On-line Admission Control and Packet Scheduling with Interleaving

Juan A. Garay Joseph (Seffi) Naor Bülent Yener Peng Zhao

Abstract— **This paper presents a comprehensive study of the effect of job interleaving by preemption on the throughput of a single server where requests arrive with a given processing time and slack. The problem is to decide which requests to serve so as to maximize the server's utilization. This simple model captures many situations, both at the application (e.g., delivery of video) as well as at the network/transmission levels (e.g., scheduling of packets from input to output interface of a switch). The problem is on-line in nature, and thus we use competitive analysis for measuring the performance of our scheduling algorithms. We consider two modes of operation: with and without commitment, and derive upper and lower bounds for each case. Since competitive analysis is based on the worst-case scenario, the average-case performance of the algorithms is also examined by a simulation study.**

I. INTRODUCTION

This paper investigates the effect on the utilization of a single server of job interleaving by preemption, i.e., the interruption and later continuation of a job. We perform the investigation at an abstract level, where requests ("jobs") arrive at a resource with given processing times and deadlines, and where the benefit is measured by the resource utilization, or of time that the resource is busy serving the requests—i.e., the so-called "uniform value density" model.

This abstract model captures many situations, both at the application and at the network/transmission layers. Consider a server which is a communication channel for the delivery of video or file transfers, where streams, or flows can be interrupted and later resumed without disruption of the application's result, as long as their deadline is met. An example is data over cable service which is expected to support several QoS levels for data traffic [2]. Data over Cable Service Interface Specification (DOCSIS) proposes dividing an upstream channel into *minislots* [14], where each QoS class corresponds to a queue with different minislot allocation. The head end or CM terminating system (CMTS) can slice the upstream channel using the fine granularity of minislots and interleaves the MAC frames at the minislot level [2].

At the network level, consider legacy routers, which are limited to using simple schedulers such as FIFO. As the circuit technology advances towards enabling programmable and faster architectures, more advanced algorithms are to be deployed in the next generation network elements while ensuring line packet processing. Thus, a potential application is the admission control and scheduling of flows assuming negligible transmission time for a job [12], [18]. There is already progress in this direction; for example, job interleaving is possible in the packet processing and switching mechanism deployed in programmable network processors (e.g., Intel's IXP 1200 Network Processor [17]) and at high performance switches (e.g., Lucent's Atlanta chip set [1]). For example, in Intel's Network Processor variable-size packets received in the input port are broken into fixed-size (64 Byte) cells, and then cells from different packets can be interleaved over the crossbar switch to the output ports.

We use *slack* to denote the amount of time between the job's arrival and the latest time at which it could be started in order to meet the job's deadline.[1] The slack captures certain aspects of Quality-of-Service (QoS) provisioning. As in [11], in the analysis we assume that the slack of a job and its processing time are related; specifically, if an incoming job J has processing time $p(J)$, then it has a slack of at least $\kappa \cdot p(J)$, $\kappa > 0$. This is a natural requirement for admission control, and this or similar restrictions have been considered in the literature [6], [10], [11], [16]. An example is video-on-demand, where users specify a window of time during which delivery is acceptable; it is reasonable to assume that the wait time for a full video-on-demand service (e.g., 15 min.) would be proportional to the duration of the movie (2 hours).

Since in many applications the distribution of incoming requests is not known in advance, simple schedulers such as FIFO and round-robin might not guarantee an adequate

[1] Other terms used in the literature are *delay*, *laxity*, and *wait time*.

Juan A. Garay, Bell Laboratories, Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974. E-mail: garay@research.bell-labs.com.

Joseph (Seffi) Naor, Computer Science Dept., Technion, Haifa 32000, Israel. naor@.cs.technion.ac.il.

Bülent Yener, Bell Laboratories, Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974. yener@research.bell-labs.com.

Peng Zhao, Department of Communication Systems EPFL, Lausanne, Switzerland. peng.zhao@epfl.ch.

performance. We are interested in the design of **robust** strategies for the scheduling of a resource with preemption. Thus, we make no assumptions on the distribution of requests, and assume that the algorithm has no knowledge about any job until the job arrives, at which point it learns its processing time and deadline. We analyze the performance of such an *on-line* algorithm using *competitive analysis* [4], the schedule produced by the algorithm to the optimal schedule produced by an algorithm with full knowledge of incoming jobs. The ratio between the performance of the optimal schedule and the performance of the on-line algorithm is called the *competitive ratio*. A deterministic on-line algorithm is *c-competitive* if its ratio is at most c for every possible request sequence.

We consider the following two scheduling modes: *with commitment*, where once accepted the jobs must be satisfied to completion, and *without commitment*, where accepted requests may be abandoned, and present upper and lower bounds for each case. We also characterize and establish the complexity of determining schedules that are feasible. This contrasts with models that only assume feasible request sequences (e.g., those that assume reservation or rate negotiation in advance).

A. *Prior and related work*

On-line admission control in various models has been studied extensively in the algorithms community. The effect of deadlines (delays, slack) on on-line admission control was investigated in [10], [6], [11], [16], [9]. In particular, Goldwasser [11] studies the same model as in this paper, but without preemption, and shows matching upper and lower bounds of $2 + \frac{1}{\kappa}$ on the competitive ratio of deterministic algorithms, where $\kappa > 0$ is the minimum ratio between the slack and the processing time of all jobs in the request sequence. In part, our work is an extension of [11] to the case where preemption is allowed.

Two other closely related works are those by Baruah and Haritsa [6] and by DasGupta and Palis [9], which also allow for preemption. [6] does not consider the model *with commitment*, and only considers overload scenarios. The competitive ratio, used in our paper, is a more general measure. It considers all possible scenarios, including, in particular, those where the system is in "underload" (i.e., situations where it is possible to make all the tasks in the system meet their deadlines). There are two consequences of this approach: First, in underload, our algorithm for the case without commitment, is *optimal*, while the algorithm presented in [6] is not; second, in the case of overload, there might be intervals where also the optimal algorithm is idle, which is a point omitted in the analysis in [6]. The lower bound proved in [6], when translated to our notation, is greater than $(\kappa + 1)/(\kappa + 2)$, which is strictly greater than our lower bound of $\kappa/(\kappa + 1)$. Thus, we are able to provide a tighter lower bound. Independently, DasGupta and Palis [9] perform worst-case analysis on essentially the same model as ours, in the context of machine scheduling. We contrast their work in the next section.

Naturally, admission control and packet scheduling have also been studied extensively in the networking community (references omitted due to lack of space). With the objective of providing QoS guarantees in packet-switched networks, these works are typically concerned with designing scheduling policies that minimize **delay** (and, sometimes, buffer) requirements, as opposed to our goal of throughput optimization. Other typical assumptions in these works are the type of source of packets, such as "regulated" (dual leaky bucket), or "stochastic," and that of advanced reservations—delay bounds and sending rates are negotiated beforehand. One model related to ours is that of [12], where the single resource (switch, multiplexer) has a finite number of input links sharing one output link with a given rate. Under commonly-accepted traffic constraints (that the flows coming in from the input links satisfy the condition that, roughly, the sum of their rates is less than that of the output link), they show that the popular Earliest Deadline First (EDF; see, e.g., [8]) scheduling strategy has the largest "schedulable region," or variety of delay vectors, in the constrained-traffic model. In contrast, in our case the traffic is arbitrary, i.e., there are no reservations and the incoming traffic is not limited by the stability constraint. We do impose the slack proportionality constraint mentioned above, and again, our goal is to maximize the utilization of the resource. In fact, one interesting direction for future work is the combination of the optimization goals—delay and throughput.

B. *Our results*

The contributions of this paper are as follows:

Scheduling with commitment. We give a simple proof that a simple greedy algorithm achieves a competitive ratio of $2 + \frac{1}{\kappa}$, and prove a lower bound on the competitiveness of any deterministic on-line algorithm of $1 + \frac{1}{\kappa}$. Dasgupta and Palis [9] give a more elaborate proof that the competitive ratio of the algorithm is actually $1 + \frac{1}{\kappa}$.

Scheduling without commitment. We distinguish two cases, depending on the value of κ: for large values of κ the competitive factor of $2 + \frac{1}{\kappa}$ proved for the commitment case carries over (as well as the bound of [9]), while for small values of κ we present an algorithm with a competitive ratio of 5.82843. Thus, the competitive ratio for this case is the minimum between the two ratios. A lower bound of 4 follows from the related task scheduling problem in a uni-processor setting.

New approximation algorithm. The algorithm for scheduling without commitment needs to solve a subproblem that can be formulated as the Generalized Knapsack problem. We present a 2-approximation algorithm for this problem, which is based on the *local ratio* technique. Fully polynomial approximation schemes are known for this problem [15]; however, our algorithm is simpler, easy to implement (see Section VI), and it uses novel ideas.

Simulation study. We have designed and implemented in Java a discrete event simulator to study the performance of the on-line algorithms under several distributions on the job arrivals and job sizes. Under light load the on-line algorithms perform as well as the off-line algorithm, while in heavy load their performance approaches the worst-case bounds. We also study how variations of κ impact the algorithms' performance. Our results suggest that as κ increases the performance ratio of the on-line algorithms improve.

C. Organization of the paper

The rest of the paper is organized as follows: Section II presents the model and notation. Section III introduces the basic definitions and properties used in the rest of the paper. Sections IV and V present the results on the commitment and the non-commitment models, respectively. The simulation study is presented in Section VI. A summary is presented in Section VII.

II. Model and Notation

We divide time into slots of fixed duration. During each time slot, a fixed number of bytes of a job can be transmitted, or processed.[2] At every time slot, only one job can be processed, i.e., we have a single machine (using scheduling theory terminology).

A job J is defined by a triple $\{a(J), p(J), d(J)\}$, where $a(J)$ is the *arrival* time, $p(J)$ is the *processing* (transmission) time, and $d(J)$ is the *deadline*, i.e., the time by which the job has to finish its processing. We assume that $d(J) - a(J) \geq p(J)$. The interval of time $[a(J), d(J)]$ is the *window* of time, denoted by $w(J)$, in which job J has to finish its processing. Let $s(J) = d(J) - a(J) - p(J)$ be the *slack* of job J. We assume that $s(J) \geq \kappa p(J)$, where $\kappa > 0$ defines the minimum slack of every job in a request sequence. For a set of jobs $\mathcal{J}$, denote by $p(\mathcal{J})$ the sum of the processing times of the jobs belonging to $\mathcal{J}$.

A scheduling algorithm is called *preemptive* if it can interrupt a job and later on resume its processing. This work considers only preemptive schedulers. In our scheduling model, requests appear *on-line*, i.e., job J appears at time $a(J)$, and upon arrival of a job, the processing time and deadline are revealed to the scheduling algorithm. The scheduling algorithm provides admission control, i.e., upon arrival of a job it decides whether to *accept* it or *reject* it. No knowledge of future arrivals of jobs is available, and the scheduling algorithm has to make its decisions (accept or reject a new job) based on the parameters of the new job and the current state of the system. A job that was accepted by the scheduling algorithm, but has not finished its processing, is called an *active* job. For each active job J, the scheduling algorithm maintains its *remaining processing time*, denoted by $r(J)$. Initially, $r(J) = p(J)$; after each time slot in which J is processed, the value of $r(J)$ is decreased by one. When $r(J) = 0$, then the job is said to be *completed*.

We consider two scheduling models: with commitments and without commitments. In the first model, once a job is accepted by the scheduling algorithm, it must be served to completion. In the second model, the scheduling algorithm may delete a job that was accepted earlier. However, a job that is deleted does not generate any profit.

The *profit* of a scheduling algorithm is the sum of the processing times of the jobs that have met their deadlines, that is completed their processing before their deadline. We measure the quality of scheduling algorithms using competitive analysis, which compares the profit of the algorithm to that of an ideal (off-line) algorithm OPT that has full knowledge of the future and thus can make optimal decisions. We say that a deterministic on-line algorithm is *c-competitive* if its ratio is at most c for every possible request sequence.

III. Preliminaries

A. Characterizing a feasible schedule

Suppose that jobs $J_1, \ldots, J_n$, characterized by $\{a_i, p_i, d_i\}$, $1 \leq i \leq n$, are to be scheduled. We define necessary and sufficient conditions for these jobs to be scheduled in the preemptive model. (These are well-known results which we summarize here for completeness.) Without loss of generality, we can assume that $a_1 \leq a_2 \leq \cdots \leq a_n$. We begin with a simple case.

Lemma 1: Suppose that $d_1 \leq d_2 \leq \cdots \leq d_n$. The following are necessary and sufficient conditions for scheduling $J_1, \ldots, J_n$:

$$\begin{aligned} p_1 &\leq d_1 - a_1 \\ p_1 + p_2 &\leq d_2 - a_1 \\ \vdots \quad & \quad \vdots \\ p_1 + p_2 + \ldots + p_i &\leq d_i - a_1 \\ \vdots \quad & \quad \vdots \end{aligned}$$

[2] It is known that job size creates fairness issues in scheduling. Thus, the service unit in scheduling algorithms such as the weighted fair queuing (WFQ) is number of bytes instead of jobs per time slot.

$$p_1 + p_2 + \ldots + p_n \quad \leq \quad d_n - a_1$$

Proof: The proof is by induction on n, the number of jobs. For $n = 1$ the proof is immediate. For the inductive step, schedule job J_n in the interval $[d_n - p_n, d_n]$. Let k be the highest index of a job such that $d_k \leq d_n - p_n$. For all jobs J_i, for which $i > k$ (and $i \neq n$), define $d'_i = d_n - p_n$. Since $p_1 + p_2 + \ldots + p_n \leq d_n - a_1$, we get that for all $i > k$ (and $i \neq n$), $p_1 + p_2 + \ldots + p_i \leq d'_i - a_1$. Hence, we can apply the induction hypothesis to $J_1, \ldots, J_{n-1}$ and schedule them in the interval $[a_1, d_n - p_n]$. ∎

Note that the above proof implies that in the special case where $d_1, \ldots, d_n$ is a monotonically non-decreasing sequence, each job can be scheduled in consecutive time slots.

In general, $d_1, \ldots, d_n$ may not be a monotonically non-decreasing sequence. In this case, the following greedy algorithm, known in the literature as EDF (Earliest Deadline First), can test whether $J_1, \ldots, J_n$ can be scheduled together. Define τ to be the latest deadline, i.e., $\tau = \max_i d_i$. The schedule S is defined as follows: Let t range between a_1 and τ; at each point of time t, S schedules the job that has the earliest deadline among the jobs that have positive remaining service time at t. It is not hard to see that EDF succeeds in scheduling all the jobs if and only if $J_1, \ldots, J_n$ can be scheduled together.

B. The local ratio technique

The *local ratio* technique for approximation algorithms was first developed by Bar-Yehuda and Even [5] and later extended by Bafna, Berman, and Fujito [3]. We briefly outline this technique. Let (I, w) be an instance of an optimization problem, where I is a combinatorial structure (e.g., graph), and w is a non-negative weight function defined over I (e.g., on edges or vertices). The goal is to minimize a linear objective function $w \cdot \mathbf{x}$, where $\mathbf{x}$ is required to satisfy certain feasibility constraints which we denote by $\mathcal{F}$. (Typically, $\mathbf{x}$ specifies which edges, or vertices, are chosen to the solution.) A feasible solution $\mathbf{x}$ (satisfying $\mathcal{F}$) for (I, w) is said to be an r-approximate solution if $w \cdot \mathbf{x}$ is no more than r times the value of the optimal solution for (I, w).

Theorem 2 (Local Ratio) Let w_1 and w_2 be a decomposition of w into non-negative weight functions such that $w = w_1 + w_2$. Let $\mathbf{x}$ be a feasible solution satisfying $\mathcal{F}$. Suppose that $\mathbf{x}$ is an r-approximate solution with respect to (I, w_1) and (I, w_2). Then, $\mathbf{x}$ is an r-approximate solution with respect to (I, w).

An algorithm that uses the local ratio technique proceeds as follows. Initially, the solution is empty. All elements of (I, w) on which the weight function w is zero are taken to the solution. Now, the idea is to find a decomposition of w such that w_1 is an "easy" weight function in some respect, e.g., any solution which is minimal with respect to containment would be a good approximation to the optimal solution of (I, w_1). We define $w_2 = w - w_1$. Then, the local ratio algorithm continues recursively on the instance (I, w_2). We assume inductively that the solution returned recursively for the instance (I, w_2) is a good approximation and need to prove that it is also a good approximation for (I, w). As implied by Theorem 2, that requires proving that the solution returned recursively for the instance (I, w_2) is also a good approximation for the instance (I, w_1). This step is the "heart" of the proof; in certain cases, this proof may require a clean-up phase in which unnecessary elements are deleted from the solution, so that it becomes minimal with respect to containment.

IV. SCHEDULING WITH COMMITMENT

We describe our algorithm for scheduling in the model with commitments. In this model, we assume that if the algorithm accepts a request, then it must process it so that it meets its deadline. We define a scheduling algorithm for this setting, GREEDY, which always accepts a job if it can complete it, given the set of jobs to which the algorithm has already committed. That is, the new job can meet its deadline without violating the deadline of any job already in the system. This can be tested using the feasibility condition as stated in Lemma 1. Either, the processing time of the new job fits into the slack of some of the jobs already accepted into the system, or the sum of the remaining processing time of the accepted jobs fits into the slack of the new job.

A. Analysis of the algorithm

Consider the interval of time $\mathcal{I}$ that contains all the job requests considered by algorithm GREEDY. Interval $\mathcal{I}$ begins with the arrival of the first request and finishes with the latest deadline among all jobs in the request sequence. Consider an optimal schedule OPT, and without loss of generality eliminate from $\mathcal{I}$ all the time slots in which both OPT and GREEDY are idle. We partition $\mathcal{I}$ into blocks $X_1, \ldots, X_\ell$ (X-blocks) and $Y_1, \ldots, Y_\ell$ (Y-blocks), and block Y_i follows block X_i for all i. The X-blocks contain all time slots in which GREEDY is active, and the Y-blocks contain all time slots in which GREEDY is idle.

Claim 3: A Y-block cannot contain job arrivals.

Proof: The proof is immediate, as otherwise GREEDY would have accepted the request, since it is idle during the Y-blocks. Thus, yielding a contradiction. ∎

Theorem 4: For $\kappa > 0$, the competitive ratio of GREEDY is at most $2 + \frac{1}{\kappa}$.

Proof: First note that the sets of requests contained in each X-block are disjoint. Therefore in order to compute the competitive ratio of the algorithm we can restrict our attention to (X_i, Y_i) pairs of blocks and bound the ratio between the components without any interference across pairs. (We can assume that OPT is active in both the X-blocks and the Y-blocks.)

Consider a job with processing time p and window w that has arrived during X_i and was not accepted by GREEDY. Clearly, it holds that $|Y_i \cap w| \leq p-1$, since otherwise GREEDY could have accepted it and successfully finished it.

More specifically, let $I_1, I_2, \cdots, I_k$ be the set of jobs that arrived in X_i, were rejected by GREEDY, and were processed by OPT in Y_i. Suppose they have respective deadlines $d_1, d_2, \cdots, d_k$ such that $d_1 \leq d_2, \cdots \leq d_k$, and processing times $p_1, p_2, \cdots, p_k$. Let t_0 and t_1 denote the beginning and end of Y_i, respectively. It holds that $d_i - t_0 \leq p_i - 1$, $1 \leq i \leq k$. If d_k is earlier than t_1 and later than t_0, then we split Y_i at time slot d_k into two blocks, Y_i' and Y_i''. Otherwise, either $Y_i' = Y_i$ and Y_i'' is empty (if $d_k \geq t_1$), or Y_i' is empty and $Y_i'' = Y_i$ (if $d_k \leq t_0$). Clearly, $|Y_i'| \leq p_k - 1$.

On the other hand, since the window of job I_k, $w(I_k)$, satisfies $w(I_k) \geq (\kappa+1)p_k$, and I_k arrived during X_i, we get $|X_i| \geq \kappa p_k + 1$. The profit of GREEDY is $|X_i|$, while the profit of OPT is $|X_i| + |Y_i'| + |Y_i''|$. Thus,

$$\frac{|X_i| + |Y_i'|}{|X_i|} \leq \frac{(\kappa p_j + 1) + (p_j - 1)}{\kappa p_j + 1} < 1 + \frac{1}{\kappa}.$$

Observe that $\sum_{i=1}^{\ell} |Y_i''| \leq \sum_{i=1}^{\ell} |X_i|$, since OPT could only have processed jobs that GREEDY accepted in the Y_i''-blocks. Therefore,

$$\sum_{i=1}^{\ell} \frac{|X_i| + |Y_i'| + |Y_i''|}{|X_i|} < 2 + \frac{1}{\kappa},$$

yielding that the competitive ratio of GREEDY is no worse than $2 + 1/\kappa$. ■

We note that [9] have given an elaborate proof that the competitive factor is actually $1 + 1/\kappa$.

B. Lower bound

We now show a lower bound for the competitive ratio of any deterministic algorithm of $(1 + \frac{1}{\kappa})$. Recall that since this is a worst-case analysis and there are no assumptions on the distribution of requests, we (the adversary) have the freedom to come up with any request sequence, however "unnatural," that would induce a bad performance by the algorithm.

Theorem 5: For a request sequence, let $\kappa > 0$ be the minimum ratio between the window of a job and its processing time, taken over all jobs in the sequence. Then, the competitive ratio of any deterministic scheduling algorithm with commitment is at least $(1 + \frac{1}{\kappa}) - \delta$, where $\delta > 0$.

Proof: (Sketch) We study the behavior of a deterministic algorithm ALG when presented with the following request sequence by the adversary. First, at time 0, the adversary generates a batch of jobs of very small size,[3] say, ϵ, such that:

1. their deadline is x, and
2. the sum of their processing time adds up to x.

Note that for a given x, ϵ can always be chosen so as to satisfy the minimum slack constraint.

As these could be the only jobs in the execution, ALG must take a number of jobs that add up to at least $x - \epsilon$, since currently the minimum ratio between the window and the processing time of all jobs is x/ϵ. Suppose, without loss of generality that ALG accepted all jobs. (Otherwise, set $x = x - \epsilon$.)

The adversary accepts all the small jobs that ALG accepted, except for **one**, and immediately after, generates a job J of unit size and deadline $d(J) = 1 + x - \epsilon$. Clearly, ALG cannot fit this job while the adversary can accept the job.

The value of κ is determined by job J and therefore $\kappa = x - \epsilon$. The competitive ratio of ALG is

$$\frac{1 + x - \epsilon}{x} = \frac{1 + \kappa}{\kappa + \epsilon} = 1 + \frac{1}{\kappa} - \delta \quad where \quad \delta = \frac{\epsilon \cdot (\kappa + 1)}{\kappa \cdot (\kappa + \epsilon)}.$$

Note that δ tends to zero when ϵ goes to zero. This completes the proof of the lower bound. ■

V. SCHEDULING WITHOUT COMMITMENT

We describe our algorithm for scheduling in the model without commitments. In this model, the algorithm is allowed to delete a job that was previously accepted. However, the algorithm accrues profit only from jobs that completed their processing before their deadline.

We first observe that any bound on the competitive ratio that was proved in the model with commitments carries over to the model without commitments. The reason is that commitment is a property of on-line algorithms, and the profit of the optimal off-line algorithm (for a given request sequence) is independent of it. Therefore, for large values of κ we will use algorithm GREEDY. In the sequel, we describe a new algorithm, called ALG, that achieves better competitive factors for small values of κ. The exact threshold for "large" and "small" can be inferred from the

[3]For simplicity, in this proof sketch we consider a continuous model, where the sizes of jobs can be arbitrarily small, and leave the details of scaling up to the smallest unit, as well as "ceilings" and similar round-up considerations to the full version of the paper.

bounds in Theorem 8. We assume that the value of κ is known in advance to the algorithm.

At any given point of time there is a set of jobs $\mathcal{J}$ which ALG has already accepted and which can be scheduled together. Upon arrival of a new job J at time t, there are two cases to consider.

1. If job J can be scheduled together with the jobs in $\mathcal{J}$, then ALG accepts J.
2. Otherwise: (i) find a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ such that if $\mathcal{J}'$ is deleted, then job J can be accepted. (ii) Let $\alpha = 1 + \sqrt{2}$. If $p(J) \geq \alpha \cdot p(\mathcal{J}')$, then accept job J and delete the jobs belonging to $\mathcal{J}'$.

A. *Implementing the algorithm*

We now elaborate on the steps of ALG. Let $\mathcal{J} = J_1, \ldots, J_n$ and suppose that $d(J_i) \leq d(J_{i+1})$ for all i, $1 \leq i \leq n-1$. We can assume without loss of generality that $t = 0$.

Step (1) In this step we need to check whether job J can be scheduled together with the jobs belonging to $\mathcal{J}$. Suppose that $d(J_i) \leq d(J) < d(J_{i+1})$ (where $d(J_{n+1})$ is defined to be infinity). The necessary and sufficient conditions for scheduling job J with $\mathcal{J}$ are given by Lemma 1. Since the jobs in $\mathcal{J}$ can be scheduled together, this means that the algorithm needs to verify that:

$$\begin{aligned} \sum_{j=1}^{i} r(J_j) + p(J) &\leq d(J) \\ \sum_{j=1}^{i} r(J_j) + p(J) + \sum_{j=i+1}^{k'} r(J_j) &\leq d(J_{k'}) \\ \forall\, k', \text{ s.t. } i+1 \leq k' \leq k & \end{aligned}$$

Step (2) In this step we need to find a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ such that job J can be scheduled with $\mathcal{J} - \mathcal{J}'$. Clearly, in general, there may be many choices for the set $\mathcal{J}'$. We note that our analysis of the competitive factor of the algorithm in Section V-B does not depend on the specific choice of $\mathcal{J}'$, as long as $\mathcal{J}'$ is deleted if and only if $p(J) \geq \alpha \cdot p(\mathcal{J}')$. Nevertheless, the natural choice for $\mathcal{J}'$ is a set that minimizes $p(\mathcal{J}')$, among all possible choices. Unfortunately, finding an optimal set is an NP-hard problem, as indicated in the sequel. In what follows we present a 2-approximation algorithm for this problem that uses the local ratio technique. Note that even though in this paper the profit of a job is equal to its processing time, this does not hold for optimizing $\mathcal{J}'$, since the processing time of a job here is the remaining processing time while the profit is the original processing time of the job.

Let us assume that job J is taken by the schedule. This is equivalent to updating the deadlines of the jobs in $\mathcal{J}$ as follows. (The equivalence follows from the proof of Lemma 1 that if a set of jobs having the same release time can be scheduled together, then each job can be scheduled consecutively.)

$$d'(J_i) = \begin{cases} d(J_i) & \text{if } d(J_i) \leq d(J) - p(J), \\ d(J) - p(J) & \text{if } d(J) - p(J) \leq d(J_i) \leq d(J), \\ d(J_i) - p(J) & \text{otherwise.} \end{cases}$$

Note that it still holds that $d'(J_i) \leq d'(J_{i+1})$ for all i, $1 \leq i \leq n-1$. Let $\mathcal{J}'' = \mathcal{J} - \mathcal{J}'$. Recall the necessary and sufficient conditions for the jobs in $\mathcal{J}''$ to be scheduled together (with respect to their updated deadlines), as defined by Lemma 1. They state that for $1 \leq i \leq n$,

$$\sum_{I \in \{\mathcal{J}'' \cap J_1, \ldots, J_i\}} r(I) \leq d'(J_i)$$

We refer to the above inequality as *feasibility constraint* (i).

In fact, our problem can be thought of as a generalized knapsack problem. The items are $J_1, \ldots, J_n$; for each item J_i, its size is $r(J_i)$, its profit is $p(J_i)$, and $[0, d'(J_i)]$ is the part of the knapsack in which it can be packed. (Recall from the proof of Lemma 1 that if a set of jobs having the same release time can be scheduled together, then each job can be scheduled consecutively.) In the case where all deadlines are the same we get the standard knapsack problem. However, our objective function is the complement of the objective function of the knapsack problem, since our goal is to minimize the profit of the items that are not scheduled (or not packed in the knapsack). This is an NP-hard problem [13].

Since we are minimizing the profit of the jobs that are not scheduled, we refer in the sequel to the profit of a job as its *cost*. Given a cost function f on the jobs, the cost of job J with respect to f is called its f-cost. Let the *violation* of constraint (i) be defined as

$$C_i = \max\left\{ \sum_{j=1}^{i} r(J_j) \; - \; d'(J_i), 0 \right\}$$

We are now ready to define the cost decomposition needed for applying the local ratio technique to our problem. We denote by w the cost function that assigns cost $p(J_i)$ to each job J_i, $1 \leq i \leq n$. Let $\ell = \arg\max_{1 \leq i \leq k} C_i$. Define, for each job J_i, $1 \leq i \leq n$,

$$w_1(J_i) = \rho \cdot \begin{cases} \min\{C_\ell, r(J_i)\} & \text{if } 1 \leq i \leq \ell, \\ 0 & \text{otherwise.} \end{cases}$$

and let $w_2 = w - w_1$. We define ρ to be the unique scaler resulting in a non-negative w_2 weight function for all jobs, and a 0 w_2-cost for at least one of them. The function w_1 is "easy" in the sense that the size and cost of each job are equal with respect to it (unless $r(J_i)$ is too large).

Initially, $\mathcal{J}'$ is empty. At each decomposition step, we do the following:

1. For each job I, such that $w_2(I) = 0$, add I to $\mathcal{J}'$, and delete it from the list of jobs $\mathcal{J}$.
2. For all i, $1 \leq i \leq n$, update the value of C_i, the violation of constraint i. (This is needed since at least one job is added to $\mathcal{J}'$.)

We continue recursively with $(\mathcal{J}, w_2)$. At the end of the algorithm there is a clean-up step. The jobs in $\mathcal{J}'$ are considered in some arbitrary order and deleted if possible, yielding that $\mathcal{J}'$ is minimal with respect to containment. We now prove that $\mathcal{J}'$ is a 2-approximate solution. The proof is by induction on the number of recursive calls of the algorithm. At the basis of the recursion, the schedule returned is optimal (and hence a 2-approximation), since no jobs remain. For the inductive step, consider the first decomposition step. By the inductive assumption, $\mathcal{J}'$ is a 2-approximation with respect to w_2. We need to prove that $\mathcal{J}'$ is also a 2-approximation with respect to w_1. To this end we can assume without loss of generality that $\rho = 1$, since all jobs are scaled by it. A job is called *i-critical* if its deletion from $\mathcal{J}'$ will violate constraint (i).

Claim 6: The w_1-cost of the jobs in $\mathcal{J}' \cap J_1, \ldots, J_i$ is at most $2C_i$, if at least one of them is i-critical.

(The proof will appear in the full version of the paper.)

Let q be the highest index such that there is a job with positive w_1-cost which is q-critical. By the minimality of $\mathcal{J}'$, each job belonging to it is i-critical for at least one value of i. Hence, all jobs in $\mathcal{J}'$ with positive w_1-cost belong to $\mathcal{A}_q$, i.e., $\sum_{I \in \mathcal{J}'} w_1(I) = \sum_{I \in \mathcal{A}_q} w_1(I)$. By the above claim, $\sum_{I \in \mathcal{A}_q} w_1(I) \leq 2C_q$. Hence, by the maximality of C_ℓ, $\sum_{I \in \mathcal{J}'} w_1(I) \leq 2C_\ell$.

Consider an optimal solution. Since it satisfies constraint (ℓ), its w_1-cost has to be at least C_ℓ, since the costs and sizes of the jobs with respect to w_1 are equal. This finishes the proof that $\mathcal{J}'$ is a 2-approximate solution.

The algorithm can be implemented in $O(n^2)$ time, where n is the number of jobs under consideration.

B. Analysis of the algorithm

In this section we analyze the competitive factor of ALG. Consider the interval in time, $\mathcal{I}$, which is defined by the arrival of the first request and by the last deadline. Consider an optimal schedule OPT and delete from $\mathcal{I}$ all time slots in which both OPT and ALG were idle. We partition $\mathcal{I}$ into blocks $X_1, \ldots, X_\ell$ (X-blocks) and $Y_1, \ldots, Y_\ell$ (Y-blocks), and block Y_i follows block X_i for all i. The X-blocks contain all time slots in which ALG is active, either processing jobs that were completed, or jobs that were eventually deleted. The Y-blocks contain all time slots in which ALG is idle.

Claim 7: A Y-block cannot contain job arrivals.

Proof: The proof is immediate, as otherwise ALG would have accepted the request, since it is idle during the Y-blocks. Thus, yielding a contradiction. ■

In OPT, the Y-blocks can contain three types of jobs.

1. Jobs that ALG deleted, denoted by $\mathcal{J}_1$.
2. Jobs that ALG accepted, denoted by $\mathcal{J}_2$.
3. Jobs that ALG rejected, denoted by $\mathcal{J}_3$.

Let $\mathcal{F}$ denote the set of jobs that were completed by ALG. A job J belongs to $\mathcal{F}^*$, the *transitive closure* of $\mathcal{F}$, if there is a sequence of jobs $J = J_1, \ldots, J_k$, where $J_k \in \mathcal{F}$ and job J_i deleted job J_{i-1} for $1 < i \leq k$. (The jobs in $\mathcal{F}$ belong to $\mathcal{F}^*$ by definition.) Clearly, the jobs in $\mathcal{J}_1$ and $\mathcal{J}_2$ belong to the transitive closure of $\mathcal{F}$. The profit of the transitve closure of $\mathcal{F}$ is at most

$$p(\mathcal{F}^*) \leq p(\mathcal{F}) + \frac{p(\mathcal{F})}{\alpha - 1}$$

This follows since a set of jobs X is deleted (because of a job J) by ALG only if $p(J) \geq \alpha \cdot p(X)$. Hence, the "chain of deletions" is a geometric series.

We now upper bound the profit of the jobs belonging to $\mathcal{J}_3$. For every block Y_i in which there are jobs belonging to $\mathcal{J}_3$, denote by $h_i \in \mathcal{J}_3$ the job that ended the latest in OPT's schedule. Let H denote the collection of these jobs. Note that several consecutive Y-jobs may be associated with the same job from H. Let $R(h_i)$ denote the set of jobs which h_i was compared to in Step (2) of ALG. Since h_i was rejected, $p(h_i) \leq \alpha \cdot p(R(h_i))$. If H contains several jobs whose release time is in the same X-block, then we delete all of these jobs from H except for the job which has the latest finish time in OPT's schedule. This ensures that the sets $R(h_i)$ are disjoint. Let $t(h_i)$ denote the end of the X-block in which the request for job h_i appeared. Define the interval of time, $I(h_i)$, from $t(h_i)$ till h_i finishes its processing in OPT's schedule. Clearly, $p(h_i) > |I(h_i)|$, otherwise ALG would have accepted job h_i. Hence, $|I(h_i)| < \alpha \cdot p(R(h_i))$. Observe that the intervals $I(h_i)$ cover all the time slots in the Y-blocks in which a job from $\mathcal{J}_3$ is active. Hence,

$$p(\mathcal{J}_3) \leq \sum_i |I(h_i)| \leq \alpha \cdot \sum_{i=1}^{\ell} |X_i|$$

We are now ready to compute the competitive ratio. The profit of ALG is $p(\mathcal{F})$. The profit of the optimal solution is at most the profit of the transitive closure of $\mathcal{F}$ together with the jobs that were rejected by ALG and belong to $\mathcal{J}_3$. Note that $p(\mathcal{F}^*) \geq \sum_{i=1}^{\ell} |X_i|$ and therefore,

$$p(\mathcal{J}_3) \leq p(\mathcal{F}) \cdot \frac{\alpha^2}{\alpha - 1}$$

Hence, the competitive ratio is at most

$$\frac{p(\mathcal{F}) \cdot \frac{\alpha}{\alpha-1} + p(\mathcal{F}) \cdot \frac{\alpha^2}{\alpha-1}}{p(\mathcal{F})} = \frac{\alpha(\alpha+1)}{\alpha - 1}$$

which is minimized for $\alpha = 1 + \sqrt{2}$, yielding that the competitive factor is ≈ 5.82843.

Combining the above with the observation on the realitionship between the models at the beginning of the section, we get:

Theorem 8: The competitive ratio of ALG in the model without commitments is $\min\{5.82843, 2 + 1/\kappa\}$.

C. Lower bounds

A lower bound of 4 (for the case where $\kappa = 0$) on the competitive ratio of any scheduling algorithm in the model without commitments follows from the lower bound derived by Baruah *et al.* (cf. Lemma 1, [7]) for the related task scheduling problem in a uni-processor setting.

VI. SIMULATION STUDY

A. Models and setup

We implemented in Java a discrete event simulator for a single server. Each job is an instance of the class *Job* and is created using three parameters: (i) *arrivalTime* – time at which the job is presented to the on-line algorithm, (ii) *processingTime*, and (iii) *deadLine* of the job. These three parameters are set in the class *JobBox* to create a finite number of jobs (e.g., 3000) using one of the probabilistic models explained below on the job arrival. All the jobs in the JobBox are given to the off-line algorithm. In contrast the on-line algorithm has no prior information about the events in the next clock tick.

In the simulations, the *arrivalTime* and the *processingTime* parameters are used to control the traffic intensity and the job size, respectively. The deadline is computed as *deadLine = arrivalTime* $+(1 + \kappa)$ *processingTime*. The server is a class itself and at each clock tick, it processes the job with earliest deadline. Thus, if a new job with a earlier deadline than the current job's arrives, the current job is suspended and the new one is processed. If several jobs have the same deadline, then the one with the largest processing time is chosen. At each clock tick the *remainingTime* of the job currently being processed is decremented by one. A job is completed if its *remainingTime* becomes zero before its deadline.

Stochastic modeling of jobs. We consider three probability distributions for job arrivals: (i) Poisson(λ, μ), (ii) Pareto(k_1, k_2), (iii) Bursty on-off process (p, s).

Poisson model: The parameters are exponentially distributed random variables λ and μ, the mean arrival rate and the inverse of the mean job size, respectively. Given a value for λ, we choose a value uniformly at random between 0 and 1 and feed it to the Poisson distribution to compute the number of jobs (k) that arrive in a time tick i (i.e., for each of these jobs, we set the arrival time to i). Given a value for μ, the processing time of the jobs is computed similarly.

Pareto model: Although the Poisson model is simple and widely used, it does not capture the behavior of many applications in particular the ones with a self-similar, or fractal character. It is possible to define self-similar stochastic processes using a heavy-tailed distribution such as the Pareto distribution with parameters k and α for $k, \alpha > 0$, where the Hurst parameter is $H = (3 - \alpha)/2$:

$$F(x) = 0 \text{ if } x < k; \;\; F(x) = 1 - \left(\frac{k}{x}\right) \text{ o.w.}; \qquad (1)$$

$$f(x) = 0 \text{ if } x < k; \;\; f(x) = \frac{\alpha}{k}\left(\frac{k}{x}\right) \text{ o.w.}; \qquad (2)$$

$$E[X] = \frac{\alpha}{\alpha - 1} k \quad (\alpha > 1).$$

The parameter k specifies the minimum value the random variable can take. The parameter α determines the mean and variance of the random variable. In the simulations, for each *arrivalTime*, we generate a random number k_1 and set parameter $k = k_1$ in Equations (1) and (2) to compute the number of jobs x arriving in that time tick. Then for each of these jobs, we use a second random number k_2 to compute the size of that job. The parameters used in the simulation are k_1 - the minimum job arrival, and k_2 - the minimum job size.

On-Off model: The last model for traffic generation is a simplified on-off stochastic process. At each time tick, with a probability of p the process is on (i.e., there is job arrival in this tick) and with $1 - p$ it is off. If the process is on then a random number is chosen uniformly at random between 1 and an upper bound s, to determine the burst size (the number of jobs that arrive in this time tick). Then, for each job, a size is chosen uniformly at random between 1 and another upper bound.

B. Results and interpretations

The plottings in this section depict the average values obtained from 10000 simulations runs.

Performance impact of load. Our first objective is to determine how the performance ratio (PR) will change as a

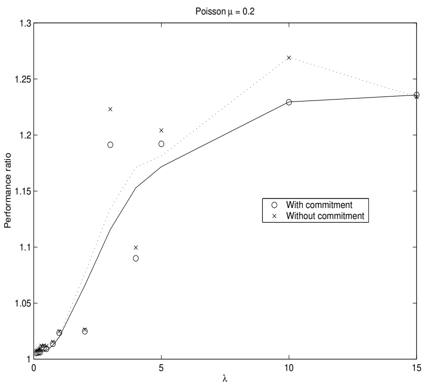

Fig. 1. PR as a function of mean arrivals

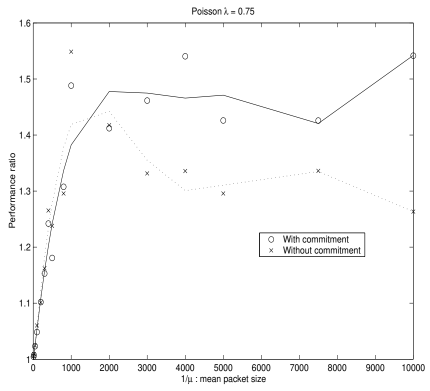

Fig. 2. PR as a function of mean job size

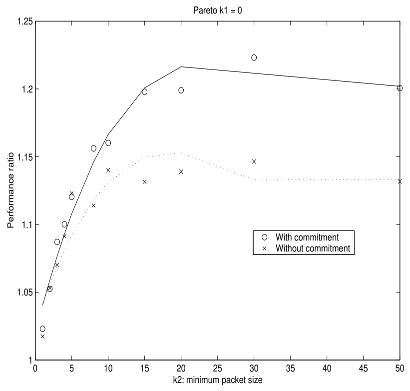

Fig. 3. PR as a function of min.job size

function of increasing server load. The server load can be controlled by tuning two parameters: the mean arrival rate and the mean job size. For this set of experiments we fixed the parameter $\kappa = 1$ and examined the impact of varying κ and the jobs' slack in the following section.

Figure 1 shows the average performance ratio of the on-line algorithms, with and without commitment, as a function of λ, with a fixed $\mu = 0.2$ in the Poisson model. Notice that when the system is underloaded (i.e., $\lambda/\mu < 1$) the ratio is close to 1 indicating that GREEDY is close to OPT. Intuitively, this is expected since it is easier for EDF to find a feasible schedule S for the jobs $J_i, J_2, \cdots, J_n$ as n gets smaller. If there always is a feasible schedule for the new arrivals, there is no need for admission control and performance of the off-line and on-line algorithms are similar. The ratio changes when admission control must be enforced. This is the case when the system is overloaded (i.e., $\lambda/\mu > 1$) as depicted in Figure 1 by increasing λ while keeping μ constant.

Figure 2 shows the average ratio of the on-line algorithms as a function of the job size (i.e., $1/\mu$), for $\lambda = 0.75$. We can clearly see that the ratio is close to 1 when the job size is small, and it increases as the job size increases. It seems to stabilize around 1.5 for the commitment case, and about 1.3 for the no-commitment case.

Similar experiments were conducted for the Pareto distribution by changing the parameters: the minimum job arrival per time unit k_1, and the minimum job size k_2. Figure 3 shows that the performance ratio is very close to 1 for small job sizes and it is linearly proportional to the job size for the minimum job arrival per time tick (i.e., for $k_1 = 0$). It stabilizes at 1.2 for the commitment case, and at 1.15 for the no-commitment case.

Figure 4 shows how the performance ratio changes as the system gets overloaded in the on-off model for $p = 0.5$. Under light load with small job size the ratio is close to 1 for both commitment and without commitment case. As the system is loaded by increasing the job size the ratio also increases (Figure 5).

The joint impact of parameters k_1 and k_2 is also examined for the Pareto model in Figures 6 and 7. They show various packet arrival rates as a function of the minimum job size for the commitment and no-commitment cases, respectively. Note that the ratios for different values of k_1 converge to the same range as k_2 changes.

C. *Impact of κ and slack on the performance*

This section examines the impact of the parameter κ and varying the jobs' slack on the performance ratio. First, we considered the case where all jobs have the same ratio between their slack and processing time ($= \kappa$), for different values of κ. Next we allowed each job to choose their ratios uniformly at random from a $[\kappa_{\min}, \kappa_{\max}]$ interval.

Jobs with fixed ratio. We considered three load models with the following parameters: (i) Poisson process with $\lambda = 0.75, \mu = 500$, (ii) Pareto distribution with $k_1 = 0, k_2 = 10$, and (iii) Poisson process with $\lambda = 0.75, \mu = 2000$. We collected the data from 10000 different simulation runs to determine experimentally the maximum performance ratio. The first plotting in Fig. 8 depicts the results for the commitment case. Note that the performance ratio depends on the value of κ: for small values of κ, the performance of the on-line algorithm is much worse. With Poisson(0.75,500), the worst ratio does not show a big variation with a change in κ, while with Pareto(0.75,2000) the server is saturated.

We performed the same experiments for the case without commitment. In this case, recall that the worst upper bound is defined by the minimum of 5.82843 and 2+1/k, thus there is a break at $\kappa = 0.2612$. The second plotting in Fig. 8 shows that there is an experimental break at about $\kappa = 0.4$. Note that for $\kappa \leq 0.3$, the performance in the case without commitment is much better than in the case with commitment.

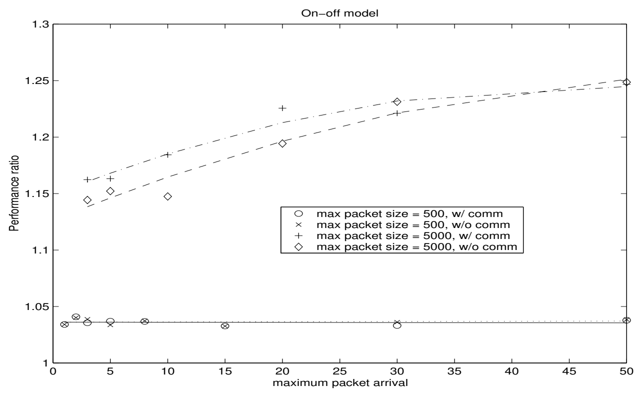

Fig. 4. PR as a function of job arrival rate

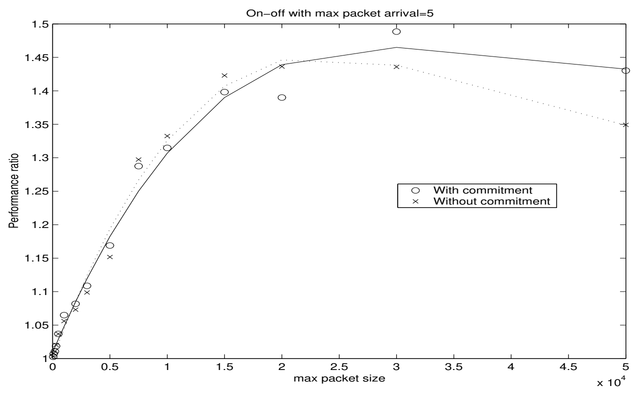

Fig. 5. PR as a function of maximum packet size

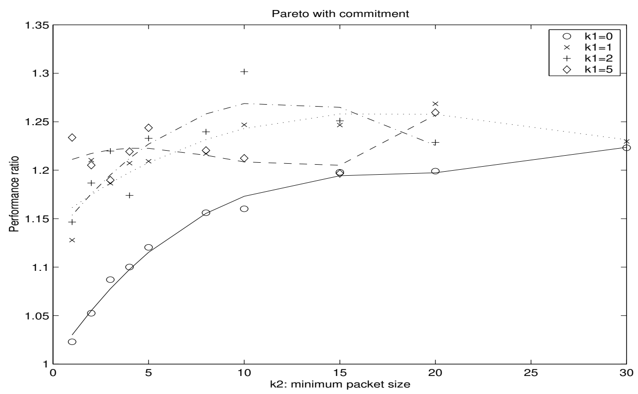

Fig. 6. Joint impact of arrival rate and minimum job size with commitment

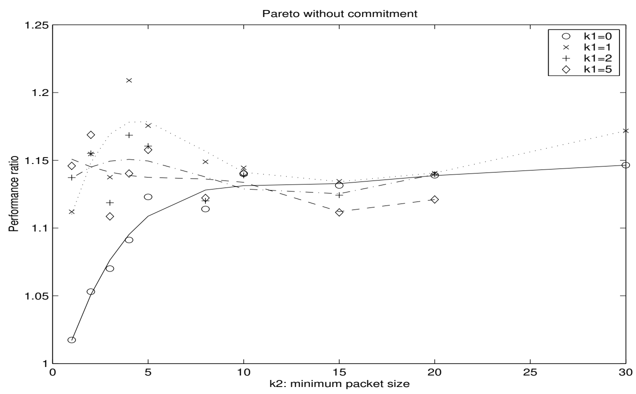

Fig. 7. Joint impact of arrival rate and minimum job size w/o commitment

The first plotting in Fig. 9 shows that the average performance ratio decreases when κ increases. The magnitude varies depending on the distribution (Pareto, Poisson) and the case (with or w/o commitment). Overall, the PR improves when more slack is given to the jobs.

Jobs with variable ratio. Next, we considered choosing a job's slack/processing time ratio uniformly at random from a $[\kappa_{\min}, \kappa_{\max}]$ interval. The second plotting in Fig. 9 shows the impact on the performance ratio. The traffic follows the Pareto distribution with parameters $k_1 = 0, k_2 = 10$, and $\kappa_{\min}$ is set to 0.3 while $\kappa_{\max}$ is varied up to 4. Fig. 9 shows that when $\kappa_{\max}$ increases, the ratio decreases. This is because, effectively, as jobs have more slack, the on-line algorithms have more flexibility.

VII. Conclusions

This paper presents a comprehensive study of the effect of job interleaving by preemption on the utilization of a single server. The problem is investigated at an abstract level to ensure its generality and applicability to many scenarios. The work considers two scheduling modes, one with commitment to complete the jobs accepted and one without such a commitment.

We treat the subject at both the analytical and experimental levels. In the former, we show for the commitment case that a simple greedy algorithm achieves a competitive ratio of $2 + \frac{1}{\kappa}$, $\kappa > 0$, and prove a lower bound on the competitiveness of any deterministic on-line algorithm of $1 + \frac{1}{\kappa}$. In the case of scheduling without commitment, we distinguish between two cases, depending on the value of κ: for large values of κ the competitive factor of $2 + \frac{1}{\kappa}$ proved for the commitment case carries over, while for small values of κ we present an algorithm with a competitive ratio of 5.82843. Thus, the competitive ratio for this case is the minimum between the two ratios. A lower bound of 4 follows from the related task scheduling problem in a uni-processor setting.

The experimental performance of the algorithms is determined by simulations. The study examines the impact of the intensity of jobs arrivals and their sizes under several probability distributions. The impact of the parameter κ is also investigated. The results show that under light load the algorithms perform as well as the off-line algorithm, while under heavy load their performance converges to the

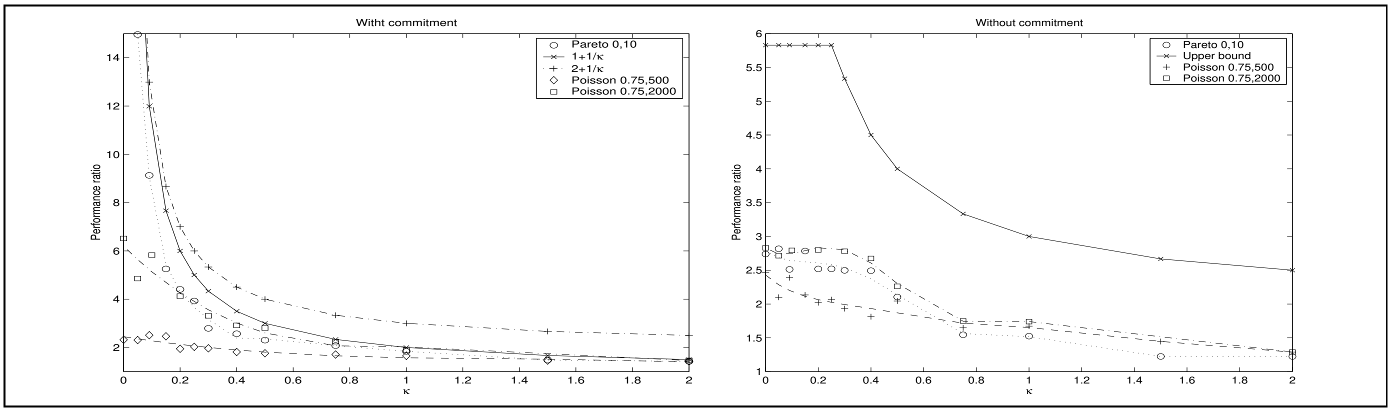

Fig. 8. Experimental maximum for PR as a function of κ

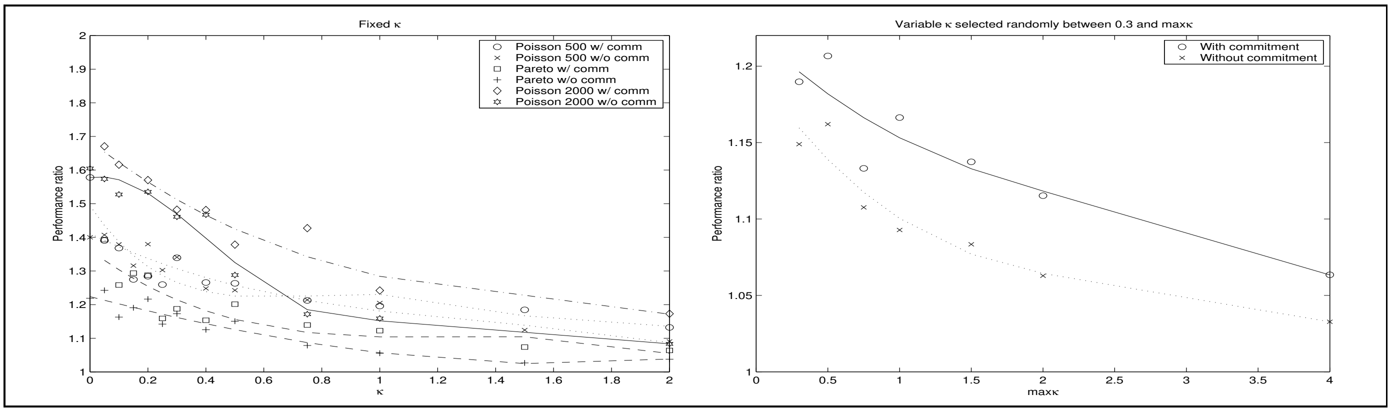

Fig. 9. Average performance ratio (a) as a function of κ; (b) as a function of the jobs' slack

worst-case bounds. Changes in the jobs' slack have an impact on the performance ratio, and our results suggest that as the slack increases the performance of the on-line algorithms improves.

REFERENCES

[1] *Atlanta* ATM ICs. `http://Atlanta.web.lucent.com`.

[2] G. ARMITAGE, Quality of service in IP networks, Pearson Higher Education, 2000.

[3] V. BAFNA, P. BERMAN, AND T. FUJITO, A 2-approximation algorithm for the undirected feedback vertex set problem, *SIAM J. on Disc. Mathematics*, Vol. 12, pp. 289–297, 1999.

[4] A. BORODIN AND R. EL-YANIV, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[5] R. BAR-YEHUDA AND S. EVEN, A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, Vol. 25, pp. 27–46, 1985.

[6] S. BARUAH AND J. HARITSA. Scheduling for overload in real-time systems. *IEEE Transactions on Computers*, 46(9):1034-1039, 1997.

[7] S. BARUAH, G. KOREN, D. MAO, B. MISHRA, A. RAGHUNATHAN, L. ROSIER, D. SHASHA AND F. WANG, On the competitiveness of on-line real-time task scheduling, *Real-Time Systems*, Vol. 4, pp. 125–144, 1992.

[8] CS CHANG, *Performance Guarantees in Communication Networks*, Springer-Verlag, 2000.

[9] B. DASGUPTA AND M. A. PALIS. Online real-time preemptive scheduling of jobs with deadlines, *Proc. APPROX 2000*, pp. 96-107.

[10] A. FELDMAN, B. MAGGS, J. SGALL, D. SLEATOR, AND A. TOMKINS. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, 1995.

[11] M. GOLDWASSER, Patience is a virtue: the effect of slack on competitiveness for admission control. *Proceedings of 10th Annual ACM-SIAM on Discrete Algorithms*, pp. 396-405, 1999.

[12] L. GEORGIADIS, R. GUÉRIN AND A. PAREKH. Optimal multiplexing on a single link: Delay and buffer requirements. *IEEE Transactions on Information Theory*, 43(5):1518-1535, 1997.

[13] M. GAREY AND D. JOHNSON, Computers and intractability: a guide to the theory of NP-completeness, *W. H. Freeman*, 1979.

[14] CABLELABS., Data-over-Cable Service Interface Specification Radio Frequency Interface Specification SP-RFIv1.1-I01-990311, 1999.

[15] G. V. GENS AND E. V. LEVNER, Computational complexity of approximation algorithms for combinatorial problems, *Mathematical Foundations of Computer Science*, pp. 292-300, 1979 (Lecture Notes in Computer Science, Vol. 74).

[16] S. A. GOLDMAN, J. PARWATIKAR, AND S. SURI, Online scheduling with hard deadlines. *Journal of Algorithms*, Vol. 34, pp. 370-389, 2000.

[17] IXP 1200 Network Processor. Available at `http://www.intel.com/design/network/products/npfamily/ixp1200.htm`.

[18] J. LIEBEHERR, D. WREGE, AND D. FERRARI. Exact admission control for networks with a bounded delay service. *IEEE/ACM Transactions on Networking*, 4(6):885-901, December 1996.