Optimal Job Scheduling With Resource Packing for Heterogeneous Servers

Huanle Xu, *Member, IEEE*, Yang Liu, *Student Member, IEEE*, and Wing Cheong Lau, *Senior Member, IEEE, Member, ACM*

***Abstract*— Jobs in modern computing clusters have highly diverse processing duration and heterogeneous resource requirements. In this paper, we consider the problem of job scheduling for a computing cluster comprised of multiple servers with heterogeneous computation resources, while taking the different resource demands of the jobs into account. Our focus is to achieve a low overall job response time for the system (which is also referred to as the job flowtime) while providing fairness between small and large jobs. Since the job flowtime minimization problem under multiple (even homogeneous) servers are known to be NP-hard, we propose an approximation algorithm to tackle the original online scheduling problem by adopting the recently-proposed notion of *fractional job flowtime* as a surrogate objective for minimization. For the general online job arrival case with multi-dimensional resource requirements, we apply Online Convex Optimization (OCO) techniques to design the corresponding scheduling algorithm with performance guarantees. In the single-dimensional resource setting, we show that the *dynamic fit* of the online version of our approximate algorithm grows only sublinearly with respect to time and derive a bound for its *dynamic regret* when comparing to its offline counterpart. While the baseline version of our proposed scheduling algorithm assumes the possibilities of job preemption and job migration across different servers, we show that the extent of job preemption and migration can be well controlled by augmenting the objective function with the corresponding switching costs.**

***Index Terms*— Online job scheduling, resource packing, online convex optimization, dynamic regret, switching cost.**

I. Introduction

THE job profiles in today's computing clusters are becoming increasingly diverse as small, but latency-sensitive jobs coexist with large batch-processing applications that may take hours to weeks to complete [2]. Jobs usually have highly diverse resource requirements such as the number of CPU cores they need and/or amount of memory. To satisfy these job demands, a modern computing cluster often scales out to hundreds or even thousands of servers. At the same time, every commodity server within a cluster is equipped with increasingly larger amount of resources. As such, a scheduler usually needs to assign (i.e. pack) multiple jobs to each server simultaneously in order to fully utilize the computation resource of the cluster. Given the potentially heterogeneous nature of servers within a cluster and the diversity of processing time as well as resource requirements of its jobs, there is a critical need of effective job scheduling schemes for large-scale computing clusters. Towards this end, various schedulers, e.g. [3]–[9], have been implemented and deployed by the industry. However, these schedulers are mostly designed based on heuristic arguments in order to achieve high scalability. Relatively little modeling or analytical effort is taken to characterize and optimize the performance trade-offs of the resultant design in a systematic, theoretically vigorous manner. On the other hand, many schedulers have also been proposed by the academic research community, e.g., [10]–[17]. However, these schemes assume each server can only hold one job at any time. They neither pack multiple jobs into a server nor consider all the relevant resources demanded by the jobs. As such, they can easily result in fragmentation and over-allocation of resources [10].

The shortcomings of the existing design process motivate us to take an alternate, analytical approach of building cluster schedulers based on an optimization-based framework which, on one hand, can tackle the challenges of resource packing among heterogeneous servers and, on the other hand, amenable to practical implementation for large-scale clusters. Our focus is to achieve a low overall job response time for the system (which is also referred to as the job flowtime) while providing fairness between jobs of different sizes and diverse resource requirements. To achieve this, we build a simplified model for online job scheduling of computing clusters consisting of multiple servers with different resource capacities. Under our model, jobs arrive at the cluster over time with known resource requirements (in terms of no. of CPU cores, amount of Memory and processing time). At the beginning of each timeslot, the scheduler needs to decide which job(s) to schedule, together with their server assignments subject to server capacity constraints, without knowing future job arrivals. In addition, the scheduler can preempt a running job and later resume its execution on the same or other available machine(s) while minimizing the sum of the l_k norm of job flowtime vector.

Manuscript received September 16, 2019; revised June 4, 2020 and November 11, 2020; accepted March 5, 2021. The research was supported in part by the CUHK Direct Grant #4055108, the CUHK MobiTeC R&D Fund and in part by the National Natural Science Foundation of China under Grant Number 61802060. This article was approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Kalyvianaki. Part of this work has been presented in IEEE Infocom 2019. Date of publication April 1, 2021; date of current version August 18, 2021. *(Huanle Xu and Yang Liu are co-first authors.) (Corresponding author: Huanle Xu.)*

Huanle Xu is with the Department of Computer and Information Science, University of Macau, Taipa, Macau (e-mail: xhlcuhk@gmail.com).

Yang Liu and Wing Cheong Lau are with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: ly016@ie.cuhk.edu.hk; wclau@ie.cuhk.edu.hk).

Digital Object Identifier 10.1109/TNET.2021.3068201

The above model yields a combinatorial optimization problem, which is NP-hard, even for the offline scenario (in which the arrival time of all jobs are known in advance). To make the problem tractable, we use fractional job flowtime [18] to approximate the job flowtime, and take the sum of fractional job flowtime as our surrogate optimization objective. On one hand, this approximated formulation guarantees that the resultant overall job flowtime is within a constant factor of the original optimal solution. More importantly, such approximation enables us to formulate the corresponding offline and online scheduling problems under a tractable convex optimization framework. In particular, we adopt online convex optimization (OCO) techniques to solve the problem in each time-slot in an efficient and scalable manner.

There is also a well-known negative result [19], [20] showing that, it is not possible to achieve a constant competitive ratio for any online job scheduling algorithm in the multi-server setting, even the resource demand of each job is single-dimensional. As such, we also extend the definitions of dynamic regret and dynamic fit in recent OCO literature [21] to quantify the competitive performance of our proposed online scheduling algorithm. Here, we define a modified version of dynamic regret to measure the difference of the cost between our scheme and that of the optimal solution for the problem which may involve *short-term violation* of the job completion constraint. The notion of dynamic fit was first introduced to track the cumulation of short-term constraint violation by some prior OCO literature, e.g. [22], [23]. In this work, we use dynamic fit to measure the total amount of queued workload that has not been completed by the cluster so far. Based on these two metrics, we show that, our scheme can achieve a $O(T^{\beta})$ fit with a small regret over a duration of T where $0 \leq \beta < 1$. Moreover, for the transient scheduling case where all jobs arrive the cluster at the same time, we show that the smallest-volume-first scheme achieves a constant competitive ratio of two for minimizing the sum of the exact job flowtime. Under this scheme, priority is given to jobs with small volume where the volume is defined as the product of the processing time and the CPU demand of a job.

In general, the above online scheduling scheme may result in job preemptions and migrations across different servers. In practice, we should control and limit the number of job preemption/ migration events due to their overheads. To reach this goal, we extend the original formulation by augmenting its job flowtime minimization objective with the switching cost associated with job preemptions and migrations. In particular, we define the switching cost as the square of the difference between the values of the scheduling decision variables for every pair of consecutive time-slots. With this extension, we only need to change the objective function in the original formulation of OCORP + M to support controllable job preemption and migration. Towards this end, we also extend our optimization framework to include MapReduce-like jobs where each job may consist of multiple small tasks with precedence constrains.

To summarize, we have made the following technical contributions in this paper:

- In Section II, we present a new online optimization framework to design and analyze job scheduling problems with multi-dimensional resource packing on multiple heterogeneous servers . We apply the OCO framework to design OCORP + M with a low complexity in Section III.
- For the setting of single-dimensional resource requirement, we characterize the competitive performance of OCORP + M based on an extended definition of dynamic regret and dynamic fit in Section IV. Moreover, we also limit the number of job preemption/ migration events under OCORP + M and design a transient scheduling algorithm which can achieve a constant competitive ratio. In addition, we discuss in Section V how to include MapReduce jobs in our scheduling framework. Besides the cluster scheduling problem, one can also leverage our techniques to study other general online convex optimization problems which have both long-term and short-term constraints.
- Before reviewing the related work in Section VII and making conclusions in Section VIII, we conduct trace-driven simulations in Section VI to demonstrate that OCOPR+M can reduce the total job flowtime by nearly 34% and achieve better fairness comparing to existing scheduling schemes. In addition, we also show that OCORP+ can limit/control the number of migrations and preemptions effectively with little impact ($<5\%$ increase) in the total job flowtime.

II. System Model and Problem Formulation

A. System Model

Consider a computing cluster which consists of M machines (i.e. servers) indexed from 1 to M. Machine i has C_i CPU cores and a Memory of M_i GB, and can process multiple jobs simultaneously as long as the total demand of these jobs does not exceed its capacity. Time is slotted and job j arrives the cluster at time a_j where the job arrival process, $(a_1, a_2, \ldots, a_N)$, is an arbitrary deterministic time sequence. Upon arrival, job j joins a global queue managed by a scheduler, waiting to be assigned to available machine(s) for execution. The scheduler makes scheduling decisions, i.e., which job(s) to schedule, together with their server assignment at the beginning of each time-slot. In addition, job j requires d_j CPU cores and a minimum memory size of m_j to run. When the demand of both CPU cores and memory are fully satisfied, job j needs p_j units of time to complete. All d_j, m_j and p_j are integers, which are declared to the scheduler when job j arrives at the cluster. In this paper, we assume preemption and migration are allowed, i.e., a running job can be check-pointed, preempted and resumed later on the same or a different machine.

We define the quantity $v_j = p_j d_j$ to be the processing volume of job j. Let $\mathbb{N}$ be the set of non-negative integers and $u_j^i(t) \in \mathbb{N}$ be the number of CPU cores allocated to job j on machine i in time-slot t. We consider that a job can be processed on a machine even if it is allocated fewer than d_j CPU cores (e.g. via time-division-multiplexing of CPU

cores). In this case, however, the job needs to take more than p_j time-slots to complete. In addition, we introduce another variable, i.e., the Memory allocation rate $z_j^i(t) \in \{0,1\}$ to indicate whether job j is scheduled on machine i with memory requirement satisfied in time slot t. A job completes when its processing volume is fully served, thus, the job completion time, c_j satisfies:

$$\sum_{t=a_j+1}^{c_j} \sum_{i=1}^{M} z_j^i(t) \cdot u_j^i(t) \geq d_j p_j, \tag{1}$$

and the job flowtime is denoted by $f_j = c_j - a_j$. Moreover, our model assumes job j will not be able to take advantage of any extra number of cores beyond d_j even it is allocated with such additional resources, i.e.,

$$\sum_{i=1}^{M} u_j^i(t) \leq d_j. \tag{2}$$

For ease of presentation and analysis, we shall make use of the notion of CPU allocation rate in the sequel. In other words, let $x_j^i(t) = u_j^i(t)/d_j$ be the CPU allocation rate to job j on machine i, then, Eq. (1) can be reformulated as:

$$\sum_{t=a_j+1}^{c_j} \sum_{i=1}^{M} x_j^i(t) \cdot z_j^i(t) \geq p_j. \tag{3}$$

B. Problem Formulation

In this paper, we focus on finding a feasible online schedule, which strikes a balance between fairness and job latency. As such, we adopt the l_k norm of job flowtime, i.e., f_j^k as the performance metric to optimize.

Define $\boldsymbol{x}_j(t) = \{x_j^1(t), x_j^2(t), \ldots, x_j^M(t)\}$ and $\boldsymbol{x}(t) = \{\boldsymbol{x}_1(t), \boldsymbol{x}_2(t), \ldots, \boldsymbol{x}_N(t)\}$ as the vector of CPU scheduling decision variables. Correspondingly, define $\boldsymbol{z}_j(t) = \{z_j^1(t), z_j^2(t), \ldots, z_j^M(t)\}$ and $\boldsymbol{z}(t) = \{\boldsymbol{z}_1(t), \boldsymbol{z}_2(t), \ldots, \boldsymbol{z}_N(t)\}$ as the vector of Memory scheduling decision variables. Our objective is to minimize the sum of f_j^k by determining optimal settings for $\{\boldsymbol{x}(t)\}$ and $\{\boldsymbol{z}(t)\}$. This results in the following optimization problem P1 :

$$\min_{\{\boldsymbol{x}(t),\boldsymbol{z}(t)\}} \quad \sum_{j=1}^{N} (c_j - a_j)^k \tag{4a}$$

$$s.t. \quad \sum_{j:t>a_j} d_j \cdot x_j^i(t) \leq C_i, \quad \forall i,t, \tag{4b}$$

$$\sum_{j:t>a_j} m_j \cdot z_j^i(t) \leq M_i, \quad \forall i,t, \tag{4c}$$

$$\sum_{i} x_j^i(t) \leq 1, \quad \forall j,t, \tag{4d}$$

$$\sum_{t=a_j+1}^{c_j} \sum_{i=1}^{M} x_j^i(t) \cdot z_j^i(t) \geq p_j, \quad \forall j, \tag{4e}$$

$$u_j^i(t) = d_j x_j^i(t) \in \mathbb{N}, \quad z_j^i \in \{0,1\}, \ \forall j,i,t. \tag{4f}$$

Constraint Eq. (4b) and Eq. (4c) states that, the total number of allocated CPU cores and the amount of allocated Memory on Machine i at time t cannot exceed its capacity C_i and M_i respectively. Constraint Eq. (4d) is due to Eq.(2). Constraint Eq. (4f) states that the number of CPU cores allocated to job j on machine i at any time must be a non-negative integer.

Remark 1: For the job requires a large number of CPU cores to process, the CPU-core demand may be split into multiple portions (i.e., $x_j^i(t)$ is a fractional number) and each portion is assigned to one machine. However, we show in Google traces that the number of CPU cores required by a job is much smaller comparing to the server capacity. In this case, the resulted number of fractional CPU rates is usually very small.

TABLE I
CPU ALLOCATION RATES UNDER THE FAIR SCHEDULER

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
Job 1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0
Job 2	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	0
Job 3	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	1

C. Fractional Job Flowtime as an Approximation

When $d_j = 1$ for all j and $C_i = 1$ for all i. P1 becomes the traditional job scheduling problem on multiple identical servers, which is NP-hard even when there are only two machines in the cluster [24]. To tackle P1, we adopt the same method introduced in [18] by approximating f_j^k with its fractional job flowtime counterpart, which is defined as the time weighted sum of all the fractional job allocation rates, i.e.,

$$f_j^k \simeq \sum_{t=a_j+1}^{\infty} \sum_{i=1}^{M} \Big(\frac{(t-a_j)^k}{p_j} + p_j^{k-1} \Big) x_j^i(t) \cdot z_j^i(t), \tag{5}$$

where $\left(\frac{(t-a_j)^k}{p_j} + p_j^{k-1} \right)$ is treated as the weight of the fraction rate $x_j^i(t) \cdot z_j^i(t)$ and it increases over time. With this approximation, a job needs to be completed as soon as possible, so as to yield a small flowtime. In addition, we show in the following example that, the fractional job flowtime can well approximate the actual job flowtime.

Example 1: To better illustrate the use of fractional job flowtime, we construct a simple example here. There are three jobs in the cluster which arrive to the cluster at time zero. Job i takes i units of time to process and its normalized resource demand is 0.9 ($i = 1,2,3$) for CPU. The memory demand of each job is small enough comparing to the machine capacity and therefore, we ignore it. We consider the Fair Scheduler under which resource is shared between the active jobs. In this case, the CPU allocation rate is given in Table I.

As such, the factional flowtime of Job 1 is $(1+2+3)\cdot\frac{1}{3}+1 = 3$. *The fractional job flowtime of Job 2 is* $\frac{1+2+3}{2} \cdot \frac{1}{3} + \frac{4+5}{2} \cdot \frac{1}{2} + 2 = 5.25$ *and Job 3 is* $\frac{1+2+3}{3} \cdot \frac{1}{3} + \frac{4+5}{3} \cdot \frac{1}{2} + \frac{6}{3} + 2 = 7\frac{1}{6}$. *By contrast, the actual flowtime of these three jobs are 3, 5, 6 respectively, thus, the approximation ratio of fractional flowtime is* 1.1.

We call the optimization problem using the fractional job flowtime in its objective (per Eq. (5)) as P2, which can be

formulated as:

$$\min_{\{\boldsymbol{x}(t),\boldsymbol{z}(t)\}} \sum_{t=a_j+1}^{\infty} \sum_{j:t>a_j} \sum_{i=1}^{M} \Big(\frac{(t-a_j)^k}{p_j} + p_j^{k-1}\Big) x_j^i(t) \cdot z_j^i(t)$$
$$\text{s.t.} \quad \text{Constraints } Eq.\ (4b) - (4f) \text{ are satisfied.}$$

Denote by OPT_{P1}^*, OPT_{P2}^* the optimal objective values achieved by P1 and P2 respectively. By using the same argument as that of [18], one can readily show that:

$$OPT_{P2}^* \leq 2 \cdot OPT_{P1}^*. \tag{6}$$

More importantly, by adopting the sum of fractional job flowtime as the surrogate minimization objective, P2 becomes a tractable, online optimization problem which we will analyze in the sequel.

III. Online Algorithm Design and Performance Analysis

In this section, we adopt the OCO framework to tackle P2 and consider a horizon of T time-slots. Since P2 includes both long-term and short-term constraints, we shall first change the long-term constraint in P2, i.e., Constraint Eq. (4e) to fit the OCO framework as follows:

$$g_j\big(\boldsymbol{x_j}(t), \boldsymbol{z_j}(t)\big) = \frac{p_j}{\Gamma - a_j} - \sum_{i=1}^{M} x_j^i(t) z_j^i(t) \leq 0, \tag{7}$$

where $\boldsymbol{x_j}(t) = \big\{x_j^1(t), x_j^2(t), \ldots, x_j^M(t)\big\}$ is the CPU rate vector of job j, $\boldsymbol{z_j}(t) = \big\{z_j^1(t), z_j^2(t), \ldots, z_j^M(t)\big\}$ is the Memory rate vector and, Γ is an algorithm parameter which control the amount of unfinished work for each job over certain time-scale. The value of Γ shall be determined in the following sections. Here, Eq. (7) is treated as a soft constraint, which can be violated during some time-slots, i.e. in the short-term.

Let $\boldsymbol{x}(t) = \big\{\boldsymbol{x_1}(t), \boldsymbol{x_2}(t), \ldots, \boldsymbol{x_{n(t)}}(t)\big\}$ and $\boldsymbol{z}(t) = \big\{\boldsymbol{z_1}(t), \boldsymbol{z_2}(t), \ldots, \boldsymbol{z_{n(t)}}(t)\big\}$. We further define the decision set of $\boldsymbol{x}(t)$ and $\boldsymbol{z}(t)$, i.e., $X(t)$ and $Z(t)$ as follows:

$$X(t) = \Big\{\boldsymbol{x}(t) : \sum_i x_j^i(t) \leq 1 \text{ with (4b) satisfied}, d_j x_j^i(t) \in \mathbb{N}\Big\}.$$

$$Z(t) = \Big\{\boldsymbol{z}(t) : z_j^i(t) \in \{0,1\} \text{ with (4c) satisfied.}\Big\}.$$

It is noteworthy that, $X(t)$ ($Z(t)$) is a time-varying set, which is different from the conventional settings of OCO in existing literature, e.g. [21]–[23] where the decision set is time-invariant. With the use of Eq. (7) and $X(t)$, $Z(t)$, P2 becomes:

$$\min_{\boldsymbol{x_t} \in X(t), \boldsymbol{z_t} \in Z(t)} \sum_{t=1}^{T} \sum_{j:t \geq a_j+1} \Big(\frac{(t-a_j)^k}{p_j} + p_j^{k-1}\Big) \times \sum_{i=1}^{M} x_j^i(t) z_j^i(t)$$
$$s.t. \sum_{t=a_j+1}^{T} g_j\big(\boldsymbol{x_j}(t), \boldsymbol{z_j}(t)\big) \leq 0, \quad \forall j.$$

A. Online Dual Algorithm Design

In this section, we adopt the online dual approach to tackle the above optimization problem. Let $\lambda_j(t)$ be the multiplier associated with job j in time-slot t and $\boldsymbol{\lambda}(t) = \big\{\lambda_1(t), \lambda_2(t), \ldots, \lambda_N(t)\big\}$. Thus, the per-slot Lagrangian function can be expressed as:

$$L_t(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{\lambda}(t)) = \sum_{j:t>a_j} \Big(\frac{(t-a_j)^k}{p_j} + p_j^{k-1} - \lambda_j(t)\Big) \cdot \sum_{i=1}^{M} x_j^i(t) z_j^i(t). \tag{9}$$

The dual approach works as follows: Given the dual variable $\boldsymbol{\lambda}(t)$ in current time-slot, the CPU and Memory allocation rate, $\boldsymbol{x}(t)$ and $\boldsymbol{z}(t)$ are determined by:

$$\min_{\{\boldsymbol{x}(t),\boldsymbol{z}(t)\}} \quad L_t(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{\lambda}(t))$$
$$\text{s.t. } \boldsymbol{x}(t) \in X(t), \quad \boldsymbol{z}(t) \in Z(t), \tag{P3}$$

Furthermore, the dual variable updates take the form of:

$$\lambda_j(t+1) = \max\Big\{0\ ,\ \lambda_j(t) + \mu \cdot g_j(\boldsymbol{x_j}(t), \boldsymbol{z_j}(t))\Big\}, \tag{10}$$

where μ is the step-size to be designed in the sequel.

It is worth noting that, in P3, the Memory allocation rate $z_j^i(t)$ is a binary variable since each job can only be executed when the memory requirement is fully satisfied. As such, P3 becomes an integer programming problem. In addition, the two resource allocation variables $z_j^i(t)$ and $x_j^i(t)$ are multiplied together, which makes the objective function nonconvex and difficult to tackle. To decouple the multiplication of $z_j^i(t)$ and $x_j^i(t)$ in P3, we solve two separate optimization problems iteratively by fixing one variable in each problem. Specifically, we first fix $z_j^i(t) = 1$ for all i, j in P3 and obtain the optimal solutions to $\boldsymbol{x}(t)$, i.e., $\boldsymbol{x^1}(t)$. After that, we substitute $\boldsymbol{x^1}(t)$ into P3 and obtain the optimal solutions to $\boldsymbol{z}(t)$, i.e., $\boldsymbol{z^1}(t)$. We then repeat this procedure until convergence. This process yields the online scheduling algorithm called OCORP + M, which accounts for Online Convex Optimization based job scheduling with Resource Packing in multi-dimensional case. Algorithm 1 shows the pseudo-code of OCORP + M.

Remark 2: We adopt the online dual approach to update both the primal and dual variables since both the objective function and the short-term constraint are known before the scheduling decision is made for time-slot t. In contrast, for previous OCO works, e.g., [21]–[23], the knowledge of $f_t(\boldsymbol{x}(t), \boldsymbol{z}(t))$ and $g_j(\boldsymbol{x}(t), \boldsymbol{z}(t))$ is one time-slot behind the decision-making epoch. As such, those prior works have adopted the online saddle-point method, under which a first-order approximation of the Lagrangian function is made at $\boldsymbol{x} = \boldsymbol{x}(t-1)$.

B. Performance of OCORP + M

The time complexity and performance of OCORP + M can be characterized by the following theorems:

Theorem 1: The time complexity of OCORP + M is $O(n(t) \cdot \phi)$ where $\phi = \max_{i \in \{1,2,\cdots,M\}} M_i$ and $n(t)$ is the total number of unfinished jobs for a fixed time t.

Algorithm 1 Online Scheduling Algorithm in the Multi-dimensional Resource Setting

1 **for** $t = 1, 2, \cdots, T$ **do**
2 When job j arrives at a_j, initialize $p_j(a_j) = p_j$ and $\lambda_j(a_j + 1)$;
3 Initialize the unfinished job set in time t, i.e., $U(t) = \{1, 2, \cdots, n(t)\}$;
4 Initialize $\boldsymbol{z^0}(t) = \mathbf{1}$;
5 **while** $||(\boldsymbol{x^k}(t), \boldsymbol{z^k}(t)) - (\boldsymbol{x^{k+1}}(t), \boldsymbol{z^{k+1}}(t))|| > \epsilon$ **do**
6

$$\boldsymbol{x^{k+1}}(t) = \arg\min_{\boldsymbol{x}\in X(t)} L_t(\boldsymbol{x}, \boldsymbol{z^k}(t), \boldsymbol{\lambda}(t)). \quad (11)$$

$$\boldsymbol{z^{k+1}}(t) = \arg\min_{\boldsymbol{z}\in Z(t)} L_t(\boldsymbol{x^{k+1}}(t), \boldsymbol{z}(t), \boldsymbol{\lambda}(t)). \quad (12)$$

7 $\lambda_j(t+1) = \max\left\{0, \lambda_j(t) + \mu\left(\frac{p_j}{\Gamma - a_j} - \sum_{i=1}^{M} x_j^i(t) z_j^i(t)\right)\right\}$;
8 Assign job j to servers in the cluster based on $x_j^i(t)$;
9 $p_j(t+1) = p_j(t) - \sum_{i=1}^{M} x_j^i(t) \cdot z_j^i(t)$;
10 **if** $p_j(t+1) \le 0$ **then**
11 Label job j as finished;

Proof: It can be readily show that, the updates in Line 5 and 6 of Algorithm 1 shall converge within two iterations. On the one hand, the update of $\boldsymbol{x}(t)$ in Eq.(11) requires one to solve a linear optimization problem, which leads to a time complexity of $O(n(t))$. On the other hand, to update $\boldsymbol{z}(t)$, we need to solve M individual knapsack problems with each leading to a complexity of $O(n(t)M_i)$. Therefore, the total complexity of OCORP + M is $O(n(t)\cdot\phi)$. □

Theorem 2: OCORP + M achieves an approximation ratio of $\min_{i\in 1,2,\cdots,M} \frac{M_i - m_{max}}{C_i + 2d_{min}} \cdot \frac{d_{min}}{m_{max}}$ *for solving P3 where* $d_{min} = \min_{j\in 1,2,\cdots,N} d_j$ *and* $m_{max} = \max_{j\in 1,2,\cdots,N} m_j$.

Proof: Without loss of generality, we omit time t and use x_j^i (z_j^i) to represent $x_j^i(t)$ ($z_j^i(t)$). For ease of presentation, let $\pi_j = -\left(\frac{(t-a_j)^k}{p_j} + p_j^{k-1} - \lambda_j(t)\right)$, thus, P3 is equivalent to the following optimization problem:

$$\max_{\{\boldsymbol{x},\boldsymbol{z}\}} \sum_{j:t>a_j} \pi_j \sum_{i=1}^{M} x_j^i z_j^i$$
$$\text{s.t. } \boldsymbol{x} \in X(t), \quad \boldsymbol{z} \in Z(t).$$

Let $\{x_j^{i,*}\}_{i,j}, \{z_j^{i,*}\}_{i,j}$ denote the optimal solution to the above program. On the one hand, OCORP + M produces $\{x_j^{i,1}\}_{i,j}$ via fixing $z_j^i = 1$ for all i, j and following Eq. (22), thus, it follows that,

$$\sum_{j:t>a_j} \pi_j \sum_{i=1}^{M} x_j^{i,1} \ge \sum_{j:t>a_j} \pi_j \sum_{i=1}^{M} x_j^{i,*} = \Omega. \quad (13)$$

which implies:

$$\sum_{j:t>a_j} \pi_j \sum_{i=1}^{M} x_j^{i,*} \cdot z_j^{i,*} \le \Omega. \quad (14)$$

On the other hand, OCORP + M produces $\{z_j^{i,1}\}_{i,j}$ by fixing $x_j^i = x_j^{i,1}$ and solving the following knapsack problem on each machine i:

$$\max_{\boldsymbol{z}\in Z(t)} \sum_{j:t>a_j} \pi_j x_j^{i,1} \cdot z_j^i. \quad (15)$$

If $\sum_{j:t>a_j} m_j \mathbf{I}(x_j^{i,1} > 0) \le M_i$ where $\mathbf{I}(\cdot)$ is an indicator function, then $z_j^{i,1} = 1$ when $x_j^{i,1} > 0$, in this case, we have:

$$\sum_{j:t>a_j} \pi_j x_j^{i,1} = \sum_{j:t>a_j} \pi_j x_j^{i,1} \cdot z_j^{i,1}, \quad (16)$$

otherwise, we construct one feasible solution for z_j^i as follows. We sort jobs in $\{j : t > a_j\}$ based on the decreasing order of $\pi_j x_j^{i,1}$ such that $\pi_{j_1} x_{j_1}^{i,1} \ge \pi_{j_2} x_{j_2}^{i,1} \ge \pi_{j_2} x_{j_2}^{i,1} \cdots \ge \pi_{j_n} x_{j_n}^{i,1}$ where $n = \left|\left\{j : t > a_j \text{ and } x_j^{i,1} > 0\right\}\right|$. Then, we assign $z_{j_k}^i = 1$ if the following equation is satisfied:

$$\sum_{l=1}^{k} m_{j_l} \cdot z_{j_l}^{i,1} \le M_i. \quad (17)$$

Let ρ denote the largest index such that $z_{j_\rho}^i = 1$, then $\rho \ge \frac{M_i - m_{max}}{m_{max}}$. In addition, based on the packing policy, it follows that: $n \le \frac{C_i - 2}{d_{min}} + 2$. As such, we have:

$$\frac{\sum_{j:t>a_j} \pi_j x_j^{i,1} \cdot z_j^i}{\sum_{j:t>a_j} \pi_j x_j^{i,1}} \ge \frac{\rho}{n} \ge \frac{M_i - m_{max}}{C_i + 2d_{min}} \cdot \frac{d_{min}}{m_{max}}. \quad (18)$$

Since $\{z_j^{i,1}\}_{i,j}$ is an optimal solution to the optimization problem in Eq. (15), thus,

$$\sum_{j:t>a_j} \pi_j x_j^{i,1} \cdot z_j^{i,1} \ge \sum_{j:t>a_j} \pi_j x_j^{i,1} \cdot z_j^i, \quad (19)$$

which implies:

$$\sum_{j:t>a_j} \pi_j \sum_{i=1}^{M} x_j^{i,1} \cdot z_j^{i,1} \ge \min_{i\in 1,2,\cdots,M} \frac{M_i - m_{max}}{C_i + 2d_{min}} \cdot \frac{d_{min}}{m_{max}} \cdot \Omega. \quad (20)$$

Combine Eq.(14) and Eq. (20), the result immediately follows, this completes the proof of Theorem 2. □

Theorem 2 states that, OCORP + M manages to achieve a good approximation result for solving P3 in each individual time slot. However, it cannot provide a competitive performance guarantee in a long run for optimizing P2. In the following section, we shall provide a competitive analysis for OCORP + M in some special settings.

IV. Competitive Analysis in the Setting of Single-Dimensional Resource

In this section, we present a competitive analysis for OCORP + M in the setting where only CPU resource is a bottleneck. In this case, we set $z_j^i(t)$ to one for all i, j, t in P2. We shall first analyze the regret performance of OCORP + M in this setting and then show how to design a transient scheduling algorithm that can achieve a constant competitive ratio.

A. Regret Analysis in the Online Setting

In the online setting, we let $y_j(t) = \sum_i x_j^i(t)$ be the overall CPU rates allocated to job j at time t and $\boldsymbol{y}(t) = \{y_1(t), y_2(t), \ldots, y_n(t)\}$. We first show one can construct a feasible solution $\boldsymbol{x}(t)$ based on $\boldsymbol{y}(t)$, which can be attained with a low complexity. After that, we study the regret performance of OCORP + M in this single-dimensional resource setting.

Define $\omega_j(t) = \Big(\frac{(t-a_j)^k}{p_j} + p_j^{k-1} - \lambda_j(t)\Big)/d_j$ where $\lambda_j(t)$ is given by Eq. (10) and assume without loss of generality that the unfinished jobs are sorted such that:

$$\omega_1(t) \leq \omega_2(t) \leq \cdots \leq \omega_{n(t)}(t) \leq 0, \tag{21}$$

where $n(t)$ is the number of unfinished jobs with negative $\omega_j(t)$ in time t. The following lemma provides an efficient way to derive the optimal solution to P3.

Lemma 1: The optimal solution to P3 when $z_j^i(t) = 1$ for all i, j, t, $\boldsymbol{y}(t)$, is as follows:

$$y_j(t) = \begin{cases} 1 & \sum_{n=1}^{j} d_n \leq \sum_{i=1}^{M} C_j, \\ 0 & \sum_{n=1}^{j-1} d_n \geq \sum_{i=1}^{M} C_j, \\ \dfrac{\sum_{i=1}^{M} C_j - \sum_{n=1}^{j-1} d_n}{d_j} & \text{otherwise}. \end{cases} \tag{22}$$

Lemma 22 indicates that, the optimal overall CPU allocation rate, $\boldsymbol{y}(t)$ can be obtained with a complexity of $O(n(t))$. It remains to produce the CPU allocation rate on each machine (i.e., $\boldsymbol{x}(t)$) based on $\boldsymbol{y}(t)$ and the process is as follows. The scheduler selects the servers from the largest capacity to the smallest one to pack jobs. For each server, the scheduler chooses the unassigned jobs with largest CPU demands and packs them into the server, as long as the total demand of these jobs does not exceed the server capacity.

We proceed to characterize the competitive performance of OCORP + M based on a regret analysis in this single-dimensional resource case. In particular, we adopt a modified definition of dynamic regret as follows:

$$\text{Reg}_T^d := \sum_{t=1}^{T} f_t(\boldsymbol{y}(t)) - \sum_{t=1}^{T} f_t(\boldsymbol{y}^*(t)), \tag{23}$$

where $f_t(\boldsymbol{y}(t))$ is given by:

$$f_t(\boldsymbol{y}(t)) := \sum_{j:t \geq a_j+1} \Big(\frac{(t-a_j)^k}{p_j} + p_j^{k-1}\Big) y_j(t), \tag{24}$$

and $\boldsymbol{y}^*(t)$ is an optimal solution to the following optimization problem:

$$\boldsymbol{y}^*(t) = \arg \min_{\boldsymbol{x} \in X(t)} f_t(\boldsymbol{x}), \tag{25}$$

which does not necessarily satisfy the short-term constraint in Eq. (7). By contrast, the dynamic regret defined in [21] requires the benchmark to satisfy the short-term constraint in every time-slot. As such, our new definition of dynamic regret allows the benchmark to make a better performance and is therefore more valuable than that of [21].

To ensure feasibility of OCORP + M, we adopt the notion of dynamic fit, which is defined as:

$$\begin{aligned} \text{Fit}_T^d &= \sum_{t=1}^{T} \sum_{j:t \geq a_j+1} g_j(y_j(t)) \\ &= \sum_{t=1}^{T} \sum_{j:t \geq a_j+1} \Big(\frac{p_j}{\Gamma - a_j} - y_j(t)\Big) \\ &= \sum_{j=1}^{N} \Big(\frac{T - a_j}{\Gamma - a_j} p_j - \sum_{t:t \geq a_j+1}^{T} y_j(t)\Big). \end{aligned} \tag{26}$$

Thus, Fit_T^d gives an upper bound for the total amount of unfinished work for all jobs that have arrived the cluster by time $T \geq \Gamma$.

Theorem 3: When $z_j^i(t) = 1$ for all i, j, t, by setting Γ such that:

$$\Gamma = \max \left\{ \max_{j \in \{1,2,\cdots,N\}} \left\{ a_j + 2p_j, 2a_j \right\}, \ \frac{4\sum_{j=1}^{N} p_j d_j}{\sum_{i=1}^{M} C_i} \right\}, \tag{27}$$

and choosing $\mu = T^{k-\beta}$ where $0 \leq \beta < 1$, when $T \leq \Gamma$, the dynamic fit in Eq. (26) under OCORP + M is bounded by:

$$\textit{Fit}_T^d \leq O(T^{\beta}\sqrt{N_T}), \tag{28}$$

where N_T is the number of jobs that have entered into the cluster by time T.

Proof Sketch. We adopt the Lyapunov-drift approach for proving this theorem. In order to bound the dual variable, which can be interpreted as the queue length, we carefully construct a feasible primal solution such that the slater condition is satisfied. Moreover, we show that the total gap between the two successive dual variables in all time slots is bounded by a constant. By conducting these two key steps and incorporating the Lyapunov approach, we show the dual variable can be bounded by a small constant. With this result, we then show the dynamic fit is sublinear with respect to time T.

Note that $\frac{\sum_{j=1}^{N} p_j d_j}{\sum_{i=1}^{M} C_i}$ in Eq. (27) characterizes the minimum number of time-slots required to complete all jobs under any algorithm. As such, Γ can be viewed as a parameter to set an approximated completion time of all jobs. Since dynamic fit serves as a metric to measure the unfinished workload for all jobs arriving the cluster, Theorem 3 implies that most of the jobs would have completed by $T \geq \Gamma$.

Corollary 1: Within an interval of $\widetilde{T}$, each unfinished job will be processed at least once under OCORP + M where $\widetilde{T}$ is upper-bounded by:

$$\widetilde{T} \leq O(T^{\beta}). \tag{29}$$

Corollary 1 implies that OCORP + M provides some fairness guarantee between small and large jobs.

Theorem 4: Under the same setting as Theorem 3, the dynamic regret under OCORP + M is upper bounded by:

$$\textit{Reg}_T^d \leq \frac{C \cdot OPT^{1-\beta}}{2} \max_{j \in \{1,2,\cdots,N\}} \frac{p_j}{d_j}, \tag{30}$$

where OPT is the optimal objective value achieved by the original online scheduling problem.

Proof Sketch: Based on the optimality of $\boldsymbol{y}(t)$ and the fact that $g_j(y_j(t)) \leq \frac{p_j}{\Gamma - a_j}$, we first bound the regret performance in terms of T. We then characterize a lower bound for the optimal objective, OPT. With these two results, we can easily bound the regret with respect to OPT.

One implication of Theorem 3 and Theorem 4, when considering OCORP + M over a time-scale of Γ time-slots while choosing $\mu = O(\Gamma^k)$, the total amount of unfinished work is upper bounded by $O(\sqrt{N_T})$. At the same time, the cumulated regret is bounded by $O(\text{OPT})$. In other words, when the optimal offline algorithm completes all jobs, OCORP + M would have no more than $O(\sqrt{N_T})$ unfinished work while achieving a cost with the same order as that of the optimal offline algorithm.

As mentioned before, [18] also adopts the approximation of fractional job flowtime to analyze the performance of scheduling algorithms. However, this work only investigates the Fair scheduler under which the CPU resources are shared between all jobs in the cluster. Furthermore, [18] applies the primal-dual fitting framework to bound the competitive performance of Fair scheduler using a heavy resource augmentation [25]. And the derived performance bound is quite loose. In this paper, we extend the conventional OCO framework to design an efficient scheduling algorithm and more importantly, manages to show a small competitive ratio with a bounded violation of job requirements. As such, our theoretical findings substantially improve the previous results on online job scheduling with heterogeneous resource demands.

Our analytical framework provides a different angle to illustrate the regret performance of online algorithms and it also fills the gap between the regret performance and competitive performance. Traditional OCO methods usually define the regret as a function of the total rounds (time T) that have been played, e.g., [21], [26], [27], and a sublinear regret is desirable. For online job scheduling problems, such a sublinear regret does not make much sense as the number of jobs can be much smaller comparing to T. By contrast, our derived regret bound is sublinear with respect to the optimal objective only and therefore is much more meaningful.

B. Analysis for Job Scheduling With Switching Cost

In general, the job scheduling schemes designed in OCORP + M in Section III may result in job preemptions and migrations. In practice, we should limit the number of job preemptions and migrations due to their performance overheads. In this section, we formulate another optimization problem which minimizes (1) the l_k norm of the fractional job flowtime vector and (2) cost of job preemptions and migrations (which we will refer as the *switching cost* going forward).

We define the switching cost between the scheduling decisions (i.e. the job-server assignment variables) for two consecutive time-slots as follows:

$$S(t) = \sum_{j:t \geq a_j+2} \sum_{i=1}^{M} \left(x_j^i(t) - x_j^i(t-1) \right)^2. \tag{31}$$

The per-slot Lagrangian function then becomes:

$$L_t(\boldsymbol{x}(t), \boldsymbol{\lambda}(t)) = \sum_{j:t \geq a_j+1} \sum_{i=1}^{M} \left(\frac{(t-a_j)^k}{p_j} + p_j^{k-1} - \lambda_j(t) \right) \times x_j^i(t) + \alpha S(t), \tag{32}$$

where α is a parameter to control the trade-offs between job flowtime and switching cost.

Similar to Theorem 3 and Theorem 4, the competitive performance of OCORP + M with switching cost can be characterized by the following theorem:

Theorem 5: By setting the same Γ as that in Eq. (27) *and choosing $\mu = O(T^{k-\beta})$ where $0 \leq \beta < 1$, the dynamic fit in Eq.* (26) *under OCORP + M with switching cost is bounded by:*

$$Fit_T^d \leq O(T^{\beta}\sqrt{N_T}). \tag{33}$$

When $T \leq \Gamma$, the dynamic regret in Eq. (23) *is bounded by:*

$$Reg_T^d \leq O(OPT^{1-\frac{\beta}{k+1}}). \tag{34}$$

Proof Sketch: The technique is similar to the proof of Theorem 3 except that we need to construct a feasible solution $\tilde{x}_j^i(t)$ for $t \geq (a_j + 1)$ instead of $\tilde{\boldsymbol{y}}_j(t)$, i.e., $\tilde{x}_j^i(t) = \frac{C_i}{\sum_{i=1}^{M} C_i}\left(\frac{p_j}{\Gamma - a_j} + \epsilon\right)$.

C. Competitive-Ratio Based Analysis for Transient Scheduling

Though OCORP + M can yield a sublinear regret in the single-dimensional resource case, it fails to achieve a constant competitive ratio. Previous results even show that, no onine algorithm can achieve a bounded competitive ratio for minimizing the overall job flowtime on multiple identical machines [25]. We show, however, one can design a competitive algorithm in the transient scheduling case where all the jobs enter into the system at the same time with single-resource demand. In particular, we consider that $a_j = 0$ for all j and let $y_j(t) = \sum_i x_j^i(t)$ with $z_j^i(t) = 0$. In this case, the objective in P3 is equivalent to the following formula:

$$\min_{\{\boldsymbol{y}(t)\}} \sum_{t=1}^{L} t^k \sum_{j=1}^{N} \frac{y_j(t)}{p_j}, \tag{35}$$

where $L = \sum_{j=1}^{N} p_j$ is the largest number of time-slots it takes to complete all jobs. In addition, the optimal solution in the transient case, denoted by $\{\boldsymbol{y}^*(t)\}$, satisfies:

$$\sum_{j=1}^{N} \sum_{t=1}^{L} \frac{y_j^*(t)}{p_j} = N. \tag{36}$$

Applying the Rearrangement Inequality [28] to the objective in Eq. (35), we have:

$$\sum_{j=1}^{N} \frac{y_j^*(1)}{p_j} \geq \sum_{j=1}^{N} \frac{y_j^*(2)}{p_j} \geq \cdots \geq \sum_{j=1}^{N} \frac{y_j^*(L)}{p_j}. \tag{37}$$

In other words, the sequence $\left\{ \sum_{j=1} \frac{y_j^*(t)}{p_j} \right\}$ is non-increasing in time t. Such a property motivates us to design the transient

scheduling algorithm by solving the following local optimization problem:

$$\max_{\boldsymbol{y}(t)} \sum_{j=1}^{N} \frac{y_j(t)}{p_j} \tag{38a}$$

$$s.t. \sum_{j=1}^{N} d_j \cdot y_j(t) \le C, \tag{38b}$$

$$y_j(t) \le p_j(t), \tag{38c}$$

$$0 \le y_j(t) \le 1, \quad \forall j, \tag{38d}$$

where $C = \sum_{i=1}^{M} C_i$ and $p_j(t)$ is the remaining processing time for job j at time t. We assume without loss of generality that jobs have been ordered such that $p_1 d_1 \le p_2 d_2 \le \cdots \le p_N d_N$, then, the following lemma gives an optimal solution to this local optimization problem.

Lemma 2: Define $\gamma_j(t) = C - \sum_{n=1}^{j-1} d_n y_n^l(t)$, the optimal solution to Eq. (38)*, $\{y_j^l(t)\}$, is as follows:*

$$y_j^l(t) = \begin{cases} \max\left\{1, p_j(t)\right\} & \gamma_j(t) \ge d_j, \\ 0 & \gamma_j(t) \le 0, \\ \max\left\{\dfrac{\gamma_j(t)}{d_j}, p_j(t)\right\} & \textit{otherwise}, \end{cases} \tag{39}$$

and $d_j y_j^l(t)$ is an integer.

The local optimal solutions in Eq. (39) indicate that, giving scheduling priority to jobs with small volume, i.e., the product of job resource demand and processing time shall yield a good performance. Following this insight, we design a simple transient scheduling algorithm to allocate CPU rates as follows: At the beginning of each time-slot, the scheduler schedules the unfinished jobs with smallest volume and allocates CPU rates to them based on Eq.(39). The scheduler then assigns jobs with nonzero CPU rates to different servers following the same procedure as the online algorithm.

In the sequel, we shall show that the transient solution above is optimal in terms of minimizing the l_k norm of the fractional job flowtime vector via the following theorem:

Theorem 6: The transient scheduling scheme produces an optimal solution to P3.

Theorem 6 and Eq.(6) together immediately implies the following corollary:

Corollary 2: Let $Flow^s$ be the exact job flowtime achieved by the transient solution, we have:

$$Flow^s \le OPT_{P1}^* + \sum_{j=1}^{N} p_j/2 \le \frac{3}{2} \cdot OPT_{P1}^*.$$

Basically, Corollary 2 states that, the transient scheduling algorithm is $\frac{3}{2}$-competitive in terms of minimizing the overall exact job flowtime in the transient case.

Remark 3: Our designed transient algorithm basically implements the SVF scheme (smallest volume first) to compute the CPU allocation rate for all jobs in each time slot. While [29] also applies this policy to determine the scheduling order of jobs in the transient setting, it assigns a job to the earliest available machine. In contrast, we assign jobs to each machine based on the exact CPU allocation rate. By doing this, we can make the most use of CPU resources in the cluster. Furthermore, [29] can only achieve a competitive ratio of $\frac{3}{1-\alpha}$ in terms of minimizing the overall job flowtime, by making an assumption that the maximum job demand is no larger than α. As a comparison, our algorithm can yield a constant competitive ratio of $\frac{3}{2}$ without any assumption on the job resource demand and processing time.

V. Extensions to MapReduce Jobs

In this section, we show how to extend our job scheduling framework in Section II to include MapReduce jobs. In this setting, each job has two sequential phases, i.e., the map phase and the reduce phase. A map (reduce) phase further consists of multiple small map (reduce) tasks which can run in parallel. Each map task needs to get the input data from the Hadoop Distributed File Systems before its execution. Moreover, a reduce task can only begin its execution after all the map tasks within the same job complete, since it needs to fetch data from the output of all the map tasks through network or local disks.

For ease of presentation, throughout this section, we use $z \in \{m, r\}$ to denote the map- or reduce-related statements for all the tasks. When z is used, it is set to either m or r. To be specific, we consider job j consists of n_j^z tasks and each requires d_j^z CPU cores to run and p_j^z units of time to complete when its demand of d_j^z CPU cores are fully satisfied after its input data is available. Let $x_j^{k,z}(t)$ denote the CPU allocation rate for the kth task from the map (reduce) phase of job j. Paralleling Eq. (3), we have:

$$\sum_{t=a_j+1} x_j^{k,z}(t) \ge p_j^z, \quad \forall j, z \in \{m, r\} \text{ and } 1 \le k \le n_j^z. \tag{40}$$

Similarly, the capacity constraint becomes:

$$\sum_{j:t>a_j} \sum_{z\in\{m,r\}} \sum_{k=1}^{n_j^z} d_j^z \cdot x_j^{k,z}(t) \le \sum_{i=1}^{M} C_i, \quad \forall t, \tag{41}$$

and the CPU allocation rate satisfies:

$$0 \le x_j^{k,z}(t) \le 1 \text{ and } d_j^z \cdot x_j^{k,z}(t) \in \mathbb{N}, \quad \forall j, z, k, t. \tag{42}$$

Moreover, the dependency between the map phase and the reduce phase implies:

$$x_j^{l,r}(\tau) = 0, \quad \text{if } \sum_{t=a_j+1}^{\tau-1} \sum_{k=1}^{n_j^m} x_j^{k,m}(t) < n_j^m \cdot p_j^m, \ \forall j, l, \tau. \tag{43}$$

In this case, minimizing the overall l_k norm of fractional job flowtime shall yield the following optimization problem (P5):

$$\min_{\boldsymbol{x}} \sum_{j:t\ge a_j+1} \sum_{z\in\{m,r\}} \frac{2(t-a_j)^k + (p_j^m + p_j^r)^k}{2n_j^z \cdot p_j^z} \cdot \sum_{k=1}^{n_j^z} x_j^{k,z}(t)$$

s.t. Constraints(40), (41), (42) and (43) are satisfied.

We still apply the OCO techniques to tackle P5. To begin with, we transfer Eq. (40) to short-term constraints as follows:

$$g_j(x_j^{k,z}(t)) = \frac{n_j^m p_j^m + n_j^r p_j^r}{n_j^z(\Gamma - a_j)} - x_j^{k,z}(t) \le 0, \tag{44}$$

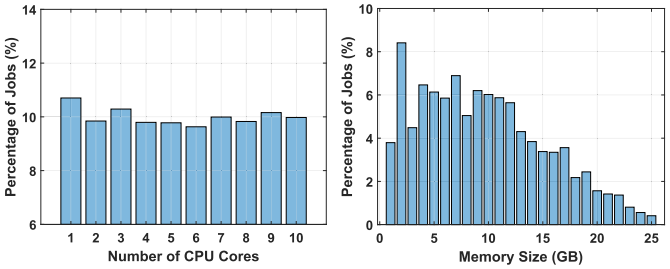


Fig. 1. The distribution of job resource demand.

where $\Gamma = \max\left\{\max\left\{a_j + 2 \quad p_j, 2a_j\right\}, \frac{4\sum_{j=1}^{N}\sum_{z\in\{m,r\}} p_j^z d_j^z n_j^z}{\sum_{i=1}^{M} C_i}\right\}$.

To simplify the algorithm design, we further create for each job j two dummy jobs, i.e., one map job and one reduce job. The reduce job is active only after the map job completes, thus, Constraint Eq. (43) can be ignored. By doing this, we can adopt the same approach as that in Section III-A to determine the CPU allocation rate for each job, i.e., $\sum_{k=1}^{n_j^z} x_j^{k,z}(t)$. With this rate, we then try to assign each task to a machine which can yield a small network latency of fetching the input data, by solving a linear optimization problem aiming at minimizing the overall network latency of all tasks. More specifically, we let $\xi_{i,j}^{k,z}$ to characterize the network latency of fetching the input data for task k of job j from machine i, then the objective function at time t is given by:

$$\Psi(t) = \sum_{j:t>a_j} \sum_{i=1}^{M} \sum_{z\in\{m,r\}} \sum_{k=1}^{n_j^z} \xi_{i,j}^{k,z} \cdot x_{i,j}^{k,z}(t), \qquad (45)$$

where $x_{i,j}^{k,z}$ denotes the CPU allocation rate on machine i and it satisfies: $\sum_{i=1}^{M}\sum_{k=1}^{n_j^z} x_{i,j}^{k,z}(t) = \sum_{k=1}^{n_j^z} x_j^{k,z}(t)$.

VI. Performance Evaluation

In this section, we evaluate the performance of our proposed online algorithms via extensive simulations driven by Google cluster workload traces [30]. The traces contain job submission and completion time of different Google services on a cluster consisting of three types of servers, each having 64, 32 and 16 CPU cores respectively. From the traces, we extract the statistics of processing time and resource demand of 6487 jobs (which is illustrated in Fig. 1) during a 10-hour period. Since a job may consist of multiple small tasks in the traces, we randomly select one task as its representative. For ease of presentation, we let OCORP denote the single-resource version of OCORP + M and OCORP+ denote the version with switching cost respectively in the rest of this paper.

1) Simulation Methodology: In our simulations, we set the OCO step-size, μ to $t^{k+0.5}$ in time-slot t for OCORP, OCORP+, and OCORP + M. In addition, we choose the length of each time-slot to be ten seconds. For the first three simulations, we run the experiment once till all the jobs contained in the trace complete. In the last simulation, i.e., the study of the mis-estimation of resource demands, we repeat the experiment five times and take the average for evaluation.

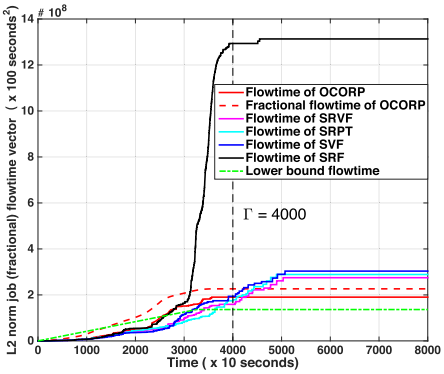


Fig. 2. The overall (fractional) job flowtime in 1D resource case.

2) Baseline Algorithms: We use the following four schemes as baselines for comparison with our proposed algorithms. These baselines are widely-adopted and studied for scheduling jobs with heterogeneous resource demands in today's computing clusters, e.g., [10], [29].

- **SRPT**: (Shortest Remaining Processing Time). In each time-slot, the cluster scheduler chooses jobs with shortest remaining processing time to schedule.
- **SRVF**: (Smallest Residual Volume First). In each time-slot, the cluster chooses jobs with smallest residual volume, which is defined as the product of the number of CPU cores and its remaining processing time.
- **SVF** (Smallest Volume First). In each time-slot, the cluster chooses jobs with the smallest volume.
- **SRF** (Smallest Resource First). In each time-slot, the cluster chooses jobs which request the smallest number of CPU cores.

A. Performance of OCORP

In this section, we make a comprehensive comparison between OCORP and other four baseline schemes.

Fig. 2 shows the l_2 norm of the exact job flowtime vector when different algorithms iterate over time. For OCORP, the curve showing the fractional job flowtime is always above that of the actual job flowtime but the former tracks the latter closely throughout the entire simulation. Based on this observation and the argument in Eq. (6), it is therefore justified to adopt the fractional job flowtime as a surrogate optimization objective for the design of schedulers which aim to minimize job flowtime. Fig. 2 also shows that, OCORP reduces the overall l_2 norm of the *actual* job flowtime vector by nearly 34% comparing to SVF, SRVF and SRPT upon completion of all jobs. Fig. 2 also includes a curve which depicts a lower bound of the overall job flowtime achieved under the optimal offline algorithm. The lower bound is quantified simply by taking each job size (processing time) as the corresponding job flowtime. One can observe from Fig. 2 that the overall job flowtime under OCORP is only 30% larger than this derived lower bound.

In Fig. 3, we first plot the total offered workload of all jobs entering the cluster up to the current time-slot. The other curves show the amount of unfinished workload for jobs entering the cluster so far under different scheduling

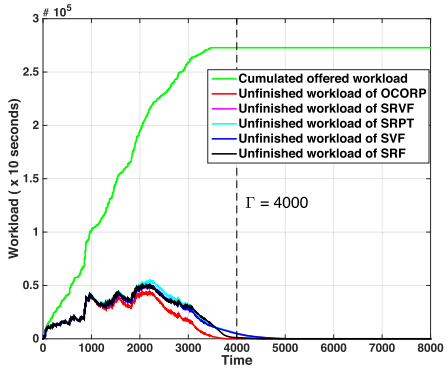


Fig. 3. The amount of unfinished work in 1D resource case.

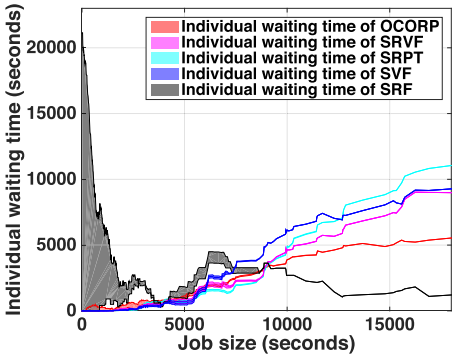


Fig. 4. The waiting time of jobs with different sizes in 1D resource case.

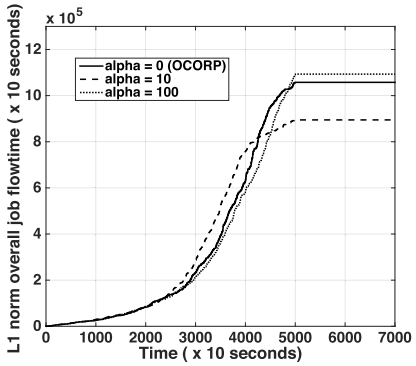


Fig. 5. The overall job flowtime under OCORP+.

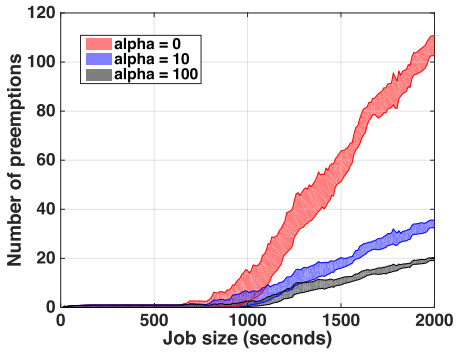


Fig. 6. The number of preemptions under different job sizes.

algorithms. Observe that OCORP has the smallest amount of unfinished workload for all time-slots comparing to other four scheduling schemes. For this experiment, our design target completion time for all jobs, namely, Γ is set to 4000 and according to Theorem 3, most of the jobs should have completed by $T \geq \Gamma$ under OCORP. This is indeed the case as observed from Fig. 3. For each algorithm, its makespan, i.e. the overall duration required to finish all jobs, is equal to the time at which its curve in Fig. 2 becomes plateau. This is the same time when its curve of unfinished workload decreases to zero in Fig. 3.

For the Google traces, jobs with small resource demand usually require a long time to complete. In this sense, SRF has the same effect as the longest-job-first scheduling policy and therefore can reduce the makespan substantially comparing to other baselines. Nevertheless, OCORP still yields a makespan comparable to that of SRF, which is around 40000 seconds long. By contrast, SRPT and SVF achieve a makespan of more than 50000 seconds. In other words, OCORP reduces the makespan by 20% comparing to SRPT and SVF while providing better fairness as we will show next.

To quantify the fairness of OCORP, we show the waiting time (i.e., $(f_j - p_j)$) for jobs with different sizes under all algorithms in Fig. 4. For each algorithm, we plot two curves where the upper one shows the largest waiting time achieved under each job size, and the lower curve corresponds to the smallest waiting time experienced by jobs of the same size. Observe from the figure that the variation among waiting times of jobs with the same job size under SRF is quite large. By contrast, other four schemes including OCORP offer smaller variance in waiting time for jobs of the same size. More importantly, OCORP achieves a much lower waiting time for big jobs than SVF and SRPT. This can be explained by Corollary 1 which guarantees OCORP to serve every unfinished job at least once within an interval of $\widetilde{T}$. In contrast, SVF and SRPT do not have such fairness provision and will always give priority to small jobs which results in the starvation of large jobs. In particular, Fig. 4 shows that, for jobs with processing time (size) longer than 20000 seconds, OCORP can reduce their waiting time by more than 40% comparing to SVF and SRPT. At the same time, OCORP still performs (nearly) equally well as SVF and SRPT for small jobs.

B. Performance of OCORP+

In this section, we study the effectiveness of controlling the number of job preemption and migration events by augmenting the OCO objective to reflect the associated switching costs. Fig. 5 illustrates how the overall job flowtime accumulates when OCORP+ updates over time under different values of α. Observe that, when all jobs in the cluster have completed, the overall job flowtime achieved by OCORP+ does not change much as α varies from 0 to 100. In particular, the overall job flowtime under $\alpha = 100$ is only 5% more than the case when $\alpha = 0$, which corresponds to the OCORP scheme. In other words, the overall job flowtime is not sensitive to the changes of α under OCORP+.

Fig. 6 shows the number of preemptions for jobs with different sizes under OCORP+ where the number of preemptions decreases substantially for all jobs as α increases. In particular,

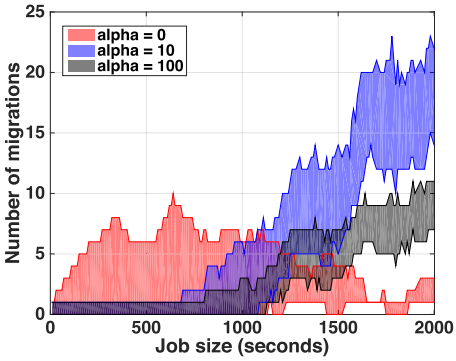


Fig. 7. The number of migrations under different job sizes.

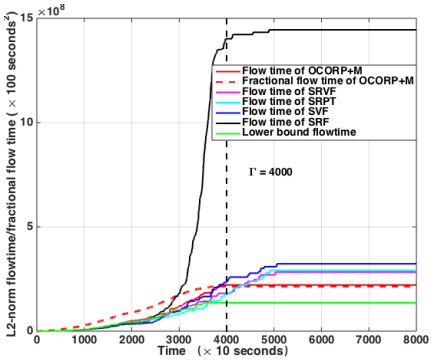


Fig. 8. The overall (fractional) job flowtime in 2D resource case.

the number of preemptions for jobs that require 2000 seconds to complete reduces nearly 80% by tuning α from 0 to 100. When $\alpha = 100$, most jobs will only be preempted no more than once every 100 seconds. Since most jobs contained in the Google traces have processing time of less than 100 seconds, preemption would rarely happen under OCORP+. This shows that, augmenting the objective function with a switching cost is quite effective for controlling and limiting the number of job preemptions.

Fig. 7 depicts the distribution of migrations under different job sizes. Note that migration seldom happens comparing to preemption. For jobs which require more than 10000 seconds to complete, the number of migrations under $\alpha = 0$ is smaller than that under $\alpha = 10$ or 100. In fact, to achieve a low job flowtime, OCORP+ prefers, if possible, to migrate a job across different machines within the cluster instead of preempting it so that the job can run continuously and achieve a small job flowtime. Nevertheless, if one continues to increase α to a relatively large value, the number of migrations will also be reduced eventually. In summary, by increasing the value of α under OCORP+, the total number of job preemptions and migrations can be reduced substantially while having negligible impact on the overall job flowtime performance.

C. Performance of OCORP + M

In this section, we evaluate the performance of the proposed OCORP + M Algorithm based on the same Google traces. In this case, each job specifies both the CPU resource (no. of CPU cores) and the Memory requirement when it arrives.

Fig. 8 shows that OCORP + M reduces the overall l_2 norm of the actual job flowtime vector by nearly 24% compared to other baseline schemes. Moreover, Fig. 9 shows that OCORP + M always has the smallest amount of unfinished workload and most of the jobs have been completed by $T \geq \Gamma$. Finally, Fig. 10 shows that OCORP + M can lead to a smaller waiting time for big jobs and thus implies OCORP + M provides a better fairness.

D. The Impact of Demand Misestimations

We also study the impact of the resource demand misestimations on the overall job performance. In particular, we introduce X percent error in our estimated resource demand for each job, i.e., the estimated demand is $d_j \cdot (1+X)$ ($m_j(1+X)$ in 2D case) for job j. X follows a Gaussian distribution with a mean of zero and a variance of σ^2, i.e., $X \sim \mathbb{N}(0, \sigma^2)$ where we tune σ^2 from 0.2 to 1. In addition, when we multiply d_j by a random noise, we round up the product to yield an integer. Furthermore, if the resulted estimation is negative, we modify the demand estimation of CPU to be 1 core and Memory to be 1GB. Since there are 6000 jobs in this experiment, the actual noise is still very close to σ^2. For instance, the actual variance for CPU and Memory after rounding is 0.198954 (0.383085) and 0.204134 (0.390934) respectively, when the setting of σ^2 is 0.2 (0.4).

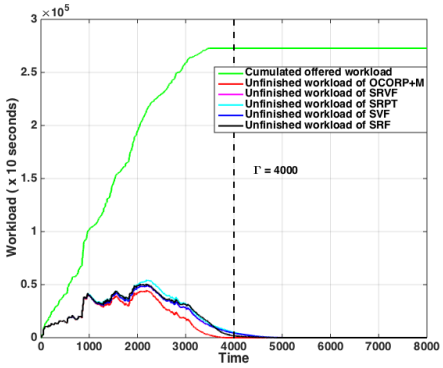


Fig. 9. The amount of unfinished work in 2D resource case.

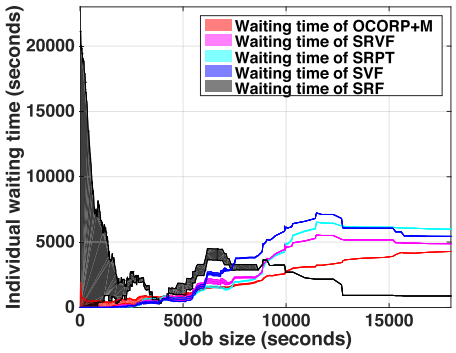
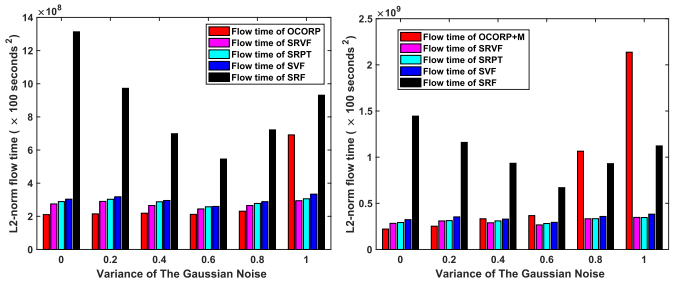


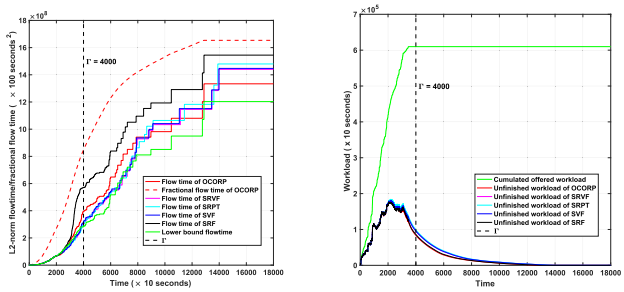
Fig. 10. The waiting time of jobs with different sizes in 2D resource case.

We depict the evaluation results in Fig. 11 and the left panel illustrates the 1D resource case, whereas the right panel shows the 2D resource case. As shown in Fig. 11a, when σ^2 does not exceed 0.8, our designed OCORP can still beat



(a) One-dimensional case (b) Two-dimensional case

Fig. 11. The impact of demand misestimation on the job performance.



(a) Accumulated job flowtime (b) Remaining workload

Fig. 12. The job flowtime and remaining workload over time under different schemes.

the other four baseline schemes substantially. In particular, when $\sigma^2 = 0.8$, OCORP reduces the overall job flowtime by nearly 20% comparing to the best baseline, i.e., SRVF. Interestingly, OCORP also performs quite stable when $\sigma^2 \leq 0.6$. When going to the 2D case, Fig.11b shows that OCORP + M beats the baselines by 20% when $\sigma^2 = 0.2$. However, if σ^2 further goes to 0.4, OCORP + M performs similarly to others. Nevertheless, the resource demand can be efficiently estimated in production clusters via analyzing the statistics of the previously submitted jobs such the Carbyne Scheduler [31], or using machine-learning based approach, e.g., [32], [33]. By using advanced machine learning techniques [32], the estimation error is usually less than 0.4, which means the variance is below 0.16. In addition, the Carbyne Scheduler also set the maximum estimation error to be 50%. As such, we conclude OCORP + M can still work well even if there are estimation errors in real clusters.

E. Evaluation for MapReduce-Like Jobs

In this part, we conduct additional experiments to evaluate the performance of OCORP in a MapReduce cluster driven by Google traces. In this case, each job may consist of multiple map tasks which can run in parallel. We apply the extended OCORP Algorithm as discussed in Section V to schedule the tasks of each job where we only consider the CPU demand.

We depict the evaluation results in Fig. 12. The left panel captures the l_2 norm of the exact job flowtime vector under different algorithms over time. When all the jobs complete, the overall l_2 norm under OCORP is the closest one to the lower bound among all these schemes. Moreover, OCORP can reduce the total job flowtime by 10% comparing to the SRVF scheme. The right panel characterizes the total remaining workload under all schemes. And the results indicate that, all these schemes complete roughly the same amount of workload when they iterate over time.

VII. Related Work

The design and implementation of job scheduling and resource allocation algorithms for computing clusters have also attracted much attention from the research community lately. One major thread of research is to derive performance bounds to minimize the total job completion time such as [13], [15]. Tan *et al.* design the *Coupling scheduler* [14] to mitigate the starvation problem caused by Reduce tasks in MapReduce Clusters. It is well known in scheduling literature that the SRPT scheduler (Shortest Remaining Processing Time) is optimal for minimizing the overall flow-time on a single server. Others have extended the SRPT scheduler to minimize the total job flow-times under different settings, e.g., [12], [15]–[17]. However, all of these studies do not consider resource packing and assume each server/ slot can only hold one job/ task at any time. To tackle this limitation, Grandl *et al.* present Tetris [10], which is a scheduler that assigns tasks to machines based on the requirements of different resource types which include CPU, Memory, Disk and Network. Tetris combines the SRPT scheduler (Shortest Remaining Processing Time) and heuristic algorithms for the multi-dimensional resource packing problem to compute a weighted score for each of the mapping pairs between the available server and unscheduled tasks. Then, Tetris assigns a task with the highest score to the available server.

The metric of fractional job flowtime was first introduced in [34] to analyze the competitive-ratio of the SRPT scheduler instead of designing a new scheduling algorithm on multiple identical servers. Afterwards, [18], [35] adopt this metric to bound the competitive ratio for the proposed job scheduling algorithms in homogeneous clusters via using the dual-fitting approach. As a comparison, this work adopts the fractional job flowtime as a new optimization metric. With this new yet approximated objective, we can design online algorithms directly by applying optimization techniques instead of exploring some simple heuristics.

Recently, Im *et al.* begin to study the online job scheduling problems with the objective to minimize the overall weighted job flowtime under polyhedral constraints in [36]–[38]. In this setting, the whole cluster holds one resource pool and each job is allocated a resource rate over time subject to the packing constraints. In these works, the Proportional Fairness (PF) Algorithm is designed under which the resource rates are allocated via solving a utility maximization problem. By adopting the potential function analysis, it shows that PF achieves a constant competitive ratio under the resource augmentation setting. By contrast, this paper studies a more difficult problem where resource allocation decision and machine assignment policy need to be designed jointly, whereas [36]–[38] only need to determine the resource allocation rates of jobs. The most similar work to this paper appears in [29] which studies a scheduling problem of jobs with non-uniform demands on

multiple servers and characterizes the performance of different schemes in terms of the overall job flowtime. However, [29] only analyzes the transient scheduling case and assume all servers are identical. By contrast, this paper considers a heterogeneous setting of server capacities and moreover designs an efficient machine packing policy to avoid large machine fragmentations.

OCO was first introduced by [39], [40] and now is a widely adopted method for online learning as well as sequential inference, especially when the choice of convex cost functions can vary in an adversary manner and the decision needs to be made before the future cost functions are revealed. While early OCO techniques can only deal with unconstrained convex problems, recent advances in the field have enabled one to analyze and solve OCO problems with long-term constraints, e.g., [22], [23]. In addition, [22], [23] introduce the notion of static regret to quantify the difference of the cost between the online solution and the overall best static solution. However, as a static optimal benchmark may perform poorly, even a scheme which can achieve a small static regret is still not attractive for practical purpose. As such, [21] extends the analysis of static regret to that of dynamic regret, where the performance of an OCO algorithm is benchmarked by the best dynamic solution with a priori information one-slot-ahead on the cost function subjecting to a time-varying constraint which is known in advance. Although we also adopt the OCO approach to develop our algorithms, the scheduling problem investigated in this paper is quite different from the prior OCO problems in two major aspects. Firstly, the uncertainty is only due to job arrivals in this paper, whereas both the cost function and constraints are unknown before making decisions in [21]–[23]. Secondly, our scheduling problem includes both short-term constraints which need to be satisfied strictly at each time-slot and other constraints that need to be satisfied in the long term. [22], [23], [26] can only handle a single type of constraints. Thirdly, we even generalize the definition of dynamic regret in [22], [23], which still allows temporarily violations of long-term constraints in the benchmark optimal solution. While the benchmark optimal solution needs to satisfy the long-term constraint strictly in [22], [23], [26].

VIII. Conclusion and Future Work

This paper is an attempt to address the impact of two key sources of diversities in computing clusters: job processing size and resource demand. Our primary goal and contribution are to design both offline and online scheduling algorithms with theoretical performance guarantees by adopting the OCO framework, and to advance the modeling techniques of job scheduling aimed at minimizing the overall job flowtime with resource packing on multiple heterogeneous servers. Our proposed algorithms can be generalized to deal with scheduling problems with multi-dimensional resources and integer variables, and further implemented in form of extensions of open-source frameworks like Hadoop YARN [3]. As future work, we plan to extend our scheduling algorithms for jobs consisting of multiple small tasks. Possible extensions of this work also include job scheduling in non-migration modes and scenarios where there is machine service variability.

References

[1] Y. Liu, H. Xu, and W. C. Lau, "Online job scheduling with resource packing on a cluster of heterogeneous servers," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1441–1449.

[2] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proc. 24th ACM Symp. Operating Syst. Princ.*, Nov. 2013, pp. 423–438.

[3] V. K. Vavilapalli *et al.*, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, Oct. 2013, pp. 1–16.

[4] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proc. 24th ACM Symp. Operating Syst. Princ.*, Nov. 2013, pp. 69–84.

[5] B. Hindman *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. NSDI*, 2011, p. 22.

[6] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *Proc. 8th ACM Eur. Conf. Comput. Syst. - EuroSys*, 2013, pp. 351–364.

[7] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, Apr. 2015, pp. 1–17.

[8] E. Boutin *et al.*, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proc. OSDI*, 2014, pp. 285–300.

[9] K. Karanasos *et al.*, "Mercury: Hybrid centralized and distributed scheduling in large shared clusters," in *Proc. ATC*, 2015, pp. 485–497.

[10] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 1–12.

[11] X. Ren, G. Ananthanarayanan, A. Wierman1, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," in *Proc. Sigcomm*, 2015, pp. 379–392.

[12] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós, "On scheduling in map-reduce and flow-shops," in *Proc. 23rd ACM Symp. Parallelism Algorithms Archit. - SPAA*, 2011, pp. 289–298.

[13] F. Chen, M. Kodialam, and T. V. Lakshman, "Joint scheduling of processing and shuffle phases in MapReduce systems," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1143–1151.

[14] J. Tan, X. Meng, and L. Zhang, "Delay tails in MapReduce scheduling," in *Proc. 12th ACM SIGMETRICS/Perform. Joint Int. Conf. Meas. Modeling Comput. Syst. - SIGMETRICS*, 2012, pp. 5–16.

[15] Y. Yuan, D. Wang, and J. Liu, "Joint scheduling of MapReduce jobs with servers: Performance bounds and experiments," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 2175–2183.

[16] M. Lin, L. Zhang, A. Wierman, and J. Tan, "Joint optimization of overlapping phases in MapReduce," in *Proc. SIGMETRICS*, 2013, pp. 16–18.

[17] Y. Zheng, N. B. Shroff, and P. Sinha, "A new analytical technique for designing provably efficient MapReduce schedulers," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 1600–1608.

[18] S. Im, J. Kulkarni, and B. Moseley, "Temporal fairness of round robin: Competitive analysis for lk-norms of flow time," in *Proc. 27th ACM Symp. Parallelism Algorithms Archit.*, Jun. 2015, pp. 155–160.

[19] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," *J. ACM*, vol. 47, no. 4, pp. 617–643, Jul. 2000.

[20] S. Im, B. Moseley, K. Pruhs, and E. Torng, "Competitively scheduling tasks with intermediate parallelizability," in *Proc. 26th ACM Symp. Parallelism Algorithms Archit.*, Jun. 2014, pp. 22–29.

[21] T. Chen, Q. Ling, and B. G. Giannakis, "An online convex optimization approach to dynamic network resource allocation," *IEEE Trans. Signal Process.*, vol. 65, no. 24, pp. 6350–6364, Dec. 2017.

[22] M. Mahdavi, R. Jin, and T. Yang, "Trading regret for efficiency: Online convex optimization with long term constraints," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2503–2528, 2012.

[23] H. Yu and J. M. Neely, "A low complexity algorithm with $O(\sqrt{T})$ regret and constraint violations for online convex optimization with long term constraints," 2016, *arXiv:1604.02218*. [Online]. Available: https://arxiv.org/abs/1604.02218

[24] S. Leonardi and D. Raz, "Approximating total flow time on parallel machines," in *Proc. 29th Annu. ACM Symp. Theory Comput. - STOC*, 1997, pp. 110–119.

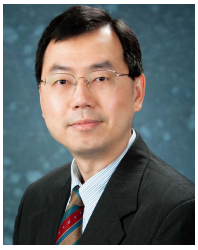
[25] K. Fox and B. Moseley, "Online scheduling on identical machines using SRPT," in *Proc. 22nd Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2011, pp. 120–128.

[26] M. J. Neely and H. Yu, "Online convex optimization with time-varying constraints," 2017, *arXiv:1702.04783*. [Online]. Available: http://arxiv.org/abs/1702.04783

[27] R. Jenatton, J. Huang, and C. Archambeau, "Adaptive algorithms for online convex optimization with long-term constraints," in *Proc. ICML*, 2016, pp. 402–411.
[28] G. H. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities* (Cambridge Mathematical Library). Cambridge, U.K.: Cambridge Univ. Press, 1934.
[29] S. Im, M. Naghshnejad, and M. Singhal, "Scheduling jobs with non-uniform demands on multiple servers without interruption," in *Proc. IEEE INFOCOM - 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.
[30] C. Reiss, J. Wilkes, and L. J. Hellerstein. (May 2011). *Google Cluster-Usage Traces*. [Online]. Available: http://code.google.com/p/googleclusterdata
[31] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *Proc. OSDI*, 2016, pp. 65–80.
[32] X. L. Li, N. Qi, Y. He, and B. McMillan, "Practical resource usage prediction method for large memory jobs in HPC clusters," in *Supercomputing Frontiers* (Lecture Notes in Computer Science). New York, NY, USA: Springer, 2019.
[33] F. Schmidt, M. Niepert, and F. Huici, "Representation learning for resource usage prediction," 2018, *arXiv:1802.00673*. [Online]. Available: http://arxiv.org/abs/1802.00673
[34] S. Anand, N. Garg, and A. Kumar, "Resource augmentation for weighted flow-time explained by dual fitting," in *Proc. 23rd Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2012, pp. 1228–1241.
[35] N. R. Devanur and Z. Huang, "Primal dual gives almost optimal energy efficient online algorithms," in *Proc. 25th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2014, pp. 1123–1140.
[36] S. Im, J. Kulkarni, and K. Munagala, "Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints," in *Proc. 46th Annu. ACM Symp. Theory Comput.*, May 2014.
[37] S. Im, J. Kulkarni, and K. Munagala, "Competitive flow time algorithms for polyhedral scheduling," in *Proc. IEEE 56th Annu. Symp. Found. Comput. Sci.*, Oct. 2015, pp. 313–322.
[38] S. Im, J. Kulkarni, B. Moseley, and K. Munagala, "A competitive flow time algorithm for heterogeneous clusters under polytope constraints," in *Proc. APPROX-RANDOM*, 2016, pp. 1–15.
[39] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. ICML*, 2003, pp. 928–936.
[40] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Mach. Learn.*, vol. 69, nos. 2–3, pp. 169–192, Oct. 2007.

Huanle Xu (Member, IEEE) received the B.Sc.(Eng.) degree from the Department of Information Engineering, Shanghai Jiao Tong University (SJTU), in 2012, and the Ph.D. degree from the Department of Information Engineering, The Chinese University of Hong Kong (CUHK), in 2016. His primary research interests focus on job scheduling and resource allocation in cloud computing, decentralized social networks, parallel graph algorithms, and machine learning. He is particularly interested in designing and implementing wonderful algorithms for large-scale systems and their applications using optimization tools and theories.

Yang Liu (Student Member, IEEE) received the B.Sc.(Eng.) degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University (SJTU), in 2016. He is currently pursuing the Ph.D. degree with the Department of Information Engineering, The Chinese University of Hong Kong (CUHK), supervised by Prof. Wing Cheong Lau. His research interests include job scheduling and resource management in the cloud and computing cluster, online learning, and multi-armed bandit algorithms. He is particularly interested in designing and implementing resource management algorithms for distributed systems.



Wing Cheong Lau (Senior Member, IEEE) received the B.S.E.E. degree from The University of Hong Kong and the M.S. and Ph.D. degrees in electrical and computer engineering from The University of Texas at Austin. Before joining CUHK, he spent ten years in USA with Bell Labs, Holmdel and Qualcomm, and San Diego. He is currently an Associate Professor with the Department of Information Engineering and the Director of the Mobile Technologies Center, The Chinese University of Hong Kong (CUHK). His recent research projects include Resource Allocation and Management for Data-Center-Scale Computing, Online Social Network Privacy and Vulnerabilities, Authenticated 2D Barcodes, Decentralized Social Networking Protocols/Systems. He holds 19 U.S. patents. Related research findings have culminated in more than 100 scientific papers in major international journals and conferences. His research interests include networking protocol design and performance analysis, network/ systems security, mobile computing, and system modeling. He is a member of Tau Beta Pi. He is/has been with the Technical Program Committee for various international conferences, including ACM Sigmetrics Mobihoc, IEEE Infocom, SECON, ICC, Globecom, WCNC, VTC, and ITC. He also served as a Guest Editor for the Special Issue on High-Speed Network Security of the IEEE JOURNAL OF SELECTED AREAS IN COMMUNICATIONS (JSAC). He is a member of the ACM.