Abstract

We introduce resource augmentation as a method for analyzing online scheduling problems. In resource augmentation analysis the online scheduler is given more resources, say faster processors or more processors, than the adversary. We apply this analysis to two well known online scheduling problems, the classic uni-processor CPU scheduling problem $1|r_i, pmtn|\sum F_i$, and the best-effort firm real-time scheduling problem $1|r_i, pmtn|\sum w_i(1 - U_i)$. It is known that there are no constant competitive online algorithms for these problems. We show that there are simple online scheduling algorithms for these problems that are constant competitive if the online scheduler is equipped with a slightly faster processor than the adversary. Thus a moderate increase in processor speed effectively gives the online scheduler the power of clairvoyance. Furthermore, the online scheduler can be constant competitive on all inputs that are not closely correlated with processor speed. We also show that the performance of an online scheduler in best-effort real time scheduling can be significantly improved if the system is designed in such a way that the laxity of every job is proportional to its length.

Speed is as Powerful as Clairvoyance[1]

Bala Kalyanasundaram[2]
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260

Kirk Pruhs[3]
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260

[1]A preliminary version of this paper appeared in the 1995 IEEE Symposium on Foundations of Computer Science

[2]Supported in part by NSF under grant CCR-9202158. kalyan@cs.pitt.edu, http://www.cs.pitt.edu/~kalyan

[3]Supported in part by NSF under grant CCR-9209283. kirk@cs.pitt.edu, http://www.cs.pitt.edu/~kirk

1 Introduction

1.1 Motivation

In online scheduling problems the jobs arrive over time, and the online scheduler S must make scheduling decisions without knowledge of the future. The standard way to measure the adverse effect of nonclairvoyance is the competitive ratio. For a minimization problem, the competitive ratio of an algorithm S is

$$\rho(n) = \max_I \frac{S(I)}{A(I)}$$

where $S(I)$ denotes the cost of the schedule produced by the algorithm S on input I, $A(I)$ denotes the cost of the optimal schedule, and the maximum is over all inputs I with n jobs. 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 A that specifies the input I, and schedules I optimally.

Competitive analysis has been criticized as being deficient for the following reasons:

- It often yields ratios that are unrealistically high for "normal" inputs.
- It often fails to differentiate between online algorithms that are empirically observed to have significantly different performance.
- The optimally competitive algorithms often seem unnecessarily complicated.

These deficiencies lead us to suggest resource augmentation analysis as a method for analyzing scheduling problems. In resource augmentation analysis the online scheduler is given more resources, say faster processors or more processors, than the adversary. For concreteness, consider augmenting the online algorithm with a faster processor in a uni-processor scheduling problem. We say that algorithm S is an s-speed c-approximation algorithm if

$$\max_I \frac{S_s(I)}{A_1(I)} \leq c$$

where the subscripts denote the speed of the processor used by the corresponding algorithm. This terminology, as well as the term resource augmentation, was suggested in follow-up paper by Phillips, Stein, Torng, and Wein [28]. Resource augmentation has the following benefits:

- It gives the system designer a practical way, buying more or faster processors, to combat the adverse effect of nonclairvoyance.
- In our analysis, it allows us to exclude those abnormal instances where the value of objective function changes drastically in response to a small change in processor speed.

We elaborate on these two benefits in the following three paragraphs.

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. If the optimal competitive ratio is $\omega(1)$, this is generally interpreted as saying that it is not possible to bound the relative cost of nonclairvoyance. But we argue that this is not quite correct. If there is an an $O(1)$-speed $O(1)$-approximation online algorithm for some problem, then this says that in some sense you can bound the the relative cost of nonclairvoyance. That is, by having recourses that are only a constant factor better, the online algorithm can bound the relative cost of its solution with respect to optimal. If there is an $f(\frac{1}{\epsilon})$-speed $(1+\epsilon)$-approximation online algorithm for some problem, and $f(x)$ is a reasonably slowly growing function, then this means that a moderate increase in processor speed will effectively buy the power of clairvoyance.

Systems designers seek ways to build a system so that it performs well. An analysis is useful if it suggests practical ways to improve the performance of the system. Resource augmentation analysis can suggest practical ways to improve performance besides buying more or better resources. For example, in the best-effort effort real time scheduling problem our resource augmentation analysis suggests that the online scheduler can perform well if the laxity of every job is at least a constant fraction of its length.

If S be an $(1+\epsilon)$-speed $f(\frac{1}{\epsilon})$-approximation algorithm, then S has a bounded competitive ratio for all inputs I where $A_1(I)/A_{1+\epsilon}(I)$ is bounded. 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. Thus S has a bounded competitive ratio for all inputs that fall under this formulation of "normal".

We apply resource augmentation analysis to the online version of two well-known scheduling problems:

- the classic uni-processor CPU scheduling problem $1|r_i, pmtn|\sum F_i$ [12, 25, 27], and
- the best-effort firm real-time scheduling problem $1|r_i, pmtn|\sum w_i(1-U_i)$ [2, 3, 4, 5, 14, 20, 21, 32].

It is known that there are no constant competitive online algorithms for these problems. We give simple online algorithms for these problems with the property that if the online algorithm is given a $(1+\epsilon)$ speed processor then the online algorithm is $f(\frac{1}{\epsilon})$-competitive (here $f(x)$ is a function that goes to one as x increases). We believe that this is evidence that, for many online scheduling problems, resource augmentation analysis suffers less from the deficiencies mentioned above than does competitive analysis.

The first example of resource augmentation analysis in the literature is apparently the analysis of the classic paging/caching problem in the famous paper [31]. In [31] it is shown that it is possible for an online paging algorithm to be constant competitive if it is equipped with a constant fraction more caches than the adversary. Prior to this paper, all of the examples of resource augmentation in the literature, that we are aware of, deal with online server problems, such as the k-server problem (e.g. [33]), and online weighted matching [15]. In each case the online algorithm is given more servers. Other methods have been suggested to address the limitations of competitive analysis in the context of server problems (e.g. [6, 18, 29]).

General background on scheduling, including the three field scheduling notation, can be found in [9, 23]. Background on online algorithms can be found in [8]. Background on online scheduling can be found in [30]. Background on approximation algorithms for offline scheduling problems can be found in [13].

1.2 Basic Definitions

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. Each job J_i has a *release time* r_i and a *length* x_i. The job 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. An *online* scheduler is not aware of J_i until time r_i. A *nonclairvoyant* scheduler is an online scheduler that learns nothing else about J_i at time r_i other than that it is released. A *online clairvoyant* scheduler learns all of the relevant information about J_i, such as x_i, at time r_i. We consider only *preemptive scheduling*, that is, a job can always be restarted from the point of last execution. We make the standard assumption that such context switches require no time. The *completion time* $c_i(S)$ of a job J_i is the time at which J_i has been allocated enough time to finish execution by the algorithm S. The *response time*, or equivalently *flow time*, is $F_i(S) = c_i(S) - r_i$. The *idle time* for a speed s processor is $F_i(S) - x_i/s$. If S is understood, then we drop it from the notation. If a job J_i is run on an s speed processor for y units of time, then we say that the amount of *work* done on J_i is sy steps.

Definition 1 Let S be an algorithm, and s be the speed of the processor used by S. We define $U_S(t, s)$ as the set of jobs that have been released before time t, but have not been finished by algorithm S by time t.

Definition 2 Given an online algorithm, we say that a job is *active* if it has been released but not finished by the online algorithm.

1.3 Average Response Time

In section 2, we consider the nonclairvoyant version of $1|r_i, pmtn|\sum F_i$, the problem of minimizing the average response time of a collection of preemptable jobs on a uni-processor system. For this problem, the optimal deterministic competitive ratio is $\Omega(n^{1/3})$, and the optimal randomized competitive ratio is $\Theta(\log n)$ [16, 27]. It can easily be shown that any algorithm that doesn't unnecessarily idle the processors has a competitive ratio of $O(n)$. The competitive ratio for the Round Robin algorithm, which shares the processor equally among all jobs, is $\Theta(n/\log n)$ [11, 25, 27]. This is the best known upper bound on the competitive ratio for deterministic algorithms.

In subsection 2.1, we consider an online algorithm that we call Balance, which always runs the job that has been run the least to date. The algorithm Balance is also known as Shortest-Elapsed-Time-First in the folklore. We then show that Balance guarantees that at all times t:

$$|U_B(t, 1+\epsilon)| \leq \frac{1+\epsilon}{\epsilon}|U_A(t, 1)|$$

where B is shorthand for Balance. This immediately gives us that Balance is a $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-approximation algorithm.

In section 2.2, we want to show that a second $\epsilon < 1$ speed processor is just as valuable to a nonclairvoyant scheduler as a $(1+\epsilon)$ speed processor. Note that a $(1+\epsilon)$ speed processor can always simulate a unit speed processor and an ϵ speed processor, but the reverse is not necessarily true. In this case, we give a nonclairvoyant scheduling algorithm BALANCE2 that is a slight modification of BALANCE. We show that if $\frac{1}{\epsilon}$ is an integer, then BALANCE2 with a unit speed processor and an ϵ speed processor is, like BALANCE, $1+\frac{1}{\epsilon}$ competitive against an adversary with a single unit speed processor.

Finally, in subsection 2.3, we present three results. We show that for every nonclairvoyant deterministic scheduling algorithm S there is an input I and a time t such that $|U_S(t,1)|/|U_A(t,1)| = n/2$. Thus one can not prove that a nonclairvoyant deterministic algorithm is $o(n)$-competitive by bounding the number of active jobs at every time. We show that if Round Robin is equipped with a $(1+\epsilon)$ speed processor, $0 < \epsilon < 1$, then there are instances where the average response time for Round Robin is $\Omega(n^{1-\epsilon})$ times optimal. So a small increase in speed does not allow Round Robin to be constant competitive. Finally, we demonstrate that BALANCE is a $(1+\epsilon)$-speed $\frac{1}{\epsilon}$-approximation algorithm with respect to maximum idle time. Thus BALANCE does not obtain good average case response time by starving some jobs.

1.4 Real-time Scheduling

In section 3, we consider the online clairvoyant version of $1|r_i, pmtn|\sum w_i(1-U_i)$, the problem of maximizing the benefit of the jobs completed before their deadline on a uni-processor system with preemption. Here, each job J_i has a deadline d_i, and a benefit b_i, in addition to a release time and an execution time. Since real-time systems are generally embedded systems, the scheduler is generally aware of the characteristics of a job at the time of its release, and thus the standard assumption is that the online scheduling learns x_i, d_i, and b_i at time r_i. If J_i is finished by time d_i then the algorithm accrues a benefit of b_i, otherwise no benefit is accrued. The goal of the scheduler is to maximize the total benefit of the jobs that it completes. Since this is a maximization problem, the ratio definitions in the introduction have to be modified by inverting the ratios. So for example, S is an s-speed c-approximation algorithm if and only if

$$\max_I \frac{A_1(I)}{S_s(I)} \le c$$

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 to be $d_i - x_i - r_i$. The laxity is the maximum amount of time that J_i can be delayed if it is to be completed before its deadline on a unit speed processor. The optimal deterministic competitive ratio for this problem is $\Theta(\Phi)$ [4, 5, 20, 32], and the randomized competitive ratio is $\Theta(\min(\log\Phi, \log\Delta))$ [14, 21], 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 optimal deterministic competitive ratio is unbounded even in the special case where each $b_i = 1$ [3].

In section 3, we present an online algorithm SLACKER. Roughly speaking, SLACKER tries to run the highest density job that still retains a constant fraction of its original laxity.

In subsection 3.1, we first assume that both the online scheduler and the adversary have unit speed processors, and that the laxity of each job J_i is at least ϵx_i. If an algorithm S is σ-competitive, under this laxity assumption, then S is essentially a $(1+\epsilon)$-speed σ-approximation algorithm. The reason for this is that if a job J_i can't be delayed by at least ϵx_i on a $(1+\epsilon)$ speed processor, then J_i can't be finished by a unit speed processor by its deadline. This formulation also has the added advantage of showing the effect of laxity. Under these laxity assumptions, we show that SLACKER is $f(\frac{1}{\epsilon})$-competitive, where $f(x)$ is a somewhat complicated function that 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 [2, 10].

In subsection 3.2, we extend these results to show that SLACKER is a $(1+\epsilon)$-speed $g(\frac{1}{\epsilon})$-approximation algorithm, where $g(x)$ is a somewhat complicated function that approaches one as ϵ increases.

2 Average Response Time

In this section we consider the nonclairvoyant version of $1|r_i, pmtn|\sum F_i$. Without loss of generality we re-scale time so that the length of the smallest job is 1.

2.1 Faster Processor

In this subsection we assume that the online scheduler has a $1+\epsilon$ speed processor and that adversary has a unit speed processor. We describe an algorithm BALANCE, and show that it is a $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-approximation algorithm. For any job J_i, and time t, we define $w(i,t)$ to be the amount of work done on J_i by BALANCE before time t.

Algorithm BALANCE : At all times t, BALANCE splits the processing time equally among all jobs J_i that have minimum $w(i,t)$.

We use B as shorthand for BALANCE, and A for the optimal adversary scheduler. We also use c_j as shorthand for $c_j(B)$.

Observation 3 If $w(i,u) \leq w(j,u)$ and $u \geq r_j$, then for all $t \in [u, c_j]$ it is the case that $w(i,t) \leq w(j,t)$.

The following simple lemma explains why we turn our attention to bounding $|U_B(t,s)|$.

Lemma 4 For any scheduling algorithm S with a speed s processor, the total response time is $\int_0^\infty |U_S(t,s)|dt$.

Proof: During any period δ of time where there are k active jobs, the response time of each of these k jobs increases by δ. ∎

Hence, to show that an algorithm is an $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-approximation algorithm it is sufficient to show that for all inputs, and for all times t, $|U_B(t, 1+\epsilon)| \leq (1+\frac{1}{\epsilon})|U_A(t,1)|$. To accomplish this, we fix some arbitrary input and some arbitrary time t. We will only be interested in the state of BALANCE and the state of the adversary at or before time t. So unless we explicitly state otherwise, all times are implicitly assumed to be at or before time t. Also we only consider the collection $\mathcal{J}$ of jobs released at or before time t, since jobs released after time t do not affect the state of any scheduling algorithm at time t. For simplicity we use U_B as shorthand for $U_B(t, 1+\epsilon)$, and U_A as shorthand for $U_A(t, 1)$. We will also on occasion use $w(i)$ as shorthand for $w(i,t)$. Our goal for the rest of this subsection is to show that $\epsilon|U_B - U_A| \leq |U_A|$. This will immediately allow us to conclude that $|U_B| \leq (1+\frac{1}{\epsilon})|U_A|$, since $|U_B| - |U_A| \leq |U_B - U_A|$. Before preceding we will need the following definitions.

Definition 5
(a) We say that a job J_i can *immediately borrow* from a job J_j $(i \neq j)$, denoted by $J_i \hookleftarrow J_j$, if BALANCE ran J_j at some time t' satisfying $r_i \leq t' \leq \min(t, c_i)$.
(b) If $J_i \hookleftarrow J_j$ then we say that the *time certificate* of this is the earliest time $f(i,j)$, satisfying $r_i \leq f(i,j) \leq c_i$, such that BALANCE ran J_j at time $f(i,j)$
(c) The *borrow graph* is a directed graph $G_t = (V, E)$ where the vertex set V is the collection $\mathcal{J}$ of jobs, and there is a directed edge $(J_i, J_j) \in E$ if and only if $J_i \hookleftarrow J_j$.
(d) Let R_i be the collection of jobs reachable from J_i in G_t by a path with at least one edge, and let $R_i^+ = R_i \cup \{J_i\}$.

We first establish some basic properties of BALANCE with respect to G_t.

Observation 6 If $J_a \hookleftarrow J_b$ then both J_a and J_b are active at time $f(a,b)$.

Lemma 7 Suppose that BALANCE was idle at some time $s < t$, that J_a is a job that is active at some time after s, and that J_b is a job that is active at some time before s. Then there is no directed path from J_a to J_b in the borrow graph G_t.

Proof: Since BALANCE is idle at time s, any job that is active before time s must be completed by time s. Similarly, any job active after time s must be released after time s. Suppose to reach a contradiction that there is a directed path from J_a to J_b. Then along this path there are two jobs J_x and J_y with a directed edge from J_x to J_y such that J_x was active after time s, while J_y is active before time s. Both J_x and J_y must be active at time $f(x,y)$ by the definition of $f(x,y)$, which is a contradiction. ■

Lemma 8 Let J_i be a job not completed by BALANCE at time t, and let P be a path from J_i to J_a in G_t. For any $s \in [r_a, t]$, at least one job in the path P is active at time s.

Proof: First we label each directed edge (J_x, J_y) with the interval $[f(x,y), c_x]$. Notice that the job J_x is active throughout the time interval $[f(x,y), c_x]$. The proof is by induction on the length of the path P. The base case of $a = i$ follows since J_i is active during the interval $[r_i, t]$. by BALANCE at time t. Now let J_b be the job immediately before J_a on the path P as it is traversed from J_i to J_a. By induction, for any $s \in [r_b, t]$ at least one job in the path from J_i to J_b along P is active at time s. Now observe that J_a is active during $[r_a, f(b,a)]$. Observe that $r_b \leq f(b,a)$. Therefore, for any $s \in [r_a, t]$, at least one job in the path P is active at time s. ■

We give some intuition before preceding. Let J_i and J_j be jobs such that BALANCE is running J_j during the time interval $[a,b] \subseteq [r_i, c_i]$. Hence, $J_i \hookleftarrow J_j$ by the definition of immediately borrow. The adversary may then do $(b-a)$ units of work on J_i during time $[a,b]$. We think of this as the adversary borrowing $(b-a)$ units of work from J_j to give to J_i. This borrowing might be advantageous to the adversary if $J_i \in U_B - U_A$ and $J_j \in U_A$, or if $J_i \in U_B - U_A$ and J_j can borrow from some job in U_A, etc. If the adversary is going to finish a job $J_i \in U_B - U_A$, then it must arrange for J_i to borrow $\epsilon w(i)$ units of work since the adversary's processor is ϵ slower than the processor used by BALANCE. This borrowing must come from jobs in $R_i \cap U_A$. The borrowing must come from jobs in R_i because these are the only jobs that can lend time to J_i, and the borrowing must come from jobs in U_A because the work done by the adversary on jobs in U_A is the only work done by the adversary that didn't contribute toward the adversary finishing a job by time t. Intuitively, by the description of the online algorithm, BALANCE ran jobs in R_i no longer than it ran J_i. Hence, we would expect that each job in U_A can only be used to finish $\frac{1}{\epsilon}$ jobs in $U_B - U_A$. We now make this intuition more formal.

Lemma 9 For all $J_i \in U_B$, and for all $J_j \in R_i^+$, $w(i) \geq w(j)$.

Proof: The proof is by induction on the length of a shortest path P (the length of a path is the number of edges in the path) between J_i and J_j in the borrow graph. Note that there is a path from J_i to J_j in the borrow graph by the definition of R_i. By renumbering the jobs, let the vertices in the directed path P be $J_i = J_1, \ldots, J_j$. The base case is when P contains no edges. In this case $j = i$ and the result is obvious. Now we assume that $j \geq 2$, and we inductively know that $w(i) \geq w(j-1)$. Thus it is sufficient to show that $w(j-1) \geq w(j)$.

Let ℓ_j be the last time that J_j was run by BALANCE before time t. We now wish to establish that $c_{j-1} \geq \ell_j$. Recall that we may assume that $j \geq 2$. If $j = 2$, then $j - 1 = 1 = i$ and $c_{j-1} \geq \ell_j$ follows immediately since $J_i \in U_B$. Now consider the case where $j > 2$. We will now establish that $\ell_j < f(j-2, j-1)$. So assume to reach a contradiction that $\ell_j \geq f(j-2, j-1)$. Applying observation 6 we have $f(j-2, j-1) \geq r_{j-2}$. Therefore, $\ell_j \in [r_{j-2}, t]$. Applying lemma 8, since $\ell_j \in [r_{j-2}, t]$ at least one job, say J_x, in the collection $J_1, J_2, \ldots, J_{j-2}$ is active at time ℓ_j. Then it must be the case that $J_x \hookleftarrow J_j$ by the definition of immediately borrow, which contradicts the fact that P is a shortest path.

Applying observation 6, we know that $f(j-2, j-1) \leq c_{j-1}$. Therefore we get $\ell_j \leq c_{j-1}$.

We are now ready to establish that $w(j-1) \geq w(j)$. Since $J_{j-1} \hookleftarrow J_j$, then by the definition of BALANCE, $w(j-1, y) \geq w(j, y)$ at some time y before c_{j-1}. Suppose for the moment that $c_{j-1} \geq t$. Then by observation 3, it must be the case that $w(j-1) \geq w(j)$. Otherwise, suppose that $c_{j-1} < t$. By observation 3, it must be the case that $w(j-1, c_{j-1}) \geq w(j, c_{j-1})$. Since $c_{j-1} \geq \ell_j$, it must be the case that $w(j) = w(j, c_{j-1})$. Obviously, $w(j-1) = x_j = w(j-1, c_{j-1})$. Hence, again we reach the conclusion that it must be that case that $w(j-1) \geq w(j)$. ∎

For the rest of this subsection, we renumber the jobs such that $U_B - U_A = \{J_i : 1 \leq i \leq p\}$, for $1 \leq i < j \leq p$ we have that $r_i \geq r_j$, and $U_A = \{J_i : p+1 \leq i \leq q\}$. Let $W(\mathcal{K}, I)$ denote the total work done by BALANCE on the jobs in $\mathcal{K}$ during the time interval I.

Observation 10 For all i, $1 \leq i \leq p$, $w_i > 0$.

Lemma 11 For each $1 \le i \le p$,

$$W(R_i^+ \cap U_A, [0,t]) \ge \epsilon \sum_{j=1}^{i} w(j)$$

Proof: We fix an arbitrary i, $1 \le i \le p$. By our indexing scheme, the jobs $J_1, \ldots, J_i$ must be released on or after r_i. Let $t_i = r_{h(i)}$ be the earliest release time of a job $J_{h(i)}$ in R_i^+.

Observe that applying lemma 7, we know that BALANCE did not idle at any time $s \in [t_i, t]$. During $[r_i, t]$ the adversary needs to do at least $\sum_{j=1}^{i} w(j)$ units of work on $J_1, J_2, \ldots, J_i$ in order to complete these jobs by time t. Therefore, $t - r_i \ge \sum_{j=1}^{i} w(j)$ since the adversary has a unit speed processor.

We now claim that any job run by BALANCE during $[t_i, t]$ is in R_i^+. Suppose J_k be some job that BALANCE ran during time $s \in [t_i, t]$. Consider any path P from J_i to $J_{h(i)}$ in G_t. Applying lemma 8, at least one job, say J_a, in the path P is active at s. Therefore, $J_a \hookleftarrow J_k$ by the definition of immediately borrow. Since J_a is in the path P, we have $J_a \in R_i^+$. As a result, $J_k \in R_i^+$ by the transitivity of the borrow relation.

Since BALANCE does not idle during the interval $[t_i, t]$, and any job that BALANCE runs during $[t_i, t]$ is in R_i^+, we get that $W(R_i^+, [t_i, t]) = (1+\epsilon)(t - t_i)$. Using the fact that $r_i \ge t_i$, we get that $W(R_i^+, [t_i, t]) \ge (t - t_i) + \epsilon(t - r_i)$. Using the fact that $t - r_i \ge \sum_{j=1}^{i} w(j)$, we get that $W(R_i^+, [t_i, t]) \ge (t - t_i) + \epsilon \sum_{j=1}^{i} w(j)$.

Now $W(R_i^+ - U_A, [t_i, t]) \le t - t_i$, since the adversary must complete jobs not in U_A by time t, and the adversary may run jobs in R_i^+ only during the interval $[t_i, t]$. Therefore, $W(R_i^+ \cap U_A, [t_i, t]) \ge \epsilon \sum_{j=1}^{i} w(j)$. The desired result follows since $[t_i, t]$ is a subinterval of $[0, t]$. ■

Lemma 12 The work in $W(U_A, [0,t])$ can be partitioned into p parts, $P_1, \ldots, P_p$ (with perhaps some work left over), such that P_i, $1 \le i \le p$, contains $\epsilon w(i)$ units of work done on jobs in $R_i^+ \cap U_A$.

Proof: Let P_i, $1 \le i \le p$, be the an arbitrary $\epsilon w(i)$ units of work from $W(R_i^+ \cap A, [0,t])$ that was not assigned to $P_1, P_2, \ldots, P_{i-1}$. By lemma 11 one can see that such an assignment is possible, and that this partition satisfies the requirements. ■

Lemma 13 $\epsilon |U_B - U_A| \le |U_A|$.

Proof: For $1 \le i \le p$, and for $J_j \in R_i^+ \cap U_A$, let $g(i,j)$ be the fraction of work in P_i that is work that BALANCE did on J_j. If $i > p$, or $J_j \notin R_i^+ \cap U_A$ then $g(i,j) = 0$.

For all $J_i \in U_B - U_A$ it is the case that

$$\sum_{j=p+1}^{q} g(i,j) = \epsilon w(i)$$

by lemma 12 and the definition of $g(i,j)$. Hence,

$$\epsilon |U_B - U_A| = \epsilon p = \sum_{i=1}^{p} \frac{1}{w(i)} \epsilon w(i) = \sum_{i=1}^{p} \frac{1}{w(i)} \sum_{j=p+1}^{q} g(i,j) = \sum_{i=1}^{p} \sum_{j=p+1}^{q} \frac{g(i,j)}{w(i)}$$

We now focus on upper bounding this last term. By lemma 9, for all J_i, $1 \leq i \leq p$, and for $J_j \in R_i^+ \cap U_A$, it is the case that $w(j) \leq w(i)$, and hence $\frac{g(i,j)}{w(i)} \leq \frac{g(i,j)}{w(j)}$; For notational convenience, we define $\frac{g(i,j)}{w(j)} = 0$ when $g(i,j) = 0$ and $w(j) = 0$. Also note that for all j, $p+1 \leq j \leq q$, $\sum_{i=1}^{p} \frac{g(i,j)}{w(j)} \leq 1$ by the definition of $g(i,j)$. Therefore.

$$\sum_{i=1}^{p} \sum_{j=p+1}^{q} \frac{g(i,j)}{w(i)} \leq \sum_{i=1}^{p} \sum_{j=p+1}^{q} \frac{g(i,j)}{w(j)} = \sum_{j=p+1}^{q} \sum_{i=1}^{p} \frac{g(i,j)}{w(j)} \leq \sum_{j=p+1}^{q} 1 = q - p = |U_A|$$

The result then follows. ∎

The following theorem completes the analysis of BALANCE.

Theorem 14 For the problem of minimizing the average response time, BALANCE is a $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-approximation algorithm.

Proof: Since $|U_B| - |U_A| \leq |U_B - U_A|$, applying lemma 13, we get $|U_B| \leq (1+\frac{1}{\epsilon})|U_A|$. Now applying lemma 4 we get the desired result. ∎

2.2 Two Processors

In this subsection we assume that the nonclairvoyant online scheduler is equipped with a unit speed processor and an ϵ speed processor. We assume that $\frac{1}{\epsilon}$ is an integer. We present an algorithm BALANCE2, and show that under these circumstances the average response time for BALANCE2 is at most $1+\frac{1}{\epsilon}$ times the average response for an adversary with a single unit speed processor.

Algorithm BALANCE2 : At time t, let J_a be the job that BALANCE2 ran the least and did not complete. Also among the remaining jobs that BALANCE2 did not complete, let J_b be the job that BALANCE2 ran the least.

BALANCE2 runs J_a on the unit speed processor and runs J_b on ϵ speed processor.

In order to compare the cardinalities of U_A and $U_B - U_A$ we will have to analyze a following combinatorial problem, which we call the Rebate Problem.

2.2.1 The Politician Problem

As before, the analysis of BALANCE2 depends on analyzing the cardinalities of the jobs that are unfinished by BALANCE2 and of the jobs unfinished by the adversary at an arbitrary time. In order to accomplish this, we will have to analyze the following combinatorial problem.

The Politician Problem: There are p politicians s donors. Each donor j, $1 \leq j \leq s$, wishes to contribute at most C_j dollars, and $C_1 \leq C_2 \leq \ldots \leq C_s$. Associated with politician i, $1 \leq i \leq p$, is a non-negative value S_i, and $0 < S_1 \leq S_2 \leq \ldots \leq S_p$. For $2 \leq i \leq p$, ith politician must raise at least ϵS_{i-1} dollars from the donors. According to the *election rule*, donor j can contribute to politician i only if $C_j \leq S_i$.

A solution to this instance of the Politician Problem is an assignment $g(i, j)$, the contribution of donor j to politician i, that satisfies the following conditions:

1. For all i, $1 \leq i \leq p$, and for all j, $1 \leq i \leq s$, $g(i, j) \geq 0$.
2. For all i, $2 \leq i \leq p$, $\sum_{j=1}^{s} g(i, j) = \epsilon S_{i-1}$ dollars. That is, the demand of each of the last $p-1$ politicians are met.
3. For all j, $1 \leq j \leq s$, $g(1, j) = 0$. That is, the first politician does not get any contribution.
4. For all j, $1 \leq j \leq s$, $\sum_{i=1}^{p} g(i, j) \leq C_i$ dollars. Each donor does not contribute beyond his/her capacity.
5. For all i, $1 \leq i \leq p$, and for all j, $1 \leq i \leq s$, $g(i, j) \neq 0$ only if $C_j \leq S_i$. The *election rule* is satisfied.

We say that a *swap* is two tuples (i_1, j_2) and (i_2, j_1) such that $i_1 < i_2$, $j_1 < j_2$, $g(i_1, j_2) > 0$, and $g(i_2, j_1) > 0$.

Lemma 15 If there is a solution for an instance of the Politician Problem then there is a solution with no swaps.

Proof: In a solution to an instance of the Politician Problem that contains swaps, we call a politician i and a donor j a witness pair if i is the largest integer such that there is a swap that includes politician i, and j is the largest integer such that there is a swap that includes donor j. Notice that there must exist an $i' < i$ and an $j' < j$ such that politician i' gets non-zero contribution from donor j, and politician i gets non-zero contribution from donor j' by the definition of i and j. Further notice that the two tuples (i, j') and (i', j) is a swap. Thus every witness pair is involved in a common swap.

We now show how to remove swaps. Consider an arbitrary solution g. Let i and j be the witness pair for g. Suppose g' is a solution such that for all $1 \leq i' \leq p$ and $1 \leq j' \leq s$, if $i' > i$ or $j' > j$ then $g'(i', j') = g(i', j')$. We then say that the solution g' is compatible with g. Among all solutions compatible to g, we choose a g' such that $g'(i, j)$ is maximal. We now claim that the witness pair for g' is the pair a $(\leq i)$ and b $(\leq j)$ such that $a + b < i + j$. Therefore, repeating the process mentioned in this paragraph at most $p + s$ times we will get a solution without any swap.

We now complete the proof by showing that the witness pair (a, b) for g' has the property $a \leq i$, $b \leq i$ and $a + b < i + j$. Since g' is compatible with g and the witness pair for g is (i, j), we have $a \leq i$ and $b \leq j$. Our proof will be complete once we show that politician i and donor j are not involved in a common swap in g'

Suppose, to the contrary, let (i, j') and (i', j) be a swap in g'. Notice that $i' < i$ and $j' < j$ since g' is compatible to g and (i, j) is the witness pair for g. Then let g'' be the solution identical to g' except that $g''(i, j) = g'(i, j) + \min(g'(i, j'), g'(i', j))$, $g''(i', j') = g'(i', j') + \min(g'(i, j'), g'(i', j))$, $g''(i, j') = g'(i, j') - \min(g'(i, j'), g'(i', j))$, and $g''(i', j) = g'(i', j) - \min(g'(i, j'), g'(i', j))$. Observe now that g'' is also compatible to g and $g''(i, j) > g'(i, j)$, a contradiction to the construction of g'. ■

Lemma 16 **If there is a solution g for an instance of the Politician Problem where $\epsilon = 1$, then $s \geq p - 1$.**

Proof: If $p = 1$ the result is trivially true, so from here on assume that $p > 1$. By lemma 15 we may assume, without loss of generality, that g contains no swaps. We prove by induction on $p - i$ that for any i, $0 \leq i \leq p - 2$, donors $s, s - 1, \ldots, s - i$ do not contribute to a politician $k < p - i$. Note that the inductive hypothesis also guarantees the existence of donors donors $s, s - 1, \ldots, s - i$, that is $s - i \geq 1$. It immediately follows that $s \geq p - 1$.

For the base case consider $i = 0$. If $s = 0$ then obviously there is no solution. So assume that $s \geq 1$. Consider donor s and assume to reach a contradiction that a politician k, $k < p$, gets non-zero contribution from s. Note that it must be the case that $C_s \leq S_{p-1}$, otherwise customer s could then not contribute to politician k by the election rule. Politician p cannot get contribution from a donor j, $j < s$, or (k, s) and (p, j) would be a swap. Thus politician p must get all of his/her contribution of S_{p-1} dollars from donor s. Since $S_{p-1} \geq C_s$, the only way that this is possible is if $S_{p-1} = C_s$, but this then contradicts the assumption that donor s contributes to politician k.

Now assuming that the inductive hypothesis holds for $i < p - 2$, we show that it holds for $i + 1$. By the induction hypothesis, donors $s, s - 1, \ldots, s - i$ cannot contribute to politician $p - i - 1$. Thus in order for a solution to exist there must exist a contributor $s - i - 1$. Assume to reach a contradiction that donor $s - i - 1$ contributes to a politician k, $k < p - i - 1$. Then, according to the election rule, it must be the case that $C_{s-i-1} \leq S_{p-i-2}$. Politician $p - i - 1$ cannot get contribution from donor j, $j < \alpha(i)$, since otherwise $(k, s - i - 1)$ and $(p - i - 1, j)$ would be a swap. By our induction hypothesis, politician $p - i - 1$ cannot get contribution from donors $s - i, s - i + 1, \ldots s$. Thus politician $p - i - 1$ must get all his/her contribution of S_{p-i-2} dollars from donor $p - i - 1$. Since $C_{s-i-1} \leq S_{p-i-2}$, the only way that this is possible is if $S_{p-i-2} = C_{s-i-1}$. But this then contradicts the assumption that donor $s - i - 1$ contributes to politician k. ■

Lemma 17 **If there is a solution g for the Politician Problem and $\frac{1}{\epsilon}$ is an integer then $s \geq \epsilon(p - 1)$.**

Proof: We create a new instance of the Politician Problem, with $\epsilon' = 1$, in the following manner. We use primes to denote the value of the variables in the new instance. Set $p' = p$. For each i, $1 \leq i \leq p'$, $S'_i = S_i$. Observe that politician i must raise at least $\epsilon' S_{i-1}$ dollars. In order to accommodate the demand, we also increase the number of donors to s/ϵ by replacing each donor j, $1 \leq j \leq s$ by $\delta = 1/\epsilon$ identical donors $j_1, \ldots, j_\delta$. Then by letting $g'(i, j_a) = g(i, j)$, for all $1 \leq i \leq p$, $1 \leq j \leq s$, and $1 \leq a \leq \delta$, we get that g' is a solution to the new instance of the Politician Problem. Hence, by lemma 16 we get $\frac{s}{\epsilon} \geq (p - 1)$. ■

Finally, we explain why we need the assumption that $\frac{1}{\epsilon}$ is an integer for our analysis.

Lemma 18 **If there is a solution g for the Politician Problem when $\frac{1}{\epsilon}$ is not an integer, then it need not be the case that $s \geq \epsilon(p - 1)$.**

Proof: Consider the instance $\epsilon = \frac{2}{3}$, $p = 3$, $S_3 = S_2 = 30$, $S_1 = 15$, $s = 1$, and $C_1 = 30$. Consider the solution, $g(3, 1) = 20$, $g(2, 1) = 10$, and $g(1, 1) = 0$. But note that $\frac{s}{\epsilon} < p - 1$ in this case. ■

2.2.2 Analysis

As before, we on occasion use B as shorthand for BALANCE2. We modify the definition of immediately borrow in the following way. A job J_i can *immediately borrow* from another job J_j if at some time t, satisfying $r_i \le t \le c_i(B)$, it is either the case that:

- BALANCE2 was running J_j on one of the processors at time t, and was not running J_i on either processor at time t, or
- BALANCE2 was running J_j on the unit speed processor and was running J_i on the ϵ speed processor at time t.

We define U_A, U_B, the borrow graph G_t, $w(i,t)$, and R_i as before. As in the analysis of BALANCE, our goal is to establish that $|U_B| - 1 \le (1 + \frac{1}{\epsilon})|U_A|$ at any point t in time. So, we now fix a time t as before. We will now establish some combinatorial facts that will provide a solution to a particular instance of the Politician Problem where the jobs in $U_B - U_A$ are the politicians and the jobs in U_A are the donors.

Observation 19 If $w(i,u) \le w(j,u)$ and $u \ge r_j$, then for all $s \in [u, c_j]$ it is the case that $w(i,s) \le w(j,s)$.

Lemma 20 For all $J_i \in U_B$, and for all $J_j \in R_i^+$, $w(i) \ge w(j)$.

Proof: We extend the borrow graph G_t to a graph H_t by adding additional edges in the following way. For each pair of jobs J_a and J_b such that there is no edge (J_b, J_a) in G_t, we add an edge (J_b, J_a) if and only if there is a time t such that both J_a and J_b are *active* at time t and $w(a,t) \le w(b,t)$. For any such new edge (J_b, J_a), let the time certificate, $f(b,a)$, be the earliest time such that both J_a and J_b are active at time $f(b,a)$, and and $w(a, f(b,a)) \le w(b, f(b,a))$. For each edge (J_b, J_a) in H_t, we label the edge with with the interval $[f(b,a), c_b]$. By observation 19, $w(a,u) \le w(b,u)$ for all $u \in [f(b,a), c_b]$.

Let P be the path from J_i to a job J_j in H_t with the fewest number of edges. By renumbering the jobs, let the vertices in the directed path P be $J_i = J_1, J_2, \ldots, J_j$. Also for $2 \le a \le j$, let $[t_a, c_{a-1}]$ be the interval label associated with the directed edge (J_{a-1}, J_a) along the path P in H_t (see figure 1). Note that c_a is the completion time of job J_a in BALANCE2's schedule. Note that for all a, $2 \le a \le j-1$, we have $t_a \le c_a$ by the definition of t_a.

We now argue that there can not exist an integer a, $2 \le a \le j-1$, such that $t_a < t_{a+1}$. Assume to reach a contradiction that such an a exists. We break the proof into two cases. In the first case suppose that $t_{a+1} \le c_{a-1}$ (see figure 2(A)). Note that $w(a+1, t_{a+1}) \le w(a, t_{a+1})$ by the definition of t_{a+1}. Note that $w(a, t_{a+1}) \le w(a-1, t_{a+1})$ since $t_{a+1} \in [t_a, c_{a-1}]$ by assumption. Therefore, $w(a+1, t_{a+1}) \le w(a-1, t_{a+1})$. Since J_{a-1} and J_{a+1} are active at time t_{a+1}, there must be an edge from J_{a-1} to J_{a+1} in H_t by the definition of H_t. This contradictions that assumption that P is the path from J_i to J_j with the fewest number of edges in it.

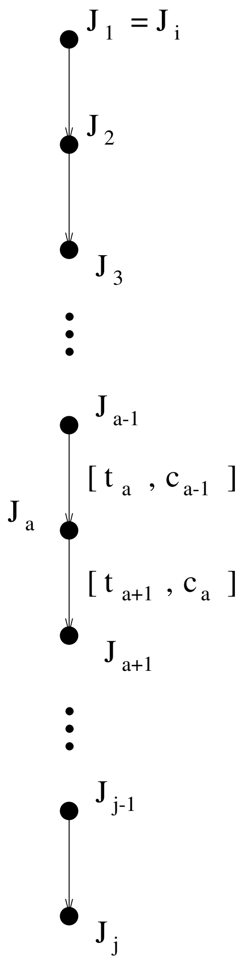

Figure 1. Path from J_1 to J_j

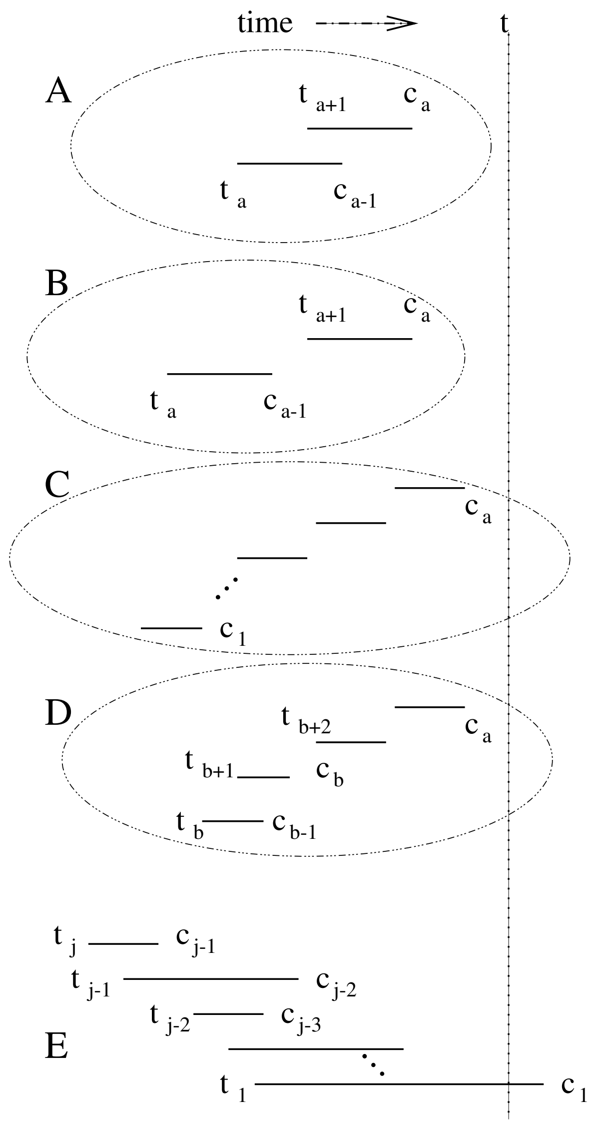

Figure 2. Various Cases.

In the second case suppose that $c_{a-1} < t_{a+1}$ (see figure 2(B)). Note that $a \neq 2$ since $c_1 > t$. We claim that in this case there must exist an integer b, $2 \leq b \leq a-1$, such that $c_b < t_{b+2}$ and $c_{b-1} \geq t_{b+1}$ (see figure 2(D)). Otherwise, if this was not the case we get that

$$t_{a+1} \geq c_{a-1} \geq t_a \geq c_{a-2} \geq \ldots \geq t_3 \geq c_1$$

which is a contradiction since obviously $t_{a+1} \leq t$ and $t < c_1$ (see figure 2(C)). Consider the time $s = \max(t_{b+1}, t_b)$. First note that $s \in [t_{b+1}, c_b]$; If $s = t_{b+1}$ then it is obvious, otherwise it follows because we have already established that $t_{b+1} \leq t_b \leq c_b$. Therefore, $w(b+1, s) \leq w(b, s)$ by observation 19. Then note that $s \in [t_b, c_{b-1}]$; If $s = t_b$ then it is obvious, otherwise it follows because we have already established that $t_b \leq t_{b+1} \leq c_{b-1}$. Therefore, $w(b, s) \leq w(b-1, s)$ by observation 19. Therefore by transitivity $w(b+1,s) \le w(b-1,s)$. Since both J_{b-1} and J_{b+1} are active at time s, there must be a directed edge from J_{b-1} to J_{b+1} in H_t by the definition of H_t. This contradicts the assumption that P is the path from J_i to J_j in H_t with the fewest number of edges.

Hence, it must be the case that $t_a \ge t_{a+1}$ for all a, $2 \le a \le j-1$ (see figure 2(E)). We argue that for all a, $2 \le a \le j-1$, and for all $s \in [t_{a+1}, c_a]$, it is the case that $w(j,s) \le w(a,s)$. The proof is by induction on $j-a$. The base case $j-a=1$, or equivalently $a=j-1$, holds by observation 19. Now, for the inductive step, suppose that $w(j,s) \le w(a,s)$ for all $s \in [t_{a+1}, c_a]$ for $a \ge 2$. Since $t_a \in [t_{a+1}, c_a]$, applying induction hypothesis we get $w(j,t_a) \le w(a,t_a)$. Note that $w(a,t_a) \le w(a-1,t_a)$ by the definition of t_a. Hence, $w(j,t_a) \le w(a-1,t_a)$. Applying observation 19, we get $w(j,s) \le w(a-1,s)$ for $s \in [t_a, c_{a-1}]$, thus establishing induction hypothesis for $j-a+1$. As a consequence, by setting $a=2$, we get. $w(j,s) \le w(1,s)$ for $s \in [t_2, c_1]$. Since $t \in [t_2, c_1]$, we get $w(j) \le w(1) = w(i)$, the desired result. ■

We again renumber the jobs such that $U_B - U_A = \{J_i : 1 \le i \le p\}$, for $1 \le i < j \le p$ we have that $r_i \ge r_j$, and $U_A = \{J_i : p+1 \le i \le q\}$. The following observation is obvious from the description of BALANCE2, and the definition of U_B.

Observation 21 For $1 \le i \le p-1$, $0 < w(i) \le w(i+1)$.

Lemma 22 Suppose that BALANCE2 was idle at some time $s < t$, that J_a is a job that is active at some time after s, and that J_b is a job that is active at some time before s. Then there is no directed path from J_a to J_b in the borrow graph G_t.

Proof: The proof is identical to the proof of lemma 7. ■

Lemma 23 Let J_i be a job not completed by BALANCE2 at time t, and let P be a path from J_i to J_a in G_t. For any $s \in [r_a, t]$, at least one job in the path P is active at time s.

Proof: The proof is identical to the proof of lemma 8. ■

Let $W(\mathcal{K}, I)$ denote the total work done by BALANCE on the jobs in $\mathcal{K}$ during the time interval I.

Lemma 24 For each $1 \le i \le p-1$,

$$W(R^+_{i+1} \cap U_A, [0,t]) \ge \epsilon \sum_{j=1}^{i} w(j)$$

Proof: We fix an arbitrary i, $1 \le i < p$. By our indexing scheme, the jobs $J_1, \ldots, J_i$ must be released on or after r_i. Let $t_{i+1} = r_{h(i+1)}$ be the earliest release time of a job $J_{h(i+1)}$ in R^+_{i+1}.

During $[r_i, t]$ the adversary must complete at least $\sum_{j=1}^{i} w(j)$ units of work on jobs $J_1, J_2, \ldots, J_i$ in order to complete these jobs by time t. Therefore, $t - r_i \ge \sum_{j=1}^{i} w(j)$ since the adversary uses only a unit speed processor.

We claim that any job J_k run by BALANCE2 on its unit speed processor at some time $s \in [t_{i+1}, t]$ is in R^+_{i+1}. To see this note that by lemma 23, that there is at least one job $J_a \in R^+_{i+1}$ active at time s. Then $J_a \hookleftarrow J_k$ by the definition of immediately borrow. And finally $J_k \in R^+_{i+1}$ by transitivity.

We claim that BALANCE2 did not idle its unit speed processor at any time $s \in [t_{i+1}, t]$. To reach a contradiction, suppose BALANCE2 idled its unit speed processor at some time $s \in [t_{i+1}, t]$. By the description of BALANCE2, the ϵ speed processor is also idle. Note that $J_{h(i+1)}$ is released, and is thus active, before time s. Thus by lemma 22 there is no path from J_i to $J_{h(i+1)}$, a contradiction.

We now claim that any job J_k run by BALANCE on its ϵ speed processor at some time $s \in [r_i, t]$ is in R^+_{i+1}. To see this note that J_i and J_{i+1} were released at or before time r_i, and J_i and J_{i+1} are in $U_B - U_A$. Hence J_i and J_{i+1} were active at time s. At time s, only one of J_i or J_{i+1} can be run by BALANCE2 on its unit speed processor. We thus consider two cases (note that these cases are not strictly mutually exclusive). In the first case assume that J_{i+1} is not being run by BALANCE on the unit speed processor at time s. Then $J_{i+1} \hookleftarrow J_k$ by the definition of immediately borrow, and $J_k \in R_{i+1}$ by the definition of R_{i+1}. In the second case assume that J_i is not being run by BALANCE on the unit speed processor at time s. Then $J_i \hookleftarrow J_k$ by the definition of immediately borrow. Note that $J_{i+1} \hookleftarrow J_i$ since $r_i \geq r_{i+1}$ and $J_{i+1} \in U_B - U_A$. Hence $J_k \in R_{i+1}$ by transitivity.

Putting together the claims in the preceding four paragraphs we get that $W(R^+_{i+1}, [t_{i+1}, t]) \geq (t - t_i) + \epsilon(t - r_i)$. Using the fact that $t - r_i \geq \sum_{j=1}^{i} w(j)$, we get that $W(R^+_{i+1}, [t_{i+1}, t]) \geq (t - t_{i+1}) + \epsilon \sum_{j=1}^{i} w(j)$.

Now $W(R^+_{i+1} - U_A, [t_{i+1}, t]) \leq t - t_{i+1}$, since the adversary must complete these jobs by time t, and the adversary may run jobs in R^+_{i+1} only during the interval $[t_{i+1}, t]$. Therefore, $W(R^+_{i+1} \cap U_A, [t_{i+1}, t]) \geq \epsilon \sum_{j=1}^{i} w(j)$. The desired result follows since $[t_{i+1}, t]$ is a subinterval of $[0, t]$. ∎

The proof lemma 25 is identical to the proof lemma 12.

Lemma 25 The work in $W(U_A, [0, t])$ can be partitioned into $p - 1$ parts, $P_1, \ldots, P_{p-1}$ (with perhaps some work left over), such that P_i, $1 \leq i \leq p - 1$ contains $\epsilon w(i)$ units of work done on jobs in $R^+_{i+1} \cap U_A$.

We are now ready to apply the Politician Problem.

Lemma 26 If $\frac{1}{\epsilon}$ is an integer, then $(1 + \frac{1}{\epsilon})|U_A| \geq |U_B| - 1$.

Proof: We first construct an instance of the Politician Problem. We let the politicians be the jobs in $U_B - U_A$, and the donors be the jobs in U_A. Politician i, $1 \leq i \leq p$, corresponds to job J_i, and $S_i = w(i)$, $1 \leq i \leq p$. Thus politician i, $2 \leq 1 \leq p$, wishes to raise $\epsilon S_{i-1} = \epsilon w(i-1)$ dollars. Note that $0 < S_i \leq S_{i+1}$ by observation 21.

For $2 \leq i \leq p$, and for $J_j \in R^+_i \cap U_A$, let $g(i, j)$ be the fraction of work in P_{i-1} that is work that BALANCE2 did on J_j. If $i = 1$, $i > p$, or $J_j \notin R^+_i \cap U_A$ then $g(i, j) = 0$. Let $s = (p+q) - (p+1) + 1 = p - q$. For a $j \in [p+1, q]$, let demand $C_j = \sum_{i=1}^{p} g(i, j)$. We then renumber the donors so that $C_j \leq C_{j+1}$ for $1 \leq j \leq s - 1$.

Lemma 25 and lemma 17 assert that g is a solution to this instance of the Politician Problem; Lemma 20 asserts that the election rule is satisfied. Thus by lemma 17, we get that $\frac{|U_A|}{\epsilon} \geq |U_B - U_A| - 1$. The claim then follows by noting that $|U_B - U_A| \geq |U_B| - |U_A|$. ∎

Lemma 27 Let T be a maximal length interval during which BALANCE2 never idles its unit speed processor. Let $A(T)$ be the total response time for the adversary for jobs released during T. Let $B(T)$ be the total response time for BALANCE2 for jobs released during T. Then $\frac{1+\epsilon}{\epsilon}A(T) \leq B(T)$.

Proof: Note first that it is never profitable for the adversary to unnecessarily idle its processor. Hence, the adversary must be running a job whenever BALANCE2 is running a job. Let L be the total work, at the end of T, that the adversary has left to do on the jobs released during T. We grant the adversary the additional power to finish these jobs at the end of T, while incurring only an additional charge of L to the adversary's total response time. This strictly benefits the adversary. Let S be the times during T when BALANCE2 was idling its ϵ speed processor, and let D be the times during T when BALANCE2 was running both of its processors. We $|X|$ to denote the measure of a collection X of intervals.

Observe that the amount of work done by BALANCE2 during the entire interval T is $|T|+\epsilon|D|$. Since the maximum work done by the adversary during T is at most $|T|$, we get that $L \geq \epsilon|D|$. Hence, by lemma 4,

$$A(T) \geq |S| + L + \int_{t\in D} U_A(t)dt \geq |S| + \epsilon|D| + \int_{t\in D} U_A(t)dt$$

Hence,

$$\frac{1+\epsilon}{\epsilon}A(T) \geq \frac{1+\epsilon}{\epsilon}|S| + (1+\epsilon)|D| + \frac{1+\epsilon}{\epsilon}\int_{t\in D} U_A(t)dt$$

By lemma 26,

$$|D| + \frac{1+\epsilon}{\epsilon}\int_{t\in D} U_A(t)dt \geq \int_{t\in D} U_B(t)dt$$

Hence,

$$\frac{1+\epsilon}{\epsilon}A(T) \geq |S| + \epsilon|D| + \int_{t\in D} U_B(t)dt$$

Then we obtain that $\frac{1+\epsilon}{\epsilon}A(T) \geq B(T)$ by noting that $B(T) = S + \int_{t\in D} U_B(t)dt$ by lemma 4. ∎

Theorem 28 Let ϵ be some constant such that $\frac{1}{\epsilon}$ is an integer. Then the average response time of BALANCE2, with a unit speed processor and an ϵ speed processor, is at most $1+\frac{1}{\epsilon}$ times the average response time of an adversary with a unit speed processor.

Proof: The proof follows immediately from lemma 27. ∎

It is not hard to see that BALANCE2 produces an $O(1)$ approximate schedule even if $\frac{1}{\epsilon}$ is not an integer.

2.3 Miscellaneous Results

Lemma 29 For every nonclairvoyant scheduling algorithm S there is an input I and a time t such that $|U_S(t,1)|/|U_A(t,1)| = n/2$.

Proof: Let $t_0 = 0$ and $t_i = t_{i-1} + 1/2^{i-1}$, $1 \le i \le n-1$. Two jobs are released at time t_0. One job is released at time t_i, $1 \le t \le n-1$. The adversary sets the jobs lengths so that S hasn't finished any jobs by time t_{n-1}. In addition, the sum of the length of all jobs does not exceed 2 so that the adversary can complete the jobs on or before 2. Inductively, 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, $i \ge 0$, the sum σ_i of the remaining unfinished lengths of the two jobs, say $J_{a(i)}$ and $J_{b(i)}$, that are active for the adversary satisfies $1/2^i \le \sigma_i \le 1/2^{i-1}$. Otherwise, the total length of the jobs will exceed 2. Therefore, the remaining length of the shorter of the two jobs $J_{a(i)}$ and $J_{b(i)}$ is at most $1/2^i$. Therefore, on or before time t_{i+1}, the adversary can finish the shorter job, thus leaving only one job unfinished. ■

We now show that the algorithm Round Robin [12, 26] does not have a bounded approximation ratio if it is given a slightly faster processor than the adversary. Round Robin splits the processing time evenly among all unfinished jobs. This construction is a modification of the Round Robin lower bound constructions in [25, 27].

Lemma 30 For the problem of minimizing the average response time, for each ϵ satisfying $0 < \epsilon < 1$, Round Robin is a $(1+\epsilon)$-speed $\Omega(n^{1-\epsilon})$-approximation algorithm.

Proof: We divide time into stages. Let the ith stage, $i \ge 0$ 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, $i \ge 1$, with length $s(i)$ that is exactly the same length as Round Robin has left on each of the previous jobs. In order to guarantee that the adversary can finish the job released at time t_{i-1} by time t_i, $i \ge 2$, we let $t_i = t_{i-1} + s(i-1)$. Observe that during the interval $[t_{i-1}, t_i]$, Round Robin executes each of the $i+1$ jobs for $\frac{s(i-1)}{i+1}$ units of time. Since Round Robin also uses a $1+\epsilon$ speed processor, the effective work done on a job during that interval is $\frac{(1+\epsilon)s(i-1)}{i+1}$. Therefore, we get the recurrence

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

Expanding this we get

$$s(i) = \left(\frac{\prod_{j=0}^{i-2}(i-j-\epsilon)}{\prod_{j=0}^{i-2}(i+1-j)}\right) s(1) = \Theta\left(\frac{\Gamma(i+1-\epsilon)}{\Gamma(i+2)}\right)$$

where $\Gamma(x)$ is Legendre's gamma function [19, page 48]. It is known that $\Gamma(x+1) = \Theta(\sqrt{x}\left(\frac{x}{e}\right)^x)$ [19, exercise 6, section 1.1.11.2]. Hence,

$$\begin{aligned} s(i) &= \Theta\left(\frac{\sqrt{i-\epsilon}\left(\frac{i-\epsilon}{e}\right)^{i-\epsilon}}{\sqrt{i+1}\left(\frac{i+1}{e}\right)^{i+1}}\right) \\ &= \Theta\left(\frac{(i-\epsilon)^{i-\epsilon}}{(i+1)^{i+1}}\right) \end{aligned}$$

$$
\begin{aligned}
&= \Theta\left(\left(\frac{1}{(i-\epsilon)^{1+\epsilon}}\right)\left(\frac{i-\epsilon}{i+1}\right)^{i+1}\right) \\
&= \Theta\left(\left(\frac{1}{(i-\epsilon)^{1+\epsilon}}\right)\left(1-\frac{1+\epsilon}{i+1}\right)^{i+1}\right) \\
&= \Theta\left(\frac{1}{(i-\epsilon)^{1+\epsilon}}\right)
\end{aligned}
$$

The total response time for the adversary is then $\Theta(\sum_{i=1}^{n} \frac{1}{(i-\epsilon)^{1+\epsilon}})$, which is a convergent sum. The total response time for Round Robin is then $\Theta(\sum_{i=1}^{n} \frac{i-\epsilon}{i^{1+\epsilon}})$, which is $\Theta(n^{1-\epsilon})$. The result then follows. ■

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

Theorem 31 For the problem of minimizing the maximum idle time, BALANCE is a $(1+\epsilon)$-speed $(\frac{1}{\epsilon})$-approximation algorithm.

Proof: Let t be the time at which BALANCE completed a job J_i for which the idle time is maximum. Without loss of generality, assume that BALANCE did not idle during the interval $[0, t]$. If BALANCE last idled at time s, then we may consider only jobs released during $[s, t]$. This will not change the maximum idle time experienced by BALANCE, and can not increase the maximum idle time experienced by the adversary. By shortening other unfinished jobs, let us assume, without loss of generality, that BALANCE completed all jobs by time t. This does not affect the maximum idle time for BALANCE, and can not increase the maximum idle time for the adversary.

Let $d = t - \frac{x_i}{1+\epsilon} - r_i$ be the idle time experienced by J_i when using BALANCE. By lemma 9, the amount of work done by BALANCE on any job during (r_i, t) is at most x_i. Since the adversary had a $\frac{1}{1+\epsilon}$ slower processor than the one used by BALANCE, the adversary must have finished a job J_j at time $(1+\epsilon)t$.

We now argue that $x_j \leq (1+\epsilon)(t-d)$. We consider two cases. First assume that BALANCE did not run J_j at all during the interval $[r_i, t]$. Then x_j was completed by time r_i, and hence $x_j \leq r_i \leq t-d$. Now assume that BALANCE ran J_j at some time during $[r_i, t]$. Then $x_j \leq x_i$ by the definition of BALANCE and observation 3. Also $x_i \leq (1+\epsilon)(t-d)$ by the definition of d. Therefore, again we get that $x_j \leq (1+\epsilon)(t-d)$.

Notice $r_j \leq t - \frac{x_j}{1+\epsilon}$ since BALANCE was able to finish J_j by time t. Therefore, the idle time incurred by the adversary for J_j must be at least

$$c_j(A) - r_j = (1+\epsilon)t - r_j \leq (1+\epsilon)t - (t - \frac{x_j}{1+\epsilon}) = \epsilon(t - \frac{x_j}{1+\epsilon}) \geq \epsilon d$$

The last inequality follows since $d = t - \frac{x_i}{1+\epsilon} - r_i$. The result follows because d is the maximum idle time experienced by BALANCE. ■

3 Real-time Scheduling

We now turn our attention to the online version of the problem $1|r_i, pmtn|\sum w_i(1-U_i)$. We present and analyze an algorithm SLACKER for this problem. We first need to introduce some notations and definitions.

We normalize our time units so that SLACKER is using a unit speed processor. The algorithm SLACKER is parameterized by a real number $\epsilon > 0$. Let $\delta = \epsilon/2$, and let $c > \frac{1+\delta}{\delta}$ (later we will set $c = \frac{1+\delta}{\delta} + \frac{1}{\sqrt{\delta}}$). A job J_i is *viable* at time t if SLACKER can finish J_i before time d_i, that is, if SLACKER ran J_i for at least $x_i - (d_i - t)$ time units before time t. A job J_i is *ϵ-fresh* at time t if $d_i - t > (1+\delta)x_i$. Otherwise, we say that J_i is *ϵ-stale* at time t. A job J_i is *ϵ-good* if $d_i - r_i \geq (1+\epsilon)x_i$, that is, if the laxity of J_i is at least ϵx_i. Finally, recall that v_i is the value density of J_i.

Algorithm SLACKER(ϵ): The algorithm SLACKER maintains the invariant that it is always running the job at the top of a stack M. The stack M is initially empty. The algorithm SLACKER will respond to two types of events, the release of a job, and the completion of the job at the top of M. The events are treated as follows:

- Consider the release of a job J_i at time r_i. If M is empty then J_i is pushed onto M if J_i is ϵ-good. Else let J_j be the job at the top of M. If J_i is ϵ-good and $v_i \geq cv_j$, then J_i is pushed onto the top of M. Otherwise no action is taken.

- Consider the completion of a job J_j at the top of M. First, J_j and all non-viable jobs are removed from M. At this time, among the jobs that are ϵ-good, that are *ϵ-fresh*, and that are not in M, let J_i be the highest density job. If M is empty then J_i is pushed onto M. Else let J_k be the job currently at the top of M. Then J_i is pushed onto M if and only if $v_i \geq cv_k$.

Note that the completion of a job J_j at top of M obviously occurs before d_j, and that effectively SLACKER(ϵ) ignores the existence of jobs that are not ϵ-good. 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 A be the set of jobs finished by the adversary. For brevity, we occasionally use S to denote the online algorithm SLACKER(ϵ). Let C be the set of jobs completed by SLACKER(ϵ), and let $U = \mathcal{J} - C$ be the jobs not completed by SLACKER(ϵ). Finally let R be the set of jobs that were pushed into the stack M by SLACKER(ϵ) at some time during the run of the algorithm. The set C may be a proper subset of R since SLACKER(ϵ) may not finish every job that it starts.

3.1 Laxity Proportional to Length

In this section we assume that both adversary and SLACKER(ϵ) have a unit speed processor, and that every job is ϵ-good. We show that SLACKER(ϵ) is constant competitive under these circumstances. Before beginning our analysis of SLACKER(ϵ), we give some brief intuition. If a job J_j was on a non-top position on M for so long that SLACKER(ϵ) did not finish M, then for δx_j units of time SLACKER(ϵ) was running jobs that were much denser than J_j. We show that then these denser jobs can pay for J_j in an amortized sense. So then the only way that the adversary can beat SLACKER(ϵ) is to get a lot of benefit from jobs that SLACKER(ϵ) never pushed onto M. If SLACKER(ϵ) never pushed a job J_j onto M, then for at least δx_j units of time SLACKER(ϵ) was running a job with density at least $\frac{v_j}{c}$. This will be sufficient for the jobs that SLACKER(ϵ) finished to pay for J_j in an amortized sense. We now make this intuition more formal.

Lemma 32 For each $J_i \in R - C$, it is the case that SLACKER(ϵ) was running a job with density at least cv_i for a period of at least $d_i - r_i - (1+\delta)x_i \geq \delta x_i$ time units while J_i was in M.

Proof: During the time J_i was in the stack M, SLACKER(ϵ) ran either J_i or another job with density at least cv_i. The lemma follows since each job was ϵ-fresh when it was added to M. ■

Lemma 33 For $c > 1 + \frac{1}{\delta}$, $(1 - \frac{1}{\delta(c-1)}) \ \|R\| \leq \|C\|$, or equivalently, $\|R\| \leq \frac{\delta(c-1)}{\delta(c-1)-1} \ \|C\|$.

Proof: We give an amortization scheme to establish this result. Each job in $J_i \in R$ has an account A_i associated with it. The balance of each account A_i, for $J_i \in C$, is initially b_i credits, and the balance of each A_i, for $J_i \in R - C$, is initially 0 credits. We show how to transfer funds between accounts so that for each $J_i \in R$, $A_i \geq b_i(1 - \frac{1}{\delta(c-1)})$ after these transfers. The result then follows.

We now explain how these transfers take place. For each uninterrupted period $[a, b]$ that SLACKER(ϵ) is running a job J_i, $\frac{(b-a)v_j}{\delta}$ credits will be transfered from A_i to the account A_j of each job J_j, other than J_i, that is in M during $[a, b]$.

We now argue that the account A_i, of each job $J_i \in R$, receives as least b_i credits either initially, or in transfers from other jobs in R. If $J_i \in C$ then it received b_i credits initially. By lemma 32, each $J_i \in R - C$ is in M for at least δx_i times units while some other job is run. Hence, by the amortization scheme, J_i will receive at least $\delta x_i \frac{v_i}{\delta} = b_i$ credits.

Therefore, in order to show that, for each $J_i \in R$, it is the case that $A_i \geq b_i(1 - \frac{1}{\delta(c-1)})$ after the transfers, it is sufficient to show that the net transfer from A_i is at most $b_i \frac{1}{\delta(c-1)}$. If J_i is at the top of the stack M at time t, then the density of the job J_j, that is k ($k \geq 1$) jobs down from J_i in M at time t, is at most $\frac{v_i}{c^k}$ by the description of SLACKER(ϵ). Since SLACKER(ϵ) runs J_i for at most x_i units of time, the total credits transfered from A_j then is at most

$$x_i \sum_{j=1}^{\infty} \frac{v_i}{\delta c^j} = \frac{x_i v_i}{\delta} \frac{\frac{1}{c}}{1 - \frac{1}{c}} = \frac{x_i v_i}{\delta(c-1)} = \frac{b_i}{\delta(c-1)}$$

■

We say that a job J_i is *v-dense* if $v_i \geq v$. We define $T_B(v, \mathcal{K})$ to be the total time spent by the algorithm B running jobs in $\mathcal{K}$ that are v-dense. We now show that the total time that the adversary is running v-dense jobs is at most $\frac{1+3\delta}{\delta}$ times the time that SLACKER(ϵ) is running $\frac{v}{c}$-dense jobs.

Lemma 34 Let $\mathcal{J}$ be a collection of ϵ-good jobs. For all $v > 0$, $T_A(v, \mathcal{J}) \leq \frac{1+3\delta}{\delta} T_S(\frac{v}{c}, \mathcal{J})$.

Proof: First $T_A(v, \mathcal{J}) = T_A(v, U) + T_A(v, C)$ by the definition of C and U. Since SLACKER(ϵ) completed each job in C, $T_A(v, C) \leq T_S(v, C)$. Obviously, $T_S(v, C) \leq T_S(\frac{v}{c}, C) \leq T_S(\frac{v}{c}, \mathcal{J})$. Therefore, $T_A(v, \mathcal{J}) \leq T_A(v, U) + T_S(\frac{v}{c}, \mathcal{J})$. We now turn our attention toward showing that $T_A(v, U) \leq \frac{1+2\delta}{\delta} T_S(\frac{v}{c}, \mathcal{J})$, which will complete the proof.

Let $I_1 = [s_1, e_1], \ldots, I_m = [s_m, e_m]$ be the minimum cardinality collection of time intervals during which SLACKER(ϵ) is running only $\frac{v}{c}$-dense jobs. That is, SLACKER(ϵ) is running a $\frac{v}{c}$-dense job at time t if and only if t is in an I_a, $1 \leq a \leq m$. We fix the index such that $s_1 < s_2 < \ldots < s_m$. For convenience let $e_0 = 0$ and $s_{m+1} = +\infty$. For each $0 \leq k \leq m$, no v-dense job are released during the interval (e_k, s_{k+1}), or SLACKER(ϵ) would have ran that job, and we would get a contradiction to the definitions of the intervals I_a. For $1 \leq j \leq m$, let $K_j \subseteq U$ be the set of v-dense jobs that were released during $[s_j, s_{j+1})$. Hence, $T_A(v, U) = \sum_{i=1}^{m} T_A(v, K_i)$. We will now establish that for each i, $1 \leq i \leq m$, it is the case that $T_A(v, K_i) \leq \frac{1+2\delta}{\delta}(e_i - s_i)$. The result then follows, since $\sum_{i=1}^{m}(e_i - s_i) = T_S(\frac{v}{c}, \mathcal{J})$.

To establish that $T_A(v, K_i) \leq \frac{1+2\delta}{\delta}(e_i - s_i)$, $1 \leq i \leq m$, consider an arbitrary K_i. Now consider an arbitrary job $J_j \in K_i$. Obviously J_j can only be run during the interval $[s_i, d_j]$. We now want to argue that $d_j - s_i \leq \frac{1+2\delta}{\delta}(e_i - s_i)$. This is sufficient to establish that $T_A(v, K_i) \leq \frac{1+2\delta}{\delta}(e_i - s_i)$, $1 \leq i \leq m$.

Since J_j is v-dense and $J_j \in U$, the only reason that SLACKER(ϵ) would not run J_j starting at time e_i is that J_j is ϵ-stale at time e_i. As a result, $d_j - e_i < (1 + \delta)x_j$ by the definition of ϵ-staleness. By our assumption that every job is ϵ-good, $e_i - s_i \geq e_i - r_j \geq \delta x_j$, or J_j would still be ϵ-fresh at time e_i. Therefore we get that

$$d_j - s_i = (d_j - e_i) + (e_i - s_i) \leq (1 + \delta)x_j + (e_i - s_i) \leq \frac{1+\delta}{\delta}(e_i - s_i) + (e_i - s_i) = \frac{1+2\delta}{\delta}(e_i - s_i)$$

■

Lemma 35 $\|A\| \leq \frac{1+3\delta}{\delta} \cdot c \cdot \|R\|$

Proof: Let $\{\phi_1, \phi_2, \ldots, \phi_m\}$ be the set of densities of jobs completed by the adversary where $\phi_i > \phi_{i+1}$ for $1 \leq i < m$. We set $\phi_0 = \infty$, and $\phi_{m+1} = 0$. Also, for $1 \leq i \leq m$, let ℓ_i be the length of time that the adversary ran jobs with density ϕ_i. Similarly, for $1 \leq i \leq m$, let α_i be the length of time that SLACKER(ϵ) ran jobs with densities in the range $(\frac{\phi_{i-1}}{c}, \frac{\phi_i}{c}]$. Applying lemma 34, for all y, $1 \leq y \leq m$,

$$T_A(\phi_y, \mathcal{J}) = \sum_{i=1}^{y} \ell_i \leq \frac{1+3\delta}{\delta} \sum_{i=1}^{y} \alpha_i = T_S(\frac{\phi_y}{c}, \mathcal{J})$$

Therefore,

$$\sum_{y=1}^{m} \left[(\phi_y - \phi_{y+1}) \sum_{i=1}^{y} \ell_i \right] \leq \sum_{y=1}^{m} \left[(\phi_y - \phi_{y+1}) \left(\frac{1+3\delta}{\delta} \right) \sum_{i=1}^{y} \alpha_i \right]$$

By reorganizing the sum

$$\sum_{y=1}^{m} \left[(\phi_y - \phi_{y+1}) \sum_{i=1}^{y} \ell_i \right] = \sum_{i=1}^{m} \ell_i \sum_{y=i}^{m} (\phi_y - \phi_{y+1}) = \sum_{i=1}^{m} \ell_i (\phi_i - \phi_{m+1}) = \sum_{i=1}^{m} \ell_i \phi_i$$

Similarly,

$$\sum_{y=1}^{m}\left[(\phi_y-\phi_{y+1})\left(\frac{1+3\delta}{\delta}\right)\sum_{i=1}^{y}\alpha_i\right]=\frac{1+3\delta}{\delta}\sum_{i=1}^{m}\alpha_i\phi_i$$

Therefore,

$$\sum_{i=1}^{m}\ell_i\phi_i\leq\frac{1+3\delta}{\delta}\sum_{i=1}^{m}\alpha_i\phi_i$$

Note that $||A||=\sum_{i=1}^{m}\ell_i\phi_i$ since we may assume, without loss of generality, that the adversary finishes every job that it starts. Furthermore, by the definition of R and the definition of α_i, $\sum_{i=1}^{m}\alpha_i\frac{\phi_i}{c}\leq||R||$. Hence we can conclude that $||A||\leq c\frac{1+3\delta}{\delta}\,||R||$. ∎

Lemma 36 *If every job is ϵ-good then*

$$||A||\leq\frac{(1+3\delta)c(c-1)}{\delta(c-1)-1}\,||C||$$

Proof: The proof is by algebraic simplification from lemma 35 and lemma 33. ∎

Theorem 37 *If every job is ϵ-good then the competitive ratio of* SLACKER(ϵ) *is at most*

$$\frac{(1+3\delta)(1+\delta+\sqrt{\delta})(1+\sqrt{\delta})}{\delta^{5/2}}$$

Proof: By substitution and noting that $c=\frac{1+\delta+\sqrt{\delta}}{\delta}$, $c-1=\frac{1+\sqrt{\delta}}{\delta}$, $\delta(c-1)-1=\sqrt{\delta}$. ∎

Note that this competitive ratio goes to three as δ increases.

3.2 Faster Processor

We now allow that jobs may have arbitrary laxity, but only allow the adversary to have only a $\frac{1}{1+\epsilon}$ speed processor. Note that this limitation on the speed of the adversary's processor means that the adversary can only finish ϵ-good jobs. Hence, without loss of generality, we may assume that the collection $\mathcal{J}$ of jobs contains only ϵ-good jobs. In this case, we show that $||R||$ approaches $||A||$, instead of 3 $||A||$, as ϵ increases. This will allow us to prove that SLACKER(ϵ) is almost optimal as δ increases.

Lemma 38 *For all $v>0$, $T_A(v,\mathcal{J})\leq\frac{2\delta^2+3\delta+1}{\delta}T_S(v,\mathcal{J})$.*

Proof: The proof is almost identical to the proof of lemma 34. Note that $T_A(v,\mathcal{J})=T_A(v,U)+T_A(v,C)$ by the definition of U. Since SLACKER(ϵ) completed each job in C, and the processor used by the adversary is $(1+\epsilon)$ slower than the one used by SLACKER(ϵ), $T_A(v,C)\leq(1+\epsilon)T_S(v,C)$. Obviously, $T_S(v,C)\leq T_S(\frac{v}{c},C)\leq T_S(\frac{v}{c},\mathcal{J})$. Therefore, $T_A(v,\mathcal{J})\leq T_A(v,U)+(1+\epsilon)T_S(\frac{v}{c},\mathcal{J})$. To finish it is sufficient to show that $T_A(v,U)\leq\frac{1+2\delta}{\delta}T_S(\frac{v}{c},\mathcal{J})$. This is proved identically to how it was proved in lemma 34. ∎

Lemma 39 $||A||\leq\left(\frac{c}{1+2\delta}\right)\left(\frac{2\delta^2+3\delta+1}{\delta}\right)\,||R||$

Proof: The proof is almost identical to the proof of lemma 35. Let $\{\phi_1, \phi_2, \ldots, \phi_m\}$ be the set of densities of jobs completed by the adversary where $\phi_i > \phi_{i+1}$ for $1 \leq i < m$. We set $\phi_0 = \infty$, and $\phi_{m+1} = 0$. Also, for $1 \leq i \leq m$, let ℓ_i be the length of time that the adversary ran jobs with density ϕ_i. Similarly, for $1 \leq i \leq m$, let α_i be the length of time that $\textsc{Slacker}(\epsilon)$ ran jobs with densities in the range $(\frac{\phi_{i-1}}{c}, \frac{\phi_i}{c}]$. Applying lemma 34, for all y, $1 \leq y \leq m$,

$$T_A(\phi_y, \mathcal{J}) = \sum_{i=1}^{y} \ell_i \leq \frac{2\delta^2 + 3\delta + 1}{\delta} \sum_{i=1}^{y} \alpha_i = T_S(\frac{\phi_y}{c}, \mathcal{J})$$

Therefore,

$$\sum_{y=1}^{m} \left[(\phi_y - \phi_{y+1}) \sum_{i=1}^{y} \ell_i \right] \leq \sum_{y=1}^{m} \left[(\phi_y - \phi_{y+1}) \left(\frac{2\delta^2 + 3\delta + 1}{\delta} \right) \sum_{i=1}^{y} \alpha_i \right]$$

Simplifying these sums as before, we get that

$$\sum_{i=1}^{m} \ell_i \phi_i \leq \frac{2\delta^2 + 3\delta + 1}{\delta} \sum_{i=1}^{m} \alpha_i \phi_i$$

Note that the density of a job is defined based on its length defined with respect to a unit speed processor. Since the adversary has a $\frac{1}{1+\epsilon}$ speed processor we get that $\|A\| \leq \sum_{y=1}^{m} \ell_y \frac{\phi_y}{1+\epsilon}$. Further note that, by the definition of R and the definition of α_i, $\sum_{i=1}^{m} \alpha_i \frac{\phi_i}{c} \leq \| R \|$. We can then conclude that $\|A\| \leq \left(\frac{c}{1+2\delta}\right) \left(\frac{2\delta^2+3\delta+1}{\delta}\right) \|R\|$. ∎

Lemma 40

$$\|A\| \leq \left(\frac{2\delta^2 + 3\delta + 1}{2\delta^2 + \delta} \right) \left(\frac{(1+\sqrt{\delta})(1+\delta+\sqrt{\delta})}{\delta^{3/2}} \right) \|C\|$$

Proof: By lemma 33, $(1 - \frac{1}{\delta(c-1)}) \|R\| \leq \|C\|$, and by lemma 39, $\|A\| \leq \left(\frac{c}{1+2\delta}\right) \left(\frac{2\delta^2+3\delta+1}{\delta}\right) \|R\|$. Hence,

$$\|A\| \leq \left(\frac{c}{1+2\delta} \right) \left(\frac{2\delta^2 + 3\delta + 1}{\delta} \right) \left(\frac{\delta(c-1)}{\delta(c-1) - 1} \right) \|C\|$$

Recall that $c = \frac{1+\delta+\sqrt{\delta}}{\delta}$, $c - 1 = \frac{1+\sqrt{\delta}}{\delta}$, $\delta(c-1) - 1 = \sqrt{\delta}$. By algebraic simplification we get the desired result. ∎

Theorem 41 $\textsc{Slacker}(\epsilon)$ is a $(1 + 2\delta)$-speed $\left(\frac{2\delta^2+3\delta+1}{2\delta^2+\delta}\right) \left(\frac{(1+\sqrt{\delta})(1+\delta+\sqrt{\delta})}{\delta^{3/2}}\right)$-approximation algorithm.

Proof: This follows immediately from lemma 40, and by noting that rescaling time so $\textsc{Slacker}(\epsilon)$ a $(1+\epsilon)$ speed processor and the adversary has a unit speed processor will not effect the validity of lemma 40. ∎

Note that as ϵ increases, the performance of $\textsc{Slacker}(\epsilon)$ comes arbitrary close to optimal.

4 Conclusion

This paper is the first to apply resource augmentation analysis to scheduling problems. Subsequently, there have been several other papers that apply resource augmentation analysis to various scheduling problems [7, 11, 17, 22, 24, 28]. We echo the sentiments in [28] that the ultimate judgment of this analysis will depend on the type of algorithms that it recommends, and whether it yields interesting new insights. On the first criteria, initial results seem to indicate that the algorithms recommended by resource augmentation analysis tend to be relatively simple, and are often the algorithms that are used in practice. For example, combining the results in this paper with the results in [16], one can see that the Multi-level Feedback algorithm used by Unix for CPU scheduling is a $(1+\epsilon)$-speed $f(\frac{1}{\epsilon})$-approximation algorithm. On the second criteria, we have seen that resource augmentation analysis can be a useful first step toward developing randomized algorithms. For example, in [16] we developed an almost optimally competitive randomized algorithm for the CPU scheduling problem, and in [17] we developed an optimally competitive randomized algorithm for the online version of $1|r_j, pmtn|\sum(1-U_j)$. In each case, our initial resource augmentation analysis gave us the intuition on how to design competitive randomized algorithms.

Resource augmentation may also be a useful technique for analyzing offline approximation algorithms, see for example [1, 28].

Acknowledgments: We would like to thank Jeff Edmonds, Eric Torng, Joel Wein, Daniel Mosse, Rege Colwell, Richard Suchoza, and Dimitri Zorine for helpful discussions.

References

[1] S. Albers, S. Arora, and S. Khanna, "Page replacement for generalized caching problems", ACM/SIAM Symposium on Discrete Algorithms, 31 – 40, 1999.

[2] S. Baruah, and J. Haritsa, "Scheduling for overload in real-time systems", *IEEE Trans. on Computers*, **46**, 1034–1039, 1997.

[3] S. Baruah, J. Haritsa, and N. Sharma, "On-line scheduling to maximize task completions" IEEE Real-time Systems Symposium, 228–237, 1994.

[4] 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.

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

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

[7] P. Berman and C. Coulston, "Speed is more powerful than clairvoyance", Scandinavian Workshop on Algorithms and Theory, 255 -263, 1998.

[8] A. Borodin, and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[9] P. Brucker, *Scheduling Algorithms*, Springer-Verlag, 1995.

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

[11] J. Edmonds, "Scheduling in the dark", to appear in ACM Symposium on Theory of Computing.

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

[13] L. Hall, "Approximation algorithms for scheduling", Chapter 2 of *Approximation Algorithms for NP-hard Problems*, ed. D. Hochbaum, PWS Publishing, 1997.

[14] B. Kalyanasundaram, and K. Pruhs, "Fault-tolerant real-time scheduling", *European Symposium on Algorithms*, 296–307, 1997.

[15] B. Kalyanasundaram, and K. Pruhs, "The online transportation problem", *European Symposium on Algorithms*, 484–493, 1995.

[16] B. Kalyanasundaram, and K. Pruhs, "Minimizing flow-time nonclairvoyantly", IEEE Symposium on Foundations of Computer Science, 345–352, 1997.

[17] B. Kalyanasundaram, and K. Pruhs, "Maximizing job completions online", European Symposium on Algorithms, 235 – 246, 1998.

[18] E. Koutsoupias, and C. Papadimitriou, "Beyond competitive analysis," IEEE Symposium on Foundations of Computer Science, 394–400, 1994.

[19] D. Knuth, *The Art of Computer Programming*, vol. 1, Addison Wesley, 1973.

[20] 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.

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

[22] T. Lam and K. To, "Trade-offs between speed and processor in hard-deadline scheduling", ACM/SIAM Symposium on Discrete Algorithms, 623 – 632.

[23] E. Lawler, J.K. Lenstra, A. Rinnooy Kan, and D. Shmoys, "Sequencing and scheduling: algorithms and complexity", eds. S. Graves, A. Rinnooy Kan, and P. Zipkin, *Logistics of Production and Inventory*, Handbooks in OR & MS 4, Elsevier Science, Chapter 9, 445–522, 1993.

[24] Y. Mansour, and B. Patt-Shamir, "Jitter control in QoS networks" IEEE Symposium on Foundations of Computer Science, 50 – 59, 1998.

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

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

[27] R. Motwani, S. Phillips, E. Torng, "Non-clairvoyant scheduling", *Theoretical Computer Science*, **130**, 17–47, 1994.

[28] C. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal time-critical scheduling via resource augmentation", ACM Symposium on Theory of Computing, 140–149, 1997.

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

[30] J. Sgall, "On-line scheduling - a survey", *On-Line Algorithms: The State of the Art*, eds. A. Fiat and G. Woeginger, Lecture Notes in Computer Science, Springer-Verlag.

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

[32] G. Woeginger, "On-line scheduling of jobs with fixed start and end time", *Theoretical Computer Science*, **130**, 5 – 16, 1994.

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