2015 IEEE 29th International Parallel and Distributed Processing Symposium

Resource and Deadline-aware Job Scheduling in Dynamic Hadoop Clusters

Dazhao Cheng*, Jia Rao*, Changjun Jiang† and Xiaobo Zhou*

*Department of Computer Science, University of Colorado, Colorado Springs, USA
†Department of Computer Science & Technology, Tongji University, Shanghai, China
Emails: {dcheng,jrao, xzhou}@uccs.edu, cjjiang@tongji.edu.cn

***Abstract*—As Hadoop is becoming increasingly popular in large-scale data analysis, there is a growing need for providing predictable services to users who have strict requirements on job completion times. While earliest deadline first scheduling (EDF) like algorithms are popular in guaranteeing job deadlines in real-time systems, they are not effective in a *dynamic* Hadoop environment, i.e., a Hadoop cluster with dynamically available resources. As there is a growing number of Hadoop clusters deployed on hybrid systems, e.g., infrastructure powered by mix of traditional and renewable energy, and cloud platforms hosting heterogeneous workloads, variable resource availability becomes common when running Hadoop jobs. In this paper, we propose, *RDS*, a Resource and Deadline-aware Hadoop job Scheduler that takes future resource availability into consideration when minimizing job deadline misses. We formulate the job scheduling problem as an online optimization problem and solve it using an efficient receding horizon control algorithm. To aid the control, we design a self-learning model to estimate job completion times and use a simple but effective model to predict future resource availability. We have implemented RDS in the open-source Hadoop implementation and performed evaluations with various benchmark workloads. Experimental results show that RDS substantially reduces the penalty of deadline misses by at least 36% and 10% compared with Fair Scheduler and EDF scheduler, respectively.**

I. Introduction

As industries are confronting an unprecedented volume of data everyday, Hadoop, the open source implementation of the MapReduce programming model, has become the *de facto* standard technique for storing and analyzing petascale data in a cost-efficient way. For example, the Data warehouse Hadoop cluster at Facebook contains 3000 machines and hosts on average 25000 MapReduce jobs per day [4]. However, study [11] has shown that current use of Hadoop in research and enterprises still has significant room for improvement on the performance of Hadoop jobs and the utilization of Hadoop clusters. There is a growing need for providing predictable services to Hadoop users who have strict requirements on job completion times (i.e., deadlines).

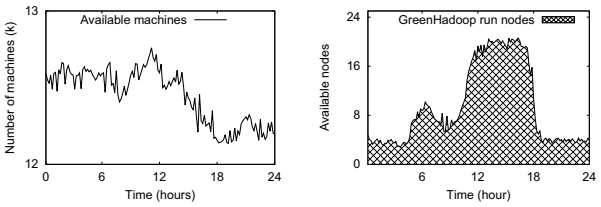
However, meeting job deadlines is difficult in current Hadoop platforms. First, because jobs have diverse resource demands, it is hard to determine how much resource is needed for each job to avoid its deadline miss. Second, Hadoop clusters are usually shared by multiple jobs and the scheduling order of these jobs can affect job completion time [24]. Thus, allocating sufficient resources alone may not guarantee job completion time effectively. While existing schedulers in Hadoop, such as the default FIFO scheduler, Fair scheduler, Capacity scheduler, the RAS scheduler [20], and their variations [9], [23], optimize job completion time without considering deadlines, there are recent studies that aim to guarantee job deadlines in Hadoop workloads by estimating job completion time and manipulating job queue ordering [24] or task scheduling [8].

A recent trend of running Hadoop in a hybrid environment further complicates the problem. To pursue cost-efficiency, Hadoop clusters can be powered by a mix of renewable energy and traditional power grid [6], [9], [17], or share the same cloud infrastructure with interactive workloads [17], [23], or run opportunistically on transient resources, e.g., Amazon Spot Instances. In these scenarios, the resources available to the Hadoop cluster are quite dynamic due to the variable supply of renewable energy, the changing intensity of co-located workloads, or the abrupt termination of market-based resources. The dynamics in the capacity of Hadoop clusters pose significant challenges on satisfying job deadlines. First, it is hard to estimate job completion time with dynamically available resources. The prediction models should be robust to the varying cluster capacity. Second, job execution and task scheduling become more complicated. When the amount of available resources drops, high priority jobs or jobs with approaching deadlines should be prioritized to improve the application performance or revenue.

In this work, we find that deadline misses in Hadoop workloads can possibly be minimized by exploiting the dynamics in resource availability and the flexibility in Hadoop task scheduling. To this end, we propose, *RDS*, a Resource and Deadline-aware Hadoop job Scheduler that dynamically allocates resources to different jobs based on the prediction of resource availability and job completion times. RDS temporarily delays low priority jobs or jobs with distant deadlines in hopes that there will be sufficient resources in the future to compensate the slowdown caused to these jobs. More specifically, we make the following technical contributions.

- We develop a self-learning fuzzy model for estimating a job's completion time. The fuzzy model performs fine-grained job completion time estimation and self-adaptation at every measurement interval. We use a simple but effective model to predict future resource availability based on recent history.
- We formulate Hadoop job scheduling as an optimization

1530-2075/15 $31.00 © 2015 IEEE
DOI 10.1109/IPDPS.2015.36



(a) Google trace. (b) GreenHadoop trace.

Fig. 1. Dynamic production and Hadoop clusters.

problem based on the prediction models of the job completion time and the resource availability. We design an efficient receding horizon control (RHC) algorithm to derive the online solution. Its solution is a task resource allocation matrix that minimizes job deadline misses considering future resource availability.

- We develop and implement the *RDS* scheduler in the open-source Hadoop implementation and perform comprehensive evaluations with various Hadoop workloads. Experimental results show that RDS effectively reduces job deadline misses by at least 36% and 10% compared to Fair scheduler and the earliest deadline first (EDF) scheduler, respectively.

The rest of this paper is organized as follows. Section II gives motivations on resource and deadline aware Hadoop scheduling. Section III describes the design of RDS. Section IV gives details on system implementation. Section V presents the experimental results. Section VI reviews related work. Section VII concludes the paper.

II. Motivation

We first introduce two dynamic clusters in a production datacenter and a research institution respectively. We then show that Hadoop job scheduling should be deadline aware in order to provide predictable services to users. We use concrete examples to demonstrate that existing Hadoop job schedulers are ineffective in meeting deadlines. Finally, we discuss the practical issues when applying EDF, the theoretical optimal scheduler, in real and dynamic Hadoop systems.

A. Cluster trace analysis

To understand the resource dynamics in production cloud datacenters, we have conducted an analysis of the time-varying capacity trace from a production cluster at Google [22]. Figure 1(a) demonstrates the number of machines available in the cluster can fluctuate significantly over time. This is due to the application of power-aware resource provisioning approach based on machine turn-on and turn-off for energy saving. We aim to provide a solution to this dynamic capacity environment by finding the optimal task scheduling approach to improve application performance while considering the cost of scheduling reconfigurations.

As the environmental impact of datacenters rapidly grows, the industry has started to explore building green datacenters that are powered by renewable energy. For example, HP Lab built up the Net-Zero Energy datacenter recently [17].

TABLE I
Job 1 and 2 submission information.

Jobs	Input	CPU demand	Arrival time	Deadline
J1	18 GB	120 GHz	0_{th} min	40_{th} min
J2	9 GB	60 GHz	10_{th} min	30_{th} min

Unlike traditional energy, the intermittency of renewable energy makes it very hard to maintain a stable cluster resource availability to process workloads. Goiri *et al.* proposed GreenHadoop [9], a MapReduce framework for Parasol, a prototype green cluster built in Rutgers University. Figure 1(b) shows that GreenHadoop has dynamic resource availability during 24 hours since it is powered by solar energy and uses electrical grid as a backup. It demonstrates that the available resource of Hadoop cluster could be highly dynamic due to the time-varying power supply.

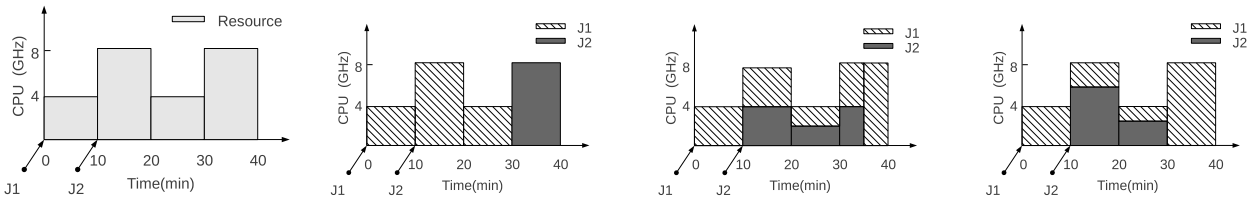
The above analysis suggests that the benefit of dynamic capacity provisioning is apparent for production datacenters from the perspective of both economics and environments. However, it also brings up a challenging task to manage dynamic datacenter clusters. In order to further explore this problem, we conduct a case study as follows.

B. Case study

We created a 5-node virtual Hadoop cluster in our university cloud testbed and ran two jobs using different schedulers. The cluster was configured with one master and four slave nodes. All the slave nodes shared a pool of CPU resources. We dynamically changed the resources allocated to the cluster using VMware's distributed resource scheduler to emulate the dynamics in resource availability of the cluster. The total resource was evenly distributed to each slave node. The master node was allocated a fixed capacity. We ran two different *word-count* [1] jobs. Two jobs have different input sizes, resource demands and deadlines. Table I gives their information.

Figure 2 shows the performance of three schedulers in the dynamic cluster. Note that the CPU demand in Table I is the cumulative resource requirement for running individual jobs. The GHz in Figures 2 and 3 is the instantaneous CPU allocation of jobs. For example, the size of the region with slanting lines (i.e., J1's allocation) in Figure 2(b) should equal the demand of J1 in Table I, which is 120 GHz. In the 10_{th} to 20_{th} and 30_{th} to 40_{th} intervals, the cluster has doubled resources than in other intervals.

Figure 2(a) shows the trace of dynamic resource availability. Figures 2(b), 2(c) and 2(d) show the job execution under three schedulers, namely First In First Out (FIFO), Fair scheduler, and an ideal scheduler considering job deadlines. FIFO schedules jobs based on their arrival times. Job J2 was delayed until job J1 finished, leading to the miss of J2's deadline. Fair scheduler allocates an equal amount of resource to each job. However, fairness in resource allocation does not guarantee that J2 met its deadline. The ideal scheduler knew the future resource availability and determined that fair allocations between J1 and J2 would lead to J2's deadline miss because the resource in the 20_{th} to 30_{th} is not sufficient.



(a) Resource availability. (b) FIFO Scheduler. (c) Fair Scheduler. (d) Ideal Scheduler.

Fig. 2. Performance of FIFO Scheduler, Fair Scheduler and an ideal scheduler in a dynamic Hadoop cluster.

Knowing that more resources would be available in the 30_{th} to 40_{th} interval, the ideal scheduler prioritized J2 with more resources from 10_{th} to 30_{th}. The flexible resource allocation effectively guaranteed the deadlines of both jobs.

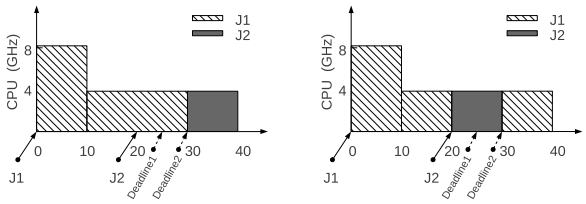
We make two observations. First, deadline-oblivious schedulers perform poorly for jobs with deadlines in a dynamic cluster. Second, knowledge on future resource availability is critical to avoiding deadline misses. For example, if the resource level in the 20_{th} to 30_{th} interval further drops, a deadline-aware scheduler should allocate more resources to J2 in the 10_{th} to 20_{th} interval. In summary, the information on future resource availability affects a deadline-aware scheduler's decisions on individual job resource allocations.

For this job setting, there exists a schedule that meets both job's deadline. For such a schedulable job set, earliest deadline first scheduling (EDF) can also meet the deadlines. EDF is optimal on preemptive uniprocessors [19]. However, EDF has practical issues when used as a Hadoop job scheduler. First, in a resource constrained scenario or an overloaded system, where not all job deadlines can be met, the performance of EDF is unpredictable and often quite bad. Figure 3(a) shows the scheduling order of two jobs under EDF. The system become overloaded as the available resources dropped at the 10_{th} minute. For this system, EDF is clearly not optimal because J2 would have meet its deadline if it was scheduled before J1 (as shown in Figure 3(b)). Second, EDF does not determine how much resource is needed to meet job deadlines [24]. It may lead to over-provisioning or under-provisioning of resources. Finally, EDF overwrites user-defined job priorities making it less attractive in a multi-user environment.

[**Summary**] We have shown that deadline-aware schedulers have significant advantage over deadline-oblivious ones. However, obtaining an optimal or near-optimal job scheduler is not trivial in clusters with dynamically variable resources. We have shown that resource-oblivious schedulers such as EDF only performs scheduling optimizations based on current observation of resource availability. Looking forward into the future resource availability can guide schedulers in making wise resource allocation decisions. These findings motivated us to develop a resource and deadline-aware scheduler for dynamic Hadoop clusters.

III. RDS Design

In this section, we present the design of *RDS*, a resource and deadline-aware Hadoop job scheduler. RDS determines the resource allocations to individual jobs based on the



(a) EDF-preemptive. (b) Ideal scheduler.

Fig. 3. Performance comparison in an overloaded system.

estimations of job completion time and the predictions on future resource availability. To realize fine-grained resource control, RDS divides job execution into control intervals and builds performance models for estimating job progress, from which overall job completion time can be inferred. Based on the job completion time estimation and resource availability prediction, RDS derives the optimal job scheduling via an online receding horizon control (RHC) algorithm.

Figure 4 shows the architecture of RDS. We describe the functionality of each component as follows:

- **Fuzzy performance model** takes allocated resources and job size as inputs and generates the estimated job completion time as outputs. The model is updated periodically based on the measured job progress at each control interval.
- **Resource predictor** takes the history information on resource availability and predicts the amount of available resources for the next few intervals.
- **Scheduling optimizer** adjusts the number of slots allocated to each running job based on an online receding horizon control algorithm.

We formulate the resource and deadline-aware scheduling as an optimization problem that minimizes job deadline miss penalty. We present detailed design of the self-adaptive fuzzy model, the resource model and the scheduling optimizer.

A. Problem Formulation

We consider a Hadoop cluster with dynamic resource availability r_a. Consider that there are J jobs running in the system and the control interval is t ($t \in [1, ..., T]$). Each job j has map tasks allocated u_j^m resource and reduce tasks allocated u_j^r resource. y_j is the actual completion time of job j and y_j^{ref} is the reference time that meets its deadline. The optimization

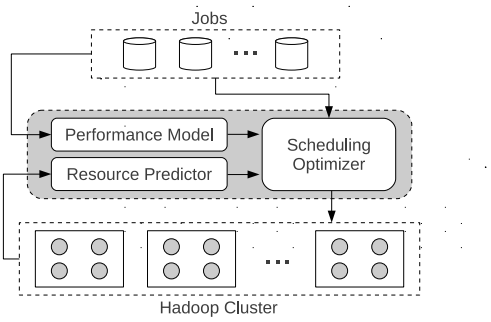


Fig. 4. The architecture of RDS.

problem is formulated as follows,

$$\min \sum_{t=1}^{T} \sum_{j=1}^{J} \omega_j (\frac{y_j - y_j^{ref}}{y_j^{ref}})^+, \tag{1}$$

$$\text{s.t. } \sum_{j=1}^{J} (u_j^m + u_j^r) \leq r_a. \tag{2}$$

The goal of RDS scheduler is to minimize the penalty of deadline misses. Objective Eq. (1) captures the lost revenue due to the deadline misses. The penalty remains zero until the job misses its deadline. ω_j is a constant that represents the priority of job j. Constraint Eq. (2) ensures that the sum of resources assigned to all running jobs must be bounded by the total available resources in the cluster.

However, a job's completion time y_j can only be measured when the job finishes. It is often too late for the Hadoop scheduler to intervene if a job already misses its deadline. To this end, we break down a job's execution into small intervals and apply calibrated deadlines for each interval. Consider a job's deadline is 100 minutes away and the execution is divided into 10 intervals. If the job can finish one tenth of the total work in each interval, it can meet the overall deadline. The Hadoop scheduler can adjust the resource allocation if a job's execution is considered slow based on its progress on individual intervals. Such a breakdown of job execution also allows the scheduler to look forward into future intervals and apply optimization considering future resource availability.

Specifically, we transform the optimization problem to a receding horizon control (RHC) problem that minimizes the following objective function:

$$\begin{aligned} J(t) = &\sum_{i=0}^{H_p} \sum_{j=1}^{J} \parallel y_j(t+i) - y_j^{ref}(t+i) \parallel_W^2 \\ &+ \sum_{i=0}^{H_c} \sum_{j=1}^{J} (\parallel \Delta u_j^m(t+i) \parallel_P^2 + \parallel \Delta u_j^r(t+i) \parallel_Q^2), \end{aligned} \tag{3}$$

where t is the measurement (or control) interval, $y_j(t)$ is the actual progress of the job in interval t and $y_j^{ref}(t)$ is the expected progress that ensures meeting the job's deadline. W is the job priority matrix. The optimization looks forward into future H_p intervals and tries to derive the optimal resource allocation considering future resource availability. $\Delta u_j^m(t)$ and $\Delta u_j^r(t)$ are the resource adjustment for map and reduce tasks in order to meet the expected job progress in each interval, respectively. They together represent the control penalty and are weighted by penalty matrices P and Q. The first part of the function serves as a penalty for not meeting a job's progress target and the second part requires the cost for changing job's resource allocations be minimized.

To consider future resource availability, the RHC controller predicts a job's performance for the next H_p control intervals. It then computes a sequence of control actions $\Delta u(t)$, $\Delta u(t+1)$, ..., $\Delta u(t+H_c)$ over H_c control periods, called the *control horizon*, to keep the predicted performance close to their expected targets. Thus, a performance model is needed to predict the progress of a job in a control interval given a certain amount of resources. The total amount of resources available in future intervals needs to be predicted.

B. Estimating Job Execution Progress

We use a multiple-input-single-output fuzzy model to predict a job's execution progress based on its input size and resource allocation in each control interval. The fuzzy model is often used to capture the complex relationship between resource allocations and a job's fine-grained execution progress [14]. However, a job's progress can be affected by many factors. First, job progress is not uniform at different execution phases, e.g., map and reduce phases. Second, even within the same phase, data skew among tasks leads to different task execution speed at different intervals. Finally, co-running jobs may unpredictably interfere with a job's execution, making the mapping of resource to job progress variable. Therefore, we design an online self-adaptive fuzzy model based on real-time measurements of job progress.

1) Fuzzy Model: The job j execution progress in the control interval t is represented as the input-output NARX type (Nonlinear Auto Regressive model with eXogenous inputs),

$$y_j(t) = R_j(u(t), d_j, \xi(t)). \tag{4}$$

R is the relationship between the input variables and the output variable. The input variables are the current resource allocation $u(t)$, the job input size d_j, and the regression vector $\xi(t)$. Here, resource allocation, $u(t) = [u^m(t), u^r(t)]$, includes both map resource allocation $u^m(t)$ and reduce resource allocation $u^r(t)$. The regression vector $\xi(t)$ contains a number of lagged outputs and inputs of the previous control periods. It is represented as

$$\begin{aligned} \xi(t) = [&(y(t-1), y(t-2), \cdots, y(t-n_y)), \\ &(u(t), u(t-1), \cdots, u(t-n_u))]^T \end{aligned} \tag{5}$$

where n_y and n_u are the number of lagged values for outputs and inputs, respectively. Let ρ denote the number of elements in the regression vector $\xi(t)$, that is,

$$\rho = n_y + n_u. \tag{6}$$

R is the rule-based fuzzy model that consists of Takagi-Sugeno rules [5]. A rule R_j is represented as

$$R_j: \text{IF } \xi_1(t) \text{ is } \Omega_{j,1}, \xi_2(t) \text{ is } \Omega_{j,2}, \cdots, \text{ and } \xi_\rho(t) \text{ is } \Omega_{j,\rho}$$
$$u(t) \text{ is } \Omega_{j,\rho+1} \text{ and } d_j \text{ is } \Omega_{j,\rho+2}$$
$$\text{THEN } y_j(t) = \zeta_j \xi(t) + \eta_j u(t) + \omega_j d_j + \theta_j. \quad (7)$$

Here, Ω_j is the antecedent fuzzy set of the jth rule, which is composed of a series of subsets: $\Omega_{j,1}, \Omega_{j,2}, \cdots, \Omega_{j,\rho+2}$. ζ_j, η_j and ω_j are parameters, and θ_j is the offset. Their values are obtained by offline training. Each fuzzy rule characterizes the nonlinear relationship between allocated resources and performance for a specific job type.

2) Online Self-Learning: Due to the dynamics of MapReduce job behaviors (e.g., data skews, different phases and multi-tenant interferences), we design an online self-learning module to adapt the fuzzy model. It aims to minimize the prediction error of the fuzzy model $e(t)$, which is the error between actual measured job progress and predicted value.

If $e(t) \neq 0$, we apply a recursive least squares (RLS) method [2] to adapt the parameters of the current fuzzy rule. The technique updates the model parameters as new measurements are sampled from the runtime system. It applies exponentially decaying weights on the sampled data so that higher weights are assigned to more recent observations.

We express the fuzzy model output in Eq.(4) as follow:

$$y(t) = \phi(t)X + e(t) \quad (8)$$

where $e(t)$ is the error between the actual output and predicted output. $\phi(t) = [\phi_1^T, \phi_2^T, .., \phi_\rho^T]$ is a vector composed of the model parameters. $X = [\sigma_1 X(t), \sigma_2 X(t), .., \sigma_\rho X(t)]$ where σ_j is the normalized degree of fulfillment or firing strength of j_{th} rule and $X(t) = [\xi(t)^T, u(t)]$ is a vector containing the previous outputs and inputs of the control system. The parameter vector $\phi(t)$ is estimated so that the error function in Eq.(9) is minimized. We apply both the current error $e(t)$ and the previous error $e(t-1)$ to estimate the parameter vector,

$$Error = \sum_{t-1}^{t} (e(t)^2 + \tau e(t-1)^2). \quad (9)$$

Where τ is called the discount factor as it gives higher weights on more recent samples in the optimization. It determines in what manner the current prediction error and old errors affect the update of parameter estimation.

C. Predicting Resource Availability

We use a simple but effective Auto-Regressive Integrated Moving Average (ARIMA) model [3] to predict the resource availability in future intervals based on history information. The ARIMA model has been used to predict resource consumption [26], and dynamic power supply [4].

In ARIMA model, the available resource of the current interval $r_a(t)$ is predicted based on the last n observations of resource availability, i.e., $r_a(t-1)$, ... $r_a(t-n+1)$.

$$r_a(t) = a_1 r_a(t-1) + a_2 r_a(t-2) + ... + a_n r_a(t-n), \quad (10)$$

where $a_1, a_2, \ldots, a_n$ are coefficients obtained via model fitting. RDS predicts future resource availability over a time window $h \in \mathbb{N}^+$. Let $r_a(t+h|t)$ denote the h_{th} step prediction of $r_a(t)$ knowing the last n observations, i.e., $r_a(t-n), ..., r_a(t-1)$. The available resource series $r_a(t|t), r_a(t+1|t), ..., r_a(t+h|t)$ are obtained by iterating the one-step prediction. Table II illustrates how one-step prediction is iterated to obtain a 3-step prediction. We study the impact of different prediction horizons in Section V.

TABLE II
3-STEP PREDICTION OF ARIMA MODEL.

Steps	Inputs of ARIMA	Output
1	$r_a(t-3\|t), r_a(t-2\|t), r_a(t-1\|t)$	$r_a(t\|t)$
2	$r_a(t-2\|t), r_a(t-1\|t), r_a(t\|t)$	$r_a(t+1\|t)$
3	$r_a(t-1\|t), r_a(t\|t), r_a(t+1\|t)$	$r_a(t+2\|t)$

D. Scheduling Optimizer

The scheduling optimizer is invoked at each control interval. It solves the RHC control problem using quadratic programming and outputs a sequence of resource adjustments that minimize fine-grained job deadline misses at each interval. Then, RDS applies the resource adjustment to individual jobs via a two-level scheduling.

Job management: RDS maintains two separate queues for current running jobs and waiting jobs. By default, incoming jobs enter the waiting queue. The scheduling optimizer determines which job is to be moved to the running queue based on the solution of the RHC control problem. For example, low priority jobs or jobs with distant deadlines may not be immediately allocated resources by the RHC control algorithm, i.e., $u_j(t) = 0$. These jobs wait until they receive resource allocation from the scheduling optimizer. Since the resource allocation is determined once for every control interval, it is possible that short jobs whose deadline is earlier than the next control interval may miss their deadlines due to the late allocation of resources. To this end, we provide a fast path for short jobs in the job queue management, that is, RDS immediately moves a short job to the running queue.

Task scheduling: RDS applies the resource adjustment to individual jobs by changing the number of execution slots assigned to each job. Assigning more homogeneous slots to a job leads to more resources allocated to the job. Although the actual resources allocated with different slots may vary, we found that the total number of slots assigned to a job is a good approximation of the job's allocated resources. RDS uses a minimally invasive approach to realize dynamic number of slots assigned to each job. Algorithm 1 shows how dynamic slots are realized via task scheduling. First, jobs are sorted according to their resource adjustment in the next control interval. At each heartbeat, the job with the largest resource adjustment (line 5) is selected to run its task. After assigning one slot to this job, the job's resource adjustment is updated by subtracting the amount of resource equivalent to the size of one slot r_{slot} (line 9). We calculate r_{slot} based on the total available resource r_a and the total number of slots in the cluster. We discuss two scenarios in Section V, where r_{slot} is treated differently.

Algorithm 1 Task scheduling with dynamic slots.

```
Update the running job queue
repeat
    if Any slot is available then
        /*Select a job in the running job queue*/
        j = arg max_j [Δu_j(t)], j ∈ {1, ··· , J}
        Select a local task from the job j
        Assign selected task to the available slot
        /*Update control adjustment scheme*/
        Δu_j(t) = Δu_j(t) − r_slot
    end if
until Σ_{j=1}^{J} (u_j^m(t) + u_j^r(t)) = r_a
```

The algorithm effectively changes the number of slots of each job. If $\Delta u_j(t) > 0$, the job eventually will be assigned free slots. If $\Delta u_j(t) < 0$, the job actually gives up the opportunity to run tasks, which is equivalent to reducing its number of allocated slots.

IV. System Implementation

A. Testbed

We built a dynamic Hadoop cluster in our university cloud, which consists of 108-core CPUs and 704 GB memory. VMware vSphere 5.1 was used for server virtualization. VMware vSphere module controls the CPU usage limits in MHz allocated to the virtual machines (VMs). It also provides an API to support the remote management of VMs. Hadoop version 1.2.1 was deployed to the cluster with 21 VMs, i.e., one master node and 20 slave nodes. We configured each slave node with two map slots and one reduce slot. The block size is configured 64 MB in our experiments. Each VM was allocated 1 virtual CPU and 2 GB memory. All VMs ran Ubuntu Server 10.04 with Linux kernel 2.6.32.

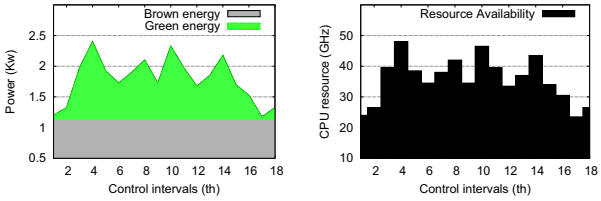
B. RDS Implementation

To implement RDS in the Hadoop environment, we added a new member `mapred.job.deadline` to store the deadline of a job to the class `JobConf`. We applied the idea of Hadoop capacity scheduler and refactoried its `QueueManager` class to implement RDS waiting and running job queues. We implemented the core components of RDS in the class `SchedulerTaskScheduling`.

Resource Predictor: We used a sensor program provided by VMware vSphere 5.1 to collect the resource availability of individual VMs. The cluster information, e.g., number of slots, was monitored by the function `getClusterStatus` in the JobTracker. We applied the ARIMA approach combined with the collected resource information of last 3 horizons to obtain the predicted resource in the next 3 intervals.

Performance Modeling: We used MATLAB Fuzzy Logic Toolbox to apply subtractive clustering and ANFIS modeling technique on the data collected from cluster. At runtime, the performance models were updated based on new measurements collected from the JobTracker using RLS algorithm.

Receding Horizon Control: The RHC controller module invoked a quadratic programming solver, *quadprog*, in MATLAB to compute the local control solution. We used MATLAB Builder JA to create a Java class from the MATLAB program calling *quadprog*. This Java class was integrated into RDS and deployed in the master node of the cluster. Based on the observations, we empirically set the control penalty weight matrix $P = [0.0107, 0.0096, 0.0132]$ and $Q = [0.0173, 0.0168, 0.0194]$ for map and reduce task, respectively. We set the control interval to 10 minutes.



(a) Energy supply. (b) Dynamic resource.

Fig. 5. A dynamic Hadoop cluster.

C. Workloads

For performance evaluations, we used a set of representative MapReduce applications from the PUMA benchmark [1], i.e., *Wordcount*, *Terasort* and *Grep*. By default, we set the same priority for each benchmark. We study the impact of different job priorities in Section V-D. For each application, we submit multiple copies with different input sizes as shown in the Table III that contains jobs with widely varying execution times and data set sizes, emulating a scenario where the cluster is used to run many different types of MapReduce applications. Similarly as Natjam et al. [8], we set the expected execution time of individual jobs to 2.5 times of the job completion time in dedicated cluster with sufficient resources. Then, we derived jobs' deadlines according to their arrival times. According to the study [25], we set inter-arrival time of jobs to 10 minutes.

D. Dynamic Resource Trace

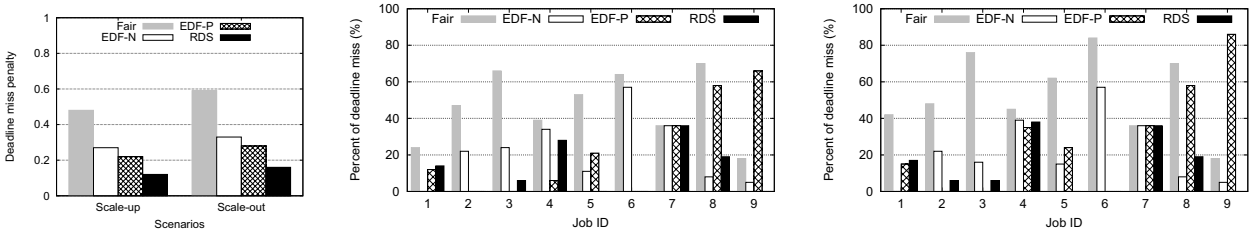
Multiple factors can cause dynamic resources in a Hadoop cluster. Here, we consider a Hadoop cluster running partially on renewable energy. We assume that the cluster is powered by half traditional energy and half green energy. Since the supply of renewable energy depends on weather conditions, the available resource to the cluster varies over time. We use the renewable power trace from National Renewable Energy Laboratory [18] and derive the traces for resource availability using the power model proposed by [26]. Figures 5(a) and 5(b) show the power trace and the derived resource trace, respectively. We scaled down the resource trace to match the cluster settings and replayed the trace by dynamically limiting the available resources in the cluster. We used vSphere API to change CPU resource allocation on individual VMs according to the cluster resource trace.

V. Evaluation

In this section, we evaluate RDS performance using various representative Hadoop benchmarks and dynamic resource traces. We first study the effectiveness of RDS in minimizing

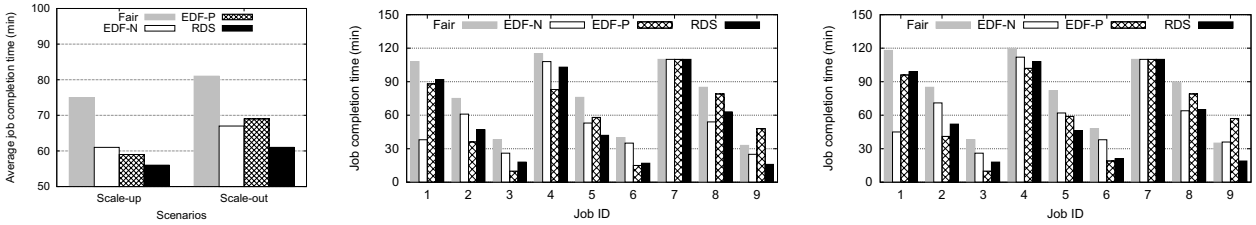
TABLE III
VARIOUS WORKLOAD CHARACTERISTICS.

Category	Label	Input size (GB)	Input data	# Maps	# Reduces	Deadline (min)
Wordcount	J1/J2/J3	60/35/15	Wikipedia	960/560/250	200/ 100/ 50	80/ 50/ 25
Grep	J4/J5/J6	80/40/20	Wikipedia	1280/640/320	200/ 100/ 50	80/ 50/ 25
Terasort	J7/J8/J9	50/25/12	TeraGen	800/450/192	200/ 100/ 50	80/ 50/ 25



(a) Deadline miss penalty. (b) The scale-up scenario. (c) The scale-out scenario.

Fig. 6. The job deadline misses due to dynamic resources by Fair Scheduler, EDF and RDS.



(a) Job completion time. (b) The scale-up scenario. (c) The scale-out scenario.

Fig. 7. The job completion time due to dynamic resources allocated by Fair Scheduler, EDF and RDS.

deadline miss penalty and reducing job completion time. Then, we evaluate the accuracy of the fuzzy modeling and the effectiveness of the RHC control. Finally, we study the parameter sensitivity and overhead of RDS.

We study two methods for organizing cluster resources when the total available resource varies dynamically.

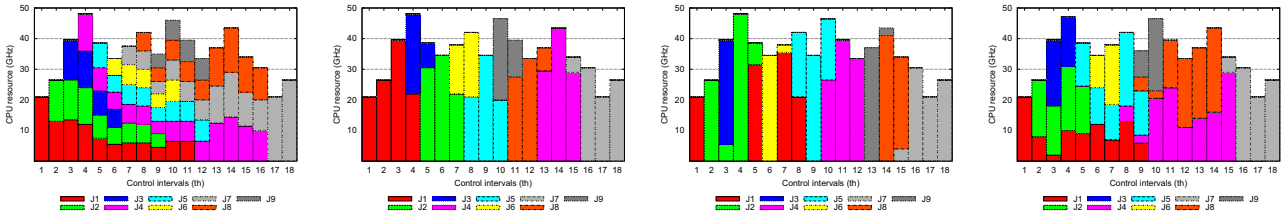
- **Scale-up**: The number of nodes in the Hadoop cluster is fixed while the capacity of individual nodes is changed in proportion to the change of total resource. In this scenario, r_{slot} changes with cluster capacity.
- **Scale-out**: The capacity of individual nodes is fixed while the number of nodes varies as the total resource changes. In this scenario, r_{slot} is also fixed.

We compare RDS with Hadoop Fair Scheduler. We also implement the Earliest Deadline First Scheduling (EDF) in scale-up and scale-out scenarios. Fair Scheduler allocates a fair number of slots to jobs sharing the cluster. EDF always runs the job that has the earliest deadline first requirement and gives the job as many slots as it needs. We implement two versions of EDF: non-preemptive (EDF-N) and preemptive (EDF-P). EDF-N enforces the EDF policy at the job queue – there is always only one job, the job with the earliest deadline, in the running queue. Even if a job with an earlier deadline arrives, it needs to wait for the running job's completion. EDF-P allows multiple jobs in the running queue. A job can preempt a running job's slots. However, EDF-P does not kill the running tasks of the preempted job but waits for their completion. The preempted job will not be assigned any slots.

A. Effectiveness of RDS

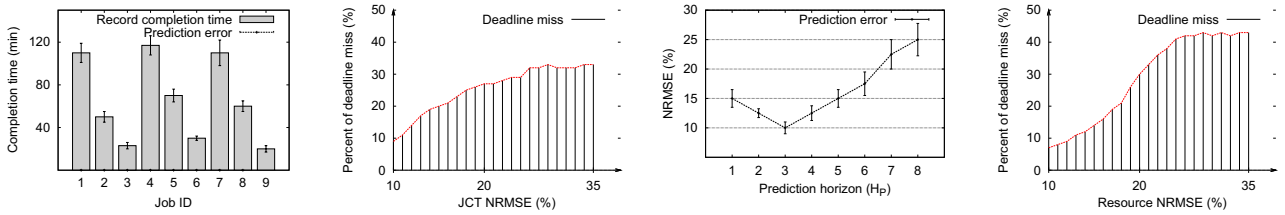
Minimizing deadline miss penalty. Figure 6 compares the deadline miss penalty for jobs J1-J9 (listed in Table III) incurred by Fair Scheduler, EDF-N, EDF-P and RDS. Note that the deadline penalty is zero if the deadline is met, and it increases linearly with the completion time. Figure 6(a) shows that RDS incurred 10%, 15% and 36% less penalty compared with EDF-N, EDF-P and Fair Scheduler in the scale-up scenario, respectively. In the scale-out scenario, RDS reduced the deadline miss penalty 12%, 17% and 43% compared with EDF-N, EDF-P and Fair Scheduler, respectively.

Figures 6(b) and 6(c) quantify the deadline misses for individual jobs. We can see that Fair scheduler incurred most significant misses due to its obliviousness of job deadlines. EDF-N and EDF-P effectively reduced the extent of deadline misses and met some jobs' deadlines such as J1 and J2. RDS further reduced the miss penalty. Note that RDS did not outperform both EDF-N and EDF-P in all cases. For example, EDF-N was the optimal policy for J1, which was the first job submitted. Overall, RDS achieved consistently better performance than either EDF-N or EDF-P. This is due to its capability to optimize job scheduling considering future



(a) Fair Scheduler. (b) EDF-N. (c) EDF-P. (d) RDS.

Fig. 8. The adaptive resource allocations by Fair Scheduler, EDF-N, EDF-P and RDS.



(a) JCT prediction error. (b) Impact of JCT error. (c) Resource prediction error. (d) Prediction error impact.

Fig. 9. The prediction accuracy and error impact of RHC control.

resource availability in the prediction window. Such resource-aware optimization is critical in overloaded clusters.

Reducing job completion time. Figure 7 compares the job completion time achieved by Fair Scheduler, EDF-N, EDF-P and RDS. Figure 7(a) shows that RDS improved the average job completion time by 5%, 9% and 19% compared with EDF-P, EDF-N and Fair Scheduler, respectively. In particular, Figures 7(b) and 7(c) show that RDS significantly reduced the completion time of small jobs, e.g., J3, J6 and J9. Relatively, RDS was less effective for large jobs, e.g., J1, J4 and J7. This is due to the fact that large jobs usually have long deadlines and the misses will not immediate lead to large penalties. Thus, such jobs are not favored by the optimization. Figure 7(a) also reveals that the average job completion time of the scale-out scenario was 7% longer than that of the scale-up scenario. This is due to the larger operation overhead when adding or removing Hadoop nodes. Job data needs to be balanced across remaining nodes, which inevitably incurs overhead.

B. *Effectiveness of RHC Control and Fuzzy Modeling*

Adaptive resource allocation. Figure 8 depicts the dynamic resource allocations when running J1-J9 by different approaches. Figure 8(a) shows that Fair Scheduler evenly allocated resource to all jobs in running queue. However, the fairness in allocation leads to significant deadline misses. Figure 8(b) shows that EDF-N prioritized the job with the earliest deadline and the running job monopolized the cluster until it completed. Figure 8(c) shows that EDF-P preempted the current running job if a job with an earlier deadline arrived. In contrast, Figure 8(d) shows that RDS scheduler dynamically adjusted the resource allocation to different jobs according to the available resource prediction and various job deadlines. This is due to the fact that RHC controller directed more resource to the job with an approaching deadline. Thus, we conclude that the RHC controller effectively allocated resources to jobs considering deadlines and dynamic resource availability.

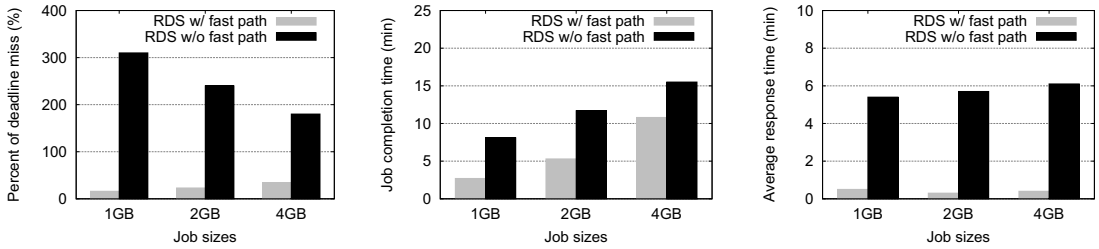
Accuracy of performance and resource modeling. To evaluate the accuracy of the fuzzy models, we compare the error between the predicted job completion times and the actual completion times. The accuracy is measured by the normalized root mean square error (NRMSE), a standard metric for deviation [14]. Figure 9(a) shows that the prediction was quite accurate, with on average 9.6% NRMSE. To study how the prediction error affect the working of RHC, we injected errors into the prediction. Figure 9(b) shows that the deadline misses increased from 10.5% to 33% as the prediction error increased from 10% to 35%.

Figures 9(c) and 9(d) plot the error of resource prediction and its impact on the deadline misses. Figure 9(c) shows that the NRMSE of resource prediction varied with different prediction horizons. It suggests that a prediction horizon of 3 steps give the minimum error. Figure 9(d) shows that deadline misses increased significantly as the resource prediction error increases. This observation again confirms that resource-aware scheduling is critical to avoiding deadline misses.

C. *Mix of Big and Small jobs*

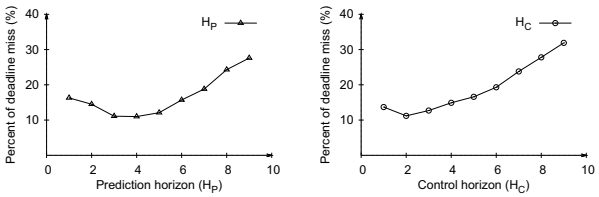
RDS provides a fast path for small jobs with deadlines earlier than the next control interval. We evaluate the effectiveness of this mechanism whiling handling small jobs.

We add three small instances of *Wordcount* with 1 GB, 2 GB and 4 GB input sizes into the Hadoop cluster at the 5_{th}, 15_{th} and 25_{th} minute respectively. Their deadlines are set to 3, 5 and 10 minutes away from their submission times. As shown in Figure 10, we compare the performance of RDS with/without the fast path. Figure 10(a) shows that RDS with fast path for small jobs significantly reduced small jobs' deadline misses. Without the fast path, deadline misses reduced as the job size increased. Figure 10(b) shows that the



(a) Deadline miss. (b) Job completion time. (c) Average response time.

Fig. 10. The small job performance improvements by fast path policy of RDS.



(a) Impact of prediction horizon. (b) Impact of control horizon.

Fig. 11. Prediction and control horizon impact.

TABLE IV
THE CONTROL ADJUSTMENT OVERHEADS.

Tasks	Wordcount	Grep	Terasort	Average
Map	5.3s	3.6s	7.2s	5.2s
Reduce	9.3s	7.8s	12.5s	10.6s

job completion time achieved by the fast path RDS was 50% less. Moreover, Figure 10(c) show that small jobs were able to join the running queue (i.e., lower response time) almost instantly when submitted.

D. Parameters Sensitivity

Control horizon. Settings of the prediction (i.e., H_p) and control horizon (H_c) may affect the RHC control algorithm. We change their values and plot their effect on job deadline misses in Figure 11. The results show that deadline misses initially drop as H_P and H_c increase. However, further increasing both parameters leads to worse deadline misses. Thus, we empirically set $H_p = H_c = 3$ in all experiments.

Job priority support. Figure 12 shows the different deadline misses as the workload priority varies. Figure 12(a) shows that deadline misses of *wordcount* decreased slowly as its priority increased from 1 to 3. At the same time, the deadline misses of *grep* increased quickly compared with that of *terasort*. This is because *wordcount* obtained more resource from *grep* than from *terasort*. Figures 12(b) and 12(c) show similar results as the priorities of *grep* and *terasort* varied. In summary, priorities are effectively supported by RDS.

E. Overhead

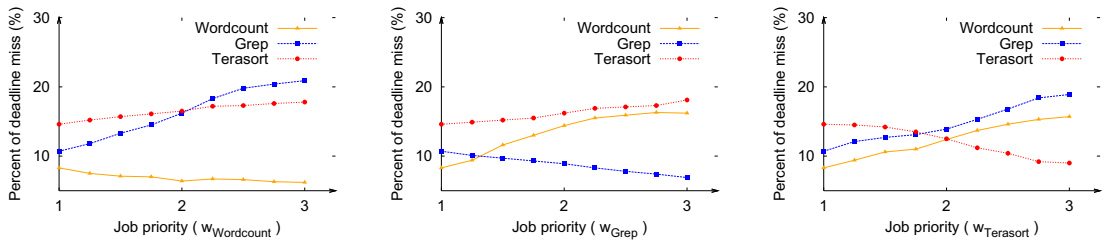
The overhead of RDS comes from two sources: (1) the time required to activate control adjustment according to Algorithm 1; (2) the time required to perform scheduling optimizer in each control interval. We first measured the activation overhead of RDS under different job types. It took on average of 5.2 seconds to activate a map task adjustment and 10.6 seconds for a reduce task. Table IV shows the costs to activate the control adjustment for *wordcount*, *grep* and *terasort*, respectively. Recall that the overhead caused by control adjustment is considered as control penalty and is weighted by penalty matrixes P and Q in the scheduling optimizer, respectively. We also found that the scheduling optimization algorithm took approximately 1.2 seconds to complete. This overhead is negligible compared to the control interval of 10 minutes.

VI. RELATED WORK

Hadoop optimization: There are growing interests on MapReduce performance optimization with various techniques, e.g., resource provisioning [10], [24], job scheduling [16] and self-tuning configuration [7], [13]. Rao *et al.* proposed a new MapReduce framework, Sailfish [21], to improve performance by aggregating intermediate data. Jinda *et al.* [12] proposed a new data layout, coined Trojan Layout, that internally organizes data blocks into attribute groups in order to improve data access times. None of these existing approaches consider to optimize MapReduce performance by dynamic job scheduling in the multi-user environment.

Task scheduling: Many prior studies have shown that MapReduce performance can be significantly improved by various scheduling techniques [8], [25]. The default FIFO Scheduler in Hadoop implementation may not work well since a long job can exclusively take the computing resource on the cluster, and cause large delays for other jobs. This is the reason that many schedulers, e.g., Capacity Scheduler, Fair Scheduler, can share resources among multiple jobs. Recently, a few studies [8], [24] start to optimize the performance of MapReduce jobs with respect to their performance goals. Wolf *et al.* described FLEX [25], a flexible and intelligent allocation scheme for MapReduce workloads. Our work differs from these efforts in that we consider a Hadoop cluster with fluctuating resource availability.

Hybrid environment: In fact, existing studies have demonstrated that MapReduce cluster size usually fluctuates for many reasons in a real system. Sharma *et al.* described MapReduce



(a) Wordcount impact. (b) Grep impact. (c) Terasort impact.

Fig. 12. The job priority impact by Wordcount, Grep and Terasort respectively.

workload typically shared physical resources with other workloads, e.g., interactive workload [23]. Thus, the actual amount of resources available for MapReduce applications can vary over time. Lang *et al.* proposed AIS [15], an alternative energy management framework to reduce the energy consumption of a MapReduce cluster and lead the cluster size to be time varying. Polo *et al.* [20] presented RAS, a resource aware adaptive scheduler for MapReduce. It aims to improve resource utilization across machines while considering completion time goals. However, the dynamic resource availability and the reconfiguration cost of job scheduling are not considered. Furthermore, our work differs from theirs in that we take advantage of cluster resource prediction to minimize job deadline misses and we also provide job priority support.

VII. Conclusion and Discussions

In this paper, we tackle the realistic but challenging problem of job scheduling in the Hadoop cluster with dynamic resource availability. We find that deadline misses in Hadoop workloads can be minimized by exploiting the dynamics in resource availability and the flexibility in Hadoop task scheduling. We propose, RDS, a resource and deadline-aware Hadoop job scheduler that allocates resources to jobs according to resource prediction and job completion time estimation. RDS uses an efficient online receding horizon control algorithm to possibly derive the optimal resource allocations to jobs. Experimental results show that RDS effectively reduces job deadline misses by at least 36% and 10% compared to Fair scheduler and the EDF scheduler, respectively.

Acknowledgement

This research was supported in part by U.S. NSF CAREER award CNS-0844983, grants CNS-1422119, CNS-1320122 and CNS-1217979, and NSF of China grant No.61328203.

References

[1] PUMA: Purdue mapreduce benchmark suite. http://web.ics.purdue.edu/~fahmad/benchmarks.htm.
[2] K. Astrom and B. Vittenmark. *Adaptive Control*. Prentice Hall, 1995.
[3] G. Box, G. Jenkins, and G. Reinsel. *Time Series Analysis, Forecasting, and Control*. Prentice-Hall, third edition, 1994.
[4] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Proc. of ACM/USENIX EuroSys*, 2012.
[5] Y. Chen, B. Yang, A. Abraham, and L. Peng. Automatic design of hierarchical takagisugeno type fuzzy systems using evolutionary algorithms. In *IEEE Trans. on Fuzzy Systems*, 2007.
[6] D. Cheng, C. Jiang, and X. Zhou. Heterogeneity-aware workload placement and migration in distributed sustainable datacenters. In *Proc. IEEE IPDPS*, 2014.
[7] D. Cheng, J. Rao, Y. Guo, and X. Zhou. Improving mapreduce performance in heterogeneous environments with adaptive task tuning. In *Proc. of ACM/IFIP/USENIX Middleware*, 2014.
[8] B. Cho, M. Rahman, T. Chajed, I. Gupta, C. Abad, N. Roberts, and P. Lin. Natjam: Eviction policies for supporting priorities and deadlines in mapreduce clusters. In *Proc. of ACM SoCC*, 2013.
[9] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: Leveraging green energy in data-processing frameworks. In *Proc. of ACM/USENIX EuroSys*, 2012.
[10] Y. Guo, J. Rao, C. Jiang, and X. Zhou. Flexslot: Moving hadoop into the cloud with flexible slot management. In *Proc. of ACM/IEEE SC*, 2014.
[11] Y. Guo, J. Rao, and X. Zhou. ishuffle: Improving hadoop performance with shuffle-on-write. In *Proc. of USNIX ICAC*, 2013.
[12] A. Jindal, J. Quiané-Ruiz, and J. Dittrich. Trojan data layouts: Right shoes for a running elephant. In *Proc. of ACM SoCC*, 2011.
[13] P. Lama and X. Zhou. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proc. of ACM ICAC*, 2012.
[14] P. Lama and X. Zhou. NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers. In *Proc. of IEEE/IFIP DSN*, 2012.
[15] W. Lang and J. M. Patel. Energy management for mapreduce clusters. In *Proc. of VLDB*, 2010.
[16] S. Li, S. Hu, s. Wang, L. Su, T. Abdelzaher, I. Gupta, and R. Pace. Woha: Deadline-aware map-reduce workflow scheduling framework over hadoop clusters. In *Proc. of IEEE ICDCS*, 2014.
[17] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *Proc. of ACM SIGMETRICS*, 2012.
[18] NREL. Measurement data center. http://www.nrel.gov/midc/.
[19] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.
[20] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguade. Resource-aware adaptive scheduling for mapreduce clusters. In *Proc. of ACM/IFIP/USENIX Middleware*, 2011.
[21] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves. Sailfish: A framework for large scale data processing. In *Proc. of ACM SoCC*, 2012.
[22] A. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of ACM SoCC*, 2012.
[23] B. Sharma, T. Wood, and C. R. Das. Hybridmr: A hierarchical mapreduce scheduler for hybrid data centers. In *Proc.of ICDCS*, 2013.
[24] A. Verma, L. Cherkasova, V. S. Kumar, and R. H. Campbell. Deadline-based workload management for mapreduce environments:pieces of the performance puzzle. In *Proc. of IEEE/IFIP NOMS*, 2012.
[25] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K. Wu, and A. Balmin. Flex: A slot allocation scheduling optimizer for mapreduce workloads. In *Proc. of ACM/IFIP/USENIX Middleware*, 2010.
[26] Q. Zhang, F. Mohamed, S. Zhang, Q. Zhu, B. Raouf, and L. Joseph. Dynamic energy-aware capacity provisioning for cloud computing environments. In *Proc. of ACM ICAC*, 2012.