Scheduling Multi-Server Jobs with Sublinear Regrets via Online Learning

Hailiang Zhao, Shuiguang Deng, *Senior Member, IEEE,* Zhengzhe Xiang, Xueqiang Yan, Jianwei Yin, Schahram Dustdar, *Fellow, IEEE*, and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Multi-server jobs that request multiple computing resources and hold onto them during their execution dominate modern computing clusters. When allocating the multi-type resources to several co-located multi-server jobs simultaneously in online settings, it is difficult to make the tradeoff between the parallel computation gain and the internal communication overhead, apart from the resource contention between jobs. To study the computation-communication tradeoff, we model the computation gain as the speedup on the job completion time when it is executed in parallelism on multiple computing instances, and fit it with utilities of different concavities. Meanwhile, we take the dominant communication overhead as the penalty to be subtracted. To achieve a better gain-overhead tradeoff, we formulate an cumulative reward maximization program and design an online algorithm, named OgaSched, to schedule multi-server jobs. OgaSched allocates the multi-type resources to each arrived job in the ascending direction of the reward gradients. It has several parallel sub-procedures to accelerate its computation, which greatly reduces the complexity. We proved that it has a sublinear regret with general concave rewards. We also conduct extensive trace-driven simulations to validate the performance of OgaSched. The results demonstrate that OgaSched outperforms widely used heuristics by 11.33%, 7.75%, 13.89%, and 13.44%, respectively.

Index Terms—Multi-server job, online gradient ascent, online scheduling, regret analysis.

1 Introduction

In today's computing clusters, whether in the cloud data centers or at the network edge, many jobs request multiple resources (CPUs, GPUs, etc.) simultaneously and hold onto them during their executions. For example, graph computations [1], federated learning [2], distributed DNN model trainings [3], etc. In this paper, we refer to these jobs as *multi-server jobs* [4], [5]. Multi-server jobs of diverse resource requirements arrive at the cluster online, which puts great pressure to current resource allocation policies to achieve a high computation efficiency.

When allocating the multi-type resources to several co-located multi-server jobs simultaneously in online settings, it is difficult to make the tradeoff between *the parallel computation gain* and *the internal communication overhead*, apart from the resource contention between jobs. Here the parallel computation gain refers to the speedup on the job completion time when it is executed in parallelism on multiple computing instances, which could be modeled with a function of the allocated multi-type resources [6]. Correspondingly, the internal communication overhead refers to the cost caused by non-computation operations such as data synchronization, averaging, message passing, etc., between the distributed workers. To achieve better computation efficiency, we need to consider the following key challenges.

- *Resource contention with service locality.* With service locality, a multi-server job can only be processed by a subset of computing instances where the resource requirements, session affinity [7], and other obligatory constraints are satisfied. When several multi-server jobs arrive simultaneously, how to allocate the limited resources to them without degenerating the computation efficiency is challenging.
- *Unknown arrival patterns of jobs.* In real-life scenarios, the resource allocation should be made online without the knowledge of future job arrivals. The lack of information on the problem space could lead to a solution far from the global optimum.
- *The parallel computation gain does not increase in a linear rate with the quantity of allocated resources.* For instance, in distributed DNN model training or federated learning, adding workers (that request more resources) does not improve the training speed linearly [3], [6]. This is because the overhead of all-reduce operation between workers or the averaging of local gradients increase with the number of participated workers, especially when the workers are distributed in different machines and communicate with each other through network [8], [9], [10]. Compared with high-speed intra-node communication channels such

- H. Zhao and S. Deng are with Hainan Institute of Zhejiang University (Sanya 572025, China) and the College of Computer Science and Technology, Zhejiang University (Hangzhou 310027, China). E-mail: {hliangzhao, dengsg}@zju.edu.cn
- Z. Xiang is with the School of Computer Science and Technology, Hangzhou City University, Hangzhou 310015, China. E-mail: xiangzz@zucc.edu.cn
- X. Yan is with Huawei Technologies Co Ltd, Shanghai 201206, China. E-mail: yanxueqiang1@huawei.com
- J. Yin is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China. E-mail: zjuyjw@zju.edu.cn
- S. Dustdar is with the Distributed Systems Group, Technische Universität Wien, 1040 Vienna, Austria. E-mail: dustdar@dsg.tuwien.ac.at
- A. Y. Zomaya is with the School of Computer Science, University of Sydney, Sydney, NSW 2006, Australia. E-mail: albert.zomaya@sydney.edu.au
- Shuiguang Deng is the corresponding author.

as NVLink, the inter-node bandwidth through NIC is relatively much slower. Another example is graph computation. Without a well-designed graph partition policy, the speedup of message-passing between graph nodes can be significantly slowed down [1], [11].

- *The type of resource which dominates the communication overhead varies to different job types.* For example, in graph computation jobs, the dominant communication overhead lies in the internal input-output data transferring between the interdependent CPU- and memory-intensive tasks [12]. However, the dominant overhead of the distributed training of DNNs lies in the data averaging and synchronizing between the GPU-intensive workers through network [13]. This variety greatly complicates the theoretical analysis for the gain-overhead tradeoff.

Despite the vast literature on the online resource allocation algorithms [3], [6], [13], [14], [15], [16], [17], [18], their model formulation and theoretical analysis which places emphasis on the gain-overhead tradeoff is limited. To fill the theoretical gap, in this paper, we propose an online scheduling algorithm, termed as OGASCHED, to *learn* to allocate multi-type resources to co-located multi-server jobs online to maximize the overall computation efficiency. We try to analyze the tradeoff in a generic way. The generality is embodied in the following points. First of all, different from the specific works on deep learning jobs [6], [13], [3] or query jobs [12], we allow different types of multi-server jobs to *co-locate* in the cluster which consists of heterogeneous computing resources. Different job types can have different resource requirements while different computing instances can be equipped with diverse quantities and types of resources. Secondly, we adopt general zero-startup non-decreasing utility functions to model the parallel computation gain in terms of the job completion time. Compared to existing literature, we allow the utilities to be diverse in their *level of concavity*. Specifically, we provide both analysis and experiments on linear, polynomial, logarithmic, and reciprocal utilities. Thirdly, we makes no assumptions on the arrival patterns of multi-server jobs. OGASCHED requests no knowledge on the job arrival distributions but tries to learn them to make better scheduling decisions.

In our model formulation, the computation efficiency is modeled in the way of cumulative reward. Time is slotted, and the cumulative reward is obtained by summing up the reward in each time slot, where a single-time reward is a linear aggregation of each job's reward. Further, a job's reward at each time is designed as the achieved parallel computation gain aggregated over the allocated resources minus the penalty introduced by the dominant communication overhead. At each time, OGASCHED allocates resources to each arrived job in the direction that *makes the gradient of the reward increase*. OGASCHED is capable of handling high dimensional inputs in stochastic scenarios with unpredictable behaviors. We adopt regret, i.e., the gap on the cumulative reward between the proposed online algorithm and the offline optimum achieved by an oracle [19], to analyze the performance lower bound of OGASCHED. We prove that, OGASCHED has a State-of-the-Art (SOTA) regret, which is sublinear with the time slot length and the number of job types. This work fulfills one of the key deficiencies of the past works in the modeling and analysis of the gain-overhead tradeoff for multi-server jobs. The contributions are summarized as follows.

- We systematically study the resource allocation of co-located multi-server jobs in terms of the tradeoff between the parallel computation gains and the internal communication overheads. Our study is general in scenario settings and it sufficiently takes the characters of the diminishing marginal effect of gains into consideration.
- We propose an algorithm, i.e., OGASCHED, to learn to strike a balanced computation-communication tradeoff. OGASCHED has no assumptions on the job arrival patterns. With a nice setup (defined in Sec. 3.1), OGASCHED achieves a SOTA regret $\mathcal{O}\big(\mathcal{H}_{\mathcal{G}} \cdot \sqrt{T}\big)$ for general concave non-linear rewards, where T is the time slot length, and $\mathcal{H}_{\mathcal{G}}$ (formally defined in (49)) is parameter that characterizes the bipartite graph model. OGASCHED is accelerated by well-designed parallel sub-procedures. The parallelism helps yield a complexity of $\mathcal{O}\big(\log(K)\big)$, where K is the number of resource types.
- We conduct extensive trace-driven simulations to validate the performance of OGASCHED. The simulation results show that OGASCHED outperforms widely used heuristics including DRF [20], FAIRNESS, BINPACKING, and SPREADING by 11.33%, 7.75%, 13.89%, and 13.44%, respectively. We also provide large-scale validations.

The rest of this paper is organized as follows. We formulate the online scheduling problem for multi-server jobs in Sec. 2. We then present the design details of OGASCHED with regret analysis and discuss its extensions in Sec. 3. We demonstrate the experimental results in Sec. 4, and discuss related works in Sec. 5. Finally, we conclude this paper in Sec. 6.

2 BIPARTITE SCHEDULING WITH REGRETS

We consider a cluster of heterogenous computing instances serving several types of multi-server jobs. Here the computing instances can be VMs in clouds, or local servers at the network edge. The computing instances work collaboratively to provide resources to serve the considered jobs. Different computing instances are equipped with different types and quantities/specifications of resources, including CPU cores, memory, bandwidth, GPUs, etc. Jobs of different types can have different demands on them. Key notations used in this paper are summarized in Tab. 1.

2.1 Online Bipartite Scheduling

We use a bipartite graph $\mathcal{G} = (\mathcal{L}, \mathcal{R}, \mathcal{E})$ to model the job-server constraints, as shown in Fig. 1. In graph $\mathcal{G}$, $\mathcal{L}$ is the set of job types and indexed by l while $\mathcal{R}$ is the set of computing instances and indexed by r. The connections between the job types and the computing instances are recorded in $\mathcal{E}$.

TABLE 1
Summary of key notations.

NOTATION	DESCRIPTION
$\mathcal{T}$	Time horizon of length T
$\mathcal{G} = (\mathcal{L}, \mathcal{R}, \mathcal{E})$	The bipartite graph
$l \in \mathcal{L}$	A job type (port)
$r \in \mathcal{R}$	A computing instance
$(l, r) \in \mathcal{E}$	The edge (channel) between l and r
$\forall r : \mathcal{L}_r$	The set of job types connect to r
$\forall l : \mathcal{R}_l$	The set of computing instances connect to l
$\boldsymbol{x}(t)$	The job arrival status at time t
$\boldsymbol{y}(t)$	The scheduling decision at time t
$\mathcal{K}$	The set of different types of resources
$\forall l : \boldsymbol{a}_l$	Resource requirements of type-l job
$\forall r, k : c_r^k$	The number of type-k resources equipped by r
$q(\boldsymbol{x}(t), \boldsymbol{y}(t))$	The reward of time t
$\forall k : f_k(\cdot)$	Computation gain of type-k devices
$\forall k : \beta_k \in [0, 1]$	Coefficient of type-k communication overhead

Because of the job-server constraints, type-l job may only be served by a subset of $\mathcal{R}$. We denote the subset by

$$\mathcal{R}_l = \Big\{ r \in \mathcal{R} \mid (l, r) \in \mathcal{E} \Big\}. \tag{1}$$

Similarly, we use

$$\mathcal{L}_r = \Big\{ l \in \mathcal{L} \mid (l, r) \in \mathcal{E} \Big\}. \tag{2}$$

to represent the set of job types that connect to computing instance r. We designate each job type $l \in \mathcal{L}$ as *port* and each connection $(l, r) \in \mathcal{E}$ as *channel*. $\mathcal{G}$ is called right d-regular *iff* the indegree of each right vertex is d, i.e., $\forall r \in \mathcal{R}, |\mathcal{L}_r| = d$.

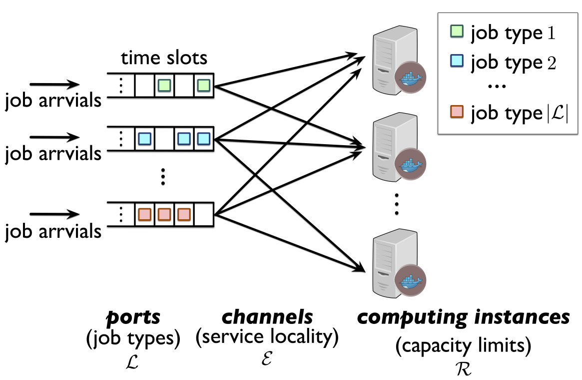


Fig. 1. The bipartite graph model for online job scheduling.

Time is discretized, and at each time $t \in \mathcal{T} \triangleq \{1, ..., T\}$, from each port, at most one job yields. Let us denote by

$$\boldsymbol{x}(t) = \Big[x_l(t) \Big]_{l \in \mathcal{L}} \in \{0, 1\}^{|\mathcal{L}|} \tag{3}$$

the job arrival status at time t. We do not make any assumption on the job arrival patterns or distributions. The cluster has K types of resources, and computing instance r has c_r^k type-k resources, where $k \in \mathcal{K} \triangleq \{1, 2, .., K\}$. For each type-$l$ job, we denote its maximum requests on each resource by $\boldsymbol{a}_l = [a_l^k]_{k \in \mathcal{K}} \in \mathbb{N}^{|\mathcal{K}|}$. At time t, we use

$$\boldsymbol{y}(t) = \Big[y_{(l,r)}^k(t) \Big]_{l \in \mathcal{L}, r \in \mathcal{R}_l, k \in \mathcal{K}} \in \mathbb{R}_{\geq 0}^{\sum_{l \in \mathcal{L}} |\mathcal{R}_l| \times K} \tag{4}$$

to denote the scheduling decision. Here we allow $y_{(l,r)}^k(t)$ to be fractional. Taking GPU as example, Machine-Learning-as-a-Service (MLaaS) platforms support GPU sharing in a space- and time-multiplexed manner by intercepting CUDA APIs [21], [22], [23].

The first constraint is that, through each channel, a job should not be allocated with resources more than it requires. Formally, we have

$$0 \leq y_{(l,r)}^k(t) \leq a_l^k, \forall l, r, k, t. \tag{5}$$

The second constraint $\boldsymbol{y}(t)$ should satisfy is that, the resources allocated out from any computing instance r should not more than it has:

$$\sum_{l \in \mathcal{L}_r} y_{(l,r)}^k(t) \leq c_r^k, \forall r, k, t. \tag{6}$$

We denote by $\mathcal{Y} \triangleq \{ \boldsymbol{y} \in \mathbb{R}^{\sum_{l \in \mathcal{L}} |\mathcal{R}_l| \times K} \mid$ (5) and (6) hold$\}$ to represent the solution space from here on.

2.2 Computation-Communication Tradeoff

The performance metric we use for online bipartite scheduling is designed as the gain obtained by the parallel computation through multi-type resources minus the penalty introduced by the dominant communication overheads. Specifically, we denote by $q_l(\boldsymbol{x}(t), \boldsymbol{y}(t))$ the reward of port l at time t , and it is formulated as

$$q_l(\boldsymbol{x}(t), \boldsymbol{y}(t)) = x_l(t) \Big[\sum_{k \in \mathcal{K}} f_k \Big(\sum_{r \in \mathcal{R}_l} y_{(l,r)}^k(t) \Big) - \max_{k \in \mathcal{K}} \Big\{ \beta_k \sum_{r \in \mathcal{R}_l} y_{(l,r)}^k(t) \Big\} \Big]. \tag{7}$$

In this formulation, the first part, $\sum_{k \in \mathcal{K}} f_k(\sum_{r \in \mathcal{R}_l} y_{(l,r)}^k(t))$, is the parallel computation gain, which is linearly aggregated over each type of resource, in proportional to each resource's weight. Jobs of different types can have different combinations of weights. $f_k(\cdot)$ is the gain achieved by $\sum_{r \in \mathcal{R}_l} y_{(l,r)}^k(t)$ type-k resources collaboratively, where $f_k(\cdot)$ is a zero-startup concave utility defined in $\mathbb{R}_{\geq 0}$. Note that $\sum_{r \in \mathcal{R}_l} y_{(l,r)}^k(t)$ is the quota of the type-k resources allocated to the type-l job at t. As we have analyzed before, $\{f_k(\cdot)\}_{k \in \mathcal{K}}$ are non-decreasing concave functions because the marginal effect of parallel computation decreases successively when increasing participated resources [24], [25]. We expect $\{f_k(\cdot)\}_{k \in \mathcal{K}}$ to be *continuously differentiable* because it helps design a policy that yields a nice lower bound of the reward. The details will be demonstrated in Sec. 3.3. If $\{f_k(\cdot)\}_{k \in \mathcal{K}}$ are not differentiable everywhere, we can apply subgradient ascent-related techniques in the policy design. The second part in (7) is $\max_{k \in \mathcal{K}} \{\beta_k \sum_{r \in \mathcal{R}_l} y_{(l,r)}^k(t)\}$, which reflects the dominant weighted communication overheads over different types of resources. For example, in federated learning at the edge, the dominant communication overhead lies in the averaging and synchronizing of data between each edge server over the network [26]. Another example is graph computation, in which the job is organized into a direct acyclic graph (DAG), and the dominant communication overhead falls into the data & message passing between CPU- and memory-intensive tasks [12]. $\{\beta_k\}_{k \in \mathcal{K}}$ are the coefficients to balance the gain and the overhead. W.L.O.G., we set each $\beta_k \in [0, 1]$. Theoretically, the second part of (7) is a penalty, the minimization of which guides the scheduling decisions to balance the communication overheads of different device types. Our reward design encourages each job to be served with the balance between the computation gain and the communication overhead being achieved.

2.3 Regret Minimizing

Based on the above, we define the overall reward at time t as the linear aggregation over each port:

$$q\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big) = \sum_{l \in \mathcal{L}} q_l\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big). \tag{8}$$

The cumulative reward of scheduling policy π over the time horizon $\mathcal{T}$ is obtained by summing up the rewards obtained at each time until T:

$$Q^{\pi}\Big(\{\boldsymbol{x}(t)\}_1^T, \{\boldsymbol{y}(t)\}_1^T\Big) = \sum_{t \in \mathcal{T}} q\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big), \tag{9}$$

where the scheduling decisions $\{\boldsymbol{y}(t)\}_1^T$ are made under the guidance of policy π. In the following, we just use Q and drop the superscript π for simplification.

We do not make any assumption on the distribution of the job arrival trajectory $\{\boldsymbol{x}(t)\}_1^T$. To obtain a non-trivial performance measure, we cast the multi-server bipartite scheduling problem into the framework of online learning, which prompts us to compare the performance of the online policy π with the best offline stationary policy π^* [27], [28]. Let us denote by $\boldsymbol{y}^*$ the optimal offline stationary resource allocation decision guided by policy π^*, i.e.,

$$\boldsymbol{y}^* = \arg\sup_{\boldsymbol{y} \in \mathcal{Y}} Q\Big(\{\boldsymbol{x}(t)\}_1^T, \boldsymbol{y}\Big), \tag{10}$$

Physically, $\boldsymbol{y}^*$ is the optimal stationary resource reservation decisions for each port whatever the actual job arrival status $\boldsymbol{x}(t)$ is. Formally, we define the regret $R_T^{\pi}\big(\{\boldsymbol{x}(t)\}_1^T\big)$ for the job arrival trajectory $\{\boldsymbol{x}(t)\}_1^T$ as

$$R_T^{\pi}\Big(\{\boldsymbol{x}(t)\}_1^T\Big) \triangleq Q\Big(\{\boldsymbol{x}(t)\}_1^T, \boldsymbol{y}^*\Big) - Q\Big(\{\boldsymbol{x}(t)\}_1^T, \{\boldsymbol{y}(t)\}_1^T\Big).$$

The regret of policy π is further defined as the maximum regret achieved over every possible job arrival trajectory:

$$R_T^{\pi} \triangleq \sup_{\forall \{\boldsymbol{x}(t)\}_1^T} R_T^{\pi}\Big(\{\boldsymbol{x}(t)\}_1^T\Big). \tag{11}$$

Our goal is to find a policy π, under which a sequence of bipartite scheduling decisions $\{\boldsymbol{y}(t)\}_1^T$ is yielded, to minimize R_T^{π}.

3 Online Gradient Ascent

To minimize the regret R_T^{π}, we resort to an online variant of the gradient-based methods, online gradient ascent (OGA) [29]. A series of recent works have demonstrated that OGA achieves the best possible regret for online caching problems in different network settings when the rewards are linear [28], [30], [31], [32]. In this paper, we extend OGA to the online bipartite scheduling problem for multi-server jobs with non-linear rewards. Before presenting the design details, we first give some preliminary definitions and analysis.

3.1 Preliminaries

Definition 1. Nice Setup. *If all the utilities $\{f_k\}_{k \in \mathcal{K}}$ are (i) linearly separable over computing instances, i.e.,*

$$f_k\Big(\sum_{r \in \mathcal{R}_l} y_{(l,r)}^k\Big) = \sum_{r \in \mathcal{R}_l} f_r^k\Big(y_{(l,r)}^k\Big), \tag{12}$$

and each concave utility $f_r^k(\cdot)$ is (ii) continuously differentiable in $\mathbb{R}_+$, and (iii) there exist $\varpi_r^k > 0$ such that

$$(f_r^k)'(0) \leq \varpi_r^k, \forall r, k, \tag{13}$$

we say this is a nice setup.

The following proposition demonstrates the property of the regret minimization problem, which will be used in the design and analysis of OgaSched.

Proposition 1. Convexity. *(i) The feasible solution space $\mathcal{Y}$ is convex. (ii) With a nice setup, at each time t, the single-slot reward function $q\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big)$ is a concave function of $\boldsymbol{y}(t)$.*

Proof. In the following proof, we just drop (t) from $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$ for simplification. Besides, we only prove the case that $\mathcal{G}$ is right d-regular and $d = |\mathcal{L}|$. The left cases can be easily proved with the same techniques used in this proof.

We firstly prove (i). To do this, let us arrange the vector $\boldsymbol{y}$ as

$$\boldsymbol{y} = \Big[\underbrace{\boldsymbol{y}^1}_{k=1}; \underbrace{\boldsymbol{y}^2}_{k=2}; ...; \underbrace{\boldsymbol{y}^K}_{k=K}\Big], \tag{14}$$

where $\boldsymbol{y}^k \in \mathbb{R}^{(|\mathcal{L}| \times |\mathcal{R}|)}$ is arranged as

$$\boldsymbol{y}^k = \Big[\underbrace{y_{(1,1)}^k; ...; y_{(1,|\mathcal{R}|)}^k}_{l=1}; ...; \underbrace{y_{(|\mathcal{L}|,1)}^k; ...; y_{(|\mathcal{L}|,|\mathcal{R}|)}^k}_{l=|\mathcal{L}|}\Big]. \tag{15}$$

With this arrangement, the vector representation of (5) is

$$\mathbf{0} \leq \boldsymbol{y} \leq \boldsymbol{a}, \tag{16}$$

where $\boldsymbol{a} = \big[\boldsymbol{a}^1; ...; \boldsymbol{a}^K\big]$, and

$$\boldsymbol{a}^k = \Big[\underbrace{a_1^k; ...; a_1^k}_{\text{of size } |\mathcal{R}|}; ...; \underbrace{a_{|\mathcal{L}|}^k; ...; a_{|\mathcal{L}|}^k}_{\text{of size } |\mathcal{R}|}\Big]. \tag{17}$$

Similarly, we want to construct a matrix $\mathbf{B}$ and a vector $\boldsymbol{c}$ for the vector representation of (6). $\forall k \in \mathcal{K}$, we design $\mathbf{B}' \in \mathbb{R}^{|\mathcal{R}| \times (|\mathcal{L}| \times |\mathcal{R}|)}$ as

$$\big[\mathbf{B}'\big]_{ij} = \begin{cases} 1 & (j-i) \mid |\mathcal{R}| \\ 0 & \text{o.w.} \end{cases} \tag{18}$$

and $\boldsymbol{c}^k = \big[c_1^k; ...; c_{|\mathcal{R}|}^k\big]$. Then, we have

$$\mathbf{B}'\boldsymbol{y}^k \leq \boldsymbol{c}^k, \forall k. \tag{19}$$

We can transform (19) into

$$\mathbf{B}'\boldsymbol{y}^k + \sum_{k' \neq k} \mathbf{O}\boldsymbol{y}^{k'} \leq \boldsymbol{c}^k, \forall k, \tag{20}$$

where $\mathbf{O}$ is a zero matrix. As a result, (6) is equivalent to

$$\mathbf{B}\boldsymbol{y} \leq \boldsymbol{c}, \tag{21}$$

where $\boldsymbol{c} = \big[\boldsymbol{c}^1; ...; \boldsymbol{c}^K\big] \in \mathbb{R}^{(|\mathcal{R}| \times K)}$, and

$$\mathbf{B} = \operatorname{diag}\big(\mathbf{B}'\big) \in \mathbb{R}^{(|\mathcal{R}| \times K) \times (|\mathcal{L}| \times |\mathcal{R}| \times K)}.$$

The above analysis leads to $\mathcal{Y} = \{\boldsymbol{y} \mid \mathbf{0} \leq \boldsymbol{y} \leq \boldsymbol{a}, \mathbf{B}\boldsymbol{y} \leq \boldsymbol{c}\}$ being a polyhedron, which is well known to be a convex set.

We now prove (ii). Similarly, we try to find the vectorized representation of $q(\boldsymbol{x},\boldsymbol{y})$. To do this, we define the operator $\boldsymbol{f}:\mathbb{R}^{|\mathcal{L}|\times|\mathcal{R}|\times K}\to\mathbb{R}^{|\mathcal{L}|\times|\mathcal{R}|\times K}$ as

$$\boldsymbol{f}=\left[\boldsymbol{f}^1;...;\boldsymbol{f}^K\right], \tag{22}$$

where

$$\boldsymbol{f}^k=\Big[\underbrace{f_1^k;...;f_{|\mathcal{R}|}^k}_{\text{make }|\mathcal{L}|\text{ replicas}};...;\underbrace{f_1^k;...;f_{|\mathcal{R}|}^k}_{l=|\mathcal{L}|}\Big]. \tag{23}$$

Then, the first part of (7) can be transformed into

$$\begin{aligned}\sum_{l\in\mathcal{L}}x_l\sum_{k\in\mathcal{K}}f_k\Big(\sum_{r\in\mathcal{R}_l}y_{(l,r)}^k\Big)&=\sum_{l\in\mathcal{L}}\sum_{k\in\mathcal{K}}\sum_{r\in\mathcal{R}_l}x_lf_r^k\Big(y_{(l,r)}^k\Big)\\&=\boldsymbol{\chi}\cdot\boldsymbol{f}(\boldsymbol{y}),\end{aligned} \tag{24}$$

where

$$\boldsymbol{\chi}=\Big[\underbrace{\underbrace{x_1;...;x_1}_{l=1,\forall r};...;\underbrace{x_{|\mathcal{L}|};...;x_{|\mathcal{L}|}}_{l=|\mathcal{L}|,\forall r}}_{k=1,\text{ make }K\text{ replicas}};...;\underbrace{x_1;...;x_{|\mathcal{L}|}}_{k=K}\Big]. \tag{25}$$

For the second part of (7), we have

$$\sum_{l\in\mathcal{L}}x_l\max_{k\in\mathcal{K}}\Big\{\beta_k\sum_{r\in\mathcal{R}_l}y_{(l,r)}^k\Big\}=\sum_{l\in\mathcal{L}}x_l\beta_{k^*}\sum_{r\in\mathcal{R}_l}y_{(l,r)}^{k^*}, \tag{26}$$

where

$$k^*=\operatorname*{argmax}_{k\in\mathcal{K}}\Big\{\beta_k\sum_{r\in\mathcal{R}_l}y_{(l,r)}^k\Big\}. \tag{27}$$

Without loss of generality, we assume that $k^*=1$. Then, the second part can be represented as $\boldsymbol{\beta}\cdot\boldsymbol{y}$, where

$$\boldsymbol{\beta}=\Big[\underbrace{\underbrace{x_1\beta_1;...;x_1\beta_1}_{l=1,\forall r};...;\underbrace{x_{|\mathcal{L}|}\beta_1;....;x_{|\mathcal{L}|}\beta_1}_{l=|\mathcal{L}|,\forall r}}_{k^*=1};\underbrace{\boldsymbol{0}}_{\forall k\neq k^*}\Big]. \tag{28}$$

The above analysis leads to

$$q(\boldsymbol{x},\boldsymbol{y})=\boldsymbol{\chi}\cdot\boldsymbol{f}(\boldsymbol{y})-\boldsymbol{\beta}\cdot\boldsymbol{y}. \tag{29}$$

With the concavity of $f_r^k(\cdot)$, the result (ii) is immediate. □

As a result, the derivative of $q(\cdot)$ at time t is

$$\frac{\partial q(\boldsymbol{x}(t),\boldsymbol{y}(t))}{\partial y_{(l,r)}^k(t)}=\begin{cases}x_l(t)\Big((f_r^k)'(y_{(l,r)}^k(t))-\beta_k\Big) & k=k^*\\ x_l(t)(f_r^k)'(y_{(l,r)}^k(t)) & \text{o.w.,}\end{cases} \tag{30}$$

where k^* is defined by (27).

3.2 Online Gradient Ascent

In this section, we give the design details of the OGA-based bipartite scheduling policy.

Definition 2. THE OGA POLICY. *For any feasible initial bipartite scheduling decision $\boldsymbol{y}(1)\in\mathcal{Y}$, at each time $t\in\mathcal{T}$, the OGA policy gets $\boldsymbol{y}(t+1)$ in the direction of ascending the gradient of $q(\boldsymbol{x}(t),\boldsymbol{y}(t))$:*

$$\boldsymbol{y}(t+1)=\Pi_{\mathcal{Y}}\Big(\boldsymbol{y}(t)+\eta_t\nabla q(\boldsymbol{x}(t),\boldsymbol{y}(t))\Big), \tag{31}$$

where η_t is the step size, and

$$\Pi_{\mathcal{Y}}(\boldsymbol{z})=\operatorname*{argmin}_{\hat{\boldsymbol{y}}\in\mathcal{Y}}\big\|\hat{\boldsymbol{y}}-\boldsymbol{z}\big\|_2^2 \tag{32}$$

is the Euclidean projection of $\boldsymbol{z}$ onto $\mathcal{Y}$.

To implement the projection (32) with low complexity, we propose OGASCHED, which is a combination of the OGA policy and the following fast projection technique. Firstly, we introduce the Lagrangian of the projection (32) as

$$\begin{aligned}L(\hat{\boldsymbol{y}},\boldsymbol{\rho},\boldsymbol{\mu},\boldsymbol{\lambda})&=\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\in\mathcal{K}}\Big(\hat{y}_{(l,r)}^k-z_{(l,r)}^k\Big)^2\\&+\sum_{r\in\mathcal{R}}\sum_{k\in\mathcal{K}}\rho_r^k\Big(\sum_{l\in\mathcal{L}_r}\hat{y}_{(l,r)}^k-c_r^k\Big)-\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\in\mathcal{K}}\lambda_{l,r}^k\hat{y}_{(l,r)}^k\\&+\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\in\mathcal{K}}\mu_{l,r}^k\Big(\hat{y}_{(l,r)}^k-a_l^k\Big),\end{aligned} \tag{33}$$

where $\boldsymbol{\rho}$ is the dual variable for (6), $\boldsymbol{\mu}$ is the dual variable for $\boldsymbol{y}(t)\le\boldsymbol{a}$, and $\boldsymbol{\lambda}$ is the dual variable for $\boldsymbol{y}(t)\ge\boldsymbol{0}$. Then, we can write the KKT conditions of the projection as

$$\begin{cases}2(\hat{y}_{(l,r)}^k-z_{(l,r)}^k)+\rho_r^k-\lambda_{l,r}^k+\mu_{l,r}^k=0\\ \sum_{l\in\mathcal{L}_r}\hat{y}_{(l,r)}^k=c_r^k\ \&\ \rho_r^k>0\\ \hat{y}_{(l,r)}^k=a_l^k\ \&\ \mu_{l,r}^k>0\\ \hat{y}_{(l,r)}^k=0\ \&\ \lambda_{l,r}^k>0\end{cases} \tag{34}$$

for every l,r,k.

Our fast projection is implemented for each pair of (r,k) in parallel. Specifically, for each $r\in\mathcal{R}$ and each $k\in\mathcal{K}$, we divide the ports $l\in\mathcal{L}$ into three disjoint sets:

$$\begin{cases}\mathcal{B}_{rk}^1=\Big\{l\in\mathcal{L}_r\mid\forall(l,r,k):\hat{y}_{(l,r)}^k=a_l^k\Big\}\\ \mathcal{B}_{rk}^2=\Big\{l\in\mathcal{L}_r\mid\forall(l,r,k):\hat{y}_{(l,r)}^k=0\Big\}\\ \mathcal{B}_{rk}^3=\Big\{l\in\mathcal{L}_r\mid\forall(l,r,k):2(\hat{y}_{(l,r)}^k-z_{(l,r)}^k)+\rho_r^k=0\Big\},\end{cases}$$

where

$$\rho_r^k=\frac{2}{|\mathcal{B}_{rk}^3|}\Big(\sum_{l\in\mathcal{B}_{rk}^3}z_{(l,r)}^k-c_r^k+\sum_{l\in\mathcal{B}_{rk}^1}a_l^k\Big),\forall r,k. \tag{35}$$

The fast projection works by solving the equation system (34) iteratively. Specifically, for each pair of (r,k), we sort the elements of $\boldsymbol{z}_{(:,r)}^k$ in descending order (step 7), and initialize $\mathcal{B}_{rk}^1$ and $\mathcal{B}_{rk}^2$ as $\varnothing$ while initializing $\mathcal{B}_{rk}^3$ as $\mathcal{L}_r$ (step 10 and 12). Then, we repeat a loop, in which we calculate ρ_r^k with (35), and update the value of $\hat{y}_{(l,r)}^k$ for each port l in $\mathcal{B}_{rk}^3$ (step 25). Since the elements of $\boldsymbol{z}_{(:,r)}^k$ are sorted from largest to smallest, if some $\hat{y}_{(l,r)}^k<0$, we can derive that for all the $l'\in\mathcal{S}_{rk}:=\{l,...,|\mathcal{L}_r|\}$, we have $\hat{y}_{(l',r)}^k<0$. Thus, the resource allocation for all the ports in $\mathcal{S}_{rk}$ is illegal, since $\hat{y}_{(l,r)}^k\ge0$ must hold. As a result, we update the sets $\mathcal{B}_{rk}^2$ and $\mathcal{B}_{rk}^3$, and repeat the calculate loop again (step 29). The calculation loop stops when there are no illegal resource allocations, i.e., $\forall l\in\mathcal{L}_r$, we have $\hat{y}_{(l,r)}^k\ge0$. In other words, $\mathcal{S}_{rk}=\varnothing$. We call the calculation loop in step 18 $\sim$ step 30 the innner loop. The outer loop is the while loop defined in step 9. To exit the while loop, we need to guarantee that $\hat{y}_{(1,r)}^k\le a_1^k$. Otherwise, the resource allocation is also illegal. Note that here we only need to check for $l=1$ since the elements in $\boldsymbol{z}_{(:,r)}^k$ are sorted.

The number of projections is linearly proportional to the size of the solution's dimensions, i.e., $\sum_{l\in\mathcal{L}}|\mathcal{R}_l|\times K$. Nevertheless, as we have mentioned, we can do the projections for different combinations of r and k in parallel because they are not interwoven. Thus, the time complexity of the fast projection is of $\mathcal{O}(|\mathcal{L}|\times\log(K\sum_{l\in\mathcal{L}}|\mathcal{R}_l|))$ in each time slot, where the $\log(\cdot)$ operator comes from the sorting operation (step 7). The multiplier $|\mathcal{L}|$ outside $\log(\cdot)$ comes from the inner loop (step 19). In our experiments, the repeat loop's execution count is significantly less than the number of job types $|\mathcal{L}|$.

3.3 Regret Analysis

In this section, we discuss the regret of OGASCHED. The main result is summarized in Theorem 1.

Theorem 1. REGRET UPPER BOUND. *With a nice setup, the regret of* OGASCHED *is upper bounded by*

$$R_T^{\text{OGASCHED}} \le \sqrt{2T\sum_{k\in\mathcal{K}}\sum_{r\in\mathcal{R}}\bar{a}^k c_r^k} \times \sqrt{\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\Big((\beta^*)^2 + K(\varpi_r^*)^2\Big)}, \tag{36}$$

where $\bar{a}^k := \max_{l\in\mathcal{L}} a_l^k$, $\beta^* := \max_{k\in\mathcal{K}} \beta_k$, *and* $\varpi_r^* := \max_{k\in\mathcal{K}} \varpi_r^k$.

Proof. The result is based on the non-expansiveness property of Euclidean projection and the concavity of $\{f_r^k(\cdot)\}_{r,k}$. Our proof has two parts. The first part gives the general form of the upper bound, which is similar to Theorem 2.13 in [33] and Theorem 3 in [30]. Meanwhile, the second part gives the specific upper bounds of involved variables.

At each time $t > 1$, for the $\boldsymbol{y}(t)$ yielded by OGASCHED, we have

$$\begin{aligned}\|\boldsymbol{y}(t) - \boldsymbol{y}^*\|^2 &= \big\|\Pi_{\mathcal{Y}}\big(\boldsymbol{y}(t-1) + \eta_t\nabla q(t-1)\big) - \boldsymbol{y}^*\big\|^2 \\ &\overset{(i)}{\le} \|\boldsymbol{y}(t-1) - \boldsymbol{y}^*\|^2 + \eta_t^2\|\nabla q(t-1)\|^2 \\ &\quad + 2\eta_t\nabla q\big(\boldsymbol{y}(t-1)\big)^{\mathrm{T}}\big(\boldsymbol{y}(t-1) - \boldsymbol{y}^*\big),\end{aligned} \tag{37}$$

where $\nabla q\big(\boldsymbol{y}(t-1)\big)$ is a shorthand for $\nabla q\big(\boldsymbol{x}(t-1), \boldsymbol{y}(t-1)\big)$. (i) is because the non-expansiveness property of the Euclidean projection. By moving $\|\boldsymbol{y}(t-1) - \boldsymbol{y}^*\|^2$ to the LHS of (37) and summing the inequality telescopically over $\mathcal{T}$, we have

$$\begin{aligned}&\sum_{t=2}^{T+1}\nabla q(\boldsymbol{y}(t-1))^{\mathrm{T}}\big(\boldsymbol{y}^* - \boldsymbol{y}(t-1)\big) \\ \overset{(i)}{\le}\ & \frac{\eta\sum_{t=1}^{T}\|\nabla q(\boldsymbol{y}(t))\|^2}{2} + \frac{\|\boldsymbol{y}(1) - \boldsymbol{y}^*\|^2 - \|\boldsymbol{y}(T) - \boldsymbol{y}^*\|^2}{2\eta} \\ \overset{(ii)}{\le}\ & \frac{\eta T(\max\|\nabla q\|)^2}{2} + \frac{diam(\mathcal{Y})^2}{2\eta}.\end{aligned} \tag{38}$$

Inequality (i) is because $\forall t \in \mathcal{T}$ we set $\eta_t \equiv \eta$. In (ii), we use the fact that $\|\boldsymbol{y}(T) - \boldsymbol{y}^*\| \ge 0$. In (37), $\max\|\nabla q\|$ is the maximum Euclidean norm of the gradient of $q(\boldsymbol{x}(t), \boldsymbol{y}(t))$ over every possible $\boldsymbol{y}(t)$, and $diam(\mathcal{Y})$ is the largest Euclidean distance between any two elements of $\mathcal{Y}$. Because $q(\cdot)$ is a concave function of $\boldsymbol{y}(t)$, we have

$$\begin{aligned}R_T^{\text{OGASCHED}} &= \sup_{\forall\{\boldsymbol{x}(t)\}_1^T}\sum_{t=1}^{T}\Big(q\big(\boldsymbol{x}(t), \boldsymbol{y}^*\big) - q\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big)\Big) \\ &\le \sup_{\forall\{\boldsymbol{x}(t)\}_1^T}\sum_{t=1}^{T}\nabla q\big(\boldsymbol{y}(t)\big)^{\mathrm{T}}\big(\boldsymbol{y}^* - \boldsymbol{y}(t)\big) \qquad \triangleright (37) \\ &\le \frac{diam(\mathcal{Y})^2}{2\eta} + \frac{\eta T(\max\|\nabla q\|)^2}{2}.\end{aligned} \tag{39}$$

In the following, we give the upper bound of $\max\|\nabla q\|$ and *diam*$(\mathcal{Y})$, respectively.

Algorithm 1: OGASCHED

Input: Graph $\mathcal{G}$, requirements $\boldsymbol{a}$, capacities $\boldsymbol{c}$, and the decay λ
Output: Scheduling decisions $\{\boldsymbol{y}(t)\}_{t\in\mathcal{T}}$

Initialize $\boldsymbol{y}(1) \in \mathcal{Y}$ and η_0
for *t from* 1 *to* T **do**
  Observe the job arrival status $\boldsymbol{x}(t)$
  Calculate the gradient $\nabla q\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big)$ with (30)
  $\boldsymbol{z}(t+1) \leftarrow \boldsymbol{y}(t) + \eta_t\nabla q\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big)$
  foreach (r, k) *in* `zip`$(\mathcal{R}, \mathcal{K})$ **do in parallel**
    Sort the elements of $\boldsymbol{z}_{(:,r)}^k(t+1)$ in descending order
    initialized $\leftarrow$ `False`
    while `True` **do**
      $\mathcal{B}_{rk}^2 \leftarrow \varnothing$
      if not *initialized* **then**
        $\mathcal{B}_{rk}^1 \leftarrow \varnothing, \mathcal{B}_{rk}^3 \leftarrow \mathcal{L}_r, \hat{\boldsymbol{y}} \leftarrow \boldsymbol{0}$
        initialized $\leftarrow$ `True`
      else
        if $\hat{y}_{(1,r)}^k > a_1^k$ **then**
          $\mathcal{B}_{rk}^1 \leftarrow \{1\}, \mathcal{B}_{rk}^3 \leftarrow \mathcal{L}_r\backslash\{1\}$
        else
          break
      repeat
        Calculate ρ_r^k with (35)
        for $l \in \mathcal{L}_r$ **do**
          if $l \in \mathcal{B}_{rk}^1$ **then**
            $\hat{y}_{(l,r)}^k \leftarrow a_l^k$
          else if $l \in \mathcal{B}_{rk}^3$ **then**
            $\hat{y}_{(l,r)}^k \leftarrow z_{(l,r)}^k(t+1) - \rho_r^k/2$
            if $\hat{y}_{(l,r)}^k < 0$ **then**
              $\mathcal{S}_{rk} \leftarrow \Big\{l, l+1, ..., |\mathcal{L}_r|\Big\}$
              break
        `// Update then re-calculate`
        $\mathcal{B}_{rk}^2 \leftarrow \mathcal{B}_{rk}^2 \cup \mathcal{S}_{rk}, B_{rk}^3 \leftarrow B_{rk}^3\backslash\mathcal{S}_{rk}$
      until $\mathcal{S}_{rk} = \varnothing$;
  $\boldsymbol{y}(t+1) \leftarrow \hat{\boldsymbol{y}}$
  $\eta_{t+1} \leftarrow \lambda\eta_t$ `// Update learning rate`
return *the sequence of decisions* $\{\boldsymbol{y}(t)\}_{t\in\mathcal{T}}$

---

1) *The upper bound of* $\max\|\nabla q\|$. With the result of (30), we have

$$\begin{aligned}\|\nabla q\|^2 &= \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\left[x_l(t)^2\Big((f_r^{k^*})'\big(y_{(l,r)}^{k^*}(t)\big) - \beta_{k^*}\Big)^2\right] \\ &\quad + \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\neq k^*} x_l(t)^2(f_r^k)'\Big(y_{(l,r)}^k(t)\Big)^2 \\ &= \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l} x_l(t)^2\Bigg[\sum_{k\in\mathcal{K}}\Big((f_r^k)'\big(y_{(l,r)}^k(t)\big)\Big)^2 \\ &\quad - 2\beta_{k^*}(f_r^{k^*})'\big(y_{(l,r)}^{k^*}(t)\big)\Bigg] + \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l} x_l(t)^2\beta_{k^*}^2.\end{aligned} \tag{40}$$

where k^* is defined in (27). The second part of (40) can be upper bounded by

$$\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l} x_l(t)^2\beta_{k^*}^2 \le \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}(\beta^*)^2, \tag{41}$$

where $\beta^* = \max_{k\in\mathcal{K}}\beta_k$. If $\mathcal{G}$ is right d-regular, the bound reduces to $d|\mathcal{R}|(\beta^*)^2$. For the first part of (40), we use $(f_r^{k^*})'$ to replace $(f_r^{k^*})'(y_{(l,r)}^{k^*}(t))$ for simplification. Then we have

$$\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l} x_l(t)^2\Big[\sum_{k\in\mathcal{K}}\big((f_r^k)'\big)^2 - 2\beta_{k^*}(f_r^{k^*})'\Big]$$
$$\le \underbrace{\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\neq k^*}\big((f_r^k)'\big)^2}_{\text{PART-A}} + \underbrace{\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}(f_r^{k^*})'\big((f_r^{k^*})' - 2\beta_{k^*}\big)}_{\text{PART-B}}.$$

For PART-A we have

$$\text{PART-A} \le (K-1)\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}(\varpi_r^*)^2, \tag{42}$$

where $\varpi_r^* = \max_{k\in\mathcal{K}}\varpi_r^k$. If $\mathcal{G}$ is right d-regular, the bound reduces to $d|\mathcal{R}|(K-1)(\varpi_r^*)^2$. To analyze the upper bound of PART-B, we need to partition the computing instances into two disjoint sets:

$$\mathcal{R}_1 = \Big\{r\in\mathcal{R} : \varpi_r^{k^*} \le 2\beta_{k^*}\Big\}$$
$$\mathcal{R}_2 = \Big\{r\in\mathcal{R} : \varpi_r^{k^*} > 2\beta_{k^*}\Big\}.$$

For each $r\in\mathcal{R}_1$, the maximum of $(f_r^{k^*})'\big((f_r^{k^*})' - 2\beta_{k^*}\big)$ is 0 since $(f_r^{k^*})' \ge 0$ holds. For each $r\in\mathcal{R}_2$, the maximum is $(\varpi_r^{k^*})^2 - 2\beta_{k^*}\varpi_r^{k^*}$. Thus,

$$\text{PART-B} \le \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l\cap\mathcal{R}_2}\Big((\varpi_r^{k^*})^2 - 2\beta_{k^*}\varpi_r^{k^*}\Big). \tag{43}$$

Recall that in (43) $\mathcal{R}_l$ is the set of computing instances that connects to port l. Because $\beta_k\in[0,1]$ holds for each $k\in\mathcal{K}$, $\forall l\in\mathcal{L}, r\in\mathcal{R}_l\cap\mathcal{R}_2$, we have

$$(\varpi_r^{k^*})^2 - 2\beta_{k^*}\varpi_r^{k^*} \le (\varpi_r^*)^2 - 2\beta_{k^*}\varpi_r^* \le (\varpi_r^*)^2, \tag{44}$$

Finally, we can get

$$\|\nabla q\|^2 \le \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\Big((\beta^*)^2 + K(\varpi_r^*)^2\Big). \tag{45}$$

For the upper bound in (45), all the computing instances $r\in\mathcal{R}_l$ fall into the set $\mathcal{R}_2$.

2) *The upper bound of diam($\mathcal{Y}$)*. By definition we have

$$\mathit{diam}(\mathcal{Y}) = \sup_{\boldsymbol{y},\boldsymbol{z}\in\mathcal{Y}}\|\boldsymbol{y}-\boldsymbol{z}\|. \tag{46}$$

To find the upper bound of $\|\boldsymbol{y}-\boldsymbol{z}\|$, we can get

$$\begin{aligned}\|\boldsymbol{y}-\boldsymbol{z}\|^2 &= \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\in\mathcal{K}}\Big(y_{(l,r)}^k - z_{(l,r)}^k\Big)^2\\ &\overset{(i)}{\le} \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\in\mathcal{K}}\big|y_{(l,r)}^k - z_{(l,r)}^k\big|\cdot a_l^k\\ &\le \sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\sum_{k\in\mathcal{K}} a_l^k\Big(y_{(l,r)}^k + z_{(l,r)}^k\Big)\\ &\le \sum_{k\in\mathcal{K}}\bar{a}^k\sum_{r\in\mathcal{R}}\Big(\sum_{l\in\mathcal{L}_r}y_{(l,r)}^k + \sum_{l\in\mathcal{L}_r}z_{(l,r)}^k\Big)\\ &\overset{(ii)}{\le} 2\sum_{k\in\mathcal{K}}\bar{a}^k\sum_{r\in\mathcal{R}}c_r^k,\end{aligned} \tag{47}$$

where $\bar{a}^k = \max_{l\in\mathcal{L}}a_l^k$. In (47), (i) is because the constraint (5). In (ii), we use the capacity constraint (6). As a result, we have

$$\mathit{diam}(\mathcal{Y}) \le \sqrt{2\sum_{k\in\mathcal{K}}\bar{a}^k\sum_{r\in\mathcal{R}}c_r^k}. \tag{48}$$

Combing the result (45) and (48), and set η as $\frac{\mathit{diam}(\mathcal{Y})}{\|\nabla q\|\sqrt{T}}$, we finally get the result. □

The theorem shows that the suboptimality gap between OGASCHED and the offline optimal is of $\Theta(\mathcal{H}_\mathcal{G}\times\sqrt{T})$, where

$$\mathcal{H}_\mathcal{G} := \sqrt{2\sum_{k\in\mathcal{K}}\sum_{r\in\mathcal{R}}\bar{a}^k c_r^k} \times \sqrt{\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l}\Big((\beta^*)^2 + K(\varpi_r^*)^2\Big)} \tag{49}$$

is a factor characterized the scale of the bipartite graph $\mathcal{G}$. In addition, we can find that the regret grows sublinearly with the number of job types $|\mathcal{L}|$. To the best of our knowledge, this is the best regret for the online bipartite scheduling problem with non-linear rewards. The proof also indicates that, to achieve a not-too-bad cumulative reward, at each time t, the learning rate η_t should be set as

$$\eta_t = \frac{\mathit{diam}(\mathcal{Y})}{\|\nabla q(\boldsymbol{x}(t),\boldsymbol{y}(t))\|\sqrt{T}}. \tag{50}$$

3.4 Extending to Multiple Job Arrivals

OGASCHED can be applied to the scenarios where multiple jobs are yielded from each port in each time slot. In this case, the job arrival status $\boldsymbol{x}(t)$ is re-formulated as $\boldsymbol{x}(t) = \big[x_l(t)\big]_{l\in\mathcal{L}} \in \mathbb{N}^{|\mathcal{L}|}$, where $x_l(t)$ indicates the number of jobs arrive at port l at time t. Further, the scheduling decisions at time t is re-formulated as

$$\boldsymbol{y}(t) = \Big[y_{(l,r)}^{j,k}\Big]_{l,j,r,k} \in \mathbb{R}^{\sum_{l\in\mathcal{L}}J_l\times|\mathcal{R}_l|\times K},$$

where J_l is the maximum number of the type-l jobs arrive during each time slot, i.e. $J_l = \max_{t\in\mathcal{T}}x_l(t)$. Correspondingly, the port-l reward is re-formulated as

$$q_l(\boldsymbol{x}(t),\boldsymbol{y}(t)) = \sum_{j=1}^{J_l}\mathbb{1}\{j\le x_l(t)\}\Big[\sum_{k\in\mathcal{K}}f_k\Big(\sum_{r\in\mathcal{R}_l}y_{(l,r)}^{j,k}(t)\Big) - \max_{k\in\mathcal{K}}\Big\{\beta_k\sum_{r\in\mathcal{R}_l}y_{(l,r)}^{j,k}(t)\Big\}\Big],$$

where $\mathbb{1}\{p\}$ is the indicator function: $\mathbb{1}\{p\}$ is 1 if the predicate p is true, otherwise 0. The new formulated problem can be solved by native OGASCHED after transformations.

3.5 Extending to Gang Scheduling

OGASCHED can be extended to the Gang Scheduling scenarios, where the scheduling decisions for the task instances of a job follows the ALL-OR-NOTHING property. In other words, only when *all* tasks[1] of a job are successfully scheduled, the job could be launched.

In the following, we show briefly how Gang Scheduling can be modeled. To start with, for each job type $l\in\mathcal{L}$, we denote the corresponding set of task components by $\mathcal{Q}_l$

1. In practice, not all tasks of a job need to be scheduled. In Kubernetes, the job submitter can specify the minimum number of tasks that must be scheduled successfully. In the following, we use $m_l(t)$ to represent the minimum number of tasks that should be scheduled at time t of the type-l job.

and indexed by q. Correspondingly, the job requests $\boldsymbol{a}_l$ is redefined as

$$\boldsymbol{a}_l = \left[a_l^{q,k}\right]_{l,q,k} \in \mathbb{R}_{\geq 0}^{\sum_{l\in\mathcal{L}}|\mathcal{Q}_l|\times K}.$$

Similarly, we redefine the scheduling decisions at time t as

$$\boldsymbol{y}(t) = \left[y_{(l,r)}^{q,k}\right]_{l,q,r,k} \in \mathbb{R}_{\geq 0}^{\sum_{l\in\mathcal{L}}|\mathcal{Q}_l|\times|\mathcal{R}_l|\times K}.$$

As a result, the solution space $\mathcal{Y}$ turns to

$$\begin{aligned}\mathcal{Y} = \Big\{y_{(l,r)}^{q,k} \mid \sum_{q\in\mathcal{Q}_l} \mathbb{1}\{\sum_{r\in\mathcal{R}_l}\sum_{k\in\mathcal{K}} y_{(l,r)}^{q,k} > 0\} \geq m_l(t), \forall l,\\ 0 \leq y_{(l,r)}^{q,k}(t) \leq a_l^{q,k}, \forall l, r, q, k, t,\\ \sum_{l\in\mathcal{L}_r}\sum_{q\in\mathcal{Q}_l} y_{(l,r)}^{q,k}(t) \leq c_r^k, \forall r, k, t\Big\},\end{aligned}$$

where in the first inequality, $m_l(t)$ is the minimum number of task components that should be scheduled at time t of type-l job. The port-l reward at time t is re-formulated as

$$\begin{aligned}q_l\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big) = x_l(t)\Big[\sum_{k\in\mathcal{K}} f_k\Big(\sum_{q\in\mathcal{Q}_l}\sum_{r\in\mathcal{R}_l} y_{(l,r)}^{q,k}(t)\Big) -\\ \max_{k\in\mathcal{K}}\Big\{\beta_k \sum_{q\in\mathcal{Q}_l}\sum_{r\in\mathcal{R}_l} y_{(l,r)}^{q,k}(t)\Big\}\Big].\end{aligned}$$

The new formulated problem is more difficult because $\mathcal{Y}$ is no longer a convex set and $q_l\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big)$ is not differentiable everywhere. Nevertheless, we can still develop a similar online scheduling algorithm with the subgradient ascent and mirror ascent related techniques which retains a sublinear regret. The design detail is omitted due to space limits.

4 Experimental Results

In this section, we conduct extensive experiments to validate the performance of OGASCHED. Based on the Alibaba cluster trace datasets [34], we first examine the theoretically guaranteed superiority of OGASCHED against several baselines on the cumulative and average rewards. Then, we analyze the generality and robustness of it under different cluster settings. At last, we validate the efficacy of OGASCHED in large-scale scenarios. The trace-driven simulation is conducted on a server with 48 Intel Xeon Silver 4214 CPUs, 256 GB memory, and 2 Tesla P100 GPUs.

Traces. We hybrid the traces from cluster-trace-v2018 and cluster-trace-gpu-v2020 of the Alibaba Cluster Trace Program. Specifically, we leverage the specifications of the machines, the arrival patterns, and the resource requirements of different kinds of jobs to generate our simulation environment.

Baselines. The following widely used baselines are implemented to make comparisons with OGASCHED.

- DRF [20]. It is adopted by YARN [35] and Mesos [36]. In our scenario, DRF allocates resources to ports that yield jobs in the ascending order of their dominant resource shares. The dominant share s_l of port l is calculated as $s_l = \max_{k\in\mathcal{K}}\{a_l^k / \sum_{r\in\mathcal{R}_l} c_r^k\}$.
- FAIRNESS. We implement FAIRNESS in this way: at each time t, we allocate the type-k resource of each node r to each port l that yield a job according to the job's share $a_l^k / \sum_{l\in\mathcal{L}_r} a_l^k$.
- BINPACKING. It is optional in Kubernetes with the name of MOSTALLOCATED strategy and supported in Volcano as a configurable plugin [37]. Specifically, it scores the computing instances based on the utilization of resources, favoring the ones with higher allocation.
- SPREADING. It is similar to BINPACKING in procedures but with an opposite favor. The nodes with lower utilizations of resources have higher scores.

Default Settings. In default settings, our simulation environment has 128 computing instances, each equipped with 6 types of resources (CPUs, MEM, GPUs, NPUs, TPUs, and FPGAs), and 10 job types of different resource requirements. Large-scale validations will be demonstrated in Sec. 4.3. The computing instances and jobs are carefully selected from the trace to reflect heterogenity. We support 4 types of utilities:

$$f_r^k(y) = \begin{cases} \alpha y & \textit{linear} \\ \alpha\ln(y+1) & \textit{log} \\ \alpha^{-1} - (y+\alpha)^{-1} & \textit{reciprocal} \\ \alpha\sqrt{y+1} - \alpha & \textit{poly}, \end{cases} \tag{51}$$

The default settings of main parameters are listed in Tab. 2. In this table, the initial learning rate and the decay are used to tune the learning rate at each time t around the value (50). Job arrival probability ρ is adopted to adjust the job arrival status with Bernoulli Distributions. This parameter is applied based on the actual arrival patterns from the trace to increase stochasticity. The contention level, located at the last cell of this table, is designed to tune the level of resource contention. The larger this value, the more fierce the contention. It is a multiplier to the resource requirements of jobs. The effect of it will be analyzed in detail in Sec. 4.2.

TABLE 2
Default parameter settings.

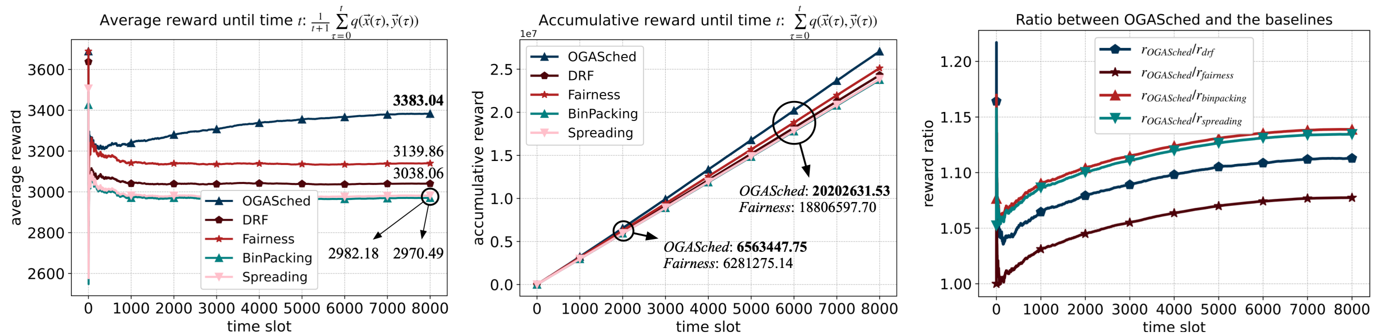
PARAMETER	VALUE	PARAMETER	VALUE
job types num. $\lvert\mathcal{L}\rvert$	10	node num. $\lvert\mathcal{R}\rvert$	128
device type num. K	6	time slot num. T	2000
range of α	$[1.0, 1.5]$	range of β	$[0.3, 0.5]$
initial learning rate η_0	25	decay λ	0.9999
job arrival prob. ρ	0.7	contention level	10

Note that in Sec. 4.1, the time slot length T is set as 8000. For the left experiments, the time slot length is 2000, unless otherwise stated.

4.1 Performance Verification

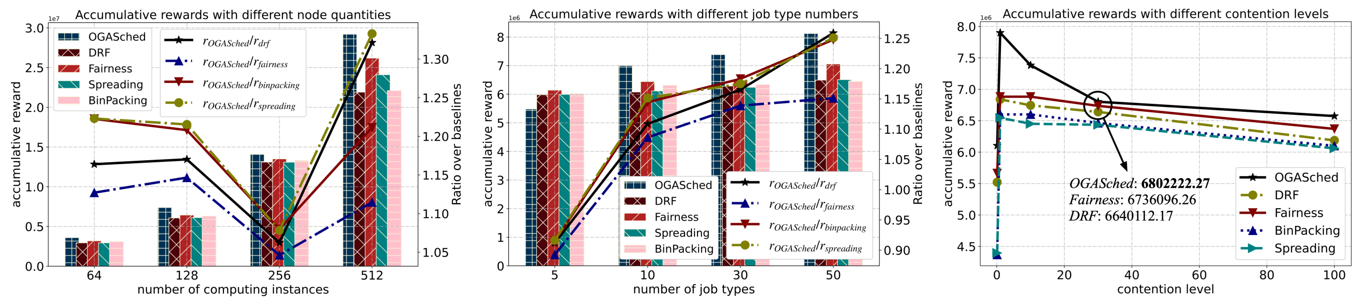
In this section, we compare the performance of OGASCHED with the baselines in terms of the achieved cumulative and average rewards.

In Fig. 2(a), the y-axis is the average reward unitl time t, i.e., $\frac{1}{t}\sum_{\tau=1}^{t} q\big(\boldsymbol{x}(\tau), \boldsymbol{y}(\tau)\big)$. Compared with the baselines, OGASCHED has a clear advantage on the performance (with the increases of 11.33%, 7.75%, 13.89%, and 13.44% compared with DRF, FARINESS, BINPACKING, and SPREADING, respectively). Besides, it shows that the performance of OGASCHED tends to increase as the length of the time horizon increases. The curve of OGASCHED starts steep and later flattens. The reason is that, as a learning-powered algorithm, OGASCHED learns the underlying distribution



(a) Average rewards. (b) Accumulative rewards. (c) Ratios between OGASCHED and baselines.

Fig. 2. Performance verification of OGASCHED. It takes one hour for OGASCHED to finish when $T = 8000$, $\beta \in [0.4, 0.6]$, and contention level is 11.



(a) Accumulative rewards vs. $|\mathcal{R}|$. (b) Accumulative rewards vs. $|\mathcal{L}|$. (c) Accumulative rewards vs. contention.

Fig. 3. Scalability verification of OGASCHED under different scales of the bipartite graph and the contention levels.

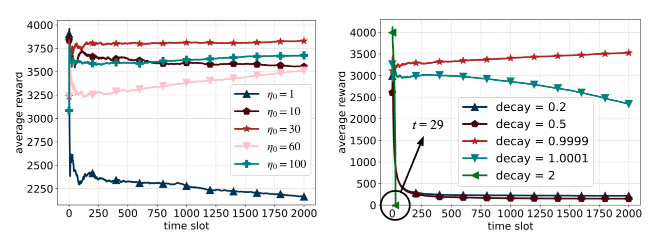
TABLE 3
Generality and Robustness validation under different scenario settings.

Avg. Reward	Time Horizon Length T 1000	2000	5000	10000	Job Arrival Probability ρ 0.3	0.5	0.7	0.9	Graph Dense ≈ 2	≈ 2.5	≈ 3
OGASCHED	**2578.53**	**2886.33**	**2911.37**	**3104.98**	**1904.87**	**2154.18**	**3117.29**	**2938.22**	**2816.18**	**2904.51**	**3127.47**
DRF	2422.47	2493.02	2449.23	**2497.85**	**1364.53**	**2086.59**	2503.01	2755.41	2417.08	2786.94	2795.42
FAIRNESS	**2532.24**	**2582.80**	**2552.41**	2436.22	1295.53	1997.19	**2628.02**	**2873.84**	**2501.54**	**2857.60**	**2918.98**
BINPACKING	2386.01	2449.15	2444.32	2365.13	1246.39	1897.79	2518.98	2740.19	2374.31	2757.71	2829.19
SPREADING	2382.01	2466.71	2436.60	2362.88	1250.67	1888.06	2519.37	2737.93	2382.87	2766.07	2836.37

of job arrival patterns and it can make better decisions by adjusting the step directions. It is interesting to find that the rewards oscillate at the beginning time slots. One of the leading factors is that OGASCHED is not boosted with a well-designed initial solution. In our experiments, a 8000-time slot training only takes one hour. Thus, not surprisingly, the rewards achieved in the beginning can be easily surpassed when the time slot is sufficiently large.

It is not a surprise that FAIRNESS achieves the best among the baselines. FAIRNESS adopts a proportional allocation strategy and allocates resources to each non-empty port *without bias*, which increases the computation gains adequately. When the contention is not fierce while the communication overhead is low, the advantages of FAIRNESS will be more steady. By contrast, the advantages of BINPACKING and SPREADING are respectively high resource utilization and job isolation, which do not contribute to the reward directly.

Fig. 2(b) shows that the cumulative rewards achieved by all the five algorithms. In the beginning, FAIRNESS and DRF have the slight edge, benefiting by the propotional allocation idea. Nevertheless, as the time slot increases, OGASCHED is able to surpass them without difficulty. Fig. 2(c) demonstrates the ratio on the achieved average rewards between OGASCHED and the baselines. Similarly, the ratios oscillate at the beginning. After that, they increase steeply and later flattens.



(a) Impact of η_0. (b) Impact of decay.

Fig. 4. The performance of OGASCHED with different hyper-parameters.

The hyper-parameters of OGASCHED, especially the initial learning rate η_0 and the decay, have a remarkable impact on its performance. From Fig. 4 we can find that, a wrong setting of these hyper-parameters could lead to a poor performance, even the decrease of the cumulative reward (which means, the reward is negative in many time slots). At the last of Sec. 3.3, we claim that, to achieve an cumulative reward with a lower bound guarantee, at each time t, the learning rate should be set around (50). Note that in (50), the learning rate is encouraged to be larger and larger as time moves, which is counterintuitive and it goes against the convergence to a local optimum. The curves in Fig. 4(b) also verify that, setting decay as 0.9999 is better than 1.0001. The best decay in practice does not follow the guidance of theory because the regret analysis only gives the *worst* case guarantee on the cumulative rewards. In our experiments, the best range for decay is $[0.995, 0.9999]$.

4.2 Scalability, Generality and Robustness Evaluations

In this section, we evaluate the performance of OGASCHED under different scales of scenario settings. Fig. 3(a) and Fig. 3(b) demonstrate the impact of the scale of the bipartite graph $\mathcal{G}$. In these two figures, the left y-axis is the cumulative reward while the right y-axis is the ratio r_a/r_b, where r_a is the cumulative reward achieved by OGASCHED, and r_b is the baselines'. Firstly, we observe that, whatever the number of the computing instances is, OGASCHED takes the leading position. Besides, as $|\mathcal{R}|$ increases, all the algorithms obtain a larger cumulative reward. The result is evident because a large cluster can provide sufficient resources, which leads to jobs being fully served. It is also worth noting that, when $|\mathcal{R}|$ increases, the superiority of OGASCHED over the baselines firstly increases then decreases. It demonstrates that the resource contention is fierce when $|\mathcal{R}| \in [128, 256]$. In this case, it is necessary for OGASCHED to be trained with a larger time slot. Fig. 3(b) shows that the number of job types, i.e., $|\mathcal{L}|$, has a weaker impact than $|\mathcal{R}|$ to the performance of OGASCHED. The phenomenon verifies the conclusion we have concluded, i.e., the regret grows linearly with $|\mathcal{R}|$, but it is sublinear with $|\mathcal{L}|$.

Fig. 3(c) shows the impact of contention level. This parameter works as a multiplier to the resource requirements of jobs. We can observe that, when moving contention level from 0.1 to 1, all the achieved cumulative rewards increase. This is obvious because a larger resource requirement leads to a larger computation gain on the premise of low contention. However, increasing the multiplier further leads to the downgrade of performances and the reduction of the superiority of OGASCHED. Even so, OGASCHED always performs the best. Fig. 6 shows the average computation gain and communication overhead penalty of each time slot under different contention levels. We can find that the penalty increases with the contention level slowly.

Fig. 7 demonstrates the cumulative rewards with different utilities. Because of the diminishing marginal effect, the rewards with *ploy*, *log*, and *reciprocal* utilities are significantly less than the rewards with *linear* utilities. Nevertheless, the diminishing marginal effect does not change the superiority of OGASCHED against the baselines. Even in the *all reciprocal* utility settings, for FAIRNESS, OGASCHED has its advantages.

In addition to the above evaluations, we also test the generality and robustness of OGASCHED under different settings of the following parameters: the time horizon length T, the job arrival probability ρ, and the dense of the bipartite graph. The graph dense is calculated as $\sum_{r\in\mathcal{R}} |\mathcal{L}_r|/|\mathcal{R}|$. The results are shown in Tab. 3. The two largest values in each column of the table are made bold. Besides, for each parameter and each algorithm, the setting which leads to the largest reward is marked with a light-grey background. We summarize the key findings as follows.

- Firstly, whatever the parameter settings, OGASCHED always performs the best, and its performance has a positive correlation with the time horizon length T. As we have analyzed, a large time horizon provides more chances for OGASCHED to learn the underlying distributions, thereby increasing the reward in the gradient ascent directions.
- Increasing the job arrival probability can lead to a high resource utilization, thereby increasing the rewards. However, a large job arrival probability also brings in a fierce resource contention. A direct consequence of it is that, for OGASCHED, many elements in the vector $\boldsymbol{y}(t)$ fall into the interior of $\mathcal{Y}$, rather than the boundaries, thereby leading to a reward reduction. The phenomenon can be observed when moving ρ from 0.7 to 0.9.
- Graph dense has a similar effect on the reward to the job arrival probability. Nevertheless, the reasons behind are distinct. A larger graph dense increases the opportunities for a job to be served with a large possible parallelism, thereby increasing the computation gain. By contrast, the communication overhead has a slow rate of growth.

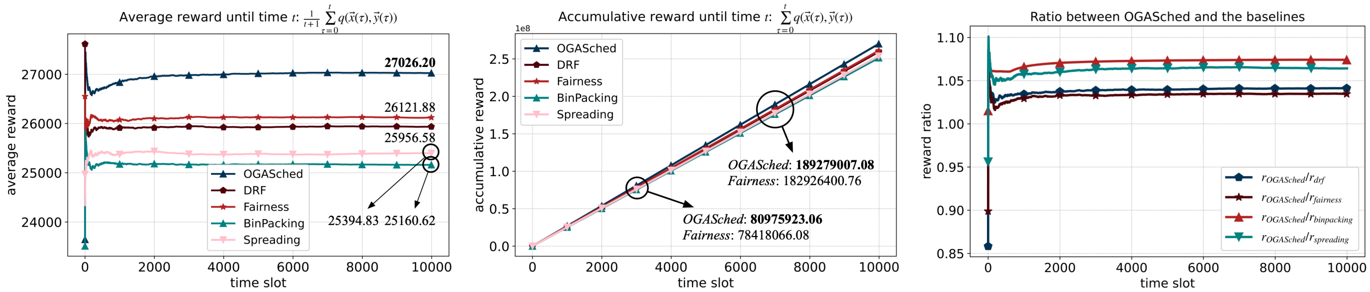
4.3 Large-Scale Validations

To test the efficacy of OGASCHED in large-scale scenarios, we conduct the following experiments. In these experiments, the number of the job types is set as 100 while the quantity of the computing instances is 1024 in default. The results in Fig. 5 show that the superiority of OGASCHED is preserved even in large-scale scenarios.

5 RELATED WORKS

The design of online job scheduling algorithms that yield a nice theoretical bound is always the focus of attention from the research community. Existing online job scheduling algorithms can be organized into two categories.

In the first category, the online algorithms are elaborately designed for specific job types, such as DNN model training [38], [39], [25], [6], [18], [13], [40], big-data query & analytics [12], [41], multi-stage workflows [42], [17], [43], [44], etc. A typical work on DNN model training is [18], where the authors fully take the layered structure of DNNs into consideration and develop an efficient resource scheduling algorithm based on the sum-of-ratios multi-type-knapsack decomposition method. The authors further prove that the proposed algorithm has a SOTA approximation ratio within a polynomial running time. [13] is another work that fully explores the Bulk Synchronous Parallel (BSP) property of the DNN training jobs. The authors develop an algorithm which is $\mathcal{O}(\ln|\mathcal{M}|)$-approximate with high probability, where $\mathcal{M}$ is the set of resources. These works are designed for specific job types, and they do not provide a general analysis of



(a) Average rewards. (b) Accumulative rewards. (c) Ratios over the baselines.

Fig. 5. Large-scale validations. It takes 15 hours for OGASCHED to complete when $T = 10000$, $\beta \in [0.01, 0.015]$, and contention level is 5.

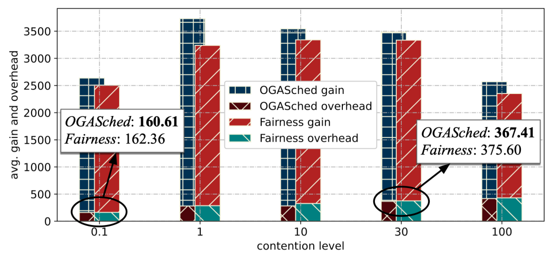


Fig. 6. Average computation gain and communication overhead of each time slot under different contention levels.

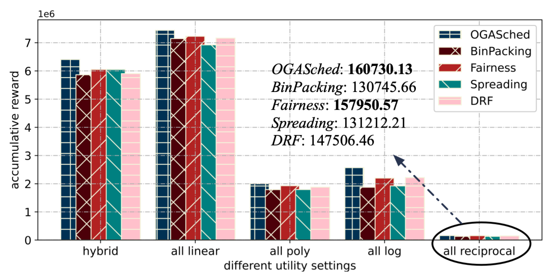


Fig. 7. Accumulative rewards with different utilities.

the gain-overhead tradeoff for multi-server jobs. This paper intends to fill the gap.

In the second category, the types of job are not specified, while the theoretical superiority is highlighted. The algorithms are designed with different theoretical basis, including online approximate algorithms [45], [15], [46], Online Convex Optimization (OCO) techniques [14], game-theoretical approaches [16], online learning and DRL-based algorithms [47], [48], etc. In these works, the performance of the proposed algorithms is usually analyzed with approximate ratio, competitive ratio, Price of Anarchy (PoA), and regret. A typical recent work is [14]. The authors develop an algorithm whose dynamic regret is upper bounded by $\mathcal{O}(\text{OPT}^{1-\beta})$, where $\beta \in [0, 1)$. None of the existing works analyze the gain-overhead tradeoff and provide a regret of $\mathcal{O}(\sqrt{|\mathcal{L}|T})$ as this paper demonstrates.

6 Conclusions

In this paper, we study the online scheduling of multi-server jobs in terms of the gain-overhead tradeoff. The problem is formulated as an cumulative reward maximization program. The reward of scheduling a job is designed as the difference between the computation gain and the penalty on the dominant communication overhead. We propose an algorithm, i.e. OGASCHED, to learn the best possible scheduling decision in the ascending direction of the reward gradients. OGASCHED is the first algorithm that has a sublinear regret w.r.t. the number of job types and time slot length, which is a SOTA result for concave rewards. OGASCHED is well designed to be parallelized, which makes large-scale applications possible. The superiority of OGASCHED is also validated with extensive trace-driven simulations. Future extensions may include, i.e., more elaborate modeling and analysis of the intra-node and inter-node communication overheads.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4500100, the National Science Foundation of China under Grants 62125206 and U20A20173, and the Key Research Project of Zhejiang Province under Grant 2022C01145.

References

[1] M. Wu, F. Yang, J. Xue, W. Xiao, Y. Miao, L. Wei, H. Lin, Y. Dai, and L. Zhou, "Gram: Scaling graph computation to the trillions," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 408–421. [Online]. Available: https://doi.org/10.1145/2806777.2806849

[2] L. WANG, W. WANG, and B. LI, "Cmfl: Mitigating communication overhead for federated learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 954–964.

[3] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Meng, and W. Lin, "Dl2: A deep learning-driven scheduler for deep learning clusters," *IEEE Transactions on Parallel & Distributed Systems*, vol. 32, no. 08, pp. 1947–1960, aug 2021.

[4] W. Wang, Q. Xie, and M. Harchol-Balter, "Zero queueing for multi-server jobs," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 1, Feb. 2021.

[5] H. Zhao, S. Deng, F. Chen, J. Yin, S. Dustdar, and A. Y. Zomaya, "Learning to schedule multi-server jobs with fluctuated processing speeds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 234–245, 2023.

[6] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 495–503.

[7] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys (CSUR)*, 2022.

[8] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 463–479. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/jiang

[9] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing, J. E. Gonzalez, and I. Stoica, "Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 559–578. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

[10] X. Jia, L. Jiang, A. Wang, W. Xiao, Z. Shi, J. Zhang, X. Li, L. Chen, Y. Li, Z. Zheng, X. Liu, and W. Lin, "Whale: Efficient giant model training over heterogeneous GPUs," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 673–688. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/jia-xianyan

[11] V. Prabhakaran, M. Wu, X. Weng, F. McSherry, L. Zhou, and M. Haradasan, "Managing large graphs on Multi-Cores with graph awareness," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX Association, Jun. 2012, pp. 41–52. [Online]. Available: https://www.usenix.org/conference/atc12/technical-sessions/presentation/prabhakaran

[12] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19, 2019, p. 270–288.

[13] Z. Han, H. Tan, S. H.-C. Jiang, X. Fu, W. Cao, and F. C. Lau, "Scheduling placement-sensitive bsp jobs with inaccurate execution time estimation," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, p. 1053–1062.

[14] Y. Liu, H. Xu, and W. C. Lau, "Online job scheduling with resource packing on a cluster of heterogeneous servers," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1441–1449.

[15] K. Psychas and J. Ghaderi, "Scheduling jobs with random resource requirements in computing clusters," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2269–2277.

[16] R. Burra, C. Singh, and J. Kuri, "Service scheduling for bernoulli requests and quadratic cost," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2584–2592.

[17] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 864–872.

[18] M. Yu, C. Wu, B. Ji, and J. Liu, "A sum-of-ratios multi-dimensional-knapsack decomposition for dnn resource scheduling," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[19] T. Anderson, *The theory and practice of online learning*. Athabasca University Press, 2008.

[20] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, Mar. 2011.

[21] F. Khorasani, H. Asghari Esfeden, A. Farmahini-Farahani, N. Jayasena, and V. Sarkar, "Regmutex: Inter-warp gpu register time-sharing," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 816–828.

[22] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: Compute allocation in hybrid clusters," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20, 2020.

[23] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, Renton, WA, Apr. 2022, pp. 945–960.

[24] J. F. Kurose and R. Simha, "A microeconomic approach to optimal resource allocation in distributed computer systems," *IEEE Transactions on computers*, vol. 38, no. 5, pp. 705–717, 1989.

[25] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Meng, and W. Lin, "Dl2: A deep learning-driven scheduler for deep learning clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 1947–1960, 2021.

[26] G. Bao and P. Guo, "Federated learning in cloud-edge collaborative architecture: key technologies, applications and challenges," *Journal of Cloud Computing*, vol. 11, no. 1, p. 94, 2022.

[27] S. Shalev-Shwartz, "Online learning and online convex optimization," *Foundations and Trends® in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2012.

[28] R. Bhattacharjee, S. Banerjee, and A. Sinha, "Fundamental limits on the regret of online network-caching," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, Jun. 2020.

[29] Y. Ying and M. Pontil, "Online gradient descent learning algorithms," *Foundations of Computational Mathematics*, vol. 8, no. 5, pp. 561–596, 2008.

[30] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 235–243.

[31] G. S. Paschos, A. Destounis, and G. Iosifidis, "Online convex optimization for caching networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 625–638, 2020.

[32] G. I. Ricardo, A. Tuholukova, G. Neglia, and T. Spyropoulos, "Caching policies for delay minimization in small cell networks with coordinated multi-point joint transmissions," *IEEE/ACM Transactions on Networking*, 2021.

[33] F. Orabona, "A modern introduction to online learning," *arXiv preprint arXiv:1912.13213*, 2019.

[34] "Alibaba cluster trace," https://github.com/alibaba/clusterdata, 2022.

[35] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: Association for Computing Machinery, 2013.

[36] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for Fine-Grained resource sharing in the data center," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. Boston, MA: USENIX Association, Mar. 2011.

[37] volcano sh, "Volcano," https://github.com/volcano-sh/volcano, 2022.

[38] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, Jul. 2021, pp. 1–18.

[39] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 495–503.

[40] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 2020, pp. 481–498.

[41] D. Cheng, X. Zhou, Y. Xu, L. Liu, and C. Jiang, "Deadline-aware mapreduce job scheduling with dynamic resource availability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 814–826, 2019.

[42] Z. Hu, B. Li, C. Chen, and X. Ke, "Flowtime: Dynamic scheduling of deadline-aware workflows and ad-hoc jobs," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 929–938.

[43] D. Hu and B. Krishnamachari, "Throughput optimized scheduler for dispersed computing systems," in *2019 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2019, pp. 76–84.

[44] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Communication-aware scheduling of serial tasks for dispersed computing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1330–1343, 2019.

[45] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1270–1286, 2020.

[46] Z. Han, H. Tan, X.-Y. Li, S. H.-C. Jiang, Y. Li, and F. C. M. Lau, "Ondisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2472–2485, 2019.

[47] S. Liang, Z. Yang, F. Jin, and Y. Chen, "Data centers job scheduling with deep reinforcement learning," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2020, pp. 906–917.

[48] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. Leung, "Tailored learning-based scheduling for kubernetes-oriented edge-cloud

system," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

Hailiang Zhao received the B.S. degree in 2019 from the school of computer science and technology, Wuhan University of Technology, Wuhan, China. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include cloud & edge computing, distributed systems and optimization algorithms. He has published several papers in flagship conferences and journals including IEEE ICWS 2019, IEEE TPDS, IEEE TMC, etc. He has been a recipient of the Best Student Paper Award of IEEE ICWS 2019. He is a reviewer for IEEE TSC and Internet of Things Journal.

Shuiguang Deng is currently a full professor at the College of Computer Science and Technology in Zhejiang University, China, where he received a BS and PhD degree both in Computer Science in 2002 and 2007, respectively. He previously worked at the Massachusetts Institute of Technology in 2014 and Stanford University in 2015 as a visiting scholar. His research interests include Edge Computing, Service Computing, Cloud Computing, and Business Process Management. He serves for the journal IEEE Trans. on Services Computing, Knowledge and Information Systems, Computing, and IET Cyber-Physical Systems: Theory & Applications as an Associate Editor. Up to now, he has published more than 100 papers in journals and refereed conferences. In 2018, he was granted the Rising Star Award by IEEE TCSVC. He is a fellow of IET and a senior member of IEEE.

Zhengzhe Xiang received the B.S. and Ph.D. degree of Computer Science and Technology in Zhejiang University, Hangzhou, China. He was previously a visiting student worked at the Karlstad University, Sweden in 2018. He is currently a Lecturer with Zhejiang University City College, Hangzhou, China. His research interests lie in the fields of Service Computing, Cloud Computing, and Edge Computing.

Xueqiang Yan is currently a technology expert with the Wireless Technology Lab, Huawei Technologies. He was a member of technical staff at Bell Labs from 2000 to 2004. From 2004 to 2016, he was the director of the Strategy Department, Alcatel-Lucent Shanghai Bell. His current research interests include wireless networking, the Internet of Things, edge AI, future mobile network architecture, network convergence, and evolution.

Jianwei Yin received the Ph.D. degree in computer science from Zhejiang University (ZJU) in 2001. He was a Visiting Scholar with the Georgia Institute of Technology. He is currently a Full Professor with the College of Computer Science, ZJU. Up to now, he has published more than 100 papers in top international journals and conferences. His current research interests include service computing and business process management. He is an Associate Editor of the IEEE Transactions on Services Computing.

Schahram Dustdar is a Full Professor of Computer Science (Informatics) with a focus on Internet Technologies heading the Distributed Systems Group at the TU Wien. He is founding co-Editor-in-Chief of ACM Transactions on Internet of Things (ACM TIoT) as well as Editor-in-Chief of Computing (Springer). He is an Associate Editor of IEEE Transactions on Services Computing, IEEE Transactions on Cloud Computing, ACM Computing Surveys, ACM Transactions on the Web, and ACM Transactions on Internet Technology, as well as on the editorial board of IEEE Internet Computing and IEEE Computer. Dustdar is recipient of multiple awards: TCI Distinguished Service Award (2021), IEEE TCSVC Outstanding Leadership Award (2018), IEEE TCSC Award for Excellence in Scalable Computing (2019), ACM Distinguished Scientist (2009), ACM Distinguished Speaker (2021), IBM Faculty Award (2012). He is an elected member of the Academia Europaea: The Academy of Europe, where he is chairman of the Informatics Section, as well as an IEEE Fellow (2016), an Asia-Pacific Artificial Intelligence Association (AAIA) President (2021) and Fellow (2021). He is an EAI Fellow (2021) and an I2CICC Fellow (2021). He is a Member of the 2022 IEEE Computer Society Fellow Evaluating Committee (2022).

Albert Y. Zomaya is the Peter Nicol Russell Chair Professor of Computer Science and Director of the Centre for Distributed and High-Performance Computing at the University of Sydney. To date, he has published > 600 scientific papers and articles and is (co-)author/editor of > 30 books. A sought-after speaker, he has delivered > 250 keynote addresses, invited seminars, and media briefings. His research interests span several areas in parallel and distributed computing and complex systems. He is currently the Editor in Chief of the ACM Computing Surveys and processed in the past as Editor in Chief of the IEEE Transactions on Computers (2010-2014) and the IEEE Transactions on Sustainable Computing (2016-2020).

Professor Zomaya is a decorated scholar with numerous accolades including Fellowship of the IEEE, the American Association for the Advancement of Science, and the Institution of Engineering and Technology (UK). Also, he is an Elected Fellow of the Royal Society of New South Wales and an Elected Foreign Member of Academia Europaea. He is the recipient of the 1997 Edgeworth David Medal from the Royal Society of New South Wales for outstanding contributions to Australian Science, the IEEE Technical Committee on Parallel Processing Outstanding Service Award (2011), IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing (2011), IEEE Computer Society Technical Achievement Award (2014), ACM MSWIM Reginald A. Fessenden Award (2017), the New South Wales Premier's Prize of Excellence in Engineering and Information and Communications Technology (2019), and the Research Innovation Award, IEEE Technical Committee on Cloud Computing (2021).