Partial Servicing of On-Line Jobs

Rob van Stee* and Han La Poutré

Centre for Mathematics and Computer Science (CWI)
Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands
(e-mail: {rvs,hlp}@cwi.nl)

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.

1 Introduction

Partial execution or computation of jobs has been an important topic of research in several papers [2, 4–9, 12, 13]. 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].

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,

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

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 $1 + \alpha - \alpha^2$.

This is an extended abstract in which we do not give complete proofs. For more details, we refer to the full paper [10].

1.1 Related Work

We give a short overview of some related work.

In overloaded real-time systems, *imprecise computation*[8, 6, 7] is a well-known method to ensure graceful degradation. On-line scheduling of imprecise computation jobs is studied in [9, 2], but mainly on task sets that already satisfy the *(weak) feasible mandatory constraint*: at no time may a job arrive which makes it infeasible to complete all mandatory subtasks (for the offline algorithm). This is quite a strong constraint. *Anytime algorithms* are introduced in [5] and studied further in [13]. This is a type of algorithm that may be interrupted at any point, returning a result with a quality that depends on the execution time.

In [4], a model similar to the one in this paper is studied, but on a single machine and using stochastic processes and analysis in stead of competitive 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. The execution of jobs can either be switched from one level to the other, or 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. 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 receive 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 [12], this model is studied on more machines, again using probabilistic analysis.

2 Definitions and notations

By n, we denote the number of machines. The performance measure 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

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

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.

Lemma 1. *If job sizes can vary without bound, no algorithm that schedules 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: $x_1 = 1, x_2 = r, x_i = r^{i-1} (i = 3, \ldots, n), x_{n+1} = 2r(1 + \ldots + r^{n-1})$. 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 $1+r+\ldots+r^{i-2}$, while the adversary earns $1 + r + \ldots + r^{i-1}$. We have

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

This implies A must accept the first n jobs. However, it then earns at most $1 + \ldots + r^{n-1}$. 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.

If for all job sizes x we have $1 \leq x \leq M$, we can use similar methods to those used in studying the video on demand problem studied in [1] to give lower and upper bounds for our problem.

In [1], a central server has 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.

Several technical adjustments in both the proof of the lower bound and in the construction of the algorithm `Harmonic` are required. We refer to the full paper[10] for details.

Theorem 1. *Let r be the optimal competitive ratio of this scheduling problem with different job sizes. Then $r = \Omega(\ln M)$. For $M = \Omega(2^n)$, we have $r = \Omega(n(\sqrt[n]{M} - 1))$.* `Adapted Harmonic`, *which requires $n = \Omega(MH_M)$, has a competitive ratio of $O(\ln M)$.*

4 Uniform job sizes

We will now study the case of identical job sizes. For convenience, we take the job sizes to be 1. In this section we allow that the scheduling algorithm is completely free in choosing how long it serves any job. 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.

Lemma 2. *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. We refer to the full paper [10].

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

Theorem 2. *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 [11].

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, then 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 $r(A) \geq 3/2$. □

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*

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

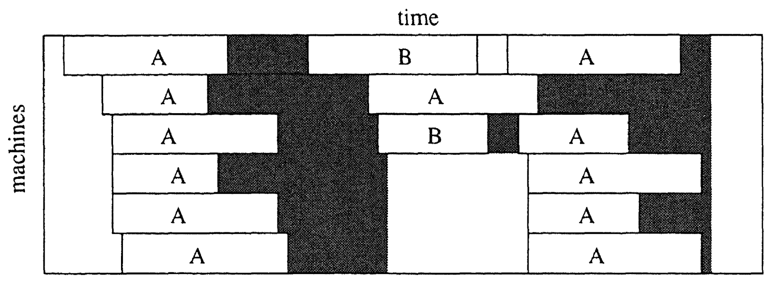

Fig. 1. A run of SL

Consider a run of SL as in Figure 1. 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 1.)
- *Lost earnings* are earnings of the adversary that SL misses. (In Figure 1, 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 idle, 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 3. *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 3, 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 4. *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 2, where we have taken $t_2 = 0$. Note that j_1 must

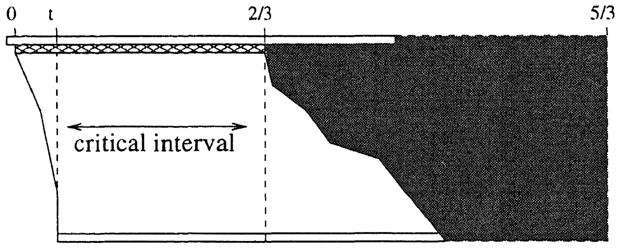

Fig. 2. j_2 is short

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 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.)

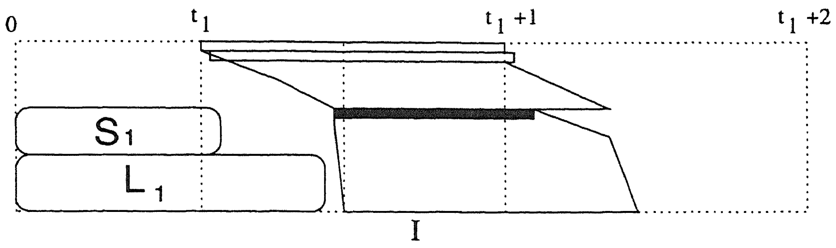

Fig. 3. j_2 is long

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 4.

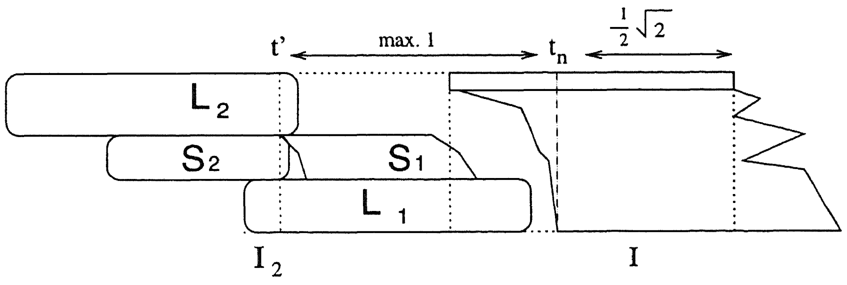

Fig. 4. j_2 is long and there is another critical interval

Say I_2 contains x short jobs that are not in S_1 $(0 \leq x \leq s_0)$. 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 4 and have that the adversary earns at most

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

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

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

so SL compensates for all lost earnings after I_2.

Case 3. j_2 is long and $t_2 - t_1 \geq 1 - \frac{1}{2}\sqrt{2}$. 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 $t_3 - t_2 < 1 - \frac{1}{2}\sqrt{2}$, 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 5. *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 4. 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 4. 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 4 (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 5.

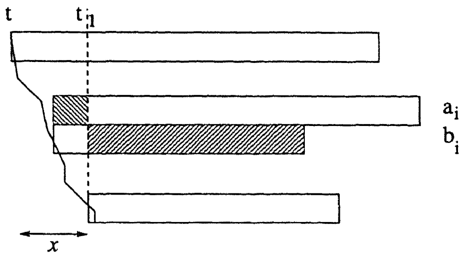

Fig. 5. Pairs of long and short jobs

We compare the adversary's earnings now to its earnings in Case 1 of Lemma 4. 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 5 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 show in the full paper 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 6. *If a job sequence ends with a group of critical intervals, SL compensates for all the lost earnings after the first critical interval.*

Proof. The proof consists of showing that in all cases, the lost earnings between and after the critical intervals are small compared to SL's earnings on the jobs it runs. A typical case is shown in Figure 6. For details, see [10]. □

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 6 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. □

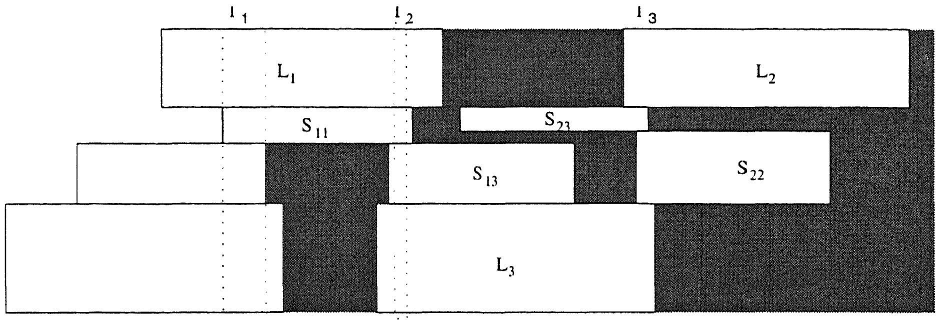

Fig. 6. A sequence of critical intervals

6 Fixed Levels

Finally, we study the case where jobs can only be run at two levels [4,12]. 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.

Theorem 5. *Let r be the optimal competitive ratio of this scheduling problem with different job sizes and two fixed run levels. Then $r = \Omega(\ln M)$. For $M = \Omega(2^n)$, we have $r = \Omega(n(\sqrt[n]{M} - 1))$.* Adapted Harmonic, *which requires $n = \Omega(MH_M)$, has a competitive ratio of $O(\ln M)$.*

Proof. We refer to the full paper [10].

For the case of uniform jobs, we have the following bound.

Theorem 6. *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.

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, where methods from [1] can be used. 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. R. van Stee and J. A. La Poutré. On-line partial service of jobs. Technical Report SEN-R00xx, CWI, Amsterdam, in preparation.
11. A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 12th ACM Symposium on Theory of Computing*, 1980.
12. W. Zhao, S. Vrbsky, and J.W.S. Liu. Performance of scheduling algorithms for multi-server imprecise systems. In *Proc. Fifth Int. Conf. Parallel and Distributed Computing and Systems*, 1992.
13. S. Zilberstein. Constructing utility-driven real-time systems using anytime algorithms. In *Proc. 1st IEEE Workshop on Imprecise and Approximate Computation*, 1992.