Online Load Balancing and Correlated Randomness

Sharayu Moharir, Sujay Sanghavi
Wireless Networking and Communications Group (WNCG)
Department of Electrical & Computer Engineering
The University of Texas at Austin
Austin, TX 78712, USA
Email: sharayu.moharir@gmail.com, sanghavi@mail.utexas.edu

***Abstract*—This paper looks at online load balancing, in a setting where each job can only be served by a subset of the servers. The subsets are revealed only on arrival, and can be arbitrary. The cost of an allocation is the sum of cost for each server, which in turn is a convex increasing function of the number of jobs allocated to it. There are no departures.**

A natural class of policies are those which always put a job into one of its servers that has the least load at the time of arrival. However, it turns out that not all (randomized) ways of breaking ties is the same. We propose an algorithm - TIERED RANKING - that breaks ties in a very particular, correlated random way; it is inspired by the online matching work of Karp, Vazirani and Vazirani. We show that it is optimal (in terms of competitive ratio) in the above class, for *all* convex cost functions that grow slower than e^x (which includes all ℓ_p norms). We also prove it strictly outperforms any deterministic algorithm; simulations show that it also visibly outperforms the naive randomized algorithm, that breaks ties among lowest-loaded servers uniformly at random.

I. Introduction

We consider the online problem of allocating jobs to a group of servers. We study the case where each job can only be served by a subset of the servers. The jobs arrive in a sequence and there are no departures. An example of such a system is a data center where there is a bank of servers, and several pieces of content. Each piece of content has been replicated and put on a few of the servers. As jobs come in and make demands for a particular content piece, they need to be allocated to one of the servers having that content.

We define the cost of an allocation as the sum of the cost borne by the individual servers. We focus on the cases where the cost function for each server is a convex function of the number of jobs allocated to that server. Many such cost functions have been studied in literature, for example, the ℓ_p norm for $p < \infty$ of the load vector was studied in [4], the ℓ_∞ norm was studied in [2] and is popularly known as the make-span problem. The **main challenge** is to design an online algorithm that is universal in the sense that it provides the optimal performance for *any* convex cost function.

In this work, the performance of an online algorithm is compared with the performance of the optimal offline algorithm which knows the sequence of job arrivals in advance. We characterize the performance of an online algorithm by its competitive ratio which is the ratio of the cost of the online algorithm to the cost of the optimal offline algorithm, maximized over all input sequences.

We are interested in a class of algorithms which we call the *Load the Lowest Queue (LLQ) algorithms*. The LLQ class of algorithms include all deterministic and randomized algorithms where an incoming job is *always* allocated to one of the currently lowest loaded servers that can serve it.

A. Contributions

Our main contribution is the TIERED RANKING algorithm and its performance analysis. The TIERED RANKING algorithm is an LLQ algorithm, but it breaks ties between several lowest-loaded servers in a very particular correlated-random way: first a random permutation is chosen for every load-level, then, every tie at a particular level is broken according to that permutation. We show

- An upper bound (i.e. achievable) on the competitive ratio of TIERED RANKING (Theorem 1).
- A lower (i.e. outer) bound on the competitive ratio of *any* randomized LLQ algorithm for convex cost functions that grow slower than e^x. This shows the optimality of TIERED RANKING in this class (Theorem 2).
- A lower bound for all deterministic algorithms, showing them to be strictly worse than randomized algorithms (Theorem 3).

We use these results to prove the optimality of the TIERED RANKING algorithm for load balancing in the

ℓ_p norm for *any* $p < \infty$ which has been of interest in [4], [3]. Our results improve the known competitive ratio bounds for online load balancing in the ℓ_p norm [4] and we compare the results in Section VII. Finally we show via simulations that TIERED RANKING visibly outperforms the simple random LLQ algorithm.

B. Related Work

Our tie-breaking rule is inspired by the work on online matching. We now review this and previous work on online allocation.

1) Online Matching: The problem of online matching in bipartite graphs has been studied in [5] by Karp, Vazirani and Vazirani. This work provides tight bounds on the competitive ratio. An online algorithm called RANKING was proposed in [5] and it was shown that the competitive ratio of RANKING is greater than $1-\frac{1}{e}$. It was also shown that the competitive ratio for any online algorithm is less than $1-\frac{1}{e}+o(1)$.

Various extensions to weighted graphs, stochastic arrivals etc. can be found in the references of [1].

2) Online Load Balancing: [2] looks at the make-span problem, i.e., minimizing the largest load among servers. The randomized algorithm proposed in [2] is called AR. The algorithm was shown to be optimal in the class of all randomized algorithms for this objective function.

[3] looks at the ℓ_p norm of the load vector for the GREEDY algorithm (load the lowest loaded server, breaks ties uniformly at random). For the Euclidean norm, they proved that competitive ratio is at most 2.41.

An improvement on the results of [3] was made in [4] which looked at the problem of scheduling over unrelated machines. The term unrelated machines means that each job takes a different service time on each of the machines with no correlation between jobs or machines. This is the first analysis of randomized algorithms for ℓ_p norms for p = 2,3.., 137. The proposed randomized algorithm is called BALANCE.

II. Setting

There is a set U of servers and jobs arrive sequentially. Each job can be served by any one of the servers in a subset of servers. The appearance of a job is equivalent to the disclosure of the set of servers which can serve it. We consider a convex cost function f such that the cost of allocating n_u jobs to server u is $f(n_u)$. We assume that $f(0)=0$. The goal is to design an online allocation algorithm which minimizes the total cost C where

$$C = \sum_{u \in U} f(n_u).$$

The online algorithm must assign each job to a server when it arrives. We consider the adversarial setting where we assume the presence of an oblivious adversary, i.e., one who knows the online allocation algorithm, but is unaware of the results of the coin flips during its execution. We restrict ourselves to arrival sequences such that it is possible for the optimal algorithm to assign exactly one job to each server. Let the set of such arrival sequences be $\mathcal{A}$. Thus in the set $\mathcal{A}$, the number of jobs is equal to the number of servers = n. This restriction is required for the proofs in this paper, but, we drop this restriction in the simulations. The performance of the algorithm is measured in terms of the competitive ratio ρ, where

$$\rho(\text{algorithm}) = \max_{A \in \mathcal{A}} \left(\frac{C_{algorithm}\ (\text{A})}{C_{optimal}(\text{A})} \right).$$

Note that $C_{optimal}(A) = nf(1)$ by our assumption on A. $C_{algorithm}(A)$ is the expected cost of the online algorithm on A. By definition, this ratio is always >1. The goal is to design an algorithm which minimizes this quantity.

III. Our Algorithm

Algorithm 1 TIERED RANKING

```
1: Choose a uniformly random sequence of permutations $\pi_i$, $i \geq 0$.
2: Initialize Load($j$)=0 for all servers $j$.
3: for arriving job $p$ with server set $S_p$ do
4:     From its set $S_p$, find the subset $L_p \subset S_p$ of the lowest loaded servers.
       $L_p = \{j \in S_p | \text{Load}(j) \leq \text{Load}(k), \forall k \in S_p\}$
       Let $l$ be the load in the servers in $L_p$.
5:     Allocate $p$ to $j^* \in L_p$ such that
       $\pi_l(j^*) \leq \pi_l(j)$ for all $j \in L_p$.
6:     Increment Load($j^*$) $\leftarrow$ Load($j^*$) + 1
7: end for
```

Remark: This algorithm was studied in [2] where its performance was studied for the makespan problem. Our proofs use the following equivalent visualization of the allocation process. Construct a sequence of copies of the servers in set U and name them U_1, U_2, ... We will refer to all servers in U_i as servers at level i. Let π_{i-1} induce priorities on U_i. In this larger set of servers, we allocate at most 1 job to each server. When a job arrives, find the smallest i such that the job can be served by a server

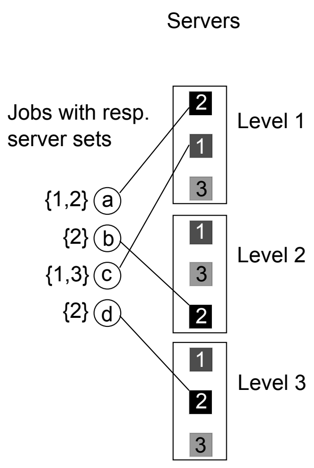

Figure 1. *TIERED RANKING*

in U_i. Assign the job to that available node in U_i which can serve this job and has the highest priority according to π_{i-1}. An illustrative example with 3 servers can be seen in Figure 1.

Consider an arrival sequence $A \in \mathcal{A}$ on which algorithm TIERED RANKING is executed. Let P_1 be the set of jobs allotted to servers in U_1 during this execution. Consider now a new arrival process $A' = A \setminus P_1$ obtained from A by removing the jobs in P_1. Now the algorithm TIERED RANKING is executed on A' with permutations $\pi'_i = \pi_{i+1}$, and with the same order of job arrivals as in A. The allocation that TIERED RANKING produces in U'_1 is identical to the allocation in U_2 of the original execution [2].

Lemma 1. *Let the total number of job arrivals in A be n and let N_i be the number of jobs that do not get allocated to servers in U_1, ... U_{i-1} by the TIERED RANKING algorithm. Then,*

$$E[N_i] \leq n\left(\frac{1}{e}\right)^{i-1}.$$

Proof:
We prove the property by induction. The statement clearly holds for $i = 1$. Assuming that the statement holds for $i = k$, we have that,

$$E[N_k] \leq n\left(\frac{1}{e}\right)^{k-1}.$$

Let M_k denote the jobs allocated to servers at level k by the TIERED RANKING algorithm. This allocation is equivalent to a allocation by the TIERED RANKING algorithm with $\pi_0 = \pi_k$ with the arrival process being $A \setminus M_1 ... \setminus M_{k-1}$. Using Theorem 1 in [5], we have that,

$$E[M_k] \geq \left(1 - \frac{1}{e}\right)E[N_k].$$

Therefore,

$$\begin{aligned} E[N_{k+1}] &= E[N_k] - E[M_k] \\ &\leq n\left(\frac{1}{e}\right)^{k-1} - E[M_k] \\ &\leq n\left(\frac{1}{e}\right)^{k-1} - n\left(\frac{1}{e}\right)^{k-1}\left(1 - \frac{1}{e}\right) \\ &= n\left(\frac{1}{e}\right)^{k}, \end{aligned}$$

which completes the proof by induction.

■

By Theorems 4.2 and 4.3 in [2], we have that for the ℓ_∞ norm i.e. the makespan, the TIERED RANKING algorithm has an expected competitive ratio which is at most $\log n + 1$ which is optimal in the class of all randomized online algorithms.

IV. Upper Bound for TIERED RANKING for convex cost functions

Theorem 1. *Let $|U| = n$ and let $A \in \mathcal{A}$ be an arrival sequence of n jobs as defined in Section II. Define*

$$S_k^{(f)} = f(k+1) - 2f(k) + f(k-1).$$

The expected competitive ratio of Algorithm TIERED RANKING for the cost function f is at most

$$1 + \sum_{k=1}^{n-1} \left(\frac{1}{e}\right)^k \frac{S_k^{(f)}}{f(1)}.$$

Proof:
For this proof, recall the visualization outlined in Section III. All servers in U_i are assigned a cost of $C_i^{(f)}$ units where

$$C_i^{(f)} = f(i) - f(i-1). \tag{1}$$

Note that with this definition of $C_i^{(f)}$, if only the first k copies of a node $u \in U$ are allocated a job, their combined contribution to the cost of the algorithm is $f(k)$.

Recall from the definition of N_i that the number of jobs allocated to a server in U_i is $N_i - N_{i+1}$. Hence the total cost C of an allocation is

$$\begin{aligned} C &= \sum_{k=1}^{n} (N_k - N_{k+1})C_k^{(f)} \\ &= nf(1) + \sum_{k=2}^{n} N_k(C_{k+1}^{(f)} - C_k^{(f)}), \end{aligned}$$

which can be rewritten as

$$C = nf(1) + \sum_{k=2}^{n} N_k S_{k-1}^{(f)}.$$

If f is a convex function, then,

$$S_k^{(f)} = f(k+1) - 2f(k) + f(k-1) \geq 0.$$

Using the linearity of expectation, Lemma 1 and the fact that $S_k^{(f)} \geq 0$ for all $k \geq 1$,

$$\begin{aligned} E[C] &= nf(1) + \sum_{k=2}^{n} E[N_k] S_{k-1}^{(f)} \\ &\leq nf(1) + n \sum_{k=1}^{n-1} \left(\frac{1}{e}\right)^k S_k^{(f)}. \end{aligned}$$

Since the cost of the offline optimal allocation is $nf(1)$, the result follows.

■

V. Lower Bound for LLQ Algorithms

We now provide example arrival sequences which provide lower bounds (i.e. outer bounds) on the LLQ algorithms.

A. Deterministic LLQ Algorithms

Lemma 2. *Let $|U| = n = 2^k$. The competitive ratio for any deterministic algorithm for the cost function f is at least*

$$\frac{f(k+1)}{2^k f(1)} + \sum_{i=1}^{k-1} \frac{f(i)}{f(1)} \frac{1}{2^{i+1}}.$$

Proof: For the purpose of illustration, we divide the 2^k job arrivals into k phases. In the first phase, 2^{k-1} jobs are introduced, each one can be served by all the 2^k servers. The servers that are assigned jobs by the algorithm in this phase are retained for the next phase. In the next phase, 2^{k-2} jobs are introduced, each one of them can be served by all the 2^{k-1} retained servers. The same process continues for the next $k-3$ phases. In the last phase, the adversary presents one job that can be served only by both the remaining servers and then presents an additional job that can be served only by the server that was assigned the first job in this phase. The cost of the allocation done by the deterministic algorithm is

$$2^k f(1) + 2^k \sum_{i=1}^{k} \left(\frac{1}{2}\right)^i S_i^{(f)} + S_{k+1}^{(f)}.$$

The optimal allocation has cost $2^k f(1)$. Therefore, the competitive ratio is

$$1 + \sum_{i=1}^{k} \left(\frac{1}{2}\right)^i \frac{S_i^{(f)}}{f(1)} + \frac{S_{k+1}^{(f)}}{2^k f(1)}. \quad (2)$$

So, the upper bound for the TIERED RANKING is lower than the lower bound for any deterministic LLQ algorithm.

■

B. Randomized LLQ Algorithms

Consider the following arrival pattern. Let T be the $n \times n$ complete upper-triangular matrix. Let the columns of this matrix represent jobs and the rows represent servers. We assume that jobs i.e. columns arrive from right to left. The entry $T(i,j) = 1$ if job j can be served by server i and 0 otherwise. In this section, we consider the set of all Randomized LLQ algorithms. For example for $n = 8$, the arrival matrix T is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Remark: We now remark on a particular recursive property of the above arrival sequence, that we use in our bounds. For any sample path of a randomized allocation algorithm, consider the sub-matrix formed by all jobs allocated at level i or above. In particular, consider the sub-matrix formed as follows: fix any number i, and remove all jobs (i.e. columns) that were allocated to servers that already had at most $i-1$ jobs (including that particular job). Then, remove servers which cannot be used by any of the remaining jobs (i.e. remove rows which, after the column removal, have no non-zero element).

Then, this sub-matrix is *upper triangular*. This is easy to see: let j be the job (i.e. column) that was the first to face all servers with loads $i-1$ or higher at arrival. Since every subsequent job can only be served by a subset of the servers that j can use, this means all of them are retained in the sub-matrix above. Also, every server that j could not use will be removed as well in the above construction. Thus the remaining matrix is a square top-left corner sub matrix of the above matrix, and hence is also upper triangular.

Lemma 3. *For the arrival sequence described above for $n > c$ where c is an absolute constant and for every level $l \geq 1$, the number N_l of jobs placed at levels higher than $l - 1$ satisfies:*

$$E[N_l] \geq n\left(\frac{1}{e}\right)^{l-1} - \sum_{i=0}^{l-1} \frac{1}{e^i}.$$

Proof: We prove the property by induction. For the upper triangular arrival sequence, for $n > c$ where c is an absolute constant, by Lemma 16 in [5], we have that

$$\begin{aligned} E[N_1] &\geq n\left(\frac{e^{1/n}}{e}\right) - 1 \\ &\geq \frac{n}{e} - 1. \end{aligned}$$

Therefore, the statement holds for $l = 1$. Assuming that the statement holds for $l = k$, we have that,

$$E[N_k] \geq n\left(\frac{1}{e}\right)^{k-1} - \sum_{i=0}^{k-1} \frac{1}{e^i}.$$

Let M_k denote the allocation obtained at level k by the randomized LLQ algorithm. We know that the input matrix at level k is upper triangular. For an upper triangular matrix, by Lemma 16 in [5], we have that,

$$\begin{aligned} E[M_k] &\leq \left(1 - \frac{e^{\frac{1}{N_k}}}{e}\right)N_k + 1 \\ &\leq \left(1 - \frac{1}{e}\right)N_k + 1. \end{aligned}$$

Therefore,

$$\begin{aligned} E[N_{k+1}] &= E[N_k] - E[M_k] \\ &\geq E[N_k] - \left(1 - \frac{1}{e}\right)E[N_k] - 1 \\ &= \frac{1}{e}E[N_k] - 1 \\ &\geq \frac{1}{e}\left(n\left(\frac{1}{e}\right)^{k-1} - \sum_{i=0}^{k-1} \frac{1}{e^i}\right) - 1 \\ &= n\left(\frac{1}{e}\right)^{k} - \sum_{i=0}^{k} \frac{1}{e^i}. \end{aligned}$$

This completes the proof by induction.

■

Theorem 2. *Let $C_i^{(f)}$ be as defined in (1). If the function f is such that*

$$\sum_{i=1}^{n} \frac{1}{e^i} f(i) = o(n),$$

then the expected competitive ratio of any Randomized LLQ Algorithm for the cost function f as $n \to \infty$ is at least

$$1 + \sum_{i=1}^{\infty} \left(\frac{1}{e}\right)^i \frac{S_i^{(f)}}{f(1)}.$$

Proof: We show this by evaluating any Randomized LLQ algorithm on the upper triangular arrival sequence defined above. Let C be the total cost of an allocation. We have that,

$$C = n + \sum_{i=1}^{n-1} N_i S_{i-1}^{(f)}.$$

By Lemma 3, we know that:

$$E[N_l] \geq n\left(\frac{1}{e}\right)^{l-1} - \sum_{i=0}^{l-1} \frac{1}{e^i}.$$

Since $S_i^{(f)} \geq 0$ for all $i \geq 1$, we get that

$$\begin{aligned} E[C] &\geq nf(1) + \sum_{i=1}^{n-1} \left(n\left(\frac{1}{e}\right)^i - \sum_{j=0}^{i-1} \frac{1}{e^j}\right) S_i^{(f)} \\ &= nf(1) + \sum_{i=1}^{n-1} n\left(\frac{1}{e}\right)^i S_i^{(f)} - \sum_{i=1}^{n-1}\sum_{j=0}^{i-1} \frac{1}{e^j} S_i^{(f)} \\ &= nf(1) + \sum_{i=1}^{n-1} n\left(\frac{1}{e}\right)^i S_i^{(f)} - \sum_{j=0}^{n-1} \frac{1}{e^j} \sum_{i=j}^{n-1} S_i^{(f)}. \end{aligned}$$

Note that,

$$\lim_{n\to\infty} \sum_{i=j}^{n-1} S_i^{(f)} \to \sum_{i=j}^{\infty} S_i^{(f)} = -C_j^{(f)}.$$

By the assumption on the arrival process, we have that cost of the optimal algorithm is $nf(1)$. Additionally, if we have that,

$$\sum_{i=0}^{n-1} \frac{1}{e^i} f(i) \approx o(n),$$

then, we have that,

$$\sum_{i=0}^{n-1} \frac{1}{e^i} C_i^{(f)} = \frac{e}{e-1} \sum_{i=0}^{n-1} \frac{1}{e^i} f(i) \approx o(n).$$

Therefore,

$$\lim_{n\to\infty} \frac{C}{nf(1)} \geq 1 + \sum_{i=1}^{\infty} \left(\frac{1}{e}\right)^i \frac{S_i^{(f)}}{f(1)}, \qquad (3)$$

which gives us the desired result. From (2) and (3), we can see that as $n \to \infty$, the randomized algorithms have a smaller lower bound for the expected competitive ratio than the deterministic algorithms. Therefore, in the class of all online LLQ algorithms, the competitive ratio is at least

$$1+\sum_{i=1}^{\infty}\left(\frac{1}{e}\right)^{i}\frac{S_i^{(f)}}{f(1)}.$$

■

VI. Lower Bound for Deterministic Algorithms

Theorem 3. *Let $|U| = n = 2^k$. The competitive ratio for any deterministic algorithm for the function f as $k \to \infty$ is*

$$1+\sum_{i=1}^{\infty}\left(\frac{1}{2}\right)^{i}\frac{S_i^{(f)}}{f(1)}.$$

Proof: For the purpose of illustration, we divide the n job arrivals into k phases. In the first phase, the adversary forms 2^{k-1} pairs. For each pair, the adversary presents one job that can be served only by both serves in that pair. The servers that are assigned jobs by the algorithm in this phase are retained for the next phase. The same process continues for the next $k-2$ phases. In the last phase, the adversary presents one job that can be served only by both the remaining servers and then presents an additional job that can be served only by the server that was assigned the first job in this phase. The cost of this allocation is

$$S_{k+1}^{(f)}+2^k\sum_{i=2}^{k-1}\left(\frac{1}{2}\right)^{i}S_i^{(f)}+2^kf(1).$$

As $k\to\infty$, the competitive ratio tends to

$$1+\sum_{i=1}^{\infty}\left(\frac{1}{2}\right)^{i}\frac{S_i^{(f)}}{f(1)}.$$

■

VII. ℓ_p norms

By Theorems 1 and 2, the TIERED RANKING algorithm is the optimal LLQ algorithm all ℓ_p norms for $p>1$. The case $p=2$ is of special interest as the square of the ℓ_2 norm of the load vector can be interpreted as the average delay of the jobs in the system assuming that the service discipline is processor sharing and each job needs a service of 1 unit.

Corollary 4. *The expected competitive ratio of Algorithm TIERED RANKING for average delay is at most $\dfrac{e+1}{e-1}$.*

Proof: For $p=2$, consider the cost function $f(i)=i^2$. Therefore,

$$S_i^{(2)}=(i+1)^2-2i^2+(i-1)^2=2.$$

The cost C of an allotment for $f(i)=i^2$ is

$$C=n+2\sum_{k=2}^{n}N_k.$$

Using the linearity of expectation and Lemma 1,

$$\begin{aligned}E[C] &= n+2\sum_{k=2}^{n}E[N_k]\\ &\le n+2n\sum_{k=1}^{\infty}\left(\frac{1}{e}\right)^{k}\\ &= n+2n\left(\frac{1}{e-1}\right)\\ &= n\left(\frac{e+1}{e-1}\right).\end{aligned}$$

Since the offline optimal has an ℓ_2 norm of $\sqrt{n}$ units, the expected competitive ratio for the ℓ_2 norm is less than $\sqrt{\dfrac{e+1}{e-1}}$. The competitive ratio for the average delay is less than $\dfrac{e+1}{e-1}=2.16$.

■

Table I
Upper Bounds for Expected Competitive Ratios

p	TIERED RANKING (Upper Bound)	BALANCE [7] (Upper Bound)
2	1.47	2.23
3	1.86	3.66
4	2.37	4.88
5	2.64	6.11
6	3.03	7.33

Table I lists the competitive ratios for two algorithms for a few values of p. The algorithm BALANCE introduced in [4] was analyzed for a more general setting, where the weight of each job on every server that can serve it was independent of its weight on any other server. The bounds listed for the BALANCE algorithm are for this more general setting.

In Figure 2 we can see that as $n\to\infty$, TIERED RANKING outperforms any deterministic algorithm for the ℓ_p norm.

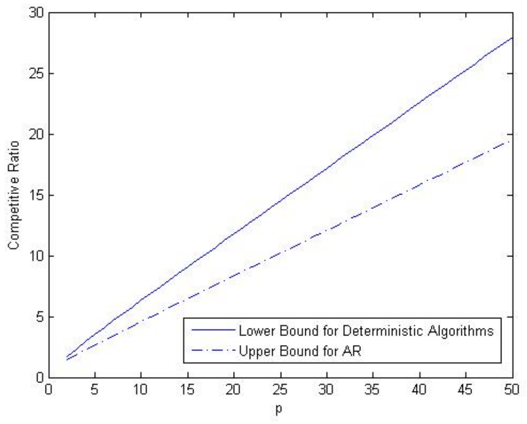

Figure 2. *Competitive Ratios for the ℓ_p Norm*

VIII. Simulations: RANDOM LLQ vs TIERED RANKING

In this section, we compare the performance of TIERED RANKING with another randomized LLQ algorithm for the cost function being the square of the ℓ_2 norm. We call the new algorithm RANDOM LLQ. When a job comes in, the algorithm RANDOM LLQ allots it to any one of the lowest loaded servers that can serve it, and breaks ties uniformly at random, and independent of past choices.

Consider an arrival matrix such that $T(i,i) = 1$ for $1 \le i \le n$, $T(i,j) = 1$ if $j \ge n/2$, $i \le n/2$, $T(i,j) = 1$ if $j \ge n/4$, $i \le n/4$ and $T(i,j) = 0$ otherwise. Recall that columns represent jobs and rows represent servers. The columns arrive from right to left. For example, for $n = 8$, this matrix is

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We now analyse the performance of the RANDOM LLQ algorithm for this matrix. We first focus on the first $n/2$ servers. Let $x(t)$ and $y(t)$ be random variables denoting the number of jobs remaining and the number of servers among the first $n/2$ servers which have not been allocated a packet by time t. Let

$$\Delta x = x(t) - x(t+1),$$
$$\Delta y = y(t) - y(t+1).$$

Therefore, $\Delta x = -1$ and

$$\Delta y = -1 \text{ w.p. } \frac{y}{y+1}$$

and 0 otherwise. We therefore have that

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

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

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

For $x = n/2$, we have that:

$$y + \log\frac{2y}{n} = 0. \tag{4}$$

Consider the function

$$f(y) = e^{-y} - \frac{2y}{n}.$$

The solution to Equation 4 is the same as the solution to $f(y) = 0$. We have that $f(0) = 1$, f is a strictly decreasing function and $f(\log(n)) = (1 - 2\log(n))/n < 0$ for n large enough. Therefore, y^*, the solution to Equation 4 is $< \log(n)$. Therefore, $M_1 \le n/2 + o(n)$. Carrying out the same analysis for the next $n/4$ packet arrivals, we have that $M_2 \le n/4 + o(n)$ and $M_3 = n - (M_1 + M_2)$. Therefore the total cost C of allotment is

$$\begin{aligned} C &= M_1 + 3M_2 + 5M_3 \\ &= M_1 + 3M_3 + 5(n - (M_1 + M_2)) \\ &= 5n - 4M_1 - 2M_2 \\ &\ge 2.5n + o(n). \end{aligned}$$

Therefore the competitive ratio for RANDOM LLQ is ≥ 2.5 which is greater than the upper bound for the competitive ratio of TIERED RANKING for the square of the ℓ_2 norm (2.16).

In Figure 3 we see the competitive ratios for RANDOM and TIERED RANKING for the matrix T for different values of n. It can be seen that TIERED RANKING outperforms RANDOM LLQ for this input sequence.

IX. Conclusions and Discussion

We considered a natural model for online allocation of jobs to servers with a constraint on which serve can serve each job. Our algorithm, and analysis established the (somewhat counter-intuitive) importance of *correlated* randomness in this setting.

Several extensions naturally suggest themselves:

1) looking at systems with departures.
2) different cost functions for different servers.
3) stochastic vs adversarial arrivals.
4) more general arrival sequences.

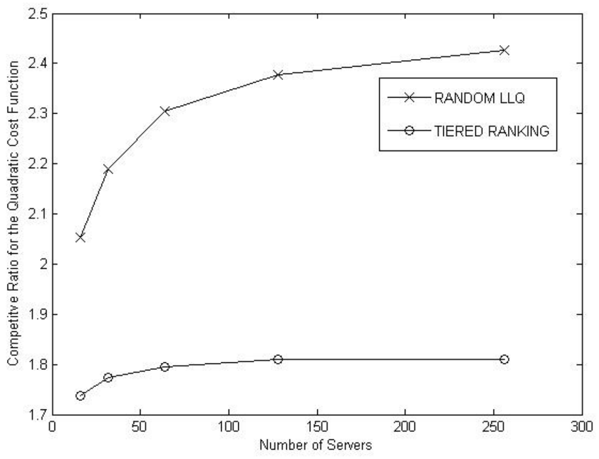

Figure 3. *Competitive Ratios for $f(i) = i^2$ for T*

References

[1] U. Vazirani V. Vazirani A. Mehta, A. Saberi. Adwords and generalized on-line matching. *Proceedings of FOCS*, 2005.

[2] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, Orlando, FL, September 1992.

[3] E. F. Grove M.-Y. Kao P. Krishnan B. Awerbuch, Y. Azar and J. S. Vitter. Load balancing in the ℓ_p norm.

[4] I. Caragiannis. Better bounds for online load balancing on unrelated machines. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, San Francisco, California, 2008.

[5] 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*, Baltimore, Maryland, May 1990.