Optimal Capacity Provisioning for Online Job Allocation with Hard Allocation Ratio Requirement

Han Deng and I-Hong Hou

***Abstract*—The problem of allocating jobs to appropriate servers in cloud computing is studied in this paper. We consider that jobs of various types arrive in some unpredictable pattern and the system is required to allocate a certain ratio of jobs. In order to meet the hard allocation ratio requirement in the presence of unknown arrival patterns, one can increase the capacity of servers by expanding the size of data centers. We then aim to find the minimum capacity needed to meet a given allocation ratio requirement.**

We study this problem for both systems with persistent jobs, such as video streaming, and systems with dynamic jobs, such as database queries. For both systems, we propose online job allocation policies with low complexity. For systems with persistent jobs, we prove that our policies can achieve a given hard allocation ratio requirement with the least capacity. For systems with dynamic jobs, the capacity needed for our policies to achieve the hard allocation ratio requirement is close to a theoretical lower bound. Two other popular policies are studied, and we demonstrate that they need at least an order higher capacity to meet the same hard allocation ratio requirement. Simulation results demonstrate that our policies remain far superior than the other two even when jobs arrive according to some random process.

***Index Terms*—Job allocation; online algorithm; competitive ratio; cloud computing**

I. Introduction

In this paper, we discuss the problem of online job allocation for cloud computing where each job can only be served by a subset of servers. Such a problem exists in many emerging Internet services, such as YouTube, Netflix, etc. For example, in the case of YouTube, each video is replicated only in a small number of servers, and each server can only serve a limited number of streams simultaneously. When a user accesses YouTube and makes a request to watch a video, this request needs to be allocated to one of the servers that not only stores the video but also has remaining capacity to process this request. If no server can process this request, the request is dropped, which can significantly impact user satisfaction.

Han Deng is with Houston Methodist Research Institute, Houston, Texas, 77030, USA. Email: hdeng@houstonmethodist.org

I-Hong Hou is with Department of ECE, Texas A&M University, College Station, Texas, 77843-3128, USA. Email: ihou@tamu.edu

This material is based upon work supported in part by NSF under contract number CNS-1719384, and the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number W911NF-15-1-0279. Part of this work has been presented in IEEE Infocom 2016 [1].

There are many studies on obtaining statistics about user behaviors, video popularity, and access locality [2]–[6]. These studies provide important insights about system planning. However, they are not sufficient for job allocation. It is usually difficult to predict which video will go viral and generate most requests. Therefore, online job allocation policy that makes decisions solely based on current system state is needed.

The problem of online job allocation has attracted much attention. Most current studies study the performance of online policies by comparing the number of allocated jobs under online policies against that of the an offline policy with full knowledge about all future arrivals. For example, Karp, Vazirani and Vazirani [7] propose an online policy that is guaranteed $1 - \frac{1}{e} \approx 63\%$ of jobs, given that the offline policy allocates all jobs. They also show that no online policy can guarantee to allocate more jobs than their proposed policy.

These studies are still insufficient for many practical scenarios. In particular, one can argue that most practical applications demand a much higher ratio of allocated jobs than 63%. In order to meet this demand, current practice is to add redundancy and increase the capacity of data centers to accommodate unpredictable patterns of job arrivals. Motivated by this observation, we seek to address the following question: How much capacity is needed to guarantee the allocation of, say, 95% of jobs?

We address this problem for two different categories of jobs: persistent jobs and dynamic jobs. Persistent jobs, such as requests for video streaming, require immediate service upon arrival, and need continuous service after being allocated to a server. On the other hand, dynamic jobs, such as database queries, have small execution time and leave the system upon completion. Dynamic jobs can also tolerate a small amount of delay, as long as they are completed before their respective deadlines. For each of these two categories of jobs, we formulate the problem of job allocation as a linear programming problem. The uncertainty in future job arrivals corresponds to the uncertainty in some of the parameters. Also, increasing the capacity of the servers corresponds to relaxing some of the constraints in the linear programming problem.

For the allocation of persistent jobs, we propose two simple online allocation policies and derive closed-form expressions for their performance. Specifically, we prove that, in order to allocate at least $1 - \frac{1}{\theta}$ of jobs, the two

policies only need to increase the capacity by $\ln\theta$ times. We also prove that no online policy can guarantee to allocate the same ratio of jobs with less capacity, and hence our policies are optimal.

We further evaluate two popular online job allocation policies for allocating persistent jobs. Surprisingly, we prove that, to guarantee to allocate at least $1-\frac{1}{\theta}$ of jobs, these two policies require at least $\theta-1$ times capacity. Therefore, they are both an order worse than our policies.

For the allocation of dynamic jobs, we also propose a simple online allocation policy. We show that, when the deadlines of all jobs are at most T, and each server is constrained by both its computation capacity and its storage to buffer unfinished jobs, then our policy only needs $\ln\theta+\ln(T+1)$ times capacity to guarantee the allocation of $1-\frac{1}{\theta}$ of jobs. We also prove that the capacity needed by our policy is at most $\ln(T+1)$ larger than a theoretical lower bound, regardless of θ. Further, we show that our policy achieves the optimal performance when each server has sufficiently large storage, and is only constrained by its computation capacity.

The allocation ratio guarantees stated above need to hold for every sample path. We also consider the scenario where jobs arrivals are generated by some unknown random process, and one only needs guarantees on the expected value of allocation ratio. By both theoretical analysis and numerical studies, we demonstrate that our policies remain much better than the other existing policies for this scenario.

The rest of the paper is organized as follows: Section II describes our system model for persistent jobs and problem formulation. Section III introduces two online job allocation policies for persistent jobs allocation and studies their performance. Section IV derives a performance bound for all online policies. Section V compares our proposed policies with two widely used policies. Section VI describes our model for dynamic jobs and problem formulation. Section VII introduces an online algorithm for dynamic jobs allocation as well as studies its performance. Section VIII provides simulation comparison of our policies with other commonly used policies. Section IX reviews some previous work on online learning and load balancing problem. Finally, Section X concludes this paper.

II. System Model for Persistent Jobs

We first study the problem of job allocation for persistent jobs. When a persistent job arrives at the system, it needs to be allocated to a server for service immediately. Further, once a job is allocated to a server, it stays in the system for a long time. Examples of persistent jobs include on-demand video streaming for movies, and live stream services.

We consider a system with multiple non-identical servers. Jobs arrive at the system sequentially. Jobs are of different types, and each job can only be served by a subset of the servers. For example, in the application of on-demand video streaming, a job is a request for one video, and can therefore only be served by servers that possess the video. We assume that when a job enters the system, it needs to be allocated to a server immediately. Jobs that cannot be allocated upon arrivals are discarded from the system. We also assume that jobs cannot be moved once they are allocated to servers, as moving jobs between servers cause additional costs on job migration. A similar model has been used in [8].

We use $\mathcal{J}$ to denote the set of servers, and $\mathcal{I}=\{1,2,\dots\}$ to denote the arrival sequence of jobs. Each job i can be served by a subset $K_i\subseteq\mathcal{J}$ of servers. Upon its arrival, job i reveals its K_i, and the system either allocates it to a server or discards it. Each job takes one unit of capacity in the server to which the job is allocated to. A server j has a total amount of C_j units capacity, and can therefore at most serve C_j jobs. We use X_{ij} to denote the assignment of the jobs. If $X_{ij}=1$, then job i is assigned to server j. If $X_{ij}=0$, job i is not assigned to server j.

We aim to maximize the number of jobs that can be served. The problem of maximizing the number of served jobs can be formulated as the following linear programming problem for allocating persistent jobs:

AllocP:

$$Max \sum_{ij:j\in K_i} X_{ij} \tag{1}$$

$$s.t. \sum_{i:j\in K_i} X_{ij}\le C_j, \forall j\in\mathcal{J}, \tag{2}$$

$$\sum_{j:j\in K_i} X_{ij}\le 1, \forall i\in\mathcal{I}, \tag{3}$$

$$X_{ij}\ge 0, \forall i\in\mathcal{I}, j\in\mathcal{J}. \tag{4}$$

Since $X_{ij}=1$ if job i is served by j, (1) is the total number of served jobs. On the other hand, (2) states that each server j can at most serve C_j jobs, and (3) states that each job can be served by at most one server. In this formulation, we allow X_{ij} to be any real number between 0 and 1, while X_{ij} needs to be either 0 or 1 according to our model. Therefore, **AllocP** describes an upper-bound of the number of jobs that can be allocated.

Solving **AllocP** is straightforward when one has knowledge of all its parameters $\{K_i\}$ and $\{C_j\}$. However, as jobs arrive sequentially, the system needs to make allocation decisions without knowledge of future jobs. We say that an allocation policy is an *online policy* if it makes allocation decisions only based on jobs that have already arrived. On the other hand, an allocation policy is an *offline policy* if it has full knowledge about all future job arrivals, and can therefore find the optimal solution to **AllocP**.

We consider that the service provider can increase server capacity by, for example, purchasing more servers as redundancy, to allocate more jobs. Suppose the service provider purchases R times more servers so that each server j has R identical copies. We can instead say that the server j increases its capacity by R times, and

can now server RC_j jobs. We can now formulate the following linear programming problem:

AllocP(R):

$$Max \sum_{ij} X_{ij} \tag{5}$$

$$s.t. \sum_{i:j\in K_i} X_{ij} \le RC_j, \forall j \in \mathcal{J}, \tag{6}$$

$$\sum_{j:j\in K_i} X_{ij} \le 1, \forall i \in \mathcal{I}, \tag{7}$$

$$X_{ij} \ge 0, \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{8}$$

We evaluate the performance of online policies by comparing the number of allocated jobs under online policies with R times capacity against that under offline policy with unit capacity. Specifically, given $\{K_i\}$ and $\{C_j\}$, let Γ_{opt} be the optimal value of $\sum_{ij} X_{ij}$ in **AllocP**, and $\Gamma_\eta(R)$ be the value of $\sum_{ij} X_{ij}$ in **AllocP(R)** under policy η. We define the *competitive ratio per sample path* as follows:

Definition 1: An online policy η is said to be (R,θ)-*competitive-per-sample-path* if $\Gamma_{opt}/\Gamma_\eta(R) \le \theta$, for all $\{K_i\}$ and $\{C_j\}$ with $C_j \ge C_{min}$, as $C_{min} \to \infty$.

Definition 1 defines competitive ratio based on the worst-case sample path. This definition may ignore effects of statistic multiplexing. In practice, jobs may arrive according to some random process. Therefore, the arrivals of different types of jobs are likely to be interwined.

We can expand our model to accommodate the random nature of job arrivals. Given $\{K_i\}$, we can consider the case where the actual arrival sequence is a random permutation of $\mathcal{I} = \{1, 2, \dots\}$. Let $E[\Gamma_\eta(R)]$ be the expected number of allocated jobs under η with R times capacity when the arrival sequence is a random permutation. We then define the *expected competitive ratio* as follows

Definition 2: An online policy η is said to be (R,θ)-*competitive-in-expectation* if $\Gamma_{opt}/E[\Gamma_\eta(R)] \le \theta$, for all $\{K_i\}$ and $\{C_j\}$ with $C_j \ge C_{min}$, as $C_{min} \to \infty$.

It is obvious that the competitive ratio per sample path cannot be better than the expected competitive ratio.

Lemma 1: A (R,θ)-competitive-per-sample-path policy is (R,θ)-competitive-in-expectation.

III. Two Online Allocation Policies for Persistent Jobs and Their Competitive Ratios

In this section, we propose online policies and analyze their competitive ratios. Our analysis is based on the Weak Duality Theorem of linear programming [9]. The dual problem of **AllocP** can be written as:

DualP:

$$Min \sum_j C_j\alpha_j + \sum_i \beta_i, \tag{9}$$

$$s.t.\ \alpha_j + \beta_i \ge 1, \forall i \in \mathcal{I}, j \in K_i \tag{10}$$

$$\alpha_j \ge 0, \forall j, \tag{11}$$

$$\beta_i \ge 0, \forall i, \tag{12}$$

where each α_j corresponds to a constraint in (2), and each β_i corresponds to a constraint in (3). The following lemma is then a direct result of the Weak Duality Theorem.

Lemma 2: Given any vectors of $\{\alpha_j\}$ and $\{\beta_i\}$ that satisfy the constraints (10)–(12), we have $\sum_j C_j\alpha_j + \sum_i \beta_i \ge \Gamma_{opt}$.

We now introduce an online policy. This policy maintains a variable α_j for each server j. When the system starts, it sets $\alpha_j \equiv 0$ initially. When a job i arrives, the policy checks the values of α_j for all $j \in K_i$, and selects j^* as the one with the minimum value of α_j. If $\alpha_{j^*} < 1$, job i is assigned to server j^*, and therefore $X_{ij^*} = 1$. The value of α_{j^*} is updated to be $\alpha_{j^*}(1+\frac{1}{C_{j^*}}) + \frac{1}{(d_{j^*}-1)C_{j^*}}$, where we set $d_j = (1+1/C_j)^{RC_j}$, for all j. The value of d_j is chosen to achieve the optimal competitive ratio, as will be shown in the proof of Lemma 3. On the other hand, if $\alpha_{j^*} \ge 1$, job i is discarded. The complete policy is described in Algorithm 1.

Algorithm 1 PD Algorithm for Persistent Jobs

```
Initially, α_j = 0, β_i = 0, X_ij = 0.
d_j ← (1 + 1/C_j)^{RC_j}, ∀j.
for each arriving job i do
    j* ← argmin_{j∈K_i} α_j.
    if α_{j*} < 1 then
        β_i ← 1 − α_{j*}.
        α_{j*} ← α_{j*}(1 + 1/C_{j*}) + 1/((d_{j*} − 1)C_{j*}).
        X_{ij*} ← 1.
        Job i is assigned to server j*.
    else
        Discard job i.
```

We first need to show that the vector $\{X_{ij}\}$ produced by this policy satisfies all constraints of **AllocP(R)**, so that the policy never assigns a job to a server that is already fully utilized.

Lemma 3: Let $\alpha_j[n]$ be the value of α_j after n jobs have been allocated to server j. Then,

$$\alpha_j[n] = (\frac{1}{d_j - 1})(d_j^{n/RC_j} - 1). \tag{13}$$

Proof: Here we prove (13) by induction.

Initially, when $n = 0$, $\alpha_j[0] = 0 = (\frac{1}{d_j-1})(d_j^0 - 1)$ and (13) holds.

Suppose (13) holds when $n = k$. When the $(k+1)$-th job is allocated server j, we have

$$\begin{aligned}\alpha_j[k+1] =& \alpha_j[k](1+\frac{1}{C_j}) + \frac{1}{(d_j-1)C_j} \\ =& \frac{1}{(d_j-1)}(d_j^{k/RC_j} - 1)(1+\frac{1}{C_j}) + \frac{1}{(d_j-1)C_j} \\ =& \frac{1}{(d_j-1)}[d_j^{(k+1)/RC_j} - 1],\end{aligned}$$

and (13) still holds for $n = k+1$. By induction, (13) holds for all n. ∎

With Lemma 3, $\alpha_j = 1$ when RC_j jobs have been allocated to server j. Since Algorithm 1 only allocates jobs to servers with $\alpha_j < 1$, our policy does not violate any constraints in **AllocP(R)**.

Next, we study the competitive ratio of Algorithm 1.

Theorem 1: Algorithm 1 is $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path.

Proof: We prove Theorem 1 by three steps:

First, we show that solutions $\{\alpha_j\}$ and $\{\beta_i\}$ satisfy all constraints in **DualP**. Initially, α_j and β_i are set to be 0. By step 7 in Algorithm 1, α_j is non-decreasing throughout the execution of the policy, and hence (11) holds. Also, by Lemma 3, $\alpha_j \leq 1$, for all j. When a job i arrives, our policy sets $j^* \leftarrow \underset{j\in K_i}{\operatorname{argmin}}\{\alpha_j\}$. If $\alpha_{j^*} = 1$, we have $\alpha_j = 1$ for all $j \in K_i$ and $\beta_i = 0$. Hence, both constraints (10) and (12) hold. On the other hand, if $\alpha_{j^*} < 1$, $\beta_i = 1-\alpha_{j^*} \geq 1-\alpha_j$, for all $j \in K_i$. Both constraints (10) and (12) still hold.

Next, we derive the ratio between $\sum_{ij} X_{ij}$ and $\sum_j C_j\alpha_j + \sum_i \beta_i$. Both formulas are initially 0. We now consider the amounts of change of these two formulas when a job i arrives. We use $\Delta P(R)$ to denote the change of $\sum_{ij} X_{ij}$, and ΔD to denote the change of $\sum_j C_j\alpha_j + \sum_i \beta_i$.

If job i is discarded, then $\{X_{ij}\}$, $\{\alpha_j\}$ and $\{\beta_i\}$ remain unchanged, and therefore $\Delta P(R) = \Delta D = 0$.

On the other hand, consider the case when job i is assigned to server j. We have $X_{ij} = 1$ and $\Delta P(R) = 1$. We also have

$$\begin{aligned}\frac{\Delta D}{\Delta P(R)} = \Delta D =& C_j(\frac{\alpha_j}{C_j} + \frac{1}{(d_j-1)C_j}) + 1 - \alpha_j \\ =& 1 + \frac{1}{d_j - 1} = \frac{d_j}{d_j - 1} \\ =& \frac{(1+1/C_j)^{RC_j}}{(1+1/C_j)^{RC_j} - 1}.\end{aligned}$$

When we imposes a lower bound on C_j by requiring $C_j \geq C_{min}$, for all j, and let $C_{min} \to \infty$, we have $\frac{\Delta D}{\Delta P(R)} \to \frac{e^R}{e^R-1}$, whenever a job i is allocated to some server. Therefore, we have, under Algorithm 1,

$$\frac{\sum_j C_j\alpha_j + \sum_i \beta_i}{\sum_{ij} X_{ij}} = \frac{e^R}{e^R - 1}. \tag{14}$$

Finally, by Lemma 2, we establish that Algorithm 1 is $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path. ∎

This paper defines competitive ratios only for the limiting case when $C_{min} \to \infty$. As can be seen in the above proof, the condition that $C_{min} \to \infty$ is only needed so that $\frac{\Delta D}{\Delta P(R)} = \frac{(1+1/C_j)^{RC_j}}{(1+1/C_j)^{RC_j}-1} \to \frac{e^R}{e^R-1}$. It is straightforward to extend our definition on competitive ratio per sample path for the general case where $C_{min} < \infty$, and a similar proof yields that the competitive per sample path is $\frac{(1+1/C_{min})^{RC_{min}}}{(1+1/C_{min})^{RC_{min}}-1}$. Fig. 1 plots the value of $\frac{(1+1/C_{min})^{RC_{min}}}{(1+1/C_{min})^{RC_{min}}-1}$. It can be seen that $\frac{(1+1/C_{min})^{RC_{min}}}{(1+1/C_{min})^{RC_{min}}-1}$ converges to $\frac{e^R}{e^R-1}$ rapidly. Hence, our analysis still holds well for practical systems with finite C_{min}.

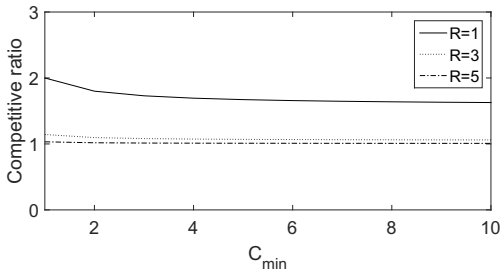


Fig. 1: Convergence of the competitive ratio per sample path with finite C_{min}.

Algorithm 1 relies on the usage of artificial variables $\{\alpha_j\}$ and $\{d_j\}$. Below, we introduce a second online policy that not only is simpler, but also conveys better intuition. The policy is called Join-Least-Utilization (JLU) policy. When a job arrives, JLU simply allocates the job to the server with the smallest utilization ratio, which is the number of allocated jobs at a server divided by its capacity. Specifically, let n_j be the number of jobs that have already been allocated to server j. When job i arrives, it is allocated to $\operatorname{argmin}_{j\in K_i} \frac{n_j}{RC_j}$.

The complete policy is described in Algorithm 2. While the algorithm still involves $\{\alpha_j\}$, $\{d_j\}$, and $\{\beta_i\}$, these variables are introduced solely for the purpose of the analysis of competitive ratio. They can be omitted in actual implementation.

Algorithm 2 JLU

```
Initially, α_j = 0, β_i = 0, X_ij = 0, n_j = 0.
d_j ← (1 + 1/C_j)^{RC_j}, ∀j.
for each arriving job i do
    j* ← argmin_{j∈K_i} {n_j / RC_j}.
    if n_{j*} < RC_{j*} then
        X_{ij*} ← 1.
        α_{j*} ← α_{j*}(1 + 1/C_{j*}) + 1/((d_{j*} − 1)C_{j*}).
        n_{j*} ← n_{j*} + 1.
        Job i is assigned to server j*.
        β_i ← 1 − min_{j∈K_i} α_j.
    else
        Discard job i.
```

Lemma 4: For any $\delta > 0$, there exists a finite C_{min} such that, by requiring $C_j > C_{min}$, for all j, we have

$$\alpha_{j_1} - \alpha_{j_2} > \delta \Longrightarrow \frac{n_{j_1}}{RC_{j_1}} > \frac{n_{j_2}}{RC_{j_2}}, \tag{15}$$

for all $j_1, j_2 \in \mathcal{J}$, throughout the execution of JLU.

Proof: The equation for updating α_j in JLU is the same as that in Algorithm 1. By Lemma 3, at any point

of time, we have

$$\alpha_j = (\frac{1}{d_j - 1})(d_j^{n_j/RC_j} - 1) \quad (16)$$

$$= \frac{1}{(1+1/C_j)^{RC_j} - 1}[(1+1/C_j)^{n_j} - 1], \quad (17)$$

as $d_j = (1+1/C_j)^{RC_j}$.

Note that $\alpha_j \to \frac{e^{n_j/C_j}-1}{e^R-1}$ for a fixed R and all $\frac{n_j}{C_j} \leq R$, as $C_j \to \infty$. Thus, for any $\delta > 0$, there exist a finite C_{min} such that, by requiring $C_j > C_{min}$ for all j, we have $|\alpha_j - \frac{e^{n_j/C_j}-1}{e^R-1}| < \delta/2$ for all j. Therefore, for any two servers j_1 and j_2, we have

$$|(\alpha_{j_1} - \alpha_{j_2}) - (\frac{e^{n_{j_1}/C_{j_1}} - 1}{e^R - 1} - \frac{e^{n_{j_2}/C_{j_2}} - 1}{e^R - 1})| \quad (18)$$

$$< |\alpha_{j_1} - \frac{e^{n_{j_1}/C_{j_1}} - 1}{e^R - 1}| + |\alpha_{j_2} - \frac{e^{n_{j_2}/C_{j_2}} - 1}{e^R - 1}| < \delta. \quad (19)$$

This implies that

$$\alpha_{j_1} - \alpha_{j_2} > \delta \Longrightarrow \frac{e^{n_{j_1}/C_{j_1}} - 1}{e^R - 1} > \frac{e^{n_{j_2}/C_{j_2}} - 1}{e^R - 1} \quad (20)$$

$$\Longrightarrow \frac{n_{j_1}}{RC_{j_1}} > \frac{n_{j_2}}{RC_{j_2}}, \quad (21)$$

and the proof is complete. ∎

Theorem 2: JLU is $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path.

Proof: The proof is very similar to that of Theorem 1.

By Lemma 3, $\frac{n_j}{RC_j} < 1 \Leftrightarrow \alpha_j < 1$. Therefore, under JLU, an arriving job i is allocated if and only if $\min_{j \in K_i} \alpha_j < 1$. For any $\delta > 0$, we pick a sufficiently large C_{min} so that (15) holds.

We can establish that the solutions $\{X_{ij}\}$, $\{\alpha_j\}$, and $\{\beta_i\}$ produced by JLU satisfy all constraints in **AllocP(R)** and **DualP** using an argument that is virtually the same as that in the proof of Theorem 1.

Next, we derive the ratio between $\sum_{ij} X_{ij}$ and $\sum_j C_j\alpha_j + \sum_i \beta_i$. Both formulas are initially 0. We now consider the amounts of change of these two formulas when a job i arrives. We use $\Delta P(R)$ to denote the change of $\sum_{ij} X_{ij}$, and ΔD to denote the change of $\sum_j C_j\alpha_j + \sum_i \beta_i$.

If job i is discarded, then $\Delta P(R) = \Delta D = 0$.

On the other hand, consider the case when job i is assigned to server $j^* = \text{argmin}_{j \in K_i} \frac{n_j}{RC_j}$. By Lemma 4, $\alpha_j \geq \alpha_{j^*} - \delta$, for all $j \in K_i$.

We now have

$$\frac{\Delta D}{\Delta P(R)} = \Delta D$$

$$= C_{j^*}(\frac{\alpha_{j^*}}{C_{j^*}} + \frac{1}{(d_{j^*}-1)C_{j^*}}) + 1 - \min_{j \in K_i} \alpha_j$$

$$\leq C_{j^*}(\frac{\alpha_{j^*}}{C_{j^*}} + \frac{1}{(d_{j^*}-1)C_{j^*}}) + 1 - \alpha_{j^*} + \delta$$

$$= \frac{(1+1/C_{j^*})^{RC_{j^*}}}{(1+1/C_{j^*})^{RC_{j^*}} - 1} + \delta.$$

Let $C_{min} \to \infty$, and we have under JLU,

$$\frac{\sum_j C_j\alpha_j + \sum_i \beta_i}{\sum_{ij} X_{ij}} \leq \frac{e^R}{e^R - 1} + \delta, \quad (22)$$

for any $\delta > 0$

Finally, by Lemma 2, we establish that JLU is $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path. ∎

By Lemma 1, we also have the following theorem.

Theorem 3: Both Algorithm 1 and JLU are $(R, \frac{e^R}{e^R-1})$-competitive-in-expectation.

As a final remark, we note that the complexity of both Alg. 1 and 2 is linear with the number of servers for each job arrival, since each job i needs to find the server that minimizes either α_j or $\frac{n_j}{RC_j}$. Hence, the computation overhead of our algorithms is small.

IV. Lower Bounds of Competitive Ratios

In Section III, we show that our online policies are both $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path. In this section, we will study the lower bound for the competitive ratio per sample path. We focus on a special class of systems described below:

A system in this class has N servers with capacity C each, where C is chosen to be a multiple of $N!$. A total number of NC jobs arrive in sequence, and they are separated into N groups, where the k-th group contains jobs $\{(k-1)C+1, (k-1)C+2, \ldots, kC\}$. Jobs in the same group can be served by the same subset of servers. The subset of servers that can serve a job i, i.e., K_i is constructed as follows: Jobs in the first group $\{1, 2, \ldots, C\}$ can be served by all servers, i.e., $K_i = \mathcal{J}$. For each job in the $(k+1)$-th group, its K_i is obtained by removing one element from that for jobs in the k-th group. More specifically, for a job i_1 in the k-th group and a job i_2 in the $(k+1)$-th group, we have $K_{i_2} \subsetneq K_{i_1}$ and $|K_{i_2}| = |K_{i_1}| - 1$. It is easy to verify that an offline policy can allocate all NC jobs for all systems in this class.

We consider a policy, namely, EVEN, that evenly distributes jobs in the same group among all available servers. We first establish the number of jobs that can be allocated by EVEN, and then show that no policy can guarantee to allocate more jobs than EVEN within this class of systems.

Lemma 5: When the capacity is increased by R times, EVEN serves at most $(N - \frac{N+1}{e^R} + 2)C$ jobs.

Proof: Since EVEN allocates jobs evenly on all available servers, each server gets $\frac{C}{N}$ jobs in the first group of C jobs. Similarly, as jobs in the k-th group can be served by $N-k+1$ servers, each server that can serve this group gets $\frac{C}{N-k+1}$ jobs in this group, unless the server is already fully utilized.

Consider the case when each server has RC capacity. Suppose the system can only serve up to the $(k+1)$-th group, that is, servers are still not fully utilized after

serving the k-th group. We then have

$$C(\frac{1}{N}+\frac{1}{N-1}+\frac{1}{N-2}+\ldots+\frac{1}{N-k+1})<RC$$
$$\Rightarrow \int_{N-k+1}^{N+1}\frac{1}{x}dx<(\frac{1}{N}+\frac{1}{N-1}+\ldots+\frac{1}{N-k+1})<R$$
$$\Rightarrow \log(N+1)-\log(N-k+1)=\log\frac{N+1}{N-k+1}<R$$
$$\Rightarrow k<N+1-\frac{N+1}{e^R}$$

Since servers can serve up to the $(k+1)$-th group and become fully utilized after the arrival of this group, the number of jobs served in the system is then at most $(k+1)C<(N-\frac{N+1}{e^R}+2)C$ ■

Lemma 6: When the parameters R, N, and C are fixed, no online policy can guarantee to allocate more jobs than EVEN.

Proof: We consider an alternative policy ALT and show that it cannot allocate more jobs than EVEN. Given R, N, and C, we construct K_i iteratively as follows: The first group can be served by all servers. Let j_1 be the server with the least jobs. We then choose $K_i=\mathcal{J}\backslash\{j_1\}$ for the second group. Similarly, let j_k be the server with the least jobs among severs that can serve the k-th group. We choose $K_i=\mathcal{J}\backslash\{j_1,j_2\ldots,j_k\}$ for the $(k+1)$-th group. Under this arrival sequence, the total amount of unused capacity in all servers is at least $(RC-\frac{C}{N})+(RC-\frac{C}{N}-\frac{C}{N-1})+\ldots$, which is the total amount of unused capacity by EVEN. Therefore, ALT cannot allocate more jobs than EVEN. ■

Theorem 4: Any online policy for allocating persistent jobs cannot be better than $(R,\frac{e^R}{e^R-1})$-competitive-per-sample-path.

Proof: This is a direct result of Lemmas 5 and 6. ■

Section III has shown that our two online policies are both $(R,\frac{e^R}{e^R-1})$-competitive-per-sample-path. Therefore, Theorem 4 demonstrates that our online policies are indeed optimal.

V. Competitive Ratios of Other Widely Used Policies

In this section, we study the competitive ratios of two widely used polices.

A. Join the Shortest Queue

The first policy is the join the shortest queue (JSQ) policy [10], which allocates jobs to servers with the smallest number of jobs. Specifically, let n_j be the number of jobs that have already been allocated to server j. When a new job i arrives, it is allocated to $\text{argmin}_{j\in K_i}\{n_j|n_j<RC_j\}$, if there exists a server $j\in K_i$ with $n_j<RC_j$.

Theorem 5: JSQ cannot be better than $(R,1+\frac{1}{R})$-competitive-per-sample-path.

Proof: Given R, we construct a system with two types of servers, $J1$ and $J2$, and two types of jobs, $I1$ and $I2$.

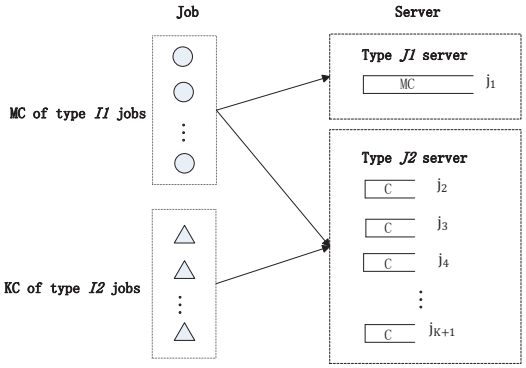


Fig. 2: System illustration for the analysis of JSQ.

Type $I1$ jobs can be served by all servers, while type $I2$ jobs can only be served by type $J2$ severs.

The system is described as Fig 2. It has one type $J1$ server with capacity MC and K type $J2$ servers with capacity C. The job arrival sequence is as follows: first MC jobs of type $I1$ arrive; then KC jobs of type $I2$ arrive. The values of M and K are chosen such that $R=M/(K+1)$. The jobs (or servers) of same type are in the same square box. An arrow line indicates that the job can be allocated to the server.

The optimal offline policy is to allocate all type $I1$ jobs to the server of type $J1$, and allocate all type $I2$ jobs to type $J2$ servers. This allocation can allocate all $MC+KC$ jobs, and $\Gamma_{opt}=MC+KC$.

Now, consider the performance of JSQ when the server capacity is increased by R times. After the increase, the type $J1$ server has RMC capacity, and all other servers have RC capacity. The first MC arrivals are all type $I1$ jobs, who can be served by all servers. Therefore, JSQ evenly distribute these jobs to all servers, and each server gets $MC/(K+1)=RC$ jobs. Next, type $I2$ jobs arrive, and they can only be served by type $J2$ servers. However, at this point, all type $J2$ servers are fully utilized, and no type $I2$ job can be served. The total number of served jobs under JSQ is $\Gamma_{JSQ}(R)=MC$.

We then have

$$\frac{\Gamma_{opt}}{\Gamma_{JSQ}(R)}=\frac{MC+KC}{MC} \tag{23}$$
$$=\frac{M+K}{M}\to 1+\frac{1}{R}, \tag{24}$$

as $K\to\infty$, and $M=R(K+1)$.

■

Theorem 6: JSQ cannot be better than $(R,1+\frac{1}{R^2+2R})$-competitive-in-expectation.

Proof: Given R, we use the same system as that in the proof of Theorem 5, but consider that the actual arrival sequence is a random permutation of all jobs. We still have $\Gamma_{opt}=MC+KC$. In this proof, we choose M and K such that $R=\frac{M}{K}$.

Now, consider the performance of JSQ when the server capacity is increased by R times. Type $J2$ servers can

serve all jobs, while the type $J1$ server can only serve jobs of type $I1$. Under JSQ, whenever a type $J2$ server has the least jobs among all servers, the next job will be allocated to this server, regardless of the type of the job. Therefore, under JSQ, all type $J2$ servers will have at least one less job than the number of jobs at the $J1$ server before all type $J2$ servers are fully utilized. Hence, after $(K+1)RC$ arrivals, all type $J2$ servers are fully utilized. From then on, only type $I1$ jobs can be served. Further, there are MC type $I1$ jobs and KC type $I2$ jobs. Under a random permutation, the average number of type $I1$ jobs that arrive after the first $(K+1)RC$ arrivals is $\frac{MC}{MC+KC}[MC+KC-(K+1)RC] = MC - \frac{M(K+1)RC}{M+K}$. The expected number of jobs served by JSQ is then

$$E[\Gamma_{JSQ}(R)] \leq (K+1)RC + MC - \frac{M(K+1)RC}{M+K} \quad (25)$$

$$= MC + (K+1)RC\frac{K}{M+K}, \quad (26)$$

and hence

$$\begin{aligned}\frac{\Gamma_{opt}}{E[\Gamma_{JSQ}(R)]} &\geq \frac{MC+KC}{MC+(K+1)RC\frac{K}{M+K}} \\ &= \frac{RK+K}{RK+(K+1)RK/(RK+K)} \\ &\to \frac{(R+1)^2}{R(R+1)+R} \\ &= 1+\frac{1}{R^2+2R},\end{aligned}$$

as $K \to \infty$ and $M = KR$. ■

B. Join the Most Residue Queue

It may seem that JSQ performs poorly only because it makes decisions solely based on n_j, and does not consider C_j. A straightforward extension of JSQ is the join the most residue queue (JMQ) algorithm, which allocates jobs to servers with most remaining space, which is the server capacity minus the number of allocated jobs in this server. Let RC_j be the capacity of server j, n_j be the number of jobs that have already been allocated to j. The arriving job i is allocated to server $\text{argmax}_{j\in K_i}\{RC_j - n_j | n_j < RC_j\}$, if there exists a server $j \in K_i$ with $n_j < RC_j$.

Theorem 7: Join the most residue queue policy cannot be better than $(R, 1+\frac{1}{R})$-competitive-per-sample-path.

Proof: Given R, we construct a system with two types of servers, $J1$ and $J2$, and two types of jobs, $I1$ and $I2$. Type $I1$ jobs can only be served by type $J1$ servers, while type $I2$ jobs can be served by all servers.

The system is shown in Fig. 3. It has one type $J1$ server with capacity $(M+1)C$, and K type $J2$ servers with capacity C. The job arrival sequence is as follows: first KC jobs of type $I2$ arrive; then $(M+1)C$ jobs of type $I1$ arrive. The value of M and K are chose such that $R = K/M$.

The optimal offline policy is to allocate all type $I2$ jobs to servers of type $J2$ and all type $I1$ jobs to the type $J1$ server. The total number of jobs allocated by this policy is $(M+1)C + KC$, and $\Gamma_{opt} = (M+1)C + KC$.

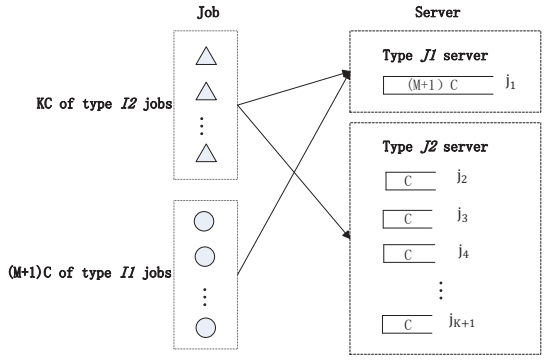


Fig. 3: System illustration for the analysis of JMQ.

When the server capacity is increased by R times, type $J1$ server has $R(M+1)C$ capacity, and type $J2$ servers have RC capacity. The first arriving $KC = MRC$ jobs are of type $I2$. They can be served by both type $J1$ and $J2$ servers. JMQ allocates all these jobs to type $J1$ server. Type $J1$ server can therefore serve only RC jobs of type $I1$. The total number of jobs served is $R(M+1)C$.

We then have

$$\frac{\Gamma_{opt}}{\Gamma_{JMQ}(R)} = \frac{(M+1)C+KC}{R(M+1)C} \quad (27)$$

$$= \frac{M}{M+1} + \frac{1}{R} \to 1 + \frac{1}{R}, \quad (28)$$

as $M \to \infty$, and $K = MR$. ■

Theorem 8: JMQ cannot be better than $(R, 1+\frac{1}{R^2+2R})$-competitive-in-expectation.

Proof: We use the system in the proof of Theorem 7 but consider that the actual arrival sequence is a random permutation of all jobs. First, we have $\Gamma_{opt} = (M+1)C + KC$. Now we study $E[\Gamma_{JMQ}(R)]$. In this proof, we choose M and K such that $K = MR - 1$.

The type $J1$ server can serve all jobs and has MRC more capacity than others. Thus, the first MRC jobs will be allocated to the type $J1$ server, regardless of job types. After the first MRC arrivals, the type $J1$ server has RC capacity left and hence at most RC more type $I1$ jobs can be served. Since there are $(M+1)C$ type $I1$ jobs and KC type $I2$ jobs, the average number of type $I1$ jobs among the first MRC arrivals is $MRC \cdot \frac{M+1}{M+1+K}$. The expected number of allocated jobs is then no more than $RC + MRC\frac{M+1}{M+1+K} + KC$. Therefore we have:

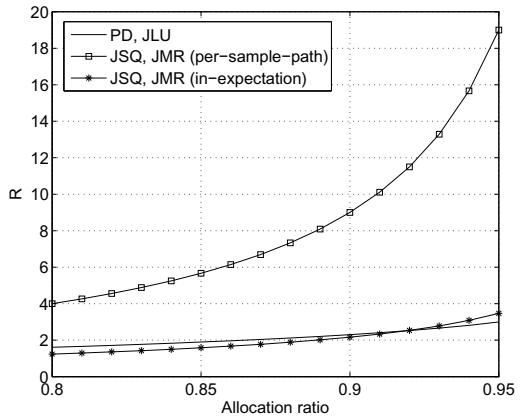


Fig. 4: Capacity requirements of different policies.

$$\frac{\Gamma_{opt}}{E[\Gamma_{JMQ}(R)]} \tag{29}$$

$$\geq \frac{MC+C+KC}{RC+MRC\frac{M+1}{M+1+K}+KC} \tag{30}$$

$$= \frac{M+MR}{MR-1+R+MR\frac{M+1}{M+MR}} \tag{31}$$

$$= \frac{M^2(R+1)^2}{(M+1)RM(R+1)-M(R+1)+MR(M+1)} \tag{32}$$

$$\to \frac{(R+1)^2}{R(R+1)+R} \tag{33}$$

$$= 1+\frac{1}{R^2+2R} \tag{34}$$

as $M \to \infty$ and $K = MR-1$. ■

C. Discussions

We have shown that our policies are $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path, while JSQ and JMR are no better than $(R, 1+\frac{1}{R})$-competitive-per-sample-path, and no better than $(R, 1+\frac{1}{R^2+2R})$-competitive-in-expectation. Suppose we are given a system where the offline policy can allocate all jobs. In order to guarantee that at least $1-\frac{1}{\theta}$ of the jobs are allocated, our policy only needs to increase the capacity by R times so that $\frac{e^R}{e^R-1} \leq 1/(1-\frac{1}{\theta})$. Therefore, choosing $R = \ln\theta$ is sufficient. In contrast, the two commonly used policies, JSQ and JMQ, require to increase the server capacity by at least $(\theta-1)$ times. Even when we consider that the arrival sequence is a random permutation of all jobs, and only require JSQ and JMQ to allocate $1-\frac{1}{\theta}$ of the jobs *on average*, they still need to increase the capacity by at least $\sqrt{\theta}-1$ times.

Fig. 4 plots the capacity requirement for different allocation ratios. From the figure we can observe that as the allocation ratio approaches 1, the capacity requirement using JSQ or JMQ increases much faster than that using PD or JLU. For example, if we need to allocate at least 95% of the jobs, i.e., $\theta = 20$, our policies only require $R = 3$, while JSQ and JMQ both require $R \geq 19$. Even when the arrival sequence is a random permutation of all jobs, JSQ and JMQ still need $R \geq 3.5$. Further, one may notice that the expected competitive ratios of JSQ and JMQ seem to outperform the competitive ratio per sample path of our policies when the allocation ratio is low. However, the comparison is not fair, since, by Lemma 1, the expected competitive ratio is always better than the competitive ratio per sample path for any policy. In Section VIII, we will demonstrate that our policies still outperform JSQ and JMQ when the actual arrival sequence is a random permutation of all jobs.

VI. System Model for Dynamic Jobs

We now turn our attention to the problem of allocating dynamic jobs. Compared to persistent jobs, dynamic jobs have two important features: First, dynamic jobs require only a small amount of execution time, and they leave the system once they are completed. Second, dynamic jobs may not require immediate service. Rather, they only require to be completed within their specified delay bounds. Jobs that are not served immediately can be stored in the buffer of a server, as long as there is enough space in the buffer. Examples of dynamic jobs include web searches and database queries.

We now formally describe our system model for dynamic jobs. We consider a system with multiple non-identical servers with different service capacities and buffer sizes. Time is slotted and indexed by $t = 0, 1, 2 \ldots$. At the beginning of each time slot, some jobs arrive at the system sequentially, and each job can only be served by a subset of the servers. Each job specifies the subset of servers that can serve it, and a hard delay bound, when it arrives at the system. Upon the arrival of a job, the system needs to immediately allocate it to a server, either to be executed immediately or to be stored in the buffer. Jobs that cannot be allocated immediately, or cannot be completed within their delay bounds, are dropped from the system.

We assume that all jobs require the same amount of computation resource. A server j has a service capacity of executing C_j jobs per time slot, and a buffer that can store B_j jobs that are yet to be completed. We use $a(i)$ to denote the arrival time of job i. When job i arrives, it reveals its server subset K_i and delay bound $T(i)$. We assume that the delay bounds are upper-bounded by $T(i) \leq T, \forall i$. We define k_{ijt} as the indicator function that $K_i = 1$ and $a(i) \leq t \leq a(i)+T(i)$. In other words, we have $k_{ijt} = 1$ if job i can be completed by server j at time t without violating its delay bound.

The decision of allocating a dynamic job consists of two parts: deciding *which* server to serve this job, as well as *when* to serve it. We use X_{ijt} to denote the allocation decision for job i. If $X_{ijt} = 1$, then job i is served by server j at time t. When a job is allocated to a server, it waits in the buffer until it is completed. Therefore, at

each time slot τ, the jobs in the buffer of server j are those that satisfy the following three conditions: (i) they arrive on or before time τ, (ii) they are allocated to server j, and (iii) they are scheduled to be executed on or after time τ. The number of jobs in the buffer of server j at time τ can then be written as $\sum_{i,t:a(i)\leq\tau\leq t} k_{ijt}X_{ijt}$.

We formulate the problem of maximizing the number of completed dynamic jobs as the following linear programming problem:

AllocD:

$$Max \sum_{ijt} k_{ijt}X_{ijt} \tag{35}$$

$$s.t. \sum_{i} k_{ijt}X_{ijt} \leq C_j, \forall t, j \in \mathcal{J}, \tag{36}$$

$$\sum_{i,t:\tau\in[a(i),t]} k_{ijt}X_{ijt} \leq B_j, \forall \tau, j \in \mathcal{J}, \tag{37}$$

$$\sum_{jt} X_{ijt} \leq 1, \forall i \in \mathcal{I}, \tag{38}$$

$$X_{ijt} \geq 0, \forall i \in \mathcal{I}, j \in \mathcal{J}, t. \tag{39}$$

In the above problem, (35) is the total number of completed jobs. (36) states that each server j can serve at most C_j jobs in each time slot. (37) states that at any time, the total number of jobs in the buffer of server j is at most B_j. Finally, (38) states that each job can be served at most once.

Similar to the previous case, we consider that the service provider can increase both capacities to allocate more jobs. When the capacities are increased by R times, the service capacity becomes RC_j and the buffer capacity becomes RB_j. We have our linear programming as follows:

AllocD(R):

$$Max \sum_{ijt} k_{ijt}X_{ijt} \tag{40}$$

$$s.t. \sum_{i} k_{ijt}X_{ijt} \leq RC_j, \forall t, j \in \mathcal{J}, \tag{41}$$

$$\sum_{\substack{i,\\ t:\tau\in[a(i),t]}} k_{ijt}X_{ijt} \leq RB_j, \forall \tau, j \in \mathcal{J}, \tag{42}$$

$$\sum_{jt} X_{ijt} \leq 1, \forall i \in \mathcal{I}, \tag{43}$$

$$X_{ijt} \geq 0, \forall i \in \mathcal{I}, j \in \mathcal{J}, t. \tag{44}$$

The definition of the competitive ratio per sample path for persistent jobs in Section II can be naturally applied to systems with dynamic jobs.

Before we proceed to the next section, we note that, by setting $B_j = \infty$ for all j, and making all jobs arrive at the beginning of the first time slot with $T(i) = 1$ for all i, the problems of **AllocD** and **AllocD(R)** become equivalent to **AllocP** and **AllocP(R)**, respectively. In other words, the system with persistent jobs can be thought of as a special case of a system with dynamic jobs. Hence, the competitive ratio per sample path for dynamic jobs cannot be better than that for persistent jobs, and we immediately have the following lower bound:

Theorem 9: Any online policy for allocating dynamic jobs cannot be better than $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path.

Proof: This is a direct result of Theorem 4. ■

VII. An Online Policy for Dynamic Jobs

A. Policy Design and Performance Analysis

This section proposes an online policy for allocating dynamic jobs, and analyzes its competitive ratio. Many proofs in this section are similar to those in Section III, and hence we move all proofs to the appendix.

Similar to Section III, we first find the dual problem of **AllocD** as:

DualD:

$$Min \sum_{jt} C_j\alpha_{jt} + + \sum_{j\tau} B_j\gamma_{j\tau} + \sum_{i} \beta_i, \tag{45}$$

$$s.t.\ k_{ijt}\alpha_{jt} + \sum_{\tau\in[a(i),t]} k_{ijt}\gamma_{j\tau} + \beta_i \geq k_{ijt}, \forall i \in \mathcal{I}, j \in \mathcal{J}, t \tag{46}$$

$$\alpha_{jt} \geq 0, \forall j \in \mathcal{J}, t, \tag{47}$$

$$\gamma_{j\tau} \geq 0, \forall j \in \mathcal{J}, \tau, \tag{48}$$

$$\beta_i \geq 0, \forall i \in \mathcal{I}. \tag{49}$$

In **DualD**, each α_{jt} corresponds to a constraint in (41), each $\gamma_{j\tau}$ corresponds to a constraint in (42), and each β_i corresponds to a constraint in (43).

Now we introduce our online scheduling policy. The policy maintains two sets of variables α_{jt} and $\gamma_{j\tau}$. They can be seen as the monitors for the usage of service capacity and buffer capacity, respectively. Initially, they are both set to be 0.

When a job i arrives, the policy finds a server j^* and a time t^* that maximizes $k_{ijt}(1-\alpha_{jt}-\sum_{\tau\in[a(i),t]}\gamma_{j\tau})$ for all j and t. If $k_{ij^*t^*}(1-\alpha_{j^*t^*}-\sum_{\tau\in[a(i),t^*]}\gamma_{j\tau^*}) > 0$, then job i is scheduled to be executed by server j^* at time t^*, and hence $X_{ij^*t^*} = 1$. The job is dropped if $k_{ij^*t^*}(1-\alpha_{j^*t^*}-\sum_{\tau\in[a(i),t^*]}\gamma_{j\tau^*}) \leq 0$.

When the job i is scheduled to be executed by server j^* at time t^*, it consumes two kinds of resource: First, it requires one unit of service capacity of server j^* at time t^*. Second, it is stored in the buffer of server j^* from its arrival time, $a(i)$, to time t. The corresponding dual variables for these two kinds of resources are $\alpha_{j^*t^*}$ and $\gamma_{j^*\tau}$, for $\tau \in [a(i), t^*]$, and they are updated by

$$\alpha_{j^*t^*} \leftarrow \alpha_{j^*t^*}(1+\frac{1}{C_{j^*}}) + \frac{1}{(d_{j^*}-1)C_{j^*}},$$

$$\gamma_{j^*\tau} \leftarrow \gamma_{j^*\tau}(1+\frac{1}{B_{j^*}}) + \frac{1}{(f_{j^*}-1)B_{j^*}},$$

where $d_j = (1+1/C_j)^{RC_j}$ and $f_j = (1+1/B_j)^{RB_j}$, for all j. The complete policy is described in Algorithm 3. It is straightforward to check that the complexity of the policy is $O(\mathcal{J}T)$.

Algorithm 3 PD Algorithm for Dynamic Jobs

Initially, $\alpha_{jt} = 0$, $\gamma_{j\tau} = 0$, $X_{ijt} = 0$.
$d_j \leftarrow (1 + 1/C_j)^{RC_j}, \forall j$.
$f_j \leftarrow (1 + 1/B_j)^{RB_j}, \forall j$.
for each arriving job i **do**
  $(j^*, t^*) \leftarrow \text{argmax}_{(j,t)} k_{ijt}(1 - \alpha_{jt} - \sum_{\tau \in [a(i),t]} \gamma_{j\tau})$.
  if $k_{ij^*t^*}(1 - \alpha_{j^*t^*} - \sum_{\tau \in [a(i),t^*]} \gamma_{j\tau^*}) > 0$ **then**
    $\beta_i \leftarrow 1 - k_{ij^*t^*}(1 - \alpha_{j^*t^*} - \sum_{\tau \in [a(i),t^*]} \gamma_{j\tau^*})$
    $\alpha_{j^*t^*} \leftarrow \alpha_{j^*t^*}(1 + \frac{1}{C_{j^*}}) + \frac{1}{(d_{j^*} - 1)C_{j^*}}$.
    $\gamma_{j\tau^*} \leftarrow \gamma_{j\tau^*}(1 + \frac{1}{B_{j^*}}) + \frac{1}{(f_{j^*} - 1)B_{j^*}}$, $\forall \tau \in [a(i), t^*]$
    $X_{ij^*t^*} \leftarrow 1$.
    Job i is assigned to server j^*, to be served at t^*.
  else
    Discard job i.

We first need to show that the vector $\{X_{ijt}\}$ produced by this policy satisfies all constraints of **AllocD(R)**.

Lemma 7: Let $\alpha_{jt}[n]$ be the value of α_{jt} after n jobs are scheduled to be served by j at t. Let $\gamma_{j\tau}[n]$ be the value of $\gamma_{j\tau}$ when n jobs are waiting in server j at τ and to be served at a later time t. Then,

$$\alpha_{jt}[n] = (\frac{1}{d_j - 1})(d_j^{n/RC_j} - 1). \quad (50)$$

$$\gamma_{j\tau}[n] = (\frac{1}{f_j - 1})(f_j^{n/RB_j} - 1). \quad (51)$$

Proof: See Appendix A. ∎

With Lemma 7, $\alpha_{jt} = 1$ when RC_j jobs have been scheduled to be served by j at t, and $\gamma_{j\tau} = 1$ when RB_j jobs are waiting in the buffer of server j at τ. In Algorithm 3, jobs are only allocates to servers with $k_{ijt}(1-\alpha_{jt}-\sum_{\tau \in [a(i),t]} \gamma_{j\tau}) > 0$, which guarantees $\alpha_{jt} \leq 1$ and $\gamma_{j\tau} \leq 1$. Thus our policy does not violate any constraints in **AllocD(R)**.

Next, we study the competitive ratio of Algorithm 3.

Theorem 10: Algorithm 3 is $(R, \frac{e^R+T}{e^R-1})$-competitive-per-sample-path.

Proof: See Appendix B. ∎

B. Discussions and Performance Analysis for Infinite Buffers

We now discuss the practical implications of Theorem 10. Suppose we are given a system where the offline policy can allocate all jobs. In order to guarantee that at least $1 - \frac{1}{\theta}$ of the jobs are allocated, Algorithm 3 needs to increase the capacity by R times so that $\frac{e^R+T}{e^R-1} \leq 1/(1-\frac{1}{\theta})$. We then have $R = \ln(\theta T + \theta - T) < \ln\theta + \ln(T + 1)$. Meanwhile, Theorem 9 states that no online policy can be better than $(R, \frac{e^R}{e^R-1})$-competitive. Therefore, any online policy would require at least $R \geq \ln\theta$ to ensure that at least $1 - \frac{1}{\theta}$ of the jobs are allocated. This result shows the amount of redundancy needed by Algorithm 3 is at most $\ln(T + 1)$ larger than the theoretical lower-bound, regardless of the service guarantee $1 - \frac{1}{\theta}$.

In the analysis of Theorem 10, we need to consider constraints on both service capacity and buffer capacity. In many applications, the major performance bottleneck is service capacity. For such applications, we can assume that the buffer size B_j is infinite for all j. As a result, constraint (42) in **AllocD(R)** and variables γ_{jt} in **DualD** no longer exist. We can then modify the design of Algorithm 3 by simply removing all variables γ_{jt}. The complete algorithm is shown in Algorithm 4.

Algorithm 4 PD Algorithm for Infinite Buffers

Initially, $\alpha_{jt} = 0$, $X_{ijt} = 0$.
$d_j \leftarrow (1 + 1/C_j)^{RC_j}, \forall j$.
for each arriving job i **do**
  $(j^*, t^*) \leftarrow \text{argmax}_{(j,t)} k_{ijt}(1 - \alpha_{jt})$.
  if $k_{ij^*t^*}(1 - \alpha_{j^*t^*}) > 0$ **then**
    $\alpha_{j^*t^*} \leftarrow \alpha_{j^*t^*}(1 + \frac{1}{C_{j^*}}) + \frac{1}{(d_{j^*} - 1)C_{j^*}}$.
    Job i is assigned to server j^*, and to be served at time t^*.
  else
    Discard job i.

It is straightforward to show that Algorithm 4 achieves the optimal competitive ratio per sample path.

Theorem 11: Algorithm 4 is $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path.

VIII. Simulation

In this section, we evaluate the performance of both persistent job model and dynamic job model.

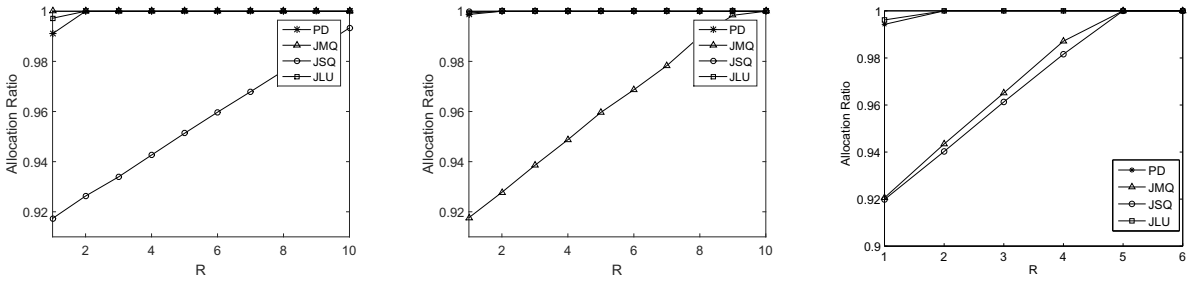
A. Simulation of Persistent Job Model

We evaluate the performance of the four policies, including PD, JLU, JSQ, and JMQ, discussed in this paper by simulations. We consider three different scenarios:

In the first scenario, we construct a system with two types of servers $J1,\ J2$; and two types of jobs $I1,\ I2$. Type $I1$ job can be served by type $J2$ server, type $I2$ job can be served by both $J1$ and $J2$ server. The number of jobs and capacities of servers are shown in Table I. In the second scenario, we construct a system with two types of servers: $J1,\ J2$; and two types of jobs: $I1,\ I2$. The setting for jobs and servers are shown in Table II. Last, we construct a system with four types of servers: $J1$, $J2$, $J3$, and $J4$; and four types of jobs: $I1,\ I2,\ I3,\ I4$. The detailed setting for servers and jobs are listed in Table III, which is a combination of the first two scenarios.

In each scenario, the arrival sequence is a random permutation of all jobs. Simulation results are the average of 10 runs. Under different R, we computer the allocation ratio by the number of jobs allocated with online policy dividing that with optimal offline policy.

Simulation results are shown in Fig. 5. We can see that the allocation ratios of all four policies converge to 1 as R increases. However, JSQ and JMQ converge much slower



(a) First scenario (b) Second scenario (c) Third scenario

Fig. 5: Simulation results for the persistent job model.

TABLE I: System setting for the first scenario.

Server Type	Number of Servers	Capacity	Job Type	Number of Jobs	K_i
$J1$	1	25000	$I1$	2500	$J2$
$J2$	50	50	$I2$	25000	$J1,\ J2$

TABLE II: System setting for the second scenario.

Server Type	Number of Servers	Capacity	Job Type	Number of Jobs	K_i
$J1$	1	2550	$I1$	2550	$J1$
$J2$	500	50	$I2$	25000	$J1,\ J2$

than our proposed PD and JLU. Moreover, we notice that our proposed PD and JLU policies almost have identical performance.

In Section III we prove that PD and JLU are $(R, \frac{e^R}{e^R-1})$-competitive-in-expectation. In Secion V we prove that both JSQ and JMQ are no better than $(R, 1+\frac{1}{R^2+2R})$-competitive-in-expectation. Although it looks like that the competitive ratio in expectation of our policies is not as good as that of the other two policies when R is small, the simulation results show that our policies have better allocation ratio even with small R. Thus with random permutation of the jobs arrival sequence, the performance of our policies are better than of JSQ and JMR policies.

B. Simulation of Dynamic Job Model

For the case of dynamic jobs, we will use Revised JSQ (RJSQ), Revised JMQ (RJMQ), and Quincy as comparisons to our proposed policy.

The RJSQ (or RJMQ) policy always allocates jobs to the buffer with the smallest number of jobs (or most remaining space). Then the system follows earliest deadline first policy to serve jobs in the buffer. Quincy is the policy proposed in [11]. In Quincy, when a job arrives, it joins all servers that can serve it. The servers coordinate among themselves so that when a server begins processing a job, all other servers discard the job immediately. Obviously, Quincy would result in high coordination overhead among servers. Moreover, as we will demonstrate below, our policy still outperforms Quincy even when we ignore the coordination overhead.

TABLE III: System setting for the third scenario.

Server Type	Number of Servers	Capacity	Job Type	Number of Jobs	K_i
$J1$	1	10000	$I1$	2500	$J2$
$J2$	50	50	$I2$	10000	$J1$,$J2$
$J3$	1	2550	$I3$	2550	$J3$
$J4$	200	50	$I4$	10000	$J3$,$J4$

TABLE IV: Server setting for dynamic job model.

Server Type	Number of Servers	Service Rate per Time Slot	Buffer
$J1$	1	100	300
$J2$	5	5	15
$J3$	1	25	75
$J4$	20	5	15

TABLE V: Jobs setting for dynamic job model.

Job Type	Expected Number of Jobs per Time Slot	K_i
$I1$	25	$J2$
$I2$	100	$J1$,$J2$
$I3$	25	$J3$
$I4$	100	$J3$,$J4$

We consider the system constructed as below: there are four types of servers: $J1$, $J2$, $J3$, and $J4$; and four types of jobs $I1$, $I2$, $I3$, and $I4$. The setting of servers is shown in Table IV. In each time slot, job arrival follows the Poisson random process with expected number of each type of job shown in Table V. The arrival sequence is a random permutation of all these jobs. Each job has a end-to-end deadline uniformly selected from 1 to 5. The system is run for 60 time slots and there are jobs arriving the system for the first 50 slots. Simulation results are shown in Fig. 6. We can observe that our policy outperforms all other policies.

IX. Related Work

The online job allocation problem is an online matching procedure which aim to make the best decision on job-server pair to maximize the number of jobs get matched. Many works have been done on persistent job allocation. The problem of online bipartite matching was studied by Karp, Vazirani, and Vazirani [7]. They use an adversary model and studied GREEDY which achieves a matching ratio of $1/2$ and RANKING which achieves $1-1/e$. They further showed that no algorithm can achieve a better

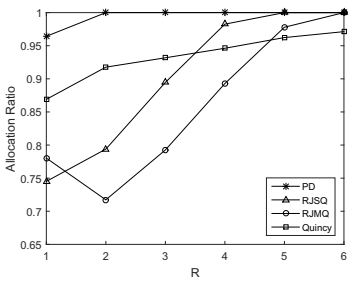


Fig. 6: Simulation results for the dynamic model.

ratio than $1 - 1/e$. Other models, which are based on further assumption on arrival pattern, have also been studied. Random arrival model has been studied by Goel and Mehta [12], and Karande, Mehta and Tripathi [13]. They show that GREEDY achieves a matching ratio of $1-1/e$ and RANKING achieves greater than $1-1/e$. Known distribution model was introduced by Feldman, Mehta, Mirrokni, and Muthukrishnan [14]. They provide a two-suggested-matching algorithm which achieves a ratio of 67%. Kalyanasundaram and Pruhs studied the online b-matching problem [15] which can be seen as the job allocation problem with server capacity b. They presented BALANCE algorithm and proved that it approaches $1-1/e$. Applications of online matching to ad-words problem, which is an allocation of bidders to key words within the budget limit of each bidder, have been studied in [12], [16]. However, none of these studies can precisely quantify the amount of capacity needed to guarantee a certain allocation ratio, which is the goal of this paper.

There are many studies on YouTube videos about their statistical properties [6], [17], [18]. Studies on online learning further investigate the possibility to predict the future video requests. The problem of "learn from expert advise" was first studied by Littlestone and Warmuth [19], DeSantis, Markowsky, and Wegman [20]. Later "learn from examples" was studied. The Winnow algorithm was proposed and studied by Littlestone and Nicholas [21], [22]. The algorithm applies well to practical tasks such as on World Wide Web [23]. Other sequence prediciton research also include the studies by Nicolo and Gabor [24], Hutter [25]. These studies assume that the job arrivals follow some well-defined random process, and, conceptually speaking, they aim to find policies that learn the parameters of the random process on-the-fly.

Also, many studies have consider the online scheduling with time constraints. Moharir, Sanghavi, and Shakkottai [26] have extended the RANKING algorithm for scheduling jobs with time constraints and shown that their policy achieves a matching ratio of $1-1/e$. Koo, et al. [27] have studied the case with uni-processor and the jobs to be scheduled have tight deadlines and preemption is allowed at no cost. The approach is to improve the processor's speed and the result shows that a processor $O(1)$ times faster is sufficient to guarantee a competitive ratio of 1 if jobs have general value densities. Dürr, Jeż, and Nguyen [28] have studied a problem with preemption where jobs take different processing time and have different deadlines. They aim to maximize the total weight of jobs completed and show a competitive ratio of $O(k/\log k)$. Liu et al. [29] have studied the preemptive scheduling for Hadoop jobs with deadline and implemented the first real preemptive job scheduler to meet deadlines on Hadoop. Khalib, Ahmad, and Ong [30] have studied a non-preemptive scheduling of jobs with soft real time system which can tolerant some percentage of missing deadline. They propose algorithms to group jobs with near deadline together and then schedule jobs within a group to improve the earliest deadline first (EDF) policy.

In addition to theoretical research, there have also been many efforts on building systems for job allocation that perform well in practical scenarios. Babaioff, et al. [31] have proposed a framework that allow the implementations of both allocation algorithms and pricing policies. Hindman, et al. [32] have proposed a platform that enables resource sharing. Zaharia, et al. [33] have proposed the concept of "delay scheduling" and shown that it performs well when each jobs consists of a large number of smaller tasks. While these studies demonstrate good performance in practical settings, they lack the theoretical guarantees on "worst-case" performance.

X. Conclusion

In this paper, we study the job allocation problem with unknown job arriving pattern under hard allocation ratio requirement. Given the capacity of current data center which serves all jobs offline, we aim to find how much capacity we need to expand to meet the allocation ratio requirement for any unknown job arrival sequence.

We consider system models for both persistent jobs and dynamic jobs. For the case of persistent jobs, we propose two online policies PD and JLU which are both $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path. We also prove that our policies can achieve any allocation ratio requirement with the least capacity. Next we study the performance of two widely used policies, JSQ and JMQ. We prove that both policies are no better than $(R, 1+\frac{1}{R})$-competitive-per-sample-path. Therefore, they need an order higher capacity to achieve the same allocation ratio requirement than our policies. We further prove that JSQ and JMQ are no better than $(R, 1+\frac{1}{R^2+2R})$-competitive-in-expectation by taking random permutation of the jobs in the arrival sequence. For the case of dynamic jobs, we propose an online policy PD which is $(R, \frac{e^R+T}{e^R-1})$-competitive-per-sample-path. When buffers are not the bottleneck, the algorithm is $(R, \frac{e^R}{e^R-1})$-competitive-per-sample-path, which achieves the optimal competitive ratio per sample path. We also demonstrate that our policies outperform other state-of-the-art policies by simulations.

There are several important limitations in our current model. For example, in the persistent model, we assume that on-demand video streaming jobs never leave the system. In practice, streaming jobs may terminate at arbitrary

times based on user behaviors. In the dynamic model, we assume that all jobs require the same computation time, while in practice different jobs obviously require different times. Further, we only focus on the allocation ratio of jobs, and ignore that different jobs may have different values. Relaxing these assumptions to better capture the behavior of real systems can be important future work.

APPENDIX

A. Proof of Lemma 7

Proof: Here we prove (50) and (51) by induction.

Initially, when $n = 0$, $\alpha_{jt}[0] = 0 = (\frac{1}{d_j - 1})(d_j^0 - 1)$ and (50) holds.

Suppose (50) holds when $n = k$. When the $(k+1)$-th job is scheduled to be served by j at t, we have

$$\begin{aligned}\alpha_{jt}[k+1] =& \alpha_{jt}[k](1 + \frac{1}{C_j}) + \frac{1}{(d_j - 1)C_j} \\ =& \frac{1}{(d_j - 1)}(d_j^{k/RC_j} - 1)(1 + \frac{1}{C_j}) + \frac{1}{(d_j - 1)C_j} \\ =& \frac{1}{(d_j - 1)}[d_j^{(k+1)/RC_j} - 1],\end{aligned}$$

and (50) still holds for $n = k+1$. By induction, (50) holds for all n.

Similarly, when $n = 0$, $\gamma_{j\tau}[0] = 0 = (\frac{1}{f_j - 1})(f_j^0 - 1)$ and (51) holds.

Suppose (51) holds when $n = k$, i.e., there are k jobs waiting for service in server j. When the $(k+1)$-th job is scheduled to be served by j at time t, where $t > \tau$, we have

$$\begin{aligned}\gamma_{j\tau}[k+1] =& \gamma_{j\tau}[k](1 + \frac{1}{B_j}) + \frac{1}{(f_j - 1)B_j} \\ =& \frac{1}{(f_j - 1)}(f_j^{k/RB_j} - 1)(1 + \frac{1}{B_j}) + \frac{1}{(f_j - 1)B_j} \\ =& \frac{1}{(f_j - 1)}[f_j^{(k+1)/RB_j} - 1],\end{aligned}$$

and (51) still holds for $n = k+1$. By induction, (51) holds for all n. ■

B. Proof of Theorem 10

Proof: We prove Theorem 10 by three steps:

First, we show that solutions $\{\alpha_{jt}\}$, $\{\gamma_{j\tau}\}$ and $\{\beta_i\}$ satisfy all constraints in **DualD**. Initially, α_{jt} and $\gamma_{j\tau}$ are set to be 0. By step 8 in Algorithm 3, α_{jt} and $\gamma_{j\tau}$ are non-decreasing. Hence (47) and (48)holds. Also, by Lemma 7, $\alpha_{jt} \leq 1$, $\gamma_{j\tau} \leq 1$, for all j. When a job i arrives, our policy selects $(j^*, t^*) \leftarrow \text{argmax}_{(j,t)}\, k_{ijt}(1 - \alpha_{jt} - \sum_{\tau \in [a(i),t]} \gamma_{j\tau})$ according to step 5. When $(\alpha_{j^*t^*} + \sum_{\tau \in [a(i),t^*]} \gamma_{j\tau^*}) < 1$, we have $k_{ijt}(\alpha_{jt} + \sum_{\tau \in [a(i),t]} \gamma_{j\tau}) \geq k_{ij^*t^*}(\alpha_{j^*t^*} + \sum_{\tau \in [a(i),t^*]} \gamma_{j\tau^*}) = 1 - \beta_i$, then (46) and (49) hold. When $(\alpha_{j^*t^*} + \sum_{\tau \in [a(i),t^*]} \gamma_{j\tau^*}) = 1$, we have $(\alpha_{jt} + \sum_{\tau \in [a(i),t]} \gamma_{j\tau}) \geq 1$ and $\beta_i = 0$, (46) and (49) hold.

Next, we derive the ratio between (45) and (40). Both formulas are initially 0. When a job i arrives, we use $\Delta P(R)$ to denote the change of $\sum_{ijt} k_{ijt} X_{ijt}$, and ΔD to denote the change of $\sum_{jt} C_j \alpha_{jt} + + \sum_{j\tau} B_j \gamma_{j\tau} + \sum_i \beta_i$. If job i is discarded, both formulas remain unchanged, therefore $\Delta P(R) = \Delta D = 0$. If job i is scheduled to be serverd by j at t, then we have $X_{ijt} = 1$ and $\Delta P(R) = 1$. Also we have

$$\begin{aligned}&\frac{\Delta D}{\Delta P(R)} = \Delta D \\ =& C_j(\frac{\alpha_{jt}}{C_j} + \frac{1}{(d_j - 1)C_j}) + \sum_{\tau \in [a(i),t]} B_j(\frac{\gamma_{j\tau}}{B_j} + \frac{1}{(f_j - 1)B_j}) \\ &+ 1 - \alpha_{jt} - \sum_{\tau \in [a(i),t]} \gamma_{j\tau} \\ =& 1 + \frac{1}{d_j - 1} + \sum_{\tau \in [a(i),t]} \frac{1}{f_j - 1} \\ \leq& 1 + \frac{1}{d_j - 1} + \frac{T}{f_j - 1}\end{aligned}$$

When we imposes a lower bound on C_j and B_j by requiring $C_j \geq C_{min}$ and $B_j \geq B_{min}$, for all j, and let $C_{min} \to \infty$, $B_{min} \to \infty$, we have $d_j \to (e^R - 1)$ and $f_j \to (e^R - 1)$, and $\frac{\Delta D}{\Delta P(R)} \to \frac{e^R + T}{e^R - 1}$, whenever a job i is allocated to some server. Therefore, we have, under Algorithm 3,

$$\frac{\sum_{jt} C_j \alpha_{jt} + + \sum_{j\tau} B_j \gamma_{j\tau} + \sum_i \beta_i}{\sum_{ijt} k_{ijt} X_{ijt}} \leq \frac{e^R + T}{e^R - 1}.$$

Finally, by weak duality theorem, we establish that Algorithm 3 is $(R, \frac{e^R+T}{e^R-1})$-competitive-per-sample-path. By Lemma 1, Algorithm 3 is also $(R, \frac{e^R+T}{e^R-1})$-competitive-in-expectation. ■

REFERENCES

[1] H. Deng and I.-H. Hou, "Online job allocation with hard allocation ratio requirement," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pp. 1–9, IEEE, 2016.

[2] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," in *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, (New York, NY, USA), pp. 333–344, ACM, 2006.

[3] S. Acharya, B. P. Smith, and P. Parnes, "Characterizing user access to videos on the world wide web," in *IS&T/SPIE Conference on Multimedia Computing and Networking 2000: 24/01/2000-26/01/2000*, pp. 130–141, SPIE-International Society for Optical Engineering, 1999.

[4] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 133–144, 2007.

[5] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, pp. 229–238, June 2008.

[6] G. Chatzopoulou, C. Sheng, and M. Faloutsos, "A first step towards understanding popularity in youtube," in *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, pp. 1–6, March 2010.

[7] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for on-line bipartite matching," in *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, (New York, NY, USA), pp. 352–358, ACM, 1990.

[8] S. Moharir and S. Sanghavi, "Online load balancing and correlated randomness," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pp. 746–753, Oct 2012.

[9] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization.* Athena Scientific series in optimization and neural computation, Belmont, Mass. : Athena Scientific, c1997., 1997.

[10] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 190–203, 2016.

[11] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 261–276, ACM, 2009.

[12] G. Goel and A. Mehta, "Online budgeted matching in random input models with applications to adwords," in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, (Philadelphia, PA, USA), pp. 982–991, Society for Industrial and Applied Mathematics, 2008.

[13] C. Karande, A. Mehta, and P. Tripathi, "Online bipartite matching with unknown distributions," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 587–596, ACM, 2011.

[14] J. Feldman, A. Mehta, V. Mirrokni, and S. Muthukrishnan, "Online stochastic matching: Beating 1-1/e," in *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pp. 117–126, IEEE, 2009.

[15] B. Kalyanasundaram and K. R. Pruhs, "An optimal deterministic algorithm for online b-matching," *Theoretical Computer Science*, vol. 233, no. 1, pp. 319–325, 2000.

[16] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, "Adwords and generalized on-line matching," in *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pp. 264–273, Oct 2005.

[17] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, pp. 229–238, June 2008.

[18] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "Analyzing the video popularity characteristics of large-scale user generated content systems," *Networking, IEEE/ACM Transactions on*, vol. 17, pp. 1357–1370, Oct 2009.

[19] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Information and computation*, vol. 108, no. 2, pp. 212–261, 1994.

[20] A. DeSantis, G. Markowsky, and M. Wegman, "Learning probabilistic prediction functions," in *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pp. 110–119, Oct 1988.

[21] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Mach. Learn.*, vol. 2, pp. 285–318, Apr. 1988.

[22] N. Littlestone, "Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow," in *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, COLT '91, (San Francisco, CA, USA), pp. 147–156, Morgan Kaufmann Publishers Inc., 1991.

[23] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell, "Webwatcher: A learning apprentice for the world wide web," pp. 6–12, AAAI Press, 1995.

[24] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. New York, NY, USA: Cambridge University Press, 2006.

[25] M. Hutter, "On the foundations of universal sequence prediction," in *International Conference on Theory and Applications of Models of Computation*, pp. 408–420, Springer, 2006.

[26] S. Moharir, S. Sanghavi, and S. Shakkottai, "Online load balancing under graph constraints," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 1690–1703, June 2016.

[27] C.-Y. Koo, T.-W. Lam, T.-W. Ngan, K. Sadakane, and K.-K. To, "Online scheduling with tight deadlines," *Theoretical Computer Science*, vol. 295, no. 1, pp. 251 – 261, 2003. Mathematical Foundations of Computer Science.

[28] C. Dürr, Ł. Jeż, and K. T. Nguyen, *Online Scheduling of Bounded Length Jobs to Maximize Throughput*, pp. 116–127. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

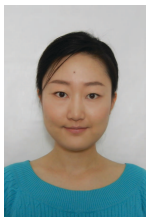
[29] L. Liu, Y. Zhou, M. Liu, G. Xu, X. Chen, D. Fan, and Q. Wang, "Preemptive hadoop jobs scheduling under a deadline," in *2012 Eighth International Conference on Semantics, Knowledge and Grids*, pp. 72–79, Oct 2012.

[30] Z. I. A. Khalib, B. R. Ahmad, and O. B. L. Ong, "High deadline meeting rate of non-preemptive dynamic soft real time scheduling algorithm," in *2012 IEEE International Conference on Control System, Computing and Engineering*, pp. 296–301, Nov 2012.

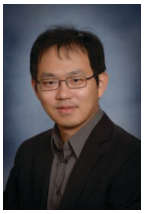
[31] M. Babaioff, Y. Mansour, N. Nisan, G. Noti, C. Curino, N. Ganapathy, I. Menache, O. Reingold, M. Tennenholtz, and E. Timnat, "Era: A framework for economic resource allocation for the cloud," in *Proceedings of the 26th International Conference on World Wide Web Companion*, pp. 635–642, International World Wide Web Conferences Steering Committee, 2017.

[32] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center.," in *NSDI*, vol. 11, pp. 22–22, 2011.

[33] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*, pp. 265–278, ACM, 2010.



Han Deng received her Ph.D. in Computer Engineering from Texas A&M University in 2017, M.S. in Electrical and Computer Engineering from Oakland University, MI, USA in 2012 and B.S. in Information Engineering in Beijing Institute of Technology, Beijing, China in 2009. She is now a postdoctoral fellow at Houston Methodist Research Institute, USA. Her research interests are in optimization and machine learning.



I-Hong Hou (S10-M12) received the B.S. in Electrical Engineering from National Taiwan University in 2004, and his M.S. and Ph.D. in Computer Science from University of Illinois, Urbana-Champaign in 2008 and 2011, respectively.

In 2012, he joined the department of Electrical and Computer Engineering at the Texas A&M University, where he is currently an assistant professor. His research interests include wireless networks, wireless sensor networks, real-time systems, distributed systems, and vehicular ad hoc networks.

Dr. Hou received the Best Paper Award in ACM MobiHoc 2018, the Best Student Paper Award in WiOpt 2018, and the C.W. Gear Outstanding Graduate Student Award from the University of Illinois at Urbana-Champaign.