Online Server Allocation in a Server Farm via Benefit Task Systems

[Extended Abstract]

T.S. Jayram* Tracy Kimbrel† Robert Krauthgamer‡
Baruch Schieber† Maxim Sviridenko†

ABSTRACT

A web content hosting service provider needs to dynamically allocate servers in a server farm to its customers' web sites. Ideally, the allocation to a site should always suffice to handle its load. However, due to a limited number of servers and the overhead incurred in changing the allocation of a server from one site to another, the system may become overloaded. The problem faced by the web hosting service provider is how to *allocate* the available *servers* in the most profitable way. Adding to the complexity of this problem is the fact that future loads of the sites are either unknown or known only for the very near future.

In this paper we model this *server allocation problem*, and consider both its offline and online versions. We give a polynomial time algorithm for computing the optimal offline allocation. In the online setting, we show almost optimal algorithms (both deterministic and randomized) for any positive lookahead. The quality of the solution improves as the lookahead increases. We also consider several special cases of practical interest. Finally, we present some experimental results using actual trace data that show that one of our online algorithm performs very close to optimal.

Interestingly, the online server allocation problem can be cast as a more general *benefit task system* that we define. Our results extend to this task system, which captures also the benefit maximization variants of the k-server problem and the metrical task system problem. It follows that the benefit maximization variants of these problems are more tractable than their cost minimization variants.

*IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. E-mail: jayram@almaden.ibm.com
†IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.
E-mail: {kimbrel,sbar,sviri}@watson.ibm.com
‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: robi@wisdom.weizmann.ac.il. Part of this work was done while this author was visiting IBM T.J. Watson Research Center.

STOC'01, July 6-8, 2001, Hersonissos, Crete, Greece.

1. INTRODUCTION

Web content hosting is an important emerging market. Data centers and web server "farms" are proliferating (see, e.g., [19, 20]). The rationale for using such centers is that service providers can benefit from economies of scale and sharing of resources among multiple customers. This benefit translates, in turn, to lower cost of maintenance for the customers who purchase these hosting services. Web content hosting services are structured in many ways. One of the most prevailing ways is *outsourcing*: the customers deliver their web site content to the web content hosting service provider, and the service provider is responsible for serving this content in response to HTTP requests. Most of the service providers use "farms" of commodity servers to achieve this goal.

One of the components in the payment for such a service is "pay per served request". Thus, one of the main objectives of the service provider is to maximize the revenue from served requests while keeping a tab on the amount of resources used. Ideally, the allocation to a web site should always suffice to serve its requests. However, due to a limited number of servers and the overhead incurred in changing the allocation of a server from one site to another, the system may become overloaded, and requests may be left unserved. We make the realistic assumption that requests are not queued, and a request is lost if it is not served at the time it is received. The problem faced by the web hosting service provider is how to utilize the available servers in the most profitable way. Adding to the complexity of this problem is the fact that future loads of the sites are either unknown or known only for the very near future. In this paper we model this problem, and consider both its offline and online versions.

In our model time is divided into intervals of fixed length. We assume that each site's demand is uniformly spread throughout each such interval. Server allocations remain fixed for the duration of an interval. We assume that servers are reallocated only at the beginning of an interval, and that a reallocated server is unavailable for the length of the interval during which it is reallocated. This represents the time to "scrub" the old site (customer data) to which the server was allocated, to reboot the server and to load the new site to which the server has been allocated. The length of the time interval is set to be equal to the nonnegligible amount of time required for a server to prepare to serve a new customer. In current technology, this time is in the order of 5 to 10 minutes. In the sequel, we normalize time so that the length of a time interval is one unit.

Algorithm	Lookahead	Lower Bound	Upper Bound
Deterministic	$L = 1$	$4 - \epsilon$	4
	general L	$1 + \frac{1}{L} + \frac{1}{L^2+1}$	$\min\left\{1 + \frac{4}{L-7}\ ,\ \left(1 + \frac{1}{L}\right)\ \sqrt[L]{L+1}\right\}$
Randomized	general L	$1 + \frac{1}{L} - \epsilon$	$1 + \frac{1}{L}$

Table 1: Results

Each server has a rate of requests it can serve in a time interval. For simplicity, we assume that all rates are identical. Due to practical concerns (security constraints placed by customers), we do not allow sharing of servers at the same time. Customers share servers only in the sense of using the same servers at different times, but do not use the same servers at the same time. Thus, even in case of overload, some of the servers may be underutilized if they are allocated to sites with rates of requests lower than the servers' rate. In the sequel, we normalize rates so that the servers' rate is 1, and allow the demand of sites to be fractional.

We assume that each customer's demand is associated with a benefit gained by the service provider in case a unit demand is satisfied. Given the number of available servers, the objective of the service provider is to find a time-varying server allocation that maximizes the benefit gained by satisfying sites' demand. We call this server allocation problem the *web server farm problem*. In the offline version of this problem, future demands of the sites are known. In the fully online version, only the demand of the current interval is known at the beginning of the interval. It makes sense to assume that some amount of future demand is known to the service provider, and we model it by *lookahead*. Lookahead L means that the demand of the sites is known for the next L time intervals (following the current interval). In reality, lookahead will require that some forecasting mechanism be used to predict future demands. Fortunately, web traffic exhibits properties of predictability over the short term which render this practical as demonstrated in [13, 18]. We measure the performance of an online allocation algorithm using its *competitive ratio*; that is, the maximum over all instances of the ratio of the benefit gained by the optimal offline allocation to the benefit gained by the online allocation on the same instance. Since we are dealing with a maximization problem, this ratio is always greater than 1, and the goal is to make it as small as possible.

Interestingly, our model can be cast as a more general *benefit task system*. In this task system we are given a set of states for each time t and a benefit function. The system can be in a single state at each time, and the benefit gained between times t and $t+1$ is a function of the system states at times t and $t+1$. The goal is to find a time-varying sequence of states that maximizes the total benefit gained; that is, at each time t we need to determine to which state should the system move (and this will be the state of the system at time $t+1$), and we gain the benefit that is determined by the benefit function. Similar to the web server farm problem, in the offline version, the benefit function is known in advance. In the fully online version, only the possible benefits of the current step are known at the time a move is determined, and in case of lookahead L, the benefits of the next L steps (in addition to the current) are known. Note that number of states in a benefit task system might be super-polynomial if the states are not given explicitly.

It can be shown that benefit task systems capture also benefit maximization variants of well studied problems, such as the k-server problem [16, 9] and metrical task systems [10, 9] (see Section 2). Thus, our results hold for these variants as well, and show that the benefit variants of these problems may be more tractable than their cost minimization variants.

Our results

We observe that the offline version of the web server farm problem can be solved in polynomial time. In the online setting, we first note that the competitive ratio in the fully online version is unbounded; we then consider the case of a positive lookahead for both deterministic algorithms and randomized ones. We obtain tight bounds on the competitive ratio for the randomized case and almost tight bounds for the deterministic case. In the deterministic case we also consider the special case $L = 1$. The specific results are summarized in Table 1. Our bounds on the competitive ratio extend to the benefit task system problem, assuming that an optimal sequence of moves for the period of the lookahead (i.e. for the next $L + 1$ time steps) is efficiently computable. Therefore, our upper bounds (i.e. algorithms) are presented in the more general framework of the benefit task system problem, and the lower bounds are given for the more restricted web server farm problem.

In addition, we consider two special cases of the web server farm problem with lookahead 1. One is the basic case of one server and two sites. For the competitive ratio in this case we obtain a lower bound of $3\sqrt{3}/2 \approx 2.598$ and an upper bound of $1 + \phi \approx 2.618$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The second case is that of k servers and arbitrary number of sites, for which we give an algorithm that achieves a competitive ratio 2 if k is even and $2 + 2/(2k - 1)$ if k is odd.

Several of our algorithms use the natural approach of "discounting the future". Specifically, when the algorithm faces the choice between a guaranteed benefit immediately and a potential benefit in the future, the decision is made by comparing the guaranteed benefit value with a discounted value of the potential future benefit. This discount factor is exponential in the number of time units that it would take the potential benefit to be materialized. The reason for discounting future benefits is straightforward. By the time a benefit will be materialized, things might change and the algorithm might decide to make another choice for a potential (even larger) benefit.

Another approach used by our algorithms in the case of lookahead is an "intermittent reset". Some number of future steps are determined and then a single time step is "wasted" to reallocate the servers in preparation for another sequence of future steps. The time step for reallocation is chosen so that the amount of benefit unrealized due to the reallocation is guaranteed to be small with respect to the benefit that is gained.

We use actual web traffic data of commercial sites and randomly generated data to evaluate our algorithms. Our experiments show that the discount algorithm performs better, with less lookahead, than the intermittent reset algorithm, and is within 0.2% of optimal on all the commercial trace instances, even with lookahead one.

Related work

Our web server farm problem can be viewed as a scheduling problem. The requests are represented by unit length jobs, and the k servers are machines. Each job has a deadline which is exactly one time unit after its release, and a value. The goal is to maximize the throughput; that is, to maximize the value of the scheduled jobs. An additional factor is *setup*: it takes one time unit to set up a machine to handle a job for one web site if it handled a job in the previous time unit for a different site.

Several papers have considered the problem of throughput maximization in scheduling jobs on parallel machines with release times and deadlines, both in the offline and online settings [5, 3, 7, 8, 14]. In the standard notation, this type of problem is denoted by $P|r_i|\sum w_i(1-U_i)$. The type of setup considered here is related to "batch setup" scheduling, for which several offline problems have been studied (see [17] for a survey). The online problems considered here also relate to previous work on online call admission and bandwidth allocation [12, 15, 4, 6]. Again, the issue of setups does not appear in any of these papers.

The benefit task system can also be viewed as a profit maximization version of the request-answer game of [9]. To the best of our knowledge this is the first attempt to consider a task system for profit maximization rather than cost minimization problems. Awerbuch et al. [2] considered a similar problem in which a decision maker that has no way to accurately predict future performance of several options has to choose one option that will have the best future performance. This problem has an unbounded deterministic competitive ratio, and the authors propose a randomized algorithm. Lookahead is not considered in [2].

Organization. In the next section we give more accurate definitions of our models and show their relations to the k-server model and metrical task systems. In Section 3 we describe an offline algorithm for the server farm problem. In Section 4 we devise two deterministic online algorithms for the benefit task system, based on discounting the future and on an intermittent reset, and we give upper bounds on their competitive ratio. In Section 5 we prove lower bounds on the competitive ratio of any deterministic online algorithm for the more restricted web server farm problem. In Section 6 we consider some special cases of the online deterministic web server farm problem. In Section 7 we give tight bounds for the randomized case. We conclude in Section 8 with some experimental results based on traces of commercial retail sites.

2. THE PROBLEMS

The web server farm problem: Suppose that we are given s web sites that are to be served by k web servers. (For simplicity, we assume that all servers are identical.) Time is divided into units (a.k.a. steps). It is assumed that the demand of a web site is uniform in each time unit. Each server has a "service rate" which is the number of requests to a web site each server can serve in a time unit. Without loss of generality, we normalize the demands by the service rate so that a server can serve unit demand per time unit and demands of a site may be fractional. A web server can be allocated to no more than one site at each time unit and it takes a time unit to change the allocation of a server.

A problem instance consists of the number of servers k, the number of sites s, a nonnegative benefit matrix $\{b_{i,t}\}$ denoting the benefit gained by serving a request of site $i \in [1..s]$ for time step $t \in [1..\tau]$, and a nonnegative demand matrix $\{d_{i,t}\}$ denoting the number of requests at site i for time step t. The goal is to find for each site $i \in [1..s]$, a time-varying allocation $\{a_{i,t}\}$ of servers, so as to maximize the total benefit, as follows. The allocation must satisfy that for each t, $\sum_{i=1}^{s} a_{i,t} \le k$. Only $a'_{i,t} = \min\{a_{i,t-1}, a_{i,t}\}$ of the servers allocated to site i for time step t are "productive", i.e., actually serve requests. We get that the total benefit of an allocation $\{a_{i,t}\}$ is $\sum_{t=1}^{\tau}\sum_{i=1}^{s} b_{i,t} \cdot \min\{d_{i,t}, a'_{i,t}\} = \sum_{t=1}^{\tau}\sum_{i=1}^{s} b_{i,t} \cdot \min\{d_{i,t}, a_{i,t}, a_{i,t-1}\}$. The initial allocation is fixed to be, say, $a_{i,0} = 0$ for all i.

In the *offline* version of this problem we are given the complete demand matrix $\{d_{i,t}\}$ and we need to compute the complete allocation $\{a_{i,t}\}$. In the *fully online* version we need to compute the allocation $\{a_{i,t}\}$ at time t given the demands so far, i.e., $\{d_{i,t'}\}$ for all i and $t' \le t$. It is not difficult to show that any fully online algorithm may perform arbitrarily badly with respect to the offline optimal solution. This is true because of the one time unit lag in a server's availability. Thus we consider online algorithms with lookahead. In the *online* version with *lookahead* L, at each time t we are additionally given the demands and benefits for times $t+1, \ldots, t+L$, i.e., the entries $d_{i,t'}$ and $b_{i,t'}$ for $i \in [1..s]$ and $t' \in [t+1..t+L]$, and we need to compute the allocation at time t, i.e., $a_{i,t}$ for $i \in [1..s]$.

The benefit task system problem: The web server farm problem is a special case of the generalized task system benefit problem. In this problem we are given (i) a set of possible states U_t, for each time $t \in [0..\tau]$, and (ii) a nonnegative benefit function B whose domain is $\bigcup_t (U_t \times U_{t+1})$, that denotes, for each time t, the benefit that is accrued (at time $t+1$) by the transition from a state in U_t to a state in U_{t+1}. The goal is to choose a state s_t for each time t so as to maximize the total benefit $\sum_{t=0}^{\tau-1} B(s_t, s_{t+1})$. The initial state is fixed by letting that $U_0 = \{s_0\}$.

In the *offline* version of the problem all the state sets and the benefit function are known in advance. In the *online* version with lookahead L, the state $s_t \in U_t$ should be computed based on knowing only $U_{t'}$ for $t' \le t + L$, and the restriction of the function B to pairs of these sets of states.

Observe that the web server farm problem can be cast in this setting by identifying each possible allocation of servers to sites at time t with a state $S_{i,t}$, and defining the benefit function $B(S_{i,t}, S_{j,t+1})$ to be the benefit gained by (serving some number of requests while) changing the allocation at time t from the one represented by $S_{i,t}$ to the allocation at time $t+1$ represented by $S_{j,t+1}$. (In a sense, the set of states for all times are the same.) The number of states is exponential in the number of servers k, so the states and the benefit function are implicit and follow from the more succinct representation of the web server farm problem. For example, the values $B(S_{i,t}, S_{j,t+1})$ are not listed explicitly, and any single value can be efficiently computed when necessary.

Benefit maximization variants of well known problems can also be cast in this setting of a benefit task system. Consider the benefit version of the k-server problem. This version of the k-server is similar to the classical k-server problem [16] with one difference. Instead of minimizing the cost of satisfying all requests, we define for each time t and any possible pair of server configurations, one at time $t-1$ and one at time t, a net benefit gained by satisfying the request starting from the configuration at time $t-1$ and ending at the configuration at time t. (Note that the configuration at time t must include at least one server at the point of the request.) This benefit has to be nonnegative and is composed of a fixed positive component that is reduced by the cost to move from the configuration at time $t-1$ to the configuration at time t. The goal in this case is to maximize the total benefit. It is not difficult to see that this problem can also be modeled by the benefit task system defined above and thus all our results apply to the benefit version of the k-server problem.

Similarly, consider the benefit version of a metrical task system. This version is similar to the classical metrical task system [10], with the difference that each task is associated with a vector of benefits, one for each state, such that the net benefit after subtracting the transition cost is nonnegative. This model as well can be cast as a benefit task system and our results apply.

3. OFFLINE ALGORITHM FOR THE WEB SERVER FARM PROBLEM

In this section we show that the offline web server farm problem can be solved in polynomial-time. Note that an algorithm that solves this offline problem is also a component in the deterministic online algorithms of Section 4, as these online algorithms rely on an ability to find an optimal sequence of allocations in the limited future given by the lookahead (i.e., the next $L+1$ time units).

We reduce the web server farm problem to the well-known minimum-cost network flow problem, which can be solved in polynomial time; see for example [1, 11]. We remark that when s^k is polynomial in the input size, the web server farm problem can be solved in polynomial time also using dynamic programming. However, in general k might be exponential in the input size.

THEOREM 1. *The offline web server farm problem can be reduced in polynomial time to a minimum-cost network flow problem. Hence, it can be solved in polynomial time.*

PROOF. Recall that the input for the minimum cost network flow problem is a directed network, two special vertices called the source vertex and the sink vertex, an amount of flow to be injected into the source vertex, and a nonnegative capacity and a cost for each edge. The goal is to find from all the flows from s to t that respect the edge capacities and are of size k, one that has a minimal cost, where the cost of a flow is the sum, over all edges, of the product of the flow on this edge and its cost.

In fact, we describe below a reduction from the web server farm problem to the analogous maximum-cost network flow problem. The latter problem is essentially the same as the minimum-cost network flow problem (and thus can be solved in polynomial time), since the edge costs are not restricted in sign, i.e., are allowed to be negative. We remark that in our network, all the paths from s to t are of equal length, and therefore another way to guarantee that all costs are nonnegative is to increase the costs of all the edges by the same sufficiently large number.

The outline of the reduction is as follows. Given an instance of the web server farm problem, we construct a (directed) network with s "rails", one per site. Each rail is a chain of edges, each representing one time step. (Actually, we will split these edges into three parallel edges, for reasons which will become clear shortly.) Flow along a rail represents the allocation of servers to the corresponding site. In addition, we construct a set of "free pool" nodes, one per time step, through which flow will pass when servers are reallocated from one site to another.

Let $\{d_{i,t}\}$ with $i \in [1..s]$ and $t \in [1..\tau]$ be the demand matrix. We construct nodes $n_{i,t}$ for $i \in [1..s]$ and $t \in [1..\tau]$, along with nodes f_t for $t \in [1..\tau]$. For each site s and each time step t, we construct three edges from $n_{i,t-1}$ to $n_{i,t}$. The first has capacity $\lfloor d_{i,t} \rfloor$ and cost $b_{i,t}$. The second has capacity one and cost $b_{i,t} \cdot (d_{i,t} - \lfloor d_{i,t} \rfloor)$. The last has infinite capacity and cost 0. Flow along the first edge represents the benefit of allocating servers to site i during time step t, up to the integer part of $d_{i,t}$. Flow along the second represents the remaining benefit, $b_{i,t}$ times the fractional part of $d_{i,t}$, to be collected by one more server (that will be only partially productive). Flow along the third represents the fact that extra servers can remain allocated to i, but do not collect any benefit. Clearly the first edge will saturate before flow is present on the second, and similarly the second will saturate before flow is present on the third.

We also construct edges of infinite capacity and cost 0 from $n_{i,t-1}$ to f_t and from f_t to $n_{i,t}$, for each $t \in [1..\tau]$ and $i \in [1..s]$. These represent the movement of servers from one site to another. (We use an intermediate node f_t to keep the number of edges linear.)

Finally, we construct a source into which we inject flow k, with infinite capacity zero cost edges to each $n_{i,1}$, and a sink with infinite capacity zero cost edges from each $n_{i,\tau}$.

It is not hard to see that an integral flow of cost C in this network corresponds to an allocation $\{a_{i,t}\}$ with benefit equal to C, and vice versa. It is well known that since all the edge capacities are integral, there is minimum-cost (or maximum-cost) flow in the network that is integral, and that, furthermore, such a flow can be efficiently found. This implies an efficient algorithm that finds an optimal allocation $\{a_{i,t}\}$. Note that the size of the network (number of nodes and edges) is linear in the size of the demand matrix $\{d_{i,t}\}$, and thus the offline web server farm problem can be solved within roughly the the same time as the best algorithm for minimum-cost network flow on a network with $O(s \cdot \tau)$ nodes and edges. □

4. DETERMINISTIC ONLINE ALGORITHMS FOR BENEFIT TASK SYSTEMS

In this section we describe two deterministic algorithms for benefit task systems with lookahead L. The first algorithm is a future discounting algorithm that achieves a competitive ratio of $(1+1/L)\sqrt[L]{L+1}$. Asymptotically in L this ratio is $1+\Theta(\log L)/L$. The second algorithm is an intermittent reset algorithm that achieves a competitive ratio of $1+4/(L-7)$, which is better than the first algorithm

when L is sufficiently large. Each algorithm requires that one can efficiently find a sequence of moves that is optimal (in a certain sense) for the period of the lookahead (i.e. for the next $L+1$ time steps). In the particular case of the web server farm problem, we can achieve this requirement by using the offline algorithm of Section 3.

4.1 The future discounting algorithm

We present a deterministic algorithm that achieves competitive ratio $(1+1/L)\sqrt[L]{L+1}$ for the benefit task system problem. The algorithm is based on discounting future benefits in an exponential way, as follows. Consider a sequence of $L+1$ moves that collects the benefits $b_0, b_1, \ldots, b_L$ (in this order) in the next $L+1$ time steps. We define the *anticipated benefit* of this sequence to be $b_0 + b_1/\alpha + \ldots + b_L/\alpha^L$ (for a parameter $\alpha > 1$ that we later choose to be $\alpha = \sqrt[L]{L+1}$).

The discounting algorithm greedily follows at each time step the largest anticipated benefit possible at that time. Namely, at every time step the algorithm finds a sequence of $L+1$ moves (for the next $L+1$ time steps) whose anticipated benefit is maximal, and performs the first move in this sequence. We stress that a new sequence is calculated at every time step, and that the sequence found in the current time step does not have to agree with the one found in the previous time step.

Note that this discounting algorithm requires efficient computation of a sequence of $L+1$ moves with the maximum anticipated benefit. The following lemma provides this requirement when the number of states at every time t is polynomial in the input size, e.g. when the states of the benefit task system are given explicitly. In the web server farm problem, the number of (implicit) states is super-polynomial, but we can use the offline algorithm of Section 3, with straightforward modifications for handling the initial state and the discount.

LEMMA 2. *Suppose that the number of states possible for any one time is at most M. Then a sequence of $L+1$ moves with the largest anticipated benefit can be computed in time that is polynomial in M (and L).*

PROOF. Use dynamic programming that goes iteratively over the time steps and accumulates discounted benefits. As the additional benefit of the next time step depends only on the current state, it suffices to have for each time step t a table of size M, and compute the table for time step $t+1$ from that of time step t. □

THEOREM 3. *The competitive ratio of the above discounting algorithm is at most $(1+1/L)\sqrt[L]{L+1}$.*

PROOF. Consider among the sequences of $L+1$ moves that are available for the online algorithm at time t, the sequence with the largest anticipated benefit, and let $b_0, b_1, \ldots, b_L$ be the benefits collected by this sequence. Define $ON_t = b_0$ and $ON^{*L}_{t+1} = b_1/\alpha + \ldots + b_L/\alpha^L$, and then $ON_t + ON^{*L}_{t+1}$ is the largest anticipated benefit over all sequences of $L+1$ moves that are available to the online algorithm at time t. The online algorithm performs the first move in this sequence, and thus the benefit it collects at time t is ON_t.

Fixing an offline algorithm arbitrarily, let OFF_t denote the benefit that this offline algorithm collects at time t, and let OFF^L_t be a shorthand for $OFF_{t+1}/\alpha + \ldots + OFF_{t+L}/\alpha^L$.

One sequence of moves that the online algorithm considers at time t is to first join the offline algorithm, (i.e., move to the same state as the offline algorithm at time t), and then follow the offline algorithm for the next L steps. The anticipated benefit of this sequence is at least OFF^L_t. Since the online algorithm follows at time t the sequence of moves with the largest anticipated benefit, we get that

$$ON_t + ON^{*L}_{t+1} \geq OFF^L_{t+1} \tag{1}$$

Another sequence of moves that is considered by the online algorithm at time t is to follow the sequence that had the largest anticipated benefit at the previous time step $t-1$ (without the first step of that sequence, which was performed at time $t-1$, and with an arbitrary move added at the end of the sequence.) The contribution of these benefits to the anticipated benefit at time t is larger by a factor of α than their contribution at time $t-1$, and so the anticipated benefit of this sequence of moves is at least αON^{*L}_t. Since the online algorithm follows at time t the sequence of moves with the largest anticipated benefit, we get that

$$ON_t + ON^{*L}_{t+1} \geq \alpha ON^{*L}_t \tag{2}$$

Adding $(1-\frac{1}{\alpha})$ times inequality (1) and $\frac{1}{\alpha}$ times inequality (2), we get that[1]

$$ON_t + ON^{*L}_{t+1} \geq ON^{*L}_t + (1-\frac{1}{\alpha})OFF^L_{t+1} \ . \tag{3}$$

Adding up inequality (3) over all time steps t we get that (it is straightforward that we can assume, without loss of generality, that all the benefits in the first and last $L+1$ time steps are zero regardless of the state):

$$\begin{aligned} \sum_t ON_t &\geq (1-\frac{1}{\alpha})\sum_t OFF^L_{t+1} \\ &= (1-\frac{1}{\alpha})\left(\frac{1}{\alpha} + \ldots + \frac{1}{\alpha^L}\right)\sum_t OFF_t \ . \end{aligned}$$

The last equality follows since every benefit that the offline collects occurs in $\sum_t OFF^L_{t+1}$ with each of the discounts $1/\alpha, \ldots, 1/\alpha^L$. We conclude that the competitive ratio of the algorithm is (choosing $\alpha = \sqrt[L]{L+1}$):

$$\frac{\sum_t OFF_t}{\sum_t ON_t} \leq \frac{1}{\frac{1}{\alpha} - \frac{1}{\alpha^{L+1}}} = \frac{\alpha^{L+1}}{\alpha^L - 1} = (1+1/L)\sqrt[L]{L+1} \ .$$

□

4.2 The intermittent reset algorithm

We present a deterministic intermittent reset online algorithm that has competitive ratio $1 + O(1/L)$ for the benefit task system problem. This algorithm is motivated by the randomized algorithm presented in Section 7 that sacrifices one out of every $L+1$ steps in order to gain the optimal benefit between sacrificed steps. Here, we sacrifice one out of every $\Theta(L)$ steps, but the choice of steps to sacrifice is done carefully in a deterministic fashion and the overall benefit that we gain is close to the expected benefit of the randomized algorithm.

The algorithm works in iterations of length at least $L/2$ and at most L. Let s_i denote the start time of iteration i. Each iteration i consists of four steps, as follows.

[1] In a sense, this is a potential function analysis. See for example [9].

STEP 1: Consider all times $t = s_i + L/2 + 1 \ldots, s_i + L$. For each such time t, let x_t denote the maximum benefit of any transition from a state at time t to a state at time $t+1$. Let T be the time t that minimizes x_t, i.e., $x_T = \min\{x_t : s_i + L/2 + 1 \leq t \leq s_i + L\}$.

STEP 2: Compute the optimal sequence of transitions that starts at any state at time $s_i + 1$ and ends at any state at time T. This can be done either by dynamic programming similar to Lemma 2 or by applying the offline algorithm of Section 3. (In the first iteration the initial state is given and thus we may compute the optimal sequence of transitions from the initial state to any state at time T. The next step has to be modified accordingly.)

STEP 3: Move from the current state at time s_i to the state at time $s_i + 1$ that is the starting state of the optimal sequence computed above, and continue with the optimal sequence of moves to the state at time T.

STEP 4: Set the starting point $s_{i+1} = T$ for the next iteration.

THEOREM 4. *The competitive ratio of the above intermittent reset algorithm is at most* $1 + \frac{4}{L-7}$.

PROOF. We assume that L is a multiple of 4, and otherwise round it downwards to the nearest multiple of 4 (at the expense of increasing the competitive ratio as explained later). We will call the starting times of the iterations "gaps." Denote the gaps by $s_1, s_2, \ldots$. We divide the transitions taken by the algorithm into two components, the moves taken in the gaps and the steps taken in the rest of the times. Consider the path taken by the online algorithm. It may not collect any benefit in the gaps, but since it computes the optimal paths between gaps (allowing arbitrary initial and terminal states adjacent to the gaps), clearly the benefit of the optimal offline between gaps is no larger than the benefit of the online algorithm between gaps.

Now we bound the benefit of the optimal offline algorithm in the gaps. Consider iteration i starting at s_i and computing $T = s_{i+1}$. Consider the portion of the online path from $s_i + L/2 + 1$ to s_{i+1}. Denote the length of this portion by a, i.e., $a = s_{i+1} - (s_i + L/2 + 1)$. Consider the portion of the path computed in the *next* iteration from $s_{i+1} + 1$ to $s_i + L + 1$. Denote the length of this portion by b, i.e., $b = s_i + L + 1 - (s_{i+1} + 1)$. By our choice of L, $L/2$ is even and $a + b = L/2 - 1$. Thus one of a and b is even and the other is odd. Assume a is odd and b is even; the proof is similar in the opposite case. (We note that ideally, we would like both a and b to be even, but choosing $a + b$ to be even may lead to a worse case when both are odd.)

We claim that the benefit achieved by the online algorithm in these portions of the path is at least $x_T \cdot (a + b - 1)/2$. This is true since in every time step in the range there is a transition of benefit x_T and thus the online algorithm can achieve benefit at least x_T on every other transition, even if it collects no benefit on the other half of the transitions. Since a is odd, it can collect at least $x_T \cdot (a-1)/2$ in the first portion, and since b is even, it can collect at least $x_T \cdot b/2$ in the second.

Putting these together along with $a+b = L/2-1$ we have that the online benefit is at least

$$x_T \cdot (\frac{a-1}{2} + \frac{b}{2}) = x_T \cdot \frac{L-4}{4}.$$

Since $s_i + L + 1$ is always less than $s_{i+1} + L/2 + 1$, these portions are disjoint among iterations. Summing over all iterations we have $ON \cdot (\frac{4}{L-4}) \geq \sum_i x_{T_i}$, where ON denotes the total benefit collected by the online algorithm and x_{T_i} denotes the maximum benefit possible in the ith gap. Accounting for the portions outside the gaps we have $ON \geq OFF - \sum_i x_{T_i}$, where OFF denotes the benefit of the optimal offline algorithm. Putting these together and allowing for values of L that are not multiples of 4 yields the desired bound of $1 + \frac{4}{L-7}$. □

5. LOWER BOUNDS FOR DETERMINISTIC ONLINE ALGORITHMS

In this section we prove three lower bounds on the competitive ratio of deterministic online algorithms for the web server farm problem with one server. The first lower bound is for general lookahead, the second is for lookahead 1 (and arbitrary number of sites), and the third is for lookahead 1 and two sites.

5.1 A lower bound for general lookahead

We show a lower bound of $1+\frac{1}{L}+\frac{1}{L^2+1}$ on the competitive ratio of a deterministic online algorithm for the web server farm problem with one server. The same lower bound then follows for the benefit task system problem. Formally, we have the following theorem.

THEOREM 5. *The competitive ratio of any deterministic online algorithm for the web server farm problem with one server (and hence for the benefit task system problem) is larger than* $1 + \frac{1}{L} + \frac{1}{L^2+1}$.

PROOF. Consider a deterministic online algorithm whose input is a $3\times(L+3)$ demand table $\{d_{i,t}\}$. Since the algorithm has lookahead L, it is given at time $t = 0$ only the first $L+1$ columns, as illustrated below. Since the first column is all zeros, we may assume without loss of generality that the server is initially allocated to the first site. We denote the sequence of locations of the server by triangles, and use a parameter a that will be determined later.

$$\{d_{i,t}\} = \begin{bmatrix} \triangleright 0 & 1 & a & \ldots & a & ? & ? \\ 0 & 1 & a & \ldots & a & ? & ? \\ 0 & 1 & a & \ldots & a & ? & ? \end{bmatrix}$$

Since at time $t = 0$ all three rows look the same, we may assume without loss of generality that the algorithm decides to allocate the server at this time to the first site, and it collects a benefit of 0. Next, at time $t = 1$ another column is revealed to the algorithm, and the situation is as follows.

$$\{d_{i,t}\} = \begin{bmatrix} \triangleright 0 & \triangleright 1 & a & \ldots & a & 0 & ? \\ 0 & 1 & a & \ldots & a & a & ? \\ 0 & 1 & a & \ldots & a & a & ? \end{bmatrix}$$

At time $t = 1$ the algorithm can either allocate the server to the first site again or to another site. Consider first the case that the algorithm allocates the server to the first site again, so the algorithm collects a benefit of 1. In this case the last column is revealed (at time $t = 2$) to be all zeros, and the situation is the following.

$$\{d_{i,t}\} = \begin{bmatrix} \triangleright 0 & \triangleright 1 & \triangleright a & \ldots & a & 0 & 0 \\ 0 & 1 & a & \ldots & a & a & 0 \\ 0 & 1 & a & \ldots & a & a & 0 \end{bmatrix}$$

The total benefit that the online can collect in this case is at most $1+(L-1)a$, while the offline could have collected the whole second row, i.e., $1+La$. Therefore, the competitive ratio of the algorithm would be at least

$$r' = \frac{1+La}{1+(L-1)a} = 1 + \frac{a}{1+(L-1)a}.$$

Consider now the case that the algorithm allocates the server at time $t=2$ to a site other than the first one, so it collects a benefit of 0. Without loss of generality we may assume that the server is allocated to the second site. At time $t=2$, the last column is revealed to the algorithm, and the situation is the following.

$$\{d_{i,t}\} = \begin{bmatrix} \triangleright 0 & \triangleright 1 & a & \dots & a & 0 & 0 \\ 0 & 1 & \triangleright a & \dots & a & a & 0 \\ 0 & 1 & a & \dots & a & a & a \end{bmatrix}$$

The total benefit that the online can collect in this case is at most La, while the offline could have collected the whole third row, i.e., $1+(L+1)a$. Therefore, the competitive ratio of the algorithm would be at least

$$r'' = \frac{1+(L+1)a}{La} = 1 + \frac{a+1}{La}.$$

We conclude that the competitive ratio of any deterministic algorithm is at least $\min\{r', r''\}$. Choosing $a = (L + \sqrt{L^2+4})/2$, we get that $a^2 = 1 + La$ and hence

$$r' = r'' = 1 + \frac{a+1}{La} = 1 + \frac{1}{L} + \frac{1}{aL}.$$

Since $a < L + 1/L$, we get that $\min\{r', r''\} > 1 + \frac{1}{L} + \frac{1}{L^2+1}$, as claimed. □

This lower bound (slightly) improves over the $1+\frac{1}{L}$ straightforward lower bound. As the latter bound is the competitive ratio threshold for randomized algorithms (see Section 7), this proves that for any fixed lookahead L, the thresholds of deterministic and randomized algorithms are different.

5.2 Lower bounds for lookahead one

We prove two lower bound results for deterministic algorithms for the web server farm problem with one server and lookahead 1. The first result shows that no algorithm can achieve a competitive ratio better than 4 for this problem. This matches our algorithm from Section 4.1 that achieves a competitive ratio 4. The second result shows that in the case where the number of sites is two, no algorithm can achieve a competitive ratio better than $3\sqrt{3}/2 \approx 2.598$. This is almost tight with our algorithm from Section 6.1 that achieves a competitive ratio roughly 2.618. The proofs of the following two theorems are omitted due to space constraints.

THEOREM 6. *Consider the web server farm problem with one server and lookahead 1. For every $c < 4$, there exists an $s > 0$ such that the competitive ratio of any deterministic online algorithm for the problem with s sites is at least c.*

THEOREM 7. *Consider the web server farm problem with two sites, one server and lookahead 1. For every $c < 3\sqrt{3}/2$, the competitive ratio of any deterministic online algorithm for the problem is at least c.*

6. SPECIAL CASES

In this section we consider two special cases of the web server farm problem. The first is the case of one server and two sites with lookahead one. The second is multiple servers with lookahead one.

6.1 One server, two sites, and lookahead one

We devise a deterministic online algorithm for the web server farm problem with one server, two sites and lookahead $L=1$. The algorithm has competitive ratio $r = 1+\phi$ where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio. This ratio is nearly the best possible in this case, as we present (in Section 5) a lower bound of $\frac{3\sqrt{3}}{2} \approx 2.598$ on the competitive ratio of any deterministic algorithm for this problem. We remark that the competitive ratio that we obtain in this case is much better than in the case of arbitrary number of sites, where the best possible competitive ratio is 4.

The algorithm is based on discounting future benefits. It is similar to the algorithm presented in Section 4.1 (specialized to the case $L=1$), except that the discount factor is taken to be ϕ. The analysis for this case adds some complication to that of the general case, in order to take advantage of the limited number of sites. For example, a discount factor of 2 optimizes the analysis of Section 4.1, and then the competitive ratio is shown to be at most 4. The actual competitive ratio in this case is 3, which can be shown by an analysis similar to the one that we give here.

The following theorem states the improved competitive ratio for this case.

THEOREM 8. *The discounting algorithm with discount factor $\phi = \frac{1+\sqrt{5}}{2}$ achieves competitive ratio $r = 1+\phi \approx 2.618$ for the web server farm problem with one server and two sites.*

PROOF. We first describe the algorithm in more detail. We associate with each possible allocation its *anticipated benefit*, which combines the immediate benefit of this allocation and its future benefit. To compensate for the uncertainty of a future benefit (the algorithm might later decide not to collect these benefits), we discount it (with respect to an immediate benefit) by a factor of ϕ. In other words, an action that causes the algorithm to collect a benefit b and to have the potential to collect a benefit b' in the future has anticipated benefit of $\phi \cdot b + b'$.

Suppose that the situation of the online algorithm at some time t is as illustrated below. Assume, without loss of generality, that the allocation of the server in the previous time unit is to the first site, as denoted by the triangle. Since the lookahead is $L=1$, the algorithm knows only the demands at the current time t and at the next time $t+1$.

$$\{d_{i,t}\} = \begin{bmatrix} \dots & \triangleright x & x' & ? & ? & \dots \\ \dots & y & y' & ? & ? & \dots \end{bmatrix}$$

The algorithm decides on the allocation of the server at time t as follows. If $\phi \cdot x + x' \geq y'$ then the previous allocation is kept, i.e., the online algorithm allocates the server at time t to the first site. Otherwise, the algorithm changes the allocation and the server is allocated to the second site. Observe that this decision achieves the largest anticipated benefit for the algorithm. If it decides to stay with the allocation as before, it collects a benefit of x and will have the option

to collect at the next time $t+1$ an additional benefit x', yielding an anticipated benefit $\phi \cdot x + x'$. If the algorithm decides to change the allocation to the other site, then it collects no immediate benefit (at this point there is no way for the online algorithm to collect the benefit y), and in the next time $t+1$ it will have the option to collect a benefit y', yielding an anticipated benefit $\phi \cdot 0 + y' = y'$.

We will prove by induction on T that[2]

$$r \sum_{t \le T} ON_t + q_{T+1} \cdot ON^*_{T+1} \ge \sum_{t \le T} OFF_t + OFF^*_{T+1} \quad (4)$$

where ON_t is the actual benefit collected at time t by the online algorithm; ON^*_t is the potential benefit that the online algorithm will collect at time t if it allocates the server to the same site as at time $t-1$; OFF_t and OFF^*_t are defined similarly for an arbitrary offline algorithm; and $q_t = \phi$ if at time $t-1$ the online and the offline algorithms have allocate the server to the same site (i.e., if they have the same "starting position" for time t), and $q_t = 1$ otherwise. (Note that ON_t is either ON^*_t in case the potential benefit is gained or 0.)

Proving (4) would complete the proof, since we can assume, without loss of generality, that all the demands in the first and last $L+1$ time steps are zero, and thus obtain for the last time step T that $r \sum_t ON_t \ge \sum_t OFF_t$, as required.

The base case of the induction follows from padding the demand matrix with zeros from both sides. For the inductive step, i.e., that the induction hypothesis (4) for time T follows from that of time $T-1$, it suffices to show that

$$\begin{aligned} r \cdot ON_T &+ q_{T+1} \cdot ON^*_{T+1} - q_T \cdot ON^*_T \\ &\ge OFF_T + OFF^*_{T+1} - OFF^*_T \end{aligned} \quad (5)$$

We prove Equation (5) by verifying it over all cases. Denote the benefits at times $T, T+1$ by x, x', y, y' as above. Assume without loss of generality that at time $T-1$ the server is allocated to the first site. The total number of cases is eight, since there are two possible decisions for the online, and four possible decisions for the offline.

Consider first the cases in which the allocation of the offline at times T and $T+1$ are the same. Then at time T the offline collects the benefit OFF^*_T, and hence $OFF_T = OFF^*_T$. In addition $ON^*_T = x$, so inequality (5) simplifies to

$$r \cdot ON_T + q_{T+1} \cdot ON^*_{T+1} - q_T \cdot x \ge OFF^*_{T+1}.$$

1. $q_T = \phi, q_{T+1} = \phi$. Then both offline and online collect x. We need to show that
$$r \cdot x + \phi \cdot x' - \phi \cdot x \ge x'$$
i.e., that $x + (\phi - 1)x' \ge 0$, which clearly holds. $\surd$

2. $q_T = 1, q_{T+1} = 1$. Then offline collects y and online collects x. We need to show that
$$r \cdot x + x' - x \ge y'$$
i.e., that $\phi \cdot x + x' \ge y'$, which holds since online keeps the previous allocation. $\surd$

3. $q_T = \phi, q_{T+1} = 1$. Then offline collects x and online does not collect x. We need to show that
$$r \cdot 0 + y' - \phi \cdot x \ge x'$$
which holds since online changes allocation. $\surd$

4. $q_T = 1, q_{T+1} = \phi$. Then offline collects y and online does not collect x. We need to show that
$$r \cdot 0 + \phi \cdot y' - x \ge y'$$
i.e., that $y' \ge \frac{1}{\phi - 1} \cdot x = \phi \cdot x$, which holds since online changes allocation. $\surd$

Consider now the cases in which the allocation of the offline at times T and $T+1$ are different. Then the benefit of the offline at time T is $OFF_T = 0$. In addition $ON^*_T = x$, so inequality (5) simplifies to

$$r \cdot ON_T + q_{T+1} \cdot ON^*_{T+1} - q_T \cdot x \ge OFF^*_{T+1} - OFF^*_T$$

5. $q_T = \phi, q_{T+1} = \phi$. Then offline and online both do not collect x. We need to show that
$$r \cdot 0 + \phi \cdot y' - \phi \cdot x \ge y' - x$$
i.e., that $y' \ge x$, which clearly holds since online changes allocation. $\surd$

6. $q_T = 1, q_{T+1} = 1$. Then offline does not collect y and online does not collect x. We need to show that
$$r \cdot 0 + y' - x \ge x' - y$$
and since online changes allocation we indeed get $y' \ge \phi \cdot x + x' \ge x + x' - y$. $\surd$

7. $q_T = \phi, q_{T+1} = 1$. Then offline does not collect x and online collects x. We need to show that
$$r \cdot x + x' - \phi \cdot x \ge y' - x$$
i.e., that $2x + x' \ge y'$, which holds since online keeps the previous allocation.

8. $q_T = 1, q_{T+1} = \phi$. Then offline does not collect y and online collects x. We need to show that
$$r \cdot x + \phi \cdot x' - x \ge x' - y$$
i.e., that $\phi \cdot x + (\phi - 1)x' + y \ge 0$, which clearly holds.

□

6.2 Multiple servers with lookahead one

We present an online algorithm for the web server farm problem with multiple servers and lookahead one, as stated in the following theorem. We omit the proof of the lower bound for lack of space.

THEOREM 9. *There is a deterministic algorithm for the web server farm problem that achieves competitive ratio 2 when the number of servers k is even, and competitive ratio $c_k = 2 + 2/(2k-1)$ when $k \ge 3$ is odd. Furthermore, no online algorithm achieves a lower competitive ratio.*

PROOF. We distinguish between even and odd numbers of servers. For an even k, the algorithm splits the set of servers into two sets of equal size. The first set of servers is allocated so as to always collect the $k/2$ largest benefits on the *odd* time steps. These servers spend the even time steps in reallocation to those sites where they will be able to collect in the following time step, which is odd, the $k/2$ largest benefits. On even time steps, these servers may collect zero benefit in the worst case. The second set of servers

[2] In a sense, this is a potential function analysis. See [9].

Workload	Avail.	Opt.	Disc. $L=1$	Disc. $L=4$	Reset. $L=4$	Reset. $L=8$
ran-24	9494	7871	7750 (98.5%)	7803 (99.1%)	7484 (95.1%)	7753 (98.5%)
com1-15	4634	3904	3899 (99.9%)	3901 (99.9%)	3875 (99.3%)	3894 (99.8%)
com1-25	4634	4634	4634 (100 %)	4634 (100%)	4629 (99.9%)	4634 (100%)
com2-15	7265	4227	4217 (99.8%)	4218 (99.8%)	4163 (98.5%)	4183 (99.0%)
com2-25	7265	6013	6007 (99.9%)	6009 (99.9%)	5966 (99.2%)	5990 (99.6%)

Table 2: Experimental results

is scheduled in a symmetric way. It collects the $k/2$ largest benefits on the even time steps and spends the odd time steps on reallocation to the best sites for the following time step which is even. We conclude that at each time t, the total benefit that the k servers collect is at least as large as the $k/2$ largest benefits at this time t, which is clearly at least half the benefit that an optimal offline algorithm can collect at this time t. Therefore, the competitive ratio of this algorithm is at most 2.

For an odd k, the algorithm splits the servers into three groups. Two groups are of size $(k-1)/2$ each, and the third group contains one server. The first group of servers collects the $(k-1)/2$ largest benefits on the odd time steps. The second group collects the $(k-1)/2$ largest benefits on the even time steps. Let F_t denote the $(k+1)/2$ largest benefit in step t. The last server tries to collect F_t on each time step, as follows. On the step t the server either collects F_t or it moves to the site q such that $d_{q,t+1} = F_{t+1}$. The server collects F_t if $2F_t > F_{t+1}$; otherwise the server moves to site q. If the server collects F_t on some step t, it sacrifices the step $t+1$ on a move to the $\frac{k+1}{2}$th best site q for step $t+2$, i.e., $d_{q,t+2} = F_{t+2}$.

For the sake of analysis we split time into phases: the first phase starts at time step 0 (by convention we assume that benefits at this time step are equal to zero). A phase ends at time step $t+1$ if the server from the third group collects F_t in step t. The next phase starts at step $t+2$, and so on. Notice that the server from the third group always starts the phase allocated to the $\frac{k+1}{2}$th best site. Consider the first phase. We claim that $F_t \geq 1/4 \sum_{i=1}^{t+1} F_i$. Indeed, $2F_i \leq F_{i+1}$ for $i = 1, \ldots, t-1$ and $2F_t > F_{t+1}$. Therefore

$$\sum_{i=1}^{t+1} F_i < F_t \sum_{i=1}^{t+1} 1/2^{t-i} \leq 2F_t \sum_{j=0}^{\infty} 1/2^j = 4F_t.$$

For $p \leq k$, if $S(\tau, p)$ is a sum of the p largest benefits in step τ, then $\sum_\tau S(\tau, p) \geq p/k \sum_\tau OFF_\tau$ and

$$\begin{aligned}
\sum_{\tau=1}^{t+1} ON_\tau &\geq \sum_{\tau=1}^{t+1} S\left(\tau, \frac{k-1}{2}\right) + F_t \\
&\geq \frac{3}{4} \sum_{\tau=1}^{t+1} S\left(\tau, \frac{k-1}{2}\right) + \frac{1}{4} \sum_{\tau=1}^{t+1} S\left(\tau, \frac{k+1}{2}\right) \\
&\geq \left(\frac{3(k-1)}{8k} + \frac{(k+1)}{8k}\right) \sum_{\tau=1}^{t+1} OFF_\tau \\
&= \frac{2k-1}{4k} OFF = c_k^{-1} OFF\ .
\end{aligned}$$

The same proof holds for any phase, and therefore we have proved that our algorithm is c_k-competitive for odd $k \geq 3$. □

7. RANDOMIZED ONLINE ALGORITHMS

In this section we derive tight bounds on the competitive ratio of randomized online algorithms with lookahead L. These bounds are stated in the following theorem. We remark that our randomized algorithm requires only $O(\log L)$ random bits, independent of the input length.

THEOREM 10. *There is a randomized online algorithm for the benefit task system problem with competitive ratio $1 + 1/L$. Furthermore, no randomized algorithm achieves a lower competitive ratio, even for the web server farm problem with one server.*

PROOF. The idea behind the algorithm is that with lookahead L, we can compute an optimal path for L time steps. However, we may have to make a poor move, in which we collect little or no benefit, in order to reach the first state on this path. After following this path, we can again choose the best state to move to during the next time step, possibly without collecting any benefit at all, but ensuring the best starting point for the next L steps. We refer to this as the "resetting" algorithm, since it operates in stages, after each of which it resets to an optimal state for the upcoming stage. The online algorithm will collect at least as much benefit as the optimal offline algorithm on every step except the resetting steps. Notice that there are $L+1$ possible "phases": phase j means (potentially) giving up all benefit during steps $i \cdot (L+1) + j$ for each i, where $0 \leq j \leq L$. Since the sets of steps whose benefit is given up is these phases are mutually disjoint, the total benefit lost in all phases is at most the offline benefit. Randomizing over these choices, the online algorithm loses in expectation only a $1/(L+1)$ fraction of the offline benefit, and the competitive ratio is at most $\frac{1}{1-1/(L+1)} = 1 + 1/L$.

For any $\epsilon > 0$, we show a lower bound of $(L+1)/(L+\epsilon)$ for the one-server problem. Our randomized adversary generates a demand matrix $\{d_{i,t}\}$ with $\lceil 1/\epsilon \rceil$ rows and $L+2$ columns (which can be repeated indefinitely), and a benefit matrix of all ones. The first column of the demand matrix contains all zeroes, the next L columns contain all ones, and exactly one randomly chosen row in the last column contains a one and the rest contain zeroes. The optimal choice is for the server to move during the first step to the row containing the one in column $L+2$, and to stay in that row for $L+1$ steps, collecting a total benefit of $L+1$. Any randomized online strategy stands only an ϵ chance of choosing this row, and with probability at least $1-\epsilon$ must either miss the benefit in this column, or forgo benefit in some earlier column in order to collect it. Thus the expected benefit collected by the randomized online algorithm is at most $L + \epsilon$. □

8. EXPERIMENTAL RESULTS

We have implemented the offline optimal, discount, and intermittent reset algorithms for the web server farm problem and measured their performance. We used a randomly generated input, and inputs generated from commercial retail web server logs. The online algorithms perform very well on these inputs, both in cases of overloaded and underloaded servers. In fact, the discount algorithm with lookahead one and the intermittent reset algorithm with lookahead eight match the optimal offline performance in the case of underloaded servers. (This is not surprising, since it can be shown that if the optimal offline algorithm is able to collect all the available benefit, then the greedy algorithm with lookahead one achieves this optimal benefit.) The discount algorithm performs better, with less lookahead, than the intermittent reset algorithm, and is within 0.2% of optimal on all the commercial trace instances, even with lookahead one. We use the (untuned) value of 2 as the discount factor.

The random problem instance has 3 customers and 400 time steps. Demands are generated uniformly between 0 and 16, and revenues uniform between 0 and 2.

The commercial traces were collected from two different retail sites, which will remain unnamed due to confidentiality constraints. One of these is separated into "browse" and "buy" transactions. We treat these separately (i.e., so that we have three different "customers"), and assign revenue three times as great to the "buy" transactions since they represent actual monetary transactions. Each trace represents the requests arriving at a web server over a 24-hour period. We separate these into 288 five-minute buckets, i.e., our time unit is five minutes. We scale the inputs using realistic server rates, which results in demands ranging in the low tens of servers for two customers, and up to two servers for the third (the one that handles the "buy" transactions).

Table 2 shows the algorithms' performance. The column labeled "Avail." denotes the total available revenue for the problem instance, i.e., the revenue that would be collected with an infinite number of servers. The next column, labeled "Opt", shows the revenue collected by the optimal offline algorithm. The next four columns show the revenue collected by the discount algorithm and the intermittent reset algorithm with differing amounts of lookahead (one and four for the discount algorithm, and four and eight for the intermittent reset algorithm). All values are rounded to the nearest integer. The "Workload" column indicates the trace used (random and the two commercial traces, com1 and com2) and the number of servers.

Acknowledgements.

We thank Maria Minkoff for helpful discussions and insights.

9. REFERENCES

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] B. Awerbuch, Y. Azar, A. Fiat and F.T. Leighton, "Making Commitments in the Face of Uncertainty: How to Pick a Winner Almost Every Time". *Proc. 28th ACM Symp. on Theory of Computing (STOC)*, pp. 519–530, 1996.

[3] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor and B. Schieber, "A unified approach to approximating resource allocation and scheduling". *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pp. 735–744, 2000.

[4] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour and B. Schieber, "Bandwidth allocation with preemption". *SIAM Journal on Computing*, 28 (1999), pp. 1806–1828.

[5] A. Bar-Noy, S. Guha, J. Naor and B. Schieber, "Approximating the throughput of multiple machines under real-time scheduling". *Proc. 31st ACM Symp. on Theory of Computing (STOC)*, pp. 622–631, 1999.

[6] A. Bar-Noy, Y. Mansour and B. Schieber, "Competitive dynamic bandwidth allocation". *Proc. 17th ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (PODC)*, pp. 31–39, 1998.

[7] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha and F. Wang, "On the competitiveness of online real-time task scheduling". *Proc. 12th IEEE Symp. on Real Time Systems*, pp. 106–115, 1991.

[8] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier and D. Shasha, "Online scheduling in the presence of overload". *Proc. 32nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pp. 101–110, 1991.

[9] A. Borodin and R. El-Yaniv, *On-Line Computation and Competitive Analysis*. Cambridge University Press, 1998.

[10] A. Borodin, N. Linial and M.E. Saks, "An optimal on-line algorithm for metrical task system". *J. ACM*, 39(1992), pp. 745–763.

[11] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial optimization*. John Wiley & Sons Inc., New York, 1998.

[12] J.A. Garay, I.S. Gopal, S. Kutten, Y. Mansour and M. Yung, "Efficient online call control algorithms". *Proc. 2nd Israel Conf. on Theory of Computing and Systems*, pp. 285–293, 1993.

[13] A.K. Iyengar, Mark S. Squillante and L. Zhang, "Analysis and Characterization of Large-Scale Web Server Access Patterns and Performance". *World Wide Web*, 2(1999), pp. 85–100.

[14] G. Koren and D. Shasha, "D-over: An optimal on-line scheduling algorithm for overloaded real-time systems". *SIAM Journal on Computing*, 24 (1995), pp. 318–339.

[15] R.J. Lipton and A. Tomkins, "On-line interval scheduling". *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 302–311, 1994.

[16] M.S. Manasse, L.A. McGeoch and D.D. Sleator, "Competitive algorithms for server problems". *J. of Algorithms*, 11 (1990), pp. 208–230.

[17] C.N. Potts and L.N.Van Wassenhove, "Integrating scheduling with batching and lot sizing: A review of algorithms and complexity". *J. of the Operational Research Society*, 43 (1992), pp. 395–406.

[18] M.S. Squillante, D.D. Yao and L. Zhang, "Web Traffic Modeling and Web Server Performance Analysis", *Proc. 38th IEEE Conf. on Decision and Control*, pp. 4432–4437, 1999.

[19] `http://www.ibm.com/services/webhosting/-full_services.html`

[20] `http://www.exodus.com/managed_services/-content_distribution/index.html`