Learning to Schedule Multi-Server Jobs With Fluctuated Processing Speeds

Hailiang Zhao, Shuiguang Deng, *Senior Member, IEEE*, Feiyi Chen, Jianwei Yin, Schahram Dustdar, *Fellow, IEEE*, and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Multi-server jobs are imperative in modern cloud computing systems. A noteworthy feature of multi-server jobs is that, they usually request multiple computing devices simultaneously for their execution. How to schedule multi-server jobs online with a high system efficiency is a topic of great concern. First, the scheduling decisions have to satisfy the service locality constraints. Second, the scheduling decisions needs to be made online without the knowledge of future job arrivals. Third, and most importantly, the actual service rate experienced by a job is usually in fluctuation because of the dynamic voltage and frequency scaling (DVFS) and power oversubscription techniques when multiple types of jobs co-locate. A majority of online algorithms with theoretical performance guarantees are proposed. However, most of them require the processing speeds to be knowable, thereby the job completion times can be exactly calculated. To present a theoretically guaranteed online scheduling algorithm for multi-server jobs without knowing actual processing speeds apriori, in this article, we propose ESDP (Efficient Sampling-based Dynamic Programming), which learns the distribution of the fluctuated processing speeds over time and simultaneously seeks to maximize the cumulative overall utility. The cumulative overall utility is formulated as the sum of the utilities of successfully serving each multi-server job minus the penalty on the operating, maintaining, and energy cost. ESDP is proved to have a polynomial complexity and a logarithmic regret, which is a State-of-the-Art result. We also validate it with extensive simulations and the results show that the proposed algorithm outperforms several benchmark policies with improvements by up to 73%, 36%, and 28%, respectively.

Index Terms—Bipartite graph, dynamic programming, multi-server job, online learning, regret analysis

1 Introduction

Today's computing clusters have plenty of multi-server jobs, e.g., the distributed training of deep neural networks [1], [2] and large-scale graph computations [3], [4]. A notable feature of multi-server jobs is that they usually request multiple computing devices *simultaneously* such as CPUs and GPUs and hold onto them during their execution. From Google cluster trace [5], we can observe that more than 90% jobs request multiple CPU cores and nearly 20% jobs request CPU cores no less than 1000.

It is difficult for the cluster scheduler to allocate an appropriate number of computing devices to each multi-server job with a high system efficiency. The major challenges are discussed as follows.

- *Service Locality*. Service locality is common in modern cloud and edge computing systems, especially for *Machine Learning as a Service* (MLaaS) [6] and Serverless computing [7], [8]. With service locality, a multi-server job may only be processed by a subset of servers where the computing device request, software dependencies, and other requirements such as geographical constraints are satisfied. For instance, in a resource-constrained cluster, service locality could lead to a situation where all the DNN training jobs are scheduled to the *only*server with GPUs and the rest of them have to wait until the GPUs are released.
- *Unknown Arrival Patterns of Jobs.* In real-world scenarios, multi-server jobs arrive to the cluster online. The scheduler needs to make the resource allocation decisions without knowing the job arrival patterns apriori. The lack of the job arrival distributions could lead to the scheduling decision to *a local optimum*.
- *The Processing Speeds Experienced by each Job is Fluctuated*. In production systems where different multi-server jobs co-locate, such as computation-intensive jobs, IO-intensive jobs, and latency-critical jobs etc., the processing speeds may fluctuate over time and could be highly variable occasionally. The reason is that the server is always in multi-tasking of different jobs, and the hardware techniques such as Dynamic Voltage and Frequency Scaling (DVFS) [9] and power oversubscription [10] adjust the CPU cycle frequency constantly.

A majority of online scheduling algorithms with theoretical guarantees have been proposed by formulating

- *Hailiang Zhao, Shuiguang Deng, Feiyi Chen, and Jianwei Yin are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China. E-mail: {hliangzhao, dengsg, chenfeiyi, zjuyjw}@zju.edu.cn.*
- *Schahram Dustdar is with the Distributed Systems Group, Technische Universität Wien, 1040 Vienna, Austria. E-mail: dustdar@dsg.tuwien.ac.at.*
- *Albert Y. Zomaya is with the School of Computer Science, University of Sydney, Sydney, NSW 2006, Australia. E-mail: albert.zomaya@sydney.edu.au.*

Manuscript received 8 April 2022; revised 16 October 2022; accepted 17 October 2022. Date of publication 20 October 2022; date of current version 16 November 2022.
This work was partially supported in part by the National Science Foundation of China (NSFC) under Grants U20A20173 and 62125206, and in part by the Key Research Project of Zhejiang Province under Grant 2022C01145. Schahram Dustdar's work was supported by the Zhejiang University Deqing Institute of Advanced technology and Industrilization (ZDATI).
(Corresponding author: Shuiguang Deng.)
Recommended for acceptance by S. Wang.
Digital Object Identifier no. 10.1109/TPDS.2022.3215947

combinatorial optimization problems with scenario-oriented constraints [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. To solve these combinatorial programs, algorithms are designed with various theoretical approaches. Typical approaches include relaxed integer programming [12], online primal-dual alternating updates [13], online approximate algorithms [21], [22], [23], heuristics [14], [15], deep reinforcement learning (DRL) [16], [17], etc. However, despite the vast literature of them, their model formulations which tackle with the fluctuated processing speeds of multi-server jobs are limited. To execute these online algorithms, the processing speeds of servers are required to be *knowable* when making the scheduling decisions, thereby the job completion times can be exactly calculated. However, as we have analyzed above, in production systems where different types of multi-server jobs co-locate, the actual processing speeds experienced by jobs is unknown and fluctuated when making the scheduling decisions.

To present a theoretically guaranteed online scheduling algorithm for multi-server jobs *without knowing the distributions of the processing speeds apriori*, in this paper, we propose Esdp (Efficient Sampling-based Dynamic Programming) to learn the distributions of the fluctuated processing speeds with sufficient exploration-exploitation and simultaneously to maximize the cumulative overall utility (Aou). Aou is formulated as the sum of the obtained utilities of successfully processing each multi-server job minus the penalty on the operating, maintaining, and energy cost for serving them over each time slot. Further, the utility of a job is fitted by a *stochastic quasi-linear* function of allocated computing devices in terms of its completion time. Our work is built on the intuition that, for a multi-server job, its completion time is mainly determined by the actual processing speed it experiences, which is linear with the allocated computing devices. Our basic assumption is that, although the actual processing speeds are fluctuated over time, *they come from some certain distributions, which are determined by the hardware specifications of the underlying physical machines*. It is exactly Esdp's job to *learn* the underlying processing speed distributions and leverage them to guide the computing device allocations. Specifically, Esdp casts the online multi-server job scheduling problem into the framework of online learning [24], and it makes the scheduling decisions for each arrived job with sufficient *exploration-exploitation*. Based on the exploited patterns, Esdp introduces several deterministic maximization problems whose targets are the expectation of Aou approximated by statistics. Then, Esdp solves these deterministic problems with a dynamic programming subroutine in polynomial time. We use regret [24], i.e., the gap on Aou between Esdp's and *the offline optimum* achieved by the oracle, to analyze the performance of Esdp. We provide a rigorous proof to show that Esdp has a best-so-far regret, i.e., $\mathcal{O}(\ln T)$, where T is the time slot length. Our contribution fulfills one of the key deficiencies of current literature in the stochastic scheduling of mutli-server jobs without knowing processing speeds apriori. The main contributions are summarized as follows.

- We propose an online algorithm, i.e., Esdp, to schedule multi-server jobs without exact processing speeds apriori. Esdp makes no assumptions on the job arrival patterns, and it fully takes service locality into consideration. We prove that Esdp has a best-so-far regret $\mathcal{O}(\ln T)$, which grows logarithmically with the time slot length.
- Esdp casts the online stochastic scheduling problem into the framework of online learning, and adopts several dynamic programming subroutines to solve the approximated deterministic problems in polynomial time.
- We validate the performance of Esdp with extensive simulations. Experimental results show that, in default settings, Esdp significantly outperforms several widely used heuristics with improvements by up to 73%, 36%, and 28%, respectively.

The rest of this paper is organized as follows. We formulate the stochastic multi-server job scheduling problem with a bipartite graph in Section 2. We then present the design details of Esdp with theoretical analysis in Section 3. Numerical results are presented in Section 4. We discuss related works in Section 5 and close this paper in Section 6.

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 multi-server job type (port)
$r \in \mathcal{R}$	A server/node
$(l, r) \in \mathcal{E}$	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 servers connect to l
$\rho_l(t)$	The job arrival probability of port l at time t
$\mathbb{1}_l(t) \in \{0, 1\}$	The job arrival status of port l at time t
$\mathcal{K}$	The set of different types of computing devices
$\boldsymbol{a}_k$	The ovearll request on device k
c_k	The number of the type-k devices in the cluster
$\boldsymbol{x}(t)$	The scheduling decision at time t
$\forall k : c_k$	The total number of type-k devices
$U_l(t)$	The utility if job l at time t
$\forall k : f_k(\cdot)$	Cost of provisioning type-k devices
$\mathrm{U}(\boldsymbol{x}(t))$	The overall utility at time t
$\mathrm{Re}(T)$	The regret over the time horizon $\mathcal{T}$

2 System Model

We consider a computing cluster of heterogeneous servers serving several types of multi-server jobs. Different servers are equipped with different types and quantities of computing devices, including CPUs, GPUs, etc. Multi-server jobs of different types have different requests on computing devices under the service locality constraints. Key notations used in this paper are summarized in Table 1.

2.1 Bipartite Graph Model Under Service Localities

We use a bipartite graph $(\mathcal{L}, \mathcal{R}, \mathcal{E})$ to model service locality, where $\mathcal{L}$ and $\mathcal{R}$ are the set of left vertices and right vertices, respectively, and $\mathcal{E}$ is the set of edges between the two sets of vertices. The vertices in $\mathcal{L}$ are indexed by l and viewed as job types, while the vertices in $\mathcal{R}$ are indexed by r and represent servers. For a vertex $l \in \mathcal{L}$, we use $\mathcal{R}_l \subseteq \mathcal{R}$ to represent the set

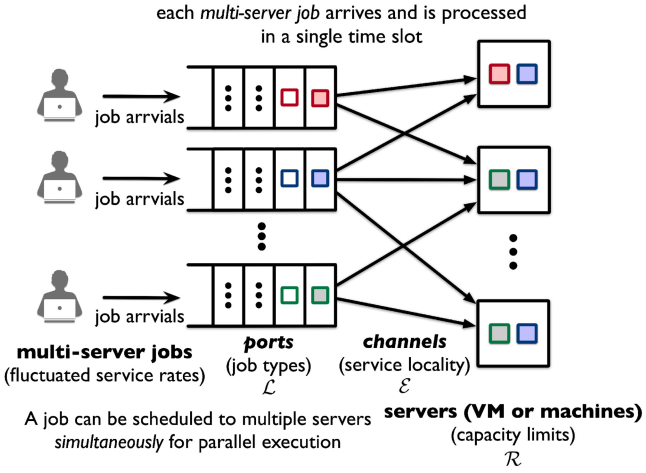

Fig. 1. The bipartite graph model for multi-server job scheduling.

of right vertices it connects with. Similarly, we use $\mathcal{L}_r \subseteq \mathcal{L}$ to represent the set of left vertices for $r \in \mathcal{R}$.

We designate each vertex $l \in \mathcal{L}$ as *port* and each edge (l, r) as *channel*. The bipartite graph model is visualized in Fig. 1.

2.2 Job Scheduling With Restricted Capacities

In our formulation, time is slotted, and at each time $t \in \mathcal{T} := \{1, \ldots, T\}$, for each port, at most one job arrives[1]. Concretely, at the beginning of time t, a job is yielded from port l with probability $\rho_l(t)$, and with probability $1 - \rho_l(t)$, there is no job. It is worth noting that, the probabilities $\{\rho_l(t)\}_{l \in \mathcal{L}}$ are only used for generating job arrival instances, which is not required by the to-be-proposed algorithm Esdp when making the online decisions.

There are K types of computing devices in the cluster, including CPUs, GPUs, NPUs, and FPGAs. For each type-l job, we denote by $a_k^{(l,r)} \in \mathbb{N}^+$ its request on the type-k computing device when it is processed by server r through the channel (l, r). The total number of the type-k computing devices in the cluster, where $k \in \mathcal{K} := \{1, .., K\}$, is represented by $c_k \in \mathbb{N}^+$.

At time t, we use

$$\boldsymbol{x}(t) := \left[x_{(l,r)}(t)\right]^{\mathrm{T}}_{\forall (l,r) \in \mathcal{E}} \in \mathcal{X} := \{0, 1\}^{|\mathcal{E}|} \tag{1}$$

to represent the scheduling decision. A job can be scheduled to multiple servers simultaneously for parallel execution. A constraint $\boldsymbol{x}(t)$ should satisfy is that, the computing devices allocated out from the cluster should not more than it has:

$$\sum_{(l,r) \in \mathcal{E}} a_k^{(l,r)} x_{(l,r)}(t) \le c_k, \forall k \in \mathcal{K}, t \in \mathcal{T}. \tag{2}$$

Note that if port l yields no job at t, denoted by $\mathbb{1}_l(t) = 0$, then $x_{(l,r)}(t) = 0$ for all $r \in \mathcal{R}_l$.

The multi-server job scheduling problem is studied for maximizing the Aou, which is formulated as the sum of the utilities of successfully serving each multi-server job minus the penalty on the operating, maintaining, and energy cost for serving them over each time slot. We denote by $U_l(t)$ the utility of the type-l job at time t, and it is formulated as

$$U_l(t) := \sum_{r \in \mathcal{R}_l} x_{(l,r)}(t) Z_{(l,r)}(t) - \sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}_l} f_k\left(a_k^{(l,r)}\right) x_{(l,r)}(t), \tag{3}$$

where $Z_{(l,r)}(t)$ is a stochastic variable following an underlying distribution with the expectation of $\upsilon_{(l,r)}$. We formulate $Z_{(l,r)}(t)$ as the actual computation utility experienced by the type-l job at time t when it is processed by server r through the channel $(l, r) \in \mathcal{E}$. Correspondingly, $\upsilon_{(l,r)}$ is the expectation of the computation utility, and it is unknown when making the scheduling decisions. Our formulation is built on the assumption that, although $\upsilon_{(l,r)}$ cannot be obtained apriori, we can learn it and approximate it with sufficient *exploration-exploitation*. In addition, the computation utility is *linearly additive*, i.e., if a job is processed through multiple channels in parallel, the final utility is the sum of computation utility obtained from all these channels. The second part in (3) is the penalty on the supply cost. Thereinto, $f_k(a_k^{(l,r)})$ is the supply cost for provisioning $a_k^{(l,r)}$ units of the type-k computing device for the type-l job through the channel (l, r). $\{f_k(\cdot)\}_{\forall k \in \mathcal{K}}$ models the operating, maintaining, and energy cost for serving jobs. Different from previous works [25], [26], [27], we make no assumptions on the convexity or differentiability of $\{f_k(\cdot)\}_{\forall k \in \mathcal{K}}$.

Our goal is to maximize the expectation of Aou, i.e., the expected sum of utilities of multi-server jobs in a long-term horizon. The problem is formulated as follows.

$$\begin{aligned} \mathcal{P}_1 : \quad & \max_{\forall t \in \mathcal{T}: \boldsymbol{x}(t) \in \mathcal{X}} \lim_{T \to \infty} \sum_{t=1}^{T} \mathbb{E}\left[\sum_{l \in \mathcal{L}} U_l(t)\right] \\ s.t. \quad & (2), \\ & \sum_{r \in \mathcal{R}_l} x_{(l,r)}(t) = 0 \text{ if } \mathbb{1}_l(t) = 0, \forall l \in \mathcal{L}, t \in \mathcal{T}, \end{aligned} \tag{4}$$

With further transformation, we can get

$$\mathbb{E}\left[\sum_{l \in \mathcal{L}} U_l(t)\right] = \sum_{(l,r) \in \mathcal{E}} x_{(l,r)}(t) \left[Z_{(l,r)}(t) - \sum_{k \in \mathcal{K}} f_k\left(a_k^{(l,r)}\right)\right]. \tag{5}$$

In the following content, we use $\mathrm{U}(\boldsymbol{x}(t))$ and $\sum_{l \in \mathcal{L}} U_l(t)$ interchangeably.

3 Algorithm Design

In this section, we demonstrate the design details of Esdp, which can solve $\mathcal{P}_1$ with a *probabilistic* optimality asymptotically in polynomial time. Esdp is built on the well known ESCB policy [28], [29] and a recent derivative, called AESCB [30], for solving combinatorial semi-bandit problems. In the following content, first, we formulate the regret minimization problem that corresponds to $\mathcal{P}_1$ and bring in several evolving statistics to approximate $\mathbb{E}[\mathrm{U}(\boldsymbol{x}(t))]$ at each time t. Based on these statistics and a converge-to-zero sequence $\{\delta(t)\}_{t \in \mathcal{T}}$, we introduce a series of deterministic optimization problems. Then, we solve these deterministic problems sequentially based on dynamic programming in polynomial times. After that, we provide rigorous theoretical analysis

1. Even though, our model can be easily extended to the scenarios where multiple jobs arrive from a port in a single time slot.

for ESDP in terms of the algorithmic complexity and the regret on AOU. In the end, we discuss the possible extensions of ESDP on the Gang scheduling scenarios.

3.1 Regret Minimizing With Evolving Statistics

$\mathcal{P}_1$ is an online *stochastic* optimization problem with random variables $Z(t) = [Z_{(l,r)}(t)]^{\mathrm{T}}_{\forall (l,r)\in\mathcal{E}}$ not determined until the time t arrives. $\mathcal{P}_1$ is equivalent to the regret minimization problem listed below:

$$\mathcal{P}_2: \min_{\forall t\in T: \boldsymbol{x}(t)\in\mathcal{X}} \lim_{T\to\infty} \mathrm{RE}(T) := \sum_{t=1}^{T} \mathbb{E}[\Delta(\boldsymbol{x}(t))]$$
$$s.t. \quad (2), (4),$$

where the expected per-time slot gap $\mathbb{E}[\Delta(\boldsymbol{x}(t))]$ is

$$\mathbb{E}[\Delta(\boldsymbol{x}(t))] := \tilde{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{x}^*(t) - \mathbb{E}\left[\sum_{l\in\mathcal{L}} U_l(t)\right] \tag{6}$$

and

$$\begin{cases} \tilde{\boldsymbol{v}} := \left[v_{(l,r)} - \sum_{k\in\mathcal{K}} f_k\left(a_k^{(l,r)}\right)\right]^{\mathrm{T}}_{\forall (l,r)\in\mathcal{E}} \in [0,1]^{|\mathcal{E}|} \\ \boldsymbol{x}^*(t) := \mathrm{argmax}_{\boldsymbol{x}(t)\in\Omega(t)} \{\tilde{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{x}(t)\} \\ \Omega(t) := \{\boldsymbol{x}(t)\in\mathcal{X} \,|\, (2) \text{ \& } (4) \text{ hold at time } t\}. \end{cases} \tag{7}$$

The regret $\mathrm{RE}(T)$ is the gap between the optimal AOU achieved by an *omniscient* oracle who has the full knowledge on $\boldsymbol{v}$ and the AOU achieved by the to-be-proposed algorithm ESDP. A good algorithm should achieve a smallest possible regret $\mathrm{RE}(T)$ as T goes to infinity. For simplification, we denote by $\tilde{\boldsymbol{Z}}(t)$ the column vector

$$\left[Z_{(l,r)}(t) - \sum_{k\in\mathcal{K}} f_k(a_k^{(l,r)})\right]^{\mathrm{T}}_{\forall (l,r)\in\mathcal{E}}.$$

W.O.L.G, we normalize $\tilde{\boldsymbol{Z}}(t)$ into $[0,1]^{|\mathcal{E}|}$ by carefully tuning the parameters in $\{f_k(\cdot)\}_{k\in\mathcal{K}}$. The non-negative property is widely accepted for utility functions [25], [27], [31], [32]. Nevertheless, different from the above literature, we make no assumptions on the convexity or differentiability of $\{f_k\}_{\forall k\in\mathcal{K}}$.

$\mathcal{P}_2$ is still a stochastic optimization problem and the expectation operation is not eliminated. To make it solvable, based on the idea introduced by the ESCB policy [28], we introduce several statistics to approximate $\boldsymbol{v}$ with the explorated information. These statistics are used to supersede the random variables in $\mathcal{P}_2$. Specifically, at each time t, we define

$$n_{(l,r)}(t) := \sum_{t'=1}^{t} x_{(l,r)}(t') \tag{8}$$

as the *cumulative quantity* of channel $(l,r)\in\mathcal{E}$ been used up to time t. Based on it, we define the following statistics:

$$\hat{v}_{(l,r)}(t) := \begin{cases} \frac{\sum_{t'=1}^{t} x_{(l,r)}(t')\tilde{Z}_{(l,r)}(t')}{n_{(l,r)}(t)} & n_{(l,r)}(t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$\hat{\sigma}^2_{(l,r)}(t) := \begin{cases} \frac{g(t)}{2n_{(l,r)}(t)} & n_{(l,r)}(t) > 0 \\ +\infty & \text{otherwise,} \end{cases} \tag{10}$$

where

$$g(t) := \ln t + 4\ln(\ln t + 1)\cdot \max_{t'\in\mathcal{T}} \left\{ \max_{\boldsymbol{x}\in\Omega(t')} \|\boldsymbol{x}\|_1 \right\}. \tag{11}$$

$\hat{v}_{(l,r)}(t)$ is a non-biased estimation based on historical noisy computation utilities for type-l job when processed through channel (l,r). $\hat{\sigma}^2_{(l,r)}(t)$ is a metric proportional to the variance of the estimate $\hat{v}_{(l,r)}(t)$, proposed by [28]. We place a hat on the estimations to indicate that they are calculated and updated online.

Inspired by the ESCB and AESCB policies, at time t, we introduce the following *deterministic* problem $\mathcal{P}_3(t)$:

$$\mathcal{P}_3(t): \max_{\boldsymbol{x}(t)\in\Omega(t)} \tilde{\mathrm{U}}(\boldsymbol{x}(t)) := \delta(t) + \hat{\boldsymbol{v}}(t)^{\mathrm{T}}\boldsymbol{x}(t) + \sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}$$
$$s.t. \quad (2),$$
$$\delta(t) > 0, \lim_{t\to\infty}\delta(t) = 0, \tag{12}$$

where

$$\begin{cases} \hat{\boldsymbol{v}}(t) := \left[\hat{v}_{(l,r)}(t)\right]^{\mathrm{T}}_{(l,r)\in\mathcal{E}} \\ \hat{\boldsymbol{\sigma}}^2(t) := \left[\hat{\sigma}^2_{(l,r)}(t)\right]^{\mathrm{T}}_{(l,r)\in\mathcal{E}} \end{cases}$$

are the corresponding column vectors. Moreover, $\hat{\boldsymbol{v}}(t)$ can be efficiently calculated through matrix operations as follows:

$$\boldsymbol{\phi}\left([\boldsymbol{x}(1),\ldots,\boldsymbol{x}(t)]\left[\left(\tilde{\mathbf{Z}}(1) \oslash n(1)\right)^{\mathrm{T}},\ldots,\left(\tilde{\mathbf{Z}}(t)\oslash \boldsymbol{n}(t)\right)^{\mathrm{T}}\right]^{\mathrm{T}}\right),$$

where $\oslash$ is the element-wise division operator, $\boldsymbol{n}(t)$ is the vector $\{n_{(l,r)}(t)\}^{\mathrm{T}}_{(l,r)\in\mathcal{E}}$, and $\boldsymbol{\phi}(\cdot)$ is the inverse of function $\mathrm{diag}(\cdot)$, defined as

$$\boldsymbol{\phi}(\mathbf{M}) := \sum_{i=1}^{|\mathcal{E}|} (\boldsymbol{e}_i^{\mathrm{T}}\mathbf{M}\boldsymbol{e}_i)\boldsymbol{e}_i, \quad \mathbf{M}\in\mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}. \tag{13}$$

In (13), $\boldsymbol{e}_i$ is the i-th standard unit basis.

In $\mathcal{P}_3(t)$, $\{\delta(t)\}_{t\in\mathcal{T}}$ could be any sequence converges to zero. For instance,

$$\delta(t) := \frac{1}{\ln(\ln t + 1) + 1}. \tag{14}$$

The objective of $\mathcal{P}_3(t)$ is an approximated *statistical-based* overall computation utility at time t. From $\mathcal{P}_2$ to $\mathcal{P}_3(t)$, we remove the random variable $\boldsymbol{Z}(t)$ and thereby remove the expectation operation in the objective. As a result, we transform the original stochastic problem into a deterministic problem while keeping the solution space impervious. In most case, the following inequality should hold:

$$\left|(\tilde{\boldsymbol{v}} - \hat{\boldsymbol{v}}(t))^{\mathrm{T}}\boldsymbol{x}(t)\right| \le \sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}. \tag{15}$$

By Chebyshev's Inequality, $\hat{\boldsymbol{v}}(t)^{\mathrm{T}}\boldsymbol{x}(t) \pm \sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}$ covers nearly 60% population. To achieve a larger coverage, we can increase the numerical multiplier to the standard variance. In our formulation, setting the multiplier as 1 is enough to achieve the State-of-the-Art minimum regret upper bound. The analysis will be detailed in Section 3.3.

3.2 Polynomial-Time Dynamic Programming

If the sequence $\{\delta(t)\}_{t\in\mathcal{T}}$ is removed from $\tilde{\mathrm{U}}(\boldsymbol{x}(t))$ and (12) is dropped, $\mathcal{P}_3(t)$ is NP-hard [28], [29], i.e., it cannot be solved in polynomial time. Therefore, to solve it efficiently, inspired by the AESCB policy [30], ESDP resorts to solving several *relaxed* budgeted integer programming problems by adding the converge-to-zero sequence $\{\delta(t)\}_{t\in\mathcal{T}}$, which is exactly what we have done when formulating $\mathcal{P}_3(t)$.

In the following, we will detail how we solve $\mathcal{P}_3(t)$ with dynamic programming. First, at each time t, based on $\delta(t)$, we define the following scale-up statistics for $\hat{\upsilon}_{(l,r)}(t)$ and $\hat{\sigma}^2_{(l,r)}(t)$ respectively:

$$\hat{\Upsilon}_{(l,r)}(t) := \left\lceil \xi(t)\hat{\upsilon}_{(l,r)}(t) \right\rceil \tag{16}$$

$$\hat{\boldsymbol{\Sigma}}^2_{(l,r)}(t) := \left\lceil \xi^2(t)\hat{\sigma}^2_{(l,r)}(t) \right\rceil, \tag{17}$$

where

$$\xi(t) := \left\lceil \frac{\max_{t'\in\mathcal{T}}\{\max_{\boldsymbol{x}\in\Omega(t')}\|\boldsymbol{x}\|_1\}}{\delta(t)} \right\rceil \tag{18}$$

is the scaling size at time t. By the AESCB policy [30], at each time t, we introduce several budgeted integer programming problems $\mathcal{P}_4(s,t)$ for each $s\in\mathcal{S}(t)$, where

$$\mathcal{S}(t) := \left\{0,1,\ldots,\xi(t)\cdot\max_{t'\in\mathcal{T}}\max_{\boldsymbol{x}\in\Omega(t')}\|\boldsymbol{x}\|_1\right\}, \tag{19}$$

as follows:

$$\begin{aligned}\mathcal{P}_4(s,t):\quad & \max_{\boldsymbol{x}(t)\in\mathcal{X}} \hat{\boldsymbol{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)\\ s.t.\quad & (2),(12),\\ & \hat{\boldsymbol{\Upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)\geq s.\end{aligned} \tag{20}$$

In $\mathcal{P}_4(s,t)$, $\hat{\boldsymbol{\Sigma}}^2(t)$ and $\hat{\boldsymbol{\Upsilon}}(t)$ are the corresponding column vectors for (16) and (17), respectively. Let us use $\boldsymbol{x}^*_{\mathcal{P}_4}(s,t)$ to denote the optimal solution for $\mathcal{P}_4(s,t)$. Then, the final solution to $\max\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$ at time t, denoted by $\boldsymbol{x}^*_{\mathcal{P}_4}(t)$, is set as some $\boldsymbol{x}^*_{\mathcal{P}_4}(s^\star,t)$ where $s^\star\in\mathcal{S}(t)$ staisfies

$$s^\star \in \operatorname*{argmax}_{s\in\mathcal{S}(t)}\left\{s+\sqrt{\hat{\boldsymbol{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(s,t)}\right\}. \tag{21}$$

Now we demonstrate the detailed procedure of ESDP, which is summarized in *Algorithm 1*. ESDP solves $\mathcal{P}_1$ and $\mathcal{P}_2$ by solving the problems $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t),t\in\mathcal{T}}$. The relations between $\mathcal{P}_3(t)$ and $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$, and how the solutions of $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t),t\in\mathcal{T}}$ affect the regret $\mathrm{RE}(T)$ will be analyzed in Section 3.3.

Now, the problem is how to solve $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$ optimally within polynomial time. ESDP solves it based on dynamic programming. Concretely, at each time t, corresponding to each $\mathcal{P}_4(s,t)$, we bring in the problem $\mathcal{P}_5(s,t,\boldsymbol{c},i)$ as follows.

$$\begin{aligned}\mathcal{P}_5(s,t,\boldsymbol{c},i):\quad & \max_{\boldsymbol{x}(t)\in\mathcal{X}} \hat{\boldsymbol{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)\\ s.t.\quad & (2),(12),(20),\\ & \sum_{e=e_1}^{e_i} x_e(t)=0,\end{aligned} \tag{22}$$

where $\boldsymbol{c} := [c_k]^{\mathrm{T}}_{k\in\mathcal{K}}$ is the capacity vector in (2), $e:=(l,r)\in\mathcal{E}$ and e_i is the i-th edge (l,r) in $\mathcal{E}$. The new constraint (22) is used to set the first several scheduling decisions (until i) to 0 forcibly. Obviously, $\mathcal{P}_5(s,t,\boldsymbol{c},0)$ is equal to $\mathcal{P}_4(s,t)$ because (22) is not functioning when $i=0$. The optimal solution of $\mathcal{P}_5(s,t,\boldsymbol{c},i)$ can be obtained by recursing over s, $\boldsymbol{c}$, and i. To do this, let us use $\boldsymbol{x}^*(s,t,\boldsymbol{c},i)$ to denote the optimal solution of $\mathcal{P}_5(s,t,\boldsymbol{c},i)$, and use $V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c},i)$ to denote the corresponding objective. In the following, we demonstrate the recursing details.

Algorithm 1. The ESDP Framework

Input: The bipartite graph $(\mathcal{L},\mathcal{R},\mathcal{E})$, requirements $\{a_k^{(l,r)}\}_{k\in\mathcal{K},(l,r)\in\mathcal{E}}$, capacities $\{c_k\}_{k\in\mathcal{K}}$, cost functions $\{f_k\}_{k\in\mathcal{K}}$, and the sequence $\{\delta(t)\}_{t\in\mathcal{T}}$
Output: Online solution to $\mathcal{P}_1$ (and $\mathcal{P}_2$) at time $t\in\mathcal{T}$

while $t=1,\ldots,T$ **do**
  Observe the job arrival status from each port $l\in\mathcal{L}$
  Update $\hat{\boldsymbol{\Upsilon}}(t)$ and $\hat{\boldsymbol{\Sigma}}^2(t)$ with (16) and (17) based on $\delta(t)$, respectively
  `/* Solve` $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$ `by Algorithm 2 */`
  for *each* $s\in\mathcal{S}(t)$ **do**
    Solve $\mathcal{P}_4(s,t)$ and return $\boldsymbol{x}^*_{\mathcal{P}_4}(s,t)$
  end for
  $\boldsymbol{x}^*_{\mathcal{P}_4}(t)\leftarrow\boldsymbol{x}^*_{\mathcal{P}_4}(s^\star,t)$, where $s^\star$ staisfies (21)
  `/* Satisfy constraint (4) of` $\mathcal{P}_1$ `*/`
  for *each* $l\in\mathcal{L}$ **do**
    if $\mathbb{1}_l(t)==0$ **then**
      for *each* $r\in\mathcal{R}_l$ **do**
        Set the (l,r)-th element of $\boldsymbol{x}^*_{\mathcal{P}_4}(t)$ as 0
      end for
    end if
  end for
end while
return $\{\boldsymbol{x}^*_{\mathcal{P}_4}(t)\}_{t\in\mathcal{T}}$ and $\{\mathrm{U}(\boldsymbol{x}^*_{\mathcal{P}_4}(t))\}_{t\in\mathcal{T}}$

<u>Case I</u>: If $x^*_{e_{i+1}}(s,t,\boldsymbol{c},i)=0$, i.e., the $(i+1)$-element of $\boldsymbol{x}^*(s,t,\boldsymbol{c},i)$ is 0, then (22) is not violated for $\mathcal{P}_5(s,t,\boldsymbol{c},i+1)$. Thus, we have

$$\boldsymbol{x}^*(s,t,\boldsymbol{c},i+1)=\boldsymbol{x}^*(s,t,\boldsymbol{c},i) \tag{23}$$

and

$$V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c},i+1)=V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c},i). \tag{24}$$

The result means that $\boldsymbol{x}^*(s,t,\boldsymbol{c},i)$ is also the optimal solution to $\mathcal{P}_5(s,t,\boldsymbol{c},i+1)$.

<u>Case II</u>: If $x^*_{e_{i+1}}(s,t,\boldsymbol{c},i)=1$, the optimal substructure is much more complicated. For simplification, we define matrix $\mathbf{A}$ by

$$\mathbf{A}=\left[a_k^{(l,r)}\right]^{K\times|\mathcal{E}|}.$$

Then we have

$$\mathbf{A}(\boldsymbol{x}^*(s,t,\boldsymbol{c},i)-\boldsymbol{e}_{i+1})\leq\boldsymbol{c}-A_{:,i+1}, \tag{25}$$

where $\boldsymbol{e}_{i+1}$ is the $(i+1)$-th standard unit basis. Besides,

$$\hat{\boldsymbol{\Upsilon}}(t)^{\mathrm{T}}(\boldsymbol{x}^*(s,t,\boldsymbol{c},i)-\boldsymbol{e}_{i+1})\geq s-\hat{\Upsilon}_{e_{i+1}}(t) \tag{26}$$

and

$$\hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}(\boldsymbol{x}^*(s,t,\boldsymbol{c},i)-\boldsymbol{e}_{i+1}) = \hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*(s,t,\boldsymbol{c},i) - \hat{\Sigma}^2_{e_{i+1}}(t).$$

Combining the above formula with (25) and (26), we can get the following evolving optimal substructure:

$$\begin{aligned} V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c},i) =& V^*_{\mathcal{P}_5}\Big(\max\{s-\hat{\Upsilon}_{e_{i+1}}(t),0\Big\},t, \\ & \max\{\boldsymbol{c}-A_{:,i+1},0\},i+1) + \hat{\Sigma}^2_{e_{i+1}}(t). \end{aligned} \tag{27}$$

Thus, for every possible s, $\boldsymbol{c}$, and i, we can update the solution to $\mathcal{P}_5(s,t,\boldsymbol{c},i)$ by

$$x^*_{e_{i+1}}(s,t,\boldsymbol{c},i) = \begin{cases} 0 & V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c},i) = V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c},i+1) \\ 1 & \text{otherwise.} \end{cases}$$

The recursion starts from condition $s=0$, $\boldsymbol{c}=\mathbf{0}$, and $i=|\mathcal{E}|$. *Algorithm 2* summarizes the main procedure. It is used to substitute Step 5 $\sim$ Step 7 of ESDP. Obviously, *Algorithm 2* is of $\mathcal{O}(|\mathcal{E}|\cdot|\mathcal{S}(t)|\cdot\prod_{k\in\mathcal{K}}c_k)$-complexity, i.e., $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$ are solved in polynomial time. In the following content, we will show the relations between $\mathcal{P}_3(t)$ and $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$, and analyze how the solution obtained by ESDP affects the regret $\mathrm{RE}(T)$ defined in $\mathcal{P}_2$.

Algorithm 2. DP for Solving $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$

Input: $\mathcal{S}(t)$, resource requirements $\{a_k^{(l,r)}\}_{k\in\mathcal{K},(l,r)\in\mathcal{E}}$, and scale-up statistics $\hat{\mathbf{\Upsilon}}(t)$ and $\hat{\mathbf{\Sigma}}(t)$
Output: Optimal solution to $\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$
$\forall i\in\{0,\ldots,|\mathcal{E}|\}$, $\boldsymbol{x}^*(s,t,\boldsymbol{c},i)\leftarrow\mathbf{0}$ **for** *s from 0 to* $\xi(t)\cdot\max_{t'\in\mathcal{T}}\{\max_{\boldsymbol{x}\in\Omega(t')}\|\boldsymbol{x}\|_1\}$ **do**
  for $\boldsymbol{c}'$ *from* $\mathbf{0}$ *to* $\boldsymbol{c}$ **do**
    $V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',|\mathcal{E}|)$ **is** 0 **if** $s==0$ **else** $-\infty$
    for *i from* $|\mathcal{E}|-1$ *to 0* **do**
      if $\boldsymbol{c}'==\mathbf{0}$ **then**
        $V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i)\leftarrow V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i+1)$
        continue
      end if
      $V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i)\leftarrow\max\{V^*_{\mathcal{P}_5}(\max\{s-\hat{\Upsilon}_{e_{i+1}}(t),0\},t,\max\{\boldsymbol{c}'-A_{:,i+1},0\},i+1)+\hat{\Sigma}^2_{e_{i+1}}(t),V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i+1)\}$
      if $V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i)\neq V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i+1)$ **then**
        $\boldsymbol{x}^*(s,t,\boldsymbol{c}',i)\leftarrow\boldsymbol{x}^*(\max\{s-\hat{\Upsilon}_{e_{i+1}}(t),0\},t,\max\{\boldsymbol{c}'-A_{:,i+1},0\},i+1)$
        $x^*_{e_{i+1}}(s,t,\boldsymbol{c}',i)\leftarrow 1$ `// Update`
        if $\mathbf{A}\boldsymbol{x}^*(s,t,\boldsymbol{c}',i)\leq\boldsymbol{c}'$ *is violated* **then**
          $V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i)\leftarrow V^*_{\mathcal{P}_5}(s,t,\boldsymbol{c}',i+1)$
          $\boldsymbol{x}^*(s,t,\boldsymbol{c}',i)\leftarrow\boldsymbol{x}^*(s,t,\boldsymbol{c}rsquo;,i+1)$
        end if
      end if
    end for
  end for
  `/* Assign the solution of` $i=0$ `to` $\mathcal{P}_4(s,t)$ `*/`
  $\boldsymbol{x}^*_{\mathcal{P}_4}(s,t)\leftarrow\boldsymbol{x}^*(s,t,\boldsymbol{c},0)$
end for
return $\{\boldsymbol{x}^*_{\mathcal{P}_4}(s,t)\}_{s\in\mathcal{S}(t)}$

3.3 Optimality and Regret Analysis

In this section, we will analyze the upper bound of $\mathrm{RE}(T)$ for ESDP when T goes to infinity. The result is based on the relations between the optimal solutions of several problems we defined above. The problems and their optimal solutions are summarized in Table 2 for quick reference. Our first result is that ESDP achieves the optimal statistical-based computation utility asymptotically with a certain probability.

TABLE 2
Probelms and Their Optimal Solutions

PROBLEMS	OPTIMAL SOLUTIONS
$\mathcal{P}_1$ (defined over $\mathcal{T}$)	$\{\boldsymbol{x}^*(t)\}_{t\in\mathcal{T}}$
$\mathcal{P}_2$ (defined over $\mathcal{T}$)	$\{\boldsymbol{x}^*(t)\}_{t\in\mathcal{T}}$, because $\mathcal{P}_1\equiv\mathcal{P}_2$
$\mathcal{P}_4(s,t)$	$\boldsymbol{x}^*_{\mathcal{P}_4}(s,t)$, also $\boldsymbol{x}^*(s,t,\boldsymbol{c},0)$
$\max\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$	$\boldsymbol{x}^*_{\mathcal{P}_4}(t)$ (by Step 8 of ESDP)
$\mathcal{P}_5(s,t,\boldsymbol{c},i)$	$\boldsymbol{x}^*(s,t,\boldsymbol{c},i)$ (by **Algorithm 2**)

Theorem 1. *(Probabilistic Asymptotical Optimality) By executing ESDP for problem $\mathcal{P}_3(t)$, $\lim_{t\to\infty}\tilde{\mathrm{U}}(\boldsymbol{x}^*_{\mathcal{P}_4}(t))$ is at least*

$$\max_{\boldsymbol{x}(t)\in\Omega(t)}\left\{\hat{\boldsymbol{\upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)+\sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}\right\} \tag{28}$$

with probability at most $\exp\left[-\frac{1}{3}(|\mathcal{L}|-\sum_{l\in\mathcal{L}}\rho_l(t))^2\right]$.

Proof. Note that $\tilde{\mathrm{U}}(\cdot)$ is the objective defined in $\mathcal{P}_3(t)$ and (28) is exactly the optimal objective of $\mathcal{P}_3(t)$ without the approximate parameter $\delta(t)$. Before our proof, we define the set

$$\Phi(t) := \{\boldsymbol{x}(t)\in\mathcal{X}\,|\,(2)\text{ holds at time }t\}. \tag{29}$$

Different from the set $\Omega(t)$, $\Phi(t)$ does not require constraint (4) to hold. Thus we have $\Omega(t)\subseteq\Phi(t)$. The following proof holds for every $t\in\mathcal{T}$.

By the definitions (9), (16), (17) and the fact $\tilde{Z}\in[0,1]^{|\mathcal{E}|}$, we have

$$\hat{\boldsymbol{\upsilon}}(t)\leq\frac{\hat{\mathbf{\Upsilon}}(t)}{\xi(t)}\leq\frac{1}{\xi(t)}\mathbf{1}+\hat{\boldsymbol{\upsilon}}(t). \tag{30}$$

Thus,

$$\begin{aligned} &\max_{\boldsymbol{x}(t)\in\Omega(t)}\hat{\boldsymbol{\upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)+\sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)} \\ &\leq\frac{1}{\xi(t)}\max_{\boldsymbol{x}(t)\in\Omega(t)}\hat{\mathbf{\Upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)+\sqrt{\hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}. \end{aligned} \tag{31}$$

Further, the RHS of (31) staisfies

$$\begin{aligned} &\max_{\boldsymbol{x}(t)\in\Omega(t)}\hat{\mathbf{\Upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)+\sqrt{\hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)} \\ &=\max_{s\in\mathcal{S}(t)}\max_{\hat{\mathbf{\Upsilon}}(t)(t)^{\mathrm{T}}\boldsymbol{x}(t)=s,\boldsymbol{x}(t)\in\Omega(t)}\left\{s+\sqrt{\hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}\right\} \\ &\leq\max_{s\in\mathcal{S}(t)}\max_{\hat{\mathbf{\Upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)\geq s,\boldsymbol{x}\in\Omega(t)}\left\{s+\sqrt{\hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}\right\} \\ &\leq\max_{s\in\mathcal{S}(t)}\max_{\hat{\mathbf{\Upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)\geq s,\boldsymbol{x}\in\Phi(t)}\left\{s+\sqrt{\hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}\right\}. \end{aligned} \tag{32}$$

The RHS of (32) is exactly $\max\{\mathcal{P}_4(s,t)\}_{s\in\mathcal{S}(t)}$. The upper bound of it should be $s^\star+\sqrt{\hat{\mathbf{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t)}$ if no channel is shut down forcibly, i.e., Step 10 $\sim$ Step 16 are not executed by ESDP. To quantify the probability that no channels are forcibly shut down, we use the result of Chernoff Bounds. The upper tail of Chernoff Bounds is stated as follows.

If $X_1, \ldots, X_n \in \{0,1\}$ *are mutually independent, then* $\forall x \geq \mu$, *where* $\mu := \mathbb{E}[\sum_i X_i]$, *we have*

$$\Pr\left[\sum_i X_i \geq x\right] \leq e^{x-\mu}\left(\frac{\mu}{x}\right)^x.$$

Based on this conclusion, we can further derive that

$$\Pr\left[\sum_i X_i \geq (1+\varepsilon)\mu\right] \leq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}}\right)^{\mu}, \tag{33}$$

where $\varepsilon \geq 0$.

With the Taylor-series expansion for $\ln(x+1)$ at $x = 0$, we have

$$\ln(1+\varepsilon) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}\varepsilon^n}{n} = \varepsilon - \frac{\varepsilon^2}{2} + \frac{\varepsilon^3}{3} - \ldots \geq \varepsilon - \frac{\varepsilon^2}{2}.$$

Thus, we have

$$\frac{1}{\ln(1+\varepsilon)} \leq \frac{1}{\varepsilon(1-\frac{1}{2}\varepsilon)} = \frac{1}{\varepsilon} + \frac{1}{2-\varepsilon} \leq \frac{1}{\varepsilon} + \frac{1}{2}.$$

Applying the inequality to the RHS of (33), we can get

$$\Pr\left[\sum_i X_i \geq (1+\varepsilon)\mu\right] \leq \exp\left(-\frac{\varepsilon^2\mu}{3}\right). \tag{34}$$

Replacing X_i with $\mathbb{1}_l$ and $(1+\varepsilon)\mu$ with $|\mathcal{L}|$, (34) is transformed into

$$\Pr\left[\sum_{l\in\mathcal{L}} \mathbb{1}_l = |\mathcal{L}|\right] \leq \exp\left[-\frac{1}{3}\left(|\mathcal{L}| - \sum_{l\in\mathcal{L}} \rho_l(t)\right)^2\right], \tag{35}$$

which exactly quantifies the probability that every port yields at least one job. In this case, no channel $(l,r) \in \mathcal{E}$ is shut down forcibly. Thus, with this probability, the RHS of (32) staisfies

$$\begin{aligned}
&\max_{s(t)\in\mathcal{S}} \max_{\hat{\boldsymbol{\Upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t)\geq s, \boldsymbol{x}(t)\in\Phi(t)} \left\{s + \sqrt{\hat{\boldsymbol{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}\right\}\\
&= s^\star + \sqrt{\hat{\boldsymbol{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t)} \quad \triangleright s^\star \textit{ staisfies (21) with prob. } (35)\\
&\leq \hat{\boldsymbol{\Upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t) + \sqrt{\hat{\boldsymbol{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t)} \qquad \triangleright (20)\\
&\leq (\mathbf{1} + \xi(t)\hat{\boldsymbol{\upsilon}}(t))^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t) + \sqrt{\hat{\boldsymbol{\Sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t)} \qquad \triangleright (30)\\
&\leq \xi(t)\left(\delta(t) + \hat{\boldsymbol{\upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t) + \sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*_{\mathcal{P}_4}(t)}\right).
\end{aligned} \tag{36}$$

Combining the result of (31), (32), and (36), the following inequality holds for all the time $t \in \mathcal{T}$:

$$\max_{\boldsymbol{x}(t)\in\Omega(t)} \left\{\hat{\boldsymbol{\upsilon}}(t)^{\mathrm{T}}\boldsymbol{x}(t) + \sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}(t)}\right\} \leq \tilde{\mathrm{U}}\left(\boldsymbol{x}^*_{\mathcal{P}_4}(t)\right). \tag{37}$$

When $t \to \infty$ and $\delta(t) \to 0$, the result is tightly bounded. □

The theorem shows that ESDP can achieve approximately optimal statistical-based computation utility at each time slot with a certain probability. This optimality is important for minimizing the regret because it builds the upper bound of the optiaml computation utility $\tilde{\boldsymbol{\upsilon}}^{\mathrm{T}}\boldsymbol{x}^*(t)$ at each time t. The probabilistic regret upper bound is given by the following theorem.

Theorem 2. *(Regret Upper Bound under Certain Conditions) By executing the ESDP algorithm, as* $T \to \infty$, $\mathrm{RE}(T)$ *is upper bounded by*

$$\mathcal{O}\left(\ln T \cdot \frac{|\mathcal{E}| \cdot (\ln \boldsymbol{x}^*)^2}{\min_{t\in\mathcal{T}} \Delta\left(\boldsymbol{x}^*_{\mathcal{P}_4}(t)\right)}\right) \tag{38}$$

with probability at most $\exp(-\frac{1}{3}(|\mathcal{L}| - \sum_{l\in\mathcal{L}} \rho_l(t))^2)$. *In (38),* $\Delta(\boldsymbol{x}^*_{\mathcal{P}_4}(t))$ *is introduced by (6), and* $\boldsymbol{x}^*$ *is defined as*

$$\boldsymbol{x}^* := \operatorname*{argmax}_{\forall t\in\mathcal{T}:\boldsymbol{x}^*(t)} \|\boldsymbol{x}^*(t)\|_1. \tag{39}$$

Proof. The result is immediate with *Theorems 1* and *4.4* of [30]. The technique is to define three events $\mathcal{A}(t)$, $\mathcal{B}(t)$, $\mathcal{C}(t)$ at each time t:

$$\begin{cases}
\mathcal{A}(t) := \left\{\left|(\tilde{\boldsymbol{\upsilon}} - \hat{\boldsymbol{\upsilon}}(t))^{\mathrm{T}}\boldsymbol{x}^*(t)\right| \geq \sqrt{\hat{\boldsymbol{\sigma}}^2(t)^{\mathrm{T}}\boldsymbol{x}^*(t)}\right\}\\
\mathcal{B}(t) := \left\{\Delta(\boldsymbol{x}^*_{\mathcal{P}_4}(t)) \leq 4\delta(t)\right\}\\
\mathcal{C}(t) := \overline{\mathcal{A}(t)} \cup \overline{\mathcal{B}(t)},
\end{cases} \tag{40}$$

and study the sum of the upper bound of $\mathrm{RE}(T)$ under these events respectively. Which of these events will happen depends on the accuracy of the estimations $\hat{\boldsymbol{\upsilon}}(t)$. Considering that the proof is similar to the proof presented in [30], we will not demonstrate the complete proof here. □

3.4 Extending to Gang Scheduling

ESDP 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[2] of a job are successfully scheduled, the job could be launched. Gang scheduling is required for multi-server jobs such as distributed DNN trainings and Message Passing Interface (MPI) jobs. Take the DNN training with the parameter server (PS)-worker architecture as an exmaple, at least one PS and one worker are successfully scheduled, the training could start.

In the following, we show briefly how Gang Scheduling can be modeled. Let us re-define $\boldsymbol{x}(t)$ as

$$\boldsymbol{x}(t) := \left[x^l_{(q,r)}(t)\right]_{q\in\mathcal{Q}_l, r\in\mathcal{R}_l, l\in\mathcal{L}},$$

where $\mathcal{Q}_l$ stores the indices of tasks for the type-l job. Then, we have the following new constraints:

2. 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.

$$\begin{cases} \sum_{r\in\mathcal{R}_l}\sum_{q\in\mathcal{Q}_l} x^l_{(q,r)}(t) \geq m_l(t) & \forall l, t \\ \sum_{l\in\mathcal{L}}\sum_{q\in\mathcal{Q}_l} a^l_{(q,k)} x^l_{(q,r)}(t) \leq c_{(k,r)} & \forall k, r, t, \end{cases}$$

where $m_l(t)$ is the minimum number of tasks to be executed, $a^l_{(q,k)}$ is the requirement of the type-k resource for the q-th task of the type-l job, and $c_{(k,r)} \in \mathbb{N}^+$ is the number of the type-k computing devices available to server r. The same to (2), the new constraint also has the form of $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{c}$. The new problem can be solved by a similar approach to ESDP after several mathematical transformations.

4 Numerical Results

In this section, we conduct extensive simulations to validate the performance of ESDP. We first verify the performance of ESDP against several handcrafted benchmarking policies on the AOU. Then, we analyze the generality and robustness of it under different cluster settings. The simulations are conducted on a server with 48 Intel Xeon Silver 4214 CPUs, 256 GB memory, and 2 Tesla P40 GPUs.

Traces. We use the data from cluster-trace-v2018 of the Alibaba Cluster Trace Program[3] to generate our experiment experiments. Specifically, we leverage the specifications of the machines, the arrival patterns and resource requirements of different kind of jobs to set resource capacities $\{c_k\}_{k\in\mathcal{K}}$, job arrival probabilities $\{\rho_l\}_{l\in\mathcal{L}}$, and device requirements of jobs $\{\boldsymbol{a}_k\}_{k\in\mathcal{K}}$.

Default Scenario Settings. In default settings, our simulation environment has 40 servers, each equipped with 3 types of computing devices (CPUs, MEM, and GPUs), and 8 multi-sever job types of different resource requirements. Job arrival probabilities $\{\rho_l\}_{l\in\mathcal{L}}$ are setted to adjust the job arrival status with Bernoulli Distributions. $\{\rho_l\}_{l\in\mathcal{L}}$ are applied on the basis of the actual arrival patterns from the trace to increase stochasticity. Although $\{\boldsymbol{a}_k\}_{k\in\mathcal{K}}$ are retrieved from the trace data, we still need to set the equipped resource limits to eliminate inappropriate settings which could lead to the solution space of problem $\mathcal{P}_1$ being null. Specifically, we denote by $\overline{\|\mathbf{A}\|_2}$ and $\underline{\|\mathbf{A}\|_2}$ the upper bound and the lower bound of $\{A_{ij}\}_{\forall i,j}$, and set them to 2 and 1 in default, respectively. Correspondingly, we use $\overline{\|\boldsymbol{c}\|_2}$ and $\underline{\|\boldsymbol{c}\|_2}$ to represent the upper bound and the lower bound of $\boldsymbol{c}$, and set them to 2 and 1 in default, respectively. The settings of these bounds are normalized. For each computation utility $v_{(l,r)}$, $(l,r) \in \mathcal{E}$, we generate it from a Normal distribution as follows:

$$\mathcal{N}\left(\mu_{(l,r)} \sim U(0.1, 1), \sigma_{(l,r)} = \frac{\mu_{(l,r)}}{2}\right).$$

Correspondingly, for each device type $k \in \mathcal{K}$, the operating cost $f_k(a_k^{(l,r)})$ is generated from the Normal distribution $\mathcal{N}(0.5, 0.1)$. Note that the settings we adopt are only required to make the stochastic problem $\mathcal{P}_1$ feasible. ESDP is robust enough to make scheduling decisions of high system efficiency. The robustness will be demonstrated in detail in the following content. Besides, note that ESDP has no assumptions on the distributions of the valuations $\{v_{(l,r)}\}_{(l,r)\in\mathcal{E}}$. The Normal distributions we used here are only for problem construction. The default time slot length is 2000.

3. https://github.com/alibaba/clusterdata

TABLE 3
Default Parameter Settings

Param.	Value	Param.	Value
$\lvert\mathcal{L}\rvert$	8	$\lvert\mathcal{R}\rvert$	40
$\overline{\|\mathbf{A}\|_2}$	2	$\underline{\|\mathbf{A}\|_2}$	1
α	0.5	$\{\rho_l\}_{l\in\mathcal{L}}$	0.9
$\overline{\|\boldsymbol{c}\|_2}$	2	$\underline{\|\boldsymbol{c}\|_2}$	1
K	3	T	2000
$\{f_k\}_{k\in\mathcal{K}}$	$\sim\mathcal{N}(0.5, 0.1)$	$\{\rho_{(l,r)}\}_{(l,r)\in\mathcal{E}}$	0.1

Default Algorithmic Settings. When we implement ESDP, $\max_{t'\in\mathcal{T}}\{\max_{\boldsymbol{x}\in\Omega(t')}\|\boldsymbol{x}\|_1\}$ is calculated as $\alpha|\mathcal{E}|$, where $\alpha \in [0,1]$ is a coefficient by default to be 0.5. We set $\delta(t)$ and $g(t)$ as $(\ln(\ln(t+1)+1)+1)^{-1}$ and $\ln(t+1)+4\ln(\ln(t+1)+1)\cdot\alpha|\mathcal{E}|$, respectively in default. Considering that those two sequences significantly affect the effectiveness of ESDP, we will comprehensively discuss their variations in Section 4.2.

Baselines. ESDP is compared with the following handcrafted baselines.

- *The Accumulative Utility First (HAUF)*: HSWF is different from ESDP in the following ways. At each time t, $\boldsymbol{Z}(t)$ is estimated as the average of historical observations. With the estimate, HSWF ranks each port in the descending order of $\sum_{r\in\mathcal{R}_l} x_{(l,r)}(t)(Z_{(l,r)}(t) - \sum_{k\in\mathcal{K}} f_k(a_k^{(l,r)}))$, and set the corresponding $x_{(l,r)}(t)$ as 1 in turn until (2) can not be satisfied.
- *The Lowest Cost First (LCF)*: Similar to HSWF, at each time t, $\boldsymbol{Z}(t)$ is estimated as the average of historical observations. Then, LCF ranks each job (non-empty port) in the ascending order of cost $\sum_{k\in\mathcal{K}} f_k(a_k^{(l,r)})$, and set the corresponding $x_{(l,r)}(t)$ as 1 in turn until (2) can not be satisfied.
- *The Longest Waiting Time First (LWTF)*: LWTF is different from ESDP in two ways. First, $\boldsymbol{Z}(t)$ is estimated as the average of historical observations. Second, LWTF ranks each port in the descending order of the waiting time of jobs yield from that port, and set the corresponding $x_{(l,r)}(t)$ as 1 in turn until (2) can not be satisfied.

Note that we do not implement heuristics such as the Genetic Algorithm for comparison. This is because $\mathcal{P}_1$ is a stochastic optimization problem and traditional heuristics need to be revised carefully to match it. All of the three baselines use a similar method to estimate the historical valuation of each channel. With the estimate, the stochastic optimization problem is transformed into a deterministic one. Essentially, heuristics can be implemented by following a similar approach. However, a big problem that cannot be ignored is that heuristics are time-consuming with a non-polynomial complexity. Heuristics have to be called in every time slot, which could be very slow when the time slot length is large.

4.1 Performance Verification

In the first part, we demonstrate how the average AOU changes as time slot increases. As Fig. 2 shows, ESDP outperforms the baselines by up to nearly 73%, 36%, and 28%, respectively within 8000 time slots. In the beginning, HSWF performs better than EDSP, but as the time slots increase, ESDP gradually outperforms HSWF, and the gap between

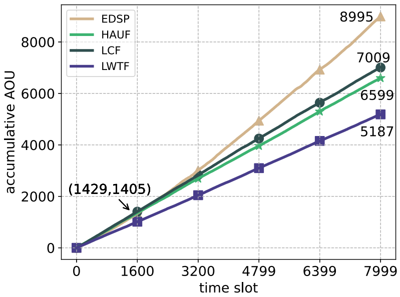

Fig. 2. The AOU versus time slots.

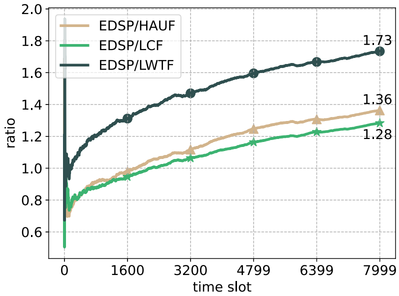

Fig. 3. The ratio between the AOUS.

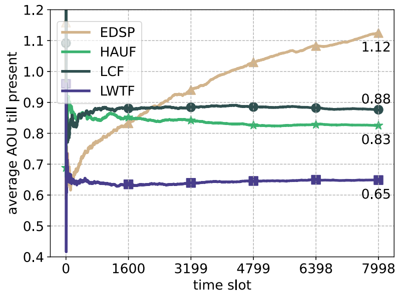

Fig. 4. The average AOU versus time slots.

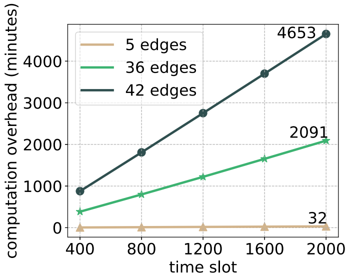

Fig. 5. Computation overheads.

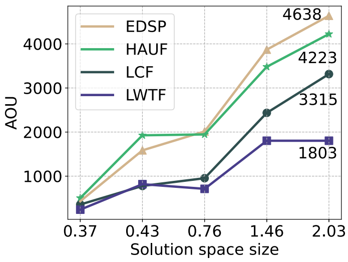

Fig. 6. AOU versus $\mathcal{X}$.

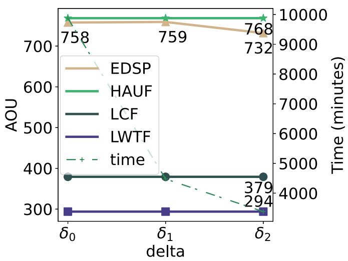

Fig. 7. AOU versus $\{\delta(t)\}_{t\in\mathcal{T}}$.

them keeps widening. ESDP is able to surpass HSWF because that, unlike HSWF, which does not adjust its strategy, EDSP constantly updates its strategy with the explorated valuation distributions. Besides, we also demonstrate the ratio between the AOU achieved by ESDP and the baselines in Fig. 3. We can conclude that, the performance of ESDP increases significantly when the time slots available to explore increase. The reason lies in that more time slots leads to more approximate estimate to $\{v_{(l,r)}\}_{(l,r)\in\mathcal{E}}$.

In Fig. 4, we calculate the average AOU in this way: for each time slot length T, the y-axis value is $\frac{1}{T}\sum_{t=1}^{T}\mathrm{U}(\boldsymbol{x}(t))$. Different from the baselines, the average AOU of ESDP increases steep and later flattens, which verifies that the AOU converges to an underlying upper bound (the AOU achieved by the oracle). The computation overhead of ESDP under different scales of the bipartite graph is shown in Fig. 5. It is interesting to find that the rewards oscillate at the beginning time slots. One of the leading factors is that ESDP is boosted with a well designed initial solution. No surprisingly, the rewards achieved in the beginning can be easily surpassed when the time slot is sufficiently large.

4.2 Sensitivity Analysis

In this section, we give a brief analysis on several important parameter settings. The first problematic parameter we test is the size of the solution space $\mathcal{X}$, which is tuned by $\mathbf{A}$ and $\boldsymbol{c}$. Recall that $\mathbf{A} := \{\boldsymbol{a}_k\}_{k\in\mathcal{K}}$ and $\boldsymbol{c} := \{c_k\}_{k\in\mathcal{K}}$ are respectively the device requirements of each type of jobs and the device capacities of the cluster. The x-axis of Fig. 6 is $\|\mathbf{A}^{-1}\boldsymbol{x}\|_2$. Without doubt, the AOU increases with the growth of $\mathcal{X}$ for all the algorithms because the number of can-be-processed jobs increase. Even though, ESDP has the highest growth in the AOU because it can fully exploit the estimated valuations.

The first algorithmic parameter we pay attention to is the sequence $\{\delta_t\}_{t\in\mathcal{T}}$, which is used to relax the NP-hard problem $\mathcal{P}_2$ to a polynomial one. The three $\{\delta_t\}_{t\in\mathcal{T}}$ shown in Fig. 7 are $(\ln(t+1)+1)^{-1}$, $(\ln(\ln(t+1)+1))^{-1}$, and $(\ln((\ln(\ln(t+1)+1))+1)+1)^{-1}$, respectively. Fig. 7 demonstrates that different settings of the sequence has little effect on the performance of ESDP, but strong affect on the computation overhead. Another algorithmic parameter we are interested on is $\{g(t)\}_{t\in\mathcal{T}}$, which is used to estimate the variance (10). The three $\{g(t)\}_{t\in\mathcal{T}}$ demonstrated in Fig. 8 are $\ln(t+1)+4\ln(\ln(t+1)+1)\cdot\alpha|\mathcal{E}|$, $4\ln(\ln(t+1)+1)\cdot\alpha|\mathcal{E}|$, and $\ln(t+1)$, respectively. We can find that the third setting has an overwhelming advantage. The reason is that, theoretically, $g(t)$ acts as a balancer between exploration and exploitation. A smaller $g(t)$ leads to a higher tendency to exploitation.

Figs. 9 and 10 demonstrate the impact of job arrival rate ρ and the density of the bipartite graph. From Fig. 9 we can find that, with the increase of ρ, the AOU achieved by nearly all the algorithms also goes up. The result is obvious because

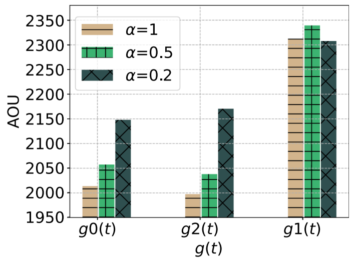

Fig. 8. AoU versus $\{g(t)\}_{t\in\mathcal{T}}$.

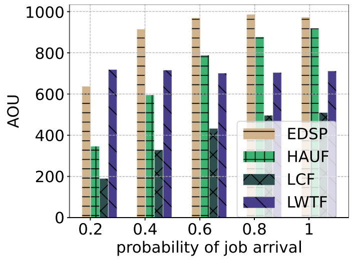

Fig. 9. AoU versus ρ.

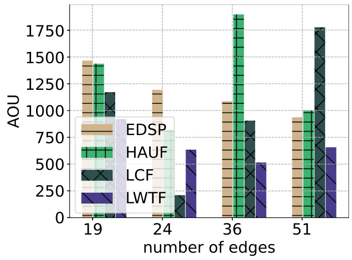

Fig. 10. AoU versus $|\mathcal{E}|$.

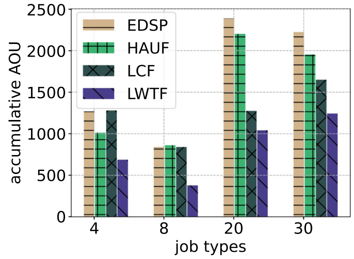

Fig. 11. AoU versus $|\mathcal{L}|$.

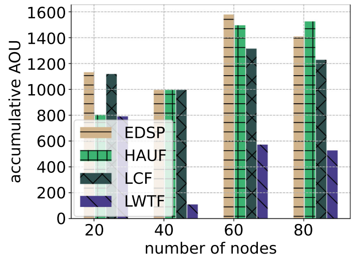

Fig. 12. AoU versus $|\mathcal{R}|$.

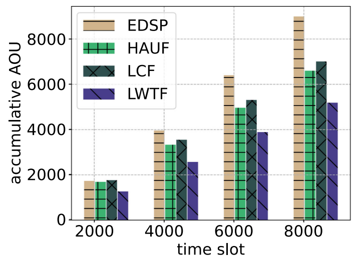

Fig. 13. AoU versus T.

more jobs can be processed within service capacities when ρ increases. It is interesting to find that, increasing the job arrival probability can lead to a high resource utilization, thereby increasing the AoU. However, a large job arrival probability also brings in a fierce resource contention. A direct consequence of it is that, for ESDP, many elements in the vector $\boldsymbol{x}(t)$ fall into the interior of $\mathcal{X}$, rather than the boundaries, thereby leading to a reward reduction. The phenomenon can be observed when moving ρ from 0.8 to 1.0. Fig. 10 demonstrates the impact of the service locality constraint. When the number of edges increases in the bipartite graph, which means the service locality constraint is relaxed, the solution space $\mathcal{X}$ becomes larger. It significantly increases the difficulty of searching the optimal solution for ESDP.

4.3 Scalability Analysis

In this section, we demonstrate the performance of ESDP under different scales of scenario settings. Figs. 11 and 12 demonstrate the impact of the scale of the bipartite graph $\mathcal{G}$. First, we observe that, whatever the number of the node is, ESDP takes the leading position. Besides, as $|\mathcal{R}|$ becomes larger, all the algorithms obtain a relatively larger cumulative AoU. The result is evident because a large cluster can provide sufficient computing devices, which leads to jobs being fully served. It is worth noting that, when $|\mathcal{R}|$ increases from 60 to 80, HAUF achieves a higher AoU than ESDP. The reason of the weak position of ESDP is that the solution space $\mathcal{X}$ increases with the node number, and ESDP need a larger time slot length to learn the underlying distributions of the computation utilities. Fig. 11 shows that the number of job types, i.e., $|\mathcal{L}|$, has a similar impact to $|\mathcal{R}|$ in terms of the performance of ESDP.

Fig. 13 shows that, whatever the parameter settings, ESDP 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 ESDP to learn the underlying distributions,thereby increasing the reward in the gradient ascent directions.

5 Related Works

Online resource allocation for co-located jobs is always the focus of attention for both industrial and research communities. Online algorithms which yield a nice theoretical performance bound can be divided into two categories.

In the first category, the online algorithms are sophistically designed for specific job types, including multi-stage data query and analysis workflows (which are organized as DAGs) [19], [33], service function chains (SFCs) [34], distributed deep neural network training jobs [12], [13], [17], [22], [35], [36], [37], etc. In these works, the algorithms are proposed by formulating combinatorial optimization problems with scenario-oriented constraints, and their performance guarantees are provided by the adopted optimization techniques. A typical work is [12], where the authors propose an algorithm, named `SPIN`, with a rounding-based randomized approximation approch, to schedule the placement-sensitive

Bulk Synchronous Parallel (BSP) jobs. Their design is built on the relaxation of the Gang scheduling constraints and the job completion time (JCT) is minimized with linear programming. The authors develop an algorithm which is $\mathcal{O}(\ln|\mathcal{M}|)$ -approximate with high probability, where $\mathcal{M}$ is the set of computing devices.

In the second category, the job type is not specified, but their theoretical superiority for job co-location and resource contention is highlighted. The algorithms are designed with different theoretical basis, including online approximate algorithms [18], [20], [23], Online Convex Optimization (OCO) techniques [21], [38], game-theoretical approaches [39], Multi-Armed Bandit (MAB) theories and DRL-based algorithms [16], [40], etc. In these works, the performance of the proposed algorithms are usually analyzed with approximate ratio, competitive ratio, Price of Anarchy (PoA), and regret. A typical recent work is [21]. Among these, the most similar work to ours is [38]. This work presents an online algorithm based on the MAB theories and the OCO techniques, which aims at make online resource allocation decisions without knowing future job arrivals according to machine availabilities. The proposed algorithm can achieve $\mathcal{O}(\sqrt{T\log\frac{T}{\delta}})$ regret with probability $1-\delta$ while guaranteeing a small fit for both the single-job and multi-job cases over a duration of T time slots. The main differences between this work and ours are summarized as follows.

- Although [38] considers the fluctuated machine service capacities, its system model does not differentiate computing device types. The authors adopt the combinatorial MAB framework to address the resource allocation problem while our algorithm ESDP is built on the AESCB policy.
- In [38], the authors propose an algorithm which has a $\mathcal{O}(\sqrt{T\log\frac{T}{\delta}})$ regret with probability $1-\delta$ for concave utility functions. By contrast, ESDP has a logarithmic regret $\mathcal{O}(\ln T)$ for linear separatable utilities. Although ESDP has a lower regret in terms of the time slot length T, its performance guarantee does suitable for the non-linear cases.

6 Conclusion

In this paper, we study the multi-server job scheduling problem without knowing the actual processing speed distributions apriori. We formulate the problem as a stochastic cumulative overall utility maximization program and cast it into the framework of online learning. We propose an online algorithm, termed as ESDP, to learn the underlying processing speed distributions and use the exploited statistics to guide the scheduling decisions. ESDP adopts dynamic programming to solve several well designed approximated deterministic problems in polynomial time. We prove that ESDP has a best possible regret, i.e., $\ln(T)$. The performance of ESDP is also validated with extensive simulations. Moreover, extending ESDP to general non-linear utilities might be an interested future research direction.

References

[1] S.-X. Zou et al., "Distributed training large-scale deep architectures," in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2017, pp. 18–32.
[2] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.
[3] D. Yan, J. Cheng, Y. Lu, and W. Ng, "Effective techniques for message reduction and load balancing in distributed graph computation," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 1307–1317.
[4] W. Xiao et al., "TUX^2: Distributed graph computation for machine learning," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 669–682.
[5] M. Tirmazi, N. Deng, M.-E. Haque, Z.-G. Qin, S. Hand, and A. Barker, "Google cluster-usage traces V3," 2019. [Online]. Available: https://github.com/google/cluster-data
[6] M. Ribeiro, K. Grolinger, and M. A. Capretz, "MLaaS: Machine learning as a service," in *Proc. IEEE 14th Int. Conf. Mach. Learn. Appl.*, 2015, pp. 896–902.
[7] I. Baldini et al., *Serverless Computing: Current Trends and Open Problems*. Singapore: Springer, 2017, pp. 1–20.
[8] E. Jonas et al., "Cloud programming simplified: A berkeley view on serverless computing," 2019, *arXiv: 1902.03383*.
[9] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Gener. Comput. Syst.*, vol. 37, pp. 141–147, 2014.
[10] A. G. Kumbhare et al., "Prediction-based power oversubscription in cloud platforms," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 473–487. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/kumbhare
[11] J. V. Gautam, H. B. Prajapati, V. K. Dabhi, and S. Chaudhary, "A survey on job scheduling algorithms in big data processing," in *Proc. IEEE Int. Conf. Elect. Comput. Commun. Technol.*, 2015, pp. 1–11.
[12] 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 *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1053–1062.
[13] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 495–503.
[14] I. Attiya, M. Abd Elaziz, and S. Xiong, "Job scheduling in cloud computing using a modified Harris hawks optimization and simulated annealing algorithm," *Comput. Intell. Neurosci.*, vol. 2020, 2020, Art. no. 3504642.
[15] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling," *IEEE Trans. Cybern.*, vol. 51, no. 4, pp. 1797–1811, Apr. 2021.
[16] S. Liang, Z. Yang, F. Jin, and Y. Chen, "Data centers job scheduling with deep reinforcement learning," in *Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining*, 2020, pp. 906–917.
[17] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 481–498.
[18] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.
[19] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 864–872.
[20] K. Psychas and J. Ghaderi, "Scheduling jobs with random resource requirements in computing clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2269–2277.
[21] Y. Liu, H. Xu, and W. C. Lau, "Online job scheduling with resource packing on a cluster of heterogeneous servers," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1441–1449.
[22] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 495–503.
[23] 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 Trans. Netw.*, vol. 27, no. 6, pp. 2472–2485, Dec. 2019.
[24] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends® Mach. Learn.*, vol. 4, no. 2, pp. 107–194, 2012.
[25] X. Tan, B. Sun, A. Leon-Garcia, Y. Wu, and D. H. Tsang, "Mechanism design for online resource allocation: A unified approach," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, pp. 1–46, Jun. 2020.
[26] H. Zhao et al., "DPoS: Decentralized, privacy-preserving, and low-complexity online slicing for multi-tenant networks," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2021.3074934.

[27] H. Zhao, S. Deng, Z. Xiang, and J. Yin, "Online social welfare maximization with spatio-temporal resource mesh for serverless," 2021, *arXiv:2112.02456*.

[28] R. Combes, M. S. Talebi, A. Proutiere, and M. Lelarge, "Combinatorial bandits revisited," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2116–2124.

[29] R. Degenne and V. Perchet, "Combinatorial semi-bandit with known covariance," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2972–2980.

[30] T. Cuvelier, R. Combes, and E. Gourdin, "Statistically efficient, polynomial-time algorithms for combinatorial semi-bandits," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 1, 2021, Art. no. 09.

[31] T. Roughgarden, "Algorithmic game theory," *Commun. ACM*, vol. 53, no. 7, pp. 78–86, 2010.

[32] B. Sun, A. Zeynali, T. Li, M. Hajiesmaili, A. Wierman, and D. H. Tsang, "Competitive algorithms for the online multiple knapsack problem with application to electric vehicle charging," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 3, 2020, Art. no. 51.

[33] Z. Hu, B. Li, C. Chen, and X. Ke, "FlowTime: Dynamic scheduling of deadline-aware workflows and ad-hoc jobs," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 929–938.

[34] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundant placement for microservice-based applications at the edge," *IEEE Trans. Services Comput.*, vol. 15, no. 3, pp. 1732–1745, May/Jun. 2022.

[35] A. Qiao et al., "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *Proc. 15th USENIX Symp. Operating Syst. Des. Implementation*, 2021, pp. 1–18.

[36] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Meng, and W. Lin, "DL2: A deep learning-driven scheduler for deep learning clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 1947–1960, Aug. 2021.

[37] M. Yu, C. Wu, B. Ji, and J. Liu, "A sum-of-ratios multi-dimensional-knapsack decomposition for DNN resource scheduling," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[38] H. Xu, Y. Liu, W. C. Lau, J. Guo, and A. Liu, "Efficient online resource allocation in heterogeneous clusters with machine variability," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 478–486.

[39] R. Burra, C. Singh, and J. Kuri, "Service scheduling for bernoulli requests and quadratic cost," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2584–2592.

[40] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. M. Leung, "Tailored learning-based scheduling for kubernetes-oriented edge-cloud system," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

Hailiang Zhao received the BS degree from the School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China, in 2019. He is currently working toward the PhD 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, the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Mobile Computing*, etc. He has been a recipient of the Best Student Paper Award of IEEE ICWS 2019. He is a reviewer of the *IEEE Transactions on Services Computing* and *Internet of Things Journal*.

Shuiguang Deng (Senior Member, IEEE) received the BS and PhD degrees in computer science from Zhejiang University, China, in 2002 and 2007, respectively. He is currently a full professor with the College of Computer Science and Technology, Zhejiang University. He previously worked with 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 Transactions 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 the IET.

Feiyi Chen received the BS degree from the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), Guangzhou, China, in 2021. She is currently working toward the master degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. Her research interests include cloud computing, edge computing, and distributed systems.

Jianwei Yin received the PhD 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 (Fellow, IEEE) is a full professor of computer science (informatics) with a focus on Internet Technologies heading the Distributed Systems Group, TU Wien. He is the founding co-editor-in-chief of the *ACM Transactions on Internet of Things* (ACM TIoT) as well as the editor-in-chief of the *Computing* (Springer). He is an associate editor of the *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 the *IEEE Internet Computing* and *IEEE Computer*. He is a 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 the chairman of the Informatics Section, as well as an Asia-Pacific Artificial Intelligence Association (AAIA) president (2021) and a 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 (Fellow, IEEE) is Peter Nicol Russell chair professor of Computer Science and director of the Centre for Distributed and High-Performance Computing with the University of Sydney. To date, he has published 700 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. He is currently the editor in chief for *ACM Computing Surveys* and served in the past as editor in chief for *IEEE Transactions on Computers* (2010–2014) and *IEEE Transactions on Sustainable Computing* (2016–2020). He is a decorated scholar with numerous accolades including fellowship of the American Association for the Advancement of Science, and the Institution of Engineering and Technology. Also, he is a fellow of the Australian Academy of Science, Royal Society of New South Wales, foreign member of Academia Europaea, and member of the European Academy of Sciences and Arts. Some of he recent awards include the *New South Wales Premiers Prize of Excellence in Engineering and Information and Communications Technology* (2019) and the *Research Innovation Award*, *IEEE Technical Committee on Cloud Computing* (2021). His research interests lie in parallel and distributed computing, networking, and complex systems.