

# Non-migratory Online Deadline Scheduling on Multiprocessors

Ho-Leung Chan<sup>\*†</sup>

Tak-Wah Lam<sup>\*†</sup>

Kar-Keung To<sup>\*</sup>

## Abstract

In this paper we consider multiprocessor scheduling with hard deadlines and investigate the cost of eliminating migration in the online setting. Let  $I$  be any set of jobs that can be completed by some migratory offline schedule on  $m$  processors. We show that  $I$  can also be completed by a non-migratory online schedule using  $m$  speed-5.828 processors (i.e., processors of 5.828 times faster). This result supplements the previous results that  $I$  can also be completed by a non-migratory offline schedule using  $6m$  unit-speed processors [8] or a migratory online schedule using  $m$  speed-2 processors [13]. Our result is based on a simple conservative scheduling algorithm called PARK which commits a processor to a job only when the processor has zero commitment before its deadline. A careful analysis of PARK further shows that the processor speed can be reduced arbitrarily close to 1 by exploiting more processors (say, using  $16m$  speed-1.8 processors). PARK also finds application in overloaded systems; it gives the first online non-migratory algorithm that can exploit moderately faster processors to match the performance of any migratory offline algorithm.

## 1 Introduction

In this paper we consider the online hard-deadline scheduling problem on multiprocessors; our focus is on eliminating migration among the processors. The hard-deadline scheduling problem is defined as follows: There are  $m$  identical processors. Given a set of jobs, each characterized by its release time, deadline, and processing time (work), the objective is to schedule all the jobs on the processors so that each job can be completed by its deadline. Note that each job can be scheduled on at most one processor at any time. Preemption is allowed and a preempted job can resume later from the point of preemption. If migration is allowed, a job can resume on a different processor. From a practical point of view, non-migratory schedules

are preferable since migration may incur significant overhead. Yet allowing migration simplifies the design of scheduling algorithms.

Let  $I$  be a set of jobs that can be completed by an offline schedule that allows migration. Kalyanasundaram and Pruhs [8] showed that migration is actually of limited power in offline scheduling; given a migratory offline schedule on  $m$  processors, it is always possible to construct a non-migratory offline schedule on  $6m - 5$  processors.<sup>1</sup> On the other hand, scheduling  $I$  online (i.e., the information about a job is known only when it is released) is very difficult even if migration is allowed. In fact,  $I$  does not admit any online schedule on  $m$  processors if  $m \geq 2$  [5].<sup>2</sup> Nevertheless, Phillips et al. showed that  $I$  can be completed by a migratory online schedule on  $m$  processors that are two times faster [13]. Migration seems to be a necessary tool in all online algorithms that are known to be able to complete  $I$  on  $m$  moderately faster processors (e.g., earliest deadline first, least laxity first, FR [13, 11]).

**Main result:** In this paper we devise a conservative online algorithm to show that  $I$  can be completed by a non-migratory online schedule on  $m$  speed-5.828 processors. This result supplements the result of Kalyanasundaram and Pruhs [8] as it illustrates that allowing migration in the online setting also gives limited power. A careful analysis of the new algorithm further shows that the speed requirement of 5.828 can also be reduced arbitrarily close to 1 by exploiting more processors (say, using  $16m$  speed-1.8 processors). More precisely, we show that for any  $\epsilon > 0$ ,  $I$  can be completed by a non-migratory online schedule on  $\lceil (1 + \frac{1}{\epsilon})^2 \rceil m$  speed- $(1+\epsilon)^2$  processors.

**Laxity assumption:** It is widely believed that jobs with very tight deadlines or very small laxity (i.e., deadline minus release time minus work) make online scheduling difficult. Our non-migratory online scheduling is no exception; in this paper we show

<sup>\*</sup>Department of Computer Science, University of Hong Kong.  
Email: {hlchan, twlam, kkto}@cs.hku.hk.

<sup>†</sup>Supported in part by Hong Kong RGC Grant HKU-7024/01E.

<sup>1</sup>The construction runs in pseudo-polynomial time, but can be improved to run in polynomial time if the processor bound is raised from  $6m - 5$  to  $12m - 5$ .

<sup>2</sup>When  $m = 1$ , the earliest deadline first (EDF) strategy guarantees to complete  $I$  [4].

that advance knowledge of the laxity of jobs in  $I$  can reduce the speed or processor requirement for a non-migratory online schedule. For example, if the work requested for every job in  $I$  is at most one eighth of its span (i.e., deadline minus release time), then we can have a non-migratory online schedule for  $I$  on  $m$  speed-2.5 processors or on  $4m$  unit-speed processors. Note that the latter result does not demand the presence of faster processors. In general, if the work-span ratio is at most  $w$ , then  $I$  can be completed by a non-migratory online schedule on  $m$  speed- $(4w + 2)$  processors or  $\lceil 2/(1 - 4w) \rceil m$  unit-speed processors. (The latter result holds only for  $w < 1/4$ .)

**PARK:** The core of our results is a simple conservative online scheduling algorithm called PARK; it does not commit a processor to any job unless the processor has zero commitment before its deadline. This algorithm often lets a job wait for a while before admitting it to a processor. The conservative nature of PARK avoids over crowding any processor and being tricked by the offline adversary. Like many other online algorithms, PARK is very simple but the analysis of its performance is relatively complicated. In this paper we present two different analyses of PARK, showing different dimensions of its performance.

**Firm-deadline scheduling:** PARK also finds applications in scheduling overloaded systems, in which there may be too many jobs to be completed and the deadlines are firm (instead of hard) in the sense that failing to complete a job by its deadline only loses the value due to that job and does not cause a system failure. Given a set  $I$  of such jobs, the objective of a scheduler is to maximize the value obtained from completing the jobs. For any  $c \geq 1$ , an online algorithm is said to be  $c$ -competitive if for any job sequence, it can obtain at least a fraction of  $1/c$  of the total value obtained by any offline schedule on  $m$  speed-1 processors. Consider the special case when the value of a job is proportional to its processing time. The work of Koren and Shasha [10] gave a non-migratory online algorithm on  $m$  speed-1 processors that is 3-competitive. If migration is allowed, Lam and To [12] showed that it is possible to devise an online algorithm on  $m$  speed-3 processors that is 1-competitive. Based on the latter result, we can actually extend PARK to give a non-migratory online algorithm on  $m$  speed-10 processors that is 1-competitive.

In the context of offline scheduling, the study of the power of migration is centered on the notion  $\omega_m$ , which is defined as the maximum over all input  $I$ , the ratio of the value attained by the optimal migratory offline schedule for  $I$  to the value attained by the optimal non-migratory offline schedule for  $I$  [9, 8]. For jobs with arbitrary values, the best upper bound of

$\omega_m$  is  $(6m - 5)/m$  [8]. This paper contributes to the knowledge of  $\omega_m$  when jobs are assumed to have a certain laxity. More precisely, our work implies that  $\omega_m \leq \min(6, \lceil \frac{2}{1-4w} \rceil)$  if the work-span ratio is at most  $w < 1/4$ .

**Organization of the paper:** Section 2 presents the new online algorithm PARK and discusses several interesting properties of PARK. Section 3 gives the first analysis of PARK, showing how to obtain a non-migratory online schedule on  $m$  speed-5.828 processors. As a by-product, we show how to improve the resource bounds when all jobs are assumed to have a certain laxity. Section 4 gives a slightly more complicated analysis of PARK, showing that the speed requirement of 5.828 can be reduced arbitrarily close to one when extra processors are used. Section 5 shows a simple lower bound result that non-migratory schedule is not feasible on  $m$  speed- $s$  processor if  $s < 2m/(m+1)$ . Also, the extension of PARK to firm-deadline scheduling is discussed.

**Notation:** Throughout this paper, we denote the release time, work (processing time), and deadline of a job  $J$  as  $r(J)$ ,  $p(J)$ , and  $d(J)$ , respectively. We also assume that jobs have distinct deadlines (ties are broken using the job IDs). The laxity and span of  $J$  is defined as  $d(J) - r(J) - p(J)$  and  $d(J) - p(J)$ , respectively. For any  $s \geq 1$ , a speed- $s$  processor refers to a processor that can process  $s$  units of work in one unit of time.

EDF refers to the strategy of scheduling jobs with earliest deadlines. Note that when a new job  $J$  is released, the current job  $J'$  on a processor will be preempted if  $J'$  has a deadline later than  $J$ 's as well as the deadlines of the current jobs on all other processors. When we say an algorithm completes a job sequence, we mean that the algorithm completes all jobs in the sequence within their respective deadlines.

## 2 The PARK algorithm

In this section, we describe a non-migratory online algorithm called PARK and discuss several interesting properties of PARK. We will show in the next section that for any sequence  $I$  of jobs that can be completed by a migratory offline schedule on  $m$  unit-speed processors,  $I$  can also be completed by PARK on  $m$  speed-5.828 processors.

PARK is a very conservative algorithm. Roughly speaking, whenever PARK admits a job  $J$  to a processor, it ensures that the processor has no commitment to other admitted jobs before the deadline of  $J$ . This concept of zero commitment is made formal through the following notion of *due*.

**DEFINITION 2.1.** Let  $t$  be the current time. Consider

any job  $J$ .

- Denote  $x_t(J)$  and  $p_t(J)$  as the amount of work done on  $J$  up to time  $t$  and the remaining work, respectively (note that  $p_t(J) = p(J) - x_t(J)$ ).
- Define the *Latest Processing Interval (LPI)* of  $J$  as the interval  $[d(J) - p_t(J), d(J)]$ . If  $d(J) - p_t(J) < t$ , we say that  $J$  *expires*.
- For any  $t' > t$ , define  $due_t(J, t')$  as the minimum amount of  $J$ 's remaining work a speed-1 processor needs to do by time  $t'$ , in order to complete  $J$  by its deadline. More formally,  $due_t(J, t') = \max\{0, t' - (d(J) - p_t(J))\}$  if  $t' \leq d(J)$ , and  $p_t(J)$  otherwise.

PARK is a non-migratory algorithm using  $pm$  speed- $s$  processors, where  $p$  and  $m$  are positive integers and  $s \geq 1$  is any real number. It maintains a central pool to store jobs that have been released but not yet admitted to any processor. PARK is non-migratory and each processor has its own queue of admitted jobs. Before describing the algorithm, we have one more definition.

**DEFINITION 2.2.** Let  $t$  be the current time. Consider any processor  $P_i$  used by PARK. For any  $t' > t$ , we define  $due_t(P_i, t')$  as the sum of  $due_t(J, t')$  over each job  $J$  in the queue of  $P_i$  at time  $t$ .

In the rest of the paper, we say that at time  $t$ , (i) a job  $J$  has  $w$  units of work due at time  $t' > t$  if  $due_t(J, t') = w$ ; (ii) a processor  $P_i$  has  $w$  units of work due at  $t'$  if  $due_t(P_i, t') = w$ ; (iii) a processor  $P_i$  is doing some work due at  $t'$  if  $P_i$  is processing a job  $J$  with  $due_t(J, t') > 0$ .

---

### Algorithm 1 PARK

---

**Job Release:** A job upon release is put into the pool if it cannot be admitted to one of the processors.

**Job Admission:** At any time  $t$ , let  $S$  be the set of jobs in the pool plus possibly the job just released at  $t$ . If  $S$  is non-empty, we attempt to admit such jobs as follows: Let  $J \in S$  be the job with the earliest deadline.

- If  $J$  expires, discard  $J$ .
- Else if there exists a processor  $P_i$  such that  $due_t(P_i, d(J)) = 0$ , admit  $J$  into the queue of  $P_i$ .

**Individual Processor Scheduling:** Each processor schedules the jobs in its queue using EDF.

---

It is worth-mentioning that though PARK is using speed- $s$  processors, the definitions of  $due_t(P_i, t')$  as well as LPI are based on a unit-speed processor. To understand the admission policy of PARK, we need to focus on the LPIs of the jobs. In general, the LPIs of

jobs may overlap with each other. For example, if two jobs have the same deadline, their LPIs always share a common right end. Yet the way PARK admits jobs aims to guarantee that at any time, all jobs admitted to the same processor have non-overlapping LPIs. This can be observed as follows. When a job  $J$  is admitted by a processor  $P_i$  at time  $t$ , all the previously admitted jobs have zero work due at  $d(J)$ . It means that at time  $t$ , the LPI of any previously admitted job starts no earlier than  $d(J)$  and cannot overlap the LPI of  $J$ . As time passes, the LPI of each individual job might shrink (if  $P_i$  has worked on it) but cannot get bigger; thus, the non-overlapping property remains. The following is a precise statement about the non-overlapping property. Figure 1 shows the LPIs of the jobs at different times when PARK is given a sequence of five jobs.

**PROPERTY 2.1. (NON-OVERLAPPING PROPERTY)** Let  $t$  be the current time. Consider any processor  $P_i$ . Let  $J_u$  and  $J_v$  be two jobs in the queue of  $P_i$ . The LPIs of  $J_u$  and  $J_v$  do not overlap.

The non-overlapping property allows us to bound the commitment of each processor easily.

**PROPERTY 2.2. (BOUNDED-COMMITMENT PROPERTY)** Let  $t$  be the current time. Consider any processor  $P_i$ .

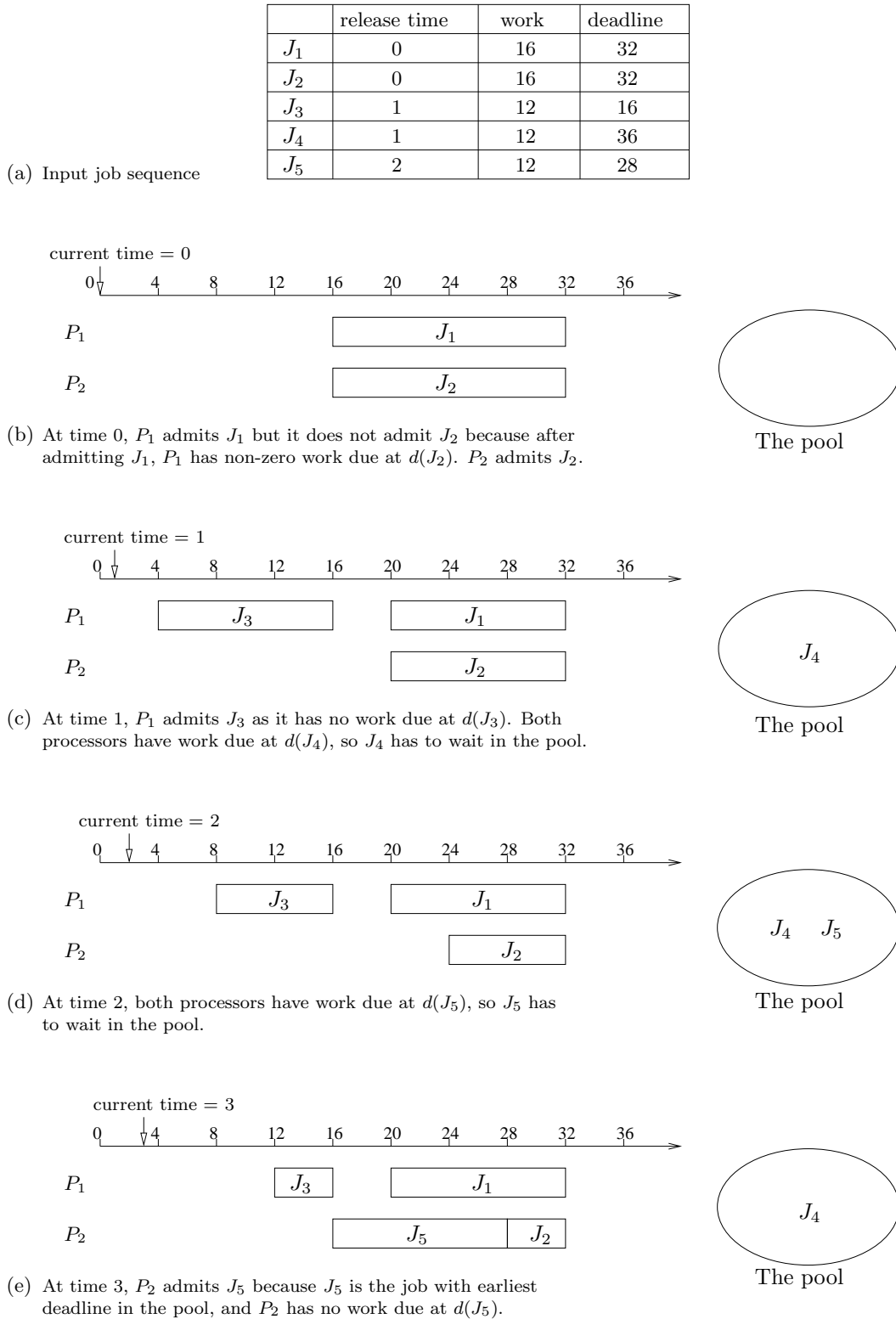
- For every job in the queue of  $P_i$ , its LPI starts no earlier than  $t$ . That is, no job in the queue of  $P_i$  expires.
- For any  $t' > t$ ,  $P_i$  has at most  $(t' - t)$  units of work due at time  $t'$ .

*Proof.* (a) For the sake of contradiction, assume that  $J$  is the first job that expires in the queue of a processor  $P_i$ . When  $J$  is admitted to  $P_i$ ,  $J$  has not yet expired. Let  $t_0$  be the moment just before the first time  $J$  is found to expire. That is, at  $t_0$ , the LPI of  $J$  starts exactly at  $t_0$ ; all other jobs in the queue of  $P_i$  have not yet expired and their LPIs, due to the non-overlapping property, must start after  $J$ 's LPI. In other words, all jobs except  $J$  have deadlines later than  $d(J)$ . Recall that  $P_i$  schedules jobs using EDF and its speed is  $s \geq 1$ . Thus,  $J$  must be processed by  $P_i$  at  $t_0$  and cannot expire immediately after  $t_0$ . A contradiction occurs.

(b) At any time  $t$ , consider the LPIs of all the jobs in queue of  $P_i$ . The LPIs are non-overlapping and by (a), the first LPI starts no earlier than  $t$ . For any  $t' > t$ , the amount of work due at  $t'$  for  $P_i$  is equal to the total length of each LPI (or its portion) that ends on or before  $t'$ . The latter is obviously upper bounded by  $t' - t$ .  $\square$

Since a job admitted by a processor  $P_i$  never expires, it can be completed by  $P_i$  by its deadline. In

Figure 1: Figure (a) shows a sequence of five jobs to be scheduled by PARK on two speed-4 processors. Figures (b)-(e) show the LPIs (not the schedules) of the jobs in each processor at different times.



the next section, we will show that for a sequence of jobs that can be completed by some offline schedule, PARK will admit every job to a processor before it expires. PARK seems to be conservative in admitting jobs and often let jobs wait in the pool. The next property shows that whenever a job is waiting, all processors are actually productive. In other words, such waiting is reasonable.

**PROPERTY 2.3. (WAITING PROPERTY)** At any time  $t$ , if there is a job  $J$  left in the pool, then,

- all processors are busy; and
- all processors are doing some work due at  $d(J)$ .

*Proof.* Suppose on the contrary that at time  $t$ , there exists a processor  $P_i$  being idle. Then  $P_i$  should admit  $J$  or another job from the pool and become busy at  $t$ . A contradiction occurs. If  $P_i$  at  $t$  is working on a job  $J_e$  that has no work due at  $d(J)$ , then the LPI of  $J_e$  starts no earlier than  $d(J)$ .  $P_i$  is using EDF and  $J_e$  has the earliest deadline among all jobs in  $P_i$ . By the non-overlapping property, at  $t$ , every job in  $P_i$  has no work due at  $d(J)$ , and  $P_i$  should admit  $J$  or another job with deadline earlier than  $d(J)$  from the pool. This contradicts that at  $t$ ,  $P_i$  is working on  $J_e$ .  $\square$

### 3 Analysis of PARK

In this section, we prove the main result that any job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors can also be completed by PARK on  $m$  speed- $(3 + 2\sqrt{2})$  processors. Note that  $3 + 2\sqrt{2} \approx 5.828$ . We also show how to improve the resource bounds when jobs are assumed to have a certain laxity. To ease our discussion, this section will first present a lemma for the special case where every job has non-zero laxity, or equivalently, a work-span ratio strictly less than one. This is to illustrate the core idea of our analysis of PARK. Then we make use of a scaling technique to prove the general theorem, followed by two corollaries that capture the results stated earlier.

**LEMMA 3.1.** *Let  $I$  be any job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors. Let  $0 < w < 1$ . If all jobs in  $I$  have a work-span ratio at most  $w$ , then PARK can complete  $I$  on  $m$  speed- $\frac{2}{1-w}$  processors.*

*Proof.* Note that the speed of the processors used by PARK is  $2/(1-w) > 1$ . As mentioned in the previous section, every job admitted by PARK to a processor can be completed by its deadline. To prove Lemma 3.1, it suffices to show that every job in  $I$  gets admitted rather than discarded in the admission step of PARK.

For the sake of contradiction, we assume that in the course of scheduling  $I$  with PARK, some job expires in the pool and is discarded. Let  $J$  be the first such job. Let  $t_0 = d(J) - w(d(J) - r(J)) = r(J) + (1-w)(d(J) - r(J))$ . Since  $J$  has a work-span ratio at most  $w$ ,  $J$  cannot expire on or before  $t_0$  and  $J$  is in the pool during the time interval  $[r(J), t_0]$ .

Let  $l \geq 0$  be the biggest number such that at any time throughout the interval  $[r(J) - l, t_0]$ , there is at least one job in the pool with deadline on or before  $d(J)$ . By the waiting property, during  $[r(J) - l, t_0]$ , all the  $m$  processors of PARK are busy and the total work done by PARK is exactly

$$m \times \frac{2}{1-w} \times (t_0 - (r(J) - l)) = 2m(d(J) - r(J)) + \frac{2lm}{1-w}.$$

By the waiting property again, at any time in  $[r(J) - l, t_0]$ , PARK schedules all processors to do some work due at  $d(J)$ . Such work can be classified into two types:

1. Work owing to jobs that are admitted to processors before  $r(J) - l$ .
2. Work owing to jobs admitted during  $[r(J) - l, t_0]$ .

By the bounded-commitment property, at time  $r(J) - l$ , each processor has at most  $d(J) - (r(J) - l)$  units of work due at  $d(J)$ , and hence the amount of type 1 work is at most  $m \times (d(J) - (r(J) - l))$ . Consider any job  $J'$  admitted during  $[r(J) - l, t_0]$ .  $J'$  has a deadline no later than  $d(J)$  and by definition of  $l$ ,  $J'$  must be released no earlier than  $r(J) - l$ . Note that the total work of jobs with release time at least  $r(J) - l$  and deadline at most  $d(J)$  cannot exceed  $m(d(J) - (r(J) - l))$ ; otherwise such jobs cannot be completed by any offline schedule on  $m$  unit-speed processors. On the other hand,  $J$  is one of such jobs but does not get admitted by PARK. The amount of type 2 work is at most  $m \times (d(J) - (r(J) - l)) - p(J)$ . In summary, the total amount of work done by PARK during  $[r(J) - l, t_0]$  is at most

$$\begin{aligned} & m(d(J) - (r(J) - l)) + m(d(J) - (r(J) - l)) - p(J) \\ & < 2m(d(J) - r(J)) + 2lm \\ & \leq 2m(d(J) - r(J)) + \frac{2lm}{1-w}. \end{aligned}$$

This leads to a contradiction and Lemma 3.1 follows.  $\square$

In the following, we present an extension to PARK so as to remove the assumption that the work-span ratio must be less than one. This extension also allows a tradeoff between the processor speed and the number of processors used by PARK.

$\text{PARK}(u)$  is a scaled version of  $\text{PARK}$ , characterized by a real number  $u > 0$ . Intuitively,  $\text{PARK}(u)$  scales every job by a factor of  $u$  and follow the schedules of  $\text{PARK}$  for the scaled jobs. When  $u = 1$ ,  $\text{PARK}(u)$  is identical to  $\text{PARK}$ . More specifically, to schedule a job sequence  $I$  on  $n$  speed- $s'$  processors,  $\text{PARK}(u)$  simulates a copy of  $\text{PARK}$  that uses  $n$  speed- $s$  processors, where  $s = us'$ . Whenever  $\text{PARK}(u)$  receives a new job  $J$ , it releases a job  $J'$  for  $\text{PARK}$  with  $r(J') = r(J)$ ,  $d(J') = d(J)$  and  $p(J') = u \times p(J)$ . Denote the processors used by  $\text{PARK}(u)$  as  $P_1, \dots, P_n$  and those used by  $\text{PARK}$  as  $P'_1, \dots, P'_n$ . At any time,  $\text{PARK}(u)$  admits a job  $J$  to a processor  $P_i$  if  $\text{PARK}$  admits the corresponding job  $J'$  to  $P'_i$ ;  $\text{PARK}(u)$  discards  $J$  if  $\text{PARK}$  discards  $J'$ ; and  $\text{PARK}(u)$  runs a job  $J$  on a processor  $P_i$  if  $\text{PARK}$  runs  $J'$  on  $P'_i$ .

We notice that  $\text{PARK}(u)$  can always synchronize with the simulated copy of  $\text{PARK}$ , because the amount of time for  $\text{PARK}(u)$  to complete a job  $J$  is exactly the same as that for  $\text{PARK}$  to complete the corresponding job  $J'$ . The following is the main theorem of this section.

**THEOREM 3.1.** *Let  $I$  be any job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors. Let  $0 < w \leq 1$ . If all jobs in  $I$  have a work-span ratio at most  $w$ , then  $\text{PARK}(u)$  can complete  $I$  using  $pm$  speed- $\frac{p+u}{pu(1-wu)}$  processors, where  $p$  is any positive integer and  $u$  is any real number such that  $0 < wu < 1$ .*

Before proving Theorem 3.1, we illustrate how to choose the parameters in Theorem 3.1 so as to obtain the results claimed in the introduction. Consider the case where  $w = p = 1$ . Choosing  $u = \sqrt{2} - 1$  would minimize the speed requirement and gives the following corollary.

**COROLLARY 3.1.** *Let  $I$  be any job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors.  $\text{PARK}(u)$  with  $u = \sqrt{2} - 1$  can complete  $I$  using  $m$  speed- $(3 + 2\sqrt{2})$  processors.*

Putting  $u = 1/(2w)$  gives a more general corollary.

**COROLLARY 3.2.** *Let  $I$  be any job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors. Let  $0 < w \leq 1$ . If all jobs in  $I$  have a work-span ratio at most  $w$ , then  $\text{PARK}(u)$  with  $u = \frac{1}{2w}$  can complete  $I$  using (i)  $pm$  speed- $(4w + 2/p)$  processors for any positive integer  $p$ , or (ii)  $\lceil 2/(s - 4w) \rceil m$  speed- $s$  processors for any  $s > 4w$ .*

*Proof of Theorem 3.1.* Recall that  $\text{PARK}(u)$  uses  $pm$  speed- $\frac{p+u}{pu(1-wu)}$  processors. To schedule a job sequence

$I$  as stated in the theorem,  $\text{PARK}(u)$  simulates a copy of  $\text{PARK}$  that uses  $pm$  speed- $\frac{p+u}{p(1-wu)}$  processors. Let  $I'$  be the sequence of the jobs created for  $\text{PARK}$  while  $\text{PARK}(u)$  schedules  $I$ . As  $I$  can be completed by some offline schedule on  $m$  speed-1 processors;  $I'$  can also be completed by some offline schedule on  $m$  speed- $u$  processors. Each job in  $I'$  has a work-span ratio at most  $wu$ . To show that  $\text{PARK}(u)$  can complete  $I$ , it suffices to show that  $\text{PARK}$  meets the deadlines of all jobs in  $I'$ . The proof is a straightforward generalization of Lemma 3.1.

First, note that  $\frac{p+u}{p(1-wu)} > 1$  as both  $u > 0$  and  $1 > wu > 0$ . Suppose, for the sake of contradiction, that  $\text{PARK}$  fails to complete a job in  $I'$ . This job must expire in the pool. Let  $J$  be the first such job. Let  $t_0 = r(J) + (1 - wu)(d(J) - r(J))$ . As the work-span ratio of  $J$  is at most  $wu$ ,  $J$  expires no earlier than  $t_0$  and must have resided in the pool during the interval  $[r(J), t_0]$ . Again, let  $l \geq 0$  be the largest number such that at any time throughout the interval  $[r(J) - l, t_0]$ , there is at least one job in  $\text{PARK}$ 's pool with deadline on or before  $d(J)$ . Note that  $l \geq 0$ . During  $[r(J) - l, t_0]$ , all  $m$  processors are busy and the total work done by  $\text{PARK}$  is exactly

$$\begin{aligned} & pm \times s \times (t_0 - (r(J) - l)) \\ &= (p + u)m(d(J) - r(J)) + \frac{(p + u)lm}{1 - wu} . \end{aligned}$$

Furthermore, at any time in  $[r(J) - l, t_0]$ , every processor of  $\text{PARK}$  is always doing work due at  $d(J)$  and such work belongs to either a job admitted before  $r(J) - l$  or during  $[r(J) - l, t_0]$ . Thus, the total work done by  $\text{PARK}$  during  $[r(J) - l, t_0]$  is at most

$$\begin{aligned} & pm(d(J) - (r(J) - l)) + mu(d(J) - (r(J) - l)) - p(J) \\ &< (p + u)m(d(J) - r(J)) + l(p + u)m \\ &\leq (p + u)m(d(J) - r(J)) + \frac{(p + u)lm}{1 - wu} . \end{aligned}$$

This leads to a contradiction and  $\text{PARK}$  must be able to complete  $I'$ . Hence, the theorem follows.  $\square$

#### 4 Alternative analysis of $\text{PARK}$

In this section, we show a more complicated analysis of  $\text{PARK}$ , resulting in the following theorem, which enables us to construct a non-migratory online algorithm for hard deadline scheduling that can exploit extra processors to reduce the speed requirement arbitrarily close to one (see Corollary 4.1).

**THEOREM 4.1.** *Let  $0 < w < 1$  be a real number. Let  $I$  be a job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors. If*

all jobs in  $I$  have a work-span ratio at most  $w$ , then  $I$  can be completed by PARK using  $pm$  speed- $s$  processors, where  $p$  is any positive integer such that  $p(1-w) > 1$  and  $s = 1 + \frac{1}{p(1-w)-1}$ .

Using the scaling technique, we can remove the assumption of work-span ratios and extend Theorem 4.1 to any job sequence  $I$ . Recall that  $\text{PARK}(u)$  scales every job of  $I$  by a factor of  $u$  and schedules the scaled jobs using PARK. Let  $u = \frac{1}{1+\epsilon}$  for some  $\epsilon > 0$ , and denote the scaled version of  $I$  as  $I'$ . Every job in  $I'$  has a work-span ratio at most  $\frac{1}{1+\epsilon}$ . By Theorem 4.1,  $I'$  can be completed by PARK using  $pm$  speed- $s$  processors where  $s = 1 + \frac{1}{p(1-\frac{1}{1+\epsilon})-1}$ .  $\text{PARK}(u)$ , using  $pm$  speed- $s/u$  processors, can synchronize with PARK and complete  $I$ . In particular, choosing  $p = \lceil (1 + \frac{1}{\epsilon})^2 \rceil$ , we have  $s/u = \left(1 + \frac{1}{p(1-\frac{1}{1+\epsilon})-1}\right) / u \leq (1+\epsilon)^2$ . This relationship is stated in the following corollary. Note that choosing a small  $\epsilon$  means using more and slower processors.

**COROLLARY 4.1.** *Let  $I$  be a job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors. Let  $\epsilon > 0$  be any real number.  $\text{PARK}(u)$  with  $u = \frac{1}{1+\epsilon}$  can complete  $I$  using  $\lceil (1 + \frac{1}{\epsilon})^2 \rceil \times m$  speed- $(1+\epsilon)^2$  processors.*

We are ready to prove Theorem 4.1. In the rest of this section, we assume that  $I$  is a job sequence that can be completed by some migratory offline schedule OPT on  $m$  speed-1 processors and all jobs in  $I$  have a work-span ratio at most  $w < 1$ . PARK is using  $pm$  speed- $s$  processors, where  $p$  is a positive integer such that  $p(1-w) > 1$  and  $s = 1 + \frac{1}{p(1-w)-1}$ .

To prove Theorem 4.1, we need to compare the total amount of work due at any particular time in PARK and in OPT. At any time  $t$ , for any  $t' > t$ , we let  $C(t, t')$  be the total amount of work due at  $t'$  in PARK, and similarly  $O(t, t')$  for OPT. Let  $L(t, t') = C(t, t') - O(t, t')$ . If  $L(t, t') = w$ , we say that at time  $t$ , PARK lags behind OPT by  $w$  units of work due at  $t'$ . Furthermore, PARK is said to be *safe* at time  $t$  if for any  $t' > t$ ,  $L(t, t')$  is proportional to the duration from  $t$  to  $t'$  (see the following definition).

**DEFINITION 4.1.** In the course of scheduling  $I$ , at any time  $t$ , for any  $t' > t$ , PARK is  $t'$ -safe if  $L(t, t') < \frac{m}{s-1}(t' - t)$ . At any time  $t$ , PARK is *safe* if PARK is  $t'$ -safe for all  $t' > t$ .

The most non-trivial observation in analyzing PARK is that at any time, PARK is safe (see Lemma 4.1). It is then relatively easy to see that whenever a job  $J$  is released to PARK, if PARK is safe at

$r(J)$ , then  $J$  will be eventually admitted by PARK (see Lemma 4.2).

**LEMMA 4.1.** *In the course of scheduling  $I$ , at any time  $t$ , PARK is safe.*

**LEMMA 4.2.** *Let  $J$  be any job in  $I$ . If PARK is safe at  $r(J)$ , then  $J$  will be admitted by PARK on or before  $t_0 = r(J) + (1-w)(d(J) - r(J))$ .*

Lemma 4.1 and 4.2 together guarantee that every job in  $I$  must be admitted by PARK. As mentioned in Section 2, PARK meets the deadlines of all admitted jobs. Thus, Theorem 4.1 follows. To ease our discussion, we will first present the proof for Lemma 4.2. We start with a more technical lemma, which will also be used in the proof of Lemma 4.1.

**PROPOSITION 4.1.** *Consider any time interval  $[a, b]$  and any time  $d > b$ . Assume that PARK is  $d$ -safe at time  $a$  and at any time in  $[a, b]$ , there exists a job in PARK's pool with deadline no later than  $d$ . Then  $b - a < \frac{1}{p(s-1)}(d - a)$ . (Recall that  $s$  is chosen as  $1 + \frac{1}{p(1-w)-1}$ ; thus,  $\frac{1}{p(s-1)} = 1 - w - \frac{1}{p}$ .)*

*Proof.* Due to the waiting property, at any time in the time interval  $[a, b]$ , all the  $pm$  processors of PARK are busy and doing work due at  $d$ . The total work done by PARK during  $[a, b]$  is exactly

$$(4.1) \quad pm \times s \times (b - a) .$$

PARK is  $d$ -safe at  $a$  and  $L(a, d) < \frac{m}{s-1}(d - a)$ . At  $a$ , PARK and OPT has  $C(a, d)$  and  $O(a, d)$  units of work due at  $d$ , respectively. Consider all the jobs that are released from  $a$  to  $b$ ; the sum of their work due at  $d$  is at most  $m(d - a) - O(a, d)$  (otherwise, even OPT cannot complete these works by  $d$ ). During the interval  $[a, b]$ , the total amount of work due at  $d$  that PARK can possibly work on is at most

$$(4.2) \quad \begin{aligned} & C(a, d) + m(d - a) - O(a, d) \\ &= L(a, d) + m(d - a) \\ &< \frac{m}{s-1}(d - a) + m(d - a) \\ &= \frac{ms}{s-1}(d - a) . \end{aligned}$$

Combining Inequalities (1) and (2), we have  $pms(b - a) < \frac{ms}{s-1}(d - a)$ , or equivalently,  $(b - a) < \frac{1}{p(s-1)}(d - a)$ .  $\square$

Lemma 4.2 is in fact a corollary of Proposition 4.1. Details are as follows.

*Proof of Lemma 4.2.* Suppose on the contrary that at time  $t_0 = r(J) + (1-w)(d(J) - r(J))$ ,  $J$  has not been admitted. As  $J$  is assumed to have a work-span ratio at most  $w$ ,  $J$  has not yet expired at  $t_0$  and it is in the pool during the interval  $[r(J), t_0]$ . Note that  $t_0 - r(J) = (1-w)(d(J) - r(J))$ , and by Proposition 4.1 (with  $a = r(J)$ ,  $b = t_0$ , and  $d = d(J)$ ), we have  $(t_0 - r(J)) < (1-w - \frac{1}{p})(d(J) - r(J))$ . This implies that  $\frac{1}{p} < 0$ , contradicting that  $p \geq 1$ .  $\square$

The rest of this section is devoted to the proof of Lemma 4.1, which is further broken into two lemmas. We first state a simple fact about how the value of  $L(t, t')$  changes over time.

**FACT 4.1.** Let  $[t_1, t_2]$  be a time interval before a certain time  $t'$ . Assume that during the interval  $[t_1, t_2]$ , PARK has done at least  $x$  units of work due at  $t'$  and OPT has done at most  $y$  units of work due at  $t'$ . Then,  $L(t_2, t') \leq L(t_1, t') - x + y$ .

Intuitively, we prove Lemma 4.1 inductively over time. Assume that PARK is safe at time  $a$ . We first consider two basic types of time intervals  $[a, b]$  and show that in either case, PARK is safe at time  $b$ . Precisely, for any  $t' > b$ , we call  $[a, b]$

- a  $t'$ -quiet period if at any time in  $[a, b]$ , there is no job in PARK's pool with work due at  $t'$ ; and
- a  $t'$ -hectic period if at any time in  $[a, b]$ , there is at least one job in PARK's pool with work due at  $t'$ .

During a  $t'$ -quiet period, any job with work due at  $t'$  in PARK is admitted to some processor. Since PARK is using speed- $s$  processors, we can argue that PARK will not lag behind OPT too much on work due at  $t'$  during a  $t'$ -quiet period (see Lemma 4.3). A hectic period is more complicated. In Lemma 4.4 we first show that with the extra speed and number of processors given to PARK, a  $t'$ -hectic period cannot last for too long. Then we notice that within such a short period, the amount of work due at  $t'$  that PARK lags behind OPT cannot change too drastically and PARK is still  $t'$ -safe at  $b$ . Below we prove the above observations regarding quiet and hectic periods (see Lemmas 4.3 and 4.4). Then it is easy to show that PARK is safe at any time (i.e., Lemma 4.1).

**LEMMA 4.3.** *Consider a time interval  $[a, b]$  and a certain time  $t' > b$ . Assume that at any time in the interval  $[a, b]$ , there is no job in PARK's pool with work due at  $t'$ . If PARK is safe at  $a$ , then PARK is  $t'$ -safe at  $b$ .*

*Proof.* We prove inductively that at any time in  $[a, b]$ , PARK is  $t'$ -safe. Let  $r > a$  be the first time a job is

released; if no jobs are released on or before  $b$ , let  $r = b$ . Below we first show that PARK is  $t'$ -safe at time  $r$ , i.e.,  $L(r, t') \leq \frac{m}{s-1}(t' - r)$ . Note that at time  $r$ , if a job  $J$  is released, it contributes exactly  $p(J)$  to both  $C(r, t')$  and  $O(r, t')$  and does not affect the value of  $L(r, t')$ . To derive an upper bound of  $L(r, t')$ , it suffices to consider the amount of work released before  $r$ . In particular, we can tighten the trivial upper bound of  $L(r, t')$  from  $C(r, t')$  to  $\hat{C}(r, t')$ , where  $\hat{C}(r, t')$  denotes, at time  $r$ , the amount of work in PARK that has been released before  $r$  and due at  $t'$ .

Denote  $\Phi_r$  the set of PARK's processors which, at  $r$ , have work released before  $r$  and due at  $t'$ . Then  $\hat{C}(r, t') \leq |\Phi_r|(t' - r)$ . If  $|\Phi_r| < \frac{m}{s-1}$ , then  $L(r, t') \leq \hat{C}(r, t') < \frac{m}{s-1}(t' - r)$ . Bound  $L(r, t')$  for the case when  $|\Phi_r| \geq \frac{m}{s-1}$  is more complicated. During  $[a, r]$ , the pool contains no job due at  $t'$  and no job is released until  $r$ . Thus, at any time  $t$  where  $a \leq t < r$ , if a processor of PARK does not have any work due at  $t'$ , this processor cannot be in  $\Phi_r$ . In other words, throughout the interval  $[a, r]$ , every processor in  $\Phi_r$  has work due at  $t'$  and PARK has done at least  $|\Phi_r|s(r - a)$  units of work due at  $t'$ . Note that OPT achieves at most  $m(r - a)$ . Thus,

$$\begin{aligned} L(r, t') &\leq L(a, t') - |\Phi_r|s(r - a) + m(r - a) \\ &< \frac{m}{s-1}(t' - a) - \frac{m}{s-1}s(r - a) + m(r - a) \\ &\hspace{10em} \text{(At time } a, \text{ PARK is } t'\text{-safe.)} \\ &= \frac{m}{s-1}(t' - r) . \end{aligned}$$

In summary, we have proven that at time  $r$ , PARK is  $t'$ -safe. If  $r < b$ , we can repeat the above argument to prove that PARK is  $t'$ -safe at each subsequent release time and eventually at time  $b$ .  $\square$

Next, we consider the case of hectic periods.

**LEMMA 4.4.** *Consider a time interval  $[a, b]$  and a certain time  $t' > b$ . Assume that just before  $a$ , there is no job in PARK's pool with work due at  $t'$  and at any time in  $[a, b]$ , there is at least one job in PARK's pool with work due at  $t'$ . If PARK is safe at  $a$ , then*

- $(b - a) < \frac{1}{s}(t' - a)$ , and
- PARK is  $t'$ -safe at  $b$ .

*Proof. Statement (i):* Due to the condition of Lemma 4.4, we know that at any time in  $[a, b]$ , there is a job  $J$  in the pool with work due at  $t'$  and  $J$  must be released on or after  $a$ . Due to the work-span ratio assumption, any job released after  $a$  and with work due at  $t'$  must have deadline on or before  $a + \frac{1}{1-w}(t' - a) \geq t'$ . Let  $d = a + \frac{1}{1-w}(t' - a)$ . At time  $a$ , PARK is safe

and in particular,  $d$ -safe. By Proposition 4.1,  $(b-a) < (1-w-\frac{1}{p})(d-a) = (1-w-\frac{1}{p})(t'-a)/(1-w)$ . Note that  $s = 1 + \frac{1}{p(1-w)-1}$  and  $\frac{1}{s} = \frac{p(1-w)-1}{p(1-w)} = (1-w-\frac{1}{p})/(1-w)$ . Thus,  $(b-a) < \frac{1}{s}(t'-a)$  and Statement (i) follows.

*Statement (ii):* Consider the  $pm$  processors used by PARK. Let  $\Psi$  be the set of PARK's processors which, at any time in the interval  $[a, b]$ , are doing work due at  $t'$ . Let  $|\Psi| = \psi$ . If  $\psi \geq \frac{m}{s-1}$ , then,

$$\begin{aligned} L(b, t') &\leq L(a, t') - \psi s(b-a) + m(b-a) \\ &< \frac{m}{s-1}(t'-a) - \frac{m}{s-1}s(b-a) + m(b-a) \\ &= \frac{m}{s-1}(t'-b) \end{aligned}$$

Next, we consider  $\psi < \frac{m}{s-1}$ . Label the processors in  $\Psi$  as  $P_1, P_2, \dots, P_\psi$ . At  $a$ , each of these processors has at most  $t'-a$  units of work due at  $t'$ . Label the remaining processors as  $P_{\psi+1}, \dots, P_{pm}$ . At  $a$ , for each of such processor  $P_i$ , let  $w_i$  be the amount of work due at  $t'$ . Let  $W = \sum_{i=\psi+1}^{pm} w_i$ . Then  $L(a, t') \leq C(a, t') \leq \psi(t'-a) + W$ .

From  $a$  to  $b$ , each of  $P_1, P_2, \dots, P_\psi$ , has done exactly  $s(b-a)$  units of work due at  $t'$ . For each  $P_i$  where  $i = \psi+1, \dots, pm$ ,  $P_i$  at some point in  $[a, b]$  is doing some work due at a time later than  $t'$ ; thus  $P_i$  has done at least  $w_i$  units of work due at  $t'$ . In summary, during  $[a, b]$ , PARK, by time  $b$ , must have done at least  $\psi \times s(b-a) + W$  units of work due at  $t'$ ; note that OPT has done at most  $m(b-a)$  units of work due at  $t'$ . Hence, we have the following conclusion.

$$\begin{aligned} L(b, t') &\leq L(a, t') - (\psi s(b-a) + W) + m(b-a) \\ &\leq \psi(t'-a) + W - (\psi s(b-a) + W) \\ &\quad + m(b-a) \\ &= \psi(t'-a - s(b-a)) + m(b-a) \\ &< \frac{m}{s-1}(t'-a - s(b-a)) + m(b-a) \\ &\quad (\text{by Lemma 4.4(i), } (t'-a - s(b-a)) > 0) \\ &= \frac{m}{s-1}(t'-b) \end{aligned}$$

In summary, no matter what the value of  $\psi$  is,  $L(b, t') < \frac{m}{s-1}(t'-b)$ . Thus, PARK is  $t'$ -safe at time  $b$ .  $\square$

With the observations on the quiet and hectic periods, proving that PARK is safe at any time (i.e., Lemma 4.1) is straightforward.

*Proof of Lemma 4.1.* We first notice that at time 0,  $L(0, t')$  is equal to zero for any  $t' > 0$ . Thus, PARK is safe at time 0. Let  $\gamma_0 = 0$  and let  $\gamma_1 = \min\{\gamma > \gamma_0 \mid$

At time  $\gamma$ , PARK switches from a  $t'$ -quiet period to a  $t'$ -hectic period for some  $t' > \gamma$ . $\}$ . Consider any time  $t \leq \gamma_1$ . For any  $t' > t$ ,  $[\gamma_0, t]$  is a  $t'$ -quiet period, and by Lemma 4.3, PARK is  $t'$ -safe. Thus, PARK is safe at any time  $t \leq \gamma_1$ . We can repeat the above argument to show inductively that PARK is safe at any time. In general, let  $\gamma_{i+1} = \min\{\gamma > \gamma_i \mid \text{At time } \gamma, \text{ PARK switches from a } t'\text{-quiet period to a } t'\text{-hectic period for some } t' > \gamma, \text{ or vice versa}\}$ . Consider any time  $t \leq \gamma_{i+1}$ . For any  $t' > t$ , let  $j \leq i$  be the smallest integer such that  $[\gamma_j, t]$  is entirely a  $t'$ -quiet period or a  $t'$ -hectic period. By Lemma 4.3 or 4.4, PARK is  $t'$ -safe.

It is worth-mentioning that at any  $\gamma_i$ , a job is either released or admitted by PARK. Thus, in the course of scheduling  $I$ , there are only a finite number of  $\gamma_i$ 's. The above argument will complete eventually to show that PARK is safe at any time.  $\square$

## 5 Remarks

**Lower bound:** Consider the following job sequence:  $m+1$  identical jobs are released at time 0, each with  $m$  units of work and deadline  $m+1$ . The set of jobs can be completed by a migratory schedule on  $m$  speed-1 processors. For a non-migratory (online or offline) schedule to complete the jobs on  $m$  processors, some processor must admit at least two jobs, thus the speed requirement is at least  $2m/(m+1) = 2 - \frac{2}{m+1}$ .

**PARK plus EDF-AC:** Recall that in the firm deadline scheduling problem, there may be too many jobs to be completed and failing to complete a job only loses the value due to that job and does not cause a system failure. Given a set  $I$  of such jobs, the objective of a scheduler is to maximize the value obtained from completing the jobs. An online algorithm is said to be  $c$ -competitive for some  $c \geq 1$  if for any job sequence  $I$ , the algorithm can obtain at least a fraction of  $1/c$  of the value obtained by the optimal offline schedule on  $m$  speed-1 processors.

Consider the special case when the value of a job is proportional to its processing time. If migration is allowed, EDF-AC using  $m$  speed-3 processors is 1-competitive [12]. Since EDF-AC decides whether to complete or discard a job once the job is released, we can use it to select jobs for scheduling in PARK without migration. The actual operation is as follows. Whenever EDF-AC decides to complete a job  $J$ , we release  $J$  to PARK with  $p(J)$  scaled down to  $p(J)/3$ . The job sequence selected by EDF-AC can be completed by  $m$  speed-3 processors, so the scaled job sequence can be completed by  $m$  speed-1 processors and all jobs in the scaled sequence have work-span ratio at most  $1/3$ . By Corollary 3.2 with  $p = 1$  and  $w = 1/3$ , the scaled job sequence can be completed by PARK using  $m$  speed- $\frac{10}{3}$

processors. As any job in the scaled sequence has work only one third of the original, to complete the job actually selected by EDF-AC, we need to further increase the speed of the processors by a factor of 3. Thus, we have a 1-competitive algorithm for the firm deadline scheduling problem on  $m$  speed-10 processors.

**Effect of laxity on  $\omega_m$ :** Consider the offline scheduling problem. Recall that  $\omega_m$  is the maximum ratio, over all possible inputs, between the value attained by the optimal migratory schedule and that by the the optimal non-migratory schedule. The analysis of PARK reveals some information about the value of  $\omega_m$  when all jobs are assumed to have a certain amount of laxity. More precisely, if all jobs have work-span ratio no greater than  $w$ , where  $w < \frac{1}{4}$ , Corollary 3.2 shows that for any job sequence, the subset of jobs that can be completed by the optimal offline migratory schedule on  $m$  processors can also be completed by  $\lceil 2/(1-4w) \rceil m$  (unit-speed) processors. By selecting the  $m$  processors that achieve the highest values, we obtain a non-migratory offline schedule attaining a value of at least  $\frac{1}{\lceil 2/(1-4w) \rceil}$  of the value of the optimal migratory schedule. Hence,  $\omega_m \leq \lceil 2/(1-4w) \rceil$ .

**Implementation of PARK:** We notice that PARK admits a simple distributed implementation which does not require a centralized scheduler. Instead, each processor can monitor the pool and admit a job according to its own status, i.e., each processor does not need to inquire the status of other processors. This is different from many other scheduling algorithms (e.g., EDF, LLF) in which the status of all processors is needed in order to make a scheduling decision. Thus, PARK is particularly useful when it is difficult to obtain the complete information of all processors.

**Open Problems:** Let  $I$  be a job sequence that can be completed by some migratory offline schedule on  $m$  speed-1 processors. Consider the processor speed required to obtain a non-migratory online schedule for  $I$ . There is a gap between the upper bound of 5.828 and the lower bound of  $2 - \frac{2}{m+1}$ . The current analysis of PARK seems to be quite loose and we believe that a better analysis could possibly reduce the speed requirement to 4. We have shown that when extra processors are given, the speed requirement of PARK can be reduced arbitrarily close to 1. However, we do not know any (migratory or non-migratory) online algorithm that can guarantee to complete  $I$  using only  $f(m)$  speed-1 processors, where  $f(m)$  is a function of  $m$ . For the problem of firm deadline scheduling, the current analysis depends on EDF-AC as the job selection module. In fact, we conjecture that PARK alone (say, with  $m$  speed-9 processors) is sufficient to match the performance of any offline schedule.

## References

- [1] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proc. 1991 IEEE Real-Time Systems Symposium*, pages 101–110, 1991.
- [2] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. In *Proc. 6th SWAT*, pages 255–263, 1998.
- [3] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania. Preemptive scheduling in overloaded systems. In *Proc. ICALP*, pages 800–811, 2002.
- [4] M. L. Dertouzos. Control robotics: the procedural control of physical processes. In *Proc. IFIP Congress*, pages 807–813, 1974.
- [5] M. L. Dertouzos and A. K. L. Mok. Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks. *IEEE Transactions on Software Engineering*, 15(12): 1497–1506, 1989.
- [6] J. Edmonds. Scheduling in the dark. In *Proc. STOC*, pages 179–188, 1999.
- [7] B. Kalyanasundaram and K. R. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.
- [8] B. Kalyanasundaram and K. R. Pruhs. Eliminating migration in multi-processor scheduling. *Journal of Algorithms*, 38(1):2–24, 2001.
- [9] G. Koren, E. Dar, and A. Amir. The power of migration in multiprocessor scheduling of real-time systems. *SIAM Journal on Computing*, 30(2):511–527, 2000.
- [10] G. Koren and D. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theoretical Computer Science*, 128:75–97, 1994.
- [11] T. W. Lam and K. K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proc. SODA*, pages 623–632, 1999.
- [12] T. W. Lam and K. K. To. Performance Guarantee for Online Deadline Scheduling in the Presence of Overload. In *Proc. SODA*, pages 755–764, 2001.
- [13] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proc. STOC*, pages 140–149, 1997.
- [14] J. Sgall. On-line scheduling — a survey. In A. Fiat and G. Woeginger, editors, *On-line Algorithms: The State of the Art*, pages 196–231. Lecture Notes in Computer Science, Springer Verlag, 1998.
- [15] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*. Kluwer Academic Publishers, 1998.