Partial Servicing of On-line Jobs

R. van Stee, H. La Poutré

Software Engineering (SEN)

SEN-R0016 June 23, 2000

Report SEN-R0016
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Partial Servicing of On-line Jobs

Rob van Stee*

Han La Poutré

Centre for Mathematics and Computer Science (CWI)

Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands

ABSTRACT

We consider the problem of scheduling jobs online, where jobs may be served partially in order to optimize the overall use of the machines. Service requests arrive online to be executed immediately; the scheduler must decide how long and if it will run a job (that is, it must fix the Quality of Service level of the job) at the time of arrival of the job: preemption is not allowed. We give lower bounds on the competitive ratio and present algorithms for jobs with varying sizes and for jobs with uniform size, and for jobs that can be run for an arbitrary time or only for some fixed fraction of their full execution time.

2000 Mathematics Subject Classification: 68M20,90B35

1998 ACM Computing Classification System: F.2.2.

Keywords and Phrases: on-line scheduling, partial scheduling, quality of service

Note: Work carried out under project SEN4 "Evolutionary Computation and Applied Algorithmics".

1. Introduction

Partial execution or computation of jobs has been an important topic of research in several papers [2, 4, 5, 6, 7, 8, 9, 11, 12]. Problems that are considered are e. g. imprecise computation, anytime algorithms and two-level jobs (see below).

In this paper, we study the problem of scheduling jobs online, where jobs may be served only partially in order to increase the overall use of the machines. This e. g. also allows downsizing of systems. The decision as to how much of a job to schedule has to be made at the start of the job.

This corresponds to choosing the Quality of Service (QoS) in multimedia systems. One could e. g. consider the transmission of pictures or other multimedia data, where the quality of the transmission has to be set in advance (like quality parameters in JPEG), cannot be changed halfway and transmissions should not be interrupted.

Another example considers the scheduling of excess services. For instance, a (mobile) network guarantees a basic service per request. Excess quality in continuous data streams can be scheduled instantaneously if and when relevant, and if sufficient resources are available (e. g. available buffer storage at a network node).

Finally, when searching in multimedia databases, the quality of the search is adjustable. The decision to possibly use a better resolution quality on parts of the search instances can only be made on-line and should be serviced instantly if excess capacity is available [3].

*Supported by SION/NWO, project number 612-30-002

In the paper, we consider the following setting. Service requests have to be accepted or rejected at the time of arrival; when (and if) they are accepted, they must be executed right away. We use competitive analysis to measure the quality of the scheduling algorithms, comparing the online performance to that of an offline algorithm that knows the future arrivals of jobs.

We first consider jobs with different job sizes. In that case, the amount by which the sizes can differ is shown to determine how well an algorithm can do: if all job sizes are between 1 and M, the competitive ratio is $\Omega(\ln M)$. We adapt the algorithm `Harmonic` from [1] and show a competitive ratio of $O(\ln M)$.

Subsequently, and most important, we focus on scheduling uniform sized jobs. We prove a randomized lower bound of 1.5, and we present a deterministic scheduling algorithm with a competitive ratio slightly above $2\sqrt{2} - 1 \approx 1.828$. Finally, we consider the case where jobs can only be run at two levels: $\alpha < 1$ and 1. We derive a lower bound of $1 + \alpha - \alpha^2$.

1.1 Related Work

Imprecise computation In overloaded real-time systems, *imprecise computation*[7] is a well-known method to ensure graceful degradation. In this model, each task is assumed to consist of a mandatory subtask and an optional subtask. The mandatory subtask must be scheduled before the deadline of the task, but the optional subtask does not need to be completed and can be stopped at any time, producing an imprecise result.

The main difference compared to the model in this paper is that we require the running time of all jobs to be determined in advance; an algorithm must decide how long a job will run before that job starts. So it cannot just stop a job when that is convenient. Also, most results are about offline versions of this problem.

In the imprecise computation model, a scheduler must schedule all mandatory subtasks. It should try to minimize the errors created by stopping optional subtasks prematurely. Many scheduling algorithms achieve an optimal tradeoff between result quality and execution time, basing this tradeoff on minimizing average error, total error, maximum error, number of discarded optional tasks, number of tasks completed late, or average response time[6].

On-line scheduling is studied in [9], but mostly on task sets that satisfy the *feasible mandatory contraint*: at the time of arrival of every new on-line task, its mandatory portion, together with the yet-to-be-completed mandatory portions of already arrived tasks, can always be precisely scheduled to complete by their deadlines. Algorithms are shown that are optimal on such task sets, and a lower bound is given for the problem of scheduling task sets that do not satisfy this constraint.

In [2], the *weak feasible mandatory constraint* is introduced: the mandatory portions of all tasks in the task set can be scheduled to complete by their deadlines by an optimal off-line algorithm. This constraint does not depend on the decisions made by a particular on-line scheduling algorithm, and is more realistic but still quite strong. Not completing some mandatory part now only means that no value is obtained for that job.

Anytime algorithms are introduced in [5]. This is a type of algorithm that may be interrupted at any point to return a result whose quality is a function of the execution time. This was later extended to algorithms whose quality depends on execution time and input quality[12].

Scheduling jobs that can run at two levels In [4], a model similar to the one in this paper is studied, but on a single machine and using stochastic processes and analysis. Jobs arrive in a Poisson process and can be executed in two ways, full level or reduced level. If they cannot start immediately, they are put in a queue. This problem is studied both for the case where the execution of a job can be switched from one level to the other, and where it cannot (as is the case in our model).

For both cases, a threshold method is proposed: the approach consists of executing jobs on a particular level depending on whether the length of the queue is more or less than a parameter M. If the algorithm is allowed to lower the level of a running job, it does this as soon as the queue length exceeds some threshold M; otherwise it waits for the current job to complete and serves the next jobs at the reduced level, until the queue length drops below M again.

The performance of this algorithm, which depends on the choice of M, is studied in terms of mean task waiting time, the mean task served computation time, and the fraction of tasks that reveice full level computation. The user can adapt M to optimize his desired objective function. There are thus no time constraints (or deadlines) in this model, and the analysis is stochastic. In [11], this model is studied on more machines, again using probabilistic analysis.

Periodic tasks In [8], besides imprecise computation, also deferred deadlines for periodic tasks are studied: the deadline of some tasks is then deferred by e. g. one period, so that the lateness of all tasks is acceptably small during an overload.

Several error measures have been studied for periodic jobs:

- error-non-cumulative: only the average error of the jobs is important
- error-cumulative: the effects of errors in the result produced in different periods are cumulative, so that some optional parts must be completed in order to get a valid result at the end
- limitedly-cumulative error: the maximal error is limited

A further extension to this model is studied in [6], namely error propagation. Here it is assumed that the output of one task is the input to a following task. Also the case where groups of tasks have deadlines in stead of individual tasks (so called end-to-end deadlines) are studied there.

2. Definitions and notations

The number of machines is n. The performance measure used is the total usage of all the machines (the total amount of time that machines are busy). For each job, a scheduling algorithm earns the time that it serves that job. The goal is to use the machines most efficiently, in other words, to serve as many requests as possible for as long as possible. The earnings of an algorithm A on a job sequence σ are denoted by $A(\sigma)$. The adversary is denoted by ADV. The competitive ratio of an algorithm A, denoted by $r(A)$, is defined as

$$r(A) = \sup_{\sigma} \frac{ADV(\sigma)}{A(\sigma)}.$$

If jobs can only be run for two specific lengths of time, we take them to be 1 and $0 < \alpha < 1$. Throughout this paper, $\varepsilon > 0$ is an arbitrarily small constant.

3. Different job sizes

We will first show that if the jobs can have different sizes, the competitive ratio of an online algorithm is not helped much by having the option of scheduling jobs partially. The most important factor is the size of the accepted and rejected jobs, and not how long they run. This even holds when the job sizes are bounded.

The proofs in this section hold both when α is fixed, and when jobs may be run for an arbitrary length of time between 0 and 1.

Lemma 1 *If job sizes can vary without bound, no algorithm to schedule jobs on n machines can attain a finite competitive ratio.*

Proof. Suppose there is a r-competitive online algorithm A, and the smallest occurring job size is 1. The following job sequence is given to the algorithm:

$$\begin{aligned} x_1 &= 1, \\ x_2 &= r, \\ x_i &= r^{i-1} \quad (i = 3, \ldots, n), \\ x_{n+1} &= 2r(1 + \cdots + r^{n+1}). \end{aligned}$$

All jobs arrive at time $t = 0$. As soon as A refuses a job, the sequence stops and no more jobs arrive.

Suppose A refuses job x_i, where $i \leq n$. Then A earns at most $1 + r + \cdots + r^{i-2}$, while the adversary earns $1 + r + \cdots + r^{i-1}$. We have

$$\frac{1 + r + \cdots + r^{i-1}}{1 + r + \cdots + r^{i-2}} > 1 + \frac{r^{i-1} - 1}{1 + r + \cdots + r^{i-2}} = 1 + r - 1 = r.$$

This implies A must accept the first n jobs. However, it then earns at most $1 + \cdots + r^{n-1}$. The adversary serves only the last job and earns $2r$ times as much. □

Note that this lemma holds even when all jobs can only run completely.

3.1 Two machines

Lemma 2 *If for all job sizes x we have $1 \leq x \leq M$, then for all algorithms A on two machines we have*

$$r(A) \geq r_{2,M} = \sqrt{4M + 1} - 1.$$

Proof. Consider any online algorithm A. The job sequence used is 1, $\frac{\sqrt{4M+1}-1}{2}$, $\frac{\sqrt{4M+1}-1}{2}$, M, M, where all jobs arrive at $t = 0$. The first job must be accepted by A to attain a finite competitive ratio, the second or the third to attain $\sqrt{4M + 1} - 1$. (If only the first job is accepted, the adversary can earn $\sqrt{4M + 1} - 1$.) The adversary then serves the last two jobs and earns $2M$, while A earns at most $\frac{\sqrt{4M+1}+1}{2}$ (less if some job is not run completely). □

Algorithm A_M for two machines is defined as follows: if both machines are available, it accepts any job. Otherwise, it accepts only jobs with size at least $\sqrt{M + 1} - 1$. All accepted jobs are run completely.

Since A_M runs all jobs completely, the strongest adversary is one that may serve jobs for any length of time. This adversary is used in the following lemma.

Lemma 3 $r(A_M) = 2\sqrt{M+1}$.

Proof. If A_M can accept every job, there is nothing to prove, because it runs all jobs completely. Suppose it refuses some jobs, then we need to show that A_M earns enough from the jobs it accepted. We can do this by allocating the missed earnings to the accepted jobs, and never allocating more than $(2\sqrt{M+1}-1)x$ of missed earnings to a job of size x. We distinguish two cases.

Case 1. For intervals in which A_M serves only one job at a time (it keeps one machine available during the complete execution of the jobs), we allocate all the jobs that arrive during such a job to that job. Say A_M runs job x, then all of the jobs arriving while x runs have sizes less than $\sqrt{M+1}-1$.

The worst case is when the adversary runs two jobs of size x for $x-\varepsilon$ time, and then two jobs of size $\sqrt{M+1}-\varepsilon$, which arrive at time $t = 1-\varepsilon$, completely. We have $\frac{ADV(\sigma)}{A_M(\sigma)} = \frac{2(x+\sqrt{M+1}-1-2\varepsilon)}{x} \to 2\sqrt{M+1}$ for $x = 1$ and $\varepsilon \to 0$.

Case 2. If A_M is running a job x, and another job arrives that A_M serves as well, we can allocate the missed earnings to this pair of jobs. Take $x = 1, y = \sqrt{M+1}-1$, where the adversary can serve two jobs of size 1 for $1-\varepsilon$ time, and then two jobs of size M. If $x = 1+x_1$, the adversary can earn $2x_1$ more while A_M earns x_1 more than when $x = 1$, so that its relative performance improves. If $y > \sqrt{M+1}-1$, the adversary earns nothing more. This shows that our choices of x and y represent the worst case.

We have $\frac{ADV(\sigma)}{A_M(\sigma)} = \frac{2M+2}{\sqrt{M+1}} = 2\sqrt{M+1}$.

Suppose a job arrives after x or y is finished, but before both are finished. Only if A_M serves it do the possible missed earnings of A_M increase, because this creates a new interval in which A_M cannot serve any jobs. However, in this case the new job has size at least $\sqrt{M+1}$ and the missed earnings increase by at most $\sqrt{M+1}$, because the time at which the adversary starts the jobs of size M can move by at most that much. □

Note that an online algorithm can do at best only slightly better than A_M if it runs some jobs partially, because Lemma 2 implies A_M is already almost optimal.

3.2 $n > 2$ machines

Lemma 4 *For $M > 2^n$, we have $r(A) > n$ for all algorithms A on n machines.*

Proof. Let $x = \sqrt[n]{M}-1$ and suppose A maintains a competitive ratio of n. If $M > 2^n$, then $x > 1$. Consider the following sequence of jobs: 1, $x_i = x(1+x)^{i-1} (i = 1, \ldots, n-1)$, where instances of every job x_i arrive repeatedly until A accepts one. A has to accept the first job and one of every x_i. For instance, if it refuses n jobs of size x_2, it earns at most $1+x_1 = 1+x$ while the adversary can earn nx_2, and then $r(A) \geq \frac{nx_2}{1+x} = nx > n$, a contradiction.

Finally, n jobs of size M arrive. A earns at most $1+x_1+\cdots+x_{n-1} = (1+x)^{n-1} = (\sqrt[n]{M})^{n-1}$, and we have $r(A) \geq \frac{nM}{(\sqrt[n]{M})^{n-1}} = n\sqrt[n]{M} > nx > n$. □

Corollary For $M > 2^n$, we have $r(A) > n(\sqrt[n]{M}-1)$ for all algorithms A.

We can also show a bound of $\ln M$ by using a method similar to [1]. This bound holds for all M and n (also if n is large). However, if $M > 2^n$, the bound above is still the strongest.

Theorem 1 *For the competitive ratio r of this scheduling problem with different job sizes we have $r > \ln M$.*

We will first introduce a job sequence, and then show that it implies the theorem. The adversary generates the job sequence in steps; in each step i, n jobs arrive of size i. This process ends when (and if) the online algorithm has assigned each machine to a job.

Suppose there is a r-competitive online algorithm A. To maintain a competitive ratio of r, A has to serve at least n/r jobs of the first n jobs. In the second step, it must serve x jobs so that $2n/(n/r + 2x) \leq r$. It follows that $x \geq n/2r$. In step i, A has to serve n/ir jobs, earning n/r in each step. This means that A will exhaust its supply of machines in step k, where k is the smallest solution of $\frac{n}{r}(1 + \frac{1}{2} + \cdots + \frac{1}{k}) \geq n$. It follows that

$$r \leq H_k = \sum_{i=1}^{k} \frac{1}{i}, \tag{3.1}$$

and that A has then earned kn/r. Note that serving some jobs partially only lowers A's profit! We must have $k \leq M$ for this method to work, otherwise not all machines are busy after the jobs of size M.

On the other hand, if this method works the adversary can serve a job of size M on every machine after A has used all its machines. Then we have $r \geq \frac{nM}{kn/r}$, implying that $k \geq M$. We conclude that $k = M$ in this case. We are now ready to prove the theorem.

Proof. Suppose there exists an online algorithm A with competitive ratio $r \leq H_{M-1}$. Then the above job sequence will cause A to use all its machines in the final step, since (3.1) is satisfied with $k = M$. Consider step $k-1$. After this step, n jobs of size M arrive, which are all served by the adversary. We find that

$$r \geq \frac{Mn}{\frac{n(k-1)}{r} + M(n - \frac{nH_{k-1}}{r})} \Rightarrow r \geq 1 + H_{k-1} + \frac{1}{M} > H_{M-1}.$$

This is a contradiction. We conclude $r > H_{M-1} \geq \ln M$. □

Compare this result to [1], where a central server had to decide which movies to show on a limited number of channels. Each movie has a certain value determined by the amount of people that have requested that movie, and the goal is to use the channels most profitably. The result there is $r \geq \ln \mu$, where μ is the total number of customers divided by the number of channels.

Algorithm `Harmonic`[1] divides the machines into $M - 1$ sets $S_1, \ldots, S_{M-1}$. Each set S_i contains $\lfloor \frac{n}{iH_{M-1}} \rfloor$ machines. Machines in set S_i serve only jobs of size at least i. They are served completely.

Again we use the strongest possible adversary, that can run jobs for any length of time, in the following proof. We will show that `Harmonic` is only a factor of 2 away from the optimal competitive ratio. This implies that an algorithm that serves some jobs partially could do at most a factor of 2 better, and probably less. This is independent of which running times are allowed for the jobs.

Theorem 2 $r(\texttt{Harmonic}) = 2H_{M-1} + 1$.

Proof. For every amount of sets that are busy in `Harmonic`'s schedule, we will show that the adversary cannot earn more than $2H_{M-1}+1$ times what `Harmonic` earns on the busy sets. We distinguish two cases.

Case 1 Suppose all machines are in use by `Harmonic`. Then it earns at least $n(M-1)/H_{M-1}$, while an adversary can earn at most $nM + \frac{n(M-1)}{H_{M-1}}$. The adversary can earn this amount from the following job sequence: at times $t = i$ $(i = 0, \ldots, M-2)$, $\frac{n}{(M-1-i)H_{M-1}}$ jobs of size $M-1-i$ arrive. The adversary serves all of these jobs until time $t = M-1-\varepsilon$.

Finally, n jobs of size M arrive at time $t = M-1-\varepsilon$, when all of `Harmonic`'s machines are still busy. See figure 1.

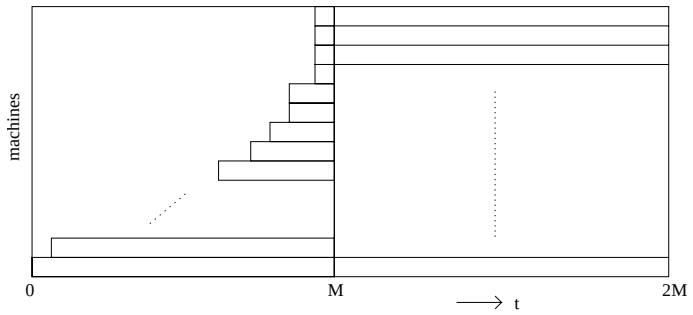

Figure 1: Worst case for `Harmonic`

In this case, we have $\frac{ADV(\sigma)}{\texttt{Harmonic}(\sigma)} \leq \frac{nM+n(M-1)/H_{M-1}}{n(M-1)/H_{M-1}} = \frac{M}{M-1}H_{M-1}+1 < 2H_{M-1}+1$.

Case 2 If not all machines are in use, `Harmonic` will only refuse jobs that are currently too small for any non-filled set. Suppose all sets $S_1, \ldots, S_k$ are in use. Then `Harmonic` earns at least nk/H_{M-1}. The adversary can earn at most $n(k+1-\varepsilon)$ additionally, by serving n jobs of size $k+1-\varepsilon$, which arrive just before all of `Harmonic`'s machines become available again. We have $\frac{ADV(\sigma)}{\texttt{Harmonic}(\sigma)} = \frac{n(k+1+k/H_{M-1})}{nk/H_{M-1}}$. This is $2H_M+1$ for $k=1$. □

Note that again, an algorithm could do at best only slightly better than `Harmonic` by serving jobs partially, because `Harmonic` is almost optimal.

4. Uniform job sizes

We will now study the case of identical job sizes, which we take to be 1. From this point onwards, we assume that the scheduling algorithm is completely free in choosing how long it serves any job.

4.1 Lower bounds

The simplest algorithm is *Greedy*, which serves all jobs completely if possible. Clearly, Greedy maintains a competitive ratio of 2, because it can miss at most 1 in earnings for every job that it serves. The following lemma shows that, like in section 3, the case of two machines forms an exception where partial scheduling is not advantageous.

Lemma 5 *For two machines and jobs of size 1, Greedy is optimal among algorithms that are free to choose the execution times of jobs between 0 and 1, and it has a competitive ratio of 2.*

Proof. Assume some algorithm A has a competitive ratio less than 2, say $2-\delta$ where $\delta > 0$. The adversary lets two jobs arrive at time $t = 0$. Say A serves them for $0 < \alpha_1 \leq 1$ and $\alpha_1 \leq \alpha_2 \leq 1$ time respectively. (If $\alpha_1 = 0$, A earns at most 1 and has a competitive ratio of at least 2.) At time $t = \alpha_1 - \varepsilon$, two jobs arrive. The adversary can now earn $2(1 + \alpha_1 - \varepsilon)$, while A earns $\alpha_1 + \alpha_2$. We have $\frac{2(1+\alpha_1-\varepsilon)}{\alpha_1+\alpha_2} \geq 2(1 - \frac{\varepsilon}{1+\alpha_1}) > 2 - \delta$ for ε small enough: a contradiction. □

We give a lower bound for the general case, which even holds for randomized algorithms.

Lemma 6 *For jobs of size 1 on $n > 2$ machines, no (randomized) algorithm that is free to choose the execution times of jobs between 0 and 1 can have a lower competitive ratio than 3/2.*

Proof. We use Yao's Minimax Principle [10].

We examine the following class of random instances. At time 0, n jobs arrive. At time $0 < t \leq 1$, n more jobs arrive, where t is uniformly distributed over the interval $(0, 1]$. The expected optimal earnings are $3n/2$: the first n jobs are served for such a time that they finish as the next n jobs arrive, which is expected to happen at time $1/2$; those n jobs are served completely.

Consider a deterministic algorithm A and say A earns x on running the first n jobs (partially). If A has $v(t)$ machines available at time t, when the next n jobs arrive, it earns at most an additional $v(t)$. Its expected earnings are at most $x + \int_{t=0}^{1} v(t)dt = n$, since $\int_{t=0}^{1} v(t)dt$ is exactly the earnings that A missed by not serving the first n jobs completely: $x = n - \int_{t=0}^{1} v(t)dt$. Therefore $r(A) \geq 3/2$. □

4.2 Algorithm SL

We now present an algorithm SL which makes use of the possibility of choosing the execution time. Although SL could run jobs for any time between 0 and 1, it runs all jobs either completely (*long* jobs) or for $\frac{1}{2}\sqrt{2}$ of the time (*short* jobs). We denote the number of running jobs of these types at time t by $l(t)$ and $s(t)$. The arrival time of job j is denoted by t_j.

The idea is to make sure that each short job is related to a unique long job which starts earlier and finishes later. To determine which long jobs to use, marks are used. Short jobs are never marked. Long jobs get marked to enable the start of a short job, or when they have run for at least $1 - \frac{1}{2}\sqrt{2}$ time. The latter is because a new short job would always run until past the end of this long job. In the algorithm, at most $s_0 = \lceil (3 - \sqrt{2})n/7 \rceil \approx 0.22654 \cdot n$ jobs are run short simultaneously at any time. We will ignore the rounding and take $s_0 = (3-\sqrt{2})n/7$ in the calculations. The algorithm is as follows.

Algorithm SL If a job arrives at time t, refuse it if all machines are busy.

If a machine is available, first mark all long jobs j for which $t - t_j \geq 1 - \frac{1}{2}\sqrt{2}$. Then if $s(t) < s_0$ and there exists an unmarked long job x, run the new job for $\frac{1}{2}\sqrt{2}$ time and mark x. Otherwise, run it completely.

Theorem 3 *SL maintains a competitive ratio of*

$$R = 2\sqrt{2} - 1 + \frac{8\sqrt{2} - 11}{n} \approx 1.8284 + \frac{0.31371}{n},$$

where n is the number of machines.

Proof. We will give the proof in the next section.

5. Analysis of Algorithm SL

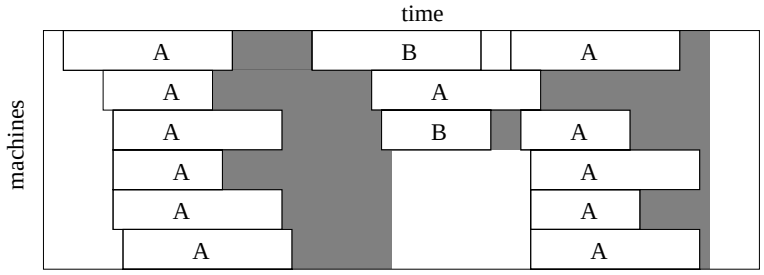

Figure 2: A run of SL

Below, we analyze the performance of algorithm SL, which was given in Section 4, and prove Theorem 3.

Consider a run of SL as in Figure 2. We introduce the following concepts.

- A job is of type A if at some moment during the execution of the job, all machines are used; otherwise it is of type B. (The jobs are marked accordingly in Figure 2.)
- *Lost earnings* are earnings of the adversary that SL misses. (In Figure 2, the lost earnings are marked grey.) Lost earnings are caused because jobs are not run or because they are run too short.
- A job or a set of jobs *compensates* for an amount x of lost earnings, if SL earns y on that job or set of jobs and $(x + y)/y \leq R$ (or $x/y \leq R - 1$). I. e., it does not violate the anticipated competitive ratio R.

A job of type B can only cause lost earnings when it is run short, because no job is refused during the time a job of type B is running. However, this causes at most $1 - \frac{1}{2}\sqrt{2}$ of lost earnings, so there is always enough compensation for these lost earnings from this job itself.

When jobs of type A are running, the adversary can earn more by running any short jobs among them longer. But it is also possible that jobs arrive while these jobs are running, so that they have to be refused, causing even more lost earnings. We will show that SL compensates for these lost earnings as well. We begin by deriving some general properties of SL.

Note first of all that if n jobs arrive simultaneously when all of SL's machines are empty, it serves s_0 of them short and earns $\frac{1}{2}s_0\sqrt{2} + (n - s_0) = (6 + 5\sqrt{2})n/14 \approx 0.93365n$. We denote this amount by x_0.

Properties of SL

1. Whenever a short job starts, a (long) job is marked that started earlier and that will finish later. This implies $l(t) \geq s(t)$ for all t.

2. When all machines are busy at some time t, SL earns at least x_0 from the jobs running at time t. (Since $s(t) \leq s_0$ at all times.)

3. Suppose that two consecutive jobs, a and b, satisfy that $t_b - t_a < 1 - \frac{1}{2}\sqrt{2}$ and that both jobs are long. Then $s(t_b) = s_0$ (and therefore $s(t_a) = s_0$), because b was run long although a was not marked yet.

Lemma 7 *If at some time t all machines are busy, at most $n - s_0$ jobs running at time t will still run for $\frac{1}{2}\sqrt{2}$ or more time after t.*

Proof. Suppose all machines are busy at time t. Consider the set L of (long) jobs that will be running for more than $\frac{1}{2}\sqrt{2}$ time, and suppose it contains $x \geq n - s_0 + 1$ jobs. We derive a contradiction.

Denote the jobs in L by $j_1, \ldots, j_x$, where the jobs are ordered by arrival time. At time t_{j_x}, the other jobs in L must have been running for less than $1 - \frac{1}{2}\sqrt{2}$ time, otherwise they would finish before time $t + \frac{1}{2}\sqrt{2}$. This implies that jobs in L can only be marked because short jobs started.

Also, if at time t_{j_x} we consider j_x not to be running yet, we know not all machines are busy at time t_{j_x}, or j_x would not have started. We have

$$n > s(t_{j_x}) + l(t_{j_x}) \geq s(t_{j_x}) + n - s_0,$$

so $s(t_{j_x}) < s_0$. Therefore, between times t_{j_1} and t_{j_x}, at most $s(t_{j_x}) \leq s_0 - 1$ short jobs can have been started and as a consequence, less than s_0 jobs in L are marked at time t_{j_x}. But then there is an unmarked job in L at time t_{j_x}, so j_x is run short. This contradicts $j_x \in L$.□

Definition A *critical interval* is an interval of time in which SL is using all its machines, and no jobs start or finish.

We call such an interval critical, since it is only in such an interval that SL refuses jobs, causing possibly much lost earnings. From Lemma 7, we see that the length of a critical interval is at most $\frac{1}{2}\sqrt{2}$.

We denote the jobs that SL runs during I by $j_1^I, \ldots, j_n^I$, where the jobs are ordered by arrival time. We denote the arrival times of these jobs by $t_1^I, \ldots, t_n^I$; I starts at time t_n^I. We will omit the superscript I if this is clear from the context. We denote the lost earnings that are caused by the jobs in I by X_I; we also sometimes say simply that X_I is caused by I. We say that a job sequence *ends with a critical interval*, if no more jobs arrive after the end of the last critical interval that occurs in SL's schedule.

Lemma 8 *If a job sequence ends with a critical interval I, and no other jobs besides $j_1^I, \ldots, j_n^I$ arrive in the interval $[t_1^I, \ldots, t_n^I]$, then SL can compensate for the lost earnings X_I.*

Proof. Note that j_1 is long, because a short job implies the existence of an earlier, long job in I by Property 1. SL earns at least x_0 from $j_1, \ldots, j_n$ by Property 2. There are three cases to consider, depending on the size and timing of j_2.

Case 1. j_2 is short. See Figure 3, where we have taken $t_2 = 0$. Note that j_1 must be the job that is marked when j_2 arrives, because any other existing jobs finish before I starts and hence before j_2 finishes. Therefore, $t_2 - t_1 < 1 - \frac{1}{2}\sqrt{2}$, so before time t_2 the adversary and

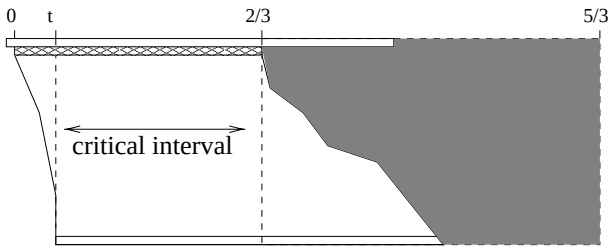

Figure 3: j_2 is short

SL earn less than $1-\frac{1}{2}\sqrt{2}$ from job 1. After time t_2, the adversary earns at most $(1+\frac{1}{2}\sqrt{2})n$ from $j_1, \ldots, j_n$ and the jobs that SL refuses during I. We have

$$(1+\frac{1}{2}\sqrt{2})n + (1-\frac{1}{2}\sqrt{2}) = R \cdot x_0,$$

so SL compensates for X_I.

Case 2. *j_2 is long and $t_2 - t_1 < 1 - \frac{1}{2}\sqrt{2}$.*

Since no job arrives between j_1 and j_2, we have by Properties 3 and 1 that $s(t_1) = s_0$ and $l(t_1) \geq s_0$. Denote the sets of these jobs by S_1 and L_1, respectively. All these jobs finish before I. (During I, SL does not start or finish any jobs.)

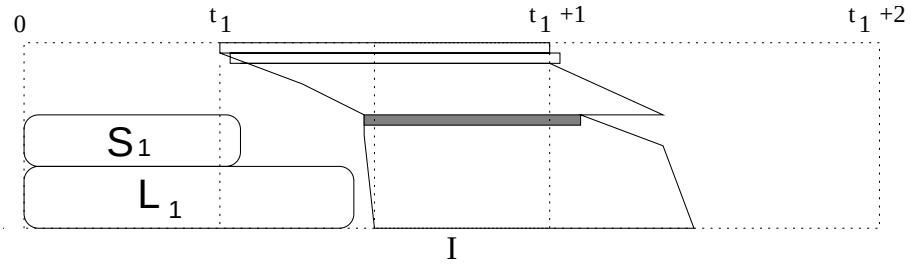

Figure 4: j_2 is long

Case 2a. *There is no critical interval while the jobs in S_1 and L_1 are running.*

Hence, the jobs in S_1 and L_1 are of type B. We consider the jobs that are running at time t_1 and the later jobs. Note that L_1 contains at least s_0 jobs, say it contains x jobs. After time t_1 the adversary earns at most $2n$, because I ends at most at time $t_1 + 1$. SL earns $\frac{1}{2}s_0\sqrt{2} + x$ from S_1 and L_1 and at least x_0 on the rest. For the adversary, we must consider only the earnings on S_1 and L_1 before time t_1; this is clearly less than $\frac{1}{2}s_0\sqrt{2} + x$.

We have

$$\frac{2n + \frac{1}{2}s_0\sqrt{2} + x}{x_0 + \frac{1}{2}s_0\sqrt{2} + x} < R \text{ for } x \geq s_0.$$

This shows SL compensates for X_I (as well as for the lost earnings caused by S_1 and L_1).

Case 2b. There exists a critical interval before I which includes a job from S_1 or L_1.

Call the earliest such interval I_2. If I_2 starts after t_1, we can calculate as in Case 2a. Otherwise, we consider the earnings on each machine after the jobs in I_2 started. Say the first job in S_1 starts at time t'. We have $t_n - t' < 1$. See Figure 5.

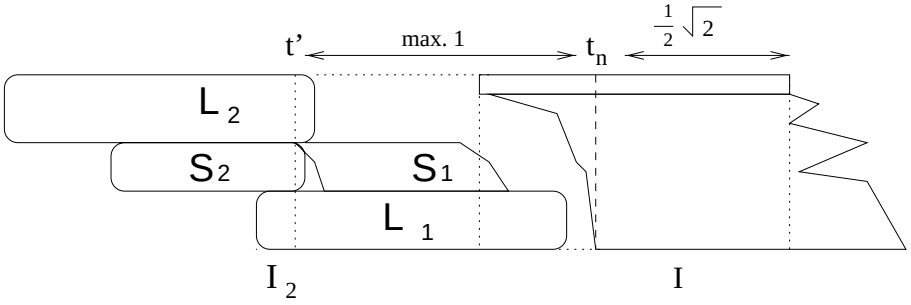

Figure 5: j_2 is long and there is another critical interval

Say I_2 contains x short jobs that are not in S_1 ($0 \le x \le s_0$). Then it contains $s_0 - x$ short jobs from S_1, and therefore at least $s_0 - x$ (long) jobs from L_1. This implies it contains at most $n - 2s_0 + x$ long jobs not from L_1. It also implies there are x short jobs in S_1 which are neither in I nor in I_2.

Using these observations, we can derive a bound on the earnings of the adversary and of SL from the jobs in I_2 and later. We divide their earnings into parts as illustrated in Figure 5 and have that the adversary earns at most

$$\begin{aligned} & (2+\frac{1}{2}\sqrt{2})n \text{ (after } t') \\ + \quad & n - 2s_0 + x \text{ (from the long jobs not in } L_1) \\ + \quad & (1-\frac{1}{2}\sqrt{2})s_0 \text{ (from } L_1 \text{ before } t') \\ + \quad & \frac{1}{2}x\sqrt{2} \text{ (from the short jobs not in } S_1) \\ = \quad & (3+\frac{1}{2}\sqrt{2})n - (1+\frac{1}{2}\sqrt{2})s_0 + x(1+\frac{1}{2}\sqrt{2}), \end{aligned}$$

while SL earns $2x_0$ (from the jobs in I and I_2) $+\frac{1}{2}x\sqrt{2}$ (from the x short jobs from S_1 between I_2 and I). We have

$$\frac{(3+\frac{1}{2}\sqrt{2})n - (1+\frac{1}{2}\sqrt{2})s_0 + x(1+\frac{1}{2}\sqrt{2})}{2x_0 + \frac{1}{2}x\sqrt{2}} \le R \text{ for } 0 \le x \le s_0$$

so SL compensates for all lost earnings after I_2.

Case 3. j_2 is long and $t_2 - t_1 \ge 1 - \frac{1}{2}\sqrt{2}$. We consider job j_3.

- If j_3 is short, then after time $t_1 + (1 - \frac{1}{2}\sqrt{2})$ the adversary earns at most $(1+\frac{1}{2}\sqrt{2})n - (n-2)((t_3 - t_1) - (1-\frac{1}{2}\sqrt{2})) - ((t_2 - t_1) - (1-\frac{1}{2}\sqrt{2}))$. Before that time, it earns of course $(1-\frac{1}{2}\sqrt{2})$ (only counting the jobs in I). So in total, it earns less than it did in Case 1.

- If j_3 is long, we have two cases. If $t_3 - t_2 < 1 - \frac{1}{2}\sqrt{2}$, then again the sets S_1 and L_1 are implied and we are in Case 2. Finally, if $t_3 - t_2 \geq 1 - \frac{1}{2}\sqrt{2}$ we know that $t_4 - t_3 < 1 - \frac{1}{2}\sqrt{2}$, so this reduces to Case 1 or 2 as well.

In all cases, we can conclude that SL compensates for X_I. □

Lemma 9 *If a job sequence ends with a critical interval I, then SL can compensate for the lost earnings X_I.*

Proof. We can follow the proof of Lemma 8. However, it is now possible that a short job j_1' starts after j_1, but finishes before I.

Suppose the first short job in I arrives at time $t' = t_1 + x$. If the job sets S_1 and L_1 exist, we can reason as in Case 2 of Lemma 8. Otherwise, all long jobs in I that arrive before time t_1' save one are followed by short jobs not in I. (If there are two such long jobs, they arrived more than $1 - \frac{1}{2}\sqrt{2}$ apart, and the adversary earns less than in Case 1 of Lemma 8 (cf. Case 3 of that lemma).)

For each pair (a_i, b_i), where a_i is long and $b_i \notin I$ is short, we have that b_i will run for at least $\frac{1}{2}\sqrt{2} - x$ more time after t', while a_i has run for at most x time. One such pair is shown in Figure 6.

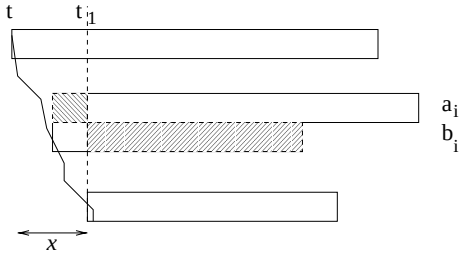

Figure 6: Pairs of long and short jobs

We compare the adversary's earnings now to its earnings in Case 1 of Lemma 8. Since $b_i \notin I$, it earns less on the machine running b_i and more on the machine running a_i (because there it earns something before time t', which was not taken into account earlier). If $x \leq \frac{1}{4}\sqrt{2}$, the adversary loses more on the machines running these pairs than it gains. On the other hand, if $x > 1 - \frac{1}{2}\sqrt{2}$, then I is shorter than $\frac{1}{2}\sqrt{2}$: the adversary earns $x - (1 - \frac{1}{2}\sqrt{2})$ less on every machine. □

It is possible that two or more critical intervals follow one another. In that case, we cannot simply apply Lemma 9 repeatedly, because some jobs may be running during two or more successive critical intervals. Thus, they would be used twice to compensate for different lost earnings. We now show that SL compensates for all lost earnings in this case as well.

Definition A *group* of critical intervals is a set $\{I_i\}_{i=1}^{k}$ of critical intervals, where I_{i+1} starts at most 1 time after I_i finishes ($i = 1, \ldots, k-1$).

Lemma 10 *If a job sequence ends with a group of two critical intervals, SL compensates for the lost earnings X_{I_1} and X_{I_2}.*

Proof. If all jobs from I_1 have finished before I_2 starts, we know X_{I_1} is compensated for by Lemma 9. On I_2 we can also use Lemma 9 to see that X_{I_2} is compensated for as well. Note that no jobs are used twice to compensate for lost earnings.

In case some jobs from I_1 are still running in I_2, I_2 ends within 1 time of the start of I_1. We denote the set of long jobs that are running in I_1 but not in I_2 by L_1, the set of long jobs in I_2 but not in I_1 by L_2 and the set of long jobs in both by L_3. Define S_1, S_2 and S_3 analogously. The sizes of these sets are denoted by l_1, l_2, l_3, s_1, s_2 and s_3, respectively. Of the s_1 jobs in S_1, s_{11} jobs had jobs in L_1 marked when they started, and s_{13} jobs had jobs in L_3 marked.

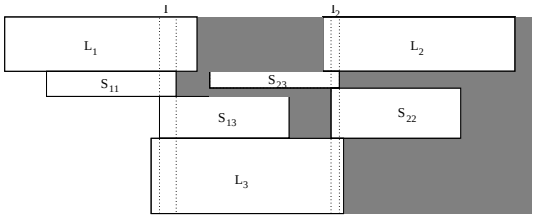

Figure 7: The sets of jobs in a run of SL

Case 1. If I_2 starts within $\frac{1}{4}\sqrt{2}$ time after I_1 has finished, the jobs that are in I_1 but not in I_2 can compensate for X_{I_1}, since the size of such a job is at least twice the size of the lost earnings following it on the same machine.

Case 2. Say I_2 starts after $\frac{1}{4}\sqrt{2}$, but within $\frac{1}{2}\sqrt{2}$ time after I_1 has finished. Ten $X_{I_1} \leq \frac{1}{2}\sqrt{2}(s_{11}+l_1)+(1-\frac{1}{2}\sqrt{2})s_{13}$. SL earns $l_1+\sqrt{2}(s_{11}+s_{13})/2$ from the jobs in S_1 and L_1. We have

$$\frac{\frac{1}{2}\sqrt{2}(s_{11}+l_1)+(1-\frac{1}{2}\sqrt{2})s_{13}}{l_1+\frac{1}{2}\sqrt{2}(s_{11}+s_{13})} \leq R-1 \text{ for all } s_{13},\ l_1 \text{ and } s_{11}.$$

(The worst case is $s_{13}=0$ and $l_1=s_{11}$, since $l_1 \geq s_{11}$). In other words, SL earns enough on S_1 and L_1 to compensate for X_{I_1}. Lemma 9 shows that X_{I_2} is compensated for too, again without using jobs twice to compensate for different things: S_1 and L_1 are not in I_2.

Case 3. Suppose I_2 starts more than $\frac{1}{2}\sqrt{2}$ time after I_1 has finished. We have $s_3=0$, $s_2 \leq s_0$ and $l_2 = n - l_3 - s_2$.

Case 3a. If $l_3 \leq s_0$, then $l_2 \geq n-2s_0 \geq n/R$. In a figure like Figure 7, we can move jobs around to aid in the calculations, without affecting profits. Here we move all the jobs in L_2 and S_2 forward, so that they start 1 time after the end of I_1. We then see that after that time, SL earns at least n/R while the adversary earns n, and before that time, we can apply Lemma 9.

Case 3b. The last case is $l_3 > s_0$ and I_2 starts more than $\frac{1}{2}\sqrt{2}$ after I_1. Since all jobs in L_3 start within $1-\frac{1}{2}\sqrt{2}$ time (because I_1 and I_2 are at least $\frac{1}{2}\sqrt{2}$ apart), and $l_3 > s_0$, it must be that $s_1 = s_0$, because each job in L_3 is available to be marked for a short job before I_1 starts. But then SL already earns $x_0+\frac{1}{2}s_0\sqrt{2}$ from S_1, L_2, S_2 and L_3 alone, because it earns

at least x_0 from the jobs in I_2. Starting at the beginning of I_1, the adversary earns at most $2n$. Since

$$\frac{2n}{x_0 + \frac{1}{2}s_0\sqrt{2}} = R,$$

we have that SL compensates for both X_{I_1} and X_{I_2}. □

Lemma 11 *If a job sequence ends with a group of critical intervals, SL compensates for all the lost earnings after the first critical interval.*

Proof. We use an induction on the number of critical intervals k. We have already proven the cases $k = 1$ and $k = 2$. Say $k \geq 3$, and consider the last three critical intervals, I_1, I_2 and I_3. See Figure 8.

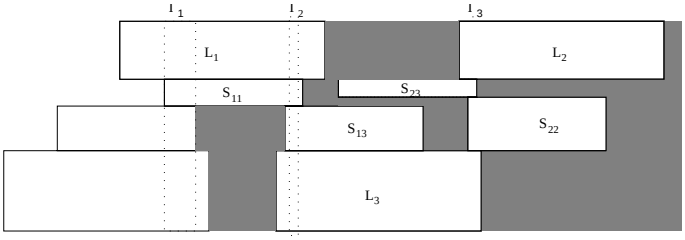

Figure 8: A long sequence of critical intervals

There are a number of cases to consider. We will abuse notation and use I_i to refer both to the i-th critical interval and the jobs that SL runs during I_i.

Case 1. There are no short jobs in $I_1 \cap I_2$. Denote the number of long jobs in $I_2 \cap I_3$ by l_3.
Case 1a. There are no short jobs in $I_2 \cap I_3$.
Case 1a1. $l_3 \leq s_0$. Now the jobs in $I_3 \backslash I_2$ are worth at least n/R and (using induction) we are done as in Lemma 10.
Case 1a2. $l_3 > s_0$. As in Lemma 10 we have that $s_2 = s_0$, where s_2 is the number of the short jobs in I_2. We find that these jobs together with the jobs in I_3 compensate for the lost earnings after I_2, and can use an induction for the rest, because we never need these jobs to compensate for earlier lost earnings.
Case 1b. There are short jobs in $I_2 \cap I_3$. The adversary earns at most $(1 + \frac{1}{2}\sqrt{2})n$ after I_2, while SL earns at least x_0 from all jobs after I_1. We are again done using induction.

Case 2. There are short jobs in $I_1 \cap I_2$.
Case 2a. $I_1 \cap I_3 = \emptyset$. By induction we know that the jobs up to and including those from I_1 can compensate for lost earnings until 1 time after the end of I_1. Since I_2 ends at most $\frac{1}{2}\sqrt{2}$ after the start of I_1, the jobs after I_1 have to compensate only for the lost earnings starting from $1 - \frac{1}{2}\sqrt{2}$ after I_2. From then on, the adversary earns at most $(1 + \frac{1}{2}\sqrt{2})n$ while SL earns at least x_0 from the jobs in I_3.
Case 2b. $I_1 \cap I_3 \neq \emptyset$. We have that I_3 finishes at most 1 after I_1 starts and we can treat this as a special case of two critical intervals, thus reducing this case to $k - 1$ critical intervals. Note that the analysis in Lemma 10 does not assume that there are no critical intervals between I_1 and I_2. □

Theorem 4 *SL maintains a competitive ratio of* $R = 2\sqrt{2} - 1 + \frac{8\sqrt{2}-11}{n}$.

Proof. If no jobs arrive within 1 time after a critical interval, the machines of both SL and the adversary are empty. New jobs arriving after that can be treated as a separate job sequence. Thus we can divide the job sequence into parts. The previous lemmas also hold for such a part of a job sequence.

Consider a (part of) a job sequence. All the jobs arriving after the last critical interval can be disregarded, since they are of type B: they compensate for themselves. Moreover, they can only decrease the amount of lost earnings caused by the last critical interval (if they start less than 1 after a critical interval).

If there is no critical interval, we are done. Otherwise, we can apply Lemma 11 and remove the last group of critical intervals from consideration. We can then remove the jobs of type B at the end and continue in this way to show that SL compensates for all lost earnings. □

6. Extensions of this model

6.1 Fixed Levels

In this section, we study the case where jobs can only be run at two levels [4, 11]. This reduces the power of the adversary and should lower the competitive ratio. If the jobs can have different sizes, the proofs from Section 3 still hold. For the case of uniform jobs, we have the following bound.

Theorem 5 *If jobs can be run at two levels,* $\alpha < 1$ *and 1, then no algorithm can have a better competitive ratio than* $1 + \alpha - \alpha^2$.

Proof. Note that each job is run either for 0, α or 1 time. Let n jobs arrive at time $t = 0$. Say A serves ϕn jobs partially and the rest completely. It earns $(1 - \phi + \alpha\phi)n$. If this is less than $n/(1 + \alpha - \alpha^2)$ we are done. Otherwise, we have $\phi \leq \frac{\alpha}{1+\alpha-\alpha^2}$. Another n jobs arrive at time $t = \alpha$. A earns at most $(1 + \alpha\phi)n$ in total, while the offline algorithm can earn $n + n\alpha$. Since $\phi \leq \frac{\alpha}{1+\alpha-\alpha^2}$, we have $r(A) \geq \frac{1+\alpha}{1+\alpha\phi} \geq 1 + \alpha - \alpha^2$. □

Note that for $\alpha = \frac{1}{2}\sqrt{2}$, SL yields a competitive ratio for this problem of at most 1.828 (but probably much better). Extending these results to more values of α is an open problem.

6.2 Nonlinear rewards

In many applications, it is reasonable to assume that serving a job partially, say for time α, an algorithm earns more than α (but strictly less than 1). For instance, if a video is transmitted using only half of all the pixel data, the perceived quality is much more than 1/2 and therefore the algorithm should earn more, say 3/4.

Consider rewards of the form $f(\alpha)$, where $f(0) = 0, f(1) = 1$, $f(\alpha) > \alpha$ and $f''(\alpha) < 0$ (concave) for all $0 < \alpha < 1$. One example is

$$f(\alpha) = \sqrt{\alpha}.$$

If α is fixed, the only change that this causes in the lower bounds is that all earnings of α are replaced by $f(\alpha)$. The timing of the jobs by the adversary does not change, and the structure of the proofs remains unchanged.

7. Conclusions and Future Work

We have studied the problem of scheduling jobs that do not have a fixed execution time on-line. We have first considered the general case with different job sizes.

Subsequently, we have given a randomized lower bound of 1.5 and a deterministic algorithm with competitive ratio ≈ 1.828 for the scheduling of uniform jobs. An open question is by how much either the lower bound or the algorithm could be improved. Especially using randomization it could be possible to find a better algorithm.

An extension of this model is to introduce either deadlines or startup times, limiting either the time at which a job should finish or the time at which it should start. Finally, algorithms for fixed level servicing can be investigated.

8. Acknowledgement

The authors wish to thank Peter Bosch for useful discussions.

References

1. S. Aggarwal, J.A. Garay, and A. Herzberg. Adaptive video on demand. In *Proc. 3rd Annual European Symp. on Algorithms*, LNCS, pages 538–553. Springer, 1995.
2. S.K. Baruah and M.E. Hickey. Competitive on-line scheduling of imprecise computations. *IEEE Trans. On Computers*, 47:1027–1032, 1998.
3. H. G. P. Bosch, N. Nes, and M. L. Kersten. Navigating through a forest of quad trees to spot images in a database. Technical Report INS-R0007, CWI, Amsterdam, February 2000.
4. E.K.P. Chong and W. Zhao. Performance evaluation of scheduling algorithms for imprecise computer systems. *J. Systems and Software*, 15:261–277, 1991.
5. T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of AAAI*, pages 49–54, 1988.
6. Wu-Chen Feng. Applications and extensions of the imprecise-computation model. Technical report, University of Illinois at Urbana-Champaign, December 1996.
7. K.J.Lin, S. Natarajan, and J.W.S. Liu. Imprecise results: Utilizing partial computations in real-time systems. In *Proc. IEEE Real-Time Systems Symp.*, pages 255–263, 1998.
8. W.-K. Shih. Scheduling in real-time systems to ensure graceful degradation: the imprecise-computation and the deferred-deadline approaches. Technical report, University of Illinois at Urbana-Champaign, December 1992.
9. W.-K. Shih and J.W.S. Liu. On-line scheduling of imprecise computations to minimize error. *SIAM J. on Computing*, 25:1105–1121, 1996.
10. A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 12th ACM Symposium on Theory of Computing*, 1980.
11. W. Zhao, S. Vrbsky, and J.W.S. Liu. Performance of scheduling algorithms for multiserver imprecise systems. In *Proc. Fifth Int. Conf. Parallel and Distributed Computing and Systems*, 1992.

12. S. Zilberstein. Constructing utility-driven real-time systems using anytime algorithms. In *Proc. 1st IEEE Workshop on Imprecise and Approximate Computation*, 1992.