Competitive Online Scheduling with Level of Service*

(Extended Abstract)

Ee-Chien Chang[1] and Chee Yap[2]

[1] Department of Computational Science
National University of Singapore
changec@cz3.nus.edu.sg
[2] Department of Computer Science
Courant Institute, New York University

Abstract. Motivated by an application in thinwire visualization, we study an abstract on-line scheduling problem. Unlike most scheduling problems, our schedulers can gain partial merit from a partially served request. Thus our problem embodies a notion of "Level of Service" that is increasingly important in multimedia applications. We give two schedulers `FirstFit` and `EndFit` based on two simple heuristics, and generalize them into a class of greedy schedulers. We show that both `FirstFit` and `EndFit` are 2-competitive, and any greedy scheduler is 3-competitive. These bounds are shown to be tight.

1 Introduction

We study an abstract on-line scheduling problem motivated by visualization across a "thinwire" network [4, 3]. An example of such a visualization problem is a server-client model where the server and client are connected by a thinwire (that is, a bandwidth-limited connection such as the Internet), with the server holding a very large image that the client wishes to visualize. The viewer on the client side can control the transmission process by moving a mouse cursor over a low-resolution copy of the image to be visualized. This mouse motion generates, in real-time, a sequence of sampled positions along the mouse cursor trajectory. Each sampled position (x, y) corresponds to a request for higher resolution data at the position. As the bandwidth is limited, we could only partially serve each request. This is where an on-line scheduler is needed to optimize the decisions. In most scheduling problems, a partially served request does not contribute to the performance of the scheduler. However, in this problem, a partially sent data can still provide useful information to the user. Thus, instead of sending all the requested data, the server has the option of lowering the "level" of the requested service in order to gain an overall better response time. This paper focuses on this level of service property. Note that there is considerable interest in similar Quality of Service (QoS) issues in multimedia research.

* This research was partially funded by NSF grant CCR-9619846.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 453–462, 2001.

We use the standard notion of "competitiveness" in the sense of Sleator and Tarjan [8]. to judge the quality of our online schedulers. A *scheduler* S produces a feasible schedule $S(I)$ for each instance I of our scheduling problem. Each $S(I)$ has an associated *merit*, where $\mathit{merit}(S(I)) \geq 0$. Let $\mathtt{opt}(I)$ denote any feasible schedule for I that maximizes the merit. We say S is *c-competitive* ($c \geq 1$) if for all I,

$$\mathit{merit}(\mathtt{opt}(I)) \leq c \cdot \mathit{merit}(S(I)) + b,$$

where b is a fixed constant. The *competitive ratio* of S is defined by

$$C(S) := \sup_I \frac{\mathit{merit}(\mathtt{opt}(I))}{\mathit{merit}(S(I))}.$$

Thus, we want schedulers S with $C(S) \geq 1$ as small as possible. There is a fairly large literature on competitive algorithms (e.g., [1, 2, 7]. The class of problems most closely related to ours is the online interval packing problem for a single server, where a schedule is a subset of non-overlapping intervals. Lipton and Tomkins [6] study a variant where the input intervals are sorted by their left endpoints. They give a randomized scheduler that is 2-competitive. As we will see, our problem is different from theirs in several ways. Woeginger [9] studied a problem that has several of the features of our problem. Other online interval packing problems can be found in [10, 5].

2 Problem Formulation

We formalize our problem as an on-line scheduling problem. Each *request* q has four parameters

$$q = (s, t, v, w),$$

where s the *start time*, t the *termination time* (or deadline), v is the *volume* (or size), and w is the *weight*. We require

$$v \geq 0 \quad \text{and} \quad w \geq 0.$$

Write $\mathtt{st}(q)$, $\mathtt{dl}(q)$, $\mathtt{sz}(q)$, $\mathtt{wt}(q)$ for the above parameters of q, respectively. Call the half-open interval $(s, t]$ the *span* of q, written $\mathit{span}(q)$. A request q can only be served within its span $(s, t]$, and at any time moment t_0, at most one request can be served.

An *instance* I is a sequence $\langle q_1, q_2 \ldots q_n \rangle$ of requests where the start times of the q_i's are in increasing order: note that we allow $\mathtt{st}(q_i) = \mathtt{st}(q_{i+1})$ even though we nominally say q_i starts before q_{i+1}. How requests are served is described by the schedule. Formally, a *schedule* for I is a piece-wise constant function

$$H : \mathbb{R} \to \{q_1, q_2, \ldots, q_n\} \cup \{\emptyset\},$$

where $|H^{-1}(q_k)| \leq \mathtt{sz}(q_k)$ and $H^{-1}(q_k) \subseteq \mathit{span}(q_k)$ for $k = 1, \ldots, n$. Intuitively, $H(t_0) = q_k$ means the kth request is served at time t_0 and $H(t_0) = \emptyset$ means

no request is being served. A time moment t_0 is called a *breakpoint* if H is discontinuous at t_0. (More precisely, for every $\varepsilon > 0$, there exists δ_i $(0 < \delta_i < \varepsilon$, $i = 1, 2)$ such that $H(t_0) = H(t_0 - \delta_1) \neq H(t_0 + \delta_2))$. We further require a schedule H to has finitely many breakpoints. In addition, for each request q, $|H^{-1}(q)| \leq \mathtt{sz}(q)$. A half-open interval of the form $(t_0, t_1]$ is called a *time-slot*. Without loss of generality, we may assume $H^{-1}(q)$ is a finite union of time slots. The merit *merit*(H) of a schedule is

$$\sum_{j=1}^{n} \mathtt{wt}(q_j)|H^{-1}(q_j)|.$$

Relative to a schedule H at any time t_0, we call

$$v' := \mathtt{sz}(q) - |H^{-1}(q) \cap (-\infty, t_0]|$$

the *residual size* of q. A request q is completely served if $v' = 0$. If $v' > 0$ and $\mathtt{st}(q) \leq t_0 \leq \mathtt{dl}(q)$, then we say q is *pending*. The *residue* of a pending q at time t_0 is the modified request $q' = (t_0, \mathtt{dl}(q), v', \mathtt{wt}(q))$.

For each completely served request, the scheduler gains $\mathtt{sz}(q)\mathtt{wt}(q)$ merit points (so weights are multiplicative). Moreover, partially served requests gains a proportional fraction of this merit. This is unlike usual scheduling problems in which partially served requests receive no merit. Our model is meaningful for the "foveated visualization" studied in [4, 3] because a scheduler can reduce the amount of requested visualization data along a finely graduated scale. This can be achieved by reducing two parameters, the *foveal radius* and/or *foveal resolution*. The foveal radius measures how fast the resolution falls away from the foveal center, and the foveal resolution measures the maximum resolution (which is at the foveal center).

Preemption. It is implicit in the above definitions that the servicing of any request can be preempted as often as we like with no penalty. Hence we may imagine the scheduler to "plan" a schedule based on all the currently residual requests. It services the requests according to this plan until the arrival of a new request. Then it suspends the current plan, recomputes a new plan based on the new set of residual requests, and repeats the process.

Optimal Schedules. We say H is *optimal* for I if *merit*(H) is maximum among all schedules for I. The existence of optimal schedules is not immediate.

Lemma 1. *For all sequences I of n requests, there exists an optimal schedule with at most $2n^2 + n$ breakpoints.*

The optimal schedules for an instance may not be unique. In the full paper, we give a canonical representation whereby the optimal schedule is unique. Let $\mathtt{opt}(I)$ denote the canonical optimal schedule for I.

Ordering of Requests. The schedulers in this paper make decisions by giving priority to heavier weighted requests. In case $wt(p) = wt(q)$, we resolve the tie by treating p as "heavier" than q if and only if p starts before q.

2.1 FirstFit

The online scheduler that always serves the heaviest residual request at each moment is called the FirstFit scheduler. Figure 1 shows the schedule produced by FirstFit on an instance of two requests q_1 and q_2. Although this example may appear contrived, we can modify q_1 to $\widetilde{q}_1$ where $\widetilde{q}_1 = (0, 2, 1, 1+\epsilon)$. For any $\epsilon > 0$, the FirstFit schedule is the one shown in Figure 1.

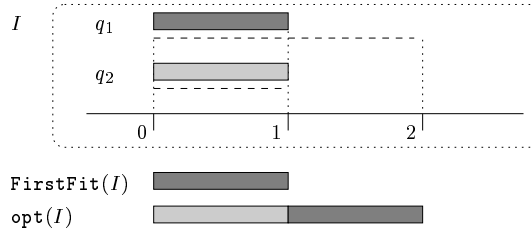

Fig. 1. The top figure illustrates the instance $I = \langle q_1, q_2 \rangle$ where $q_1 = (0, 2, 1, 1)$ and $q_2 = (0, 1, 1, 1)$. Each horizontal "dashed" line represents the span of the request. Although $\mathtt{wt}(q_1) = \mathtt{wt}(q_2)$, q_1 is "heavier" than q_2 by our tie-breaking rule. In $\mathtt{opt}(I)$, q_1 and q_2 are served in the time-slots $(1, 2]$ and $(0, 1]$, respectively. However, in $\mathtt{FirstFit}(I)$, only q_1 is served.

2.2 EndFit

Consider an online scheduler which always serves according to the optimal schedule for the current set of residual requests. This was first suggested by Estie Arkin[1]. To implement such a scheduler, we can invoke a general off-line algorithm for computing optimal schedules upon each new arrival of a request. But it turns out that a very simple scheduler can be used. This scheduler, on an arbitrary instance I, operates as follows:

> Starting from the heaviest request down to the lightest, allocates each request $q \in I$ to the latest available time-slot(s) within $span(q)$.

Call this the OffEndFit scheduler. It is an off-line algorithm because it must see the entire set of requests to make its decisions. This scheduler is optimal for a special class of instances.

Lemma 2. *If I is an instance in which all requests have a common starting time, then $\mathtt{OffEndFit}(I)$ is the canonical optimal schedule for I.*

Let EndFit be the online scheduler which always serves according to the OffEndFit schedule for the residual requests. More precisely, on arrival of a new

[1] Private communication (1997).

request q, `EndFit` preempts the current service. It computes a new schedule P for the current residual requests using `OffEndFit`, and continues by servicing P. Call P the *plan* upon arrival of q. Since all residual requests have a common starting time, P is the canonical optimal schedule. Figure 2 shows the `EndFit` schedule for an instance $I = (q_1, q_2)$. The `EndFit` schedule for this example may appear contrived. To see that it is "correct in the limit", let $I_\epsilon = \langle q_0, q_1, q_2 \rangle$ where $q_0 = (0, 1, 1, \epsilon)$. For any $0 < \epsilon < 1$, the off-line optimal schedule for the current residual requests is unique. As $\epsilon \to 0$, $\texttt{EndFit}(I_\epsilon)$ approaches the one schedule shown in Figure 2.

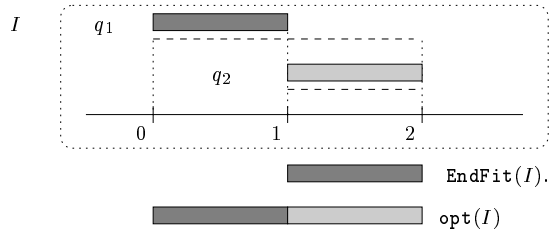

Fig. 2. The top figure illustrates the instance I of two requests: $q_1 = (0, 2, 1, 1)$ and $q_2 = (1, 2, 1, 1)$. In the $\texttt{opt}(I)$, q_1 and q_2 are served. However, in $\texttt{EndFit}(I)$, only q_1 is served.

3 Competitive Ratio of `FirstFit`

Example 1 (Figure 1) shows that the competitive ratio of `FirstFit` is at least 2. We will show that `FirstFit` is 2-competitive. Before presenting the proof, let us give two definitions.

Charging Scheme. Let H, H_1 and H_2 be schedules for an instance I. We often need to argue that the merit of H is no larger than the sum of the merits of H_1 and H_2. Our approach is to *charge* a portion of H to H_1 and the remaining to H_2. Intuitively, the charging process can be viewed as first cutting H_1 and H_2 into pieces and then piecing them together again to form another piecewise-constant function $H_{\texttt{chg}}$. Each piece, after cutting, may be translated before being placed into their slot in $H_{\texttt{chg}}$. As it may turn out that $|H_{\texttt{chg}}^{-1}(q)| > \texttt{sz}(q)$ for some request q, $H_{\texttt{chg}}$ is not necessarily a schedule. The cut-and-paste is done in a way that for all t, $\texttt{wt}(H_{\texttt{chg}}(t)) \geq \texttt{wt}(H(t))$. Therefore,

$$merit(H) \leq merit(H_{\texttt{chg}}) \leq merit(H_1) + merit(H_2). \tag{1}$$

In particular, if $H_1 = H_2$, then we have $merit(H) \leq 2 \cdot merit(H_1)$. When we use the phrase: "charge $(s', e']$ from H to H_1 at $(s, e]$", we mean that a piece $(s, e]$ is cut from H_1 and placed into the slot $(s', e']$ in $H_{\texttt{chg}}$. Equivalently, if $H^{-1}(q)$

is the interval $(s', e']$, we may say that the request q is charged from H to H_1 at $(s, e]$ (see Figure 3). Thus, to show (1), we need to charge each time slot of H to either H_1 or H_2, and ensure that no part of H_1 or H_2 is charged more than one.

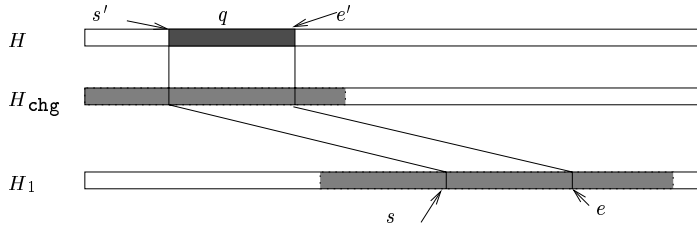

Fig. 3. Charging the request q in H to H_1 at $(s, e]$.

Intactness. A request q is *intact* in a schedule H if $H^{-1}(q)$ is connected, and either $|H^{-1}(q)| = 0$ or $\mathtt{sz}(q)$. Most of our proofs will be simplified if we assume intactness.

Theorem 1. *For any instance I,*

$$\text{merit}(\mathtt{opt}(I)) \leq 2 \cdot \text{merit}(\mathtt{FirstFit}(I)).$$

Proof. Given an instance I, let $H_{\mathtt{ff}} := \mathtt{FirstFit}(I)$ and $H_{\mathtt{opt}} := \mathtt{opt}(I)$. We can assume that requests in I are intact in both $H_{\mathtt{ff}}$ and $H_{\mathtt{opt}}$.

Let H_0 be an identical copy of $H_{\mathtt{ff}}$. We want to charge requests served in $H_{\mathtt{opt}}$ to H_0 and $H_{\mathtt{ff}}$. Let $\{t_1, t_2, \ldots, t_m\}$ be the distinct breakpoints in $H_{\mathtt{opt}}$, where $t_i < t_j$ if and only if $i < j$.

For each t_i, starting from $i := 1$ to $m - 1$, consider the time-slot $(t_i, t_{i+1}]$. Let $q_{\mathtt{opt}} := H_{\mathtt{opt}}(t_{i+1})$. Let $q_{\mathtt{ff}}$ be the lightest request served during $(t_i, t_{t+1}]$ by `FirstFit`. There are two cases:

1. If the request $q_{\mathtt{ff}}$ is not lighter than $q_{\mathtt{opt}}$, charge $q_{\mathtt{opt}}$ from $H_{\mathtt{opt}}$ to $H_{\mathtt{ff}}$ at $(t_i, t_{i+1}]$.
2. Otherwise, charge $q_{\mathtt{opt}}$ from $H_{\mathtt{opt}}$ to H_0 at $H_0^{-1}(q_{\mathtt{opt}})$.

We have to show that in the second case, $|H_0^{-1}(q_{\mathtt{opt}})| \geq |(t_i, t_{i+1}]|$. In the first place, why is the weight of $q_{\mathtt{ff}}$ lighter? The request $q_{\mathtt{ff}}$ is chosen by `FirstFit` because it is the heaviest request among the pending requests. This implies that $q_{\mathtt{opt}}$ is not a pending request, even though t_i is in the span of $q_{\mathtt{opt}}$. So $q_{\mathtt{opt}}$ must have been completely served by `FirstFit`. This implies that $|H_0^{-1}(q_{\mathtt{opt}})| \geq |(t_i, t_{i+1}]|$.

4 Competitive Ratio of `EndFit`

Example 2 (Figure 2) shows that the competitive ratio of `EndFit` is at least 2. We now show that this constant is the best possible.

The upper bound proof is considerably more subtle than the proof for FirstFit. The key result is Theorem 2 below which formalizes this observation about EndFit: it never hurts the performance of EndFit to have a request started at an earlier time. For example, in Figure 2, the performance of EndFit will improve if the request q_2 starts at an earlier time. The analogous fails for FirstFit. For example, in Figure 1, the performance of FirstFit would improve if q_1 starts at a time later than 0.

A request $\widetilde{q}$ is a *trimmed* version of q if $\mathtt{st}(\widetilde{q}) \geq \mathtt{st}(q)$, $\mathtt{dl}(\widetilde{q}) = \mathtt{dl}(q)$, $\mathtt{wt}(\widetilde{q}) = \mathtt{wt}(\widetilde{q})$ and $\mathtt{sz}(\widetilde{q}) \leq \mathtt{sz}(q)$. Thus, a trimmed version of q may start later than the original q. An instance $\widetilde{I}$ is a trimmed instance of I if there is a one-one (not necessarily onto) mapping from $\widetilde{I}$ to I such that any $\widetilde{q}$ in $\widetilde{I}$ is a trimmed version of its corresponding request in I. Clearly, $\mathit{merit}(\mathtt{opt}(\widetilde{I})) \leq \mathit{merit}(\mathtt{opt}(I))$. Similar relationship also holds for EndFit.

Theorem 2. *If $\widetilde{I}$ is a trimmed version of I then*

$$\mathrm{merit}(\mathtt{EndFit}(\widetilde{I})) \leq \mathrm{merit}(\mathtt{EndFit}(I)).$$

We now use Theorem 2 to show that EndFit is 2-competitive.

Theorem 3. *For any instance I,*

$$\mathrm{merit}(\mathtt{opt}(I)) \leq 2 \cdot \mathrm{merit}(\mathtt{EndFit}(I)).$$

Proof. We can assume that requests in I are intact in $\mathtt{opt}(I)$. Let $\widetilde{I}$ be the trimmed instance of I such that for any request $q \in I$, if $(\mathtt{opt}(I))^{-1}(q) = (t_1, t_2]$, then the corresponding trimmed request $\widetilde{q}$ satisfies $\mathtt{st}(\widetilde{q}) := t_1$ and $\mathtt{sz}(\widetilde{q}) := t_2 - t_1$; otherwise if $\mathtt{opt}(I)^{-1}(q) = \emptyset$, then $\widetilde{q}$ satisfies $\mathtt{sz}(\widetilde{q}) := 0$. We can further assume that $\widetilde{I}$ is intact in all the plans of EndFit with $\widetilde{I}$. (Recall that a plan is the optimal schedule for the residues). By definition, we have

$$\mathit{merit}(\mathtt{opt}(I)) = \mathit{merit}(\mathtt{opt}(\widetilde{I})). \tag{2}$$

The instance $\widetilde{I}$ has the nice property that requests arrive at a "constant rate", that is, if a request q starts at time t, then no other request starts during $(t, t + \mathtt{sz}(q))$. Let H_1 and H_2 be two identical copies of $\mathtt{EndFit}(\widetilde{I})$. Our theorem is proved if we show how to charge requests in $\mathtt{opt}(\widetilde{I})$ to H_1 and H_2.

Consider a request q in $\widetilde{I}$. Let P^+ be the plan upon the arrival of q and P^- be the plan just before the arrival of q. There are two cases.

1. If q is allocated in the new plan P^+, then it is possible that there are some requests which are originally allocated in P^-, but not in the new plan. Call these requests the *ousted requests*.
2. Otherwise, call q the ousted request.

Let s be the total size of the ousted requests. Note that $s \leq \mathtt{sz}(q)$ and the total merit of the ousted requests is not more than the total merit of the requests

allocated in $(\mathtt{st}(q), \mathtt{st}(q) + s]$ in P^+. Furthermore, the new plan will be carried out without interruption at least until $\mathtt{st}(q) + \mathtt{sz}(q)$. Charge the ousted requests to H_2 at $(\mathtt{st}(q), \mathtt{st}(q) + \mathtt{sz}(q)]$ and the served requests during $(\mathtt{st}(q), \mathtt{st}(q) + \mathtt{sz}(q)]$ to H_1 at $(\mathtt{st}(q), \mathtt{st}(q) + \mathtt{sz}(q)]$. The above is a valid charging scheme. Thus, we have

$$2 \cdot \mathit{merit}(\mathtt{EndFit}(\widetilde{I})) \geq \mathit{merit}(\mathtt{opt}(\widetilde{I})).$$

By Theorem 2 and (2), we have

$$2 \cdot \mathit{merit}(\mathtt{EndFit}(I)) \geq \mathit{merit}(\mathtt{opt}(I)).$$

5 A Class of Greedy Schedulers

Looking at the behavior of `EndFit` and `FirstFit` on specific examples, it appears that they are complementary in the sense that if `EndFit` performs poorly on an instance, then `FirstFit` will perform well, and vice-versa. This suggests studying some combination of these two heuristics and motivates the generalization to a class of `Greedy` schedulers.

A scheduler S in this class behaves as follows.

(A) At the moment a new request q starts, it suspends the current service (that is, it preempts the currently served request).

(B) Scheduler S computes a new *plan* H, which is a schedule for the set of residues of currently pending requests. We call H a 'plan' because the scheduler may not carry out the schedule as planned due to the subsequent new requests. The plan H is computed by considering the residues one by one, starting from the heaviest request down to the lightest request. Let p be the request being considered and call $|H^{-1}(p)|$ the allocation to p. The allocation to p is subjected to the following restriction:

($*$) The allocation to p must be maximized. For example, if it is possible to completely allocate p, the whole of p must be allocated. However, there is no restriction on where p is allocated. Time-slots, once allocated, are not subsequently revised in creating this plan.

(C) It carries out the plan until a new request starts, whereupon we go back to step (A).

Different members of the `Greedy` class differ only in their strategies for (B) subjected to the restriction($*$). Note that our first example `FirstFit` is a `Greedy` scheduler: the request p in ($*$) is allocated in the earliest possible time-slots. The second example `EndFit` is also a greedy scheduler. Its strategy for (B) is rather counter-intuitive: the request p is allocated in the *latest* possible time slots.

By combining the counter examples for `FirstFit` and `EndFit`, we can find a `Greedy` scheduler whose competitive ratio ≥ 3. The next theorem shows that this bound of 3 cannot be improved.

Theorem 4. *Every* `Greedy` *scheduler is 3-competitive.*

6 General Lower Bound

From the previous section, we know that every greedy scheduler is 3-competitive. Are there schedulers outside the `Greedy` class with competitive ratio less than 2? We note a partial result in this direction: *every deterministic scheduler has competitive ratio at least* $2(2-\sqrt{2}) > 1.17$.

In proof, consider this adversary: at time 0, the adversary releases two requests $q_0 := (0, 2, 1, 1)$ and $q_1 := (0, 1, 1, \sqrt{2}-1)$. At time $t = 1$, let the residual size of q_0 be s_0. If s_0 is less than $1/2$ then the request $q_2 := (1, 2, 1, 1)$ is released. Otherwise, no further requests will be released. It may be verified that any deterministic scheduler achieve a merit of at most $\frac{1}{2(2-\sqrt{2})}$ of the maximum possible.

Unfortunately, this lower bound of 1.17 leaves a wide gap from the current upper bound of 2. On the other hand, no simple variation of this adversary seems to give a better lower bound.

7 On the Number of Breakpoints and Multi-tasking

Both `FirstFit` and `EndFit` make $O(n)$ breakpoints where n is the number of requests. Does the number of breakpoints affect the performance of a scheduler? We give an alternative formulation of the scheduling problem which could be viewed as allowing an infinite number breakpoints. In this *multi-tasking* environment, several requests can be served concurrently, but each at a (possibly) different rate. However, in any time interval of size Δt, the total size of requests served within this interval must not exceed Δt. Equivalently, we allow "fractional service" where the total service at any moment sums to 1.

A concrete example of such a scheduler is `FirstEndFit`: It simulates `FirstFit` and `EndFit` concurrently and serves half of what `FirstFit` and `EndFit` would serve. That is, if `FirstFit` and `EndFit`will serve q and p respectively in the time slot $(s_0, t_0]$, then `FirstEndFit` will serve p and q concurrently but each at half the rate. We suspect that `FirstEndFit` is (3/2)-competitive.

Note that `FirstEndFit` can be also viewed as the following randomized scheduler under the original single-tasking setting: Before receiving any request, it tosses a fair coin; If the outcome is `head`, then it simulates `FirstFit`, otherwise it simulates `EndFit`.

Clearly, the expected merit gained by this randomized scheduler is same as the merit gained by `FirstEndFit`.

8 Conclusion

We have formulated a "level-of-service" scheduling problem that arises naturally in our thinwire visualization applications. This formulation is also useful in real-time systems where quality of jobs can be traded-off for time. We have derived several competitive algorithms in this setting. We continue to investigate the

many interesting questions that are open. Besides sharpening the results in the paper, we pose the following directions for further work: (1) Find optimal schedulers which are not restricted to be on-line. (2) Study the problem with other measures of merit (instead of multiplicative weights in this paper). (3) Introduce a model of penalty for preemption. (4) Introduce randomization.

Acknowledgments

We thank Estie Arkin and Yi-Jen Chiang for discussions about the problem.

References

1. S. Albers. Competitive online algorithms. BRICS Lecture Series LS-96-2, BRICS, Department of Computer Science, University of Aarhus, September 1996.
2. S. Albers and J. Westbrook. A survey of self-organizing data structures. Research Report MPI-I-96-1-026, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, October 1996.
3. E. Chang, C. Yap, and T.-J. Yen. Realtime visualization of large images over a thinwire. In *IEEE Visualization '97 (Late Breaking Hot Topics)*, pages 45–48, 1997.
4. E.-C. Chang. *Foveation Techniques and Scheduling Issues in Thinwire Visualization.* PhD thesis, Department of Computer Science, New York University, May 1998.
5. J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. *Integer Programming and Combinatorial Opt.*, pages 404–414, 1996.
6. R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 302–311, 1994.
7. L. A. McGeoch and D. D. Sleator, editors. *On-Line Algorithms.* DIMACS series in Discrete Mathematics and Theoretical Computer Science, volume 7. AMS, 1992.
8. D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Comm. of the ACM*, 28(2):202–208, 1985.
9. G. J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Computer Science*, 130:5–16, 1994.
10. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. 36th Annual Symp. on Foundations of Computer Science*, pages 374–382, 1995.