On-line Scheduling with Partial Job Values and Bounded Importance Ratio *

Francis Y. L. Chin and Stanley P. Y. Fung
Department of Computer Science and Information Systems
The University of Hong Kong, Hong Kong.
{chin,pyfung}@csis.hku.hk

Abstract

We consider a new form of on-line preemptive scheduling problem in which jobs are not required to be completed to get the value; instead they get partial value proportional to the amount processed. This problem first arises as a quality-of-service problem in transferring multimedia content over a narrowband network. Previous heuristics for this problem are 2-competitive. In this paper we propose a new heuristic that achieves an improved competitive ratio when the importance ratio is bounded. Specifically, for job weights within the range [1..B], our heuristic is $(2\lceil \lg B \rceil + 3)/(\lceil \lg B \rceil + 2)$*-competitive, and the bound is tight.*

Keywords: Online algorithms, scheduling, competitive analysis, partial job values.

1 Introduction

In the traditional online scheduling model, it is required to schedule jobs so as to maximize the total value received for completing the jobs [1, 6]. Partially processed jobs (i.e., those that cannot be completed before the deadline) get no value. In this paper we consider a variation in which partially processed jobs get a value proportional to the amount processed. This problem first arises in multimedia content transmission over a network with low bandwidth [3], but it also has other natural applications.

A *request* (or *job*) q is specified by a 4-tuple (s, t, p, w) where s is the *release time*, t is the *deadline*, p is the *processing time* (also called *length*), and w is the *weight*, i.e., the value obtained per unit time spent on processing the job. They are sometimes denoted by $s(q), t(q), p(q)$ and $w(q)$. The span of a job, $span(q)$, is the time interval $[s(q), t(q)]$. Scheduling is done in a uniprocessor setting, with preemption allowed at no penalty (a job can be resumed at the point where it was last preempted). A *residue job* is a job that is partially and not totally processed. A job is *pending* if it is not completed and its deadline has not been reached. A job q processed for a total time $l(q)$ gets a value $l(q) \times w(q)$.

The scheduler needs to schedule the jobs *online*, i.e., the jobs are only known when they arrive and the scheduler cannot make changes to the schedule in the past. When a job arrives, all its details, including the processing time, are known. Online algorithms are commonly analyzed in terms of their competitive ratios, introduced in [8]. An online algorithm is *c-competitive* if for any instance of jobs, the value produced by the online algorithm is at least $1/c$ that of the optimal offline algorithm. Competitive analysis and various forms of online scheduling are discussed in detail in [2, 7].

In [3], two heuristics for this problem are described. FIRSTFIT (FF) always serves the heaviest residue job, while ENDFIT (EF) starts from the heaviest residue job and allocates each residue job to the latest possible timeslot(s) within its span. For example, given $q_1(0,6,2,1), q_2(0,3,2,2), q_3(0,5,2,3)$, FF would schedule $[q_3, q_3, q_2, q_1, q_1]$ in the unit timeslots in this order, which is not optimal. EF would produce an optimal schedule $[q_1, q_2, q_2, q_3, q_3, q_1]$.

Both heuristics are shown to be 2-competitive and their bounds are tight. FF performs poorly when a job q_{max} is served, but only slightly heavier than the other job q_i whose deadline is much earlier, and consequently q_i is not served. On the other hand, EF performs poorly when a very light job q_{EF} is served immediately instead of another much heavier job q_j because of q_j's later deadline. But again q_j may not be served because of the arrival of some heavier job later. In fact,

*This work is supported in parts by an RGC grant.

these two schedulers are complementary to each other, in the sense that EF will usually perform well on those instances which FF performs poorly and vice versa. This suggests that the combination of those two heuristics might be a promising approach for a better scheduler. FIT is a new scheduler which specifies a condition for choosing q_{max} or q_{EF}. Unfortunately, FIT is also 2-competitive, but performs better when the *importance ratio* B is bounded, where the importance ratio is defined as the maximum ratio of job weights.

In Section 2, we describe our new scheduler FIT, which is a combination of FF and EF, and show that it is 2-competitive. This bound is tight and cannot be improved by choosing other parameters for the condition. In Section 3, we show that FIT is $(2\lceil\lg B\rceil + 3)/(\lceil\lg B\rceil + 2)$-competitive [1]. This bound is tight and can be much less than 2, e.g., FIT is 1.75-competitive when $B = 4$. Section 4 concludes the paper. Due to space limitations, we only give sketches in some of the proofs.

2 The Scheduler FIT

Intuitively, q_{max} or q_{EF} is scheduled based on their weights. q_{max} is scheduled when $w(q_{max})$ is much larger than $w(q_{EF})$, i.e., $w(q_{max}) > rw(q_{EF})$ for $r > 1$, otherwise q_{EF} is scheduled. Formally, for a fixed $r > 1$, our scheduler FIT-r behaves as follows:

> Let q_{max} be the heaviest job, and q_{EF} be the first job to be scheduled (possibly partially) for a time of l_{EF} in an *EF* schedule (called *EFplan*). If $w(q_{EF}) < w(q_{max})/r$, q_{max} is scheduled for a time of $\min(l_{EF}, p(q_{max}))$. Otherwise q_{EF} is scheduled for a time of l_{EF}. The scheduler is invoked again after the scheduled job has completed its scheduled time or if new jobs arrive [2].

2.1 Preliminaries

Without loss of generality, we restrict the analysis to a discrete version in which all job times are integers. We also assume all jobs are of unit length, i.e., $p = 1$. Any integer-length job (s, t, p, w) can be dissected into a set of p unit-length jobs $(s, t, 1, w)$ without affecting the schedules. These together mean that the FIT scheduler is being invoked at every integer time. We also assume the optimal scheduler can only change its job at these integer times. It could be shown that the general case has the same competitive ratio.

Time is divided into *timeslots* (or simply *slots*) of equally long time intervals. For any two slots s and s', $s < s'$ denotes s is earlier than s'. We can also use s to denote an instance of time: we say 'time s' to refer to the moment at the immediate beginning of slot s.

Let OPT and FIT denote the optimal offline schedule and the schedule produced by the FIT-r scheduler, respectively. $H(s)$ denotes the job in slot s in schedule H; in this paper $H = OPT$ or FIT. An FF-slot is a slot in FIT where q_{max} is picked, and similarly an EF-slot is a slot in FIT where q_{EF} is picked.

It would be convenient to our analysis to sometimes think of the FIT scheduler as a 2-stage process. Everytime the scheduler is invoked, it first goes into the EF-stage to compute $EFplan$ for all residue jobs (including the jobs just released), finds q_{EF}, and then determines whether q_{max} is heavy enough to be chosen instead (the FF stage). We use $EFplan(t, s)$ with $s \geq t$ to denote the job in slot s in this plan generated at time t. A pending job in $EFplan$ is said to be *planned*.

The main idea of the competitiveness proof is to 'charge' the jobs in OPT to jobs in FIT. One straightforward charging scheme is to charge $OPT(s)$ to $FIT(s)$ if the $OPT(s)$ is not too much heavier than $FIT(s)$, otherwise charge $OPT(s)$ to itself in the FIT schedule. In the following we shall show that when this happens, $OPT(s)$ must be an FF-job in the FIT schedule.

First of all, it is easy to see that if a slot s_1 is an EF-slot, $FIT(s_1) = q$, then all slots within *span*(q) are planned with jobs not lighter than q at time s_1. In the following we sketch the proof that this will remain true for all later times, even after the arrival of new jobs and/or some FF-slots in *span*(q).

- The arrival of a new job causes $EFplan$ to be changed, but only to 'squeeze' the new job into the plan, and 'pushes' other jobs lighter than itself to earlier times, or even discards them if there is no available slots left. This will make the planned jobs within *span*(q) no lighter.
- If some slots are FF-slots in *span*(q), these FF jobs must be 'new', i.e., not released yet at time s_1, otherwise they would be 'heavy enough' to make s_1 become an FF-slot. Therefore the 'old' jobs in $EFplan$ (i.e., those in $EFplan$ at time

[1] Throughout this paper lg denotes log to base 2.
[2] Assume ties are broken consistently.

s_1) are still pending, and the job weights therefore would not decrease.

The above discussion shows that: 'if q is a job, s_1 is an EF-slot, and $FIT(s_1) = q$, then for all slots within $span(q)$, $EFplan$ contains jobs no lighter than q at all later times.' The following lemma is the contrapositive version of the above statement (for a particular slot s_k). Since it will be used frequently later, we state it explicitly.

Lemma 1 *Suppose s_1 and s_k are slots ($s_k > s_1$), $FIT(s_1) = OPT(s_k) = q$, and $q' = FIT(s_k)$. If s_k is an EF-slot and $w(q) > w(q')$, or s_k is an FF-slot and $w(q) > w(q')/r$, then s_1 must be an FF-slot.*

Lemma 1 shows that when $OPT(s)$ is not too light compared to $FIT(s)$, and $OPT(s)$ exists in FIT earlier, then it must be in an FF-slot. But it does not guarantee the existence of the slot in the first place. The next lemma establishes the existence of a FIT slot for charging jobs in OPT when, at certain slot s, $OPT(s)$ is much heavier than $FIT(s)$.

Lemma 2 *For a slot s, if $w(OPT(s))/w(FIT(s)) > r$, then there exists an FF-slot $s' < s$ such that $FIT(s') = OPT(s)$.*

Proof. (Sketch) If $OPT(s)$ is not in FIT before s, then it will be pending in FIT at time s, thus it (or some heavier job) would be scheduled as an FF-job in s rather than the current $FIT(s)$. Therefore $OPT(s)$ must be in a slot s' in FIT before s. That it is an FF-slot follows from Lemma 1, since we have $w(FIT(s')) = w(OPT(s)) > rw(FIT(s))$. □

2.2 Charging Groups

We want to 'charge' the jobs in OPT to jobs in FIT. To achieve a good competitive ratio we need to guarantee the jobs in OPT are not too much heavier than that in FIT. Naturally, we can charge the OPT job to the FIT job in the same slot. If the OPT job is much heavier, Lemma 2 tells us that this heavy job appeared in FIT in an earlier FF-slot, and we may charge to there instead. This leads us to define the following *charging rule*:

> **Charging Rule.** For every slot s, if $OPT(s)$ appears in an FF-slot s' in FIT, charge $OPT(s)$ to $FIT(s')$ (note that s' may be earlier than, same as or later than s), otherwise charge $OPT(s)$ to $FIT(s)$.

Fig. 1 shows some possible scenarios. It is easy to see that all jobs in OPT are charged. Each EF-slot in FIT is charged at most once, each FF-slot charged at most twice, and charging to a different slot is possible only if the slot being charged is an FF-slot.

A *charging group* of s consists of the job at slot s in FIT and the jobs in OPT that charge to it. An *FF charging group* of s (or FF-group for short) consists of a job y in an FF-slot s, the same job y in OPT in (possibly) another slot s', and may also include a job x in $OPT(s)$ charging to y in FIT. We denote this by $(x, y; y)$. An *EF charging group* of s (or EF-group) consists of a job y in an EF-slot s and the job x in $OPT(s)$ charging to y. We denote this by $(x; y)$. When no confusion arises, the x and y inside the symbols $(x, y; y)$ and $(x; y)$ may also denote the weight of a job instead of the job itself. Fig. 1 shows some charging groups. Define the *charging ratio* of a charging group to be the sum of job weights in OPT to the job weight in FIT, i.e., $(x + y)/y$ for an FF-group and x/y for an EF-group.

Lemma 3 *The charging ratio of any charging group is at most* $\max(2, r)$.

Proof. Consider any slot s in FIT and its associated charging group, and let $x = OPT(s), y = FIT(s)$.

i. If s is an EF-slot and

  a. $w(x)/w(y) \leq r$ (e.g., Fig. 1(a) first group), the charging group is $(x; y)$ and the claim holds.

  b. $w(x)/w(y) > r$ (e.g., Fig. 1(a) third group), then x must exist in an FF-slot $s' < s$ in FIT (Lemma 2). x is therefore charged to $FIT(s')$ and not y. Thus the charging group is $(0; y)$, with charging ratio $= 0$.

ii. if s is an FF-slot and

  a. $w(x)/w(y) \leq 1$ (e.g., Fig. 1(a) second group), the charging group is $(x, y; y)$ with charging ratio $= (w(x) + w(y))/w(y) \leq 2$.

  b. $w(x)/w(y) > 1$ (e.g., Fig. 1(c) second group), then x must exist in a slot $s' < s$ in FIT (otherwise y is not the heaviest pending job at time s), and s' must be an FF-slot (Lemma 1). Therefore x is charged to $FIT(s')$ and not to y. Thus the charging group is $(0, y; y)$, giving a charging ratio of 1. □

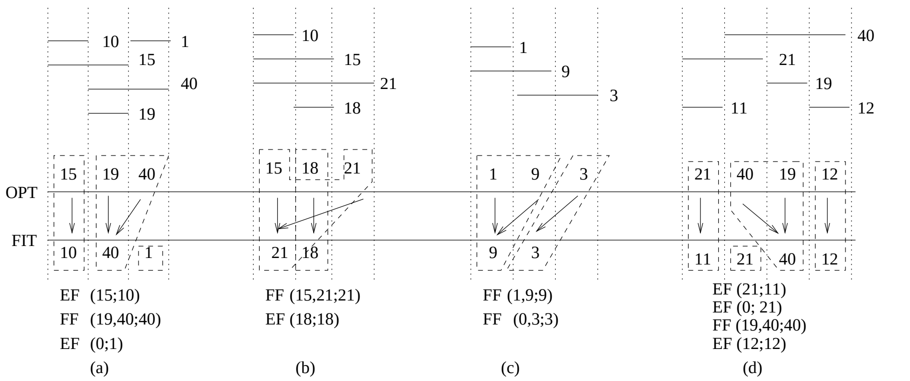

Figure 1: Charging groups. Numbers represent job weights and we assume $r = 2$.

Theorem 1 *FIT-2 is 2-competitive and the bound is tight. In fact FIT-r is no better than 2-competitive for any value of r.*

Proof. By setting $r = 2$, all charging groups have their charging ratios bounded by 2 (Lemma 3). Thus FIT-2 is 2-competitive.

Suppose $r > 2$. We consider the following instance for large k:

2^k copies of $(0,2^{k+1},1,1)$;

for $i = k, k-1, ..., 0$, 2^i copies of $(2^{k+1} - 2^{i+1}, 2^{k+1}, 1, r^{k+1-i})$;

1 copy of $(2^{k+1} - 1, 2^{k+1}, 1, r^{k+1})$.

Fig. 2 shows the instance and the schedules for $k = 2$. FIT chooses q_{EF} in all slots. The competitive ratio is

$$\frac{2^k r + ... + 4r^{k-1} + 2r^k + r^{k+1} + r^{k+1}}{2^k + ... + 4r^{k-2} + 2r^{k-1} + r^k + r^{k+1}}$$

which tends to 2 for large k.

Suppose $r < 2$. Consider the following instance: (Fig. 3)

for $i = k, k-1, ..., 0$, 2^i copies of $(2^{k+1} - 2^{i+1}, 2^{k+1}, 1, r^{k-i})$;

1 copy of $(2^{k+1} - 1, 2^{k+1}, 1, r^k)$;

for $0 \le i \le 2^{k+1} - 1$, $(i, i + 2^{k+1} + 1, 1, r^{f(i)} + \epsilon)$ where $f(i) = k + 1 - \lceil \lg(2^{k+1} - i) \rceil$.

FIT chooses q_{max} in all slots. The competitive ratio is (neglecting the small ϵ)

$$\frac{2(r^k + 2r^{k-1} + 4r^{k-2} + ... + 2^k) + r^k + r^{k+1}}{(r^k + 2r^{k-1} + 4r^{k-2} + ... + 2^k) + r^{k+1}}$$

which again tends to 2 for large k.

When $r = 2$, both schedules gives a competitive ratio 2 when k is large enough. □

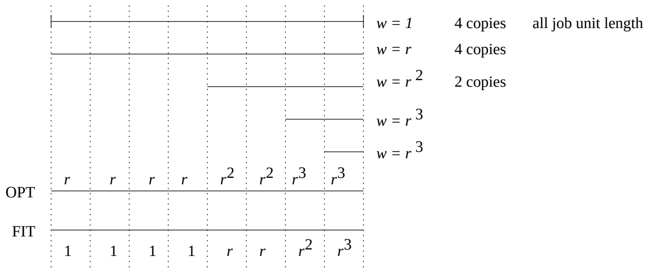

Figure 2: The instance for Theorem 1 ($r > 2$) and the schedules.

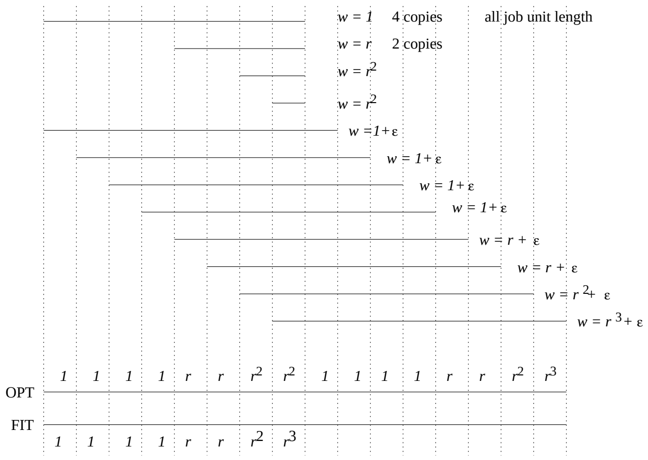

Figure 3: The instance for Theorem 1 ($r < 2$) and the schedules.

3 Competitiveness of FIT with bounded B

Let B denote the maximum ratio of job weights, i.e., all job weights w satisfy $1 \le w \le B$. This is called *importance ratio*. The constructions in Theorem 1 produce instances with unbounded importance ratio. In real applications, the importance ratio tends to be a small constant or may be known in advance; hence B can be considered as part of the input and bounded. The next subsections show that charging groups can be 'linked' together, and prove that such linkages can be used to lower the competitive ratio in the analysis.

3.1 Links

When we consider each individual charging group, the 2-competitive ratio is tight only in case i.a. and ii.a. of Lemma 3. In those cases where the OPT job charged to the FIT job in the same slot is sufficiently large, the OPT job is actually planned in a later slot in $EFplan$ at that time. For example, in Fig. 1(a),(b), the job with weight 15 is originally planned in the second slot, only to be discarded later due to arrival of a heavier job. This heavier job belongs to another charging group. In the following, we show that when a charging group has charging ratio larger than a certain value, a 'link' can be generated to connect it to another charging group (later we will see that these values are 1 for EF-group and $1 + 1/r$ for FF-group). We shall make sure that the charging group being connected to is 'heavier' than the previous group. The links may continue (i.e., this group may link to yet another group), but this linking cannot be continued indefinitely because there is a bound on B. It will terminate when it reaches a group that has a small charging ratio. We show that, when we consider all the charging groups linked together in this way, the charging ratio is smaller.

Formally, a *link* points from the OPT job of a charging group (the 'x' job in a FF-group $(x, y; y)$ or EF-group $(x; y)$) to the FIT job of another later charging group. Initially, it represents the situation that the OPT job is planned in a later slot in FIT while OPT schedules it in the current slot. These planned jobs, however, may later be dropped out of the schedule because of the arrival of heavier jobs. To model this, we require all links to satisfy the following two *link constraints*:

- *(weight constraint) For any link $x_i \to y_j$, if y_j is in an EF-slot, then $w(y_j) \geq w(x_i)$; if y_j is in an FF-slot, then $w(y_j) \geq rw(x_i)$.*
- *(span constraint) For all links $x_i \to y_j$, y_j must be in a slot within $span(x_i)$.*

We construct the links incrementally while sweeping the schedules from beginning to end. Note that, during this process, links point to future slots in $EFplan$, and therefore have to satisfy the weight constraints for EF-slots. When a slot s is first swept, the following *linking rules* are used to determine when links are generated and where they initially point to. (The location may need to be changed subsequently when we further sweep the schedules, see Lemma 5.)

Linking Rules. For each FIT slot s, a link is generated if and only if:

i. s is an EF-slot in a charging group $(x; y)$, $1 < w(x)/w(y) \leq r$.
ii. s is an FF-slot in a charging group $(x, y; y)$, $1/r \leq w(x)/w(y) \leq 1$.

In both cases, link x to $FIT(s'')$, where s'' is a FIT slot such that $EFplan(s, s'') = x$.

The following lemma shows that charging groups that have a charging ratio 'too large' must be able to generate a link.

Lemma 4 *For any EF-group with charging ratio > 1, or any FF-group with charging ratio $\geq 1 + 1/r$, a link can always be generated according to the above rules, while satisfying the link constraints.*

Proof. It is clear that any charging group with ratio larger than the specified limits must belong to either one of the cases in the linking rules, and that the link generated will satisfy the link constraints. What remains to prove is that x is in $EFplan$.

i. s is an EF-slot in a charging group $(x; y)$, $1 < w(x)/w(y) \leq r$. If x is not pending at time s, i.e., it exists in FIT in a slot $s' < s$, then s' must be an FF-slot (Lemma 1). Hence x is charged to $FIT(s')$ instead, contradicting $(x; y)$ being a charging group. Thus x is still pending, and must be planned in a slot $s'' > s$ (otherwise y will not be scheduled there since $w(x) > w(y)$). Thus there must be such a slot s'' to be linked. (e.g., Fig. 1(a) first group.)

ii. s is an FF-slot in a charging group $(x, y; y)$, $1/r \leq w(x)/w(y) \leq 1$. The proof is similar to i., and x must be planned in a slot $s'' > s$, since $w(x) \geq w(y)/r > (r \cdot w(EFplan(s, s)))/r = w(EFplan(s, s))$. (e.g., Fig. 1(b) first group.) □

A *linking slot* is a slot s such that $OPT(s)$ has a link pointing out. Other slots are called *non-linking slots*. When a new slot is swept, $EFplan$ may be changed and previous links may need to be rearranged to satisfy the weight constraints. In

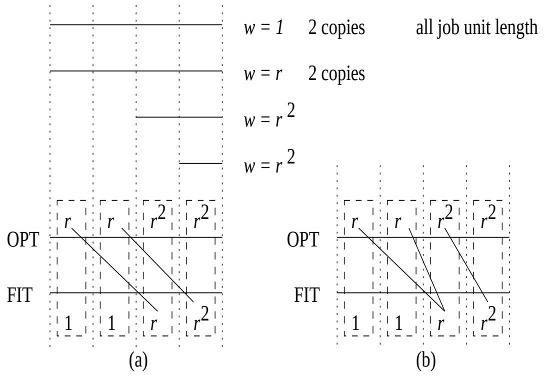

Figure 4: Arranging links: (a) after two slots are swept; (b) after three slots are swept.

the following, we show that the links can be rearranged when sweeping the schedules to maintain the following invariants:

- Only the current slot is pointed to by at most two links; all other future slots are pointed to by at most one link.
- All links satisfy the link constraints.

For each slot s being swept, the link-rearranging algorithm is as follows:

Case 1. If s is an EF-slot and generates a new link $x \to y_j$, and if there is an existing link $z_1 \to y_j$, then relink z_1 to $EFplan(s, z_1)$, or to s if z_1 is not pending at time s. Repeat this relink process if there are other links pointing to $EFplan(s, z_1)$, unless it is relinked to slot s.

Case 2. If s is an FF-slot, perform the same operation as if s is an EF-slot, and recompute $EFplan$ after the heaviest job is moved to slot s. Since some light jobs may have moved to fill in the empty slot left by the heaviest job, it is possible that some links now violate the weight constraints. For every such link $z \to y_k$, it can be shown that z must be planned in a slot after y_k. If no link points to this slot, move the link to point to this slot (i.e., $z \to y_k$ becomes $z \to z$). Otherwise we have a link $z' \to z$. Swap the links to get $z \to z$ and $z' \to y_k$. Repeat the process if this new link violates the weight constraint.

Fig. 4 shows an example of arranging links.

Lemma 5 *By the above algorithm, the links are arranged so that after the whole schedule is swept, each linking slot is pointed to by at most two links, and each non-linking slot is pointed to by at most one link. All link constraints are satisfied.*

Proof. (Sketch) It is easy to see that the first invariant is maintained by the algorithm. It can also be seen from the algorithm that a non-linking slot will not be pointed to by more than one link. These imply the first part of the lemma.

The second part of the lemma is true if we can maintain the second invariant. Suppose the invariant holds before slot s. Now we come to slot s. Let $x = OPT(s)$.

Case 1: s is an EF-slot. It is easy to see that the link constraints are satisfied if x does not generate new links, or x generates a link to a slot without other links. The invariant is also maintained when links are redirected to other slots as long as the job is in $EFplan$. However if the redirection finishes at slot s because a link $z \to y_j$ cannot be redirected, i.e., z does not exist in $EFplan$, then either z is already scheduled in FIT before s, or z is 'squeezed out' from $EFplan$. In both cases it can be shown that $w(FIT(s)) \geq w(z)$, thus satisfying the weight constraint.

Case 2: s is an FF-slot. The proof is similar to that of Case 1, except that the weight of the job in s is at least r times heavier and the link(s) to s should also satisfy the weight constraint. It can be shown that the redirection (swapping) of links as described in the algorithm ensures that the link constraints are satisfied and that it must terminate. $\square$

3.2 Charging Trees

A *charging tree* is defined as a binary tree, whose nodes are either EF-groups (called *EF-nodes*) or FF-groups (called *FF-nodes*). Nodes in a tree are connected by links, that connect charging groups as described in the previous subsection. That is, each non-root EF-node is represented by $(x_i; y_i)$ with $y_i < x_i \leq ry_i$ and non-root FF-node by $(x_i, y_i; y_i)$ with $y_i/r \leq x_i \leq y_i$. The links have to satisfy the weight constraints: a link from node i to node j (with x_i and y_j in node i and j respectively) will ensure that $x_i \leq y_j$ if j is an EF-node, and $rx_i \leq y_j$ if j is an FF-node. The *charging ratio* of a charging tree is defined as

$$\frac{\Sigma_{FFnode}(x_i + y_i) + \Sigma_{EFnode}(x_i)}{\Sigma_{FFnode}(y_i) + \Sigma_{EFnode}(y_i)}$$

summing over all nodes in the tree.

Given a pair of OPT and FIT schedules, the charging groups are linked together to form a forest of charging trees, as described in the previous subsection. Each such charging tree is a binary tree since each charging group can be pointed by at most two links (Lemma 5). However the root

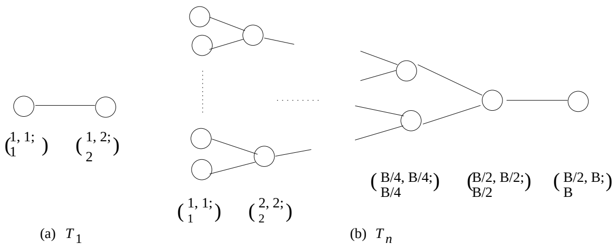

Figure 5: The 'worst' charging tree.

cannot have two children because, by Lemma 5, it would then generate another link itself and thus not a root. If we can bound the charging ratio of each charging tree, then the competitive ratio of the scheduler is also bounded by the same ratio.

From now on we set $r = 2$, and for simplicity assume $B = 2^n$ for some positive integer n. Consider the charging tree T_n for $B = 2^n$ in Fig. 5. T_n is a full binary tree (except at the root). All nodes in the same level are identical. All its nodes are FF-nodes, 'doubling' at all levels (except the topmost), and has $n + 1$ levels of nodes. We will see later that T is indeed realizable, i.e., there exists an instance of jobs such that the OPT and FIT schedules produce charging groups and links corresponding to this tree. We are going to show that any charging tree with importance ratio bounded by B will not have a charging ratio larger than that of T_n, thus providing an upper bound of the competitive ratio.

Lemma 6 *Given any charging tree T with importance ratio B, there always exists a full binary charging tree (except at the root) of FF-nodes, with the same nodes at each level, such that its importance ratio is B and charging ratio no less than that of T's.*

Proof. We first show that any charging tree can be transformed to a full binary tree with the same nodes at each level. For any subtree rooted at an internal node v, consider the left and right subtrees (including empty subtrees). If the two subtrees are not identical, replace the left subtree with an identical copy of the right subtree, or vice versa; it is easy to show that one of these operations will increase the charging ratio. Apply this process to all nodes level-by-level, starting from the leaves, gives a full binary tree (except at the root) with same nodes in the same level.

Our next step is to show that all nodes are FF-nodes. We can assume that the root is an FF-node. (If the root is an EF-node $(x; y)$, the worst case is when $x = y$ since no link is generated from the root. It can be shown that replacing it with an FF-node $(x/2, x; x)$ and changing other parts of the tree appropriately can maintain all constraints and will not decrease the charging ratio.) Because of weight constraints, all EF-nodes $(x_i; y_i)$ in the tree has $y_i \le B/2$. Thus they can be replaced by FF-nodes $(x_i, 2y_i; 2y_i)$ without affecting the importance ratio. It is easy to see that this operation will not violate the weight constraints of the links in the tree, and will not decrease the charging ratio of the original tree. □

Lemma 7 *The charging tree T_n has the maximum possible charging ratio among all charging trees with importance ratio $B = 2^n$.*

Proof. We only need to consider those charging trees stated in Lemma 6. Without loss of generality, assume the tree has at least two levels of nodes. We first prove that any charging tree can be transformed to have leaves (1,1;1) and parents-of-leaf being (x,2;2), without decreasing the charging ratio. (Here and following x denotes an arbitrary number.) Once this is proved we can restrict our attention to those trees.

Suppose the leaves are $(x_1, y_1; y_1)$. By changing them to $(x_1, x_1; x_1)$, the charging ratio is increased. Next consider the parent-of-leaf, $(x_2, y_2; y_2)$ where $y_2 \ge 2x_1$ because of the weight constraint. If $y_2 > 2x_1$, we can, without decreasing the charging ratio, scale up the leaves from $(x_1, x_1; x_1)$ to $(y_2/2, y_2/2; y_2/2)$, and then scale down the whole tree so that the lowest two levels are (1,1;1), (x,2;2).

Now we prove by induction on n that 'T_n is the worst charging tree among those with leaves (x,1;1) and job weights $\le B$'.

For the base case $n = 1$, i.e. $B = 2$, it can be verified that T_1 is a worst tree (Fig. 5(a)).

Suppose T_p is the tree having the maximum charging ratio for $B = 2^p$. Consider a charging tree with importance ratio $2B = 2^{p+1}$. Consider the subtree T' of this tree by ignoring the leaves. For the whole tree to be a worst tree, T' itself has to be worst. By induction hypothesis, a tree with leaves (x,1;1) and job weights $\le B$ is worst when it is T_n. Therefore, the worst tree with leaves (x,2;2) and job weights $\le 2B$ must be a scaled-up version of this, i.e. the nodes are (2,2;2), ..., $(B, B; B), (B, 2B; 2B)$. Combining with the leaves, the worst tree with leaves (x,1;1) and job weights $\le 2B$ is the one with nodes (1,1;1), (2,2;2), ... $(B, B; B), (B, 2B; 2B)$, which is T_{p+1}. Thus the claim is true. □

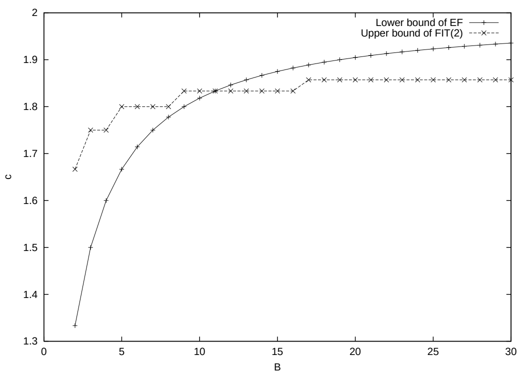

Figure 6: Comparing the competitive ratios of EF and FIT-2.

Lemma 8 *T_n has a charging ratio of $(2\lg B + 3)/(\lg B + 2)$ when $B = 2^n$.*

Proof. It is easy to see from Fig. 5(b) that the charging ratio

$$c = \frac{\sum_{k=1}^{n} 2^{k-1}(\frac{B}{2^k} \times 2) + (\frac{B}{2} + B)}{\sum_{k=1}^{n} 2^{k-1}(\frac{B}{2^k}) + B} = \frac{2\lg B + 3}{\lg B + 2}$$ □

Theorem 2 *FIT-2 gives a competitive ratio of $(2\lceil \lg B \rceil + 3)/(\lceil \lg B \rceil + 2)$, and the bound is tight.*

Proof. The competitive ratio of the scheduler is bounded by the charging ratio of T_n. When B is not an exact power of 2, we can replace B by the smallest power of 2 that is larger than B. Thus the result follows from Lemma 8. The bound is tight, as the worst-case charging tree T_n corresponds to the instance in Fig. 3 by putting $r = 2$. □

3.3 Comparisons

Note that FF remains 2-competitive even when B is very small, but EF might perform better when the job weights are bounded by the importance ratio. However we still have:

Lemma 9 *EF has a competitive ratio at least $\frac{2B}{B+1}$.*

Proof. Consider three jobs $q_1 = (0, 1, 1, 1), q_2 = (0, 2, 1, B)$ and $q_3 = (1, 2, 1, B)$. OPT schedules q_2 and q_3 giving value $2B$ while EF schedules q_1 and q_3 giving value $1 + B$. □

Fig. 6 shows a comparison between EF and FIT-2 for bounded B.

4 Concluding Remarks

In this paper we give a new online scheduling algorithm for the case where jobs can get partial value proportional to the amount processed, even if they are not completed. We showed that the new algorithm gives improved competitive ratio when the importance ratio of the jobs is bounded.

Note that a 1.8-competitive algorithm for this problem is given in [5]. However their model is different in that they use a *generalized schedule*, i.e. timesharing of tasks is allowed, so that several tasks can be processed concurrently at reduced speeds. This raises the problem of how much does timesharing helps in this scheduling problem. In fact this question is raised in [3]. Currently no non-timesharing algorithms are known to have competitive ratio $2 - \delta$ for some constant $\delta > 0$. In another paper [4] we showed that non-timesharing algorithms cannot have competitive ratio better than 1.618, and give a 1.58-competitive timesharing algorithm; thus timesharing and non-timesharing are different models with different competitive ratios.

References

[1] 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, *Real-Time Systems* 4, 125–144, 1992.

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

[3] E. Chang and C. Yap, Competitive Online Scheduling with Level of Service, *Proceedings of 7th Annual International Computing and Combinatorics Conference*, 453–462, 2001.

[4] F. Y. L. Chin and S. P. Y. Fung, Online Scheduling with Partial Job Values: Does Timesharing or Randomization Help?, manuscript, 2002.

[5] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý and N. Vakhania, Preemptive Scheduling in Overloaded Systems, to appear in *29th International Colloqium on Automata, Languages, and Programming*, 2002.

[6] G. Koren and D. Shasha, D^{over}: An Optimal Online Scheduling Algorithm for Overloaded Uniprocessor Real-time Systems, *SIAM Journal on Computing* 24, 318–339, 1995.

[7] J. Sgall, Online Scheduling, in *Online Algorithms: the State of the Art* (Fiat and Woeginger eds.), Springer-Verlag, 196–227, 1998.

[8] D. Sleator and R. Tarjan, Amortized Efficiency of List Update and Paging Rules, *Communications of the ACM* 28(2), 202–208, 1985.