Deadline-Aware MapReduce Job Scheduling with Dynamic Resource Availability

Dazhao Cheng, *Member, IEEE*, Xiaobo Zhou, *Senior Member, IEEE*, Yinggen Xu, Liu Liu, and Changjun Jiang, *Member, IEEE*

Abstract—As MapReduce is becoming ubiquitous in large-scale data analysis, many recent studies have shown that the performance of MapReduce could be improved by different job scheduling approaches, e.g., Fair Scheduler and Capacity Scheduler. However, most exiting MapReduce job schedulers focus on the scenario that MapReduce cluster is stable and pay little attention to the MapReduce cluster with dynamic resource availability. In fact, MapReduce cluster resources may fluctuate 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. Thus, there is a growing need for providing predictable services to users who have strict requirements on job completion times in such dynamic environments. 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. We further extend the design of RDS scheduler to support flexible performance goals in various dynamic clusters. In particular, we use flexible deadline time bounds instead of the single fixed job completion deadline. 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 percent compared with Fair Scheduler and Earliest Deadline First (EDF) scheduler, respectively. In a Hadoop cluster running partially on renewable energy, the experimental result shows the green power based resource prediction approach can further reduce the penalty of deadline misses by 16 percent compared to Auto-Regressive Integrated Moving Average (ARIMA) prediction approach.

Index Terms—MapReduce, job scheduling, deadline-aware, dynamic resource availability, horizon control, job completion times

1 Introduction

Today, a large amount of data has to be collected due to the growing number of Internet services and Web applications from enterprises. As rising of the data size, technologies for analyzing big data are evolving rapidly and there is significant interest in new analytic approaches such as MapReduce. For example, the Data warehouse Hadoop (i.e., the open source implementation of the MapReduce programming model) cluster at Facebook contains 3000 machines and hosts on average 25000 MapReduce jobs per day [1]. However, study [2] has shown that current use of Hadoop in research and enterprises still has significant room for improvement on the performance of MapReduce jobs and the utilization of Hadoop clusters. Although many MapReduce jobs are delay tolerant, there are still some jobs that have deadline constraints [3], [4]. One example is generating statistics of user log information on e-commerce websites which will be later used for advertisement placement optimizations or personalized content recommendations. The site performance and the revenue for the company are directly affected by whether or not jobs can finish within a given amount of time.

Unfortunately, it is challenging to meet job deadlines based on current Hadoop platforms. First, as jobs have diverse resource demands, it is difficult to estimate the desired resource amount for individual jobs to avoid its deadline miss. Second, MapReduce clusters are usually shared by multiple jobs and the scheduling approach of these jobs can affect job completion times [3]. 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 [5], and their variations [6], [7], 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 [3] or task scheduling [4].

As there is a growing number of MapReduce clusters deployed in hybrid systems, it further complicates the problem. For example, eBay [8] and Yahoo! [9] employ MapReduce to generate reports and answer historical queries (i.e., batch jobs), while deploying other platforms (e.g., Spark

- *D. Cheng is with the Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223. E-mail: dazhao.cheng@uncc.edu.*
- *X. Zhou is with the Department of Computer Science, University of Colorado, Colorado Springs, Boulder, CO 80918. E-mail: xzhou@uccs.edu.*
- *Y. Xu, L. Liu, and C. Jiang are with the Department of Computer Science & Technology, Tongji University, Shanghai, China. E-mail: {xuyg, liuliu1991, cjjiang}@tongji.edu.cn.*

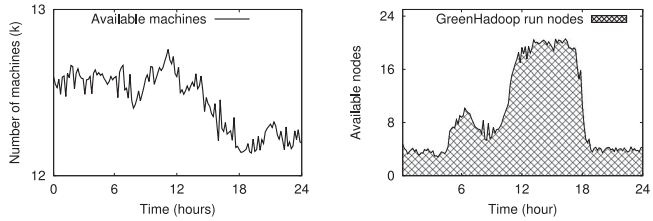
Manuscript received 18 Jan. 2018; revised 15 Aug. 2018; accepted 19 Sept. 2018. Date of publication 1 Oct. 2018; date of current version 13 Mar. 2019. (Corresponding author: Dazhao Cheng.)
Recommended for acceptance by F. Qin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2873373

and Storm) at the same time to calculate key metrics in real-time. In this case, the cluster resource will be first reserved for these real-time jobs. Thus, the available resource for MapReduce jobs may become dynamic over time. There are more scenarios may lead the resources available to MapReduce jobs quite dynamic, such as such as Greenhadoop at Rutgers university [6] and a green datacenter at the HP labs [10], and 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 violations in MapReduce workloads may be significantly 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. 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 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 percent compared to Fair scheduler and the earliest deadline first (EDF) scheduler, respectively.

A preliminary version of this paper appeared in [11]. In this manuscript, we have extended the RDS design from an adaptive job scheduling approach to a holistic performance aware job scheduling system for various dynamic clusters. More specifically, we make the following new contributions.

- We have used a service level objective (SLO) with two time bounds, t_{hard} and t_{soft}, in the decay function to discount the data throughput from jobs with violated SLOs while scheduling jobs.
- We have integrated green power prediction technique with the deadline aware job scheduling approach (i.e., RDS scheduler) to further improve the system performance in dynamic Hadoop clusters. In a Hadoop cluster running partially on renewable energy, the new experimental result shows the green power based resource prediction approach can further reduce the penalty of deadline misses by 16 percent compared to ARIMA prediction approach.
- We have demonstrated that the proposed RHC strategy achieves closed-loop stability independent of the performance parameters. We have carried out new experiments and analysis with the extended approach, and studied the impact of different deadline settings (i.e., hard and soft time bounds) on system performance.
- We have conducted new experiments to evaluate the performance and impact of different dynamic resource prediction approaches. We have also analyzed the relationship between the system overhead cost and various configurations.



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

Fig. 1. Dynamic production and Hadoop clusters.

The rest of this paper is organized as follows. Section 2 gives motivations on resource and deadline aware Hadoop scheduling. Section 3 describes the design of RDS. Section 5 gives details on system implementation. Section 6 presents the experimental results. Section 7 reviews related work. Section 8 concludes the paper.

2 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.

2.1 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 [12]. Fig. 1a 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 [10]. Unlike traditional energy, the intermittency of

TABLE 1
Job 1 and 2 Submission Information

Jobs	Input	Demand	Arrival time	Deadline
J1	18 GB	120 GHz	0th min	40th min
J2	9 GB	60 GHz	10th min	30th min

renewable energy makes it very hard to maintain a stable cluster resource availability to process workloads. Goiri et al. proposed GreenHadoop [6], a MapReduce framework for Parasol, a prototype green cluster built in Rutgers University. Fig. 1b 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.

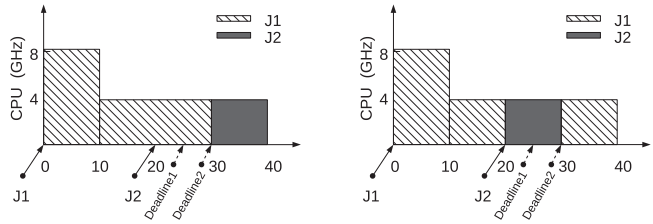
The above analysis suggests that the benefit of dynamic capacity provisioning is apparent for production datacenters from the perspective of both economics and environments. However, it also brings up a challenging task to manage dynamic datacenter clusters. In order to further explore this problem,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

2.2 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 *wordcount* [13] jobs. Two jobs have different input sizes, resource demands and deadlines. Table 1 gives their information.

Fig. 2 shows the performance of three schedulers in the dynamic cluster. Note that the CPU demand in Table 1 is the cumulative resource requirement for running individual jobs. The GHz in Figs. 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 Fig. 2b should equal the demand of J1 in Table 1, which is 120 GHz. In the 10th to 20th and 30th to 40th intervals, the cluster has doubled resources than in other intervals.

Fig. 2a shows the trace of dynamic resource availability. Figs. 2b, 2c, and 2d 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 20th to 30th is not sufficient. Knowing that more resources would be available in the 30th to 40th interval, the ideal scheduler prioritized J2 with more resources from 10th to 30th. The flexible resource allocation effectively guaranteed the deadlines of both jobs.

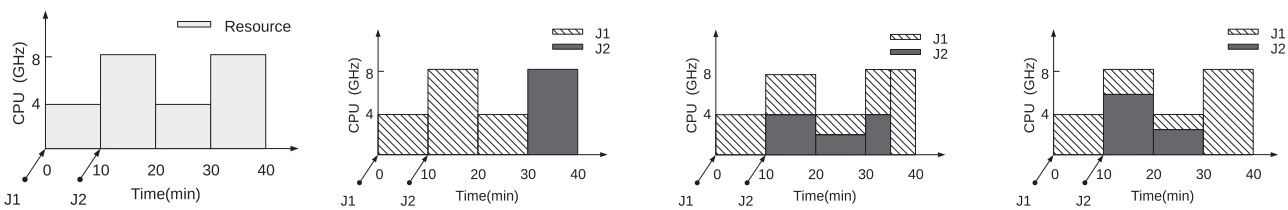


(a) EDF-preemptive. (b) Ideal scheduler.

Fig. 3. Performance in an overloaded system.

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 20th to 30th interval further drops, a deadline-aware scheduler should allocate more resources to J2 in the 10th to 20th 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 [14]. 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. Fig. 3a shows the scheduling order of two jobs under EDF. The system become overloaded as the available resources dropped at the 10th 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 Fig. 3b). Second, EDF does not determine how much resource is needed to meet job deadlines [3]. 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.



(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.

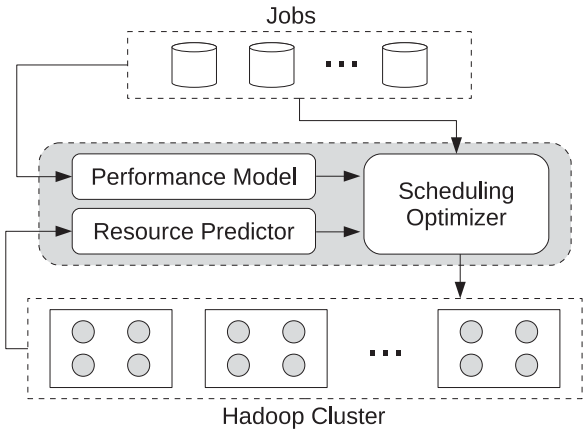


Fig. 4. The architecture of RDS.

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.

3 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 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.

Fig. 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.

3.1 Problem Formulation

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

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

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

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 at any time.

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:

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

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. To count only useful work, decay function δ discounts the data throughput from jobs with violated SLOs. 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.

We use a SLO with two time bounds, t_{hard} and t_{soft}, in the decay function. While t_{hard} sets a hard deadline for job completion, beyond which no revenue is generated, t_{soft} is a soft deadline whose violation will incur reduction in revenue. We ensure that hard deadlines to be no longer than 2 times of the soft deadlines so that the decay function never becomes negative. Accordingly, δ considers the data delivered as useful work if the observed job finish time y_j meets t_{soft}. Violations of t_{soft} and t_{hard} result in a linear decay in the counted throughput and zero work, respectively.

$$\delta = \begin{cases} 1 & \text{if } y_j < t_{soft}. \\ 1 - \frac{y_j - t_{soft}}{t_{soft}} & \text{if } t_{soft} \leq y_j \leq t_{hard}. \\ 0 & \text{if } y_j > t_{hard}. \end{cases} \tag{4}$$

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 $\Delta u(t)$, $\Delta u(t+1), \ldots, \Delta u(t+H_c)$ 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.

3.2 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 [15]. 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.

3.2.1 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),

$$y_j(t) = R_j(u(t), d_j, \xi(t)). \tag{5}$$

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

$$\begin{aligned} \xi(t) = [&(y(t-1), y(t-2), \ldots, y(t-n_y)), \\ &(u(t), u(t-1), \ldots, u(t-n_u))]^T, \end{aligned} \tag{6}$$

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,

$$\rho = n_y + n_u. \tag{7}$$

R is the rule-based fuzzy model that consists of Takagi-Sugeno rules [16]. A rule R_j is represented as

$$\begin{aligned} R_j : \text{IF } &\xi_1(t) \text{ is } \Omega_{j,1}, \xi_2(t) \text{ is } \Omega_{j,2}, \ldots, \text{ and } \xi_\rho(t) \text{ is } \Omega_{j,\rho} \\ &u(t) \text{ is } \Omega_{j,\rho+1} \text{ and } d_j \text{ is } \Omega_{j,\rho+2} \\ \text{THEN } &y_j(t) = \zeta_j \xi(t) + \eta_j u(t) + \omega_j d_j + \theta_j. \end{aligned} \tag{8}$$

Here, Ω_j is the antecedent fuzzy set of the jth rule, which is composed of a series of subsets: $\Omega_{j,1}, \Omega_{j,2}, \ldots, \Omega_{j,\rho+2}$. 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.

3.2.2 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 [17] 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. (5) as follow:

$$y(t) = \phi(t)X + e(t), \tag{9}$$

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 jth 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. (10) is minimized. We apply both the current error $e(t)$ and the previous error $e(t-1)$ to estimate the parameter vector,

$$Error = \sum_{t-1}^{t} (e(t)^2 + \tau e(t-1)^2). \tag{10}$$

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.

3.3 Stability and Convergence

We demonstrate that the above RHC achieves closed-loop stability and the closed loop system under the above RHC control is asymptotically stable. Moreover, the solution of function 3 satisfies $y_t \in \chi_N$, $u_t \in \Lambda$, $\lim_{t\to\infty} y_t = y_t^{ref}$ and $\lim_{t\to\infty} \Delta u_t = 0$, where χ_N and Λ are the range of y_t and u_t.

For the stabilization problem, without loss of generality, it is assumed that the origin ($y = 0$ and $\Delta u = 0$) is the steady state that should be stabilized. Also the prediction horizon is set equal to the control horizon, $H_p = H_c = N$. Then the cost function 3 is transferred to:

$$J_N(y, u) = \Sigma_{t=0}^{N} l(y_t, \Delta u_t) + F(y_N), \tag{11}$$

where the predicted states y_t are defined by Equation (5). Moreover, $l(y_t, \Delta u_t)$ is a continuous function and $F(y_N)$ is a local stabilizing control law, an associated Lyapunov function F_f. We define $J_N(y, u)$ has a solution $y \in Y$, which yields an optimal control sequence $\Delta u^*(y) = [\Delta u_0^*(y), \Delta u_1^*(y), \ldots, \Delta u_N^*(y)]$, an optimal trajectory $[y_0^*(y), y_1^*(y), \ldots, y_N^*(y),]$, the optimal value $V_N(y) = J_N(y, \Delta u^*(y))$. More specifically, there exist $\kappa_f : \chi \to \mathbb{R}^J, F_f : \chi \to \mathbb{R}, C_f > 0$ such that

$$\begin{aligned} F_f \text{ is continuous with } F_f(y) \geq 0 \, forally \in \chi; \\ 0 \in \chi_f := \{y \in \chi | F_f(y) \leq C_f\}; \\ \kappa_f(y) \in \Lambda \\ F_f(R(y, \kappa_f(y))) - F_f(y) \leq -l(y, \kappa_f(y)); \end{aligned} \tag{12}$$

Based on the definition of optimal trajectory, we have,

$$\begin{aligned} & F(y_N^*(y(t+1))) - F(y_{N-1}^*(y(t+1))) \\ & + l(y_{N-1}^*(y(t+1), \Delta u_{N-1}^*(y(t+1)) - l(y(t), \Delta u(t)) \\ \leq & F(R(y_N^*(y(t))), \kappa_f(y_N^*(y(t))) - F(y_N^*(y(t))) \\ & + l(y^*(y(t)), \kappa_f(y_N^*(y(t))) - l(y(t), \Delta u(t)). \end{aligned} \tag{13}$$

If $y_N^*(y(t))) \in \chi_f$, then the equation implies

$$\begin{aligned} & F(y_N^*(y(t+1)) - F(y_{N-1}^*(y(t+1)) \\ & + l(y_{N-1}^*(y(t+1), \Delta u_{N-1}^*(y(t+1)) \leq 0. \end{aligned} \tag{14}$$

The prior studies have shown that the set χ_N is invariant under the RHC law. Given above all, we have

$$\begin{aligned} & V_N(y(t+1)) - V_N(y(t)) \\ \leq & - l(y(t), \Delta u(t)) + l(y_N^*(y(t)), \kappa_f(y_N^*(y(t))) \\ & + F(R(y_N^* y(t)), \kappa_f(y_N^*(y(t))) - F(y_N^* y(t)) \\ \leq & - l(y(t), \Delta u(t)). \end{aligned} \tag{15}$$

It is illustrated that the value function is strictly decreasing so that V_N can serve as a Lyapunov function, proving stability and convergence.

4 Predicting Resource Availability

4.1 ARIMA Model Based Prediction

We use a simple but effective Auto-Regressive Integrated Moving Average (ARIMA) model [18] to predict the resource availability in future intervals based on history information. The ARIMA model has been used to predict resource consumption [19], and dynamic power supply [1].

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., $r_a(t-1)$, ... $r_a(t-n+1)$.

$$r_a(t) = a_1 r_a(t-1) + a_2 r_a(t-2) + \cdots + a_n r_a(t-n), \tag{16}$$

TABLE 2
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	$r_a(t-1\|t), r_a(t\|t), r_a(t+1\|t)$	$r_a(t+2\|t)$

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 hth step prediction of $r_a(t)$ knowing the last n observations, i.e., $r_a(t-n), \ldots, r_a(t-1)$. The available resource series $r_a(t|t)$, $r_a(t+1|t), \ldots, r_a(t+h|t)$ are obtained by iterating the one-step prediction. Table 2 illustrates how one-step prediction is iterated to obtain a 3-step prediction. We study the impact of different prediction horizons in Section 6.

4.2 Green Power Based Prediction

In contrast to brown energy, solar and wind energy generation may not provide a reliable, consistent source of energy due to the time-varying weather conditions. We use prediction methods to estimate the amount of green energy in a given interval and utilize that data to make decisions of workload placement and resource provisioning in distributed datacenters.

We use the model introduced by the work in [20] to predict solar and wind power. The solar model is based on the simple premise that energy generation is inversely related to the amount of cloud coverage. It is expressed as: $P^{solar}(k) = B^{solar}(k)(1 - CloudCover)$, where $P^{solar}(k)$ is the amount of solar power predicted for interval k, $B^{solar}(k)$ is the amount of solar power expected under ideal sunny conditions, and $CloudCover$ is the forecasted percentage cloud cover (given as a fraction between 0 and 1). We use historical data from NREL [21] to instantiate B^{solar}.

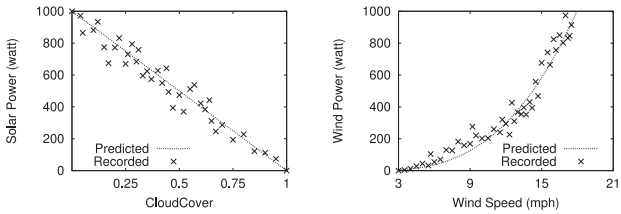
The wind power model is based on the cubic wind power production function [20]. That is, $P^{wind}(k) = a \times (v^w(k))^3 + b$, where v^w is the windspeed, a and b are parameters depending on the locations. We fit this power curve with the observed data from NREL [21] using the least-squares method to generate the wind power model. The parameters a and b used in our implementation are obtained empirically.

Given the data from NREL [21], we implement solar and wind energy prediction models at the granularity of 10 minutes. We assume that each sustainable datacenter has 7 solar panels and a micro-turbine with capability of producing $1\,KW$ respectively. Fig. 5 demonstrates that the average difference between each observed and predicted value is small.

In this work, we focus on the CPU resource allocation since CPU is the major power contributor in datacenters [6], [10], [22]. There is a linear relationship between the available CPU resource and the power consumption [22], [23]. Thus, the power constraint in the optimization problem is transformed to the resource availability constraint.

4.3 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



(a) Solar power model prediction. (b) Wind power model prediction.

Fig. 5. The accuracy of green power prediction.

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 6, where r_{slot} is treated differently.

Algorithm 1. Task Scheduling with Dynamic Slots

1: Update the running job queue
2: **repeat**
3: **if** Any slot is available **then**
4: /*Select a job in the running job queue*/
5: $j = \arg\max_j [\Delta u_j(t)], j \in \{1, \ldots, J\}$
6: Select a local task from the job j
7: Assign selected task to the available slot
8: /*Update control adjustment scheme*/
9: $\Delta u_j(t) = \Delta u_j(t) - r_{slot}$
10: **end if**
11: **until** $\sum_{j=1}^{J}(u_j^m(t) + u_j^r(t)) = r_a$

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.

5 System Implementation

5.1 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.

5.2 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 conducted a set of experiments consisting various workloads to measure the activation overhead of RDS under different job types. We then use the measured overhead in the control function and empirically set the control penalty weight matrix by MATLAB. The control interval is set to 10 minutes.

5.3 Workloads

For performance evaluations, we used a set of representative MapReduce applications from the PUMA benchmark

TABLE 3
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

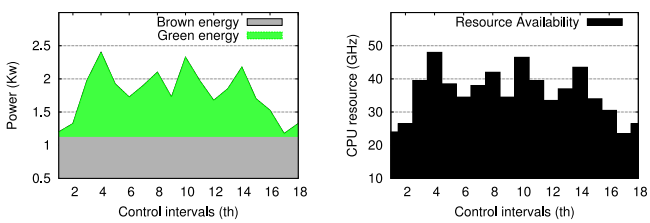
[13], i.e., *Wordcount*, *Terasort* and *Grep*. By default, we set the same priority for each benchmark. We study the impact of different job priorities in our preliminary study [11]. For each application, we submit multiple copies with different input sizes as shown in the Table 3 that contains jobs 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. [4], 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 [24], we set inter-arrival time of jobs to 10 minutes.

5.4 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 [21] and derive the traces for resource availability using the power model proposed by [19]. Figs. 6a and 6b 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.

6 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 deadline miss penalty and reducing job completion time. Then, we evaluate the impact of hard/soft deadlines,the accuracy of the fuzzy modeling and the effectiveness of the RHC control. Finally, we study the impact of different resource prediction approaches, the parameter sensitivity and the overhead of RDS.



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

Fig. 6. A dynamic Hadoop cluster.

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

- *Scale-up*: The number of nodes in the 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.

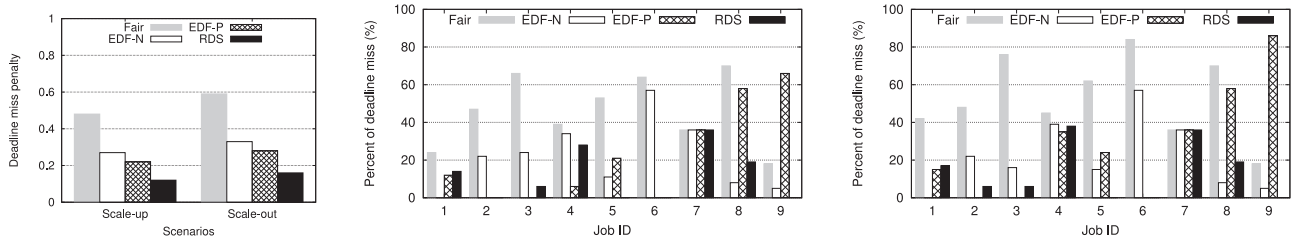
We compare RDS with Hadoop Fair Scheduler. We also implement the Earliest Deadline First Scheduling (EDF) in scale-up and scale-out scenarios. Fair Scheduler allocates a fair number of slots to jobs sharing the cluster. EDF always runs the job that has the earliest deadline first requirement and gives 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.

6.1 Effectiveness of RDS

Minimizing Deadline Miss Penalty. Fig. 7 compares the deadline miss penalty for jobs J1-J9 (listed in Table 3) 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. Fig. 7a shows that RDS incurred 10, 15 and 36 percent 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 percent compared with EDF-N, EDF-P and Fair Scheduler, respectively.

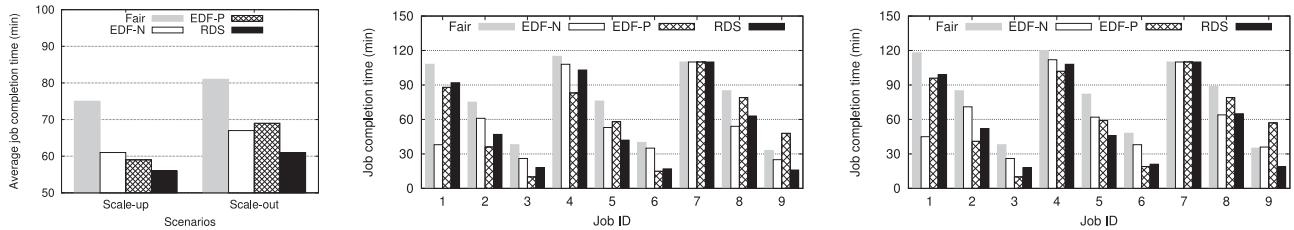
Figs. 7a and 7b quantify the deadline misses for individual jobs. We can see that Fair scheduler incurred most significant misses due to its obliviousness of job deadlines. EDF-N and EDF-P effectively reduced the extent of deadline misses and met some jobs' deadlines such as J1 and J2. RDS further reduced 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 resource availability in the prediction window. Such resource-aware optimization is critical in overloaded clusters.

Reducing Job Completion Time. Fig. 8 compares the job completion time achieved by Fair Scheduler, EDF-N, EDF-P and RDS. Fig. 8a shows that RDS improved the average job



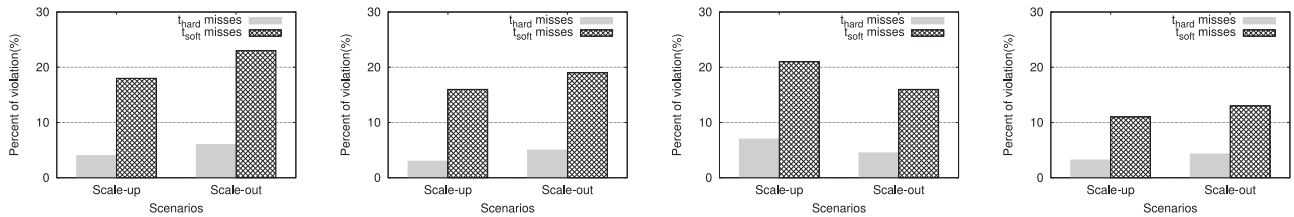
(a) Deadline miss penalty. (b) The scale-up scenario. (c) The scale-out scenario.

Fig. 7. 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. 8. The job completion time due to dynamic resources allocated by Fair Scheduler, EDF and RDS.



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

Fig. 9. The hard and soft time bounds impact by Wordcount, Grep, Terasort and Hive respectively.

completion time by 5, 9 and 19 percent compared with EDF-P, EDF-N and Fair Scheduler, respectively. In particular, Figs. 8b and 8c show that RDS significantly reduced the completion time of small jobs, e.g., J3, J6 and J9. Relatively, RDS was less effective for large job's, 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. Fig. 8a also reveals that the average job completion time of the scale-out scenario was 7 percent 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. More detailed results about the dynamic resource allocations when running J1-J9 can be found in our preliminary study [11].

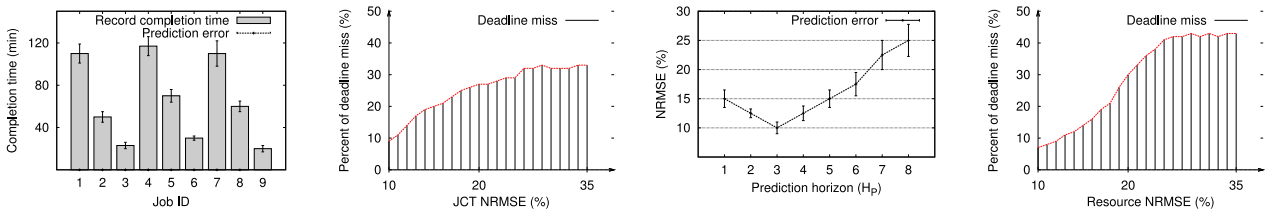
6.2 Impact of Hard and Soft Time Bounds

Fig. 9 shows that the effectiveness of RDS while setting hard and soft time bounds as deadlines in the experiment. The results demonstrate that hard deadline misses are much less than soft deadline misses for all workloads. This is due to the fact that the violating hard time bounds leads much higher penalty (as shown in the decay function δ) compared to violating of soft time bounds. The results in Figs 9a and 9b show that scale-out approach for *Wordcount* and *Grep* achieves slightly higher deadline misses than scale-up approach. In contrast, Fig. 9c shows that scale-up approach for *Terasort* leads higher deadline misses. The reason is that *Terasort* is I/O intensive workload, which can benefit from the scale-out deploy model. Furthermore, we use SQL query applications by Hive on Hadoop to query the E-commerce data set which records orders and order-product information for an E-commerce website [25]. Fig. 9d demonstrates that RDS scheduler is effective to reduce the deadline misses when running time sensitive Hive workloads on dynamic Hadoop clusters. It shows that scale-out approach for *Hive* leads higher deadline misses than scale-up does. The result also shows that *Hive*'s soft deadline misses are much lower than these traditional MapReduce workloads (*Wordcount*, *Grep* and *Terasort*). This is due to the fact that *Hive* workloads are more time sensitive and more flexible to obtain the resource.

RDS provides a fast path for small jobs with deadlines earlier than the next control interval. We evaluate the effectiveness of this mechanism when handling small jobs. In our preliminary study [11], we compared the performance of RDS with/without the fast path. The result shows that RDS with fast path significantly reduced small jobs' deadline misses. Without the fast path, deadline misses reduced as the job size increased. Our preliminary result demonstrates that the job completion time achieved by the fast path RDS was 50 percent less.

6.3 Accuracy of the Modeling

To evaluate the accuracy of the fuzzy models, we compare the error between the predicted job completion times and



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

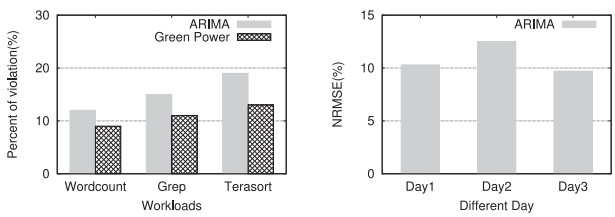
Fig. 10. The prediction accuracy and error impact of RHC control.

the actual completion times. The accuracy is measured by the normalized root mean square error (NRMSE), a standard metric for deviation [15]. Fig. 10a shows that the prediction was quite accurate, with on average 9.6 percent NRMSE. To study how the prediction error affect the working of RHC, we injected errors into the prediction. Fig. 10b shows that the deadline misses increased from 10.5 to 33 percent as the prediction error increased from 10 to 35 percent.

Figs. 10c and 10d plot the error of resource prediction and its impact on the deadline misses. Fig. 10c 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. Fig. 10d 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.

6.4 Impact by Prediction Approaches

Fig. 11a compares the effectiveness of two different available resource prediction approaches, i.e., ARIMA and green power based prediction. The result shows that green power based prediction achieves better effect (less deadline violations) in our experiment. This is because our resource availability trace is generated based on the renewable power trace from National Renewable Energy Laboratory [21]. *terasort* workload with green power based prediction reduces 6 percent of deadline violation, which means the SLO of I/O intensive workload relies more on the accuracy of the dynamic cluster resource prediction. For other scenarios, ARIMA may achieve better performance. For example, we evaluate the accuracy of ARIMA approach by the normalized root mean square error (NRMSE) based on Google's cluster trace. Fig. 11b shows the NRMSE is between 9.5 and 13 percent in a three-day scenario. It demonstrates ARIMA can achieve the similar accuracy with the green power prediction approach. Previously Fig. 11a shows green power prediction is better, but given the results in Fig. 11b, this conclusion is not true for Googles trace. Thus, ARIMA is applicable and effective for predicting the available resources of those regular dynamic clusters.



(a) Different prediction impact on renewable power trace. (b) ARIMA prediction accuracy on Googles cluster trace.

Fig. 11. The impact by prediction approaches.

6.5 Overhead and Scalability

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 4 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. Fig. 13 shows that the overhead costs for map and reduce tasks of wordcount application with different configurations. It demonstrates that larger task size leads to higher control overheads for both map and reduce tasks. The results also demonstrate that individual tasks execution time depends on the two factors, i.e., block size and the number of reduce task. For map task, the block size usually determines the average task execution time. The longer average task execution time leads larger overhead when RDS adjusts the map slot allocation. This is due to the fact RDS do not preempt any slots have been assigned to jobs. For large tasks, RDS will wait more time until the occupied slot released. 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. We have further conducted an experiment to evaluate the scalability of RDS scheduler. We scaled out the cluster resource from 20 vCPUs to 320 vCPUs in our testbed. Fig. 12 shows that the overhead becomes stable with the increasing number of tasks, and it scales from 8.3s (40 tasks/20 vCPUs) to 14.1s (640 tasks/320 vCPUs) in the experiment. For a very large cluster, multiple factors could become the bottlenecks and a further study is needed.

6.6 Discussions

Job Progress Estimation. We currently assume a uniform deadline on job progress for each control interval, i.e., 10 minutes. The per-interval deadline is calculated as the ratio of the overall deadline and the number of control intervals. Since the job progress is not uniform due to different execution phases (i.e., map and reduce phases) or data skew

TABLE 4
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

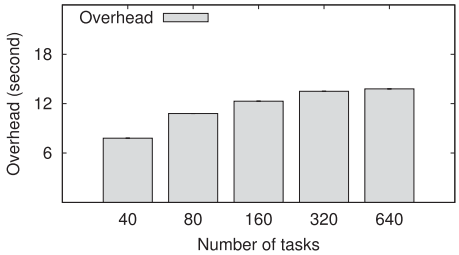
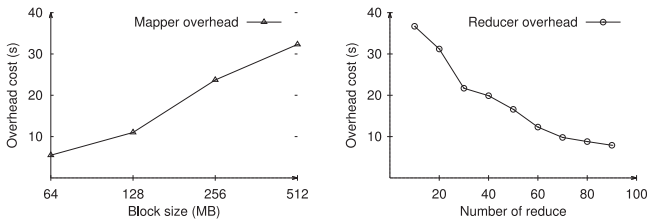


Fig. 12. Scalability of RDS scheduler.



(a) Map task overhead. (b) Reduce task overhead.

Fig. 13. The relationship between overhead cost and various configurations.

among tasks, assuming uniform per-interval deadlines possibly leaves room for improvement. For example, because the map phase usually executes faster the reduce phase, the uniform per-interval deadline may slow down the map phase with a low expectation. We plan to set different deadlines for the map and reduce phases and investigate how this change will affect job completion times.

Running Task Preemption. Another extension of our current work is to explore how preemption of current running tasks will affect deadline-aware Hadoop scheduling. Fig. 14 shows that the benefit from RDS scheduler deteriorates significantly as the size of individual tasks expending. Our experimental result reveals that RDS scheduler works well when the average task completion time is smaller than three minutes. But the performance of proposed scheduler decreases while the task size continuously grows. Thus, task preemption and killing strategy is recommended in such scenarios. Existing work has shown that killing or preempting a running task can possibly offer significant benefits but inevitably cause work loss for killed tasks [4], [26]. To kill or not to kill a running task becomes an even harder decision with variable resources.

Parameters Sensitivity. 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. 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. More results about the impact by control horizon setting and the different deadline misses as the workload priority varies can be found in our preliminary version of this paper [11].

Task Data Locality. Our current RDS scheduler has not considered the data locality issues when estimating individual tasks performance and scheduling Hadoop jobs. Existing studies have shown that there is inherent load imbalance among MapReduce tasks, skew that could come from uneven data distribution or non-uniform data processing cost creates stragglers [27], [28]. We will further explore

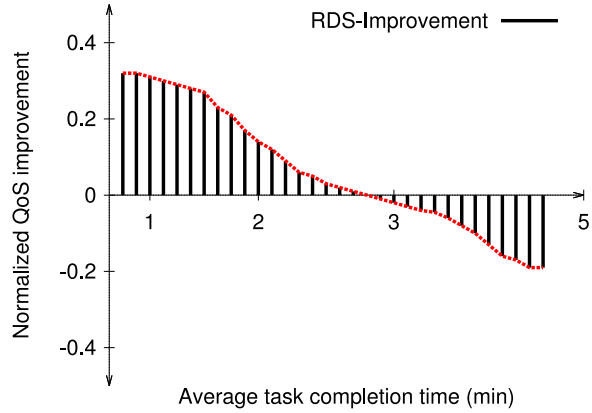


Fig. 14. RDS improvement and task sizes.

how to optimize task scheduling in dynamic environment while maximizing data locality across working sets, in an attempt to reduce network bottlenecks and increase overall system performance.

7 Related Work

Recently there are many existing studies have shown that MapReduce cluster size usually fluctuates for many reasons in a real system. Sharma et al. described MapReduce workload typically shared physical resources with other workloads, e.g., interactive workload [7]. Thus, the actual amount of resources available for MapReduce applications can vary over time. Lang et al. proposed AIS [29], 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. [5] 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.

As the environmental concerns and the energy consumption of datacenters continuously grow, developing green datacenters is becoming an increasingly important mission for major Internet service operators [22], [23]. These studies aim to achieve sustainable operation driven by green energy supply partially or completely from following aspects: (1) Studies [6], [22] focus on the energy demand side of a datacenter. (2) Studies [30], [31] focus on matching energy supply of a datacenter server cluster with its energy demand. (3) Studies [32], [33] focus on different energy storage approaches in the sustainable datacenters to improve their green energy usage efficiency. However, majority of the previous studies on the green energy management has focused on the whole datacenter rather than the specific big data application. Goiri et al. proposed GreenHadoop [6], a MapReduce framework for a datacenter powered by solar energy and using electrical grid as a backup. GreenHadoop dynamically adjusts the available cluster resources to maximize the green energy consumption by job scheduling.

Indeed, there are various techniques, e.g., resource provisioning [3], job scheduling [34] and self-tuning configuration

[35], which aim to optimize MapReduce performance. Dittrich et al. proposed Hadoop++ [36], a new index and join technique to improve runtime of MapReduce jobs. Rao et al. proposed a new MapReduce framework, Sailfish [27], to improve performance by aggregating intermediate data. Jinda et al. [37] 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.

Many existing scheduling techniques [4], [24] have demonstrated by prior studies that they can significantly improve MapReduce performance. 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. However, all these schedulers do not support different user level performance goals and are not dynamically adapted based on job progress. Recently, a few studies [3], [4] start to optimize the performance of MapReduce jobs with respect to their performance goals. Wolf et al. described FLEX [24], a flexible and intelligent allocation scheme for MapReduce workloads. It is proposed as an add-on module that worked synergistically with Fair Scheduler to provide performance guarantees. Our work differs from these efforts in that we consider a Hadoop cluster with fluctuating resource availability.

8 Conclusion

In this paper, we find there is a growing need for providing predictable services to users who have 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 Hadoop environment, especially a Hadoop cluster with dynamic resources. We further 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 percent compared to Fair scheduler and the EDF scheduler, respectively. In a Hadoop cluster running partially on renewable energy, the experimental result shows the green power based resource prediction approach can further reduce the penalty of deadline misses by 16 percent compared to ARIMA prediction approach.

Acknowledgments

Cheng's research was supported by the University of North Carolina at Charlotte. The authors are grateful to the associate editor and anonymous reviewers for their constructive comments.

References

[1] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale mapreduce workloads with significant interactive analysis," in *Proc. Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 43–56.

[2] Y. Guo, J. Rao, and X. Zhou, " iShuffle: Improving hadoop performance with shuffle-on-write," in *the Proc. 10th Int. Conf. Autonomic Comput.*, 2013, pp. 107–117.

[3] 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. IEEE Netw. Operations Manage. Symp.*, 2012, pp. 900–905.

[4] 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. 4th Annu. Symp. Cloud Comput.*, 2013, Art. no. 6.

[5] 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. ACM/IFIP/USENIX Int. Conf. Middleware*, 2011, pp. 187–207.

[6] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: Leveraging green energy in data-processing frameworks," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 57–70.

[7] B. Sharma, T. Wood, and C. R. Das, "HybridMR: A hierarchical MapReduce scheduler for hybrid data centers," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 102–111.

[8] eBay, Using Spark to Ignite Data Analytics, 2014. [Online]. Available: http://www.ebaytechblog.com/2014/05/28/using-spark-to-ignite-data-analytics/U-qUSPldUbw

[9] Yahoo!, Let Spark Fly: Advantages and Use Cases for Spark on Hadoop, 2014. [Online]. Available: https://spark-summit.org/2014/

[10] 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. 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Modeling Comput. Syst.*, 2012, pp. 175–186.

[11] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Resource and deadline-aware job scheduling in dynamic hadoop clusters," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 956–965.

[12] 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. ACM 3rd ACM Symp. Cloud Comput.*, 2012, Art. no. 7.

[13] PUMA, Purdue Mapreduce Benchmark Suite, 2012. [Online]. Available: https://engineering.purdue.edu/ puma/datasets.htm

[14] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Berlin, Germany: Springer, 2008.

[15] P. Lama and X. Zhou, "NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2012, pp. 1–12.

[16] Y. Chen, B. Yang, A. Abraham, and L. Peng, "Automatic design of hierarchical takagi sugeno type fuzzy systems using evolutionary algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 3, pp. 385–397, Jun. 2007.

[17] K. Astrom and B. Vittenmark, *Adaptive Control*, Englewood Cliffs, NJ, USA: Prentice Hall, 1995.

[18] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis, Forecasting, and Control*, 3rd ed., Englewood Cliffs, NJ, USA: Prentice-Hall, 1994.

[19] Q. Zhang, F. Mohamed, S. Zhang, Q. Zhu, B. Raouf, and L. Joseph, "Dynamic energy-aware capacity provisioning for cloud computing environments," in *Proc. ACM 9th Int. Conf. Autonomic Comput.*, 2012, pp. 145–154.

[20] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy, "Cloudy computing: Leveraging weather forecasts in energy harvesting sensor systems," in *Proc. IEEE 7th Annu. Commun. Society Conf. Sensor Mesh Ad Hoc Commun. Netw.*, 2010, pp. 1–9.

[21] NREL, "Measurement Data Center," 2012. [Online]. Available: http://www.nrel.gov/midc/

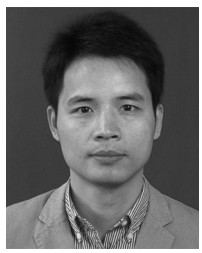
[22] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers," in *Proc. USENIX Workshop Power Aware Comput. Syst.*, 2011, Art. no. 5.

[23] N. Deng, C. Stewart, D. Gmach, M. Arlitt, and J. Kelley, "Adaptive green hosting," in *Proc. IEEE 9th Int. Conf. Autonomic Comput.*, 2012, pp. 135–144.

[24] 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. ACM/IFIP/USENIX Int. Conf. Middleware*, 2010, pp. 1–20.
[25] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive a petabyte scale data warehouse using hadoop," in *IEEE Intl Conf. Data Eng.*, 2010, pp. 996–1005.
[26] Y. Wang, J. Tan, W. Yu, L. Zhang, X. Meng, and X. Li, "Preemptive reduce task scheduling for fair and fast job completion," in *Proc. IEEE 10th Int. Conf. Autonomic Comput.*, 2013, pp. 279–289.
[27] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves, "Sailfish: A framework for large scale data processing," in *Proc. ACM 3rd ACM Symp. Cloud Comput.*, 2012, Art. no. 4.
[28] A. Verma, L. Cherkasova, and R. Campbell, "ARIA: Automatic resource inference and allocation for mapreduce environments," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2011, pp. 235–244.
[29] W. Lang and J. M. Patel, "Energy management for mapreduce clusters," *Proc. VLDB Endowment*, vol. 3, pp. 129–139, 2010.
[30] D. Gmach, J. Rolia, C. Bash, Y. Chen, T. Christian, A. Shah, R. Sharma, and Z. Wang, "Capacity planning and power management to exploit sustainable energy," in *Proc. IEEE Int. Conf. Netw. Service Manag.*, 2010, pp. 96–103.
[31] C. Li, R. Zhou, and T. Li, "Enabling distributed generation powered sustainable high-performance data center," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2013, pp. 35–46.
[32] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: Practical power-proportionality for data center storage," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 169–182.
[33] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. K. Fathy, "Energy storage in datacenters: What, where, and how much?" in *Proc. ACM 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Modeling Comput. Syst.*, 2012, pp. 187–198.
[34] 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. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, 2014, pp. 93–103.
[35] D. Cheng, J. Rao, Y. Guo, and X. Zhou, "Improving mapreduce performance in heterogeneous environments with adaptive task tuning," in *Proc. ACM/IFIP/USENIX 15th Int. Middleware Conf.*, 2014, pp. 97–108.
[36] J. Dittrich, J.-A. Quiane -Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing)," *Proc. VLDB Endowment*, vol. 3, pp. 515–529, 2010.
[37] A. Jinda, J. Quiané-Ruiz, and J. Dittrich, "Trojan data layouts: Right shoes for a running elephant," in *Proc. ACM 2nd Symp. Cloud Comput.*, 2011, Art. no. 21.

Dazhao Cheng received the BS degree in electronic engineering from the Hefei University of Technology, in 2006, the MS degree in electronic engineering from the University of Science and Technology of China, in 2009, and the PhD degree from the University of Colorado, Colorado Springs, in 2016. He is currently an assistant professor in the Department of Computer Science University of North Carolina, Charlotte. His research interests include cloud computing and Big Data processing. He is a member of the IEEE.

Xiaobo Zhou received the BS, MS, and PhD degrees in computer science from Nanjing University, in 1994, 1997, and 2000, respectively. Currently he is a professor of the Department of Computer Science, University of Colorado, Colorado Springs. His research lies broadly in computer network systems, specifically, Cloud computing and datacenters, BigData parallel and distributed processing, autonomic and sustainable computing, scalable Internet services and architectures. He was a recipient of the NSF CAREER Award in 2009. He is a senior member of the IEEE.



Yinggen Xu received the BS degree in computer science from Jiangxi Normal University, in 2010, the MS degree in computer science from Nanchang University, in 2013. Currently, he is working toward the PhD degree in computer science at Tongji University. His research interest includes resource management in data-parallel clusters.

Liu Liu received the BS degree in electronic engineering from the Chengdu University of Information Technology, in 2013, and the MS degree in electronic engineering from the HeFei University of Technology, in 2017. He is working toward the PhD degree in the Department of Computer Science, Tongji University. His research focuses on data parallel processing in Clouds.

Changjun Jiang received the PhD degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995. Currently he is a professor with the Department of Computer Science, Tongji University, Shanghai. He is also the director of professional Committee of Petri Net of China Computer Federation and the vice director of professional Committee of Management Systems of China Automation Federation. His current areas of research are concurrent theory, Petri net, and intelligent transportation systems. He is a member of the IEEE.