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 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.

*Work supported by NSF grants CCR 0204312, CCR 0309825, and CCR 0408996.

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 [12], each task T is characterized by two parameters, a worst-case execution time (WCET) $T.e$, and a period $T.p$: an instance or *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.

Timing constraints (or deadlines) in a real-time system may be classified as either hard or soft. Hard deadlines cannot be missed, while soft deadlines may occasionally be missed, if the extent of the miss is bounded. A real-time system may either be exclusively comprised of either hard real-time tasks or soft real-time tasks, or be a combination of the two.

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 such as synthetic aperture imaging (to name a few). 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.* [6], 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 not exceeding M. Under other approaches, only task sets with total utilization not exceeding a value that is slightly higher than $M/2$ may be scheduled correctly, in the worst case [13, 5, 4].

Under Pfair scheduling, each task is subdivided into quantum-length *subtasks* that are subjected to intermediate deadlines, called *pseudo-deadlines*. Subtasks are then scheduled on an earliest-pseudo-deadline-first basis, with deadline ties among subtasks resolved using tie-breaking rules. When scheduling periodic tasks, pseudo-deadlines are assigned to subtasks in a way that ensures that all job deadlines are met. An example is provided in Sec. 2.

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 and 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 [10]. 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 four 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 task invocations (*i.e.*, 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.

- 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 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. In this paper, we show that under the DVQ model and an otherwise-optimal Pfair scheduling algorithm, the amount by which deadlines can be missed is bounded. In particular, we show that 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 [8]. In addition to supporting the DVQ model, 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. Also, the early-release model of Pfair scheduling [1] provides a less-expensive and simpler alternative to using an auxiliary scheduler.

In other related work, Holman and Anderson proposed the *staggered* model [11], 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. Then, the DVQ model is presented and the main result of this paper proved in Sec. 3. Sec. 4 concludes.

2 Background on Pfair Scheduling

This section provides a brief overview of Pfair scheduling [1, 2, 3, 6, 14] under the SFQ model. In introducing Pfair scheduling concepts, we initially assume that only periodic tasks that begin execution at time 0 are to be scheduled. Such a task T has an integer *period* $T.p$, an integer (per-job) *execution cost* $T.e$, and a *weight* $wt(T) = T.e/T.p$ in the range $(0, 1]$.

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 multiples of the quantum size. For simplicity, we henceforth assume that the quantum size is one time unit. 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*. To ensure that all quanta are uniform in size, 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* S. Formally,

$$S : \tau \times \mathcal{N} \mapsto \{0, 1\}, \tag{1}$$

where τ is a task set and $\mathcal{N}$ is the set of non-

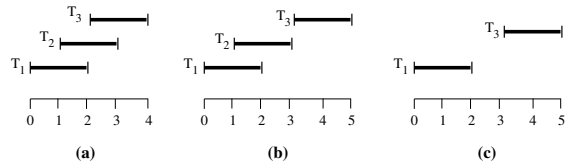

Figure 1. (a) Windows of the first job of a periodic task T with weight $3/4$. This job consists of subtasks T_1, T_2, and T_3, each of which must be scheduled within its window. (This pattern repeats for every job.) **(b)** The Pfair windows of an IS task. Subtask T_3 becomes eligible one time unit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

negative integers. $S(T,t) = 1$ iff T is scheduled in slot t. On M processors, $\sum_{T \in \tau} S(T,t) \leq M$ holds for all t.

To facilitate quantum-based scheduling, each task T is broken into a potentially infinite sequence of quantum-length *subtasks*. The i^{th} subtask of task T is denoted T_i, where $i \geq 1$. 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 schedule tasks by choosing subtasks to execute at the beginning of every slot. If a scheduled 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 [6], PD [7], and PD^2 [3] — and one suboptimal algorithm — earliest pseudo-deadline first (EPDF) [3] — are known. In all the above 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. In fact, the three optimal algorithms mentioned above differ only in their tie-breaking rules. PD^2 is the most efficient of the three and its tie-breaking rules form a subset of 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 [2, 14] 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 well-studied 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 illustrated 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.

$$r(T_i) = \theta(T_i) + \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \quad (3)$$

$$d(T_i) = \theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil \quad (4)$$

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. Formally, the offsets satisfy the following property.

$$k > i \Rightarrow \theta(T_k) \geq \theta(T_i). \quad (5)$$

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.

$$(\forall T_i :: e(T_i) \leq r(T_i) \ \wedge \ e(T_i) \leq e(T_{i+1})) \qquad (6)$$

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}, ...,T_{k-1} were present and released as early as possible. Fig. 1(c) shows an example.

If T_i is the most recently released subtask of T, then T may release T_k, where $k > i$, as its next subtask at time t, if $r(T_i) + \left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \leq t$. If a task T, after executing subtask T_i, releases subtask T_k, then T_k is called the *successor* of T_i and T_i is called the *predecessor* of T_k. Every subtask of T that is released before T_i is called an *ancestor* of T_i and every subtask that is released after T_i is called a *descendent* of T. Note that a subtask can have at most one predecessor and one successor only, but may have multiple ancestor and descendant subtasks.

A correct schedule in which no subtask misses its deadline exists for a GIS task system τ on M processors iff its total utilization is at most M, *i.e.*, $\sum_{T \in \tau} wt(T) \leq M$ holds [2]. A task system with total utilization at most M is said to be *feasible* on M processors.

Soft real-time systems and tardiness. As mentioned earlier, the jobs (and subtasks) of a soft real-time system may occasionally miss their deadlines, if the amount by which a subtask misses its deadline, referred to as its *tardiness*,

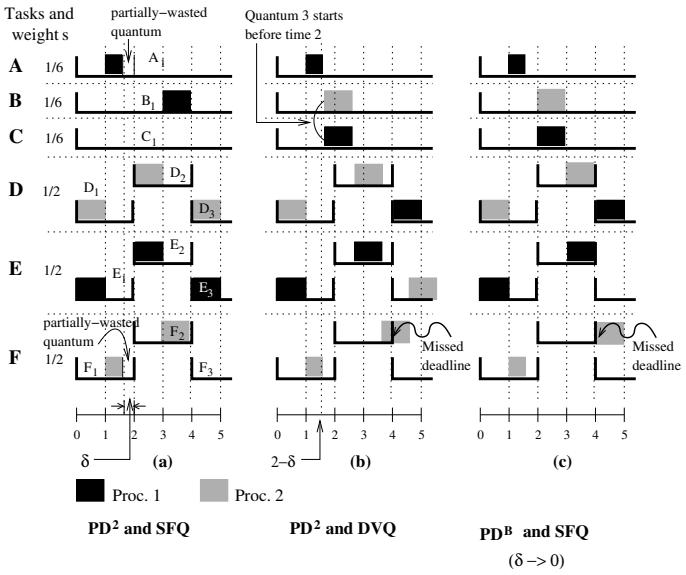

Figure 2. Difference between **(a)** the SFQ model and **(b)** the DVQ model under PD^2. **(c)** A possible schedule under PD^B in the SFQ model, obtained by postponing the allocations in (b) that do not commence on a slot boundary to the next slot boundary.

is bounded. Formally, the tardiness of a subtask T_i in schedule $\mathcal{S}$ is defined as

$$tardiness(T_i, \mathcal{S}) = \max(0, t - d(T_i)), \qquad (7)$$

where t is the time at which T_i completes executing in $\mathcal{S}$ and $d(T_i)$ is its deadline. We sometimes omit specifying the schedule if it is clear from the context. The tardiness of a task system τ under scheduling algorithm $\mathcal{A}$, denoted $tardiness(\tau, \mathcal{A})$, is defined as the maximum tardiness of any subtask in τ under any valid schedule under $\mathcal{A}$. If κ is the maximum tardiness of any task system under $\mathcal{A}$, then $\mathcal{A}$ is said to *ensure a tardiness bound of* κ.

3 Pfair in the DVQ Model

In this section, we describe the DVQ model in detail and show that under this model and the PD^2 scheduling algorithm, tasks may miss their deadlines by at most one quantum only. (Recall that PD^2 is optimal under the SFQ model.) This result also extends to most prior work on Pfair

scheduling; in general, tardiness bounds guaranteed under most previously-proposed suboptimal Pfair algorithms and schemes are worsened 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 characterization of tasks, *i.e.*, the task model, is unaltered. The SFQ model requires all quanta to be of uniform size and aligned 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, *i.e.*, yields an interval of time δ before the end of its 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. 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. 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 under the SFQ model, whereas a new quantum begins immediately under the DVQ model. Because the next subtasks of tasks D, E, and F are not eligible until $t = 2$, the two processors are assigned to subtasks B_1 and C_1 at $t = 2 - \delta$ under the DVQ model.

As seen in Fig. 2, 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. We deal with this issue by overloading this function to denote the time at which a *subtask* commences execution. Formally, if $\mathcal{S}$ is a schedule for a task set τ, then $\mathcal{S} : \{\text{subtasks in } \tau\} \mapsto \mathcal{Q}$, where $\mathcal{Q}$ is the set of all rational numbers. For schedules that conform to the SFQ model, $\mathcal{S}(T_i)$ is integral, for every subtask T_i. 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. In the example in Fig. 2(b), $\mathcal{S}(A_1) = \mathcal{S}(B_1) = 2 - \delta$ and $c(A_1) = c(B_1) = 1 - \delta$.

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, *i.e.*, T_i is scheduled somewhere in the interval $(t - 1, t]$. We henceforth assume that preemption and migration costs are zero. (Such costs can be easily accounted for by inflating task execution costs appropriately [10].) 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 also unchanged: the term "slot t" still refers to the interval $[t, t + 1)$ on the real time line.

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 whenever a lower-priority subtask (or job) executes, while a ready, higher-priority subtask waits. Under such conditions, 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 maxi-

mum size of one quantum only. To avoid reasoning directly in the DVS model, which can be quite cumbersome, and to leverage analysis techniques and results presented previously for the SFQ model, we establish the tardiness bound for the DVQ model in the following four steps.

- We consider allocations in the DVQ model when subtasks execute for a duration of $1-\delta$ in the limit $\delta \to 0$, and thus reduce them to allocations that conform to the SFQ model. For example, in the limit $\delta \to 0$, the allocations in Fig. 2(b) reduce to those in Fig. 2(c).

- 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*).

- 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.
- 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.

These four steps are elaborated on in the subsection that follows. 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. One type of priority inversion 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 and E_2 become eligible. (Note that D_2 and E_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 and E_2 are blocked by lower-priority subtasks B_1 and C_1, respectively. Their blocking time would be maximized, if subtasks A_1 and F_1 yield at time $2-\epsilon$, where ϵ is arbitrarily small. From this discussion, we have the following.

> A higher-priority subtask T_i may be blocked for an entire quantum in the first slot t of its IS-window (which implies that $e(T_i) = t$ holds), if some processor becomes available within slot $t-1$ (in $(t-1, t)$, just before the eligibility time of T_i), and the processor is allocated to a lower-priority subtask.

If a subtask is blocked in this manner, then we say that it is *eligibility-blocked*.

Another subtle priority inversion that is possible in PD^2-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 eligible 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 PD^2. (Though B_2 and D_3 have equal deadlines, D_3 has a higher priority by PD^2's tie breaking rules.) In this example, the deadline of A_1 is at time 30. Therefore, B_2 has a higher priority than A_1, and hence, suffers blocking at time 3 in this schedule. 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

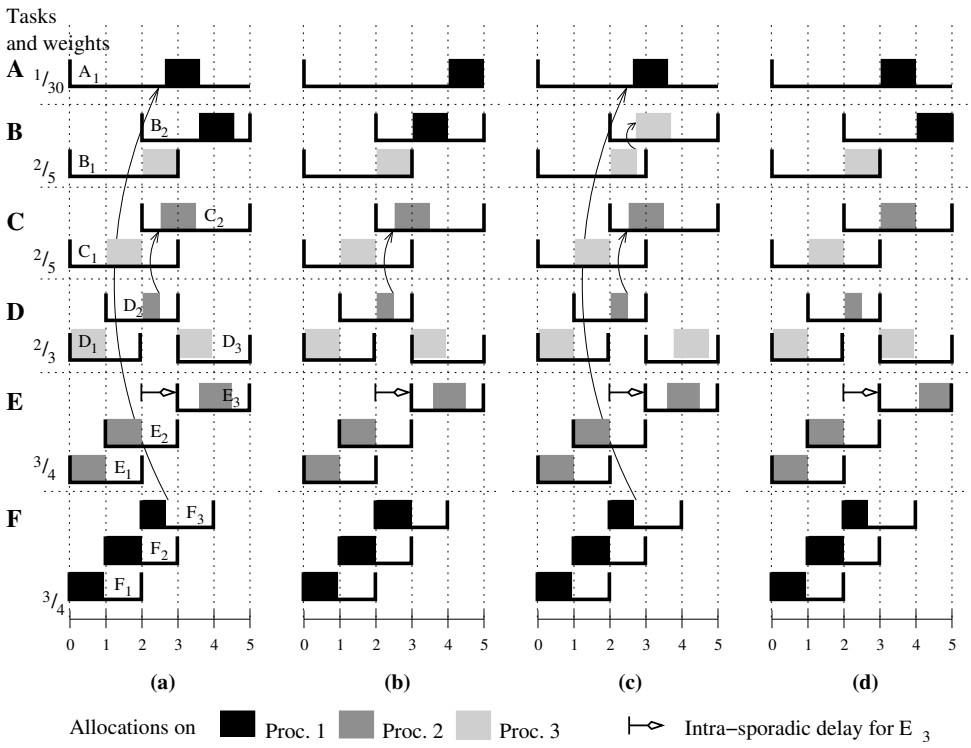

Figure 3. Illustration of the conditions under which a a subtask may be *predecessor blocked* at time t under PD^2-DVQ. **(a)** Subtask B_2 is predecessor blocked at time 3 by subtask A_1. **(b)** B_2 would not be blocked if F_3 does not yield before time 3, or **(c)** B_1 yields early. However, if B_1 yields early, then D_2 is *eligibility blocked* at time 3. **(d)** A schedule for the same task set under PD^B.

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 has a higher priority than B_2 under PD^2, and hence, would have been eligibility-blocked at time 3. Thus, on the whole, in the scenario in inset (a), B_2 is blocked by A_2 because a processor became available before its predecessor completed executing. Thus, we have the following.

> 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_l becomes available before time t and is allocated to a ready subtask with a lower priority than T_i, and at time t, P_i is allocated to a higher-priority subtask U_j, where $e(U_j) = t$.

We call this type of blocking by a subtask in a slot that is not the first in its IS-window *predecessor blocking*. Note that, in our example, for B_2 to be predecessor-blocked at time t, it is necessary for the processor B_1 executes on to be unavailable until time t, and for the higher-priority subtask D_3 to be released 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 in some schedule $\mathcal{S}$ under PD^2-DVQ. Then, there exist another set of subtasks $\mathcal{V}$ and a set of processors $\mathcal{P}$ such that the following holds.

$$|\mathcal{P}| \geq |\mathcal{U}| \ \wedge \ |\mathcal{V}| \geq |\mathcal{U}| \ \wedge \ (\forall P \in \mathcal{P} :: P \text{ makes a scheduling decision at time } t \text{ in } \mathcal{S}) \wedge (\forall V_k \in V :: e(V_k) = t \ \wedge \ \mathcal{S}(t) = t) \qquad (8)$$

Property PB says that predecessor blocking can occur only under specific circumstances; these circumstances are exploited in the proof that establishes the tardiness bound of PD^2-DVQ.

Algorithm PD^B. Having given some insight into the timing anomalies of PD^2-DVQ, we now explain algorithm PD^B in detail. PD^B is based on PD^2. In describing PD^B, 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 with respect to PD^B we use symbols $\sqsubset$, $\sqsubseteq$, $\sqsupset$, and $\sqsupseteq$.

As mentioned earlier, PD^B mimics, in the SFQ model, blockings that subtasks may be subjected to under PD^2-DVQ due to priority inversions; 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^B partitions the set of all subtasks that are *ready at* t, into three disjoint subsets EB, PB, and DB, based on subtask eligibility times, as follows.

$$EB(t) \stackrel{\text{def}}{=} \{T_i \mid e(T_i) = t\} \tag{9}$$

$$PB(t) \stackrel{\text{def}}{=} \{T_i \mid e(T_i) < t \ \wedge \ T_i \text{ could be predecessor-blocked at } t\} \tag{10}$$

$$DB(t) \stackrel{\text{def}}{=} \{T_i \mid e(T_i) < t \ \wedge \ T_i \notin \mathcal{P}(t)\} \tag{11}$$

A subtask in $EB(t)$ is not eligible until t, and hence, could *potentially* be eligibility-blocked at t under PD^2-DVQ, and a subtask in $PB(t)$ could *potentially* be predecessor-blocked. $DB(t)$ is the set of all subtasks that definitely do not block at t. For an illustration of how ready subtasks are classified at any time t, consider the PD^B 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 $EB(2)$, and the remaining are in $DB(2)$. In the schedule in Fig. 3(d), 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 $DB(3)$. Because B_1 executes until time t, B_2 cannot be scheduled at any time before t, even though $e(B_2) < t$ holds, and hence, is in $PB(3)$.

Within each subset, subtasks are prioritized in accordance with PD^2 and are scheduled in this order. Like PD^2, PD^B schedules at most M subtasks in every slot. The subtask chosen in each scheduling decision is from one of the three subsets and has the highest priority within that subset. Subtasks in $DB(t)$ are unconstrained by subtasks in the other two subsets, and hence, do not experience any blocking anywhere in time slot t, under PD^2-DVQ. Therefore, under PD^B, 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^B 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, by Property PB, PD^B ensures that a subtask W_l in $EB(t)$ is scheduled such that $W_l \preceq V_k$ holds. Note that because $V_k \prec U_j$ holds, we also have $W_l \prec U_j$. That PD^2-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^B.

PD^B priorities. Let $EB(t, r)$, $PB(t, r)$, and $DB(t, r)$ be equal to $EB(t)$, $PB(t)$, and $DB(t)$ as defined in (9), (10), and (11), respectively, with the subtasks selected in the first $r - 1$ scheduling decisions for time slot t removed. $p = |PB(t)|$ denotes the number of subtasks that could potentially be predecessor-blocked at t under PD^2-DVQ, 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 (*i.e.*, there are at least p 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^2 during the final p scheduling decisions (or the final p processors) under PD^B. 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, during which B_2 is ready. Subtasks D_3 and E_3, which are in $EB(3)$, are also ready at time 3. The subtask with the highest priority among the unscheduled ready tasks, D_3, is scheduled at time 3.

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 (the priority of T_i is at least that of U_j), for scheduling decision r, where $1 \leq r \leq M$, at time t. $T_i \sqsubset U_j$ would hold iff $T_i \sqsubseteq U_j \wedge U_j \not\sqsubseteq T_i$ holds. Similarly, $T_i \sqsupset U_j$ would hold iff $T_i \not\sqsubseteq U_j \wedge U_j \sqsubseteq T_i$ holds. (In the table entries, $\preceq$ is with respect to PD^2.)

As mentioned earlier, subtasks within each subset are scheduled by their PD^2 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^2 priorities. For example, if $r \leq M - p$, $T_i \in EB(t, r)$, $U_j \in DB(t, r)$, and $T_i \prec U_j$ holds,

Table 1. PD^B priority definition.

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

then by Table 1, both $T_i \sqsubseteq U_j$ and $U_j \sqsubseteq T_i$ hold for the r^{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, to ensure 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. This ensures that the final p scheduling decisions are by PD^2, as explained earlier.

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^B 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^2 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. □

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^B 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. ■

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^2 order among the remaining unscheduled subtasks. Therefore, if one or more subtasks in $EB(t)$ 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^2-DVQ for a task system τ. Let T_i be a subtask, $\mathcal{U}$, a set of subtasks in τ, and t, an integral time, such that the following hold.*

$$e(T_i) \leq t-1 \tag{12}$$

$$(\forall U_j \in \mathcal{U} :: e(U_j) \leq t-1 \ \wedge \ U_j \text{ is ready at or before } t) \tag{13}$$

$$(\forall U_j \in \mathcal{U} :: U_j \prec T_i) \tag{14}$$

$$T_i \text{ is executing at } t \tag{15}$$

$$(\forall U_j \in \mathcal{U} :: \mathcal{S}(U_j) > t) \tag{16}$$

Then, each of the following holds.

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

Proof: Because $\mathcal{S}$ is a PD^2-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 (15), $t-1 < t_r \leq t$ holds.

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

$$(\forall W_l :: W_l \text{ is ready at } t_r \wedge \mathcal{S}(W_l) > t_r \ \Rightarrow \ T_i \preceq W_l). \tag{17}$$

Proof of (a): Let U_j be any subtask in $\mathcal{U}$, and U_b its predecessor. By (13) and (16), U_j is ready by time t. Therefore, because $U_j \prec T_i$ holds, PD^2-DVQ would not prefer T_i to U_j at time t. Hence,

$$t-1 < t_r < t \tag{18}$$

holds, *i.e.*, T_i is scheduled in the middle of slot $t-1$ (recall that t is an integer), 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_l that is ready at or before time t_s, where $t_s < t$, and is not scheduled before t, $T_i \preceq W_l$ holds.*

Proof: By (17), for every subtask W_l that is ready at or before time t_r, and is not scheduled before t_r, $T_i \preceq W_l$ holds.

By (18), 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 $t_1, t_2, \cdots, t_n$, 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_l that is ready before time t_k, *i.e.*, is ready in $[0, t_k)$, and is not scheduled before t_k, $T_i \preceq W_l$ holds, for all k, where $1 \leq k \leq n+1$. We show this by induction on k.

By the way we defined $t_1 \cdots t_{n+1}$, no subtask that is not ready at t_r is ready in (t_r, t_1). Hence, by (17), for every subtask W_l that is ready before t_1 and is not scheduled before t_1, $T_i \preceq W_l$ 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_l in $\mathcal{W}$, $T_i \preceq W_l$ 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^2-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 $W_l \in \mathcal{W}$, is scheduled instead, which in turn implies that $W_l \preceq R_q$ holds. By the induction hypothesis, $T_i \preceq W_l$ holds, from which $T_i \preceq R_q$ follows. □

By the claim above, and (14) and (16), it follows that 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. Because the predecessors of subtasks in $\mathcal{U}$ complete executing at time t, u processors become available at t. Let m denote the number of subtasks that become ready at t, have their predecessors executing until t, and are not in $\mathcal{U}$. That is, let

$$m = |\{W_l \mid \text{The predecessor of } W_l \text{ completes executing at } t \wedge W_l \notin \mathcal{U}\}|. \quad (19)$$

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 (14) and Claim 3, it implies that 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 (19), 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. ■

Following is the counterpart of the above lemma for PD^B, proved in [9].

Lemma 2 *Let $\mathcal{S}$ be a schedule under PD^B for a task system τ. Let T_i be a subtask in τ, $\mathcal{U}$, a set of subtasks in τ, and t, an integral time, such that the following hold.*

$$e(T_i) \leq t - 1 \quad (20)$$

$$(\forall U_j \in \mathcal{U} :: e(U_j) \leq t - 1 \wedge U_j \text{ is ready at or before } t) \quad (21)$$

$$(\forall U_j \in \mathcal{U} :: U_j \prec T_i) \quad (22)$$

$$\mathcal{S}(T_i) = t \quad (23)$$

$$(\forall U_j \in \mathcal{U} :: \mathcal{S}(U_j) > t) \quad (24)$$

Then, we have the following: *There exists a set $\mathcal{V}$ of subtasks such that $|\mathcal{V}| \geq |\mathcal{U}|$ and $(\forall V_k \in \mathcal{V} :: e(V_k) = t \wedge \mathcal{S}(V_k) = t \wedge (\forall V_k \in \mathcal{V}, U_j \in \mathcal{U} :: V_k \preceq U_j)$ hold, and T_i is scheduled in slot t before every subtask in $\mathcal{V}$.*

3.2 Tardiness Bound for PD^2-DVQ

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

Let $\mathcal{S}_{DQ}$ denote a schedule for some feasible task system τ under PD^2-DVQ. Then, to establish our claim, it is sufficient to show that there exists a corresponding valid schedule under PD^B, $\mathcal{S}_B$, for some task system τ', 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 introducing some notation. Let All denote the set of all subtasks in τ, $Aligned$, the subset of all subtasks in All that commence

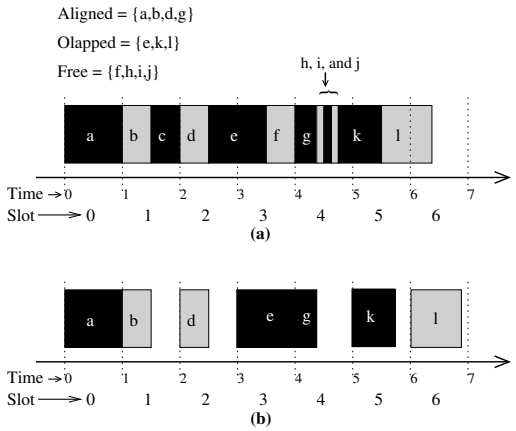

Figure 4. (a) A partial schedule under PD^2-DVQ on a single processor. Each rectangular box represents the execution of a subtask. For clarity, successive subtasks are shown in alternate shades; no other relation is implied by the shading. Subtasks in subsets $Aligned$, $Olapped$, and $Free$ are as indicated. **(b)** An equivalent schedule for subtasks in $Aligned$ and $Olapped$ under the SFSQ model.

executing on a slot boundary (*i.e.*, at some integral time t), and $Olapped$, the subset of all subtasks that *neither* commence *nor* complete executing on a slot boundary but are in the middle of execution at a slot boundary, in $\mathcal{S}_{DQ}$. Note that subsets $Aligned$ and $Olapped$ are disjoint. Formal definitions are given below.

$$\begin{aligned} All &= \{T_i \mid T_i \text{ is a subtask in } \tau\} \\ Aligned &= \{T_i \mid T_i \in All \ \wedge\ \mathcal{S}_{DQ}(T_i) \text{ is integral}\} \\ Olapped &= \{T_i \mid T_i \in All \ \wedge\ \mathcal{S}_{DQ}(T_i) \text{ is not integral} \\ &\qquad \wedge\ (\mathcal{S}_{DQ}(T_i) + c(T_i)) \text{ is not integral} \\ &\qquad \wedge\ \mathcal{S}_{DQ}(T_i) + c(T_i) = \lfloor \mathcal{S}_{DQ}(T_i) \rfloor + 1\} \end{aligned}$$

We also define two other subsets:

$$\begin{aligned} Charged &= Aligned \cup Olapped \\ Free &= All \setminus Charged \end{aligned}$$

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

We next consider a task system τ' comprised of subtasks in $Charged$ only, and construct a schedule $\mathcal{S}_B$ for τ' as follows. The actual execution cost $c(T_i)$ of every subtask T_i in τ' remains the same as in τ. Let the time at which T_i commences its execution in $\mathcal{S}_B$ be $\mathcal{S}_{DQ}(T_i)$, 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 τ' 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 τ in $\mathcal{S}_{DQ}$.

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

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 (7), implies that $tardiness(T_i, \mathcal{S}_{DQ}) \leq tardiness(T_i, \mathcal{S}_B)$. Therefore, the lemma holds if T_i is in $Charged$.

If T_i is in $Free$, then it should be scheduled in the middle of some slot t in $\mathcal{S}_{DQ}$, as shown in Fig. 5. 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 $\mathcal{S}_{DQ}(U_j) \leq t$, and let its 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: $U_j \preceq T_i$. In this case, we have $d(T_i) \geq d(U_j)$. Hence, by (7), the tardiness of T_i in $\mathcal{S}_{DQ}$ is given by

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

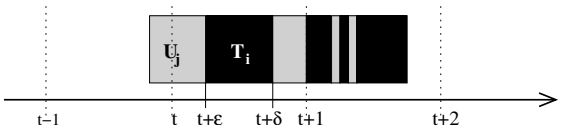

Figure 5. Lemma 4. T_i and U_j are scheduled on the same processor at the times indicated in $\mathcal{S}_{DQ}$. T_i is in *Free* and U_j is in *Charged* (and hence, in τ'). We determine the tardiness of T_i in $\mathcal{S}_{DQ}$ in terms of the tardiness of U_j in $\mathcal{S}'_B$.

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

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: $T_i \prec U_j$. The proof of this case is similar to that of Case 1, and is omitted here due to page limitations. ■

We are left with showing that $\mathcal{S}_B$ is a valid schedule for τ' under PD^B. The following lemma, proved in [9], shows that this is the case. The lemma follows because $\mathcal{S}_{DQ}$ is a valid schedule for PD^2-DVQ, and because PD^B's priority definition is designed to allow the blockings that occur under PD^2-DVQ.

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

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

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

3.3 Tardiness Bound for PD^B

In this subsection, we show that PD^B ensures a tardiness bound of one quantum for every feasible GIS task system. To prove this result, let τ^B be any feasible GIS task system, and let τ be a corresponding task system obtained by shifting the IS-window of every subtask in τ^B right by one time slot. In other words, for every subtask T_i in τ^B, τ includes a subtask T'_i such that $e(T'_i) = e(T_i) + 1$, $r(T'_i) = r(T_i) + 1$, and $d(T'_i) = d(T_i) + 1$. Let $\mathcal{S}_B$ be a schedule under PD^B for τ^B, and $\mathcal{S}$, a schedule under PD^2 for τ. Then, because PD^2 is optimal, no subtask in τ misses its deadline in $\mathcal{S}$, and if $\mathcal{S}$ is viewed as a schedule for τ^B, in which T_i is scheduled in the place of T'_i, then no subtask in τ^B misses its deadline by more than one quantum in $\mathcal{S}$. $\mathcal{S}$ would differ from $\mathcal{S}_B$ because subtasks are eligible one time slot earlier in τ^B, and hence, may be scheduled earlier in $\mathcal{S}_B$, perhaps displacing other subtasks in the process. Therefore, to prove that PD^B generates a "valid" schedule that ensures a tardiness of at most one quantum for every subtask, it is sufficient to prove that no subtask in τ misses its deadline under PD^B, if the eligibility time of each subtask is decreased by one, *i.e.*, $e(T'_i) = e(T_i)$ holds for every subtask T'_i in τ. For this, we systematically convert $\mathcal{S}$ to $\mathcal{S}_B$ by decreasing the eligibility time of exactly one subtask at a time, scheduling it in accordance with $\mathcal{S}_B$, and showing that the intermediate schedules in this process remain valid. A schedule for a task system σ is said to be *valid in time slot* t if **(i)** each subtask T_i in σ is scheduled in some slot in the interval $[e(T_i), d(T_i))$, **(ii)** no two subtasks of the same task are scheduled in slot t, and **(iii)** the number of subtasks scheduled at t is at most M. A schedule is *valid* iff it is valid in every slot.

We first define an irreflexive total order on the subtasks in τ^B by the sequence in which they are scheduled in $\mathcal{S}_B$ by PD^B. (Subtasks that are scheduled in the same slot are ordered by the order in which they are selected for that slot.) Let $rank(T_i)$ denote the position of subtask T_i in this total order. We then say that $\hat{\tau}$ is k- *compliant to* τ^B if the following hold: **(i)** there exists a bijective (one-to-one and onto) mapping from subtasks in τ^B to subtasks in $\hat{\tau}$, **(ii)** for every subtask V_k in τ^B, $d(V'_k) = d(V_k) + 1$ and

$r(V'_k) = r(V_k) + 1$ hold for its image V'_k in $\hat{\tau}$, **(iii)** for every subtask T_i with rank at most k in τ^B, $e(T'_i) = e(T_i)$ holds for its image T'_i in $\hat{\tau}$, **(iv)** for every subtask U_j with rank greater than k in τ^B, $e(U'_j) = e(U_j) + 1$ holds for its image U'_j in $\hat{\tau}$. Fig. 6 shows an example. In this example, letting τ^B be the task system in inset (a), inset (a) depicts a PD^B schedule for τ^B. (The schedule has been repeated twice for ease of comparison with the schedules in insets (b) and (c).) In the task system in inset (b), every subtask in τ^B is right-shifted by one slot. Note that in inset (a), subtask F_2 misses its deadline by one quantum. However, subtask F'_2 meets its deadline in the PD^2 schedule in inset (b) In the task system in inset (c), subtasks with ranks 1 through 4 have their eligibility times advanced by one time slot. The task system here is 4-compliant to τ^B.

Schedule $\hat{\mathcal{S}}$ for $\hat{\tau}$ is said to be *k-compliant to $\mathcal{S}_B$* if **(i)** $\hat{\mathcal{S}}$ is valid, **(ii)** for every subtask with rank at most k, T'_i and T_i are scheduled in the same slot in $\hat{\mathcal{S}}$ and $\mathcal{S}_B$, respectively, and **(iii)** subtasks with rank greater than k are scheduled according to PD^2 in $\hat{\mathcal{S}}$. The schedules in Fig. 6(b) and (c) are 0-compliant and 4-compliant, respectively, to the schedule in inset (a).

Let n denote the number of subtasks in τ^B. To show that PD^B ensures a tardiness of at most one quantum to τ^B, we show that there exist a task system τ' and a schedule $\mathcal{S}'$ for τ' that are n-compliant to τ^B and $\mathcal{S}_B$, respectively, by induction on k-compliance. As explained earlier, the task system τ derived from τ^B by right shifting each subtask in τ^B by one slot is 0-compliant to τ^B and the PD^2 schedule $\mathcal{S}$ for τ is 0-compliant to $\mathcal{S}_B$. Thus, $k = 0$ forms the base case. For the induction hypothesis, assume that there exists a task system τ^k, which is k compliant to τ^B, and a schedule $\mathcal{S}_k$ for τ^k, which is k-compliant to $\mathcal{S}_B$, where $k \geq 0$. We then show that a $(k+1)$-compliant task system and schedule exist.

Lemma 6 *There exist a task system τ^{k+1} that is $(k+1)$-compliant to τ^B, and a schedule $\mathcal{S}_{k+1}$ for τ^{k+1} that is $(k+1)$-compliant to $\mathcal{S}_B$.*

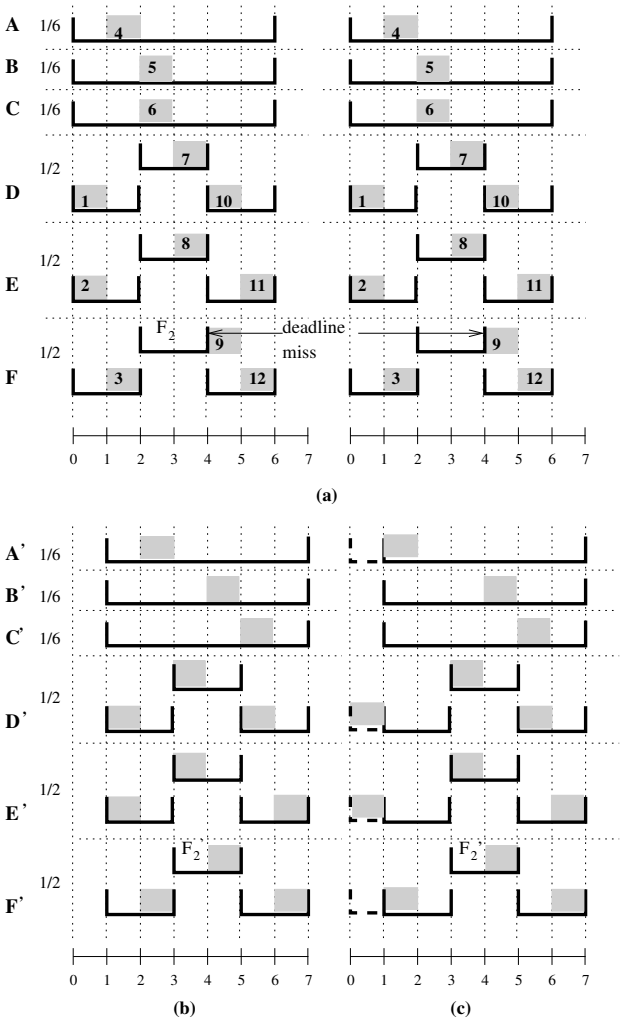

Figure 6. **(a)** A PD^B schedule $\mathcal{S}_B$ for a task system τ^B comprised of three tasks of weight $1/6$ each and three other tasks of weight $1/2$ each. The ranks of the subtasks in τ^B under $\mathcal{S}_B$ are as indicated. **(b)** A PD^2 schedule $\mathcal{S}$ for τ obtained from τ^B by shifting the IS-window of each subtask right by one slot. τ is 0-compliant to τ^B and $\mathcal{S}$ is 0-compliant to $\mathcal{S}_B$. **(c)** A task system τ' that is 4-compliant to τ^B, and a schedule for τ' that is 4-compliant to $\mathcal{S}_B$.

The proof of this lemma is somewhat tedious and is available in an appendix.

By our definition of $\mathcal{S}_B$ and k-compliance, and Lemma 6, we have the following theorem.

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

Theorems 1 and 2 imply 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 and have determined that relaxing this requirement worsens the tardiness of Pfair algorithms by less than one quantum only. This result enables the use of a relaxed Pfair scheduling model for providing both hard and 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, that which requires the execution cost of each task to be expressed as an integral multiple of the maximum size of a quantum.

References

[1] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proc. of the 12th Euromicro Conference on Real-time Systems*, pages 35–43, June 2000.

[2] 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.

[3] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.

[4] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proc. of the 15th Euromicro Conference on Real-time Systems*, pages 33–40, July 2003.

[5] S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*. To appear.

[6] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[7] 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.

[8] 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.

[9] U. Devi and J. Anderson. Desynchronized Pfair scheduling on multiprocessors(extended version). Available at http://www.cs.unc.edu/~anderson/papers.html, October 2004.

[10] P. Holman. *On the Implementation of Pfair Scheduled Multiprocessor Systems*. PhD thesis, University of North Carolina at Chapel Hill, Aug. 2004.

[11] 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.

[12] 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.

[13] J.M. Lopez, M. Garcia, J.L. Diaz, and D.F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. of the 12th Euromicro Conference on Real-time Systems*, pages 25–34, June 2000.

[14] 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.

Appendix: Proof of Lemma 6

Lemma 6 *There exist a task system τ^{k+1} that is $(k+1)$-compliant to τ^B, and a schedule $\mathcal{S}_{k+1}$ for τ^{k+1} that is $(k+1)$-compliant to $\mathcal{S}_B$.*

Proof: (In this proof, we let the eligibility time function take a task system as a second parameter. Hence, $e(T_i, \tau^k)$ gives the eligibility time of T_i in τ^k. If the task system under consideration is unambiguous from the context, then the second parameter will not be used.)

For each subtask T_i in τ^B, its image in τ^k is called T_i'. If $h > 0$ processors are idle in slot t in

a schedule, then we say that there are h *holes* in t in that schedule.

Let T_i be the subtask with rank $k+1$ in the total order described above, and let T_i be scheduled at time t in $\mathcal{S}_B$, *i.e.*, let

$$\mathcal{S}_B(T_i) = t. \tag{25}$$

Let τ^{k+1} be the task system obtained from τ^k by decreasing the eligibility time of T_i' to $e(T_i)$. Therefore, we have the following.

$$e(T_i', \tau^k) = e(T_i) + 1 \ \wedge \ e(T_i', \tau^{k+1}) = e(T_i) \tag{26}$$

Then, τ^{k+1} is $(k+1)$-compliant to τ^B. If T_i' is scheduled at t in $\mathcal{S}_k$, then take $\mathcal{S}_{k+1}$ to be $\mathcal{S}_k$. Otherwise, let

$$\mathcal{S}_k(T_i') = t'. \tag{27}$$

Let T_h denote the predecessor of T_i, if one exists. Then, we have $\mathcal{S}_B(T_h) < t$, and hence, $rank(T_h) < k+1$ holds. Therefore, because τ^k and $\mathcal{S}_k$ are k-compliant, we have

$$\mathcal{S}_k(T_h') = \mathcal{S}_B(T_h) < t. \tag{28}$$

We first claim the following.

Claim 4 $t' > t$.

Proof: The proof is quite simple, but omitted due to space constraints.

We next claim the following.

Claim 5 *There either is a hole in t in $\mathcal{S}_k$ or there exists a subtask U_j' scheduled at t in $\mathcal{S}_k$ such that $e(U_j') \leq t \ \wedge \ T_i' \preceq U_j'$ holds, and U_j is not scheduled at t in $\mathcal{S}_B$.*

Proof: Contrary to the claim, assume that there is no hole in t in $\mathcal{S}_k$ and that for every subtask V_k' scheduled at t in $\mathcal{S}_k$ such that V_k is not scheduled at t in $\mathcal{S}_B$, $V_k' \prec T_i'$ holds. Let U_j' be one such subtask. Then, the following holds.

$$\mathcal{S}_k(U_j') = t \wedge \mathcal{S}_B(U_j) \neq t \wedge U_j \prec T_i \wedge U_j' \prec T_i' \tag{29}$$

Because U_j and U_j' are scheduled in different slots in their respective schedules, $rank(U_j) > k+1$ holds. Therefore, because τ^{k+1} is $(k+1)$-compliant, $e(U_j') = e(U_j) + 1$ holds. Because $\mathcal{S}_k(U_j') = t$ holds,

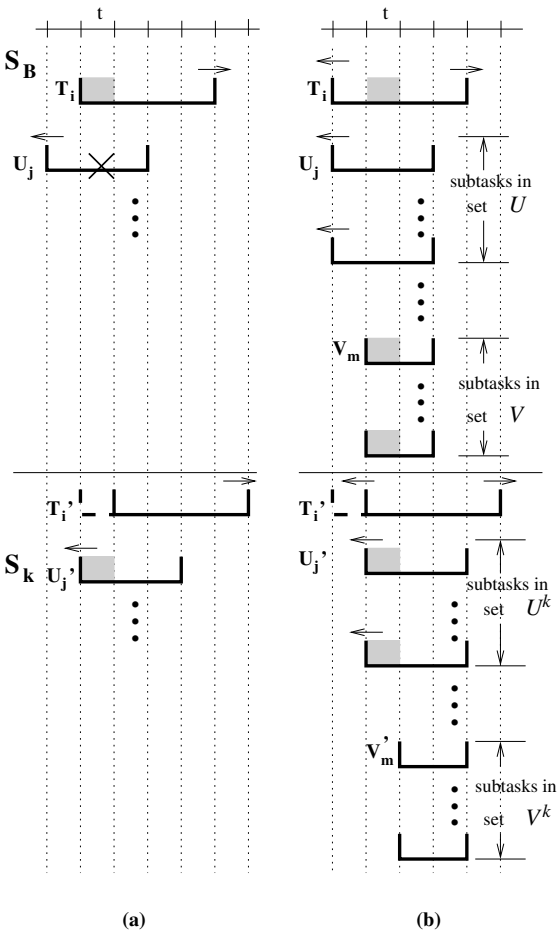

Figure 7. Lemma 5. $rank(U_j) > k+1$ holds. Thus, $e(U_j') = e(U_j) + 1$. An arrow over a window end point indicates that the end point could extend in that direction. **(a)** Case 1. $T_i \in EB(t)$. Hence, T_i cannot block a higher-priority subtask. Hence, if U_j exists, PD^B would schedule it prior to T_i. **(b)** Case 2. In this case, subtasks in set $\mathcal{V}$, which are in $EB(t)$ are scheduled at t in $\mathcal{S}_B$, while their images in τ^k are not scheduled at t in $\mathcal{S}_k$. For each subtask V_m in $\mathcal{V}$, $e(V_m') = e(V_m) + 1$ holds. Subtasks in set $\mathcal{U}$ are scheduled at t in $\mathcal{S}_k$, and $|\mathcal{V}| \geq |\mathcal{U}|$ holds. Thus, there should exist at least one a subtask W_l such that $\mathcal{S}_B(W_l) \neq t \ \wedge \ \mathcal{S}_k(W_l') = t \ \wedge \ T_i \preceq W_l$ holds.

$e(U_j') \leq t$, and hence, $e(U_j) \leq t-1$ holds. This can be generalized as follows.

$$(\forall U_j : \mathcal{S}_B(U_j) \neq t \ \wedge \ \mathcal{S}_k(U_j') = t :: e(U_j') \leq t \ \wedge e(U_j) \leq t-1) \tag{30}$$

Therefore, U_j is either in $DB(t)$ or $PB(t)$ in $\mathcal{S}_B$. By

the the priority definition of PD^B in Table 1, a subtask in $DB(t)$ is not blocked in slot t under PD^B. Therefore, if $U_j \in DB(t)$ in $\mathcal{S}_B$, then $U_j \prec T_i$ cannot hold, which contradicts (29). Hence, $U_j \in PB(t)$. Because T_i is scheduled at t in $\mathcal{S}_B$, $e(T_i) \leq t$ holds. We consider two cases.

Case 1: $e(T_i', \tau^{k+1}) = e(T_i) = t$. This case is illustrated in Fig. 7(a). In this case, T_i is in $EB(t)$ in $\mathcal{S}_B$. Let p denote the number of subtasks in $|PB(t)|$ before any scheduling decisions are made for slot t. Since U_j is in $PB(t)$, by the priority definition of PD^B in Table 1, $T_i \sqsubseteq U_j$ can hold only in the first $M-p$ scheduling decision. Hence, T_i is scheduled in one of the first $M-p$ scheduling decisions for t in $\mathcal{S}_B$. Thus, by Claim 2, no subtask that remains in $EB(t)$ or $DB(t)$ during the last p scheduling decisions for the same slot can have a higher priority than U_j. Hence, PD^B would have scheduled U_j in t in one of the final p scheduling decisions, contradicting (29).

Case 2: $e(T_i', \tau^{k+1}) = e(T_i) < t$. This case is illustrated in Fig. 7(b). Let $\mathcal{U}$ denote the set of all subtasks that are eligible before t, ready at t, are not scheduled at t in $\mathcal{S}_B$, and have a higher priority than T_i. Then, by Lemma 2, there exists a set $\mathcal{V}$ of subtasks that are in $EB(t)$ in $\mathcal{S}_B$ such that

$$|\mathcal{V}| \geq |\mathcal{U}| \tag{31}$$

holds and every subtask in $\mathcal{V}$ is scheduled at t in $\mathcal{S}_B$ and has equal or a higher priority than every subtask in $\mathcal{U}$. By the same lemma, T_i is scheduled in slot t before every subtask in $\mathcal{V}$. Therefore, the rank of every subtask in $\mathcal{V}$ is greater than that of T, *i.e.*, exceeds $k+1$. That is, we have $\mathcal{U}$ and $\mathcal{V}$ as follows.

$$\begin{aligned}\mathcal{U} &= \{U_j \mid e(U_j) \leq t-1 \ \wedge\ U_j \text{ is ready at } t \text{ in } \mathcal{S}_B \\ &\quad \wedge\ \mathcal{S}_B(U_j) > t \ \wedge\ U_j \prec T_i\} \qquad (32)\\ \mathcal{V} &= \{V_k \mid e(V_k) = t \ \wedge\ rank(V_k) > k+1 \ \wedge \\ &\quad \mathcal{S}_B(V_k) = t \ \wedge\ (\forall U_j \in \mathcal{U} : V_k \preceq U_j)\} \qquad (33)\end{aligned}$$

Let V_k be any subtask in $\mathcal{V}$. Then, because $rank(V_k) > k+1$ and $e(V_k) = t$ hold, and τ^k is $(k+1)$-compliant to τ^B, $e(V_k', \tau^{k+1}) = e(V_k) + 1 = t+1$ holds. Therefore, no subtask in $\mathcal{V}$ is scheduled in slot t in $\mathcal{S}_k$.

Let $\mathcal{T}^B$ and $\mathcal{T}^k$ denote the set of all subtasks that are scheduled at t in $\mathcal{S}_B$ and $\mathcal{S}_k$, respectively. Let $\mathcal{U}^k$ denote the set of all subtasks U_j' in τ^k, where U_j is in $\mathcal{U}$. ($\mathcal{U}^k$ is the set of the images of subtasks in $\mathcal{U}$.) We next define a one-to-one mapping from subtasks in $\mathcal{T}^B$ to those in $\mathcal{T}^k$. Because there is no hole in t in $\mathcal{S}_k$, such a mapping is possible. Let the preimage of every subtask in $\mathcal{U}^k$ that is in $\mathcal{T}^k$, be a subtask in $\mathcal{V}$. (If a' is the image of a under some mapping, then a is the preimage of a'.) By (31), such a mapping is possible. Note that, by (32), no subtask in $\mathcal{U}$ is in $\mathcal{T}^B$, and by (33), every subtask in $\mathcal{V}$ is in $\mathcal{T}^B$. Fig. 7(b) shows an example. Apart from subtasks in $\mathcal{V}$, at least subtask T_i is in $\mathcal{T}^B$. Because every subtask in $\mathcal{U}^k$ that is in $\mathcal{T}^k$ is the image of some subtask in $\mathcal{V}$, it follows that T_i should be mapped onto a subtask W_l' that is not in $\mathcal{U}^k$ such that $\mathcal{S}_B W_l > t$ holds. ($\mathcal{S}_B W_l > t$ holds because $\mathcal{S}_k$ is k-compliant.) Therefore, $rank(W_l) > k+1$, and hence, $e(W_l') = e(W_l) + 1$ holds. Because $\mathcal{S}_k(W_l') = t$ holds, we have $e(W_l) \leq t-1$. Therefore, since W_l is not in $\mathcal{U}$, by (32), we have $T_i \preceq W_l$. □

By Claim 5, (27), Claim 4, and (26) one of the following holds.

(C1) There is a hole at t in $\mathcal{S}_k$.

(C2) $e(T_i', \tau^{k+1}) = e(T_i) < t$ holds and a subtask U_j' with the same priority as T_i' under PD^2 is scheduled at t in $\mathcal{S}_k$, but U_j is not scheduled at t in $\mathcal{S}_B$. (Claim 5 actually implies that $T_i' \preceq U_j'$ holds. However, by (26), $e(T_i', \tau^k) \leq t$ holds, and by (28), T_h' is scheduled before t in $\mathcal{S}_k$. Therefore, if $T_i' \preceq U_j'$ holds, then PD^2 would schedule T_i' at t in $\mathcal{S}_k$, in preference to U_j'.)

(C3) $e(T_i') = e(T_i) = t$ holds and a subtask U_j' such that $T_i' \preceq U_j'$ holds is scheduled at t in $\mathcal{S}_k$, but U_j is not scheduled at t in $\mathcal{S}_B$.

If there is a hole in slot t in $\mathcal{S}_k$, then we can easily schedule T_i' in $\mathcal{S}_k$ and the resulting schedule will be $(k+1)$-compliant. Note that because $\mathcal{S}_k$ is k-compliant, T_i's predecessor is guaranteed not to be scheduled at t in $\mathcal{S}_k$. Therefore, for the rest of this proof, assume that there is no hole in t in $\mathcal{S}_k$.

We next show that we can construct $\mathcal{S}_{k+1}$ from $\mathcal{S}_k$, by moving T_i' into slot t and U_j' (defined in (C2) or (C3)) out. Because there are no holes in t in $\mathcal{S}_k$, we have the following.

(H) There are no holes in t in $\mathcal{S}_{k+1}$.

Let ρ denote the set of all subtasks in τ^{k+1} with rank higher than k. Let $\mathcal{S}'_k$ and $\mathcal{S}'_{k+1}$ be the schedules for ρ obtained from $\mathcal{S}_k$ and $\mathcal{S}_{k+1}$, respectively, by removing all subtasks not in ρ (*i.e.*, subtasks with rank at most k) and letting the remaining subtasks be scheduled in the same slots as in $\mathcal{S}_k$ and $\mathcal{S}_{k+1}$. Then, because $\mathcal{S}_k$ is k-compliant for τ^k, all subtasks in ρ, except T'_i if (C3) holds, are scheduled by their PD^2 priority in $\mathcal{S}_k$, and hence, in $\mathcal{S}'_k$. Also, no subtask in ρ misses its deadline in $\mathcal{S}_k$ or $\mathcal{S}'_k$. Let r denote the number of subtasks with rank greater than k that are scheduled in slot t in $\mathcal{S}_k$. Thus, we have the following:

(R1) $\mathcal{S}'_k$ is a valid schedule for ρ in which **(i)** no subtask is scheduled in the first $t-1$ slots, **(ii)** only $r \leq M$ subtasks are scheduled in slot t, and (iii) every allocation except possibly that of T'_i is in accordance with PD^2.

If (C2) holds, then $T'_i \preceq U'_j$ holds. If (C3) holds, then $T_i \in EB(t)$ in $\mathcal{S}_B$. Therefore, from the priority definition for PD^B, it can be seen that PD^B would schedule T_i at t in $\mathcal{S}_B$ in preference to U_j only if $T_i \preceq U_j$ holds. If $T_i \preceq U_j$ holds, then by the construction of τ^k, $T'_i \preceq U'_j$ holds. Thus, if either (C2) or (C3) holds, then we have $T'_i \preceq U'_j$, and hence, all subtasks in ρ are scheduled by their PD^2 priorities in $\mathcal{S}'_{k+1}$. Let r denote the number of subtasks with rank greater than k that are scheduled in slot t in $\mathcal{S}_k$. Then, by (H), r subtasks are scheduled in slot t in $\mathcal{S}'_{k+1}$.

Thus, we have the following:

(R2) $\mathcal{S}'_{k+1}$ is a schedule for ρ in which (i) and (ii) from (R1) hold, and (iii) every allocation is in accordance with PD^2.

Because (R1) and (R2) hold, it can be shown that $\mathcal{S}'_{k+1}$ is also a *valid* schedule for ρ. The proof is essentially the same as the proof that establishes the optimality of PD^2 [14]. ■