WaxElephant: A Realistic Hadoop Simulator for Parameters Tuning and Scalability Analysis

Zujie Ren, Zhijun Liu, Xianghua Xu, Jian Wan
School of Computer Science and Technology
Hangzhou Dianzi University, China
{*renzj,liuzhijun87,xhxu,wanjian*}*@hdu.edu.cn*

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

Min Zhou
Taobao Inc.
zhouchen.zm@taobao.com

***Abstract*—MapReduce is becoming the state-of-the-art computation paradigm for processing large-scale datasets on a large cluster with tens or thousands of nodes. Hadoop, an open-source implementation of MapReduce framework, has gained much popularity due to its high scalability and performance. Two challenging issues for a large-scale Hadoop cluster are how to analyze the scalability and identify the optimal parameters configurations. To address these issues, we designed and implemented a Hadoop simulator called WaxElephant, which provides the following capabilities: (1) loading real MapReduce workloads derived from the historical log of Hadoop clusters, and replaying the job execution history; (2) synthesizing workloads and executing them based on statistical characteristics of workloads; (3) identifying the optimal parameters configurations; and (4) analyzing the scalability of the cluster. Extensive experiments have been conducted to validate the accuracy of the WaxElephant simulator.**

***Keywords*-MapReduce, Hadoop simulator, Parameters tuning**

I. Introduction

MapReduce [4] is becoming the state-of-the-art computation paradigm for processing large-scale datasets on a large cluster with tens or thousands of nodes. MapReduce 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 MapReduce framework [14], 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. A lot of organizations, such as Yahoo, Facebook, Twitter, and many research groups adopt Hadoop to run their data-intensive jobs.

However, the issues of parameters tuning and scalability analysis of a Hadoop cluster are complex and challenging. The Hadoop system has hundreds of configuration parameters, which bring forth many difficulties for Hadoop operators to figure out a set of optimal parameter configurations, including the number of slots, the size of buffers etc. In most cases, the evaluation experiments for identifying optimal parameter setting cannot be conducted in a online production cluster. It is also infeasible to set up a large testing cluster with hundreds or thousands of nodes to conduct the experiments for most companies. One common solution to address the above problem is to build a small cluster with tens of nodes, and perform experiments in a real MapReduce testbed. However, evaluation conducted on a small-size cluster of nodes is hard to measure the scalability and performance accurately. Another solution commonly used is to utilize a simulator that simulates the environments of Hadoop and the execution flow of MapReduce jobs. The simulator enables to deploy a simulated Hadoop with thousands of nodes at a low cost.

Although there are many simulators available for simulating various distributed systems, such as GridSim [2], GPS [15], P2PSim [8], NS-2 [7] and CloudSim [9], the simulation of Hadoop environment has not been well-studied. There are only a few simulators specifically designed for MapReduce environments [12][6][11]. However, after detailed analysis and test, we find that the capabilities of these simulators are not satisfactory due to their simplified simulated behaviors.

In this paper, we present a realistic MapReduce simulator, called WaxElephant [13], which realizes the following capabilities:

- loads real MapReduce workloads derived from a historical log generated on production Hadoop clusters, and replaying the job execution history. WaxElephant also can synthesize workloads and execute them based on statistical characteristics of workloads.
- tunes various Hadoop parameters which affects the system performance. WaxElephant is built on a simulated cluster and computing nodes, with a set of configurable parameters, making the task of parameters easy-tuning.
- provides job execution simulation at the granularity of tasks. The simulation slots allocation and job scheduling reaches on task-level.
- compares the job scheduling polices. The job scheduling module in WaxElephant is pluggable, which is convenient for comparing different job scheduling policies in the Hadoop.
- simulates a large-scale Hadoop cluster, carry out performance tests at a low cost, thus evaluating its scalability and performance quickly and accurately.

The rest of this paper is organized as follows. Section II presents the motivations of our work. Section III presents the design goals and the overall architecture of WaxElephant. Section IV describes a detailed analysis of workload characterization based on a two-week trace from a production cluster. In Sections V and VI, we focus on the simulation of fundamental components of WaxElephant, including the workload generator and simulation of cluster. The simulation of task execution is described in Section VII. The evaluation of WaxElephant is given in Section VIII. Section IX describe the related work. Section X concludes this paper.

II. Background and Motivations

Our work was originally motivated by the Hadoop cluster of Taobao Inc. [11], named Yunti, which is an internal data platform for processing petabyte-level business data derived from http://www.taobao.com. Yunti cluster is based on Hadoop 0.19 with slight modifications. It is used to provide multiuser business large-scale data analysis service for some online applications and company staffs. Within Yunti platform, system logs, crawled pages and replicas of online databases, are gathered continuously. These data are used by numerous applications, including traffic statistics, product sales trends and recommender systems. This data warehouse runs on more than 2,000 nodes and stores more than 25PB of compressed data which is growing at the rate 30TB per day.

The first problem that motivates this work is how to optimize the performance of Yunti. The large number of configuration parameters of Yunti cluster makes the task of parameter tuning much difficult. Identifying a set of parameters that would help to reach an optimal performance is very time-consuming and labor-intensive.

The second motivation is scalability analysis about the Yunti cluster. It is a common experience that the numbers of MapReduce jobs will raise suddenly with new applications emerge. For instance, around the days of big promotion, a lot of MapReduce jobs specifically for the big promotion will be submitted. Yunti operators have to confirm whether additional servers are needed, and then how they should allocate resources for satisfying performance requirements.

Due to the infeasibility of setting up a same-scale physical Hadoop cluster for performance test, a realistic and efficient simulating tool is desired to facilitate the scalability analysis and parameters tuning. Existing simulators, such as Mumak [6], MRPerf [12], and SimMR [11], commonly have two limitations, thus fail to satisfy the requirements.

1) Lack of diverse workloads. Existing simulators fail to synthesize realistic and diverse workload that observed in traces. These simulators focus on making decisions about cluster configuration and run-time parameters that affect performance. The results obtained from simulation studies are less accurate due to the lack of realistic traces.
2) Over-abstractions and simplifications of some important factors. For instance, in Mumak, task execution is simplified, without modeling of shuffle and sort phase. The resource utilization, such as CPU, memory and network utilization re also omitted. MRPerf simplifies the task execution process and fails to simulate the complete progress of map tasks and reduce tasks.

To address these above limitations, we collected a workload trace from a production Hadoop cluster, and analyze the trace, especially the job arrival pattern, duration of task and job, job size etc. Basing on these analysis results, we obtain some observations and direct implications, which can help the simulator to synthesize realistic workload. Then, we implement WaxElephant by simulating the complex behaviors and various kinds of parameters within Hadoop environments. Up to now, WaxElephant has been utilized for Taobao engineers to generate high-intensity workload, in order to predict whether the system performance of Yunti under extreme heavy workload can satisfy the requirements.

III. Design of WaxElephant

A. Design Goals

A key challenge for implementing a realistic MapReduce simulator is to figure out the right level of abstraction. A complete simulation of every entity and every interaction in Hadoop will consume large resource and take a long time to obtain the results. In converse, an over-abstraction of the important entity and interactions will impact the accuracy of results.

To make the Hadoop simulator more practical and effective, we focus on two goals for designing WaxElephant.

1) First, WaxElephant should be used to tune the performance and analyze the scalability of a Hadoop cluster at a high accuracy. Therefore, those important steps and parameters should be taken into consideration and simulated completely.
2) Second, WaxElephant should be easy to use and effective to conduct performance test. If the trace logged from a couple of weeks forms a huge number of MapReduce jobs, it is impractical to execute these jobs at the same speed. WaxElephant should be quickly replaying workloads with different scenarios of interest, assess various what-if questions.

B. Architecture of WaxElephant

Figure 1 depicts the overall architecture of WaxElephant. WaxElephant is composed by a pluggable job scheduling module, a load generator and a simulator engine. The job scheduling module receives jobs from the load generator and monitors the status of computing nodes. The load generator is responsible for producing MapReduce jobs, either replaying jobs collected from a historical log, or synthesizing a set of jobs that follow a user-defined statistical properties. The

Table I
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
Average maps per job	42
Average reduces per job	12
Average nodes per job	19
Maximum nodes per jobs	489
Average job duration	350s
Maximum job duration	16h24m

simulator engine is responsible for simulating computing nodes and job execution process.

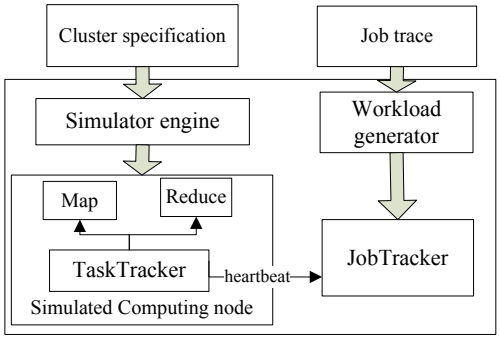


Figure 1. Architecture of WaxElephant.

IV. WORKLOAD CHARACTERIZATION

A. Description of Trace

The trace is collected over a two-week period from Dec. 4 to Dec. 20 2011. During this period, over 912,157 MapReduce jobs are generated by 572 different users belonging to fifty-six groups. Most of the jobs are submitted automatically and started execution at a regular time of a day. Table IV-A gives an overview of the dataset. On average, each job consisted of 42 map tasks and 12 reduce tasks running on 19 nodes. Maximum number of nodes allocated to a job is 489.

B. Job Statistics

1) Job Arrival Rate: Figure 2(a) shows the number of arriving jobs per 10-minute interval during a one-day period. The workload reaches the first daily-peak during 1:00 am to 4:00 am, and reaches the second peak during 1:00 pm - 4:00 pm. The maximum value of arriving jobs per 10-minute interval exceeds 1,100. The first workload peak is mainly formed by periodical jobs, and the second one is formed by temporary jobs. Figure 2(b) depicts the arriving job count per two-hour interval during two weeks. As shown in the figure, the job arrival pattern expressed on every day is very similar. **Observation 1.** Job arrival rate follows a relative-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.

2) Jobs Count on Running Status: Figure 2(c) shows the changes of concurrently running jobs count loaded on Yunti cluster during a one-day period. It is sampled per 5 seconds. The number of running jobs exceeds 600 at the peak time, and then falls down to approximately 300 at 5:00 am. 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 awaiting-state. **Observation 2.** As depicted in the Figure 2(c), we can infer that preemptions are frequent. The impact of the job completion time caused by slot preempted should not be ignored.

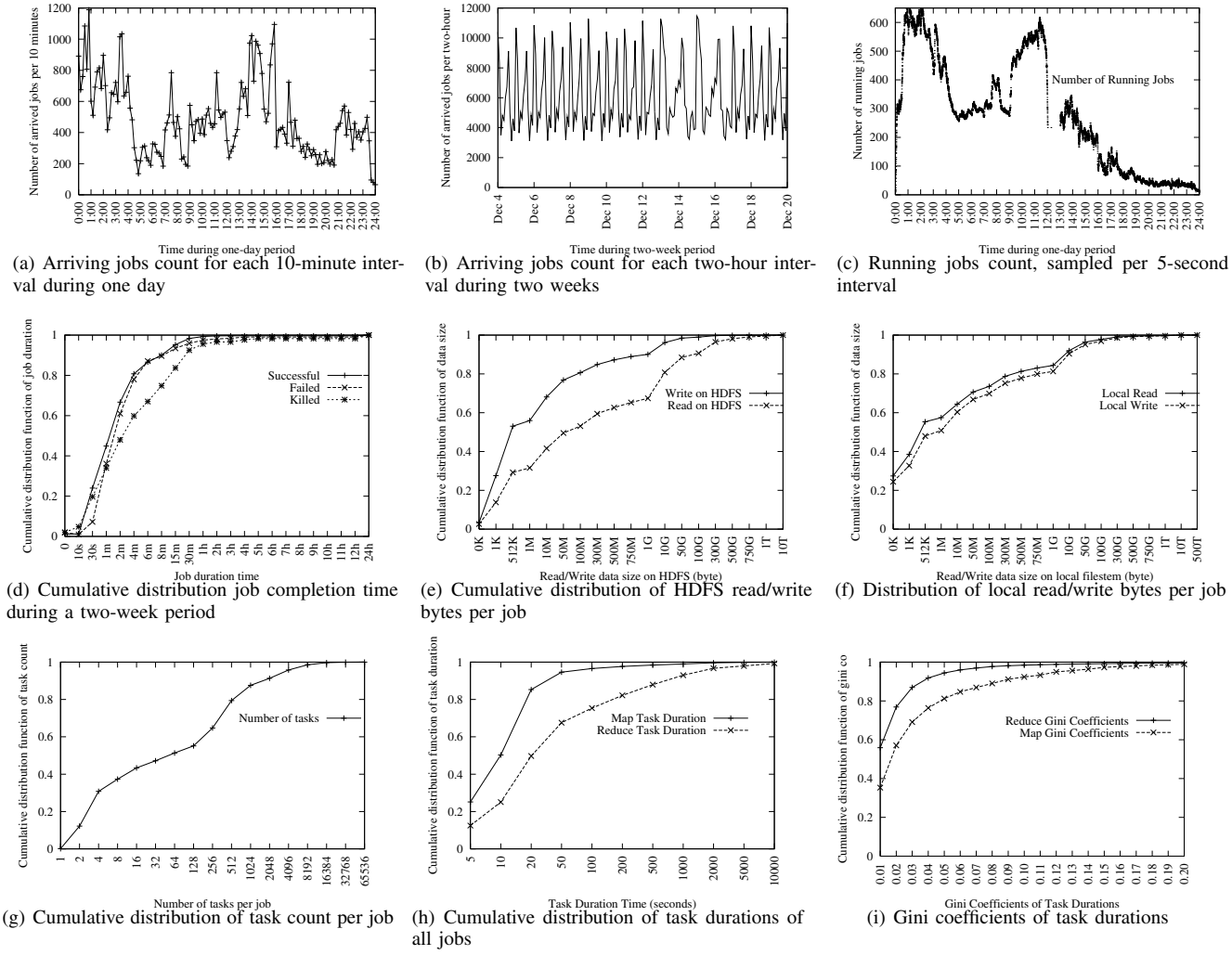
3) Job Completion Times: Figure 2(d) presents the cumulative distribution of completion time of successful jobs, failed jobs, and killed jobs during the two-week period. Failed jobs are jobs that were canceled jobs due to task failure. Killed jobs are jobs that were killed by cluster operators due to slots preemption or some other reasons. Intuitively, the jobs completion time follows a long-tail distribution. **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.

4) Job Data Size: Job executions bring forth plenty of data read/write operations at various stages of the MapReduce. The total bytes of data read/writes involves two parts, *HDFS read/write bytes* and *local read/write bytes*, which represent the bytes of data read/writes on HDFS and local disks respectively. Figures 2(e) and 2(f) show the cumulative distribution of HDFS and local read/write bytes per job. **Observation 4.** 80% of jobs write small files (#size <64MB) to HDFS.

C. Task Statistics

1) Task Count per Job: Figure 2(g) 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's task count ranges from 10 to 2000. The largest job consists of 91,798 tasks, and the smallest job contains only one map task. **Observation 5.** The task counts of jobs in Yunti are diverse. Small jobs take majority, and medium and large jobs account for a considerable part of the jobs analyzed. Thus, Hadoop schedulers need to be improved for scheduling small jobs.

2) Task Duration: Figure 2(g) shows the cumulative distribution of the map and reduce task duration. More than 50% of tasks are executed for less than 10 seconds. The



(a) Arriving jobs count for each 10-minute interval during one day

(b) Arriving jobs count for each two-hour interval during two weeks

(c) Running jobs count, sampled per 5-second interval

(d) Cumulative distribution job completion time during a two-week period

(e) Cumulative distribution of HDFS read/write bytes per job

(f) Distribution of local read/write bytes per job

(g) Cumulative distribution of task count per job

(h) Cumulative distribution of task durations of all jobs

(i) Gini coefficients of task durations

Figure 2. Job statistics during the two-week interval

longest task lasts 20 minutes and 48s. **Observation 6.** Both map task and reduce task duration time follow a log-normal distribution.

3) Task Equity: Various methods have been proposed to measure the equity for a distribution. Gini coefficient is the most commonly one. Figure 2(i) 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.

V. Workload Generation

In this section, we firstly present the modeling of an individual MapReduce job, and then discuss how to generate jobs from workload traces and how to synthesize workloads.

A. Modeling a MapReduce job

According to former discussion in section IV, we show that an individual MapReduce job is described by several features from the real cluster traces C the jobs input, output, local I/O, its running time in seconds, and it map and reduce duration, here we define the map/reduce duration as the total running time of all map/reduce tasks, that means the map duration of a job has 2 map task that each takes 10 seconds is 20 seconds. Formally, a MapReduce job is modeled by a vector as $Job_i = \{I_i, O_i, LIO_i, JD_i, MD_i, RD_i\}$, where the meanings of these symbols are presented in Table V-A.

B. Job Generation

WaxElephant supports diverse workloads, which is formed by various job groups. For example, according the job completion time, the jobs can be grouped into second-jobs,

Table II
SYMBOLS AND DESCRIPTION

Symbol	Description
Job_i	The i-th MapReduce job
I_i	The input scale of the i-th job in bytes
O_i	The output scale of the i-th job in bytes
LIO_i	The local read and write of i-th job in bytes
JD_i	The running time of the i-th job in seconds
MD_i	The total running time of all map tasks
RD_i	The total running time of all reduce tasks

minute-jobs, hour-jobs and day-jobs. A representative job is the job that has similar statistical characteristics with the some other jobs in the terms of job completion time. In this section, we investigate how to describe a real cluster workload using a small set of common jobs. We use k-means, a well-known data clustering algorithm, to extract such information.

As discussed former, we describe each job using all 6 features available from the cluster traces - the jobs input, output and local I/O data sizes in bytes, its running time in seconds, and its map and reduce durations. We linearly normalize all data dimensions to a range between 0 and 1 to account for the different measurement units in each feature. Then input the array of all jobs into k-means. K-means finds the natural clusters of data points, i.e., jobs. We consider jobs in the same cluster as belonging to a single equivalence class, and can be described by the natural clusters of data points, that is, the common job.

The more common jobs in the generated workload, the more accurate the workload is. But having too many clusters will lead to an unwieldy number of common jobs, even though the variance explained will be higher. So to find the optimal k, the number of clusters, we increment k, until there is diminishing improvement in the cluster quality. We measure cluster quality by variance explained, a standard metric computed by the difference between the total variance in all data points and the residual variance between the data points and their assigned cluster centers.

C. Simulation of Workload

WaxElephant supports simulation of a large number of concurrent jobs. The workload is defined by the following parameters.

- Job submission pattern. The submission pattern is characterized by number of submitted job per second. We use a list of $<T_k$,$J_i>$to describe a job submission pattern.
- Distribution of representative jobs. After extracts the common jobs, we count the percentage of jobs that each cluster owns. And the workload generator would generate corresponding number of jobs according to the common job. For example, WaxElephant would generate 20 jobs of kind A if its occupancy is 20% in the workload and we need a 100 jobs workload.

Workload generation from a real job trace contains the following three steps. Firstly, the total count of jobs in the job trace is counted and the job submission pattern is extracted. Then, the duration of the job trace is split into several time intervals according to a given time unit (5 seconds by default). Finally, we perform a clustering on the job trace based on one of the job characteristics, and for each cluster we record the statistical characteristics for every job characteristics, e.g. the minimum and maximum values, and the percentage of jobs that are clustered into the cluster.

VI. SIMULATION OF HADOOP CLUSTER

Simulating a realistic MapReduce cluster is a challenging work. One of the keys is to figure out the right level of abstraction. Cluster nodes are the fundamental components of a Hadoop application. The simulation schema of nodes greatly determines the accuracy and effectiveness of the Hadoop simulator. In WaxElephant, each node is modeled by a set of configurable parameters related to various resources. Some parameters for the whole cluster define the cluster topology, network transfer speed, etc.

A. Simulation of nodes

- Processors and cores. By default, each node in WaxElephant has one single-core processor. However, the number of cores and processors could be configured. One node can have one or more processors, and each processor can have multiple cores. Besides, the capability of each core is defined by the volume of data processed per second instead of CPU frequency.
- Memory. The simulation of memory involves three parameters, including size of memory, speed of memory reading and writing. Actually, the time cost on reading and writing data on memory can be omitted due to the their high speeds (about 10GB/s).
- Slots. 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. There are two kinds of slots exist in a computing node: map slots process map tasks and reduce slots process reduce tasks. By default, WaxElephant is configured with one map slot and one reduce slot for each core.
- Local disks. The intermediate data of map output is written to local disks. The cost for performing I/O from the local disks is an important factor that affect the job completion time. We simulate a local disk with read and write requests queue manager, which handle the requests in a FIFO manner. Let S represents the read/write speed of a local disk, the time cost for a read/write request for B bytes is estimated as the Equation 1.

$$T_{io} = \frac{b}{x} + T_{seek} \tag{1}$$

where T_{io} represents the time cost, and T_{seek} represents the seek time for the requested data.

B. Simulation of cluster

- Network and topology. The simulation of topology involves a set of parameters that define the cluster organization, which nodes located in which racks. By default, WaxElephant uses a single rack topology, in which all the computing nodes are connected by a router.
- Network transfer. The simulation of network transfer is implemented by two kinds of parameters: speed between routers, the speed between routers and computing nodes. In the shuffle phase, the output of the map tasks is transferred to the reduce nodes.
- Cluster scale. The cluster scale is defined by number of nodes, which can be ranged from 1 to 10,000.
- Job scheduler. Job scheduler is a pluggable part in WaxElephant, which supports three job schedulers: FIFO scheduler, Fair scheduler [16] and Capacity scheduler [3]. Different job scheduler produces different job execution sequence.

C. Simulation of HDFS

Simulation of HDFS is a critical issue for designing a MapReduce simulator, which has been ignored in most previous works [11][12]. MapReduce framework is deployed on the top of HDFS, which provides high-efficiency and reliable storage service. During the processing of a MapReduce job, it reads input data from the HDFS, and the final output are written to the HDFS. Therefore, we simulate the HDFS with three main parameters.

- Replication level. It means the mount of replication a file would be stored in HDFS. The replication level can significantly impact the mount of local map tasks. When improving the replication level, a map task has higher probability to become a local map task, which reads input data from local disk, and thus reduces the data transferred and save the bandwidth.
- Block size of the file system. Regularly, a big file is split into several fix-size blocks in HDFS, thus a number of map tasks can read and process the file concurrently.
- Data locality strategy. This parameter decides the block locations of a file, and impacts the target computing nodes the map tasks would be assigned to indirectly in some cases.

VII. Simulation of Task Execution

The simulation of task execution in WaxElephant greatly determines the accuracy of performance reported by the simulator. As the steps in map and reduce tasks are different, the simulation of map tasks and reduce tasks are separated.

A. Simulation of map tasks

A map slot would initialize a map task once the heartbeat message is returned with a task launch instruction. Then, the map task copies the chunk from HDFS, starting to process data according to the job specification. During the execution of map task, the intermediate data generated by the map task are firstly written to a buffer. Once the buffer is full, a back-end running operation of spilling is activated.

In most cases, the step of spilling produces a large number of I/O operations on local disks. At the step of spilling, the intermediate data is divided into several partitions according to a hash function, then each partition is transferred to one reduce node. A sorting and combining operation may be performed for each partition data if it has been defined. Finally, the spills are written to local disks. After the input data from all map tasks being processed, a merging operation is performed to aggregate the spills into a single file.

B. Simulation of reduce tasks

A typical reduce task is divided into three steps:

1) Shuffle: the map outputs are copied from the map nodes to a reduce node. Decompressing and partial merging may also be performed in this step.
2) Merge: the partitions from multiple map nodes are sorted to form the input of the reduce task.
3) Reduce: the inputs are processed according to the user-defined reduce function. The outputs are written to HDFS.

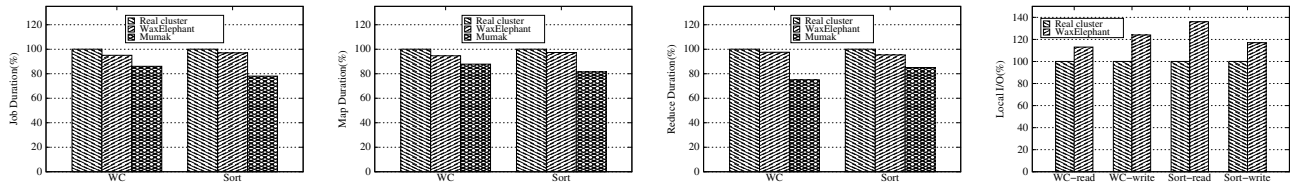
In the simulation, each map task and reduce task is abstracted as a SimMapTask and SimReduceTask respectively. Both SimMapTask and SimReduceTask are under the control of a well-designed event engine. For example, to model the process of copy input chunk to local disk from HDFS, a SMT register an event into the event engine, and when the event gets timeout, the event engine would inform the SMT that the input data is ready. The event engine is a high-performance module, and can achieve the throughput of millions TPS, thus ensure the high-performance and accuracy of the simulator.

VIII. Evaluation

In this section, we evaluate accuracy of WaxElephant by conducting several experiments on WaxElephant and compare it against the open source simulator Mumak.

A. The accuracy of individual MapReduce job

In this case, we deploy a real Hadoop cluster on 50 nodes with Intel CPU Q6600, 4GB RAM, 1 Gbps network bandwidth and running CentOS 5.6. We use Hadoop 0.19 with two nodes for JobTracker and NameNode respectively, and the other machines run TaskTrackers and DataNodes. Each TaskTracker is configured to own 4 map slots and 2 reduce slots. The block size of the file system is set to the default value, 64MB, and the replication level is set to



(a) Comparison of job duration (b) Comparison of map duration (c) Comparison of reduce duration (d) The local read accuracy of WaxElephant

Figure 3. Comparison results for a individual job between WaxElephant, Mumak and a real Hadoop cluster

3. We use WaxElephant to simulate a Hadoop cluster with the same configurations as those of the experimental cluster. Two applications are use in this experiment:

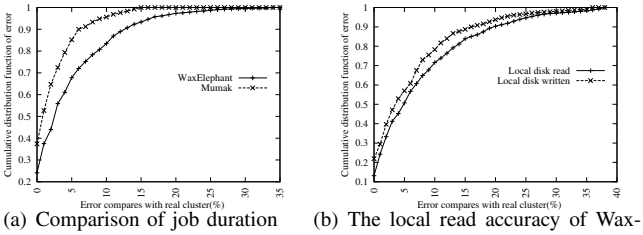
1) Word count: This application computes the occurrence frequency of each word in 5GB, 10GB and 15GB Wikipedia article history dataset.
2) Sort: The Sort application sorts 32GB, 64GB and 128GB of random data.

In this experiment, we collect a real workload trace consisting of three executions of the two applications. And then replay this trace using WaxElephant and Mumak. We validate the accuracy of WaxElephant by compare the average job and task durations and local disk I/O against that of the real cluster. We linearly normalize the results of WaxElephant and Mumak to a range between 0 and 1 by dividing the result of the real cluster.

Fig.3(a), 3(b) and 3(c) show comparisons of the job and task durations of the simulated jobs with respect to the real job and task durations across different applications. We observe that WaxElephant faithfully replays the trace with less than 5.4% error across all the applications. On the other hand, Mumak underestimates the job completion time and has more than 12.2% minimum (25% maximum) error while replaying the same trace. Fig.3(d) shows comparisons of the local I/O of the simulated jobs with respect to the real local I/O across different applications. We can confirm that WaxElephant can simulate the local I/O of applications at some level, 13% average error and 36% maximum error, while the Mumak cannot simulate those metrics.

B. The accuracy of multiple MapReduce jobs

In this experiment, we take a day-long workload trace from YunTi cluster, a production Hadoop cluster of TaoBao Inc. and the trace is consist of 67855 jobs, most of which are common Hive query jobs, and there are some other jobs such as streaming jobs, Pig jobs, and data mining jobs and so on. The YunTi cluster is consist of about 2000 nodes, and the number of available slots is about 35000. So we setup WaxElephant to 2000 nodes, each of which has 20 slots and initialize the simulated cluster with the configuration of HDFS of YunTi. In this case, we replay the job trace using WaxElephant and Mumak respectively, and then compare the results against the original trace. We linearly normalize



(a) Comparison of job duration (b) The local read accuracy of WaxElephant

Figure 4. Comparison results of multiple jobs between WaxElephant, Mumak and a real Hadoop cluster

the result of WaxElephant and Mumak to a range between 0 and 1 by dividing the result of the real cluster, and count their cumulative distribution.

Fig.4(a) shows the comparisons of the job durations of the workload trace replayed by WaxElephant and Mumak. It can be inferred from the figure that WaxElephant can replay the jobs with less than 15% error across the whole trace, and more than 94% of jobs are replayed with less than 10% error. While Mumak underestimates the job durations and results in more than 34% maximum error. We can observe that only 75% of Mumak replayed jobs can achieve less than 10% error. The errors of task durations show the same tendency, so we do not present comparison results here. Fig.4(b) shows comparisons of the local I/O of the WaxElephant replayed jobs with respect to the real local I/O across the whole trace. We observe that the error of simulated local read and written bytes are much conspicuous, with the maximum error of 36% and only about 85% of jobs have error less than 15%.

IX. Related Work

Although there are some existing tools for simulating various distributed systems, only a few simulating tools specifical for the MapReduce environment have been developed.

Mumak [6] is an open-source MapReduce simulator, which has been committed to Hadoop 0.21 and later versions. This simulator replays MapReduce workload traces collected with a log processing tool, called Rumen [1], and reproduces all conditions of a production cluster. The job submission, inter-arrival, dependencies, task completion times are obtained by Rumen. Mumak simulates a cluster

with thousands of nodes in one process. In Mumak, task executions are simplified, without modeling the shuffle and sort phase. The resource utilization, such as CPU, memory and network utilization are also omitted.

MRPerf [12] is representative MapReduce simulator for comprehending various design parameters of MapReduce. MRPerf is designed for helping Hadoop operators to know how their MapReduce jobs will behave on a specific parameters setting. According to the evaluation results presented in [12], MRPerf achieves a high accuracy in simulating the behavior of underlying network in the cluster due to its utilization of NS-2 simulator [7]. However, MRPerf simplifies the task execution process and fails to simulate the complete progress of map tasks and reduce tasks. Therefore, the simulation is difficult to reflect the behavior of a real Hadoop cluster.

Hammound *et al.* [5] developed a discrete event based MapReduce simulator, called MRSim, which uses GridSim [2] to simulate network topology and traffic, and uses SimJava [10] to simulate interactions between different entities within cluster. The authors focused on evaluating and scalability and parameter tuning. Verma *et al.* [11] build a simulator called SimMR which is capable of replaying the scheduling decisions over a large workload in a few minutes on a single machine, thereby comparing different scheduling decisions in MapReduce environment. SimMR implements a discrete event simulator engine that emulates the job scheduling and resource allocation behaviors.

X. Conclusions

In this paper we have presented WaxElephant, a Hadoop simulator that can help Hadoop operators in scalability analysis and parameters tuning. Extensive experiments have shown that the proposed WaxElephant simulator can accurate simulate a Hadoop cluster and job execution process on the Hadoop. WaxElephant can load real MapReduce workloads derived from the historical log of Hadoop clusters, and replaying the job execution history. WaxElephant also can synthesize workloads and execute them based on statistical characteristics of workloads.

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 Scientific Starting Foundation of Hangzhou Dianzi University (No.KYS055611018) and NSF of Zhejiang Province(No. LQ12F0200). 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] Rumen: a tool to extract job characterization data form job tracker logs. [Online]. Available: https://issues.apache.org/jira/browse/MAPREDUCE-751

[2] R. Buyya and M. M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *CoRR*, vol. cs.DC/0203019, 2002.

[3] Capacity Scheduler for Hadoop, "http://hadoop.apache.org/common/docs/current/capacity scheduler.html."

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

[5] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "MRSim: A discrete event based mapreduce simulator," in *FSKD*. IEEE, 2010, pp. 2993–2997.

[6] Mumak. [Online]. Available: https://issues.apache.org/jira/browse/MAPREDUCE-728

[7] NS-2. [Online]. Available: http://nsnam.isi.edu/nsnam/index.php

[8] P2PSim. [Online]. Available: http://pdos.csail.mit.edu/p2psim/

[9] Rodrigo N. Calheiros and Rajiv Ranjan and César A. F. De Rose and Rajkumar Buyya, "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," *CoRR*, vol. abs/0903.2525, 2009.

[10] SimJava. [Online]. Available: http://www.dcs.ed.ac.uk/home/hase/simjava/

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

[12] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups," in *MASCOTS*, 2009, pp. 1–11.

[13] WaxElephant. [Online]. Available: http://cloud.hdu.edu.cn/wiki/index.php/WaxElephant

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

[15] W. Yang and N. B. Abu-Ghazaleh, "GPS: A general peer-to-peer simulator and its use for modeling bittorrent," in *MASCOTS*. IEEE Computer Society, 2005, pp. 425–434.

[16] 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.