Choosing Optimal Maintenance Time for Stateless Data-processing Clusters

- A Case Study of Hadoop Cluster

Zhenyun Zhuang, Min Shen, Haricharan Ramachandra, Suja Viswesan

{zzhuang, mshen, hramachandra, sviswesan}@linkedin.com
LinkedIn Corporation, 2029 Stierlin Court
Mountain View, CA 94043 United States

Abstract. Stateless clusters such as Hadoop clusters are widely deployed to drive the business data analysis. When a cluster needs to be restarted for cluster-wide maintenance, it is desired for the administrators to choose a maintenance window that results in: (1) least disturbance to the cluster operation; and (2) maximized job processing throughput. A straightforward but naive approach is to choose maintenance time that has the least number of running jobs, but such an approach is suboptimal.
In this work, we use Hadoop as an use case and propose to determine the optimal cluster maintenance time based on the *accumulated job progress*, as opposed the *number of running jobs*. The approach can maximize the job throughput of a stateless cluster by minimizing the amount of lost works due to maintenance. Compared to the straightforward approach, the proposed approach can save up to 50% of wasted cluster resources caused by maintenance according to production cluster traces.

1 Introduction

With the rapidly growing scale of data volume, data processing is increasingly being handled by clusters that consist of multiple machines. A data processing job may take certain time to finish, hence the intermediate state (e.g., what input data are processed, what are the partial output) of the job may change over the course of the processing. The intermediate states can be persisted as the job processing runs, and the persisted intermediate state can serve useful purposes such as progress tracking. However, persisting such states also incur additional design complexity and storage overhead. Depending on whether the intermediate state can be persisted or not, data processing clusters can be characterised into two categories: *stateful* and *stateless*. Stateful clusters are able to persist intermediate state of varying granularity (e.g., percentages of processed input data), while stateless clusters do not persist such state. Examples of stateless clusters are web server clusters and Hadoop clusters.

The distinction between stateful clusters and stateless clusters goes beyond progress tracking and design complexity. One particular aspect is the impact on cluster maintenance. When cluster-level maintenance is needed, the cluster temporarily goes offline to perform hardware/software upgrade or change. After maintenance is done, the cluster goes online again and begins to serve data-processing jobs. An interesting question

is what happens to disrupted jobs due to maintenance, that is, can these jobs be resumed seamlessly? Being able to resume disrupted jobs has the advantage of avoiding repeated data processing and hence saving cost. For stateful clusters, job resuming is possible, and the degree of the saving depends on the granularity of the saved state. On the other hand, stateless clusters are unable to resume disrupted jobs. These jobs have to be started from scratch after cluster maintenance.

For stateless clusters, since cluster maintenance has to disrupt ongoing jobs and the partially finished jobs have to redo their work after the maintenance, choosing appropriate cluster maintenance time is critical for the purpose of saving computing resources. Choosing the most appropriate maintenance time is not as straightforward as people typically think. Instead, we found that the naive and straightforward approach is far from being optimal in terms of resource saving. In this work, we address this problem of choosing optimal cluster maintenance time for stateless clusters. For easy grasping the design, we use Hadoop as the use case to present our design.

Hadoop [1] clusters, being part of the data pipeline that drives many of today's business, are commonly used to carry out various types of data processing. A Hadoop cluster typically consists of one or two NameNodes, one Resource Manager [1] and up to thousands of DataNodes. The NameNode maintains the name space of the entire underlying HDFS [2], and serves as the central point of control for client interactions. Depending on Hadoop versions and configurations, the NameNodes can be configured as primary/secondary, active/standby or high availability (HA). Nevertheless, the NameNode is the single point of failure of the Hadoop cluster for non-HA setup. The Resource Manager keeps the state of Hadoop cluster resource usage (e.g., Memory and CPU) and schedules the running of submitted Hadoop jobs. Each MapReduce-based Hadoop job typically consists of a set of mapper tasks and another set of reducer tasks [2]. The mapper tasks will be scheduled first; and towards completing, the reducer tasks will be invoked to take over the data output from mappers and continue the data processing.

Hadoop cluster may occasionally need maintenance for various reasons including software upgrade (e.g., NameNode or Resource Manager upgrade), hardware failures, configuration change, and problem debugging. In this work, the notion of "Hadoop cluster maintenance" is defined as the entire cluster is not being able to run Hadoop jobs during maintenance; and we do not differentiate the causes of cluster maintenance, be it NameNode or Resource Manager. Whenever such cluster-wide maintenance is performed, all the running Hadoop jobs are destroyed and outstanding works are forfeited. Due to current limitations of Hadoop implementation, the job state is not persisted and hence the jobs cannot be resumed from last state. Once the cluster maintenance is completed, unfinished jobs require resubmission after the NameNode is started again. As a result, all unfinished jobs before maintenance will have to lose the partially done work, and the corresponding Hadoop resources (e.g., CPU, Networking) are wasted. Note that for a unfinished job, both completed and uncompleted map/reduce tasks will have to rerun.

[1] The previous version of Hadoop 1 does not have Resource Manager.

[2] There are other frameworks such as Spark based, but they are not gaining significant popularity at this time.

Though it is invariably true that an unfinished Hadoop job requires a resubmission regardless of the maintenance time chosen before it is completed, the *amount of forfeited work* varies with different maintenance time. The more forfeited work due to maintenance, the less Hadoop throughput will be expected since the lost work will be redone. The amount of forfeited work is directly affected by the *number* of mapper/reducer tasks the job has invoked and their *running time* between the job startup time and the maintenance starting time. In this work, we aim at improving Hadoop cluster throughput by minimizing the forfeited work caused by cluster maintenance.

Assuming the maintenance window length is fixed (e.g., 1 hour), the key question is *when* the maintenance should start. We further assume the maintenance is not urgent enough for an immediate maintenance, hence any maintenance window suffices as long as it is before some deadline (e.g., 1 day). This assumption in general holds for typical software-upgrade caused maintenance. Though a straightforward approach of determining the maintenance time is to look at the number of running jobs (or tasks) and choose the time when *minimum* number of jobs/tasks are *running* before the allowed deadline, as we will demonstrate later, this approach is rather naive and hence not optimal.

To improve Hadoop cluster throughput, we propose to determine cluster maintenance time based on the accumulated job progress instead of the number of running jobs. The main objective is to minimize the forfeited work while improving Hadoop throughput. We take into account the maintenance urgency and observe the amount of accumulated work in order to choose the moment to make the maintenance. Hadoop throughput is the critical performance metric analyzed. By using historical traces of a busy Hadoop cluster, we evaluate the proposal and present the significant improvements when comparing the proposal with the one where the maintenance time is chosen when the least number of jobs are running (CL-based). Based on the data, the improvement can be up to 42% in saving the wastage of Hadoop resource usage.

To summarize our work, we make the following contributions with this writing:

1. We consider the problem of determining optimal maintenance time for a stateless cluster such as Hadoop cluster. We have explained that the naive approach of "number of running jobs" is sub-optimal;
2. We propose to use *accumulated job progress* as the maintenance criteria for determining cluster maintenance time;
3. We perform experimentation and instrumentation to validate the proposal;
4. We provide analysis of key Hadoop job statistics based on one of our busiest Hadoop clusters.

For the remainder of the paper, after providing some necessary technical background in section 2, we then motivate the problem being addressed in this paper using an example scenario in Section 3. We propose the design and solution in Section 4 and perform performance evaluation and show the results in Section 5. We discuss several issues/scenarios relevant to cluster maintenance in Section 6 and present related works in Section 7. And finally Section 8 concludes the work.

2 Background and Scope

We begin by providing background information regarding Hadoop architecture and Hadoop job workflow.

2.1 Hadoop architecture and Hadoop EcoSystem

Inspired by Google File System [3], BigTable [4] and MapReduce [5], Hadoop is designed to provide distributed data storage (via HDFS [2]) and distributed data processing (via MapReduce [5]) at a massive scale. Hadoop has evolved from the core components of HDFS [2] and MapReduce to a plethora of products and tools including [6, 7]. In addition to MapReduce framework, other frameworks such as Spark [8] are also being used. Our work considers how to determine the Hadoop cluster maintenance time. Though with a focus on MapReduce framework, the problem and proposed solution also apply to other frameworks.

Hadoop has two versions as of today. In the latest version of V2, a Hadoop cluster consists of one or two NameNode, one Resource Manager Node and up to thousands of DataNodes. The NameNode manages the HDFS namespace, while Resource Manager does the job scheduling. Resource Manager works with application masters and node managers running on each DataNode to schedule and run Hadoop jobs.

2.2 Hadoop job workflow

Hadoop jobs are submitted by Hadoop clients. Once a job is submitted, Resource Managers will initiate an Application Master on a DataNode and collaborate with node managers of DataNodes to invoke a set of containers as required by the Hadoop job. Each container can run a single mapper or reducer. Mappers are firstly scheduled to run, and towards the completion, the reducers will be invoked to fetch the data output from mappers and perform reducing tasks. The data exchange between mappers and reducers are typically referred to as "shuffling".

Each container requests certain amount of memory (e.g., heap size) from the node manager. Since the memory size of the entire Hadoop cluster is typically fixed and quite limited for commodity hardware based Data Nodes, the number of concurrently running containers is also limited. [3]

The submitted Hadoop jobs will leave a state in Hadoop Job History server for later retrieval. There is typically a limit on the number of job states kept by Job History server.

3 Problem Definition and Motivation Scenarios

We provide a motivating scenario to illustrate the problem and the impact of choosing different maintenance time on Hadoop performance.

[3] Due to performance concerns, the total JVM heap size allocated to all containers on a Data Node should not exceed the physical RAM size of the node.

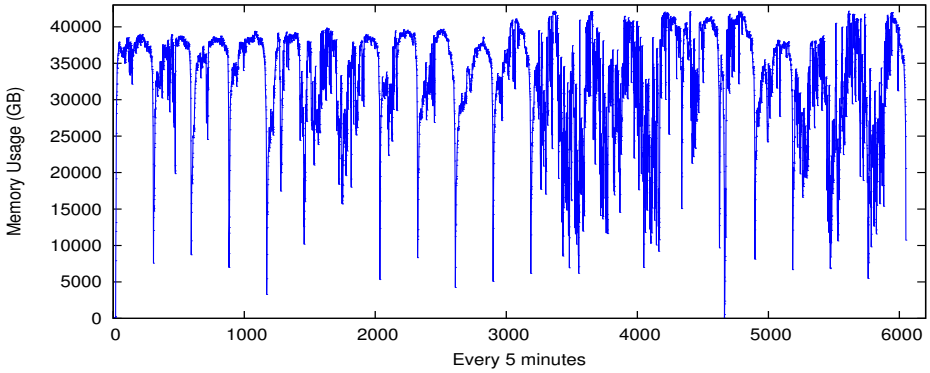


Fig. 1. Memory usage of a Hadoop cluster

3.1 Hadoop throughput is the critical performance metric

Current Hadoop implementations do not persist the job state during cluster maintenance [4]. When the Hadoop cluster is restarted from maintenance, all unfinished jobs will need to be re-submitted and start over.

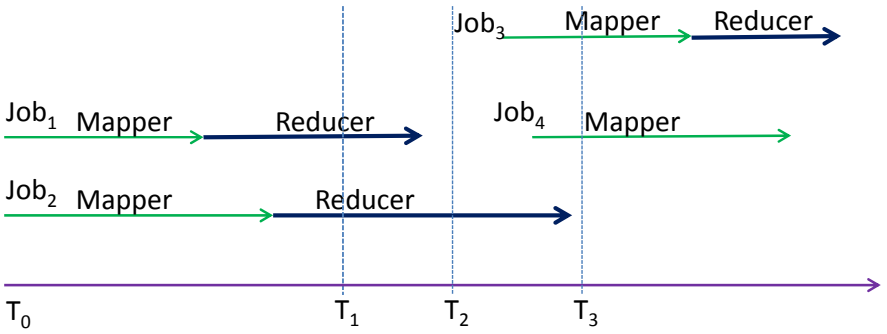
In Figure 1 we plot 21 days memory usage of a busy Hadoop cluster. This cluster has two NameNodes being configured with primary/secondary setup, and it consists of about 2000 data nodes and totally 40 TB of available memory for running Hadoop jobs. Each Hadoop container running in this cluster on average takes about 2GB of memory. We can see that most of the time the cluster is saturated with memory usage, hence the throughput is one of the critical performance metric we want to optimize. We also see that cluster maintenance is occasionally performed, which can be seen from the close-to-zero memory usage period.

Hadoop clusters are typically deployed for batch data processing and used by multiple clients. Thanks to the exploding size of today's data and the increasing demand for data processing driven by various business requirements, the throughput of Hadoop clusters (i.e., number of jobs completed in an unit of time) is the primary performance metric we should optimize.

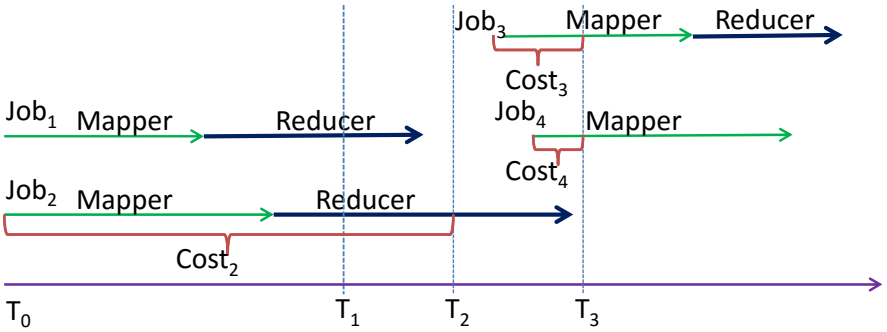
3.2 Problem we are addressing in this work

Without careful considerations of the impact of different cluster maintenance time, a poorly selected maintenance window will result in suboptimal Hadoop performance in the forms of low job throughput and wasted computing resources. We have seen cases that the Hadoop cluster maintenance window is chosen by Hadoop administrators in a rather ad hoc fashion. Though urgent remedies of the cluster require immediate maintenance, most of the maintenance requests including minor software upgrade are non-urgent, hence we can afford a delayed maintenance up to certain deadline.

[4] Efforts are going on to allow job state persistence [9], however facing the challenges of implementation complexity, usability and adoption cost.



(a) A simplified scenario with 4 Hadoop jobs



(b) What is the optimal NameNode maintenance time? T_1, T_2 or T_3?

Fig. 2. An illustrative example showing why the straightforward approach is not optimal

In this work we assume the cluster maintenance duration (i.e., the amount of time when the cluster is down for maintenance) is fixed (e.g., 1 hour), which is typically true for most software updates [5]. So the question that needs to be answered is *when* the maintenance should start. Maintenance time determination answers this question. The goal of determining optimal maintenance time is maximizing the throughput (i.e., the number of completed jobs) of the Hadoop cluster.

A straightforward but naive approach we can easily come up with is choosing the maintenance starting time with the minimum number of running jobs (or tasks). However, as we elaborate later, such an approach is not optimal.

3.3 Illustrative example

We use the following scenario to elaborate why the straightforward approach of determining cluster maintenance time based on number of running tasks or jobs is not optimal. For easy presentation, we denote such approach as Current-Load (CL) based approach. Consider the a scenario where totally four Hadoop jobs are scheduled at dif-

[5] The problem considered won't change even with non-fixed maintenance duration; but having this assumption simplifies the presentation.

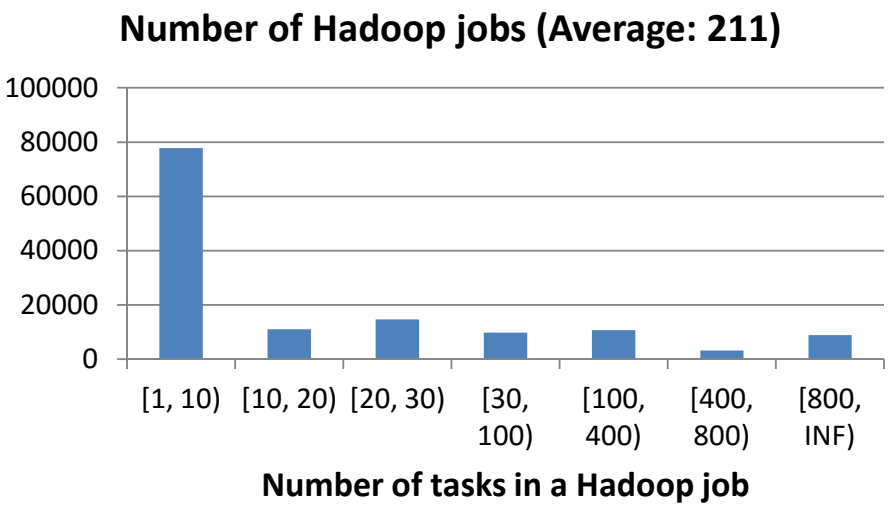


Fig. 3. Characteristics of profiled 135K Hadoop jobs (X-axis is the different buckets (number of tasks in a job); Y-axis is the number of Hadoop jobs in each bucket)

ferent time, as shown in Figure 2(a). For simplicity, we assume each job has only 1 mapper task and 1 reducer task.

In the figure we highlighted a few time points that are possible starting time for cluster maintenance. The Current-Load based approach would choose the time of T_2 to start NameNode maintenance, since there is only 1 running Hadoop job at T_2, while at T_1 and T_3 there are 2 running jobs. However, starting maintenance at T_2 would mean all the accumulated works of job J_2 will be lost, and the corresponding consumed Hadoop processing resources (i.e., CPU, networking, IO) would be wasted. As shown in Figure 2(b), the amount of wastage is non-negligible.

On the other hand, if the maintenance time starts at T_3, though the number of running jobs is 2, the aggregated amount of wasted work ($Cost_3 + Cost_4$) is much less than the wasted work ($Cost_2$) if maintenance starts at T_1. Since after maintenance all the broke Hadoop jobs will need to be re-submitted, the forfeited work (i.e., mapping and reducing) will in turn needs to be re-done, which is a waste of Hadoop processing resources and reduced Hadoop job completion rate. Having less wasted work essentially mean the Hadoop cluster can run more jobs in given amount of time, hence higher job throughput.

3.4 Characteristics of Hadoop jobs

We also obtained the characteristics of the Hadoop jobs running on one experimental Hadoop cluster. The job set includes totally 135, 808 Hadoop mapreduce jobs which are retrieved from the job history server. For each of the jobs, we measure the following metrics: (1) number of tasks (i.e., mappers and reducers) in a job; (2) total execution time; and (3) aggregated resource usage (i.e., cost).

Since Hadoop mapreduce jobs consist of both mappers and reducers, and they need to run in separate containers, the number of tasks for a Hadoop job is defined as the summation of the mappers and reducers. The execution time of a Hadoop jobs is defined

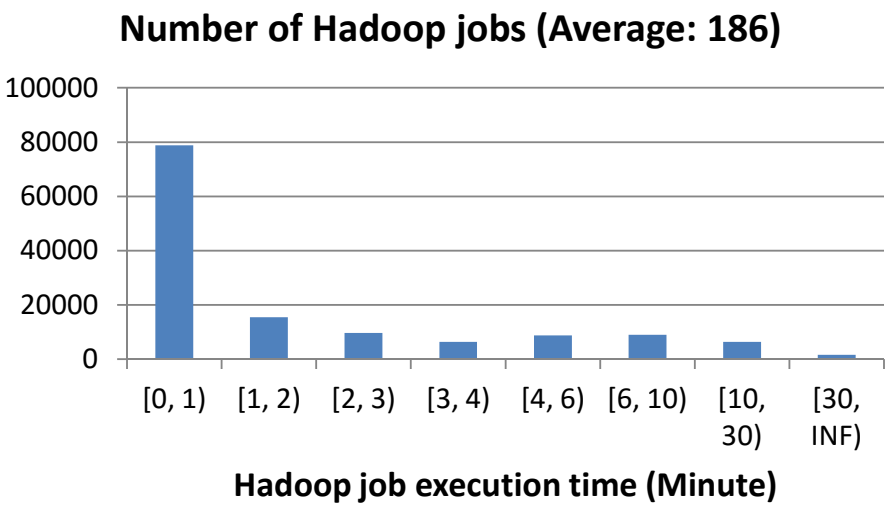


Fig. 4. Characteristics of profiled 135K Hadoop jobs (X-axis is the different buckets (Execution time in minute); Y-axis is the number of Hadoop jobs in each bucket)

as the summation of average *mapping time* and the average *reducing time* [6]. For the resource usage, we consider a custom metric of "container * (execution time)", which is the product of how many containers and how long on average a container (i.e., mapper and reducer) runs. Based on our experiences with running the experimental cluster, the top performance bottleneck of the Hadoop cluster is the limited number of concurrent running containers, hence it makes a lot of sense to use the "container * time" as the cost metric.

Figures 3, 4 and 5 display the characterizing results. For each of the metrics we considered, we show the distributions of Hadoop jobs under different buckets. The average number of tasks a Hadoop job has is 211, the average execution time is 186 seconds, and the average cost of each Hadoop job is 37775 container*second, or about 10 container*hours. If cluster maintenance is performed during the runs of these jobs, on average, half of the cost (i.e., 5 container*hours) incurred by each job will be lost.

3.5 Summary

We have thus far described that a straightforward Current-Load based approach fails to consider the nature of Hadoop resource usage, and hence not optimal. Depending on scenarios, the forfeited Hadoop work due to cluster maintenance can be significant.

By considering the resource used by running Hadoop jobs, a new approach which is based on accumulated works can achieve better resource usage efficiency and higher job throughput thanks to the minimum wasted resource usage. We denote the approach of determining cluster maintenance time based on the amount of accumulated completion of tasks as Accumulated-Work (AW) based approach.

[6] The shuffling and reducing phases may overlap, so for simplification, we define the *reducing time* as the maximum of reducing time and shuffling time reported by job history server.

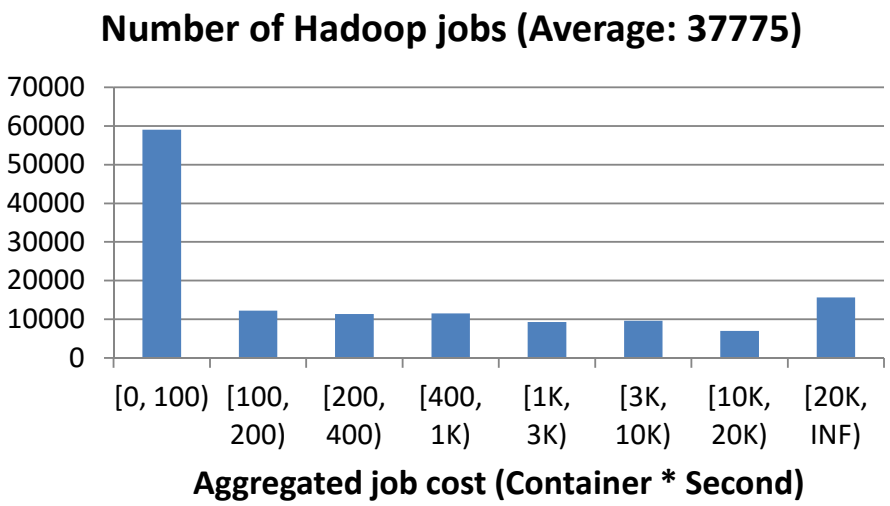


Fig. 5. Characteristics of profiled 135K Hadoop jobs (X-axis is the different buckets (Aggregated job cost of container*Second); Y-axis is the number of Hadoop jobs in each bucket)

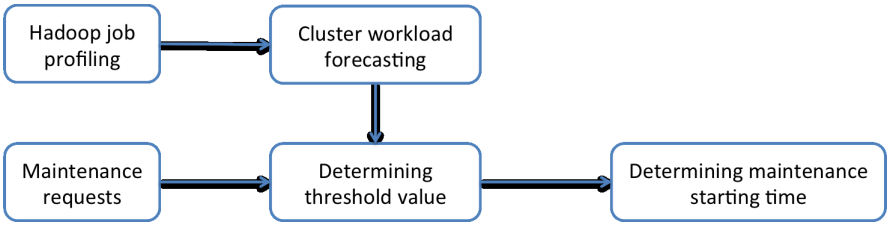


Fig. 6. Flow chart of AW-based maintenance determination algorithm

4 Solution

We now present the design and detailed algorithm of the proposed Accumulated-Work (AW) based solution.

4.1 Overview

Our proposal depends on forecasting of future workload. Given the irregularities of Hadoop jobs (i.e., starting/finishing time, number of tasks, run time), we devise a simple, but effective, predictor by assuming the distribution of those quantities will be similar over some periods (e.g., on a day-to-day basis or week-by-week basis). For simplicity, in this work we use the day-to-day model [7]. Then the way to determine appropriate maintenance time is by selecting the right threshold to trigger the maintenance.

The high level flow of the algorithm is illustrated in Figure 6. The algorithm consists of a Hadoop workload profiling component that analyzes the load of the cluster, the job

[7] Our particularly studied Hadoop cluster shows consistency of both day-to-day and week-to-week pattern.

Algorithm (a): Workload profiling and forecasting
Variables:
1 $T_{profile}$: Cycle of profiling the Hadoop workload

1 For every $T_{profile}$ of time:
2 Start a new cycle of workload profiling;
3 Obtain workload profiles (e.g., percentiles);

Fig. 7. Main algorithm (A)

properties, etc. Then the algorithm forecasts the future workload (e.g., next 1 day) using the forecasting component. Based on the forecasting result and urgency level of incoming cluster maintenance request, a threshold value is chosen. The threshold value will be used to determine the starting time of the next cluster maintenance. Whenever the monitored cluster workload (e.g., number of running jobs as in CL-based approach, amount of accumulated work as in AW-based approach) falls below the threshold, maintenance can start.

The maintenance request may come with different urgency level in the form of deadlines (e.g., within next 24 hours). Such urgency level is fed into the threshold determination component with a function of $Th = f(level)$. Many forms of the f function can be defined, and the higher urgency, the larger threshold value (so that the maintenance can be early kicked off). In this work, we choose a percentile-based approach which will be elaborated in Section 4.4.

4.2 Cost metric and objective function

The proposed AW-based solution chooses the cluster maintenance time based on the amount of accumulated works of the running Hadoop jobs. These works include both mapper and reducer tasks. Each of these tasks runs for certain amount of time. There are three types of tasks at a particular timestamp (i.e., time point): completed tasks (denoted by $Task_{completed}$), running tasks ($Task_{running}$) and tasks that are waiting to be scheduled ($Task_{waiting}$). The accumulated work is the aggregated works of all completed and running mapper and reducer tasks. For $Task_{completed}$, the accumulated work is determined by the running time of the particular task, that is, the difference between "finish time" and "start time". For $Task_{running}$, the considered run time is the difference between current timestamp and the job's "starting time".

The cost metric of AW-based solution is the aggregated accumulated work from all Hadoop jobs at any time point. Using AW_{T_i} to denote the cost at time T_i, and AW_{T_i,Job_j} to denote the accumulated work of job Job_j at T_i. So we have $AW_{T_i} = \sum AW_{T_i,Job_j}$, and AW_{T_i,Job_j} consists of the accumulated work of all $Task_{completed}$ and $Task_{running}$.

The objective function of AW-based approach is to choose a time point T_k that the minimum AW_{T_i} is achieved. By comparison, the objective function of the baseline CL-based approach is to achieve minimum number of concurrently running jobs.

Algorithm (b): determining maintenance threshold
Variables:
1 AW_{thre}: The threshold of the accumulated work for deciding maintenance time
2 $Level_{urg}$: The urgency level of maintenance

1 For every maintenance request issued;
2 Obtain forecasted workloads based on recent workload profiles;
3 Determine the maintenance urgency level $Level_{urg}$;
4 Determine AW_{thre} based on forecasted workloads and $Level_{urg}$;

Fig. 8. Main algorithm (B)

Algorithm (c): determining maintenance time
Variables:
1 T_{jh}: Cycle of fetching from Hadoop Job History server
2 AW_{aggr}: Currently aggregated computing resources used by all running jobs
3 AW_{thre}: The threshold of the accumulated work for deciding maintenance time
4 J_i: A Hadoop job started but unfinished
5 T_{mr} A finished or running Hadoop mapper/reducer task

1 Every T_{jh} of time:
2 Obtain the snapshot of the job history information;
3 For every running job J_i:
4 Get all finished or running tasks T_{mr};
5 Aggregate consumed resource to AW_{aggr};
6 If $AW_{aggr} > AW_{thre}$ or deadline expires:
7 Maintenance starts;

Fig. 9. Main algorithm (C)

4.3 Workload profiling and forecasting

In order for both CL-based and AW-based approaches to work, it is important to profile and forecast the workload of the Hadoop cluster. Profiling can be done by periodically querying the state of the Hadoop cluster, such as retrieving information from Job History server. The profiling results can come in different forms such as probability density functions; in this work, we consider a simplified form of percentiles for easier presentation, as shown in Figure 7.

For workload forecasting, it can be achieved by applying various forecasting models such as time series based ones like ARIMA [10]. Time series forecasting model breaks the data values into three parts: seasonal, trending and noises. Depending on specific scenarios, the workload may have different strengths on different parts. For instance, for a Hadoop cluster that mostly serves regularly scheduled jobs such as weekly aggregation of data or daily updates based on streaming data, the seasonal part will be very strong. On the other hand, if a Hadoop cluster mainly run ad-hoc experimental jobs, the noise part will dominate.

4.4 Determining threshold for cluster maintenance

Once the forecasting model is established, then the Hadoop administrators can predict the workload in the future. When a new cluster maintenance is needed within certain time period (e.g., 1 day), the administrators can choose a threshold value for cluster maintenance based on the forecasted workloads.

Based on our experiences with multiple Hadoop clusters, non-trivial clusters [8] typically exhibit irregular workloads and hence it is very difficult to characterize with reasonable forecasting models. For instance, on the particular cluster we used for this study, the running Hadoop jobs come from a mix of regularly scheduled ones and ad-hoc ones. Moreover, the users and jobs of the cluster are undergoing a major shift. The trending part, however, is not much.

To accommodate the generic cases where Hadoop clusters run irregular workloads, we choose a percentile-based method to determine the threshold. Specifically, we profile the previous days' workload, and calculate the percentile values of the past periods. Then we choose a particular percentile P (e.g., 5%) of the workload and base the maintenance time determination on the corresponding workload value.

Note the percentile value is determined by the maintenance urgency. For more urgent maintenance, a larger percentile value is chosen in hope of kicking off the maintenance early. On the other hand, if the maintenance is not urgent, smaller percentile value is desired. One approach is to rate the maintenance urgency on a scale of 1 to 100, where 1 indicates the most urgent level, and then choosing the corresponding percentile value as the threshold, as shown in Figure 8. With such a scaling of 1 to 100, the threshold value can be easily obtained by treating the emergency level as the percentile values, that is, $AW_{thre} = P_{Level_{urg}}$. For instance, for a low-emergency maintenance request rated at level-1, a P_1 value will be chosen, where P_1 means a value where 1% of all values are below it.

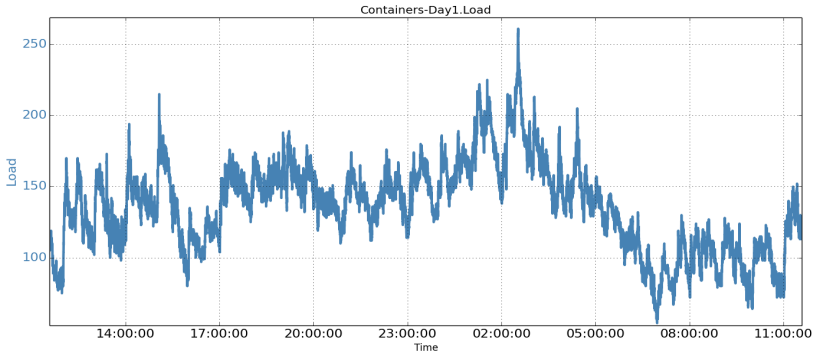
For CL-based approach, the forecasted workload indicates how many jobs are concurrently running at any time in the forecasting period. The threshold value is based on the profiled workload and maintenance urgency. The time stamps in the future that have forecasted workload intensity falls below the threshold are the possible cluster maintenance starting time.

For AW-based approach, the forecasted workload deducts the amount of accumulated workload at any future time. Once the corresponding threshold value is chosen, a process similar to that of CL-based approach, maintenance time can be similarly determined.

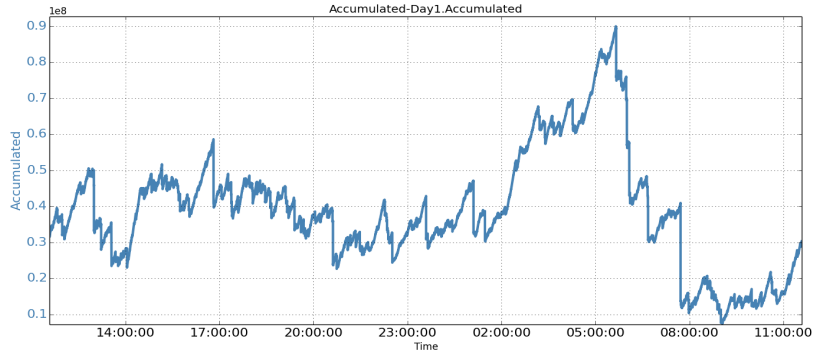
4.5 Determining cluster maintenance time

Once the cluster maintenance threshold is determined, the maintenance time can be determined by comparing to the real-time workloads to the threshold value. This step performs periodical querying of the Hadoop Job History server, as elaborated in Figure 9. Every time of T_{jh}, it obtains the snapshot of the job history information and extracts

[8] Those clusters that have sufficiently large of number of data nodes and run heterogenous workload



(a) Concurrent running jobs



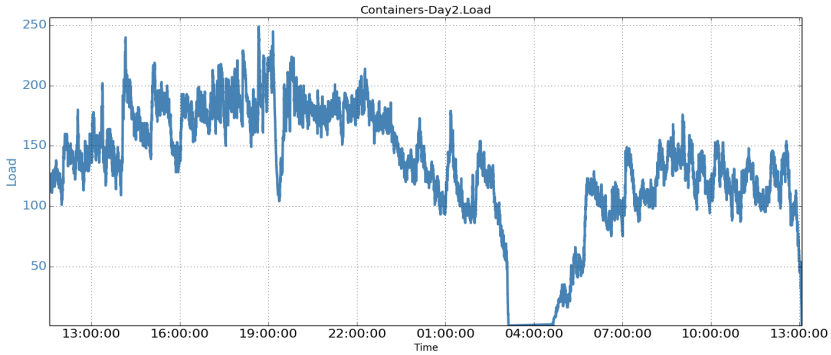
(b) Accumulated work

Fig. 10. Statistics of the first day

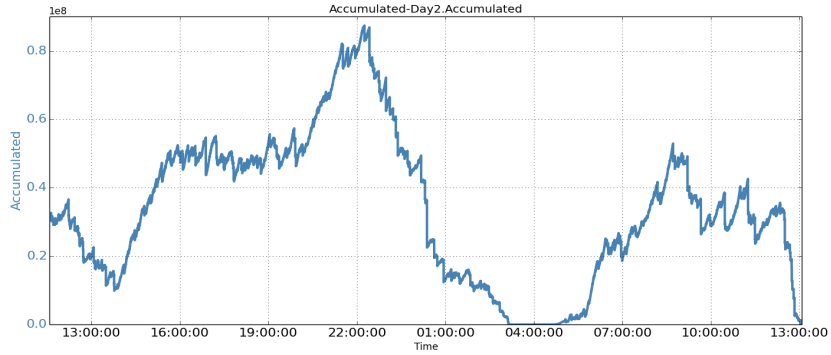
all jobs. For every running job J_i, it gathers all finished or running tasks T_{mr}. Then it iterate all the tasks and aggregates consumed computing resource to AW_{aggr}. Finally, it compares the aggregated AW_{aggr} to the threshold value AW_{thre} determined in Figure 8. If AW_{aggr} is less than AW_{thre}, the maintenance can start. Meanwhile, if the maintenance deadline (e.g., 1 day) has expired, the maintenance can start.

5 Evaluation

In this section, we will use the actual traces from our Hadoop cluster to illustrate how to to apply the proposed AW-based approach that is based on accumulated work on running jobs. For comparison, we also consider the baseline of CL-based approach, which is based on the number of running jobs.



(a) Concurrent running jobs



(b) Accumulated work

Fig. 11. Statistics of the second day

5.1 Methodology

We use the historical job information kept on Hadoop Job History server. For each Hadoop job completed, the Job History server maintains the meta data of the job and its mapper/reducer tasks. The meta data includes submission time, starting and finishing time, user account, number of mappers and reducers, etc.

The considered Hadoop cluster is able to run 20K containers concurrently, and the average job run time is only about 10 hours, hence there are up 48K mapper/reducer tasks are completed in a single day. The Job History server, however, is only able to keep most recent 20K jobs for our setup.

We have continuously collected 3-week of job history. We have to query the job history server multiple times due to configuration limits. Therere two limits in history server for how many jobs are kept, both are configurable. The first is the log retention period, which determines how long to keep the job logs on HDFS. This is by default set to 1 week. The second is the 20K limits, which is the maximum number of jobs that

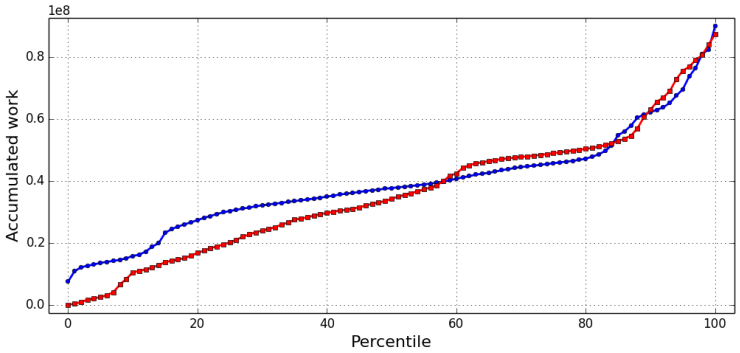


Fig. 12. Comparing the CDF (Cumulative Distribution Function) of two days

history server will load into memory and serve from the web page. For our busy cluster, 20K jobs only correspond to less than one-day of job history.

Even though we retrieve from Job History server frequently, due to the large number of jobs and frequent cluster maintenances, we believe some jobs are still missed in certain time periods. So we cleaned our data by eliminating some dirty periods. For easier presentation, we use a 2-day period with clean data to elaborate the evaluation results. The first day is used as the profiling period. [9] The second day is the period to evaluate the approaches.

5.2 Profiling statistics of first day

For the first day, we obtain the meta data of each job, hence we know the running time period (i.e., the start and finish time) and the number of mappers and reducers running during this time period. Then we can deduct the number of running jobs and the accumulated works of any time during the 24 hours. We plot the percentile values of concurrent jobs and accumulated works in Figure 10(a) and (b), respectively.

For the Hadoop cluster maintenance, we consider three scenarios based on the maintenance urgency: low-urgency, medium-urgency and high-urgency. For low-urgency maintenance request, we choose a threshold of 1%, essentially means about 1% of the entire time, the maintenance is able to kick off. If the checking interval is every minute, then on average, the expected starting maintenance time is 100 minutes (i.e., $\frac{1}{1\%}$). Similarly, for medium-urgency maintenance request, we consider two possible thresholds of 2% and 5%, which respectively have maintenance waiting time of 50 minutes and 20 minutes. For high-urgency maintenance request, we choose two possible thresholds of 10% and 20%, with expected maintenance waiting time of 10 and 5 minutes, respectively.

For both CL-based and AW-based approachs, the possible time points of maintenance (i.e., the time when the respective metrics fall below the corresponding particular percentile values.) are determined. The opportunity *cost* of each maintenance time

[9] In production, the profiling efforts are running continuously.

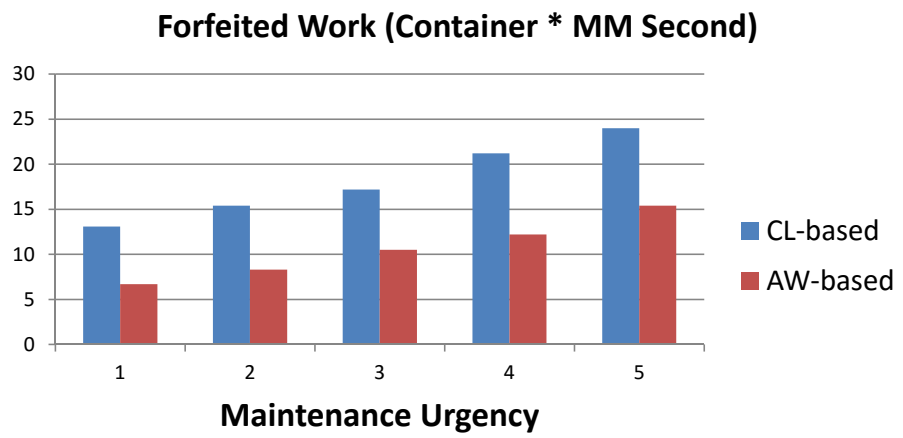


Fig. 13. Comparing the amount of forfeited work by CL-based and AW-based approach under different maintenance urgency levels

point is obtained based on the amount of forfeited work. The cost unit of the accumulated works is "container*second", intuitively indicting the resource used by some containers concurrently running for some time.

5.3 Cost results of second day

For the second day, we obtain the number of concurrent Hadoop jobs and the accumulated Hadoop work for each timestamp, shown in Figure 11. These values will be used to determine possible cluster maintenance kickoff time based on threshold values for both CL-based and AW-based approaches.

The assumption of our design is the relatively stable distribution of accumulated works across days. To understand how well the assumption holds, we plot the CDFs (Cumulative Distribution Function) of these two days in Figure 12. From the figure, we see very similar CDF curves.

We also compare the cost of both CL-based and AW-based approaches. For each approach, the threshold values corresponding to different urgency levels of cluster maintenance requests are listed in Table 1. Based on the threshold values, the timestamps of all possible maintenance kicking off are recorded. Then the opportunity costs (i.e., the amount of forfeited Hadoop works) are obtained for each timestamp. Finally the average values of all the possible opportunity costs are calculated for both approaches.

Table 1. Threshold values for kicking off cluster maintenance

Maintenance urgency	Percentile	CL-based	AW-based
Low	1	7493	8745211
$Medium_1$	2	7924	10.7M
$Medium_2$	5	8884	12.9M
$High_1$	10	9913	14.7M
$High_2$	20	11652	24.3M

The results of all 5 maintenance urgency levels are displayed in Figure 13. As shown in the figure, AW-based approach consistently results in much lower opportunity compared to CL-based approach. For some urgency levels (e.g., *Low* and $Medium_1$), the AW-based cost is only **half** of that of CL-based. For other urgency levels, the saving is about 40%.

Specifically, at urgency level of $High_1$, for CL-based approach, the average opportunity cost is about 21,157,000 container*second, or about 5877 container*hour. The average cost of AW-based approach is about 12,240,000 container*second, or about 3,400 container*hours. Compared to CL-based approach of 5, 877 container*hours, the difference is 2,477 container*hours for every cluster maintenance is done. In other words, AW-based approach can save 42% of the wasted resources as resulted from CL-based approach.

6 Discussions

In this section, we discuss several issues/clarifications related to the considered problem and proposed approach.

6.1 HDFS federation

HDFS federation is aimed at solving the single-NameNode inefficiency by splitting the entire HDFS namespace to multiple ones. Since different NameNodes correspond to different namespaces, NameNode maintenance will still lose unfinished jobs which access the particularly maintained namespace. In addition, even with HDFS federation, people usually perform maintenance on all namespace at once.

6.2 HA (High availability) setup

HA is an advanced setup for Hadoop NameNode that can alleviate the single-node failure problem. When configured with HA, two NameNodes work together to allow seamless NameNode maintenance. One of the NameNode will be active at any time, while the other be passive. When the active NameNode is down for any reason, the passive NameNode will take over the responsibility. During NameNode-caused cluster maintenance, NameNode will not lose unfinished jobs. However, the tasks might fail during rolling upgrade, hence will need to be rescheduled. Moreover, for cluster maintenances caused by other reasons (e.g., Resource Manager), Hadoop jobs are still lost.

6.3 Resource Manager maintenance

Though most of the cluster maintenances are caused by NameNode updates, Resource Manager (RM) can also trigger cluster maintenance. RM node maintenance also loses jobs. Therere features in YARN [11] that adds HA (High Availability) to RM. For now, this HA feature only allows automatically resubmission of previously unfinished jobs. New features are being added which preserve workloads.

6.4 Maintenance announcement

The scenario we consider in this work is to automatically decide the cluster maintenance time and execute the maintenance without the Hadoop users' awareness and actions. A slightly different approach is to keep users informed beforehand by announcing the forthcoming maintenance time so that users do not submit jobs that will not complete before the maintenance starts. Such an approach does not really help maximizing the key performance metric of Hadoop job throughput, since the cluster resources are anyway wasted as users stop submitting jobs. In our experiences, we have seen such under-utilization of the cluster before announced maintenance window.

Moreover, though users may be aware of the maintenance window from the announcement and stop submitting ad hoc jobs (e.g., the one-time running jobs), most non-ad-hoc jobs (e.g., hourly running jobs) can rarely take advantage of the announcement. Those jobs are scheduled to run periodically and are very inconvenient for the users to pause and resume.

In addition, the AW-based approach can be used in tandem with the announcement-based approach, as the algorithm and its associated observations can be used to determine the optimal maintenance time window to announce to users. In other words, if AW-based profiling has identified the pattern of accumulated workloads, then a maintenance window with smallest accumulated workloads can be chosen and be announced to users.

7 Related Work

7.1 Hadoop performance optimization

Various optimizations have been proposed to improve Hadoop throughput and response time [12–15]. In particular, [13] proposes to improve on the job execution mechanism. [14] studies the impact of adopting SSD for storage. [15] proposes a new MapReduce Scheduler for special environments. Our work is orthogonal with these works.

7.2 Workload forecasting

To forecast various types of computing workload (e.g., networking traffic, incoming traffic), many models such as [10, 16] have been proposed. Work [17] proposes time series based model to forecast when to add more network bandwidth. In another recent work [18], a time series based forecasting model is used to predict LinkedIn's Espresso [19]/Databus [20] traffic. Work [21] proposes a real-time rolling grey forecasting method to achieve increase in forecast accuracy. Work [22] uses a novel self-adaptive approach that selects suitable forecasting methods for a given context, and the user only has to provide a general forecasting objectives.

7.3 Determining maintenance time

We searched thoroughly for related works in the areas of determining optimal cluster maintenance time. Though there are some works [23, 24] dealing with maintenance

scheduling for different systems, to our best knowledge, we haven't seen any similar work that attempts to optimize the maintenance time for Hadoop clusters. Moreover, due to the unique characteristics of Hadoop mapreduce jobs during cluster maintenance (i.e., combinations of mappers and reducers, lost intermediate job states during cluster-wide maintenance), the impact of Hadoop cluster maintenance considerably differs from other types of maintenance. For this reason, we believe our study exhibits uniqueness with regard to these aspects.

8 Conclusion

This work presents an optimization technique to maximize overall throughput of a "stateless system" such as Hadoop cluster system. We propose to use *accumulated job progress* as the cluster maintenance criteria as opposed *number of running jobs*. Such a design can significantly save the forfeited computing resources caused by cluster maintenance.

References

1. "Apache hadoop," https://hadoop.apache.org/.
2. "Hdfs architecture guide," https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
3. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03, Bolton Landing, NY, USA, 2003, pp. 29–43.
4. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008.
5. J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, San Francisco, CA, 2004, pp. 10–10.
6. "Apache hbase," http://hbase.apache.org/.
7. V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13, 2013.
8. "Apache spark," http://spark.apache.org/.
9. "Provide ability to persist running jobs," https://issues.apache.org/jira/browse/HADOOP-3245.
10. G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1994.
11. V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13, Santa Clara, California, 2013.
12. S. B. Joshi, "Apache hadoop performance-tuning methodologies and best practices," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12, Boston, Massachusetts, USA, 2012.

13. R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, and Y. Huang, "Shadoop: Improving mapreduce performance by optimizing job execution mechanism in hadoop clusters," *J. Parallel Distrib. Comput.*, vol. 74, no. 3, pp. 2166–2179, Mar. 2014.
14. D. Wu, W. Luo, W. Xie, X. Ji, J. He, and D. Wu, "Understanding the impacts of solid-state storage on the hadoop performance," in *Proceedings of the 2013 International Conference on Advanced Cloud and Big Data*, ser. CBD '13, Washington, DC, USA, 2013.
15. B. Sharma, T. Wood, and C. R. Das, "Hybridmr: A hierarchical mapreduce scheduler for hybrid data centers," in *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, ser. ICDCS '13, Washington, DC, USA, 2013, pp. 102–111.
16. J. Durbin and S. J. Koopman, *Time series analysis by state space methods*. Oxford Univ. Press, 2001.
17. K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot, "Long-term forecasting of internet backbone traffic," *Trans. Neur. Netw.*, vol. 16, no. 5, pp. 1110–1124, Sep. 2005.
18. Z. Zhuang, H. Ramachandra, C. Tran, S. Subramaniam, C. Botev, C. Xiong, and B. Sridharan, "Capacity planning and headroom analysis for taming database replication latency: Experiences with linkedin internet traffic," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15, Austin, Texas, USA, 2015, pp. 39–50.
19. L. Qiao, K. Surlaker, S. Das, and et.al., "On brewing fresh espresso: Linkedin's distributed data serving platform," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13, 2013, pp. 1135–1146.
20. S. Das, C. Botev, and et al., "All aboard the databus!: Linkedin's scalable consistent change data capture platform," ser. SoCC '12, New York, NY, USA, 2012.
21. T. Xia, X. Jin, L. Xi, Y. Zhang, and J. Ni, "Operating load based real-time rolling grey forecasting for machine health prognosis in dynamic maintenance schedule," *J. Intell. Manuf.*, vol. 26, no. 2, pp. 269–280, Apr. 2015.
22. N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive workload classification and forecasting for proactive resource provisioning," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479899
23. K. S. Moghaddam and J. S. Usher, "Preventive maintenance and replacement scheduling for repairable and maintainable systems using dynamic programming," *Comput. Ind. Eng.*, vol. 60, no. 4, pp. 654–665, May 2011.
24. M. Carr and C. Wagner, "A study of reasoning processes in software maintenance management," *Inf. Technol. and Management*, vol. 3, no. 1-2, pp. 181–203, Jan. 2002.