

# Desynchronized Pfair Scheduling on Multiprocessors \*

UmaMaheswari C. Devi and James H. Anderson

Department of Computer Science, The University of North Carolina, Chapel Hill, NC

## Abstract

*Pfair scheduling, currently the only known way of optimally scheduling recurrent real-time tasks on multiprocessors, imposes certain requirements that may limit its practical implementation. In this paper, we address one such limitation — which requires processor time to always be allocated in units of fixed-sized quanta that are synchronized across processors — and determine the impact of relaxing it. We show that if this requirement, which may lead to wasted processor time, is relaxed, then under an otherwise-optimal Pfair scheduling algorithm deadlines are missed by at most one quantum only, which is sufficient to provide soft real-time guarantees. This result can be shown to extend to most prior work on Pfair scheduling: in general, tardiness bounds guaranteed by previously-proposed suboptimal Pfair algorithms are worsened by at most one quantum only.*

## 1. Introduction

A real-time system, unlike a non-real-time system, has to meet certain *timing constraints* to be correct. Such timing constraints are typically specified as deadline requirements. Tasks in a real-time system are often recurrent in nature. For example, in the well-studied periodic task model [9], each task  $T$  is characterized by two parameters, a worst-case execution time (WCET)  $T.e$ , and a period  $T.p$ : an instance or a *job* of  $T$  that requires up to  $T.e$  time to execute is released every  $T.p$  time units, and each such job must finish execution before the next job of  $T$  is released.

In work on real-time systems, multiprocessor designs are becoming increasingly common. This is due both to the advent of reasonably-priced multiprocessor platforms and to the prevalence of computationally-intensive real-time applications that have pushed beyond the capabilities of single-processor systems. Examples of such applications include systems that track people and machines, many computer-vision systems, and signal-processing applications. Given these observations, efficient scheduling

algorithms for multiprocessor real-time systems are of considerable value and interest.

In this paper, we consider the scheduling of recurrent real-time task systems on multiprocessor platforms comprised of  $M$  identical processors. Pfair scheduling, introduced by Baruah *et al.* [3], is the only known way of *optimally* scheduling such task systems on multiprocessors. The term “optimal” means that such algorithms are capable of scheduling on  $M$  processors any task system with total utilization at most  $M$ .

To ensure optimality, Pfair scheduling imposes certain requirements that may limit its practical implementation. One such limitation is the requirement that tasks be allocated processor time in fixed-sized quanta that align across all processors. It is known that if this requirement is not satisfied, then deadlines can be missed under an otherwise-optimal Pfair scheduling algorithm [7]. Fig. 2(b) gives an example, a detailed explanation of which is provided in Sec. 3. In this paper, we determine the impact of relaxing this limiting requirement.

We call the Pfair model that imposes the above restriction as the *synchronized and fixed-sized quantum* (SFQ) model. We consider this model to be limiting for the following reasons. First, it requires periodic timer interrupts that delineate quanta to be synchronized across all processors and drifts in the timing of interrupts on any one processor to be propagated to other processors as well. Second, because WCET estimates are generally pessimistic, many jobs will execute for less than their WCETs. When a job completes before the next quantum boundary, the rest of that quantum (on the associated processor) is wasted. Third, at the start of each quantum, the SFQ model idles all processors until scheduling decisions are made for all  $M$  processors. This idling can be reduced if the quanta are desynchronized and each processor scheduled independently. Fourth, synchronization protocols that have been proposed may cause a task to block on a lock request in the middle of a quantum [7]. Under the SFQ model, the remainder of the quantum is wasted. Finally, the SFQ model does not mesh well with general-purpose operating systems, which are characterized by event-driven scheduling, and hence, variable-sized quanta. Moreover, real-time applications deployed on such systems typically have only soft real-time

---

\* Work supported by NSF grants CCR 0204312, CCR 0309825, and CCR 0408996. The first author was also supported by an IBM Ph.D. fellowship.

requirements, and hence, bounded deadline misses may be tolerable.

We refer to the model of Pfair scheduling in which quanta may vary in size up to some maximum and need not align across processors as the *desynchronized and variable-sized quantum* (DVQ) model. This model is explained in detail in Sec. 3. In this paper, we show that under the DVQ model and an otherwise-optimal Pfair scheduling algorithm, deadlines are missed by at most the maximum size of one quantum only. The fact that deadlines are known to be missed under the DVQ model implies that our result is tight. Furthermore, this result can also be shown to extend to most prior results on Pfair scheduling.

Limitations of the SFQ model were first addressed by Chandra *et al.*, who proposed the Deadline Fair Scheduling (DFS) policy as a solution [5]. In addition to supporting the DVQ model (for a different task model than ours), DFS uses an auxiliary scheduler to allocate time that would otherwise go idle to ineligible, but runnable tasks. However, the work of Chandra *et al.* was entirely empirical, and real-time guarantees that can be provided were not derived. Though our task model is different from that of Chandra *et al.*, the tardiness bound that we provide can be used to derive service guarantees for a restricted version of their task model, as well. Also, the early-release model of Pfair scheduling [2] provides a less-expensive and simpler alternative to using an auxiliary scheduler.

In other related work, Holman and Anderson proposed the *staggered* model [8], which is a slight variant of the SFQ model. Their objective was to reduce bus contention that results due to the simultaneous scheduling of all processors necessitated by a strict alignment of quanta, on a symmetric shared-memory multiprocessor. They accomplished this by providing *fixed* offsets between the times at which quanta start on successive processors. In other words, quantum starting points are distributed on different processors uniformly over the interval of each quantum. All quanta are still restricted to be uniform in size, and the quanta on different processors are still synchronized, though not aligned.

The rest of this paper is organized as follows. Background on Pfair scheduling is provided in Sec. 2. The DVQ model and the main result of this paper are presented in Sec. 3. Sec. 4 concludes.

## 2. Background on Pfair Scheduling

This section provides a brief overview of Pfair scheduling [1, 2, 3, 10] under the SFQ model.

**System model.** A real-time system is modelled as a set of  $N$  recurrent tasks to be executed on  $M$  identical processors. Each task  $T$  is characterized by two parameters, its *worst-case execution cost*, denoted  $T.e$ , and its *period*  $T.p$ .  $T.p$  is also the relative deadline of each job of  $T$ . The ratio  $T.e/T.p$ , denoted  $wt(T)$ , is referred to as the *weight* of

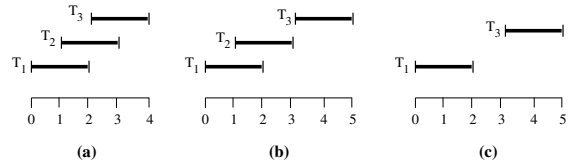


Figure 1: (a) Windows of the first job of a periodic task  $T$  with weight  $3/4$ . This job consists of subtasks  $T_1, T_2$ , and  $T_3$ , each of which must be scheduled within its window. (This pattern repeats for every job.) (b) The Pfair windows of an IS task. Subtask  $T_3$  becomes eligible one time unit late. (c) The Pfair windows of a GIS task. Subtask  $T_2$  is absent and subtask  $T_3$  becomes eligible one time unit late.

$T$ . It is required that  $0 < wt(T) \leq 1$ , *i.e.*,  $0 < T.e \leq T.p$  holds for all  $T$ .

**Scheduling model.** Pfair algorithms allocate processor time in discrete quanta that are uniform in size. The quantum size is the largest amount of time that a task is guaranteed execution without preemption. It is *required* that  $T.e$  and  $T.p$  both be expressed as integer multiples of the quantum size. Without loss of generality, it is assumed that a quantum is one time unit in duration, and hence, both  $T.e$  and  $T.p$  are required to be integers. The time interval  $[t, t + 1)$ , where  $t$  is a nonnegative integer, is called *slot*  $t$ . Time  $t$  refers to the beginning of slot  $t$  and is also called a *slot boundary*. Scheduling decisions are made at slot boundaries only. Hence, at most one task may execute on a given processor in any slot. A task may be allocated time on different processors, but not in the same slot, *i.e.*, interprocessor migration is allowed but parallelism is not. The sequence of allocation decisions over time slots defines a *schedule*  $\mathcal{S}$ . Formally,

$$\mathcal{S} : \tau \times \mathbb{N} \mapsto \{0, 1\}, \quad (1)$$

where  $\tau$  is a task set and  $\mathbb{N}$  is the set of nonnegative integers.  $\mathcal{S}(T, t) = 1$  iff  $T$  is scheduled in slot  $t$ . On  $M$  processors,  $\sum_{T \in \tau} \mathcal{S}(T, t) \leq M$  holds for all  $t$ .

To facilitate quantum-based scheduling, each task  $T$  is broken into a potentially infinite sequence of quantum-length *subtasks*,  $T_1, T_2, \dots$ . Each subtask  $T_i$  is associated with a *pseudo-release*  $r(T_i)$  and a *pseudo-deadline*  $d(T_i)$  as follows.

$$r(T_i) = \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \wedge d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil \quad (2)$$

The interval  $w(T_i) = [r(T_i), d(T_i))$  is termed the *window* of  $T_i$ . For example, in Fig. 1(a), for subtask  $T_1$ , we have  $r(T_1) = 0$ ,  $d(T_1) = 2$ , and  $w(T_1) = [0, 2)$ . For a Pfair schedule to be valid, each subtask must be scheduled within its window. This is sufficient to ensure that the deadlines of all subtasks, and hence all jobs, are met.

**Pfair scheduling algorithms.** Pfair scheduling algorithms choose at most  $M$  subtasks to execute on the  $M$  processors at the beginning of every slot. If a subtask does not execute for a full quantum, then the processor on which it

was scheduled remains idle until the next slot boundary. At present, three optimal Pfair scheduling algorithms, PF [3], PD [4], and PD<sup>2</sup> [2], and one suboptimal algorithm, earliest pseudo-deadline first (EPDF) [2], are known. In each of these algorithms, a subtask with an earlier deadline has higher priority than a subtask with a later deadline. For optimality, the optimal algorithms use additional rules to resolve ties among subtasks with the same deadline. PD<sup>2</sup> is the most efficient of the three and its tie-breaking rules subsume those of the other two algorithms; the suboptimal EPDF algorithm uses no tie-breaking rules.

**Task models.** Pfair scheduling may be used for scheduling *intra-sporadic* (IS) task systems and *generalized-intra-sporadic* (GIS) task systems [1, 10] also, in addition to periodic task systems. The IS and GIS task models provide a general notion of recurrent execution that subsumes that found in the periodic and sporadic task models. The *sporadic* model generalizes the periodic model by allowing jobs to be released “late”; the IS model generalizes the sporadic model by allowing subtasks to be released late, as shown in Fig. 1(b). More specifically, the separation between  $r(T_i)$  and  $r(T_{i+1})$  is allowed to exceed  $\lfloor i/wt(T) \rfloor - \lfloor (i-1)/wt(T) \rfloor$ , which would be the separation if  $T$  were periodic. Thus, an IS task is obtained by allowing a task’s windows to be right-shifted from where they would appear if the task were periodic.

Let  $\theta(T_i)$  denote the *offset* of subtask  $T_i$ , *i.e.*, the amount by which  $w(T_i)$  has been right-shifted. Then, by (2), we have the following.

$$r(T_i) = \theta(T_i) + \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \wedge d(T_i) = \theta(T_i) + \left\lfloor \frac{i}{wt(T)} \right\rfloor$$

The offsets are constrained so that the separation between any pair of subtask releases is at least the separation between those releases if the task were periodic. The offsets satisfy the following property:  $k > i \Rightarrow \theta(T_k) \geq \theta(T_i)$ .

Each IS subtask  $T_i$  has an additional parameter  $e(T_i)$ , which specifies the first time slot in which it is eligible to be scheduled. In particular, a subtask can become eligible before its “release” time. The following is required to hold:  $(\forall T_i :: e(T_i) \leq r(T_i) \wedge e(T_i) \leq e(T_{i+1}))$ . The intervals  $[r(T_i), d(T_i))$  and  $[e(T_i), d(T_i))$  are called the *PF-window* and *IS-window* of  $T_i$ , respectively.

**Generalized intra-sporadic task systems.** A *generalized* intra-sporadic task system is obtained by removing subtasks from a corresponding IS task system. Specifically, in a GIS task system, a task  $T$ , after releasing subtask  $T_i$ , may release subtask  $T_k$ , where  $k > i + 1$ , instead of  $T_{i+1}$ , with the following restriction:  $r(T_k) - r(T_i)$  is at least  $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$ . In other words,  $r(T_k)$  is not smaller than what it would have been if  $T_{i+1}, T_{i+2}, \dots, T_{k-1}$  were present and released as early as possible. Fig. 1(c) shows an example.

A schedule for a GIS task system  $\tau$  is *valid* or *correct* iff each subtask of  $\tau$  is scheduled in its IS-window. A valid schedule exists for a GIS task system  $\tau$  on  $M$  processors iff its total utilization is at most  $M$ , *i.e.*,  $\sum_{T \in \tau} wt(T) \leq M$  holds [1]. A task system with total utilization at most  $M$  is said to be *feasible* on  $M$  processors.

**Soft real-time systems and tardiness.** In a soft real-time system, subtasks (or jobs) may occasionally miss their deadlines, if the amount by which a subtask misses its deadline, referred to as its *tardiness*, is bounded. Formally, the tardiness of a subtask  $T_i$  in schedule  $\mathcal{S}$  is defined as

$$tardiness(T_i, \mathcal{S}) = \max(0, t - d(T_i)), \quad (3)$$

where  $t$  is the time at which  $T_i$  completes executing in  $\mathcal{S}$ . The tardiness of a task system  $\tau$  under scheduling algorithm  $\mathcal{A}$ , denoted  $tardiness(\tau, \mathcal{A})$ , is defined as the maximum tardiness of any subtask in  $\tau$  under any schedule under  $\mathcal{A}$ . If  $\kappa$  is the maximum tardiness of any task system under  $\mathcal{A}$ , then  $\mathcal{A}$  is said to *ensure a tardiness bound* of  $\kappa$ .

### 3. Pfair in the DVQ Model

We now describe the DVQ model in detail and show that under this model and the PD<sup>2</sup> scheduling algorithm, tasks may miss their deadlines by at most one quantum only.

**The DVQ model.** The DVQ model differs from the SFQ model in when subtasks are allocated processor time and for how long; the task model is unaltered. The SFQ model requires all quanta to be of uniform size and align across all processors. This tight synchrony is maintained by forcing the scheduler to be non-work-conserving in that, if a subtask does not execute for the duration of an entire quantum, then part of the quantum allocated to it is unused. The DVQ model is a work-conserving variant that reclaims this wastage by allowing the quanta to vary in size in the range  $(0, 1]$  and by not requiring the quanta on different processors to be synchronized. In particular, if a task yields before executing for a full quantum, then a new quantum begins on the associated processor immediately. However, in this model, the subtask that yields without executing for a full quantum cannot reclaim the unused time later. Fig. 2 illustrates the difference between the SFQ and DVQ models with an example. In this example, tasks  $A$ ,  $B$ , and  $C$  of weight  $1/6$  each, and tasks  $D$ ,  $E$ , and  $F$  of weight  $1/2$  each, with a total utilization two are scheduled on two processors under PD<sup>2</sup>. Subtasks  $A_1$  and  $F_1$  scheduled at  $t = 1$  execute for an interval  $1 - \delta$  only and yield their respective processors at  $t = 2 - \delta$ . Both processors idle until  $t = 2$  to start the next quantum in the SFQ model, whereas a new quantum begins immediately in the DVQ model. The two processors are assigned to the only ready subtasks  $B_1$  and  $C_1$  at  $t = 2 - \delta$  under the DVQ model.

As Fig. 2 shows, scheduling decisions in the DVQ model may be made at non-integral times. Thus, the function in (1) is not adequate to fully define a schedule. Hence,

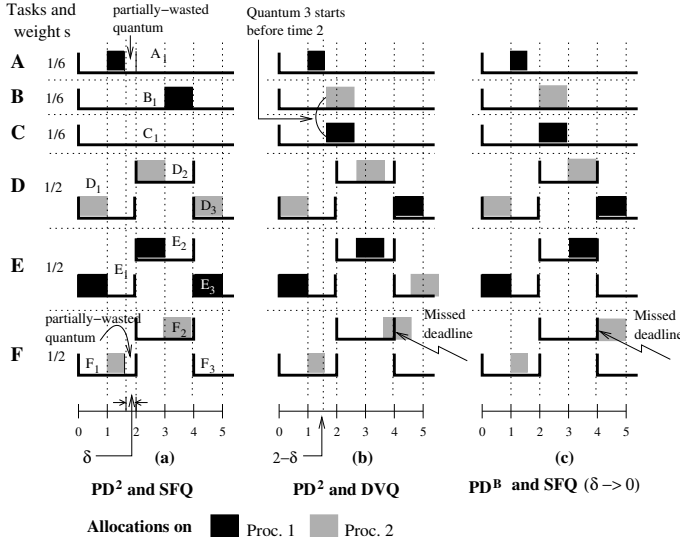


Figure 2: Difference between (a) the SFQ model and (b) the DVQ model under  $PD^2$ . (c) A possible schedule under  $PD^B$  in the SFQ model, obtained by postponing the allocations in (b) that do not commence on a slot boundary to the next slot boundary.

we overload this function to denote the time at which a *subtask* commences execution. Formally, if  $\mathcal{S}$  is a schedule for a task set  $\tau$ , then  $\mathcal{S} : \{\text{subtasks in } \tau\} \mapsto \mathcal{Q}$ , where  $\mathcal{Q}$  is the set of all rational numbers. We also associate with each subtask  $T_i$  its *actual* execution cost, denoted  $c(T_i)$ . It is required that  $c(T_i) \leq 1$  hold. (The worst-case execution cost of each subtask is 1.) In the example in Fig. 2(b),  $\mathcal{S}(B_1) = \mathcal{S}(C_1) = 2 - \delta$  and  $c(A_1) = c(F_1) = 1 - \delta$ .

In the discussion that follows, we refer to subtasks as being “scheduled at” or “executing at” some time  $t$ , where  $t$  need not be integral. When we say that  $T_i$  is *scheduled at*  $t$ , we mean that  $T_i$  commences execution at  $t$ , i.e.,  $\mathcal{S}(T_i) = t$  holds. On the other hand, if we say that  $T_i$  is *executing at*  $t$ , then we mean that  $t - 1 < \mathcal{S}(T_i) \leq t$  holds. We say that a subtask  $T_i$  is *ready* at time  $t$ , if (i)  $e(T_i) \leq t$  holds, (ii)  $T_i$  has not been scheduled before  $t$ , and (iii)  $T_i$ ’s predecessor, if any, completes execution at or before  $t$ . As noted earlier, the task model remains the same under the DVQ model. Therefore, the release time, eligibility time, and deadline of each subtask are the same as their corresponding values under the SFQ model, and hence, remain integral. Also, the notion of a *slot* is unchanged: the term “slot  $t$ ,” where  $t$  is an integer, still refers to the interval  $[t, t + 1)$ .

### Establishing bounded tardiness under the DVQ model.

In the process of making the scheduler work-conserving, the DVQ model also introduces “priority inversions,” which can lead to deadline misses. A *priority inversion* occurs if a lower-priority subtask executes, while a ready, higher-priority subtask waits. In such cases, the waiting higher-priority subtask is said to be *blocked*.

Fig. 2(b) shows that deadline misses are possible under  $PD^2$  in the DVQ model. Our goal is to show that such

misses are at most the maximum size of one quantum only. To avoid reasoning directly in the DVQ model, which can be quite cumbersome, and to leverage the analysis techniques and results presented previously for the SFQ model, we establish the tardiness bound for the DVQ model in the following four steps. (i) We consider allocations in the DVQ model when subtasks execute for a duration of  $1 - \delta$  in the limit  $\delta \rightarrow 0$ , and thus reduce them to allocations that conform to the SFQ model. For example, in the limit  $\delta \rightarrow 0$ , the allocations in Fig. 2(b) reduce to those in Fig. 2(c). (ii) We then identify a scheduling algorithm that makes the corresponding scheduling decisions in the SFQ model. We will denote this algorithm  $PD^B$  (the ‘B’ stands for *blocking*). (iii) Next, we show that the tardiness of  $PD^2$  in the DVQ model is bounded by the tardiness of  $PD^B$  in the SFQ model. (iv) Finally, we show that  $PD^B$  ensures a tardiness bound of at most one quantum in the SFQ model, which in turn establishes a bound for  $PD^2$  in the DVQ model. Steps (ii)–(iv) are elaborated upon in the subsections that follow. In the rest of this paper, unless otherwise mentioned, all references to  $PD^B$  are with respect to the SFQ model only. Also, for brevity, we refer to  $PD^2$  invoked under the DVQ model as  $PD^2$ -DVQ; invocations under the SFQ model shall simply be referred to as  $PD^2$ .

### 3.1. Worst-case Scenario for $PD^2$ -DVQ

In this subsection, we devise algorithm  $PD^B$ , which represents a worst case for  $PD^2$ -DVQ, as far as subtask tardiness is concerned. Before presenting the algorithm, we explain the priority inversions that are possible in  $PD^2$ -DVQ in detail. In the rest of the paper, we use the precedence operators  $\prec, \preceq, \succ,$  and  $\succeq$  to indicate the relative priority between two subtasks with respect to  $PD^2$ , as follows: If  $T_i$  and  $U_j$  are any two subtasks, then  $T_i \prec U_j$  ( $T_i \succ U_j$ ) denotes that the priority of  $T_i$  is strictly greater (less) than that of  $U_j$ ;  $\preceq$  ( $\succeq$ ) will be used if they may be equal. To indicate the relative priority by  $PD^B$  we use symbols  $\square, \sqsubseteq, \sqsupset,$  and  $\sqsupseteq$ .

One type of priority inversion possible under  $PD^2$ -DVQ is exemplified in Fig. 2(b). Here, allowing a new quantum to begin at time  $2 - \delta$  on both the processors leads to  $B_1$  and  $C_1$  being scheduled at time  $2 - \delta$ . Because  $B_1$  and  $C_1$  execute for an entire quantum, no processor is available at time 2, when subtasks  $D_2, E_2,$  and  $F_2$  become eligible. (Note that  $D_2, E_2,$  and  $F_2$  have an earlier deadline, and hence, a higher priority than  $B_1$  and  $C_1$ .  $B_1$  and  $C_1$  have deadlines at time 6.) Therefore, at time 2,  $D_2, E_2,$  and  $F_2$  are blocked by subtasks  $B_1$  and  $C_1$ . Their blocking time would be maximized, if subtasks  $A_1$  and  $F_1$  yield at time  $2 - \epsilon$ , where  $\epsilon$  is arbitrarily small. From this discussion, we have the following.

**Eligibility blocking:** A subtask  $T_i$  may be blocked for an entire quantum in slot  $t$ , where  $e(T_i) = t$  holds, if some processor becomes available within slot  $t - 1$  and the processor is allocated to a subtask whose priority at  $t$  is

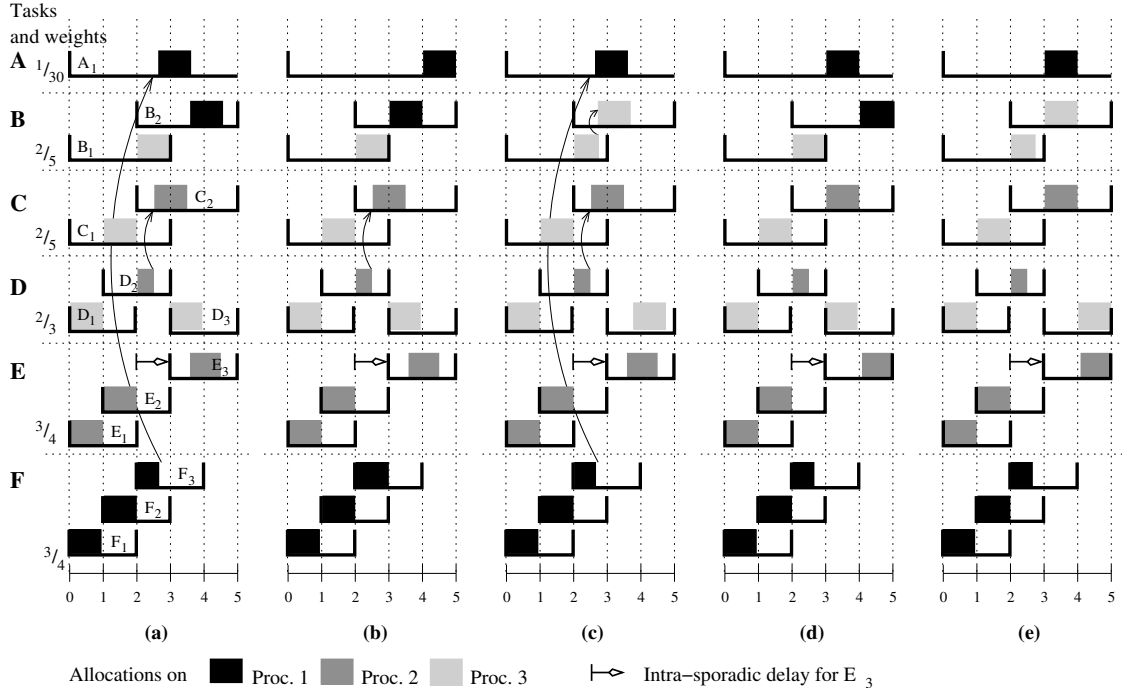


Figure 3: Conditions under which a subtask may be predecessor-blocked at time  $t$  under  $\text{PD}^2\text{-DVQ}$ . (a) Subtask  $B_2$  is predecessor-blocked at time 3 by subtask  $A_1$ . (b)  $B_2$  would not be blocked if  $F_3$  does not yield before time 3, or (c)  $B_1$  yields early. However, if  $B_1$  yields early, then  $D_2$  is eligibility-blocked at time 3. (d) A  $\text{PD}^B$  schedule corresponding to the schedule in (a). (e) A  $\text{PD}^B$  schedule corresponding to the schedule in (c).

lower than that of  $T_i$ . If a subtask is blocked in this manner, then we say that it is *eligibility-blocked*.

Another subtle priority inversion that is possible in  $\text{PD}^2\text{-DVQ}$  is illustrated in Fig. 3(a). In this example, subtasks  $D_2$  and  $F_3$ , which are scheduled in slot 2, yield before the end of that slot and the processors they executed upon are promptly allocated to the highest priority subtasks that are also ready in the second slot, which are  $C_2$  and  $A_1$ . However, subtask  $B_1$ , also scheduled in slot 2, executes for an entire quantum, and hence, processor 3 does not become available until time 3. At time 3, processor 3 is allocated to subtask  $D_3$ , which has a higher priority than  $B_1$ 's successor  $B_2$ , under  $\text{PD}^2$ . (Though  $B_2$  and  $D_3$  have equal deadlines,  $D_3$  has a higher priority by  $\text{PD}^2$ 's tie breaking rules.) In this example, the deadline of  $A_1$  is at time 30. Therefore,  $B_2 \prec A_1$  holds, and hence,  $B_2$  suffers blocking at time 3. Also note that  $B_2$  is eligible before time 3, but is constrained by its predecessor not completing execution before time 3. Thus, the blocking that  $B_2$  suffers is different from the eligibility blocking described earlier. Had  $F_3$  not yielded early, but executed until the end of slot 2, then as shown in Fig. 3(b),  $B_2$  would have been scheduled at time 3 in the place of  $A_1$ , and  $A_1$ 's execution would have been postponed to some later time. On the other hand, had  $B_1$  yielded early, before time 3, as in Fig. 3(c), then  $B_2$  would have started executing before  $D_3$ , which is not eligible until time 3;  $D_3 \prec B_2$  holds, hence  $D_3$  would have been eligibility-blocked at time 3. Thus, on the whole, in the sce-

nario in inset (a),  $B_2$  is blocked by  $A_1$  because a processor became available before its predecessor completed executing. Thus, we have the following.

**Predecessor blocking:** A subtask  $T_i$  may be blocked in a slot  $t$  that is not the first slot of its IS-window, if  $T_i$ 's predecessor executes up to time  $t$  on processor  $P_k$ , some other processor  $P_\ell$  becomes available before time  $t$  and is allocated to a ready subtask with a lower priority than  $T_i$ , and at time  $t$ ,  $P_k$  is allocated to a subtask  $U_j$  with an equal or a higher priority than  $T_i$ , where  $e(U_j) = t$ .

Blocking of the above nature that a subtask may be subjected to in a slot that is not the first in its IS-window is termed *predecessor blocking*. In our example, for  $B_2$  to be predecessor-blocked at time  $t$ , it is necessary for the processor that  $B_1$  executes on to be unavailable until time  $t$ , and for the higher-priority subtask  $D_3$  to be eligible only at  $t$ . This observation can be generalized as follows.

**Property PB:** Let  $\mathcal{U}$  denote the set of all subtasks that are predecessor-blocked at  $t$ , where  $t$  is an integer, in some schedule  $\mathcal{S}$  under  $\text{PD}^2\text{-DVQ}$ . Then, there exist another set of subtasks  $\mathcal{V}$  and a set of processors  $\mathcal{P}$  such that the following holds.

$$|\mathcal{P}| \geq |\mathcal{U}| \wedge |\mathcal{V}| \geq |\mathcal{U}| \wedge (\forall P \in \mathcal{P} :: P \text{ makes a scheduling decision at time } t \text{ in } \mathcal{S}) \wedge (\forall V_k \in \mathcal{V} :: e(V_k) = t \wedge \mathcal{S}(V_k) = t \wedge (\forall U_j \in \mathcal{U} :: U_j \preceq V_k))$$

Property PB says that predecessor blocking can occur only

under specific conditions; these conditions are used in the proof that establishes the tardiness bound of PD<sup>2</sup>-DVQ. Because the total number of processors is  $M$ ,  $|\mathcal{P}| \leq M$  holds, and hence,  $|\mathcal{U}| \leq M$  holds. That is, at most  $M$  subtasks could be predecessor-blocked at time  $t$ .

Furthermore, note that for  $T_i$  to be predecessor-blocked at time  $t$ ,  $T_i$ 's predecessor has to complete executing at time  $t$ . Hence,  $T_i$  can be predecessor-blocked at most once only. Similarly, if  $T_i$  is eligibility-blocked at  $t$ , it means that  $T_i$ 's predecessor completed executing at or before  $t$ , and hence,  $T_i$  cannot be predecessor-blocked at any later time.

**Algorithm PD<sup>B</sup>.** Having given some insight into the timing anomalies of PD<sup>2</sup>-DVQ, we now explain algorithm PD<sup>B</sup> in detail. PD<sup>B</sup> is based on PD<sup>2</sup>, and mimics, in the SFQ model, blockings due to priority inversions that subtasks may be subjected to under PD<sup>2</sup>-DVQ; hence, all times in this discussion are integral, unless stated otherwise. As described above, a subtask  $T_i$  may be blocked in slot  $t$  if either  $e(T_i) = t$  holds or  $T_i$ 's predecessor  $T_h$  completes execution at  $t$ . Therefore, at any time  $t$ , PD<sup>B</sup> partitions the set of all subtasks that are *ready* at  $t$ , into three disjoint subsets  $EB(t)$ ,  $PB(t)$ , and  $DB(t)$ , based on subtask eligibility times, as follows.

$$EB(t) \stackrel{\text{def}}{=} \{T_i \mid e(T_i) = t\} \quad (4)$$

$$PB(t) \stackrel{\text{def}}{=} \{T_i \mid e(T_i) < t \wedge T_i \text{ was not blocked before } t \text{ and could be predecessor-blocked at } t\} \quad (5)$$

$$DB(t) \stackrel{\text{def}}{=} \{T_i \mid e(T_i) < t \wedge T_i \notin PB(t)\} \quad (6)$$

$EB(t)$  is the set of all subtasks that are not eligible until  $t$ , and hence, could *potentially* be eligibility-blocked at  $t$  under PD<sup>2</sup>-DVQ. A subtask in  $PB(t)$  could *potentially* be predecessor-blocked at  $t$  under PD<sup>B</sup>.  $DB(t)$  is the set of all subtasks that definitely do not block at  $t$  under PD<sup>B</sup>.

For an illustration of how ready subtasks are classified at any time  $t$ , consider the PD<sup>B</sup> schedule in Fig. 2(c). In this example, at time 2,  $\{B_1, C_1, D_2, E_2, F_2\}$  is the set of all subtasks that are ready. Of these subtasks,  $D_2, E_2$ , and  $F_2$  are in  $EB(2)$ ; the rest are in  $DB(2)$ . In Fig. 3, the set of subtasks  $\{A_1, B_2, C_2, D_3, E_3\}$  is ready at time 3. Of these subtasks,  $D_3$  and  $E_3$  are in  $EB(3)$ , and  $A_1$  and  $C_2$  are in  $DB(3)$ .  $B_2$  could be in either  $PB(3)$  or  $DB(3)$ . If  $B_2$  is included in  $PB(3)$ , then the schedule for time 3 may be as in inset (d); otherwise, it may be as in inset (e).

Within each subset, subtasks are prioritized in accordance with PD<sup>2</sup> and are scheduled in this order. Like PD<sup>2</sup>, PD<sup>B</sup> schedules at most  $M$  subtasks in every slot. The subtask chosen in each scheduling decision has the highest priority within the subset that it belongs to. Subtasks in  $DB(t)$  should not block at  $t$ . Therefore, a subtask  $T_i$  in  $EB(t)$  or  $PB(t)$  cannot be scheduled at  $t$ , if some subtask  $U_j$  in  $DB(t)$  has not yet been scheduled at  $t$  and  $U_j \prec T_i$  holds. However, to mimic eligibility blocking, a subtask  $U_j$  from  $DB(t)$  may be scheduled prior to a subtask  $T_i$  from  $EB(t)$ , even if  $T_i \prec U_j$  holds.

To mimic predecessor blocking, PD<sup>B</sup> allows a subtask  $U_j$  from  $DB(t)$  to be scheduled prior to a subtask  $V_k$  in  $PB(t)$ , even if  $V_k \prec U_j$  holds. However, for every such  $V_k$  blocked, because Property PB holds in PD<sup>2</sup>-DVQ, PD<sup>B</sup> ensures that a subtask  $W_\ell$  in  $EB(t)$  is scheduled such that  $W_\ell \preceq V_k$  holds. Note that, because  $V_k \prec U_j$  holds, we also have  $W_\ell \prec U_j$ . That PD<sup>2</sup>-DVQ exhibits such behavior was stated earlier in Property PB, and is proved formally later in Lemma 1. We now give the priority rules for PD<sup>B</sup>.

**PD<sup>B</sup> priorities.** Let  $EB(t, r)$ ,  $PB(t, r)$ , and  $DB(t, r)$ , where  $r \geq 1$ , be equal to  $EB(t)$ ,  $PB(t)$ , and  $DB(t)$  as defined in (4), (5), and (6), respectively, with the subtasks selected in the first  $r - 1$  scheduling decisions for time slot  $t$  removed.  $p = |PB(t)| \leq M$  denotes the number of subtasks that could potentially be predecessor-blocked at  $t$ , and hence, denotes the maximum number of processors that subtasks in  $PB(t)$  could contend for. Therefore, the remaining  $M - p$  processors can be freely allocated to subtasks in  $EB(t)$  and  $DB(t)$ , and hence, a subtask in  $PB(t)$  may be assigned a lower priority in the first  $M - p$  scheduling decisions than every subtask that is not in  $PB(t)$ . Recall that by Property PB,  $p$  is also the minimum number of processors that make scheduling decisions at time  $t$  under the DVQ model. Furthermore, in that model, subtasks in  $PB(t)$  and  $EB(t)$  are ready during these decisions, and hence, at least  $p$  highest-priority subtasks cannot be blocked at time  $t$ . To mimic this correctly, scheduling is strictly by PD<sup>2</sup> during the final  $p$  scheduling decisions under PD<sup>B</sup>. Referring back to the example in Fig. 4(a),  $B_2$  is predecessor-blocked at time 3 because its predecessor executed until time 3 on processor 3; hence, a scheduling decision is made at time 3 for that processor. Apart from  $B_2$ , subtasks  $D_3$  and  $E_3$ , which are in  $EB(3)$ , are also ready at time 3, and the subtask with the highest priority,  $D_3$ , is scheduled.

Thus, the relative priority between subtasks  $T_i$  and  $U_j$  at time  $t$  depends on the sets the subtasks belong to and the number of the scheduling decision, and is given by Table 1. Rows and columns in the table indicate the sets that  $T_i$  and  $U_j$  belong to, respectively. Entries in the cells indicate the conditions under which  $T_i \sqsubseteq U_j$  holds at time  $t$  (the priority of  $T_i$  is at least that of  $U_j$ ), for scheduling decision  $r$ , where  $1 \leq r \leq M$ .  $T_i \sqsubseteq U_j$  would hold iff  $T_i \sqsubseteq U_j \wedge U_j \not\sqsubseteq T_i$  holds. Similarly,  $T_i \supseteq U_j$  would hold iff  $T_i \not\sqsubseteq U_j \wedge U_j \sqsubseteq T_i$  holds.

As mentioned earlier, subtasks within each subset are scheduled by their PD<sup>2</sup> priority; the entries along the main diagonal confirm this. Recall that a subtask in  $DB(t)$  cannot be blocked and note that the entries in the last row and column of the table ensure this. Eligibility blocking is mimicked by not giving a higher priority to subtasks in  $EB(t)$  than a subtask in  $DB(t)$  in the first  $M - p$  scheduling decisions regardless of their PD<sup>2</sup> priorities. For example, if  $r \leq M - p$ ,  $T_i \in EB(t, r)$ ,  $U_j \in DB(t, r)$ , and  $T_i \prec U_j$

Table 1: PD<sup>B</sup> priority definition.

Conditions for $T_i \sqsubseteq U_j$ to hold at $t$ , for scheduling decision $r$ , where $1 \leq r \leq M$ .			
$T_i$	$U_j$		
	$EB(t, r)$	$PB(t, r)$	$DB(t, r)$
$EB(t, r)$	$T_i \preceq U_j$	$T_i \preceq U_j \vee$ $r \leq M - p$	$T_i \preceq U_j$
$PB(t, r)$	$T_i \preceq U_j \wedge$ $r > M - p$	$T_i \preceq U_j$	$T_i \preceq U_j \wedge$ $r > M - p$
$DB(t, r)$	$T_i \preceq U_j \vee$ $r \leq M - p$	$T_i \preceq U_j \vee$ $r \leq M - p$	$T_i \preceq U_j$

holds, then by Table 1, both  $T_i \sqsubseteq U_j$  and  $U_j \sqsubseteq T_i$  hold for the  $r^{\text{th}}$  scheduling decision. So, if  $U_j$  is scheduled before  $T_i$ , then  $T_i$  may be blocked (if it does not get selected in a later scheduling decision for  $t$ ). On the other hand, if  $U_j \prec T_i$  holds, then only  $U_j \sqsubseteq T_i$  will hold. This ensures that a subtask with a lower priority in  $EB(t)$  does not get scheduled before a higher-priority subtask in  $DB(t)$ . However, if  $T_i \prec U_j$  and  $r > M - p$  hold, then only  $T_i \sqsubseteq U_j$  will hold, ensuring that the final  $p$  scheduling decisions are by PD<sup>2</sup>.

We claim the following before elaborating on how predecessor blocking is mimicked.

**Claim 1** Let  $T_i \in DB(t, r)$ ,  $U_j \in PB(t, r)$ , and let  $U_j \prec T_i$  hold. Let  $T_i$  be the subtask that is scheduled under PD<sup>B</sup> in scheduling decision  $r$  for slot  $t$ , where  $r \leq M - p$ . Then, the following holds:  $(\forall V_k \in DB(t, s), r < s \leq M :: U_j \prec V_k)$ .

**Proof:** Because subtasks in  $DB(t)$  are scheduled by their PD<sup>2</sup> priority,  $T_i \preceq V_k$  holds, with  $V_k$  as defined in the claim. Therefore, because  $U_j \prec T_i$  holds,  $U_j \prec V_k$  holds, as well.  $\square$

**Claim 2** Let  $T_i \in EB(t, r)$ ,  $U_j \in PB(t, r)$ , and let  $U_j \prec T_i$  hold. Let  $T_i$  be the subtask that is scheduled under PD<sup>B</sup> in scheduling decision  $r$  for slot  $t$ , where  $r \leq M - p$ . Then, the following holds:  $(\forall V_k \in DB(t, s) \cup EB(t, s), r < s \leq M :: U_j \prec V_k)$ .

**Proof:** Similar to the proof of Claim 1.  $\square$

Predecessor blocking is mimicked by excluding the  $p$  subtasks in  $PB(t)$  in the initial  $M - p$  scheduling decisions, as given by the entries in the middle row and column. This gives an opportunity for a subtask  $T_i$  in  $DB(t)$  with a lower priority than the lowest-priority subtask  $U_j$  in  $PB(t)$  to be scheduled. The final  $p$  decisions are in strict PD<sup>2</sup> order among the remaining unscheduled subtasks. Therefore, if one or more subtasks in  $EB(t)$  not scheduled in the initial  $M - p$  scheduling decisions have higher priority than  $U_j$ , then  $U_j$  will experience blocking. By Claim 1, no subtask remaining in  $DB(t)$  will have higher priority than  $T_i$ , and hence  $U_j$ , in the final  $p$  scheduling decisions. By Claim 2, if a subtask in  $EB(t)$  with a lower priority than  $U_j$  is scheduled in the initial  $M - p$  scheduling decisions,

then no subtask remaining in  $EB(t)$  or  $DB(t)$  would have higher priority than  $U_j$ , and hence, any subtask in  $PB(t)$ , during the final  $p$  scheduling decisions.

We now formally prove Property PB.

**Lemma 1** Let  $\mathcal{S}$  be a schedule under PD<sup>2</sup>-DVQ for a task system  $\tau$ . Let  $T_i$  be a subtask,  $\mathcal{U}$ , a set of subtasks in  $\tau$ , and  $t$ , an integral time, such that the following hold.

$$e(T_i) \leq t - 1 \quad (7)$$

$$(\forall U_j \in \mathcal{U} :: e(U_j) \leq t - 1 \wedge U_j \text{ is ready at or before } t) \quad (8)$$

$$(\forall U_j \in \mathcal{U} :: U_j \prec T_i) \quad (9)$$

$$T_i \text{ is executing at } t \quad (10)$$

$$(\forall U_j \in \mathcal{U} :: \mathcal{S}(U_j) > t) \quad (11)$$

Then, each of the following holds.

- (a)  $(\forall U_j \in \mathcal{U} :: \text{the predecessor of } U_j \text{ exists and completes executing at time } t, \text{ i.e., } U_j \text{ is not ready until } t)$ .
- (b) There exists a set  $\mathcal{V}$  of subtasks such that  $|\mathcal{V}| \geq |\mathcal{U}|$  and  $(\forall V_k \in \mathcal{V} :: e(V_k) = t \wedge \mathcal{S}(V_k) = t \wedge (\forall U_j \in \mathcal{U} :: V_k \preceq U_j))$  hold.
- (c)  $|\mathcal{U}| \leq M$ .

**Proof:** Because  $\mathcal{S}$  is a PD<sup>2</sup>-DVQ schedule, not all times referred to in this proof are integral. Let  $t_r$  be the time at which  $T_i$  is scheduled in  $\mathcal{S}$ , i.e.,  $\mathcal{S}(T_i) = t_r$ . Then, by (10),  $t - 1 < t_r \leq t$  holds.

This implies that no subtask that is ready throughout the interval  $(t - 1, t_r]$ , but is not scheduled at  $t_r$ , has a higher priority than  $T_i$ , i.e., we have

$$(\forall W_\ell :: W_\ell \text{ is ready at } t_r \wedge \mathcal{S}(W_\ell) > t_r \Rightarrow T_i \preceq W_\ell). \quad (12)$$

**Proof of (a):** Let  $U_j$  be any subtask in  $\mathcal{U}$ , and  $U_b$  its predecessor. By (8) and (11),  $U_j$  is ready by time  $t$ . Therefore, because  $U_j \prec T_i$  holds, PD<sup>2</sup>-DVQ would not prefer  $T_i$  to  $U_j$  at or after time  $t$ . Hence,

$$t - 1 < t_r < t \quad (13)$$

holds, i.e.,  $T_i$  is scheduled in the middle of slot  $t - 1$  and  $U_j$  is not ready at  $t_r$ . Because  $e(U_j) \leq t - 1$  holds, this implies that  $U_j$  is not ready at time  $t_r$ , due to  $U_b$  executing at  $t_r$ . We claim the following.

**Claim 3** For every subtask  $W_\ell$  that is ready at or before time  $t_s$ , where  $t_s < t$ , and is not scheduled before  $t$ ,  $T_i \preceq W_\ell$  holds.

**Proof:** By (13), and since the eligibility time of a subtask is integral, every subtask that becomes ready at  $t_s$ , where  $t_r < t_s < t$ , does so because its predecessor completes execution at  $t_s$ . Let  $t_1, t_2, \dots, t_n$ , where  $t_r < t_1, t_n < t$ , and  $t_k < t_{k+1}$ , for all  $1 \leq k < n$ , denote all the distinct times at which one or more subtasks become ready in  $(t_r, t)$ . Let  $t_{n+1} = t$ . Then, to establish the claim, it is sufficient to show that for every subtask  $W_\ell$  that is ready before time  $t_k$ , i.e., is ready in  $[0, t_k)$ , and is not scheduled before  $t_k$ ,  $T_i \preceq W_\ell$  holds, for all  $k$ , where  $1 \leq k \leq n + 1$ . We show this by induction on  $k$ .

By the way we defined  $t_1 \dots t_{n+1}$ , no subtask that is not ready at  $t_r$  is ready in  $(t_r, t_1)$ . Hence, by (12), for every

subtask  $W_\ell$  that is ready before  $t_1$  and is not scheduled before  $t_1$ ,  $T_i \preceq W_\ell$  holds. Thus,  $k = 1$  forms the base case. Let  $\mathcal{W}$  denote the set of all subtasks that are ready before  $t_k$  and are not scheduled before  $t_k$ , where  $1 \leq k \leq n$ . For the induction hypothesis, assume that for every subtask  $W_\ell$  in  $\mathcal{W}$ ,  $T_i \preceq W_\ell$  holds.

We next show that  $T_i \preceq R_q$  holds for every subtask  $R_q$  that becomes ready before time  $t_{k+1}$  and is not scheduled before  $t_{k+1}$ . If  $R_q$  in fact becomes ready before  $t_k$ , then  $T_i \preceq R_q$  follows by the induction hypothesis. Otherwise,  $R_q$  becomes ready at  $t_k$ , the only time in the interval  $[t_k, t_{k+1})$  that some subtask becomes ready. Let  $\mathcal{R}$  denote the set of all subtasks that become ready at  $t_k$ , and let  $m = |\mathcal{R}|$ . Because the predecessors of subtasks in  $\mathcal{R}$  complete execution at time  $t_k$ , the  $m \leq M$  processors on which they executed become available at  $t_k$ , for which PD<sup>2</sup>-DVQ makes scheduling decisions at time  $t_k$ . Thus, because  $R_q$  (which is in  $\mathcal{R}$ ) is not scheduled at  $t_k$ , some other subtask that is ready before  $t_k$ , and not scheduled before  $t_k$ , *i.e.*, a subtask  $W_\ell \in \mathcal{W}$ , is scheduled instead, which in turn implies that  $W_\ell \preceq R_q$  holds. By the induction hypothesis,  $T_i \preceq W_\ell$  holds, from which  $T_i \preceq R_q$  follows.  $\square$

By Claim 3, and (9) and (11),  $U_j$  is not ready until time  $t$ .

**Proof of (b):** Let  $u = |\mathcal{U}|$ , and let  $U_j$  be the subtask with the highest priority in  $\mathcal{U}$ . By part (a),  $U_j$  is ready only at time  $t$ . By Claim 3,  $T_i \preceq X_m$  holds for every subtask  $X_m$  that is ready before  $t$  and remains unscheduled until time  $t$ . By part (a), the predecessors of subtasks in  $\mathcal{U}$  complete executing at time  $t$ , hence  $u$  processors become available at  $t$ . Let

$$m = |\{W_\ell \mid e(W_\ell) \leq t \wedge \text{The predecessor of } W_\ell \text{ completes executing at } t \wedge W_\ell \notin \mathcal{U}\}|. \quad (14)$$

Then, at least  $u + m$  processors become available at  $t$ , for which scheduling decisions are made at time  $t$ . Therefore, if  $U_j$  is not scheduled at time  $t$ , then by (9) and Claim 3, at least  $u + m$  subtasks with priority at least that of  $U_j$  that are not ready until time  $t$  are scheduled at  $t$ . Let  $\mathcal{V}$  denote the set of all such subtasks. Then, by (14), only  $m$  subtasks in  $\mathcal{V}$  can have their predecessors executing until  $t$ . Hence, at least  $u$  subtasks in  $\mathcal{V}$  are not ready until  $t$  because they are not eligible until  $t$ .

**Proof of (c):** Follows directly from part (a).  $\blacksquare$

Following is the counterpart of the above lemma for PD<sup>B</sup>, and is proved in [6].

**Lemma 2** *Let  $\mathcal{S}$  be a schedule under PD<sup>B</sup> for a task system  $\tau$ . Let  $T_i$  be a subtask in  $\tau$ ,  $\mathcal{U}$ , a set of subtasks in  $\tau$ , and  $t$ , an integral time, such that the following hold: (i)  $e(T_i) \leq t - 1$ , (ii)  $(\forall U_j \in \mathcal{U} :: e(U_j) \leq t - 1 \wedge U_j$  is ready at or before  $t$ ), (iii)  $(\forall U_j \in \mathcal{U} :: U_j \prec T_i)$ , (iv)  $\mathcal{S}(T_i) = t$ , and (v)  $(\forall U_j \in \mathcal{U} :: \mathcal{S}(U_j) > t)$ . Then, we have the following: There exists a set  $\mathcal{V}$  of subtasks such*

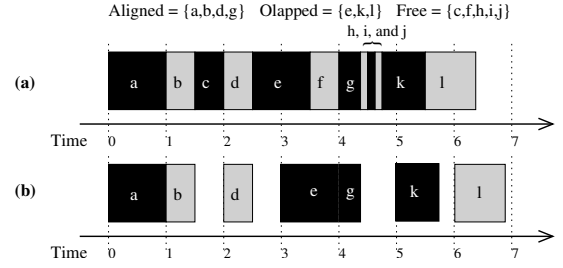


Figure 4: (a) A partial schedule under PD<sup>2</sup>-DVQ on a single processor. Each rectangular box represents the execution of a subtask. Subtasks in subsets *Aligned*, *Olapped*, and *Free* are as indicated. (b) An equivalent schedule for subtasks in *Aligned* and *Olapped* under the SFQ model.

that  $|\mathcal{V}| \geq |\mathcal{U}|$  and  $(\forall V_k \in \mathcal{V} :: e(V_k) = t \wedge \mathcal{S}(V_k) = t \wedge (\forall U_j \in \mathcal{U} :: V_k \preceq U_j))$  hold.

### 3.2. Tardiness Bound for PD<sup>2</sup>-DVQ

We now turn to showing that PD<sup>B</sup> represents a worst case for PD<sup>2</sup>-DVQ. Towards this end, we would like to show that the tardiness of every feasible GIS task system  $\tau$  under PD<sup>2</sup>-DVQ is at most the tardiness of some feasible GIS task system  $\tau'$  under PD<sup>B</sup>. However, since we later show that tardiness under PD<sup>B</sup> is at most one quantum, it would suffice to show that the tardiness of  $\tau$  under PD<sup>2</sup>-DVQ is at most the ceiling of the tardiness of  $\tau'$  under PD<sup>B</sup>, *i.e.*,  $(\forall \tau : (\exists \tau' : \text{tardiness}(\tau, \text{PD}^2\text{-DVQ}) \leq \lceil \text{tardiness}(\tau', \text{PD}^B) \rceil))$  holds.

Let  $\mathcal{S}_{DQ}$  denote a schedule for some feasible task system  $\tau$  under PD<sup>2</sup>-DVQ. Then, to establish our claim, it is sufficient to show that there exists a corresponding valid schedule under PD<sup>B</sup>,  $\mathcal{S}_B$ , for some task system  $\tau'$ , such that the tardiness of every subtask in  $\mathcal{S}_{DQ}$  is at most the ceiling of that of some subtask in  $\mathcal{S}_B$ . We begin by defining the following sets of subtasks.

$$\begin{aligned} All &= \{T_i \mid T_i \text{ is a subtask in } \tau\} \\ Aligned &= \{T_i \mid T_i \in All \wedge \mathcal{S}_{DQ}(T_i) \text{ is integral}\} \\ Olapped &= \{T_i \mid T_i \in All \wedge \mathcal{S}_{DQ}(T_i) \text{ is not integral} \\ &\quad \wedge (\mathcal{S}_{DQ}(T_i) + c(T_i)) \text{ is not integral}\} \\ Charged &= Aligned \cup Olapped \\ Free &= All \setminus Charged \end{aligned}$$

An example of this classification is shown in Fig. 4(a).

We next consider a task system  $\tau'$  comprised of subtasks in *Charged* only, and construct a schedule  $\mathcal{S}_B$  for  $\tau'$  as follows. The actual execution cost  $c(T_i)$  of every subtask  $T_i$  in  $\tau'$  remains the same as in  $\tau$ . Let the time at which  $T_i$  commences its execution in  $\mathcal{S}_B$  be  $\mathcal{S}_B(T_i)$ , if  $\mathcal{S}_{DQ}(T_i)$  is integral. Otherwise, postpone its commencement time to the beginning of the next slot, *i.e.*, to  $\lceil \mathcal{S}_{DQ}(T_i) \rceil$ . In other words, define  $\mathcal{S}_B$  as follows.

$$\begin{aligned} (\forall T_i : T_i \in Aligned :: \mathcal{S}_B(T_i) = \mathcal{S}_{DQ}(T_i)) \\ (\forall T_i : T_i \in Olapped :: \mathcal{S}_B(T_i) = \lceil \mathcal{S}_{DQ}(T_i) \rceil) \end{aligned}$$

The schedule so constructed for the subtasks in *Charged* in Fig. 4(a) is shown in Fig. 4(b). This construction ensures the following.

**Lemma 3** *The commencement time and completion time for every subtask in  $\tau'$  in  $\mathcal{S}_B$  are at least their respective values in  $\mathcal{S}_{DQ}$ .*

We now prove the following lemma concerning the tardiness of every subtask in  $\tau$  in  $\mathcal{S}_{DQ}$ .

**Lemma 4** *Let  $T_i$  be some subtask in  $\tau$ . Then  $\text{tardiness}(T_i, \mathcal{S}_{DQ}) \leq \lceil \text{tardiness}(U_j, \mathcal{S}_B) \rceil$ , where  $U_j$  is some subtask in  $\tau'$ .*

**Proof:** We consider two cases based on whether  $T_i$  is in *Charged* or *Free*. If  $T_i$  is in *Charged*, then by Lemma 3, its completion time in  $\mathcal{S}_{DQ}$  is at most its completion time in  $\mathcal{S}_B$ , which by (3), implies that  $\text{tardiness}(T_i, \mathcal{S}_{DQ}) \leq \text{tardiness}(T_i, \mathcal{S}_B)$ , and hence the lemma holds.

If  $T_i$  is in *Free*, then it should be scheduled in the middle of some slot  $t$  in  $\mathcal{S}_{DQ}$ . Let the completion time of  $T_i$  be  $t + \delta$ , where  $0 < \delta \leq 1$ . Let  $U_j$  be the subtask that was executing at time  $t$  on the same processor in the same schedule. Therefore,  $U_j$  is in *Charged* (by the definition of *Charged*). Hence, we have  $\mathcal{S}_{DQ}(U_j) \leq t$ . Let  $U_j$ 's completion time be  $t + \epsilon$ . Because  $T_i$  executes after  $U_j$  on the same processor, we have  $0 < \epsilon < \delta$ . We consider two cases.

**Case 1:**  $U_j \prec T_i$ . In this case, we have  $d(T_i) \geq d(U_j)$ . Hence, by (3), the tardiness of  $T_i$  in  $\mathcal{S}_{DQ}$  is given by

$$\begin{aligned} \text{tardiness}(T_i, \mathcal{S}_{DQ}) &= \max(0, t + \delta - d(T_i)) \\ &\leq \max(0, t + \delta - d(U_j)) \quad ; d(T_i) \geq d(U_j) \\ &\leq \max(0, t - d(U_j) + \lceil \delta \rceil) \\ &= \max(0, t - d(U_j) + \lceil \epsilon \rceil) \quad ; 0 < \delta, \epsilon < 1 \\ &= \max(0, \lceil t + \epsilon - d(U_j) \rceil) \\ &= \lceil \text{tardiness}(U_j, \mathcal{S}_{DQ}) \rceil. \end{aligned}$$

Therefore, the tardiness of  $T_i$  is at most the ceiling of the tardiness of  $U_j$  in  $\mathcal{S}_{DQ}$ . Because  $U_j$  is in *Charged*, its tardiness in  $\mathcal{S}_B$  is at least its tardiness in  $\mathcal{S}_{DQ}$ , and hence, the tardiness of  $T_i$  in  $\mathcal{S}_{DQ}$  is at most the ceiling of the tardiness of  $U_j$  in  $\mathcal{S}_B$ .

**Case 2:**  $T_i \prec U_j$ . Similar to Case 1.  $\blacksquare$

We are left with showing that  $\mathcal{S}_B$  is a valid schedule for  $\tau'$  under  $\text{PD}^B$ . The following lemma, proved in [6], shows that this is the case. The lemma follows because the priority definition of  $\text{PD}^B$  is designed to allow the blockings that are possible under  $\text{PD}^2$ -DVQ, and  $\mathcal{S}_B$ , derived from  $\mathcal{S}_{DQ}$ , exhibits such blockings only, if any.

**Lemma 5**  *$\mathcal{S}_B$  is a valid  $\text{PD}^B$  schedule for  $\tau'$ .*

Thus, by our definition of  $\mathcal{S}_{DQ}$ , the construction of  $\mathcal{S}_B$ , and Lemmas 4 and 5, we have the following theorem.

**Theorem 1** *The tardiness of every feasible GIS task system under  $\text{PD}^2$ -DVQ is at most the ceiling of the tardiness of some feasible task system under  $\text{PD}^B$ .*

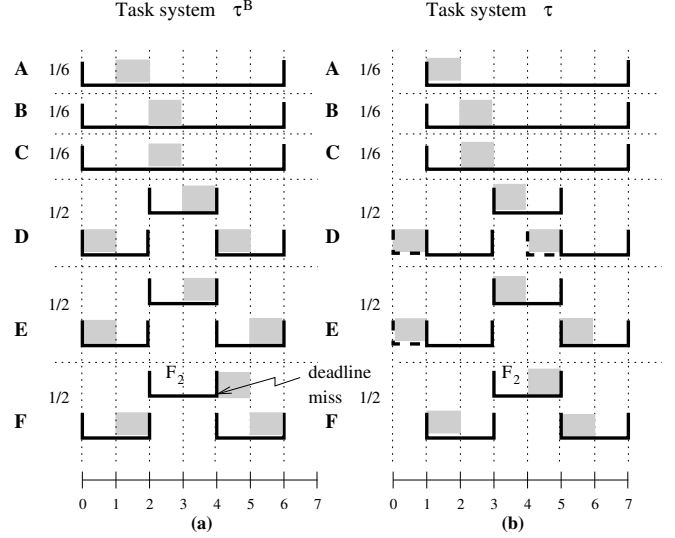


Figure 5: (a) A  $\text{PD}^B$  schedule  $\mathcal{S}$  (on two processors) with a deadline miss for  $\tau^B$ . (b)  $\mathcal{S}$  is also a schedule for  $\tau$ , which is derived from  $\tau^B$  by shifting the PF-windows of subtasks in  $\tau^B$  to the right by one time unit. The eligibility time is retained for subtasks that are scheduled in the first slots of their IS windows and is increased by one time unit for the rest.

### 3.3. Tardiness Bound for $\text{PD}^B$

In this subsection, we describe how to show that  $\text{PD}^B$  ensures a tardiness bound of one quantum for every feasible GIS task system. In the discussion that follows, we let the eligibility time, release time, and deadline functions introduced in Sec. 2 to take a task system as a second parameter (in addition to a subtask). When unambiguous, the second parameter will be omitted.

Let  $\tau^B$  be any feasible GIS task system and let  $\mathcal{S}$  be a  $\text{PD}^B$  schedule for  $\tau^B$ . Let  $\tau$  be another task system that contains every subtask that is in  $\tau^B$ , and no others, and let the parameters of the subtasks in  $\tau$  be given by the following rules: (i) The PF-window of every subtask in  $\tau^B$  is shifted to the right by one slot in  $\tau$ . That is,

$$r(T_i, \tau) = r(T_i, \tau^B) + 1 \wedge d(T_i, \tau) = d(T_i, \tau^B) + 1 \quad (15)$$

holds for every subtask  $T_i$ . (ii) If  $\mathcal{S}(T_i) = e(T_i, \tau^B)$  holds, then the eligibility time of  $T_i$  in  $\tau$  is the same as its value in  $\tau^B$ . Otherwise, the eligibility time of  $T_i$  is increased by one time slot. By these rules, we have the following.

$$(\forall T_i \in \tau^B, \mathcal{S}(T_i) = e(T_i, \tau^B) :: e(T_i, \tau) = e(T_i, \tau^B)) \quad (16)$$

$$(\forall T_i \in \tau^B, \mathcal{S}(T_i) > e(T_i, \tau^B) :: e(T_i, \tau) = e(T_i, \tau^B) + 1) \quad (17)$$

An example in which the above rules are applied is given in Fig. 5. The modifications to  $\tau^B$  by (15), (16), and (17) increase the deadline of every subtask uniformly by one time slot, but do not alter the tie break parameters of  $\text{PD}^2$  for any subtask. Therefore, at any time  $t$ , the relative priorities under  $\text{PD}^2$  among subtasks ready at  $t$  is the same in both  $\tau^B$  and  $\tau$ . Hence, it can be seen that scheduling decisions in  $\mathcal{S}$  conform to  $\text{PD}^2$  in most cases, and in time slots where priority inversions with respect to  $\text{PD}^2$  occur, the property given by the following lemma (proved in [6])

holds. The lemma establishes the conditions under which one or more subtasks in  $\tau$  may be blocked at time  $t$  in  $\mathcal{S}$ . This lemma holds by Lemma 2 and the fact that if a subtask  $T_i$  in  $\tau^B$  is eligibility-blocked at  $t$  in  $\mathcal{S}$ , then  $T_i$  in  $\tau$  does not suffer any blocking. This is because by (4),  $e(T_i, \tau^B) = t$  holds, and by (4) and (5),  $T_i$  in  $\tau^B$ , and hence in  $\tau$ , does not suffer any blocking after time  $t$ . By (17),  $e(T_i, \tau) = e(T_i, \tau^B) + 1 = t + 1$  holds, and hence,  $T_i$  in  $\tau$  does not suffer blocking at time  $t$  either.

**Lemma 6** *Let  $T_i$  be a subtask and  $\mathcal{U}$  a set of subtasks in  $\tau$  such that the following hold: (i)  $e(T_i, \tau) \leq t$ , (ii)  $(\forall U_j \in \mathcal{U} :: e(U_j, \tau) \leq t \wedge U_j$  is ready at or before  $t$ ), (iii)  $(\forall U_j \in \mathcal{U} :: U_j \prec T_i)$ , (iv)  $\mathcal{S}(T_i) = t$ , and (v)  $(\forall U_j \in \mathcal{U} :: \mathcal{S}(U_j) > t)$ . Then, we have the following: There exists a set  $\mathcal{V}$  of subtasks such that  $|\mathcal{V}| \geq |\mathcal{U}|$  and  $(\forall V_k \in \mathcal{V} :: e(V_k, \tau) = t < r(V_k, \tau) \wedge \mathcal{S}(V_k) = t \wedge (\forall U_j \in \mathcal{U} :: V_k \preceq U_j))$  holds.*

Finally, we need one more property concerning subtasks in  $\tau$ . If  $h > 0$  processors are idle in slot  $t$  in a schedule, then we say that there are  $h$  holes in that schedule. Let  $t$  be a slot with a hole in  $\mathcal{S}$  and let  $V$  be any task that is not scheduled in  $t$ . Let  $V_k$  be a task that is not scheduled at  $t$  and let  $V_k$  be the subtask of  $V$  with the largest index such that  $e(V_k, \tau^B) \leq t$  holds. Then, because there is a hole in  $t$ ,  $V_k$  should have been scheduled before  $t$ . Also,  $e(V_l, \tau^B) \geq t + 1$  holds for  $V_l$ , where  $V_l$  is the successor of  $V_k$ . If  $\mathcal{S}(V_l) > t + 1$  holds, then regardless of whether  $e(V_l, \tau^B) = t + 1$  or  $e(V_l, \tau^B) > t + 1$  holds, by (16) and (17),  $e(V_l, \tau) > t + 1$ . Lemma 7 therefore follows.

**Lemma 7** *Let  $t$  be a slot with one or more holes and let  $V$  be a task that is not scheduled at  $t$  in  $\mathcal{S}$ . Then, if  $V$  is not scheduled at  $t + 1$  also, then  $e(V_l, \tau) > t + 1$  holds, where  $V_l$  is a subtask of  $V$  that is not scheduled before  $t + 1$ .*

Now, if the tardiness of some subtask  $T_i$  in  $\tau^B$  exceeds one quantum in  $\mathcal{S}$ , then by (15),  $T_i$  in  $\tau$  misses its deadline in  $\mathcal{S}$ . Therefore, to show that the tardiness of  $\tau^B$  is at most one quantum in  $\mathcal{S}$ , it is sufficient to show that no subtask in  $\tau$  misses its deadline in  $\mathcal{S}$ . Because Lemmas 6 and 7 hold, this idea can be generalized as follows: to show that  $PD^B$  ensures a tardiness bound of one quantum for every feasible GIS task system, it is sufficient to show that in a schedule  $\hat{\mathcal{S}}$  for a feasible task system  $\hat{\tau}$ , where the subtasks of  $\hat{\tau}$  satisfy Lemma 7 and  $\hat{\mathcal{S}}$  satisfies Lemma 6, no deadlines are missed. We prove this in [6], which by our discussion in this subsection establishes Theorem 2. (The proof is very similar to the proof that establishes the optimality of  $PD^2$  [10].)

**Theorem 2**  *$PD^B$  ensures a tardiness of at most one quantum to every feasible GIS task system.*

Thus, by Theorems 1 and 2 we have the following.

**Theorem 3**  *$PD^2$  under the DVQ model ensures a tardiness of at most one quantum to every feasible GIS task system.*

## 4. Conclusion

We have addressed a limitation of Pfair scheduling that requires processor allocations to be in units of fixed-sized quanta, which may lead to wasted processor time. We have determined that relaxing this requirement worsens the tardiness of Pfair algorithms by at most one quantum only. This result enables the use of a relaxed Pfair scheduling model for providing soft real-time guarantees, and thereby, improves the practicality of Pfair scheduling. As future work, we plan on investigating the impact of relaxing another limitation of Pfair scheduling, which requires the execution cost and period of each task to be expressed as an integral multiple of the maximum size of a quantum.

## References

- [1] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proc. of the 7th International Conference on Real-time Computing Systems and Applications*, pages 297–306, Dec. 2000.
- [2] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.
- [3] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.
- [4] S. Baruah, J. Gehrke, and C. G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proc. of the 9th International Parallel Processing Symposium*, pages 280–288, Apr. 1995.
- [5] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate fair scheduling in multiprocessor systems. In *Proc. of the 7th IEEE Real-time Technology and Applications Symposium*, pages 3–14, June 2001.
- [6] U. Devi and J. Anderson. Desynchronized Pfair scheduling on multiprocessors (full version). Available at <http://www.cs.unc.edu/~anderson/papers.html>, October 2004.
- [7] P. Holman. *On the Implementation of Pfair Scheduled Multiprocessor Systems*. PhD thesis, University of North Carolina at Chapel Hill, Aug. 2004.
- [8] P. Holman and J. Anderson. Implementing pfairness on a symmetric multiprocessor. In *Proc. of the 10th IEEE Real-time Technology and Applications Symposium*, pages 544–553, May 2004.
- [9] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.
- [10] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proc. of the 34th ACM Symposium on Theory of Computing*, pages 189–198, May 2002.