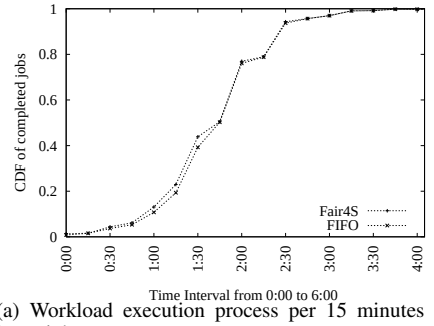
Several studies [28], [29], [30] have been conducted for workload analysis in grid environments and parallel computer systems. They proposed various methods for analyzing and modeling workload traces. However, the job characteristics and scheduling policies in grid are much different from the ones in a Hadoop system [3].

Mishra *et al.* [31] focused on workload characterization that includes behavior characteristics such as CPU and memory. The Yahoo Cloud Serving Benchmark [32] focused on characterizing the activity of database-like systems at the read/write level. Soila Kavulya *et al.* [11] 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.

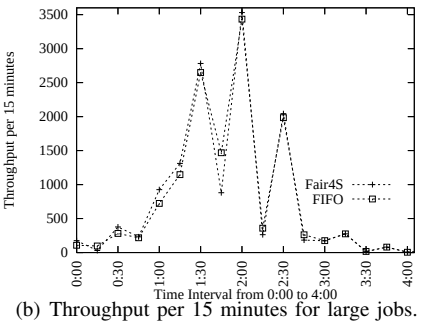
However, these studies do not analyze many of the workload characteristics discussed in this paper. For example, we report not only the job statistics but also task statistics. We also derive some direct implications on the observations of the MapReduce workload trace, and present resource utilization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

Reiss *et al.* [33] analyzed the google cluster trace and found several significant characteristics including: workload heterogeneity; highly dynamic resource demand and availability; predictable resource needs and so on. Ren *et al.* [34] analyzed Hadoop workloads from three different research clusters, where the main users are from academic institutions. The workload analysis is conducted in the terms of application workload, use behavior, IO performance, and load balance. Our research contributions, which are derived on a commercial e-commerce website, can be regarded as a meaningful complement to these works in [33], [34].

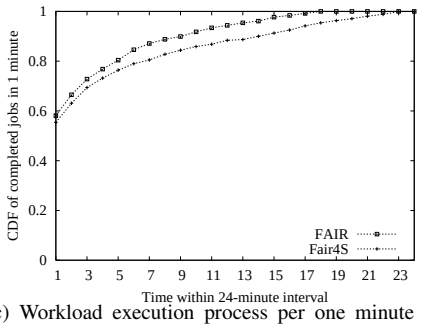
Chen *et al.* [20] analyzed and compared two production MapReduce traces derived from Internet service companies, in order to develop a vocabulary for describing MapReduce workloads. The authors show that existing benchmarks fail to reproduce synthetic workloads that express such characteristics observed 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 jobs and tasks. Our analysis of MapReduce workloads corroborates observations from previous studies of work-



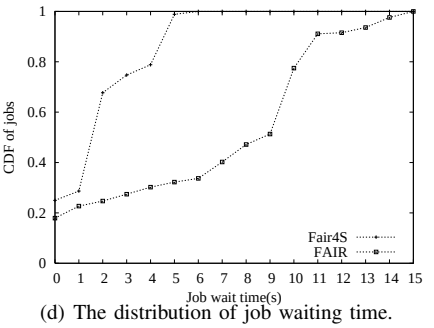
(a) Workload execution process per 15 minutes for large jobs.



(b) Throughput per 15 minutes for large jobs.



(c) Workload execution process per one minute for small jobs.



(d) The distribution of job waiting time.

Fig. 14: Scheduling performance comparison among Fair4S, FAIR and FIFO.

load on the Hadoop system [35], [11].

Wolf *et al.* [36] designed a flexible scheduling allocation scheme FLEX for Hadoop. Through collaboration with the default scheduler FAIR, FLEX provides to optimize the various scheduling metrics, such as response times, stretch, Service Level Agreements, while ensuring the same minimum job slot. The research of FLEX is orthogonal to our work, and it is also applicable for collaborated with Fair4S as an add-on module.

Although our work is close to the study presented in [35], [11], this paper has three novel aspects. (1) The jobs analyzed in this paper are representative and common in a 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. (3) We proposed and implemented a job scheduler called Fair4S, to optimize the completion time of small jobs.

9 Conclusions

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, which are representative and common in data platform for an e-commerce web site. We conduct a comprehensive analysis of the workload trace at the granularity of jobs and tasks, 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. Based on the characteristics we observed from the real-world traces, we designed and implemented Ankus to synthesize representative jobs. We proposed and implemented Fair4S, a priority-based fair scheduler aiming to optimize the efficiency of scheduling small jobs.

Acknowledgment

We would like to thank the Hadoop team at Taobao for practical experiences that guided this work. We are grateful to the anonymous reviewers for their helpful suggestions and constructive feedback. This research was supported by NSF of Zhejiang (LQ12F02002). 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] 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.

[4] Y. Chen, S. Alspaugh, and R. H. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *CoRR*, vol. abs/1208.4174, 2012.

[5] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou, "Workload characterization on a production hadoop cluster: A case study on taobao," in *IEEE IISWC*, 2012.

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

[7] 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.

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

[9] J. J. More, "The levenberg-marquardt algorithm: Implementation and theory," in *Dundee Conference on Numerical Analysis*, G. A. Watson, Ed. New York: Springer-Verlag, 1978.

[10] M. A. Stephens, "EDF statistics for goodness of fit and some comparisons," *Journal of the American Statistical Association*, vol. 69, no. 347, pp. 730–737, 1974.

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

[12] A. Verma, L. Cherkasova, and R. H. Campbell, "Play it again, SimMR!" in *CLUSTER*. IEEE, 2011, pp. 253–261.

[13] 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.

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

[15] 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.

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

[17] "Gini coefficient," in *Encyclopedia of Machine Learning*, 2010, pp. 457–458.

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

[19] J. A. Hartigan and M. A. Wong, "Algorithm AS136. A K-means clustering algorithm," *Applied Statistics*, vol. 28, pp. 100–108, 1979.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

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

[31] 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.

[32] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, 2010, pp. 143–154.

[33] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *SoCC*, 2012, p. 7.

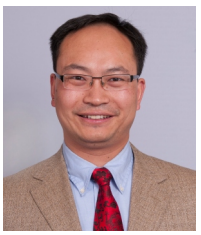
[34] K. Ren, G. Gibson, Y. Kwon, M. Balazinska, and B. Howe, "Hadoop's adolescence; a comparative workloads analysis from three research clusters," in *SC Companion*, 2012, p. 1452.

[35] Y. Chen, S. Alspaugh, and R. H. Katz, "Design insights for mapreduce from diverse production workloads," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-17, Jan 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-17.html

[36] J. L. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin, "Flex: A slot allocation scheduling optimizer for mapreduce workloads," in *Middleware*, 2010, pp. 1–20.

Zujie Ren is an assistant professor of Computer Science and Technology at the Hangzhou Dianzi University in China. He is currently working at the cloud computing research institute of Hangzhou Dianzi University. He received his PhD degree in 2010. His research interests include distributed database systems, peer-to-peer networks, information retrieval and massive data processing.

Jian Wan is the director of Grid and Service Computing Lab in Hangzhou Dianzi University and he is the dean of School of Computer Science and Technology, Hangzhou Dianzi University, China. Prof. Wan received his PhD degree in 1996 from Zhejiang University. His research areas include parallel and distributed computing systems, virtualization, grid computing.



Weisong Shi is a professor of Computer Science at Wayne State University. His research interests include computer systems, mobile and cloud computing, wireless health. He received his BS from Xidian University in 1995, and PhD from Chinese Academy of Sciences in 2000, both in Computer Engineering. He is a recipient of National Outstanding PhD dissertation award of China and the NSF CAREER award.

Xianghua Xu is a professor at Hangzhou Dianzi University and the vice director of Grid and Service Computing Lab in Hangzhou Dianzi University. Prof. Xu received his PhD degree in 2005 from Zhejiang University, China. His research areas include service computing, parallel and distributed computing system, virtualization, grid computing. Dr. Xu is a member of China Computer Federation (CCF) and the IEEE.

Min Zhou is a senior engineer and member of technical committee in Taobao, Inc. Previously, he was the lead of Taobao Hadoop development team, working on the design and optimization of Yunti. He is currently the lead of Taobao's realtime analytic processing team. His research interests include distributed systems, parallel computing systems and bytecode based virtual machines.