Speed is as Powerful as Clairvoyance

Bala Kalyanasundaram*
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260

Kirk Pruhs†
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260

Abstract

We consider several well known nonclairvoyant scheduling problems, including the problem of minimizing the average response time, and best-effort firm real-time scheduling. It is known that there are no deterministic online algorithms for these problems with bounded (or even polylogarithmic in the number of jobs) competitive ratios. We show that moderately increasing the speed of the processor used by the nonclairvoyant scheduler effectively gives this scheduler the power of clairvoyance. Furthermore, we show that there exist online algorithms with bounded competitive ratios on all inputs that are not closely correlated with processor speed.

1 Introduction

We consider several well known nonclairvoyant scheduling problems, including the problem of minimizing the average response time [13, 15], and best-effort firm real-time scheduling [1, 2, 3, 4, 8, 11, 12, 18]. (We postpone formally defining these problems until the next section.) In nonclairvoyant scheduling some relevant information, e.g. when jobs will arrive in the future, is not available to the scheduling algorithm A. The standard way to measure the adverse effect of this lack of knowledge is the competitive ratio:

$$\max_I \frac{A(I)}{Opt(I)}$$

where $A(I)$ denotes the cost of the schedule produced by the online algorithm A on input I, and $Opt(I)$ denotes the cost of the optimal schedule. The competitive ratio for a problem is then

$$\min_A \max_I \frac{A(I)}{Opt(I)}$$

where the min is over all online algorithms. The standard way to interpret the competitive ratio is as the payoff to a game played between an online algorithm and an all-powerful malevolent adversary that specifies the input I.

One of the primary goals of any analysis is to identify what works well in practice. Competitive analysis has been criticized because it often yields ratios that are unrealistically high for "normal" inputs and as a result it can fail to identify the class of online algorithms that work well. The scheduling problems that we consider are good examples of this phenomenon in that their competitive ratios are unbounded, while there are simple nonclairvoyant algorithms that perform reasonably well in practice. We explain this phenomenon by adopting what we call the *weak adversary model*, which assumes that the speed of the processor used by the nonclairvoyant scheduler is $(1+\epsilon)$ times the speed of the processor used by the clairvoyant adversary, where $\epsilon > 0$. We define the *ϵ-weak competitive ratio* of a problem to be:

$$\min_A \max_I \frac{A_{1+\epsilon}(I)}{Opt_1(I)}$$

where the subscripts denote the speed of the processor used by the corresponding algorithm.

The original motivation for the standard competitive ratio was to use the divergence of the online algorithm's output from optimal as a measure of the adverse effect of nonclairvoyance. Not only does the ϵ-weak competitive ratio give us another measure, but also suggests a practical way to combat the adverse effect of nonclairvoyance. If a problem has a small ϵ-weak competitive ratio for some moderate ϵ then this means that a moderate increase in processor speed will

*Supported in part by NSF under grant CCR-9202158.
kalyan@cs.pitt.edu, http://www.cs.pitt.edu/~kalyan
†Supported in part by NSF under grant CCR-9209283.
kirk@cs.pitt.edu, http://www.cs.pitt.edu/~kirk

effectively buy the power of clairvoyance. Therefore, the weak adversary model gives the system designer a practical way, increasing the speed of the processor, to improve the performance of the system.

On "normal" inputs, one would intuitively expect that the offline performance of the system would not degrade drastically if the speed of the processor is increased slightly. If an algorithm has a bounded ϵ-weak competitive ratio then it has a bounded competitive ratio for all inputs I where $Opt_1(I)/Opt_{1+\epsilon}(I)$ is bounded. Thus an algorithm with a bounded ϵ-weak competitive ratio has a bounded competitive ratio for all inputs that fall under this formulation of "normal".

We give algorithms for these scheduling problems that have ϵ-weak competitive ratios that are solely a function of ϵ, and not the input I. Furthermore, as ϵ increases the ϵ-weak competitive ratios quickly approach one.

2 Previous and Current Results

Our generic scheduling problem consists of a collection $\mathcal{J} = \{J_1, \ldots J_n\}$ of independent jobs to be run on a single processor. (While these results extend to the multiprocessor setting, we restrict our attention to a single processor for simplicity.) Each job J_i has a *release time* r_i and a *length* x_i. J_i can not be run before time r_i. The time required to complete J_i is x_i divided by the speed of the processor. We assume that the online/nonclairvoyant scheduler is not aware of J_i until time r_i. We consider only *preemptive scheduling*, that is, a job can always be restarted from the point of last execution. We assume that such context switches require no time.

The problem of minimizing the average response time of the jobs is a well known and widely studied problem in operating system scheduling (see for example [7, 14]). We assume that the nonclairvoyant scheduler does not learn x_i at time r_i, and more generally, can not deduce x_i until it has run J_i to completion. The *completion time* c_i of a job J_i is the time at which J_i has been allocated enough time to finish execution. Similarly, the *response time* is $w_i = c_i - r_i$, and the *idle time* for a speed s processor is $w_i - x_i/s$. For the problem of minimizing the *average response time*, $\frac{1}{n}\sum_{i=1}^{n} w_i$, the deterministic competitive ratio is $\Omega(n^{1/3})$, and the randomized competitive ratio is $\Omega(\log n)$ [15]. It can easily be shown that any algorithm that doesn't unnecessarily idle the processors has a competitive ratio of $O(n)$. Surprisingly, this is the best known upper bound on the competitive ratio, even allowing randomization. The competitive ratio for the commonly used Round Robin algorithm is $\Omega(n/\log n)$ [13, 15].

In section 3, we first consider the queue size as a function of time. Define $Q_A(t, s)$ as the set of jobs that have been released before time t, but have not been finished by algorithm A by time t assuming that A is using a speed s processor. We show that for every nonclairvoyant scheduling algorithm A there is an input I and a time t such that $|Q_A(t, 1)|/|Q_{Opt}(t, 1)| = n/2$, where $|\cdot|$ denotes set cardinality. We then give an online algorithm Balance, B for short, that guarantees that at all times that $|Q_B(t, 1+\epsilon)| \leq \frac{1+\epsilon}{\epsilon}|Q_{Opt}(t, 1)|$. This implies that Balance has an ϵ-weak competitive ratio of $1 + \frac{1}{\epsilon}$ for the problem of minimizing the average response time. In contrast, we show that the ϵ-weak competitive ratio of Round Robin is $\Omega(n^{1-\epsilon})$, for $0 < \epsilon < 1$. We then assume that the nonclairvoyant scheduler is equipped with a unit speed processor and an ϵ speed processor, instead of a $(1+\epsilon)$ speed processor. (Here we are assuming $\epsilon < 1$.) In this case we give a nonclairvoyant scheduling algorithm Balance2 with average response time at most $1+\frac{1}{\epsilon}$ times the average response time of the adversary. This means that a nonclairvoyant scheduler with a supercomputer and an old 386 PC can be constant competitive against a clairvoyant scheduler with only a supercomputer. Finally, we demonstrate that Balance is *fair* every job it sees by proving that the maximum idle time of Balance is quite comparable to that of offline.

In best-effort firm real-time scheduling each job J_i now has a deadline d_i, and a benefit b_i, in addition to a release time and an execution time. It is also useful to define the *value density* of a job J_i to be $v_i = b_i/x_i$, and the *laxity* of J_i for a speed s processor to be $d_i - x_i/s - r_i$, which is the maximum amount J_i can be delayed if it is to be completed. Since real-time systems are embedded systems, the scheduler is generally aware in advance of the jobs that it may receive. Thus, the standard assumption is that at time r_i the scheduler learns of x_i, d_i, and b_i. If J_i is finished by time d_i then the algorithm receives a benefit of b_i, otherwise no benefit is gained from this job. The goal of the scheduler is to maximize the total benefit of the jobs that it completes. Since this is a maximization problem, the competitive ratio definitions in the introduction have to be modified by inverting the ratios. So for example, the competitive ratio for this problem is then

$$\min_A \max_I \frac{Opt(I)}{A(I)}$$

The deterministic competitive ratio for this problem is $\Theta(\Phi)$ [3, 4, 11, 18], and the randomized competitive ratio is $\Theta(\min(\log \Phi, \log \Delta))$ [8, 12], where the *importance ratio* Φ is the ratio of the maximum value density of a job to the minimum value density of a job, and Δ is the ratio of the length of the longest job to the length of the shortest job. The competitive ratio is unbounded even in the special case that each $b_i = 1$ [2].

In section 4, we first assume that both the nonclairvoyant scheduler and the adversary have unit speed processors, and that the laxity of each job J_i is at least ϵx_i. An upper bound on the standard competitive ratio, under this laxity assumption, will also upper bound the ϵ-weak competitive ratio since any job J_i that doesn't have laxity at least ϵx_i for a $(1+\epsilon)$ speed processor can't be finished by a unit speed processor. This formulation also has the added advantage of showing the effect of laxity. Under these laxity assumptions, we give an algorithm SLACKER that has a competitive ratio that is only a function of ϵ, and approaches three as ϵ increases. These results show that if a real-time system is designed so that every job has laxity that is a reasonable fraction of the execution time of that job, then the resulting competitive ratio is reasonably small. The effect of laxity on the competitive ratio in the special case of $\Phi = 1$ was considered in [1, 6]. We then show that the ϵ-weak competitive ratio for SLACKER approaches one as ϵ increases.

The weak adversary model, comparing an online algorithm against a less powerful but more knowledgeable adversary, has been considered before in query-response problems such as the k-server problem and its special cases (e.g. [17, 19]), and online weighted matching [9]. In each case the adversary is handicapped by having fewer servers. One can argue that the weak competitive ratio is essentially what is called the comparative ratio in [10]. However, the results in [10] are really of a different flavor in that they are primarily concerned with the effect of partial clairvoyance. Other methods have been suggested to address the limitations of competitive analysis. These methods include restricting the input distribution to satisfy some special properties (e.g. [10, 16]), and comparing the cost of a solution produced by an online algorithm on input I to the worst-case optimal cost of any input of the same size as I [5].

3 Average Response Time

The following well known lemma explains why we first consider the queue size.

Lemma 3.1 For any scheduling algorithm A with a speed s processor, the total response time is $\int_0^\infty |Q_A(t,s)|dt$.

Lemma 3.2 For every nonclairvoyant scheduling algorithm A there is an input I and a time t such that $|Q_A(t,1)|/|Q_{Opt}(t,1)| = n/2$.

Proof Sketch: Let $t_0 = 0$ and $t_i = t_{i-1} + 1/2^{i-1}$, $1 \leq i \leq n-1$. Two jobs arrive at time t_0. One job arrives at time t_i, $1 \leq t \leq n-1$. The adversary sets the jobs lengths so that online hasn't finished any jobs by time t_{n-1}. One can show that if the adversary always runs the shorter job, then it will always have at most two active jobs. The key point to note is that at time t_i, $i \geq 0$, the sum S of the remaining unfinished lengths of the two jobs that the adversary has in its queue satisfies $1/2^i \leq S \leq 1/2^{i-1}$. ■

We now give an online algorithm BALANCE that guarantees that its queue size is not too much more than optimal under the weak adversary scenario.

Algorithm BALANCE : For any job J_i and time t we define $||J_i||_t$ to be the amount of time that BALANCE has executed J_i before time t. At all times t, BALANCE splits the processing time equally among all jobs J_i that have minimum $||J_i||_t$. ■

Our analysis of BALANCE is based on the following lemma.

Lemma 3.3 Let B be the algorithm BALANCE. At any point t in time, $|Q_B(t, 1+\epsilon)| \leq \frac{1+\epsilon}{\epsilon}|Q_{Opt}(t,1)|$.

The proof of this lemma follows from the ensuing chain of reasoning. Let U_B be the set of jobs unfinished by BALANCE, and U_A be the set of jobs unfinished by the adversary at time t mentioned in Lemma 3.3. Intuitively, the adversary can use time that BALANCE spent on jobs in U_A to finish jobs in $U_B - U_A$. We need to show that the weak adversary assumption means that in order to borrow enough time to finish jobs in $U_B - U_A$ it must be the case that U_A is reasonably large. We say that a job J_i can *immediately borrow* from another job J_j, denoted by $J_i \hookleftarrow J_j$, if BALANCE ran J_j at some time t' satisfying $r_i \leq t' \leq c_i$; So if $J_i \in U_B - U_A$ and $J_j \in U_A$ then at time t' the adversary could have been executing J_i while BALANCE was executing J_j. The *borrow* relation, denoted $J_i \overset{*}{\hookleftarrow} J_j$, is the transitive closure of the immediately borrow relation. We define $B(J_i) = \{J_j : J_i \overset{*}{\hookleftarrow} J_j\}$. Intuitively, if the adversary transfers some time from a $J_j \in U_A$ to a $J_i \in U_B - U_A$ then $J_j \in B(J_i)$.

Lemma 3.4 Let J_i be a job that BALANCE saw but did not complete by time t. For any job $J_j \in B(J_i)$, $||J_i||_t \geq ||J_j||_t$.

Proof Sketch: Suppose there is a job $J_j \in B(J_i)$ such that $\| J_j \|_t > \| J_i \|_t$. If there are many such jobs J_j, then select the one that can be reached from J_i by a shortest path P in the directed unweighted graph induced by the relation immediately borrow. Let J_k be the job in P immediately before J_j. So $J_k \hookleftarrow J_j$. By the definition of P, $\|J_k\|_t \leq \|J_i\|_t$. Furthermore let x be the last time that J_j was run before time t. Then notice that it must be the case $c_k > x$ or J_k would not be J_j's predecessor in P. Hence, $\|J_j\|_t = \|J_j\|_{\min(t,c_k)}$. By the definition of BALANCE, if $J_k \hookleftarrow J_j$, then $\| J_k \|_y \geq \| J_j \|_y$ for some time y before c_k. Hence, we deduce $\| J_k \|_t \geq \| J_j \|_t$. We reach a contradiction since $\|J_k\|_t \leq \|J_i\|_t$ and $\|J_i\|_t < \|J_j\|_t$. ∎

Let $U_B - U_A = \{J_i : 1 \leq i \leq p\}$. Since the adversary completes jobs in $U_B - U_A$, the total time spent by BALANCE on jobs in U_A must be at least $\sum_{i=1}^{p} \epsilon \, \|J_i\|_t$. We partition the time that BALANCE spent on jobs in U_A into p classes $T_1, \ldots, T_p$ such that the cumulative time in the ith class is at least $\epsilon \, \|J_i\|_t$. Note that T_i could be a collection of time intervals. We call a partition *good* if there is no job $X \in U_A$ such that some portion of time spent by BALANCE on X is included in T_i and $\|X\|_t > \|J_i\|_t$.

Lemma 3.5 There always exists a good partition.

Proof Sketch: For a job $X \in U_B - U_A$, let $t(X)$ be the earliest release time of a job in $B(X)$. Let $U_B - U_A = \{J_i : 1 \leq i \leq p\}$ where the indexing is such that $t(J_i) \geq t(J_j)$ if and only if $i \leq j$. By the definition of the relation *immediately borrow*, it must be the case that any job executed by BALANCE during the time interval $[t(J_i), t]$ must be a member of $B(J_i)$. Also, observe that for each i in the range $1 \leq i \leq p$, offline must run and finish jobs in $B(J_i) - U_A$ only during the interval $[t(J_i), t]$. Therefore, for each $1 \leq i \leq p$, it must be the case that the cumulative amount of time spent by BALANCE to jobs in $B(J_i) \cap U_A$ must be at least

$$\sum_{j=1}^{j=i} \epsilon \, \|J_j\|_t$$

Observe that if $i \leq j$ then $t(J_i) \geq t(J_j)$ and consequently $B(J_i) \subseteq B(J_j)$. So, by induction on i, time spent on jobs in U_A by BALANCE can be distributed to jobs in $U_B - U_A$ such that for each job $J_i \in U_B - U_A$ gets $\epsilon \, \|J_i\|_t$ units of time from jobs in $B(J_i) \cap U_A$. The result follows since, by lemma 3.4, $\|X\|_t \leq \|J_i\|_t$ for any $X \in B(J_i)$. ∎

The proof of the lemma 3.5 intuitively suggests that, if the adversary is going to finish a job $J_i \in U_B - U_A$, then it needs to raise $\epsilon \, \| J_i \|_t$ units of times from jobs $J_j \in U_B \cap B(J_i)$. By lemma 3.4 we know that $\|J_j\|_t \leq \|J_i\|_t$. This suggests that we analyze the following problem to get lower bound on number of jobs in U_B.

The Politician Problem: There are n politicians trying to raise money from m contributors. The ith politician must raise ϵS_i dollars, and the j contributor has C_j dollars to contribute. The *election rule* says that the jth contributor can contribute to the ith politician only if $C_j \leq S_i$. A politician can raise money from many contributors and a contributor can give money to several politicians.

Lemma 3.6 If there is a solution for the above Politicians Problem then $\epsilon n \leq m$.

Proof Sketch: Let $X(i,j)$ be the contribution from the the j contributor to the i politician. Since $X(i,j)/\epsilon S_i$ is the fraction of ϵS_i that the ith politician got from the j contributor, and since every politician is successful, it is the case that,

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{X(i,j)}{\epsilon S_i} = n$$

Now by the election rule,

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{X(i,j)}{\epsilon S_i} \leq \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{X(i,j)}{\epsilon C_j} = \sum_{j=1}^{m} \sum_{i=1}^{n} \frac{X(i,j)}{\epsilon C_j} \leq \frac{m}{\epsilon}$$

Here $\frac{X(i,j)}{\epsilon C_j}$ is the fraction of C_j that the j contributor gave to the ith politician. ∎

Proof Sketch: (Lemma 3.3) Applying the lemma 3.5 we know that for each job $J_i \in U_B - U_A$, we must find $\epsilon \, \|J_i\|_t$ units of times from jobs $J_j \in U_B \cap B(J_i)$. By lemma 3.4 $\|J_j\|_t \leq \|J_i\|_t$. Now applying lemma 3.6 to this case, we get $\frac{|U_A|}{\epsilon} \geq |U_B - U_A|$. Then since, $|U_A| \geq |U_B| - |U_B - U_A|$, we get $(1 + \frac{1}{\epsilon})|U_A| \geq |U_B|$. ∎

The following theorem then follows by lemma 3.1.

Theorem 3.1 The ϵ-weak competitive ratio of BALANCE with respect to average response time is at most $1 + \frac{1}{\epsilon}$.

We now show that the commonly used algorithm Round Robin [7, 14] does not have a constant ϵ-weak competitive ratio for small ϵ. Round Robin splits the processing time evenly among all unfinished jobs.

Lemma 3.7 For the problem of minimizing the average response time, the ϵ-weak competitive ratio of Round Robin is $\Omega(n^{1-\epsilon})$ for $0 < \epsilon < 1$.

Proof Sketch: We divide time into stages. Let the ith stage, $i \geq 0$ start at time t_i. We let $t_0 = 0$, and $t_1 = 1+\epsilon$. There are two jobs of length $(1+\epsilon)$ released at time t_0, and one job is released at each time t_i, $i \geq 1$, with length $s(i)$ that is exactly the same length as Round Robin has left on each of the previous jobs. To guarantee that the adversary can finish the job released at time t_i by time t_{i+1}, $i \geq 1$, we let $t_{i+1} = t_i + s(i)$. We then get the recurrence

$$s(i+1) = s(i) - \frac{(1+\epsilon)s(i)}{i+2}$$

Expanding this we get $s(i) = \Theta(1/i^{1+\epsilon})$. The total response time for the adversary is then $\Theta(\sum_{i=1}^{n} 1/i^{1+\epsilon})$, which is $\Theta(1)$. The total response time for Round Robin is then $\Theta(\sum_{i=1}^{n} i/i^{1+\epsilon})$, which is $\Theta(n^{1-\epsilon})$. ■

The following theorem shows that BALANCE does not overly delay any job to improve the performance of average response time.

Theorem 3.2 The ϵ-weak competitive ratio of BALANCE with respect to maximum idle time is $\frac{1}{\epsilon}$.

Proof Sketch: Let t be the time at which BALANCE completed a job J_i for which the idle time is maximum. By shortening other unfinished jobs, let us assume that BALANCE completed all jobs by time t. Let $d = t- \| J_i \|_t - r_i$ be the idle time experienced by J_i when using BALANCE. By lemma 3.4, the amount of time spent by BALANCE on any job during (r_i, t) is at most $\| J_i \|_t$. Due to the difference in speed, it must be the case that the adversary finished a job J_j at time $(1+\epsilon)t$. If BALANCE did not run J_j during (r_i, t), then $\|J_j\|_t \leq t - d$. On the other hand if BALANCE ran J_j during (r_i, t), then $\|J_j\|_t \leq \|J_i\|_t \leq t - d$. Therefore, $\|J_j\|_t$ is at most $t - d$. Notice that J_j must have arrived on or before $t- \|J_j\|_t$. Therefore, the idle time incurred by the adversary for J_j must be at least $\epsilon(t- \|J_j\|_t) \geq \epsilon d$. ■

We now assume that the nonclairvoyant scheduler equipped with a unit speed processor and an ϵ speed processor can be almost as competitive as if it was equipped with a $(1+\epsilon)$ speed processor. Here we are assuming $\epsilon < 1$. We further assume for simplicity that $1/\epsilon$ is an integer.

Algorithm BALANCE2 : Run the job J_i that has been run the least on the unit speed processor. Run the job, other than J_i, that has been run the least on the ϵ speed processor. ■

The analysis of BALANCE2 follows the same line as the analysis of BALANCE. We modify the definition of *immediately borrow* in the following way. A job J_i can *immediately borrow* time from another job J_j if at some time t', satisfying $r_i \leq t' \leq c_i$, BALANCE2 was running J_j while either, BALANCE2 was not running J_i, or was running J_i on a processor that is slower than the processor that J_j was being run on. We define U_A, U_B, *borrow*, and $B(J_i)$ as before. We define $\|J_i\|_t$ to be the initial length of J_i that has been executed before time t by BALANCE2.

Lemma 3.8 Let J_i be a job that BALANCE saw but did not complete by time t. For any job $J_j \in B(J_i)$, $\|J_i\|_t \geq \|J_j\|_t$.

The Modified Politicians Problem: There are n politicians and m original contributors. Let $S_1 \geq \ldots \geq S_n \geq 0$. The ith politician, $1 \leq i \leq n$ must refund at least ϵS_{i+1} dollars to the contributors. The jth contributor requires C_j dollars in refunds. The *election rule* says that the ith politician can refund money to the jth contributor only if $C_j \leq S_i$.

Lemma 3.9 If there is a solution for this modified politicians problem then $m \geq \epsilon(n-1)$.

Proof Sketch: Assume without loss of generality that $C_1 \geq \ldots \geq C_m$. Assume that S_{i_1} refunds to C_{j_1} and S_{i_2} refunds to C_{j_2}, with $i_1 \leq i_2$ and $j_2 \leq j_1$ We call such a situation a *swap*. By transitivity we can have S_{i_1} refund to C_{j_2} and S_{i_2} refund to C_{j_1}. It is easy to see that we can assume without loss of generality that there is a solution to the modified politician problem without any *swaps*. Let us multiply the refund of each politician by a factor of $1/\epsilon$. Simultaneously, we also increase the pool of potential contributors by a factor of $1/\epsilon$ by replacing each original contributor by $1/\epsilon$ identical contributors. By repeating the previous assignment $1/\epsilon$ times, the politicians can still be successful in refunding the money.

We prove by induction that for any i, $1 \leq i \leq n$, there are at least i contributors among the m/ϵ contributors that get refunds from politicians 1 through i, and those i contributors will not get refunds from other politicians. Assume $i = 1$. If $C_1 \leq S_2$, then C_1 gets all of its refunds from the first politician by the no swapping assumption. Otherwise, if $C_1 > S_2$, then C_1 cannot accept refunds from politicians $2 \ldots n$. Now assuming that the hypothesis holds for $i-1$, we show that the hypothesis also holds for i. By the induction hypothesis, contributors 1 through $i-1$ cannot get a refund from the ith politician. Let $C_{\alpha(i)}$ be the highest contributor that got a refund from the ith politician. If $C_{\alpha(i)} > S_{i+1}$, then $C_{\alpha(i)}$ cannot get a refund from politicians $i+1 \ldots n$. On the other hand, if $C_{\alpha(i)} \leq S_{i+1}$ then the i contributor gets all of its refund from the i politician by the no swapping assumption. ■

Lemma 3.10 Let B be the algorithm BALANCE2. At any point t in time, $|Q_B(t, 1+\epsilon)| - 1 \leq \frac{1+\epsilon}{\epsilon}|Q_{Opt}(t, 1)|$.

Proof Sketch: Once again we are going to reduce to the modified politicians problem, where the members of U_A are the contributors, and the members of $U_B - U_A$ are the politicians. Let $U_B - U_A = \{J_{\alpha(i)} : 1 \leq i \leq p\}$, $r_{\alpha(i)} \leq r_{\alpha(j)}$ for $i \leq j$. So the jobs are ordered by increasing order of release times. Since at least two jobs are unfinished during the time interval $(r_{\alpha(2)}, t)$, it must be the case that BALANCE2 was running both processors throughout this period.

Let $i \geq 2$ and assume that BALANCE2 was running two jobs J_a and J_b at a time t' between time $r_{\alpha(i)}$ and t. Then we claim $J_a, J_b \in B(J_{\alpha(i-1)}) \cup \{J_{\alpha(i-1)}\}$. If neither J_a or J_b is $J_{\alpha(i-1)}$ then both are in $B(J_{\alpha(i-1)})$ by definition. Otherwise if say $J_a = J_{\alpha(i-1)}$, then all J_b, J_a, and $J_{\alpha(i)}$ are all being executed in round robin fashion, and the claim once again follows. Notice that $B(J_a) \subseteq B(J_{\alpha(i-1)})$.

Assume that the length of each job $J_{\alpha(i)} \in U_B - U_A$ is exactly $\|J_{\alpha(i)}\|_t$. This is the best case for the adversary. Then BALANCE2 and the adversary executed the same length of each job in $(U_B - U_A) \cup (\bar{U}_A \cap \bar{U}_B)$. Hence, the extra $\epsilon(t - r_{\alpha(2)})$ work (the refunds) done by BALANCE2 must go to jobs in U_A. Consider how this might be distributed. For each $2 \leq i \leq p$, it must be the case that the extra work must during the period $(r_{\alpha(i)}, r_{\alpha(i+1)})$ must be distributed to jobs in $B(J_{\alpha(i-1)}) \cap U_A$. We think of $J_{\alpha(i-1)}$ as giving this refund to jobs in $B(J_{\alpha(i-1)}) \cap U_A$. The election rule is satisfied by lemma 3.8. We now apply the modified politicians lemma, and the rest of the argument is as before. ■

Theorem 3.3 The average response time for BALANCE2, with a unit speed processor and an $\epsilon < 1$ speed processor, is at most $1 + \frac{1}{\epsilon}$ times the average response time of an adversary given only a unit speed processor.

Proof Sketch: This theorem follows by applying lemma 3.10 and noting that the adversary must be running some jobs for a duration of $\epsilon\ell$ even after BALANCE completed all jobs at time ℓ. ■

4 Real-time Scheduling

Before describing the algorithm SLACKER we need to introduce some definitions and notation. Recall that we first assume that both the nonclairvoyant scheduler and the clairvoyant adversary have unit speed processors, and that the laxity of each job J_i is at least ϵx_i. For notational convenience, let $\delta = \epsilon/2$. A job J_i is *viable* at time t for a scheduling algorithm A if A can finish J_i before d_i, that is, if A has run J_i for at least $x_i - (d_i - t)$ units of time. Define the *slack* of a job J_i at time t by $s_i(t) = d_i - x_i - t$. A job is *fresh* at time t if $s_i(t) > (1+\delta)x_i$. Otherwise, we say the job is *stale* at time t. Let $c > (1+\delta)/\delta$ be some constant that we define later. Define the *density class* of a job J_i to be $\lfloor \log_c v_i \rfloor$. Assuming that we normalize so that the smallest value density is one, the density classes then range from 0 to $\lfloor \log_c \Phi \rfloor$. If X is a set of jobs then let $\|X\| = \sum_{J_i \in X} b_i$ denote the total benefit of jobs in X. Let *Opt* be the set of jobs finished by the adversary.

Algorithm SLACKER: At time r_i SLACKER switches to J_i if and only if J_i is in a higher density class than the job J_j that SLACKER is currently running. If this happens, J_j is saved as the *representative job* for density class $\lfloor \log_c v_j \rfloor$. If SLACKER finishes a job J_i at time t, then let α be the largest integer such that there is currently a fresh job in density class α or a viable representative job in class α. If there is a viable representative job J_i in density class α then SLACKER resumes execution of J_i. Otherwise, SLACKER starts executing an arbitrary fresh job in density class α. ■

Let S be the set of jobs completed by SLACKER, and R the set of jobs run by SLACKER. S may be a proper subset of R since SLACKER may not finish every job that it started.

Lemma 4.1 Let $J_i \in R - S$ be an arbitrary representative job in density class α that SLACKER did not complete. Then for a period of at least $d_i - r_i - x_i - \delta x_i \geq \delta x_i$ units of time between r_i and d_i SLACKER was running a job in density class $\alpha + 1$ or higher.

Lemma 4.2 Assuming the density of each job is an integral power of c, $(1 - \frac{1}{\delta(c-1)}) \|R\| \leq \|S\|$.

Proof Sketch: We imagine that each job $J_i \in R$ has an account associated with it. The account for a job $J_i \in S$ initially starts with b_i points. All other accounts start with zero points. We redistribute points from accounts of jobs in S to accounts of jobs in $R - S$. The argument is by reverse induction on the density classes. First note that SLACKER finishes every job in density class $\lfloor \log_c \Phi \rfloor$ that it begins. Assume we now are considering jobs in density class α. Let $J_i \in R-S$ be a job that was the representative job for density class α between time t_1 and t_2. If SLACKER ran a job J_j in density class $\beta > \alpha$ for t units of time between time t_1 and t_2, then tc^α/δ points are transferred from J_j's account to J_i's account. By lemma 4.1, J_i borrowed for a total time of at least δx_i, and hence borrowed at least $c^\alpha x_i$ points. Thus the account for J_i now contains at least b_i points.

We now need to examine how much the account of a job J_i in density class α can be depleted by jobs in lower density classes. The representative jobs in density class $\beta < \alpha$ take at most $c^\beta x_i/\delta$ points from the account of J_i. Hence, the number of points remaining in the account for J_i is at least

$$c^\alpha x_i - x_i \sum_{\beta=0}^{\alpha-1} c^\beta/\delta \geq x_i c^\alpha (1 - \frac{1}{\delta(c-1)}) = b_i(1 - \frac{1}{\delta(c-1)})$$

■

Lemma 4.3 Assuming the density of each job is an integral power of c, $||Opt|| \leq \frac{1+3\delta}{\delta}\ ||R||$

Proof Sketch: Assume that if the adversary ran a job with density c^α for t units of time, we credit the adversary with tc^α points regardless of whether it finished the job or not. Define a job to be *dense* if it has density c^α or greater. We then show that the total amount of time the adversary spent running dense jobs is at most $(1+3\delta)/\delta$ times the time that SLACKER was running dense jobs. We divide time up in the following way. Let σ_0 be the first point of time where SLACKER starts running a dense job. Let τ_i, $i \geq 0$, be the first point of time after σ_i where SLACKER is not running a dense job. Let σ_i, $i \geq 1$, be the first point in time after after τ_{i-1} that SLACKER begins running a dense job. Note that no dense job can arrive between any τ_i and σ_{i+1}. Consider the longest dense job J_j that arrived between a σ_i and a τ_i, and that SLACKER did not run. Then J_i had to have been stale at time τ_i. Hence, $d_j \leq \tau_i + (1+\delta)x_j$, and $(\tau_i - \sigma_i) \geq \delta x_j$. This means that the time that the adversary is running dense jobs that SLACKER didn't run is at most $\frac{1+2\delta}{\delta}$ times the time that SLACKER was running dense jobs. We must add one to this ratio for the jobs that both the adversary and SLACKER ran. The lemma then follows by reverse induction on α. ■

Theorem 4.1 Under the assumption that every job J_i has laxity at least $\epsilon x_i = 2\delta x_i$, the competitive ratio of SLACKER is at most

$$\frac{(1+3\delta)c(c-1)}{\delta(c-1)-1}$$

Proof Sketch: Result follows by applying lemma 4.2 and lemma 4.3, and by removing the condition that the density of a job is an integral power of c. ■

One can verify that by letting $c = \frac{1+\delta}{\delta} + \frac{1}{\sqrt{\delta}}$, we get a bounded competitive ratio for all $\delta > 0$, and that the competitive ratio goes to three as δ increases. If we go back to assuming that SLACKER has a $(1+\epsilon)$ speed processor, we can show that the ϵ-weak competitive ratio approaches one by modifying lemma 4.3 as follows. The proof is very similar to the proof of lemma 4.3.

Lemma 4.4 Assuming the density of each job is an integral power of c, and that SLACKER has a $(1+\epsilon)$ speed processor, $||Opt|| \leq (\frac{1}{1+2\delta} + \frac{1+\delta}{\delta(1+2\delta)} + 1)\ ||R||$

5 Conclusion

We believe that the weak adversary model will be useful in identifying online algorithms that work well in practice for other types of problems. It is important that the weakening of the adversary should be done in such a way that the corresponding strengthening of online algorithm can be achieved in practice. It is worth mentioning that for the problems considered in this paper, increasing the speed of the online processor is not the only way to weaken the adversary. For example, in the case of real-time scheduling, it suffices to design the real-time system in such a way that the laxity condition is satisfied.

Finally, we would like to mention that the weak adversary model has been used recently to show that the natural greedy algorithm, which works reasonably well in practice, is almost optimal for online weighted matching [9]. Traditional competitive analysis shows a bound of $\Theta(2^m)$ whereas the weak adversary analysis yields a bound of $\Theta(\log m)$ where m is the size of the graph.

Acknowledgements: The second author would like to thank Daniel Mosse, Rege Colwell, Richard Suchoza, and Dimitri Zorine for many helpful discussions on real-time scheduling.

References

[1] S. Baruah, and J. Haritsa, "On improved performance guarantees through the use of slack times," Tech. Report, Computer Science Dept., University of Vermont.

[2] S. Baruah, J. Haritsa, and N. Sharma, "On-line scheduling to maximize task completions" Real-time Systems Symposium, 1994.

[3] 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", *Journal of Real-Time Systems*, **4**, 124–144, 1992.

[4] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload", IEEE FOCS, 101–110, 1991.

[5] S. Ben-David, and A. Borodin, "A new measure for the study of on-line algorithms", *Algorithmica*, **11**, 73–91, 1994.

[6] R. Colwell, R. Suchoza, D. Zorine, "On-line real-time scheduling with laxities," Manuscript, 1995.

[7] P. Galvin, and A. Silverschatz, *Operating System Concepts*, Addison-Wesley, 1994.

[8] B. Kalyanasundaram, and K. Pruhs, "Fault-tolerant real-time scheduling", Technical Report, CS Dept., University of Pittsburgh, 1994.

[9] B. Kalyanasundaram, and K. Pruhs, "The on-line transportation problem", to appear, *European Symposium on Algorithms*, 1995.

[10] E. Koutsoupias, and C. Papadimitriou, "Beyond competitive analysis," FOCS, 394–400, 1994.

[11] G. Koren, and D. Shasha, "D^{over}:An optimal on-line scheduling algorithm for overloaded real-time systems", IEEE Real-time Systems Symposium, 290–299, 1992.

[12] G. Koren, and D. Shasha, "MOCA: a multiprocessor on-line competitive algorithm for real-time system scheduling," Theoretical Computer Science, **128**, 75–97, 1994.

[13] T. Matsumoto, "Competitive analysis of the round robin algorithm', *International Symposium on Algorithms and Computation*, 71 – 77, 1992.

[14] M. Milenkovic, *Operating Systems: Concepts and Designs*, McGraw-Hill, 1992.

[15] R. Motwani, S. Phillips, E. Torng, "Non-clairvoyant scheduling", *Proceedings of the ACM/SIAM Symposium on Discrete Algorithms*, 422 – 431, 1993.

[16] P. Raghavan, "A statistical adversary for on-line algorithms", *Online Algorithms*, DIMACS Series in Discrete Mathematics and Computer Science, 7:79–83, 1992.

[17] D. Sleator, and R. Tarjan, "Amortized efficiency of list update and paging rules", *Communications of the ACM*, **28**, 202 – 208, 1985.

[18] G. Woeginger, "On-line scheduling of jobs with fixed start and end time", *Theoretical Computer Science*, **130**, 1994.

[19] N. Young, "The k-server dual and loose competitiveness for paging," *Algorithmica*, **11**, 525–541, 1994.