# Online Load Balancing Under Graph Constraints

Sharayu Moharir, Sujay Sanghavi, and Sanjay Shakkottai, *Fellow, IEEE*

***Abstract*—In several data center settings, each arriving job may only be served by one of a subset of servers. Such a graph constraint can arise due to several reasons. One is locality of the data needed by a job; for example, in content farms (e.g., in Netflix or YouTube) a video request can only be served by a machine that possesses a copy. Motivated by this, we consider a setting where each job, on arrival, reveals a deadline and a subset of servers that can serve it. The job needs to be immediately allocated to one of these servers, and cannot be moved thereafter. Our objective is to maximize the fraction of jobs that are served before their deadlines. For this online load balancing problem, we prove an upper bound of $1 - 1/e$ on the competitive ratio of nonpreemptive online algorithms for systems with a large number of servers. We propose an algorithm—INSERT RANKING—which achieves this upper bound. The algorithm makes decisions in a correlated random way and it is inspired by the work of Karp, Vazirani, and Vazirani on online matching for bipartite graphs. We also show that two more natural algorithms, based on independent randomness, are strictly suboptimal, with a competitive ratio of 1/2.**

***Index Terms*—Communication systems, computer networks, content distribution networks.**

## I. Introduction

THIS paper looks at the problem of load-balancing among servers in the setting where each job can only be served by a restricted subset of the servers. Such a constraint arises in several settings, including in content farms and spatially distributed cloud servers. In content farms like Netflix [23] and YouTube [29], videos are replicated only among a subset of the servers; thus a request for a specific video can be served only by the corresponding subset. As another example, Web portals that provide a collection of services (e.g., Google providing e-mail, maps, video, storage, and news services) might not have all services replicated on every server. This naturally leads to an association between each request and a subset of servers, based on the request type. Furthermore, the content requests typically have a short fuse, and requests need to be served in (near) real time.

The above discussion motivates the problem in this paper: We consider online load balancing of jobs with deadlines on nonidentical servers (i.e., jobs have "hard" server preferences). In other words, each job has a deadline and can be served by only a (job-specific) subset of the servers, and these preferences are revealed only when the job enters the multiserver queuing system.[1]

Our objective is to design online algorithms that maximize the fraction of jobs that are served before their deadlines. In this work, the performance of an online algorithm is compared to the performance of a noncausal scheduler that has information of all future arrivals. We assume that the online algorithm has no knowledge of the statistics of the job arrival process. We characterize the performance of an online algorithm by its competitive ratio, which is the ratio of the expected number of jobs served by the online algorithm to the number of jobs served by the optimal offline algorithm, minimized over all input sequences.

We visualize the system as a bipartite graph between servers and jobs, where an edge between a job and a server indicates that that particular job can be served by that particular server. The task of allocating jobs to servers is equivalent to the task of finding a generalized matching in the bipartite graph where a generalized matching is a matching where one server could be matched to multiple jobs, but one job is matched to at most one server.

### A. Contributions

1) *Upper bound*: We show an upper (i.e., outer) bound of $1 - 1/e$ on the competitive ratio of *any* randomized load balancing algorithm (Theorem 1).
2) *Algorithm and its performance*: Our main contribution is the INSERT RANKING algorithm. If multiple servers can serve a job before its deadline elapses, INSERT RANKING breaks ties in a randomized manner and the tie-breaking policy is *correlated* across jobs arriving in a time-slot and also across multiple time-slots. We show a lower bound (i.e., achievable) of $1 - 1/e$ on the competitive ratio of INSERT RANKING (Theorem 2). This proves the optimality of INSERT RANKING.
3) *Other algorithms*: We analyze the performance of two intuitive randomized algorithms that do not use correlation in tie-breaking. One is a join the shortest queue algorithm that breaks ties uniformly at random *independent* of past choices, and the second algorithm is biased toward joining shorter queues and breaks ties in a random manner, independent of past choices. We show an upper (i.e., outer) bound of 1/2 on the competitive ratio of both algorithms and show that INSERT RANKING strictly outperforms them, highlighting the importance of correlated randomization (Theorems 3 and 4).

Manuscript received April 23, 2014; revised November 02, 2014 and February 27, 2015; accepted April 04, 2015; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor X. Wang. Date of publication July 06, 2015; date of current version June 14, 2016. This work was supported in part by the NSF under Grant CNS-0964391.

The authors re with the Wireless Networking and Communications Group (WNCG), Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: sharayu.moharir@gmail.com; sanghavi@mail.utexas.edu; shakkott@austin.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2442597

[1]Earlier versions of this work were presented at ACM SIGMETRICS, Pittsburgh, PA, USA, June 2013 [21] and a talk at the 51st Annual Allerton Conference on Communication, Control, and Computing, October 2013.

### B. Related Work

*1) Online Load Balancing With Hard Deadlines:* There is a vast amount of literature on load balancing. We focus on load balancing in the adversarial setting and similar to our setting of jobs with hard deadlines.

Online admission control in the hard deadline model when there is a single server that services a sequence of jobs in a nonpreemptive manner has been studied in [10]. Load balancing jobs with hard deadlines for systems with $m$ identical machines has been studied in [6] for jobs with identical processing time. In [6], the goal is to minimize the total number of jobs dropped from the system. The key results are an optimal 3/2 competitive online algorithm for $m = 2$ and lower bounds on deterministic algorithms for the general case. The case of $m = 2$ has also been studied in [11]. The scheduling problem for jobs with equal processing times has been studied in [5], where the goal is to schedule jobs on a single processor in a nonpreemptive manner to maximize the number of completed jobs. Load balancing for batch arrivals of jobs with equal processing times has been considered in [2]. The competitiveness of online deadline scheduling problems where jobs are nonpreemptive and the goal is to maximize the sum of the length of jobs completed before their deadlines has been analyzed in [17]. Online preemptive scheduling problem for jobs with deadlines has been studied in [4]. Closer to our setting of trying to maximize the number of jobs completed before their deadlines where all jobs need a service of one time-slot, [8] looks at load balancing these jobs on $m$ *identical* machines and shows a lower bound of $e/(e-1)$ for large values of $m$. Scheduling jobs with deadlines for wireless networks has been studied in [7], [12], [13], [22], [24], and [25].

*2) Online Matching:* The problem of online matching in bipartite graphs has been studied in [15], where the authors introduced the idea of using correlated randomness for online matching. Various extensions to weighted graphs and the application of online weighted bipartite matching to ad-words has been studied in [1], [9], and [18]. The problem of Online $b-$Matching has been studied in [14]. Closer to our setting, load balancing for bipartite graphs has been studied in [19] in the case where there are no departures from the system and no deadlines.

*3) Stochastic Queueing Systems:* In traditional stochastic queueing systems, the Lyapunov optimization method is used to prove system stability [26], [27] and can be combined with an "outer" optimization (e.g., maximize system utility subject to stability). In our setting, stability is not the key concern—indeed, the system is always stable irrespective of the policy due to hard deadlines of jobs, as those jobs that cannot be served by their deadline are dropped from the system, whereas in the traditional stochastic queueing setting, all jobs are allocated to a server and do not leave the system until they are served.

## II. System Model

We consider a multiserver discrete-time queueing system. We assume that jobs come into the system at the beginning of each slot. If there are multiple arrivals in a slot, they are ordered. Each job can only be served by a subset of the servers; *this subset is revealed only when a job arrives*. Each job takes 1 time-slot, on any of the servers that can serve it. We present some extensions for the case when jobs need more than one time-slot of service in the Appendix. Furthermore, every job $p$ has a *deadline* $d_p$ associated with it; this is the maximum number of slots it can wait, after arrival. There exists a finite maximum $d_{\max} < \infty$ such that $d_p \leq d_{\max}$ for all $p$. See Fig. 1 for an illustration of the system model.

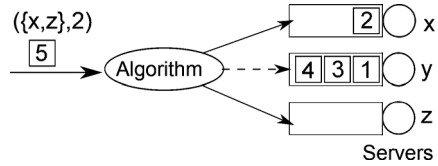



Fig. 1. System model for online load balancing: illustration of a system with 3 servers. Job 5 has a server subset $\{x, z\}$ and a deadline of 2 time-slots, and the scheduling algorithm needs to decide whether to send the job to server $x$ or $z$ or drop the job.

Our task is to allocate jobs to servers so as to maximize the number served by their deadline. Any job, once allocated, cannot be revoked/moved to another choice. Thus, a job $p$ will not be served if, at the time of its arrival, all the servers it can be allocated to already have more than $d_p$ jobs in them. We consider algorithm design in the competitive ratio (a.k.a. worst-case/adversarial) setting, where the algorithm has no knowledge of any aspect of the arrivals—i.e., it does not know how many packets arrive at each time, and what the deadline and server-subset of each one is. We will allow for both deterministic and randomized algorithms.

We measure the performance of any online allocation algorithm alg in terms of its *competitive ratio* $\rho$, where

$$\rho(\text{alg}) = \inf_{t>0} \left( \min_{A \in A_t} \left( \frac{E[S_{\text{alg}}(A)]}{S_{\text{opt}}(A)} \right) \right)$$

where $A_t$ is the set of all arrival processes such that there are no arrivals after time-slot $t$, $E[S_{\text{alg}}(A)]$ is the expected number of jobs served within their deadline by the (possibly randomized) algorithm, and $S_{\text{opt}}(A)$ is the number of jobs served within their deadline by the offline optimal algorithm, which is the "genie" algorithm that knows the entire sequence *a priori*.

## III. Main Results and Discussion

In this section, we state and discuss our main results. The proofs are discussed in Sections IV–VII.

### A. Simple Special Case

Before formally describing the algorithm in Section III-B, we provide intuition and illustration by considering a simple arrival process where all jobs arrive at the beginning of time-slot 1 and no further arrivals occur. In addition, we assume that all jobs have a deadline of $b$ where $b := d_{\max}$ and are revealed to the system sequentially. The scheduling algorithm has to assign a job to a server before the next job arrives, thus making this an online scheduling problem.

Since all jobs have a deadline of $b$, to serve all jobs that are queued at a server before their deadlines, we need to ensure that no server is allocated more than $b$ jobs.[2]

[2]This problem is equivalent to online $b-$Matching in bipartite graphs, also studied in [14].

Consider now a convenient representation of the problem, as an *extended* bipartite graph $(U_b, V, E)$, defined as follows: For each server $u \in U$, we now make $b$ copies to obtain the vertex set $U_b$ that is one partition of the graph; thus, $|U_b| = b|U|$. The other partition is just $V$, the set of jobs. Recall that each job can only be served by a subset of the servers. Correspondingly, let edge $(u, v) \in E$ if and only if $u$ is a copy of a server that can serve $v$. An example of this representation and the algorithm is in Fig. 2.

The advantage of the extended graph representation is that we have now converted the online allocation problem into one of simple online matching (i.e., one job to one (copy of) a server) on the extended graph. In particular, recall that in our system model, the vertices in $V$ arrive in arbitrary order and need to be allocated on arrival. This is exactly the same as finding a matching node $u \in U_b$ in the extended graph; the fact that each server has at most $b$ copies ensures the maximum loading is never exceeded.

With this setting, our algorithm first chooses a permutation $\pi_b$ of $U_b$ uniformly at random from all possible permutations. This permutation, which we call the *ranking*, is chosen at the beginning and *fixed* thereafter. Note that if $(u, v) \in E$, $v$ has an edge to all copies of $u$ in $\pi_b$. As each vertex in $V$ is revealed sequentially, it is matched to the highest ranked unmatched vertex in $\pi_b$ that it has an edge to.

*Note:* Every vertex in $v$ chooses its vertex according to the same ranking; indeed once this *ranking* is picked, the rest of the algorithm is deterministic. This is what we mean when we say choices are made in a *correlated* random way; the fixing of a single ranking governs the choices of all jobs.

The following proposition characterizes the performance of the algorithm in terms of fraction of jobs served, for this simple special case.

*Proposition 1:* For any set of arrivals in accordance with the special case described above—i.e., all jobs arrive in some order in time-slot 1 with deadlines $b = d_{\max}$—let $V^*$ be the number of jobs that can be served by the offline optimal "genie" algorithm. Then, the number of jobs served by our algorithm is at least

$$\sum_{i=1}^{V^*} \left(1 + \frac{1}{V^*}\right)^{-i}.$$

As $V^* \to \infty$, the competitive ratio—i.e., the above quantity divided by $V^*$—becomes $1 - \frac{1}{e}$.

Note that, as mentioned, this immediately implies the same competitive ratio for the graph-theoretic problem of online $b$-matching, resolving an open issue described in [14].

### B. Upper Bound

We now consider again the general case, where arrivals occur over time, and deadlines are completely arbitrary (but bounded by $d_{\max}$). In this section, we present an upper bound on the competitive ratio of any (possibly randomized) online allocation algorithm; in fact we prove a stronger result, that this worst case can be achieved even by sequences with homogenous deadlines, where every job has a deadline equal to $d_{\max}$, for any $d_{\max}$.

*Theorem 1:* As the number of servers $n \to \infty$, there exists an arrival sequence consisting of jobs with homogenous deadlines of $d_{\max}$ (for any value of $d_{\max}$), for which the ratio of the expected fraction of jobs matched by any online load balancing algorithm to the number of jobs matched by the optimal offline algorithm is at most $\leq 1 - \frac{1}{e}$.

Therefore, as the number of servers $n \to \infty$, the competitive ratio $\rho$, of any online load balancing algorithm for jobs with strict deadlines is $\leq 1 - \frac{1}{e}$.

Note that the competitive ratio is defined by a minimum over arrival sequences. The above is stronger because it says the minimum even over the restricted set—with homogenous deadlines—is no better, no matter what the value of this homogenous deadline.

The upper bound serves the dual purpose of showing the optimality of our algorithm and the gap to optimality of other (more intuitive) online allocation algorithms.

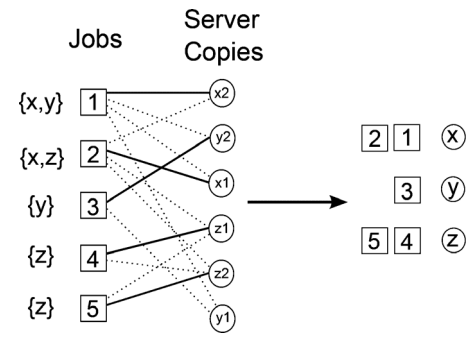



Fig. 2. Illustration of our algorithm for the simple case of arrivals only in time 1, with all deadlines being $d_{\max} = b = 2$. Here, the server set is $U = \{x, y, z\}$, and the job set is $V = \{1, 2, 3, 4, 5\}$. In the extended graph, there are two copies of each server, so $U_b = \{x1, y1, z1, x2, y2, z2\}$. Our algorithm picks a random permutation, i.e., ranking, $\pi_b$ of this set $U_b$ and fixes it, as shown on the left. Each vertex in set $V$ is then matched to the highest ranked available vertex in $U_b$; this results in the matching on the extended graph, on the left. The figure on the right shows the collapsing back from extended graph to a server allocation; it shows the resulting allocation with max load of 2 on each server.

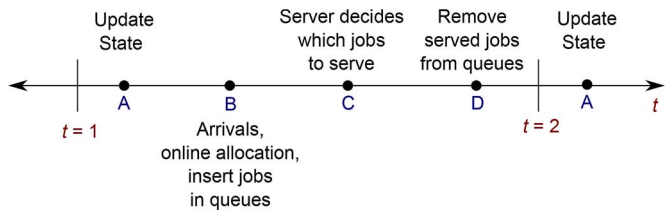



Fig. 3. INSERT RANKING: timeline.

### C. Our Algorithm and Its Performance

The algorithm consists of two parts: a dispatcher and a scheduler. The dispatcher decides which server will serve a job in an online manner, and a scheduler decides which job in its queue is served by a server in a given time-slot.

Conventional wisdom dictates that each job should be dispatched to one of the servers that can serve it and has the smallest queue, however we show that the performance of the naive join the shortest queue algorithm is suboptimal (Theorem 3).

We now describe, first in words and then formally, our online algorithm INSERT RANKING. In each time-slot, the algorithm implements a series of steps as shown in Fig. 3.

Let $S$ be the set of servers. Recall that in every time-slot, we have a sequence of jobs arriving; each job $p$ reveals a subset of

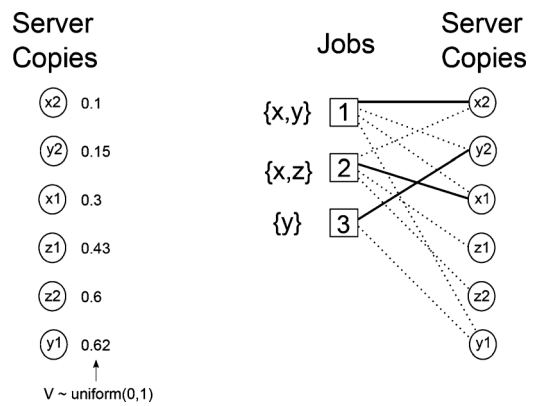



Fig. 4. Illustration of INSERT RANKING. Here, the server set is $U = \{x, y, z\}$ and $d_{\max} = 2$. Let all three incoming jobs have a deadline of 2 time-slots. The figure on the left is an illustration of INITIALIZE, and the figure on the right is an illustration of ONLINE ALLOCATION for time-slot 1.

servers $S_p \subseteq S$ that can serve it, a deadline $d_p$, and has to be allocated in an online manner.

In the first time-slot:

A. INITIALIZE: Generate $d_{\max}$ labeled copies of each server with labels $1, 2, \ldots, d_{\max}$. Let $s_i^{(j)}$ denote the copy of $s_i \in S$ labeled $j$. Each labeled server copy $s_i^{(j)}$ has a value $V_i^{(j)}$ associated with it, chosen independently according to the uniform distribution on [0, 1]. Refer to Fig. 4 for an illustrative example.

B. ONLINE ALLOCATION: All jobs arriving in time-slot 1 are matched to server copies in an online manner as follows: For each incoming job:
   1. Create the neighborhood of a job $p$, defined as all server copies of each $s_i$ in its server subset with label less than $d_p$ (the label $j$ of a server copy indicates that a job assigned to that server copy has to be served before the end of time-slot $j$. Therefore, each job $p$ defined by its server subset and deadline, $\{S_p, d_p\}$ arriving at time 1 can be served by each $s \in S_p$ in time-slots $1, \ldots, d_p$).
   2. Match each incoming job to that unmatched server copy in its neighborhood that has the lowest value $V$.

   Refer to Fig. 4 for an illustrative example.

C. SCHEDULE: At the end of job allocation for time-slot 1, all matched jobs are inserted in the queues of the corresponding servers. Each server $s_i \in S$ serves that job in its queue that was matched to the copy of $s_i$ with the smallest label. Refer to Fig. 5 for an illustrative example.

In every *subsequent* time-slot $t(> 1)$:

A) UPDATE STATE:
   1. Remove all matched server copies.
   2. Remove all server copies labeled $t - 1$.
   3. Add one new copy of each server with label $t + d_{\max}$ to the set of unmatched servers. Each new server copy $S_i^{(t+d_{\max})}$ has a value $V_i^{(t+d_{\max})}$ associated with it, chosen independently according to the uniform distribution on [0, 1].
   4. If the label $j$ of the server copy matched to the job served by a server in the previous slot $(t-1)$ is greater than $t-1$, add $s_i^{(j)}$ back to the set of unmatched server copies.

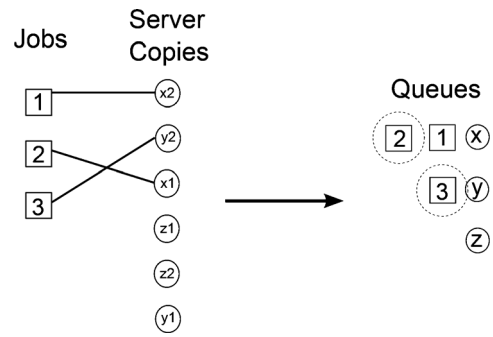



Fig. 5. Illustration of SCHEDULE for time-slot 1. Server $x$ serves job 2 since it was matched to $x1$, server $y$ servers job 3, server $z$ remains idle.

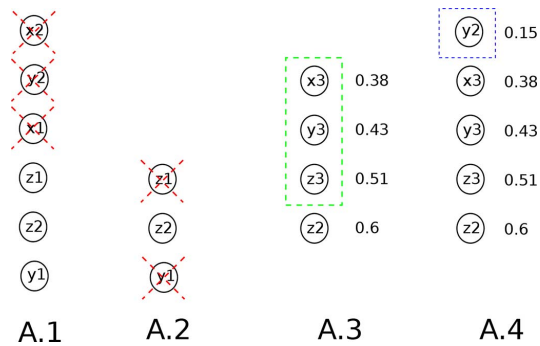



Fig. 6. Illustration of UPDATE STATE for time-slot 2.

Refer to Fig. 6 for an illustrative example.

B) ONLINE ALLOCATION: All jobs arriving in time-slot $t$ are matched to server copies in an online manner as follows: the neighborhood of a job $p$ is defined as all server copies of each $s_i$ in its server subset with label less than $t + d_p - 1$, and each incoming job is matched to that unmatched server copy in its neighborhood that has the lowest value $V$.

C) SCHEDULE: At the end of job allocation for time-slot $t$, all matched jobs are inserted in the queues of the corresponding servers. Each server $s_i \in S$ serves that job in its queue that was matched to the copy of $s_i$ with the smallest label.

A more formal definition of the algorithm is presented in Algorithm 1.

*Comments:*

1) INSERT RANKING makes its random choice in a correlated way. The correlated randomness is implemented via the choice of values $V$ for server copies. Once these values are chosen, the relative priority between two server copies is correlated across jobs in a given slot, and also across time.
2) INSERT RANKING is NOT a join the shortest queue algorithm. However, it is weighted toward joining shorter queues. We will see in Section III-D that it outperforms natural shortest-queue algorithms.
3) Servers implementing INSERT RANKING are work-conserving.
4) Before allocating jobs that arrive in time-slot $t$, all server copies with labels $\leq t - 1$ are removed from the set of unmatched server copies.

**Algorithm 1** INSERT RANKING

INITIALIZE: $t = 1$, $\{s_i^{(j)} : j = t, \ldots, t + d_{\max} - 1$ and $s_i \in S\}$ and a real number $V_i^{(j)}$ for each $s_i^{(j)}$ chosen uniformly from $[0, 1]$.
ONLINE ALLOCATION:
**for** arriving job $p = \{S_p, d_p\}$ in time-slot $t$, **do**
  make neighborhood:

$$N(p) = \{s_i^{(j)} : s_i \in S_p \text{ and } t \le j \le t + d_p - 1\}.$$

  match $p$ to currently unmatched $s \in N(p)$ that has the lowest $V$.
**end for**
SCHEDULE:
**for** each server $s_i$, **do**
  add all jobs matched to copies of server $i$ in time-slot $t$ to the queue of server $s_i$.
  serve currently queued job which was matched to the lowest labeled copy of server $s_i$.
**end for**
UPDATE STATE:
$t = t + 1$.
**for** each server $s_i$, **do**
  remove all matched copies of $s_i$.
  remove $s_i^{(t-1)}$.
  add $s_i^{(t+d_{\max})}$, choose a real number $V_i^{(t+d_{\max})}$ uniformly from $[0, 1]$.
  if $s_i$ served job $p$ in time-slot $t - 1$ and $p$ was matched to $s_i^{(j)}$ such that $j > t - 1$, add $s_i^{(j)}$ back.
**end for**
Goto 2.

5) From the previous comment, it follows that at any given time, there are at most $nd_{\max}$ unmatched server copies, therefore the storage requirement of INSERT RANKING is $O(n)$.

We now prove a lower bound on the competitive ratio of INSERT RANKING.

*Theorem 2:* The competitive ratio of INSERT RANKING is $\ge 1 - \frac{1}{e}$.

From Theorems 1 and 2, we conclude that INSERT RANKING is an asymptotically optimal load-balancing algorithm.

### *D. Performance of Other Algorithms*

We now analyze the performance of two intuitive randomized algorithms that do not employ correlation in the randomness across jobs. The first obvious candidate algorithm is a join the shortest queue (JSQ) algorithm in which ties are broken in an uniformly random manner. We refer to this algorithm as RANDOMIZED Join the Shortest Queue (RANDOMIZED JSQ). RANDOMIZED JSQ allocates each incoming job to the currently least loaded server (server with the shorted queue) breaking ties uniformly at random independent of past choices. A formal definition of the algorithm is presented in Algorithm 2.

*Theorem 3:* As the number of servers $n \to \infty$, the competitive ratio of RANDOMIZED JSQ is $\le \frac{1}{2}$.

**Algorithm 2** RANDOMIZED JSQ

Initialize all QUEUE-LENGTHS $Q_i$ to 0.
**for** each $t$ **do**
  **for** arriving job $p$ in time-slot $t$ with server set $S_p$ and deadline $d_p$ **do**
    $Q_{\min} = \min_{s_i \in S_p} Q_i$.
    **if** $Q_{\min} < d_p$ **then**
      SHORTEST QUEUES $= \{s_i : s_i \in S_p, Q_i = Q_{\min}\}$.
      Allocate $p$ to server $s_j \in$ SHORTEST QUEUES breaking ties uniformly at random independent of all past decisions.
      Update QUEUE-LENGTHS.
    **end if**
  **end for**
  Each server serves that job in its queue which has the nearest deadline. Update $Q_i$ to $(Q_i - 1)^+$ for each $i$.
**end for**

**Algorithm 3** RANDOMIZED P-JSQ

Initialize all QUEUE-LENGTHS $Q_i$ to 0.
**for** each $t$ **do**
  **for** arriving job $p$ in time-slot $t$ with server set $S_p$ and deadline $d_p$ **do**
    **if** $\max_{s_j}(d_p - Q_j) > 0$ **then**
      Allocate $p$ to server $s_j \in S_p$ w.p. $\frac{(d_p - Q_j)^+}{\sum_{s_i \in S_p}(d_p - Q_i)^+}$ independent of all past decisions.
      Update QUEUE-LENGTHS.
    **end if**
  **end for**
  Each server serves that job in its queue which has the nearest deadline. Update $Q_i$ to $(Q_i - 1)^+$ for each $i$.
**end for**

We thus conclude that INSERT RANKING strictly outperforms RANDOMIZED JSQ.

INSERT RANKING is not a join the shortest queue algorithm, but it is biased toward joining shorter queues. To compare it to an algorithm that is similarly biased toward joining shorter queues but makes its decisions in an uncorrelated manner, we propose another algorithm that we refer to as the RANDOMIZED PROBABILISTIC-Join the Shortest Queue (RANDOMIZED P-JSQ) algorithm. RANDOMIZED P-JSQ allocates each incoming job $p$ with deadline $d_p$ to server $s_i$ with probability proportional to $(d_p - Q_i)^+$ independent of past choices. A formal definition of the algorithm is presented in Algorithm 3.

*Theorem 4:* As the number of servers $n \to \infty$, the competitive ratio of RANDOMIZED P-JSQ is $\le \frac{1}{2}$.

From Theorem 4, we see that INSERT RANKING strictly outperforms RANDOMIZED P-JSQ, and we conclude that correlated randomness is key in the good performance of INSERT RANKING.

## IV. Proof of Proposition 1

Recall the extended bipartite graph representation discussed in Section III, where we converted the problem of load balancing on $G(U,V,E)$to the problem of finding a matching in the bipartite graph $G_b(U_b,V,E_b)$.

For any graph $G$, on average, the algorithm RANKING proposed in [15] applied on $G_b(U_b,V,E_b)$ matches the same number of vertices of $V$ as applying our algorithm on $G(U,V,E)$.

Therefore, once the load balancing problem on $G(U,V,E)$ is converted to a matching problem for the bipartite graph $G_b(U_b,V,E_b)$, Proposition 1 follows as a direct consequence of the updated proof of the performance of the algorithm RANKING proposed in [15] for online bipartite matching presented in [3] and [1].

## V. Proof of Theorem 1

To upper-bound the performance of all online load balancing algorithms, we construct a graph between jobs and servers and specify an arrival sequence on the jobs and upper-bound the competitive ratio of any online load balancing algorithm by upper-bounding the performance of any online algorithm for this arrival sequence.

We consider an arrival sequence in which jobs arrive only the first time-slot and all of them have to be served in the next $b = d_{\max}$ time-slots. Therefore, we are looking for an allocation where not more than $b$ jobs are allocated to any one server. We refer to such an allocation as a $b-$Matching.

We visualize the problem of $b-$Matching on a bipartite graph between jobs and servers as the problem of finding a matching in a bigger graph where each server is replicated $b$ times. The analysis in this section uses proof techniques used in [15] to upper-bound the performance of all online randomized algorithms for bipartite matching. As in [15], we use a matrix $S$ to describe the bipartite graph.

The key difference in the analysis in [15] and the analysis for $b-$Matching is that the matrix (or graph) used to upper-bound the performance of any online algorithm in [15] is the upper triangular matrix, which is not a valid matrix in the $b-$Matching case as no two rows of the upper triangular matrix are identical. Therefore, several key steps in the proof in [15] do not apply to the $b-$Matching problem.

Based on the proof in [15], we now outline our proof in five steps.

1) We first characterize a graph $S$ that we use to upper-bound the performance of all online $b-$Matching algorithms.
2) We prove that for a graph $\{S,\pi\}$, which is obtained by permuting the rows of $S$ by a permutation $\pi$, chosen uniformly at random, the best deterministic algorithm is greedy in the sense that it never leaves a column unmatched if there is an eligible row when this column arrives (Lemma 1).
3) We consider an algorithm called RANDOM [15], which matches each column in $A$ to a randomly chosen eligible row, independent of all past choices. We show that for an arrival sequence from $\{S,\pi\}$, the expected size of matching produced by any deterministic algorithm is the same as the expected size of matching produced by RANDOM on $S$ (Lemma 3).
4) We use Yao's Lemma [28] to upper-bound the expected performance of any randomized algorithm on $S$ by the expected performance of the best deterministic algorithm when given a problem instance $\{S,\pi\}$. Using this and the previous two results, we upper-bound the expected performance of any randomized algorithm on $S$ by the expected performance of RANDOM on $S$ (Lemma 4).
5) We evaluate the performance of RANDOM on $S$ to get the upper bound (Lemma 5).

The analysis in [15] also proceeds via Steps 2–5, but for the upper triangular matrix. The specific details of the proofs are different because in our case $S$ is not upper triangular. The main challenge in our analysis is Step 5. In [15], for the upper triangular matrix, it is possible to exactly characterize the performance of RANDOM. In our case, exact characterization is difficult and therefore we upper-bound the performance of RANDOM on $S$. We show that the dominant term in the bound scales correctly and the error term in sublinear in $|V|$.

We represent the arrival sequence using a matrix $A$. The columns of $A$ represent jobs, and the rows of $A$ represent servers. $A(i,j)=1$ means that job $j$ can be served by server $i$. Let $U$ denote the set of servers. As a matter of convention, we assume that jobs (columns) arrive from right to left. For the proof of Theorem 1, we consider the case with $A$ generated using the upper triangular matrix by replicating each column $b$ times, all jobs arrive at the beginning of time-slot 1, and each job has a deadline of $b$.

For instance, for $n=4$ and $b=2$

$$A=\begin{bmatrix}1&1&1&1&1&1&1&1\\0&0&1&1&1&1&1&1\\0&0&0&0&1&1&1&1\\0&0&0&0&0&0&1&1\end{bmatrix}.$$

We map the task of online load balancing for the arrival sequence $A$ to the task of online matching for a matrix $S$ that is generated using the matrix $A$ as follows.

We create $b$ copies of each server in $U$ to get a set $U_b$;the $k$th copy of server $u\in U$ represents server $u$ in time-slot $k$ for $1\le k\le b$. The rows of the matrix $S$ represent the set $U_b$, and each column of $S$ represents a job. The entry $S(i,j)=1$ if job $j$ can be served by $u\in U$, and row $i$ of $S$ represents a copy of server $u$ and 0 otherwise. Since each vertex in $u$ is replicated $b$ times in $S$, the $b$ rows of $S$ that represent $u$ are identical. In addition, since all jobs represented in matrix $A$ have a deadline of $b$, the task of load balancing for $A$ is equivalent to finding a matching for the matrix $S$.

Therefore, for $n=4$ and $b=2$

$$S=\begin{bmatrix}1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1\\0&0&1&1&1&1&1&1\\0&0&1&1&1&1&1&1\\0&0&0&0&1&1&1&1\\0&0&0&0&1&1&1&1\\0&0&0&0&0&0&1&1\\0&0&0&0&0&0&1&1\end{bmatrix}.$$

This point onward, we will focus on algorithms for bipartite matching on $S$ instead of algorithms for finding $b-$Matchings on $A$.

For the matrix $S$ defined above, with every permutation $\pi$ on $\{1,\ldots,bn\}$, associate a problem instance $\{S,\pi\}$ such that the

adjacency matrix is obtained by permuting the rows of $S$ under $\pi$ and the columns or jobs arrive in the order $bn, bn-1, \ldots, 1$. Let $Z$ denote the uniform distribution over the $(bn)!$ possible permutations.

*Lemma 1:* The best deterministic algorithm is greedy, i.e., it never leaves a column unmatched if there is an eligible row.

*Proof:* Assume that there exists a deterministic algorithm ALG that is not greedy and it is the best possible deterministic algorithm. Now consider an alternative greedy algorithm ALG-GREEDY that does the following. If ALG allocates an incoming column to a particular row, and that row is available, then ALG-GREEDY also matches them. If that row is not available, then ALG-GREEDY matches that column to an arbitrary available row. If ALG does not allocate a column to any row, ALG-GREEDY again matches that column to any arbitrary available row.

We will now prove that the number of rows matched by ALG-GREEDY is $\geq$ the number of rows matched by ALG.

CLAIM: The set of rows matched by ALG-GREEDY at time $t$ is a superset of the set of rows matched by ALG at time $t$.

We prove the claim by induction. It is clearly true for $t = 1$. Let it be true at $t$. We consider the following three cases.

1) Both ALG and ALG-GREEDY match the next column to the same row. In this case, the claim is true for $t + 1$.
2) ALG does not match the next column. In this case too, the claim is true for $t + 1$.
3) ALG matches the next column to a row $r$ and ALG-GREEDY does not match it to row $r$. This can only happen if row $r$ has already been matched by ALG-GREEDY. Therefore, the claim is true for $t + 1$.

This proves that the number of rows matched produced by ALG GREEDY is $\geq$ the number of rows matched by ALG.

This contradicts our assumption that the best deterministic algorithm is not greedy, hence the best deterministic algorithm is greedy. ∎

*Lemma 2:* For ALG on $\{S, \pi\}$ and RANDOM on $\{S, I\}$ where $I$ is the identity permutation, if the number of eligible rows at time $t$ is $k$, they are equally likely to be any set of $k$ rows from among the first $b(n - \lfloor (t-1)/b \rfloor)$ rows of $S$.

*Proof:* We first prove the lemma for RANDOM by induction. The claim is true at time 1. Assume that it is true at time $t$.

*Case I:* $((t-1) \bmod b) \neq 0$*:* By the induction hypothesis at time $t$, we have that for the algorithm RANDOM, if there are $k$ eligible rows, they are equally likely to be any of the first $b(n - \lfloor (t-1)/b \rfloor)$ rows of $S$. The number of eligible rows at time $t + 1$ will then be $k - 1$ since columns $n - t + 1$ and $n - t$ have the same neighbors. For each subset of eligible rows of size $k$ at time $t$, there are $k - 1$ possible subsets of eligible rows at time $t + 1$, and each one of them is equally likely with probability $1/k$ because the algorithm RANDOM breaks ties among eligible rows uniformly at random. We therefore have that the claim is true for time $t + 1$ for RANDOM.

*Case II:* $((t-1) \bmod b) = 0$*:* By the induction hypothesis at time $t$, we have that for the algorithm RANDOM, if there are $k$ eligible rows, they are equally likely to be any of the first $b(n - \lfloor (t-1)/b \rfloor)$ rows of $S$. Let $k'$ of these rows be such that they are eligible to be matched with the column arriving at time $t+1$. Since all sets of size $k$ are equally likely, all sets that contain $k'$ rows that can be used by the column arriving at $t+1$ are also equally likely. We consider two cases. In the first case, the column that arrives at $t$ is matched to one of these $k'$ rows. The number of eligible rows at time $t + 1$ will then be $k' - 1$. Since RANDOM breaks ties uniformly at random, this set can be any one of the $k' - 1$ sets with probability $1/k'$. The second case is when the column that arrives at $t$ is not matched to one of these $k'$ rows. In this case, the number of eligible rows at time $t + 1$ will then be $k'$ and every such set is equally likely.

The proof of the first claim for ALG also follows by induction on time. It centers on the fact that at any time, if ALG chooses to match column to a particular eligible row, that row is equally likely to be any one of the rows of $S$ that is eligible to be matched to the column arriving at time $t$. This follows from the fact that the permutation $\pi$ of the rows of $S$ to get $\{S, \pi\}$ is chosen uniformly at random from the $(nb)!$ possible permutations.

The next lemma shows that on average, any greedy deterministic algorithm on $\{S, \pi\}$ has the same performance as that of RANDOM on $\{S, I\}$. There is an analogous result in [15] for the upper triangular matrix.

*Lemma 3:* Let ALG be a greedy deterministic online algorithm. The expected size of matching produced by ALG on $\{S, \pi\}$ where $\pi$ is picked according to the distribution $Z$ is the same as the expected size of matching produced by RANDOM on $(S, I)$.

*Proof:* We claim that for each $k$ and $t$, the probability that there are $k$ eligible rows at time $t$ is the same for RANDOM run on $(S, I)$ as it is for ALG run on $\{S, \pi\}$.

The claim is true for $t = 1$. We assume that it is true at time $t$ and show that this implies that it is also true for time $t + 1$.

*Case I:* $((t-1) \bmod b) \neq 0$*:* Let $P_{\mathrm{RANDOM}}(t, k)$ be the probability that for RANDOM on $(S, I)$, there are $k$ eligible rows at time $t$ and $P_{\mathrm{ALG}}(t, k)$ the same probability for ALG. By the induction hypothesis, we have that $P_{\mathrm{RANDOM}}(t, k) = P_{\mathrm{ALG}}(t, k)$. Given this, since the columns arriving at $t$ and $t + 1$ have the same neighbors, we have that $P_{\mathrm{RANDOM}}(t + 1, k-1) = P_{\mathrm{RANDOM}}(t, k)$ and $P_{\mathrm{ALG}}(t+1, k-1) = P_{\mathrm{ALG}}(t, k)$, and therefore the claim is true for $t + 1$.

*Case II:* $((t-1) \bmod b) = 0$*:* By the induction hypothesis, we have that $P_{\mathrm{RANDOM}}(t, k) = P_{\mathrm{ALG}}(t, k)$. By the first claim, we have that each such set of $k$ rows is equally likely for both algorithms. Therefore, the probability that the two sets have $k'$ rows that can be matched to the vertex arriving at $t + 1$ is also the same. Let this probability be $\gamma(k', k)$. Let

$$\beta(k', k) = \left( \gamma(k', k).\frac{k - k'}{k} + \gamma(k' + 1, k).\frac{k' + 1}{k} \right).$$

Since RANDOM breaks ties uniformly at random and ALG breaks ties deterministically, but that particular row is equally likely to be any eligible row $S$, we have that

$$P_{\mathrm{RANDOM}}(t + 1, k') = \sum_{k \geq k'} P_{\mathrm{RANDOM}}(t, k)\beta(k', k)$$

and

$$P_{\mathrm{ALG}}(t + 1, k') = \sum_{k \geq k'} P_{\mathrm{ALG}}(t, k)\beta(k', k).$$

Therefore, we conclude that $P_{\mathrm{RANDOM}}(t+1, k') = P_{\mathrm{ALG}}(t + 1, k')$.

An incoming column at time $t$ will not be matched only when there are no rows to match it to. Since $P_{\mathrm{RANDOM}}(t, 0) = P_{\mathrm{ALG}}(t, 0)$ for all $t$, the result follows. ∎

*Lemma 4:* The expected number of rows matched by any online matching algorithm is upper-bounded by the expected number of rows matched by RANDOM on $(S, I)$.

*Proof:* Fix a randomized algorithm $R$. Let $\mathbb{D}$ be the set of all deterministic algorithms and $\mathbb{D}_G$ be the set of all greedy deterministic algorithms. Let $E[R(S,\pi)]$ be the expected size of matching produced by $R$ on $(S,\pi)$, and let $E[D(S,Z)]$ denote the expected size of matching produced by $D \in \mathbb{D}$ given an input from the distribution $Z$. By Yao's Lemma [28], we have that

$$\min_{\pi}\{E[R(S,\pi)]\} \leq \max_{D\in\mathbb{D}}\{E[D(S,Z)]\}.$$

By Lemma 1 we know that the best deterministic algorithm is greedy. Therefore, we can conclude that

$$\min_{\pi}\{E[R(S,\pi)]\} \leq \max_{D\in\mathbb{D}_G}\{E[D(S,Z)]\}.$$

By Lemma 3, we have the expected size of matching produced by any greedy deterministic algorithm given an instance $(S,\pi)$ is the same as the expected size of matching produced by RANDOM on $(S, I)$. Therefore

$$\min_{\pi}\{E[R(S,\pi)]\} \leq E[|M_{\mathrm{RANDOM}}(S,I)|]$$

where $M_{\mathrm{RANDOM}}(S,I)$ is the matching produced by RANDOM on the input $(S,I)$. Therefore, we have that the performance of any online matching algorithm is upper-bounded by the expected size of matching produced by RANDOM on $(S,I)$. ∎

*Lemma 5:* The expected number of rows matched when RANDOM is executed on $(S,I)$ is at most $bn(1-\frac{1}{e})+o(n)$.

*Proof:* We first classify the columns of the matrix $S$. Observe that columns $bk+1$ to $bk+b$ for all integral values of $k$ such that $1 \leq k \leq n$ are identical. We refer to columns $bk+1$ to $bk+b$ as columns of type $k$. Let $x(t)$ be the number of types of columns remaining at time $t$ and $y(t)$ be the number of rows eligible at time $t$. Let $\Delta x = x(bt+b) - x(bt)$ and $\Delta y = y(bt+b) - y(bt)$. For any column $j$ such that $bk+1 \leq j \leq bk+b$, we refer to the rows $bk+1$ to $bk+b$ as the good rows and the others as bad rows.

The following arguments use the fact we proved in Lemma 2 that if there are $r$ eligible rows when column $c$ arrives at time $t$, they are equally likely to be any $r$ of the first $b(n-\lfloor(t-1)/b\rfloor)$ rows of $S$.

If a column is matched to a bad row by RANDOM when there was at least one good row available, we say that algorithm RANDOM made an error. Let $E_c$ be the event that there is at least one good row available when column $c$ comes in. For any column $c$

$$\begin{aligned} P_c(\text{error}) &= P_c(\text{bad decision}, E_c) \\ &= P_c(\text{bad decision}\,|E_c)P(E_c). \end{aligned}$$

Let column $c$ arrive at time $t$. By Lemma 2, we have that

$$P(E_c) = \frac{\binom{bx(t)-1}{y-1}}{\binom{bx(t)}{y}}.$$

Dropping the time index for convenience, we have that

$$P(E_c) = \frac{y}{x}. \tag{1}$$

Additionally, we have that

$$P_c(\text{error}|E_c) \geq \frac{y-b}{y} \tag{2}$$

since there can only be at most $b$ good rows. From (1) and (2), we have that

$$P_c(\text{error}) \geq \frac{y}{x}\frac{y-b}{y} = \frac{y-b}{x}. \tag{3}$$

We now bound the expected number of errors made by all columns of a particular type. Let the first column of type $k$ come in at time $t$. Therefore, $x(t) = k$ and let $y(t) = y$. Let $P_{\mathrm{err}}(k)$ be the probability that the first column of type $k$ makes an error. We claim that

$$P_{\mathrm{err}}(k) \geq \frac{y-b}{x}. \tag{4}$$

From (4), we have that

$$E[\Delta y] \leq -b - b\frac{y-2b}{bx}.$$

Therefore, for the algorithm RANDOM, we have that

$$\frac{E[\Delta y]}{E[\Delta x]} \geq b + \frac{y-b}{x}. \tag{5}$$

Consider a system such that

$$\frac{E[\Delta y]}{E[\Delta x]} = b + \frac{y-b}{x}. \tag{6}$$

We refer to this system as System B. Next, instead of solving (6) (a stochastic difference equation), as in [15] we solve (7) (an ODE). It is known from Kurtz's theorem [16] that the corresponding two solutions tend to each other as $n \to \infty$ (and with high probability)

$$\frac{dy}{dx} = b + \frac{y-b}{x}. \tag{7}$$

Let $f(y,x)$ be such that for the original system, System A

$$\frac{E[\Delta y]}{E[\Delta x]} = f(y,x) \tag{8}$$

such that $f(y,x)$ is an increasing function of $y$ for $y > b$. Note that $f(y,x)$ is an increasing function of $y$ for $y > b$ because the more options an incoming column has, the higher the probability of making an error, since there are always at most $b$ good rows. From (5), we know that $f(y,x) \geq b + \frac{y-b}{x}$. The solution to System A can be approximated by

$$\frac{dy}{dx} = f(y,x). \tag{9}$$

Since both systems start at the same point, i.e., $y = bn, x = n$, we have that for a given value of $x$, $y_A(x) \leq y_B(x)$. We will therefore use System B to get a bound on the performance of System A. We have that

$$\frac{dy}{dx} = b + \frac{y-b}{x}.$$

Solving this, we get that

$$y = cx + bx\log x + b.$$

We know that when $x = n$, $y = bn$, therefore we get that

$$c = b - b\log n + -\frac{b}{n}.$$

Therefore

$$y = bx + bx\log\frac{x}{n} - \frac{bx}{n} + b.$$

**Algorithm 4** ALG

Create $d_{\max}+t-1$ copies of each server $s_i \in S$.
Label the $j$th copy of $s_i$ as $s_i^{(j)}$.
For each $s_i^{(j)}$, pick a value $V(s_i^{(j)})$ i.i.d. uniform [0,1].
$S = \{s_i^{(j)} : s_i \in S, 1 \le j \le t + d_{\max} - 1\}$, $M = \phi$.
**for** each time-slot $u \le t$ **do**
  **for** arriving job $p$ characterized by $\{S_p, d_p\}$ **do**
    $E(p) = \{s_i \in S_p \text{ and } u \le j \le u + d_p - 1\}$.
    **if** $(C(p) := E(p) \cap S \setminus \mathrm{M} \neq \phi)$ **then**
      $s_{i*}^{(j^*)} = \arg\min_{x \in C(p)} V(x)$, $M = M \cup s_{i*}^{(j^*)}$.
    **end if**
  **end for**
**end for**

At $y = b$, we have that

$$x + x \log \frac{x}{n} - \frac{x}{n} = 0$$

therefore

$$x = e^{-1+\frac{1}{n}} n.$$

Therefore, the number of columns matched by System B is at most

$$bn - be^{-1+\frac{1}{n}} n + b.$$

Since System A matches fewer columns than System B, we have that the expected size of matching produced by RANDOM is at most

$$bn(1 - e^{-1+\frac{1}{n}} + 1/n).$$

■

*Proof (Proof of Theorem 1):* Follows from Lemmas 4 and 5. ■

## VI. Proof of Theorem 2

We propose an algorithm called INSERT RANKING as defined in Section III. In this section, we analyze the performance of this algorithm. Recall the set RANKING defined in Algorithm 1, which is updated after each job allocation and at the end of each time-slot. Each element in RANKING had a value associated with it, which is chosen independently according to the uniform distribution on $[0, 1]$.

Consider an arrival sequence $\mathbb{A}$ that is constructed by appending all arrivals in time-slots 1 to $t$ such that the relative order of arrival of jobs is maintained.

We define a new algorithm ALG, shown in Algorithm 4, whose performance on $\mathbb{A}$ is closely related to the performance of INSERT RANKING on all the arrivals from time-slot 1 to $t$.

*Lemma 6:* For any arrival process, ALG matches at least $1 - 1/e$ fraction of the jobs matched by the optimal offline algorithm which knows the entire arrival sequence a-priori.

*Proof:* ALG is identical to the RANKING algorithm proposed in [15]. The result therefore follows from the fact that the competitive ratio of the algorithm RANKING is $1 - 1/e$. ■

For the next lemma, we focus on arrival sequences such that no jobs arrive after time-slot $t$.

*Lemma 7:* For a given sample path (i.e., for the same values of $V(s_i^j)$ for all $s_i \in S$ and $1 \le j \le t + d_{\max} - 1$), for a given arrival process, the set of jobs matched by ALG is a subset of the set of jobs matched by INSERT RANKING.

*Proof:* Recall Step 15 of Algorithm 1 where matched server copies reinserted into the set of unmatched servers.

For a job $p$, the sets of eligible server copies that it can be matched to are denoted by $N(p)$ for INSERT RANKING and $C(p)$ for ALG.

We claim the following properties:

P1) When a job $p$ arrives, the set of jobs that arrived before $p$ and were matched by INSERT RANKING is a superset of the the set of jobs that arrived before $p$ and were matched by ALG.

P2) When a job $p$ arrives in time-slot $t$, the set unmatched server copies is a superset of server copies with indices $t \le j \le t + d_{\max} - 1$ not matched by ALG.

The two algorithms are identical in the first time-slot. Consider the first arrival $p$ in the second time-slot. Consider the sets of eligible server copies that it can be matched to ($N(p)$ for INSERT RANKING and $C(p)$ for ALG). Due to reinsertion, $N(p) \supseteq C(p)$. There are three possible cases.

I) INSERT RANKING matches $p$ to a reinserted server copy. In this case, P1 and P2 follow directly.

II) INSERT RANKING matches $p$ to a non-reinserted server copy. As before, P1 follows. If ALG also matches $p$ to the same server copy, P2 also follows. If not, since $N(p) \supseteq C(p)$, INSERT RANKING matches $p$ to an non-reinserted server copy $s_i^{(j)} \in$ RANKING with a lower value. Since ALG does not match $p$ to this server, we conclude that $s_i^{(j)}$ has already been matched by ALG to some other job and therefore P2 follows.

III) INSERT RANKING does not match $p$ to any server copy. Since $N(p) \supseteq C(p)$, this implies ALG also does not match $p$ to any server copy and P1 and P2 follow.

By induction, the proof follows for all subsequent arrivals. By P1, we conclude that the set of jobs matched by ALG is a subset of the set of jobs matched by INSERT RANKING. ■

*Proof (Proof of Theorem 2):* We first bound the quantity $\rho_t(\text{INSERT RANKING})$. By Lemmas 6 and 7, we conclude that

$$\rho_t(\text{INSERT RANKING}) \ge 1 - \frac{1}{e}$$

and therefore

$$\rho(\text{INSERT RANKING}) = \inf_t \rho_t(\text{INSERT RANKING}) \ge 1 - \frac{1}{e}.$$

■

## VII. Alternative Algorithms: Proofs of Theorems 3 and 4

From Sections V and VI, we conclude that INSERT RANKING is an optimal online algorithm. In this section, we try to understand if correlated randomness is necessary to achieve optimality or if just randomization is sufficient.

### A. RANDOMIZED JSQ

The first algorithm that we study is a Join the Shortest Queue algorithm that breaks ties between the shortest queues uniformly at random independent of all past choices. We refer to this algorithm as RANDOMIZED JSQ. The algorithm has been defined in Section III.

*Proof (Proof of Theorem 3):* To compute an upper bound on the competitive ratio of RANDOMIZED JSQ, we construct a bad arrival sequence and upper-bound the performance of RANDOMIZED JSQ by its performance for that particular arrival sequence. We construct the arrival sequence as follows. Consider a matrix such that $A(i,i) = 1$ for $1 \leq i \leq n$, $A(i,j) = 1$ if $j \geq n/2$, $i \leq n/2$, and $A(i,j) = 0$ otherwise. As in Section V, columns of $A$ represent jobs and rows represent servers. The jobs (columns) arrive from right to left. For example, for $n = 8$, this matrix is

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

We set the deadline of all jobs to be 1. Consider an arrival process $Arr$, where $Arr(t) = A$ for all $t$. At time $t$, all jobs that arrived before time-slot $t$ are no longer in the system. They are either served or dropped. Since each job has a service time of 1 time-slot, for this arrival process, the task of load balancing at time $t$ is equivalent to the task of finding an online matching. The proof is based the analysis for a similar arrival sequence presented in [19]. However, as the arrival sequence is different, the proof is structurally similar, but the detailed analysis differs.

We focus on the first $n/2$ servers. Let $x(c)$ denote the number of jobs remaining in the arrival sequence when the $(n-c+1)$th column arrives, i.e., $x(c) = n-c$, and $y(c)$ be a random variable denoting the number of servers among the first $n/2$ servers that have not been matched by the time the $(n-c+1)$th column arrives. Let

$$\begin{aligned} \Delta x &= x(c) - x(c+1) \\ \Delta y &= y(c) - y(c+1) \\ \Delta y &= -1 \text{ w.p. } \frac{y}{y+1} \end{aligned}$$

and 0 otherwise. Therefore, using Kurtz's Theorem [16], we approximate this system by the system given by

$$\frac{dy}{dx} = \frac{y}{y+1}.$$

Solving for $y$ with initial conditions $x(1) = n$, $y(1) = n/2$, we get that

$$x(c) = y(c) + \frac{n}{2} + \log \frac{2y(c)}{n}.$$

For $x(c) = n/2$, i.e., $c = n/2$, we have that

$$y(c) + \log \frac{2y(c)}{n} = 0.$$

Therefore, $y(n/2) < \log n$, i.e., there are at most $\log n$ servers among the first $n/2$ servers that have not been matched to a job so far and therefore can serve one of the remaining $n/2$ jobs. Since the final $n/2$ jobs can only be served by the first $n/2$ servers, at most $y(n/2)$ of them will be matched to a server. Therefore, we have that

$$\rho_t(\text{RANDOMIZED JSQ}) \leq \frac{t(n/2 + \log n)}{tn}.$$

As $n \to \infty$

$$\rho_t(\text{RANDOMIZED JSQ}) \leq \frac{1}{2}.$$

Therefore, we have that

$$\rho(\text{RANDOMIZED JSQ}) \leq \frac{1}{2}.$$

■

### B. RANDOMIZED P-JSQ

Since INSERT RANKING is not a join the short queue algorithm, but is biased toward joining shorter queues, we analyze the performance of an algorithm which we call RANDOMIZED P-JSQ. Like INSERT RANKING, this algorithm is also biased toward joining shorter queues. The algorithm has been defined in Section III.

*Proof (Proof of Theorem 4):* To compute an upper bound on the competitive ratio of RANDOMIZED P-JSQ, we construct a bad arrival sequence and upper-bound the performance of RANDOMIZED P-JSQ by its performance for that particular arrival sequence. We construct the arrival sequence as follows.

Construct a matrix $B$ such that $B(i,i) = 1$ and $B(i,j) = 1$ for $i \leq n/2$ and $j > n/2$. For example, for $n = 4$

$$B = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We then make $d_{\max}$ copies for each column to get the arrival matrix $A$. For $d_{\max} = 2$

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

We set the deadline of all jobs to be $d_{\max}$. Like in Section V, consider an arrival process $Arr$, where $Arr(1) = A$ and $Arr(t) = \phi$ for $t \neq 1$. We construct a matrix $S$ from $A$ by creating $d_{\max}$ copies of each server and try to find a matching for $S$. In this case where $d_{\max} = 2$

$$S = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

We first classify the columns of the matrix $S$. Observe that columns $d_{\max}k+1$ to $d_{\max}k+d_{\max}$ for all integral values of $k$ such that $1 \leq k \leq n$ are identical. We refer to columns $d_{\max}k+1$ to $d_{\max}k+d_{\max}$ as columns of type $k$. Let $x(t)$ be the number of types of columns remaining at time $t$ and $y(t)$ be the number of the top $d_{\max}n/2$ rows eligible at time $t$. Let $\Delta x = x(d_{\max}t+d_{\max}) - x(d_{\max}t) = -1$ and $\Delta y = y(d_{\max}t+d_{\max}) - y(d_{\max}t)$. For any column $j$ such that $d_{\max}k+1 \leq j \leq d_{\max}k+d_{\max}$, we refer to the rows $d_{\max}k+1$ to $d_{\max}k+d_{\max}$ as the good rows and the others as bad rows.

Let the first column of type $k$ come in at time $t$. The quantity $y(t)$ decreases by at most 1 after each column gets matched since all columns of type $k$ have the same set of eligible rows. Therefore, $y > y(t) - d_{\max}$ for all columns of type $k$. For any column of type $k$, the probability of making an error is at least $\frac{y}{y+d_{\max}} \geq \frac{y(t)-d_{\max}}{y(t)+d_{\max}}$. Therefore

$$E[\Delta y] \leq -d_{\max}\frac{y(t)-d_{\max}}{y(t)+d_{\max}}.$$

Since $\Delta x = -1$, we have that

$$\frac{E[\Delta y]}{E[\Delta x]} \geq d_{\max}\frac{y(t)-d_{\max}}{y(t)+d_{\max}}.$$

Instead of solving the stochastic difference equation, we use Kurtz's theorem [16] and bound its solution by the solution to

$$\frac{dy}{dx} = d_{\max}\frac{y-d_{\max}}{y+d_{\max}}$$

with the initial condition $x(1) = n$, $y(1) = d_{\max}n/2$. We then compute the value of $y$ for $x = n/2$. We get that when $x = n/2$, $y \leq 2d_{\max}\log n$. Therefore, we have that

$$\rho_t(\text{RANDOMIZED P} - \text{JSQ}) \leq \frac{d_{\max}n/2 + 2d_{\max}\log n}{d_{\max}n}.$$

As $n \to \infty$

$$\rho_t(\text{RANDOMIZED P} - \text{JSQ}) \leq \frac{1}{2}.$$

Therefore, we have that

$$\rho(\text{RANDOMIZED P} - \text{JSQ}) \leq \frac{1}{2}.$$

■

## VIII. Simulations

Since all our results are in the limiting case of the number of servers, $n \to \infty$, we present simulation results for finite values of $n$. Fig. 7 shows the performance of INSERT RANKING for the arrival sequence described in Section V. Fig. 8 compares the performance of INSERT RANKING and RANDOMIZED JSQ for the arrival sequence described in Section VII-A. Fig. 9 compares the performance of INSERT RANKING and RANDOMIZED P-JSQ for the arrival sequence described in Section VII-B. In all the plots, we plot the average value of the fraction of jobs served, averaged over 1000 instances of the algorithms.

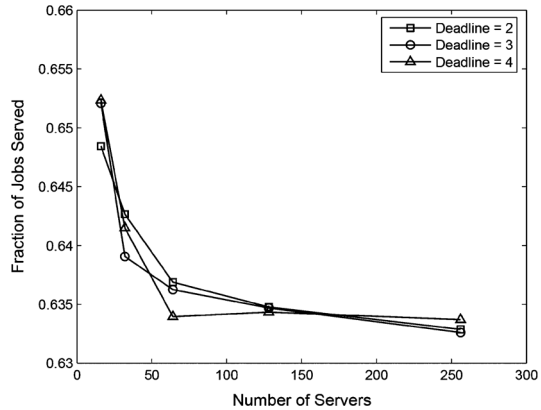



Fig. 7. INSERT RANKING on the matrix A in Section V.

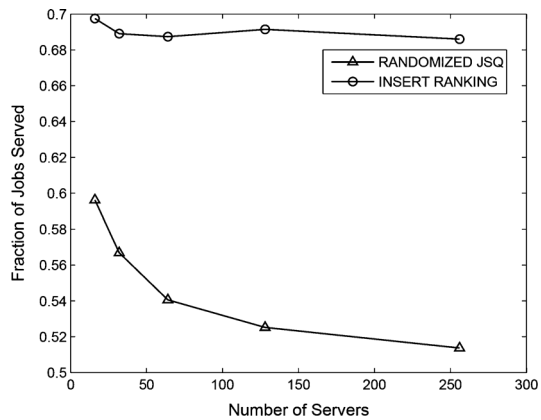



Fig. 8. Comparison of INSERT RANKING and RANDOMIZED JSQ.

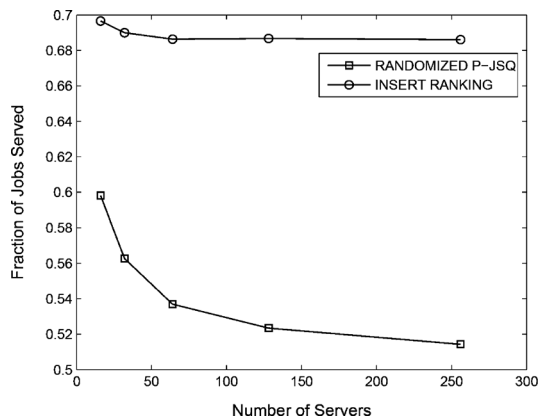



Fig. 9. Comparison of INSERT RANKING and RANDOMIZED P-JSQ.

## IX. Summary and Discussion

We study online load balancing under graph constraints in the adversarial setting. First, we design "bad" arrival patterns and use them to upper-bound the performance any online load balancing algorithm. Next, we propose an algorithm called INSERT RANKING that uses correlated randomness for load balancing, and prove that it an optimal online load balancing algorithm. The main message of this paper is that correlated randomness is important because we show that INSERT RANKING outperforms algorithms that make (the more natural) uncorrelated/independent random choices for load balancing.

Graph constrained load balancing problems are of interest for content farms where every content piece cannot be stored on every server. It would also be interesting to look at joint problem of predicting/inferring popularity, placing popular items on servers in content farms, and serving the resulting demand.

## APPENDIX

We now consider the setting where jobs might need more than one time-slot of service. The goal is to understand if correlated randomness is critical for good performance even in the setting. We first define a class of online algorithms.

*Definition 1: Greedy Online Algorithms:* An online algorithm belongs to this class if when a job comes up for allocation, given the current state of the system, if one or more servers can serve the job before its deadline elapses, the job is not dropped by the algorithm.

This is a natural class policies to consider because in the setting where future arrivals are arbitrary, there is no clear benefit in dropping a job that this system is capable of serving. Note that INSERT RANKING, RANDOMIZED JSQ, and RAMDOMIZED P-JSQ belong to this class of policies.

We show that in the setting where jobs can require more than one time-slot of service, the competitive ratio of greedy online algorithms can be arbitrarily bad, i.e., for any greedy online algorithm, $\rho(alg) \leq c$ for all values of $c > 0$.

Consider the following arrival sequence.

1) At the beginning of time-slot 1, $n$ jobs arrive in a sequence, each of which can be served by all the $n$ servers. Each of these jobs needs $k$ time-slots of service and have to be served in the first $k$ time-slots.
2) Following the first $n$ jobs, $kn$ jobs arrive, each of which can be served by all the $n$ servers. These jobs need 1 time-slot of service and have to be served in the first $k$ time-slots.

For this arrival sequence, any greedy online algorithm will serve the first $n$ jobs and, therefore, will have to drop the remaining $kn$ jobs. The optimal allocation for this arrival process is to drop the first $n$ jobs and serve the remaining $kn$ jobs by dividing them into groups of $k$ and allocating exactly one group to each of the $n$ servers. Therefore, the competitive ratio of any greedy online algorithm is at most $1/k$.

To avoid such "bad" arrival processes, for which all greedy algorithms perform poorly, we limit ourselves to the setting where all jobs need the same amount of service, i.e., all jobs need $k$ time-slots for $k \geq 1$, and have to be served in a nonpreemptive manner.

We adapt the INSERT RANKING algorithm for the setting where all jobs needs $k$ units of service. We refer to the new algorithm as the INSERT RANKING: Multi-slot (INSERT RANKING-M) algorithm. At a high level, INSERT RANKING-M clubs time-slots $(c-1)k+1$ to $ck$ into a single block for $c = 1, 2, \ldots$, and each job is allocated to a (server, block) pair that can serve it successfully, i.e., the server is equipped to serve the job, and the block of time is such that it begins after the arrival of the job, and ends before its deadline elapses. We first describe the INSERT RANKING-M algorithm in words followed by a formal definition.

In the first time-slot:

A. INITIALIZE: Generate $\lceil \frac{d_{\max}}{k} \rceil$ labeled copies of each server with labels $1, 2, \ldots, \lceil \frac{d_{\max}}{k} \rceil$. Let $s_i^{(j)}$ denote the block of $s_i \in S$ labeled $j$, which corresponds to time-slots $(j-1)k+1$ to $jk$ on server $s_i$. Each labeled server copy $s_i^{(j)}$ has a value $V_i^{(j)}$ associated with it, chosen independently according to the uniform distribution on [0, 1].
B. ONLINE ALLOCATION: All jobs arriving in time-slot 1 are matched to server copies in an online manner as follows: For each incoming job:
   1. Create the neighborhood of a job $p$, defined as all server copies of each $s_i$ in its server subset with label $j$ such that $j \leq \frac{d_p}{k}$ (the label $j$ of a server copy indicates that a job assigned to that server copy will be served in time-slot $(j-1)k+1$ to $jk$). Therefore, each job $p$ defined by its server subset and deadline $\{S_p, d_p\}$ arriving at time 1 can be served by each $s \in S_p$ in time-slots $1, \ldots, \lfloor \frac{d_p}{k} \rfloor$).
   2. Match each incoming job to that unmatched server copy in its neighborhood that has the lowest value $V$.
   3. Remove matched server copy.
C. SCHEDULE: At the end of job allocation for time-slot 1, all matched jobs are inserted in the queues of the corresponding servers. Each server $s_i \in S$ serves the job which was matched to the server copy labeled 1.

In every *subsequent* time-slot $t(>1)$,

A. UPDATE STATE:
   1. Remove all server copies labeled $\frac{t-1+k}{k}$.
   2. If $((t-1) \bmod k = 0)$, add one new copy of each server with label $\lceil \frac{d_{\max}}{k} \rceil + \frac{t-1}{k}$ to the set of unmatched servers. Each new server copy $S_i^{(\lceil \frac{d_{\max}}{k} \rceil + \frac{t-1}{k})}$ has a value $V_i^{(\lceil \frac{d_{\max}}{k} \rceil + \frac{t-1}{k})}$ associated with it, chosen independently according to the uniform distribution on [0, 1].
B. ONLINE ALLOCATION: All jobs arriving in time-slot $t$ are matched to server copies in an online manner as follows: The neighborhood of a job $p$ is defined as all server copies of each $s_i$ in its server subset with label not more than $\frac{t-1+d_p}{k}$ and less than $\frac{t-1+k}{k}$. As mentioned before, the label $j$ of a server copy indicates that a job assigned to that server copy will be served in time-slot $(j-1)k+1$ to $jk$. Therefore, $j \geq \frac{t-1+k}{k}$ ensures that $t \leq (j-1)k+1$, i.e., service starts after the arrival of the job, and $j \leq \frac{t-1+d_p}{k}$ ensures that service ends before the deadline of the job elapses.
   Each incoming job is matched to that unmatched server copy in its neighborhood that has the lowest value $V$, and the matched server copy is removed.
C. SCHEDULE: At the end of job allocation for time-slot $t$, all matched jobs are inserted in the queues of the corresponding servers. Each server $s_i \in S$ serves that job in its queue matched to the copy labeled $\lfloor (t-1)/k \rfloor + 1$.

Refer to Fig. 5 for a formal definition of the algorithm.

**Algorithm 5** INSERT RANKING: Multi-Slot

INITIALIZE: $t = 1$, $\{s_i^{(j)} : j = 1, \ldots, \lceil \frac{d_{\max}}{k} \rceil$ and $s_i \in S\}$ and a real number $V_i^{(j)}$ for each $s_i^{(j)}$ chosen uniformly from $[0, 1]$.
ONLINE ALLOCATION:
**for** arriving job $p = \{S_p, d_p\}$ in time-slot $t$, **do**
  make neighborhood: $N(p) = \{s_i^{(j)} : s_i \in S_p$ and $\frac{t-1+k}{k} \leq j \leq \frac{t-1+d_p}{k}\}$.
  match $p$ to currently unmatched $s_{i^*}^{(j^*)} \in N(p)$ that has the lowest $V$.
**end for**
SCHEDULE:
**for** each server $s_i$, **do**
  add all jobs matched to copies of server $i$ in time-slot $t$ to the queue of server $s_i$.
serve currently job matched to $s_i^{(\lfloor (t-1)/k \rfloor + 1)}$.
**end for**
UPDATE STATE:
$t = t + 1$.
**for** each server $s_i$, **do**
  remove all matched copies of $s_i$.
  remove $s_i^{(\frac{t-1+k}{k})}$.
  If $((t-1) \bmod k = 0)$, add $s_i^{(\lceil \frac{d_{\max}}{k} \rceil + \frac{t-1}{k})}$, choose a real number $V_i^{(\lceil \frac{d_{\max}}{k} \rceil + \frac{t-1}{k})}$ uniformly from $[0, 1]$.
**end for**
Goto 2.

Next, we analyze the performance of INSERT RANKING: Multi-slot, shown in Algorithm 5, for a restricted set of arrival processes.

*Theorem 5:* Let the sequence of arrivals up to time $t$ be denoted by $A(t)$. For this arrival sequence, let $S_{\text{opt}}(A(t))$ be the number of jobs served in the offline optimal allocation, and $S_{\text{IR}-\text{M}}(A(t))$ be the number of jobs served by INSERT RANKING-M. Then, if $A(t)$ is such that the offline optimal allocation for $A(t)$ is such that all $n$ servers are busy in time-slots $1, 2, \ldots, t-1$

$$\frac{E[S_{\text{IR}-\text{M}}(A(t))]}{S_{\text{opt}}(A(t))} \geq 1 - \frac{1}{e}.$$

*Proof:* We claim that the following statement is true: Consider a job $p$ that arrives at the beginning of time-slot $t$.

1) If $((t-1) \bmod k) \neq 0$, in the offline optimal allocation, $p$ is either dropped, or starts service $c_p k + k - ((t-1) \bmod k)$ time-slots after its arrival, where $c_p$ is a nonnegative integer.
2) If $((t-1) \bmod k) = 0$, $p$ is either dropped, or starts service $c_p k$ time-slots after its arrival, where $c_p$ is a nonnegative integer.

We prove the statement by induction over time-slots. Since the optimal allocation is such that all servers are busy in time-slot 1, we have that each server starts serving a job that arrives at the beginning of time-slot 1 in time-slot 1. Since each job requires exactly $k$ time-slots of service, no new jobs start service before the end of the $k$th time-slot. Moving to the next block, since all servers are busy in the $(k+1)$th time-slot, each of them starts serving a new job in the $(k+1)$th time-slot, and these jobs arrived either before or at the beginning of time-slot $k+1$. Since each job requires exactly $k$ time-slots of service, no new jobs start service before the end of the $(2k-1)$th time-slot, thus completing the proof of the second block. The proof follows by the same argument for all $t$.

By definition, if job $p$ arrives at the beginning of time-slot $t$ such that $((t-1) \bmod k) \neq 0$, the INSERT RANKING-M algorithm either drops the job, or starts serving it $c_p k + k - ((t-1) \bmod k)$ time-slots after its arrival, for some nonnegative integer $c_p$ and if $((t-1) \bmod k) = 0$, $p$ is either dropped, or starts service $c_p k$ time-slots after its arrival, where $c_p$ is a nonnegative integer. This imposes a restriction on the matching obtained by INSERT RANKING-M, which is not fundamental to the underlying load balancing problem. However, as shown above, the optimal allocation also satisfies these conditions.

Consider the bipartite graph between server copies corresponding to blocks, i.e., $s_i^{(j)}$ and all jobs that arrive in the first $t$ time-slots. Under the INSERT RANKING-M algorithm, job $p$ arriving at time $t$ has an edge and therefore can be matched to the copies of all servers equipped to server the job with indices $\frac{t-1+k}{k}$ to $\frac{t-1+d_p}{k}$. Since in the offline optimal allocation, each job is either dropped on matched to one of these blocks of server copies, we have that the maximal matching in this bipartite graph, denoted by $M(A(t))$ is such that $|M(A(t))| = S_{opt}(A(t))$.

Next, by [15, Theorem 1], we have that

$$E[S_{\text{IR}-\text{M}}(A(t))] \geq (1 - 1/e)|M(A(t))|$$

and the result follows. ∎

It is straightforward to construct examples of arrival sequence (see Section VII) that satisfy the assumptions of Theorem 5 such that

$$\frac{E[S_{\text{RANDOMIZED JSQ}}(A(t))]}{S_{opt}(A(t))} \leq 1/2 + o(1)$$

and

$$\frac{E[S_{\text{RANDOMIZED P}-\text{JSQ}}(A(t))]}{S_{opt}(A(t))} \leq 1/2 + o(1).$$

We thus conclude that in the restricted setting of multi-slot jobs with uniform service requirements, for a restricted set of arrival processes, correlated randomness is critical for good performance.

## REFERENCES


[1] G. Aggarwal, G. Goel, C. Karande, and A. Mehta, "Online vertex-weighted bipartite matching and single-bid budgeted allocations," in *Proc. 22nd Annu. ACM-SIAM Symp. Discrete Algor.*, 2011, pp. 1253–1264.

[2] P. Baptiste, P. Brucker, S. Knust, and V. Timkovsky, "Ten notes on equal-processing-time scheduling," *4OR*, vol. 2, pp. 111–127, 2004.

[3] B. Birnbaum and C. Mathieu, "On-line bipartite matching made simple," *SIGACT News*, vol. 39, no. 1, pp. 80–87, Mar. 2008.

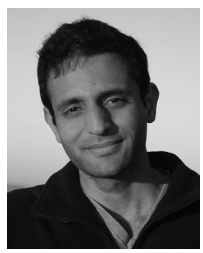
[4] S. Chen, L. Tong, and T. He, "Optimal deadline scheduling with commitment," in *Proc. 49th Annu. Allerton*, Sep. 2011, pp. 111–118.

[5] M. Chrobak, W. Jawor, J. Sgall, and T. Tich, "Online scheduling of equal-length jobs: Randomization and restarts help," in *Automata, Languages and Programming*, J. Daz, J. Karhumki, A. Lepist, and D. Sannella, Eds. Berlin/Heidelberg, Germany: Springer, 2004, vol. 3142, Lecture Notes in Computer Science, pp. 145–156.

[6] J. Ding and G. Zhang, , S.-W. Cheng and C. Poon, Eds., "Online scheduling with hard deadlines on parallel machines," in *Algorithmic Aspects in Information and Management*. Berlin/Heidelberg, Germany: Springer, 2006, vol. 4041, Lecture Notes in Computer Science, pp. 32–42.
[7] A. Dua and N. Bambos, "Downlink wireless packet scheduling with deadlines," *IEEE Trans. Mobile Comput.*, vol. 6, no. 12, pp. 1410–1425, Dec. 2007.
[8] T. Ebenlendr and J. Sgall, , E. Bampis and M. Skutella, Eds., "A lower bound for scheduling of unit jobs with immediate decision on parallel machines," in *Approximation and Online Algorithms*. Berlin/Heidelberg, Germany: Springer, 2009, vol. 5426, Lecture Notes in Computer Science, pp. 43–52.
[9] G. Goel and A. Mehta, "Online budgeted matching in random input models with applications to adwords," in *Proc. 19th Annu. ACM-SIAM SODA*, Philadelphia, PA, USA, 2008, pp. 982–991.
[10] S. A. Goldman, J. Parwatikar, and S. Suri, "Online scheduling with hard deadlines," *J. Algor.*, vol. 34, no. 2, pp. 370–389, 2000.
[11] M. H. Goldwasser and M. Pedigo, "Online nonpreemptive scheduling of equal-length jobs on two identical machines," *Trans. Algor.*, vol. 5, no. 1, pp. 2:1–2:18, Dec. 2008.
[12] I.-H. Hou and P. R. Kumar, "Queueing systems with hard delay constraints: A framework for real-time communication over unreliable wireless channels," *Queueing Syst. Theory Appl.*, vol. 71, no. 1-2, pp. 151–177, June 2012.
[13] J. J. Jaramillo, R. Srikant, and L. Ying, "Scheduling for optimal rate allocation in ad hoc networks with heterogeneous delay constraints," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 5, pp. 979–987, May 2011.
[14] B. Kalyanasundaram and K. R. Pruhs, "An optimal deterministic algorithm for online b-matching," *Theoret. Comput. Sci.*, vol. 233, p. 2000, 2000.
[15] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for on-line bipartite matching," in *Proc. 22nd Annu. ACM Symp. Theory Comput.*, Baltimore, MD, USA, May 1990, pp. 352–358.
[16] T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *J. Appl. Probab.*, vol. 7, no. 1, pp. 49–58, 1970.
[17] J.-H. Lee, "Online deadline scheduling: multiple machines and randomization," in *Proc. 15th Annu. ACM SPAA*, New York, NY, USA, 2003, pp. 19–23.
[18] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, "Adwords and generalized on-line matching," in *Proc. IEEE FOCS*, 2005, pp. 264–273.
[19] S. Moharir and S. Sanghavi, "Online load balancing and correlated randomness," in *Proc. 50th Annu. Allerton*, 2012, pp. 746–753.
[20] S. Moharir, S. Sanghavi, and S. Shakkottai, "Load balancing with deadlines and graph constraints," presented at the 51st Annu. Allerton Conf., 2013.
[21] S. Moharir, S. Sanghavi, and S. Shakkottai, "Online load balancing under graph constraints," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst.*, 2013, pp. 363–364.
[22] L. Qingwen, W. Xin, and G. B. Giannakis, "A cross-layer scheduling algorithm with qos support in wireless networks," *IEEE Trans. Veh. Technol.*, vol. 55, no. 3, pp. 839–847, May 2006.
[23] "Netflix," [Online]. Available: http://www.netflix.com
[24] V. Raghunathan, V. Borkar, M. Cao, and P. R. Kumar, "Index policies for real-time multicast scheduling for wireless broadcast systems," in *Proc. 27th IEEE INFOCOM*, Apr. 2008, pp. 1570–1578.
[25] S. Shakkottai and R. Srikant, "Scheduling real-time traffic with deadlines over a wireless channel," *Wireless Netw.*, vol. 8, no. 1, pp. 13–26, Jan. 2002.
[26] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.
[27] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Trans. Inf. Theory*, vol. 39, no. 2, pp. 466–478, Mar. 1993.
[28] A. C. Yao, "Probabilistic computations: Toward a unified measure of complexity," in *Proc. 18th Annu. Symp. Found. Comput. Sci.*, Oct. 31–Nov. 2, 1977, pp. 222–227.
[29] "YouTube," [Online]. Available: http://www.youtube.com

**Sharayu Moharir** received the B.Tech. degree in electrical engineering and M.Tech. degree in communication and signal processing from the Indian Institute of Technology, Bombay, Mumbai, India, in 2009, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 2014.

She is a Visiting Fellow with the School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India. Her research interests include algorithms and performance analysis for wireless networks and content delivery networks.



**Sujay Sanghavi** received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay. Mumbai, India, in 2000, and the M.S. degree in electrical and computer engineering (ECE), M.S. degree in mathematics, and Ph.D. degree in ECE from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2002, 2005, and 2006, respectively.

He is an Associate Professor in electrical and computer engineering with the University of Texas at Austin, Austin, TX, USA. He has been a Visiting Scientist with Google Research and Qualcomm. His research lies in the areas of statistical inference, optimization, algorithms, and networks.

Dr. Sanghavi has an NSF Career Award and a Young Investigator Award from the Department of Defense.

**Sanjay Shakkottai** (M'02–SM'11–F'14) received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2002.

He is with The University of Texas at Austin, Austin, TX, USA, where he is currently a Professor with the Department of Electrical and Computer Engineering. His current research interests include network architectures, algorithms and performance analysis for wireless networks, and learning and inference over social networks.

Prof. Shakkottai received the NSF CAREER Award in 2004.