Yang ML, Lei H, Liao Y *et al.* Improved blocking time analysis and evaluation for the multiprocessor priority ceiling protocol. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(6): 1003–1013 Nov. 2014. DOI 10.1007/s11390-014-1485-y

Improved Blocking Time Analysis and Evaluation for the Multiprocessor Priority Ceiling Protocol

Mao-Lin Yang[1] (杨茂林), *Student Member*, *IEEE*, *Hang Lei*[1] (雷 航), *Member*, *CCF*
Yong Liao[1] (廖 勇), *Member*, *CCF*, and Furkan Rabee[2]

[1]*School of Information and Software Engineering, University of Electronic Science and Technology of China Chengdu 611731, China*

[2]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731 China*

E-mail: maolyang@gmail.com; {hlei, liaoyong}@uestc.edu.cn; forkanr@yahoo.com

Received September 8, 2013; revised July 9, 2014.

Abstract The Multiprocessor Priority Ceiling Protocol (MPCP) is a classic suspension-based real-time locking protocol for partitioned fixed-priority (P-FP) scheduling. However, existing blocking time analysis is pessimistic under the P-FP + MPCP scheduling, which negatively impacts the schedulability for real-time tasks. In this paper, we model each task as an alternating sequence of normal and critical sections, and use both the best-case execution time (BCET) and the worst-case execution time (WCET) to describe the execution requirement for each section. Based on this model, a novel analysis is proposed to bound shared resource requests. This analysis uses BCET to derive the lower bound on the inter-arrival time for shared resource requests, and uses WCET to obtain the upper bound on the execution time of a task on critical sections during an arbitrary time interval of Δt. Based on this analysis, improved blocking analysis and its associated worst-case response time (WCRT) analysis are proposed for P-FP + MPCP scheduling. Schedulability experiments indicate that the proposed method outperforms the existing methods and improves the schedulability significantly.

Keywords real-time scheduling, multiprocessor scheduling, locking protocol, blocking analysis, worst-case response time

1 Introduction

In real-time systems, scheduling and locking mechanisms, supported by rigorous analysis techniques, are vital to ensuring that timing constraints of tasks will not be violated during system operation. The worst-case response time (WCRT) of a task, which is a widely used criterion for schedulability analysis, is determined by 1) the worst-case execution (and self-suspension) demand, 2) the worst-case blocking due to resource contentions, and 3) the maximum preemption delays. However, most existing analytical methods for multiprocessor locking protocols only use coarse-grained approaches to bound worst-case blockings, and thus are pessimistic and may lead to poor schedulability.

In multicore systems, tasks are usually scheduled by partitioned or global scheduling. Under partitioned scheduling, tasks are statically allocated among processing cores; while under global scheduling, tasks are dynamically allocated at runtime (i.e., tasks may migrate from one core to another during execution). Global scheduling yields higher system utilization theoretically, but may incur non-trivial run-time overheads[1-2]. In contrast, partitioned scheduling incurs less run-time overhead and is simpler to implement, and thus is a desirable choice in practice.

Once a task has locked a mutually exclusive resource (e.g., shared memory, I/O device, and critical data), it must finish executing the corresponding critical section before another task may get the lock to that resource (there are more approaches for synchronization that can be used such as wait free and block free, and we focus on mutually exclusive resources in the paper). In that case, the blocked task may busy-wait (i.e., spin) or suspend when waiting to acquire the shared resource. Intuitively, spinning consumes processor cycles, though it

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant No. 61103041, the National High Technology Research and Development 863 Program of China under Grant No. 2012AA010904, the Fundamental Research Funds for the Central Universities of China under Grant No. ZYGX2012J070, the Huawei Technology Foundation under Grant No. IRP-2012-02-07, and the Excellent Ph.D. Student Academic Support Program of UESTC under Grant No. YBXSZC20131028.

is more efficient in terms of runtime overheads[3]; while suspension is more efficient with respect to schedulability in theory, because when a task is suspended, another task can be scheduled. Empirical studies[2-4] show that suspension-based protocols are preferable especially when critical sections are long.

Another key design issue that any locking protocol must address is how to manage the order of blocked tasks. With regard to suspension-based protocols, MPCP[5] uses priority queues to order conflicting requests, while the Flexible Multiprocessor Locking Protocol (FMLP)[6] employs FIFO queues. Recently, a two-phase locking protocol under partitioned scheduling was proposed[7], which uses a global FIFO queue and several per-processor priority queues. Priority queuing, fitting for priority scheduling in nature, may lead to starvations for lower priority tasks. In contrast, FIFO queuing is simple to implement, but may lead to increased priority inversions for higher priority tasks. Broadly speaking, neither choice is categorically superior to others.

In this paper, we focus on the worst-case blocking analysis for MPCP, mainly due to the fact that: 1) priority queuing adopted in MPCP is a natural fit for fixed-priority scheduling; besides, MPCP is a direct extension of the well-studied Priority Ceiling Protocol (PCP)[8]; 2) existing blocking analysis for MPCP is pessimistic, leaving large space for improvement. Firstly, we extend the task model in [9] by using both the best-case execution time (BCET) and the worst-case execution time (WCET) to describe the lower and the upper bounds on task execution requirement. Secondly, we derive an upper bound on time, for which a task will take to execute particular critical sections during a time interval Δt. Then, we tighten the bounds on worst-case blockings. Finally, we evaluate the performance of the proposed analysis through comparative schedulability experiments.

The rest of this paper is organized as follows. Background and related work are discussed in Section 2. Section 3 proposes bounds on shared resource requests. Worst-case blockings and WCRT are analyzed in Section 4. Schedulability experiments are conducted in Section 5. The last section concludes the paper.

2 Background and Related Work

2.1 Task Model

A set of nt sporadic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_{nt}\}$ are scheduled on a multicore processor that contains m processing cores $p_1, p_2, \ldots, p_m$, and share a set of q serially reusable resources $\Phi = \{\rho_1, \rho_2, \ldots, \rho_q\}$. A sporadic task potentially generates infinite jobs. The v-th job of τ_i is denoted as $J_{i,v}$, and an arbitrary job of τ_i is also denoted as $J_{i,*}$. Each shared resource can only be held by at most one job at any time. We assume in this paper that tasks will not make nested requests[①] (in case of nested requests, we only consider the outermost ones). The priority assigned to τ_i is denoted by π_i. We further assume that tasks are indexed by decreasing order of priorities (i.e., the priority of τ_i is higher than that of τ_j if $i < j$), and all jobs share the same priority as that assigned to the task (i.e., $J_{i,x}$ has the same priority as that of $J_{i,y}$). $J_{i,l}$ is released at time $r_{i,l}$ and finishes at time $f_{i,l}$, and it is said to be pending during $(r_{i,l}, f_{i,l})$. The WCRT of τ_i is defined as $R_i = \max_{\forall l}\ (f_{i,l} - r_{i,l})$.

Following the work of Lakshmana *et al.*[9], we consider each task to be an alternating sequence of normal and critical section execution segments[②]. Let τ_i^j and τ_i^{j*} denote the j-th normal and the j-th critical section of τ_i respectively. Then, τ_i can be represented as $(\tau_i^1, \tau_i^{1*}, \tau_i^2, \tau_i^{2*}, \ldots, \tau_i^{S_i-1*}, \tau_i^{S_i})$, where S_i is the number of normal sections of τ_i. Let BC_i^j (BC_i^{j*}) and WC_i^j (WC_i^{j*}) be the BCET and the WCET of τ_i^j (τ_i^{j*}) respectively. Wherein, the WCET of each section is assumed to be known in advance. $BC_i^j \in [0, WC_i^j]$, and $BC_i^{j*} \in [0, WC_i^{j*}]$ for all sections. Let T_i be the minimum release time separation, or period, of τ_i, and the WCET of τ_i is denoted by WC_i (WC_i is not necessarily equal to $\sum_{j=1}^{S_i-1}(WC_i^j + WC_i^{j*}) + WC_i^{S_i*}$ because the task model does not strictly conform with the control flow of the task). For simplicity, we assume implicit deadlines, and do not consider release jitters and end-to-end delays in this paper.

2.2 Definition of Blocking

In uniprocessor systems, a waiting job with an unsatisfied request is not blocked if a higher-priority job is scheduled, because the delay to the lower priority job overlaps with higher priority work and the WCRTs of all tasks will remain unchanged. In contrast, a job is considered to be blocked if it is delayed due to priority inversions. Brandenburg *et al.*[7] discussed the notion of blocking in multiprocessor systems that a job incurs blocking if the completion of that job is delayed and such delay is not caused by higher priority jobs. Moreover, some delays unrelated to resource sharing (e.g., deferral delays under limited preemption scheduling[10-11]) are also referred to as blockings.

[①]The MPCP does not support fine-grained nested requests originally. However, nested requests can be supported by group locks (i.e., logically consider a nested request as a single request).

[②]This model does not necessarily conform with the exact control flow. It just describes the time interval between share resource requests (critical sections) within a task.

Unlike in uniprocessor systems or globally scheduled multiprocessors, jobs can be blocked by their local or remote jobs[③] under P-FP scheduling. For example, a job may be delayed due to resource contentions by jobs (even those with higher priorities) on other processing cores. We consider such delay as a type of blocking in this paper. To avoid ambiguity, the notation of blocking under P-FP is defined as follows.

Definition 1. *A job $J_{i,*}$ is blocked by another job $J_{x,*}$ if* 1) *$J_{x,*}$ is a lower priority local job of $J_{i,*}$, and $J_{x,*}$ is scheduled while $J_{i,*}$ is pending*; *or* 2) *$J_{x,*}$ is a remote job of $J_{i,*}$, and it has locked the global resource that $J_{i,*}$ is waiting for.*

Blocking events can be classified into local and remote blockings according to 1) and 2) in Definition 1, respectively. Local blocking is caused by local jobs due to priority inversion, while remote blocking is caused by remote jobs due to acquisition delays.

2.3 MPCP

MPCP is proposed on the basis of direct resource access, namely, each task accesses shared resources from the processor where it is assigned. Logically, shared resources can be classified into local and global resources under P-FP scheduling. Local resources are shared only by tasks assigned to the same core, and global resources can be shared by tasks deployed on different cores. Critical sections corresponding to global or local resources are called global critical sections (GCSs) or local critical sections (LCSs).

Under MPCP, a job $J_{i,*}$ blocked on a global resource ρ_k is suspended and is inserted to a priority queue on ρ_k. During the time $J_{i,*}$ is suspended, lower priority jobs are allowed to execute and lock resources. When $J_{i,*}$ has locked ρ_k, it will execute the corresponding GCS at a priority ceiling of $\Omega_k = \pi_{\text{base}} + \pi_k$, where π_{base} is a priority level greater than any base priority of tasks in the system, and π_k is the highest base priority of tasks that can lock ρ_k. Job $J_{i,*}$ within a GCS can be preempted by another job within another GCS if it has a priority higher than that of $J_{i,*}$'s GCS. Further, if the queue on ρ_k is not empty when $J_{i,*}$ attempts to release ρ_k, the head of the queue is granted to lock ρ_k, otherwise, ρ_k is released. For local resources on each core, conflicting accesses are mediated by PCP[7].

2.4 Worst-Case Blocking Analysis

Blocking analysis for the MPCP is quite intricate and error-prone in that tasks may encounter many blocking penalties, such as back-to-back executions, multiple priority inversions, and transitive interferences[5,9]. In most prior analysis, only coarse-grained exposition was used. Rajkumar[5] classified the task blocking time into five items, and proposed the first upper bound on task blocking time by means of summing the five items together. This analysis is pessimistic because there are overlaps between individual blocking factors. Further, the maximum, not individual, critical section length of each task is used during analysis. This assumption is also pessimistic, because the ratio of the maximum to the minimum critical section length can be quite large in many cases. Lakshmanan *et al.*[9] applied the classic response time analysis to bound the remote blocking time for each individual request to global resources, and proposed a WCRT analysis for P-FP + MPCP based on the deferrable task model. Schliecker *et al.*[12] improved the classic blocking analysis using a sophisticated event stream model. This model exploits the minimum distance between any two shared resource requests to capture the shared resource load of a task during any time interval, based on which the maximum blocking can be expected to reduce. More recently, Brandenburg[13] developed a linear-programming analysis for P-FP scheduling. The key insight is to reduce repeated blockings that lower priority requests suffered from remote higher priority ones.

2.5 Worst-Case Response Time for P-FP + MPCP Scheduling

In P-FP + MPCP scheduled systems, tasks suspend when blocked by remote tasks. The feasibility problem in such scenario can be analogous to that of scheduling self-suspending tasks on uniprocessors, which is proved to be NP-hard[14]. Lakshmanan *et al.*[9] provided a WCRT analysis for MPCP as follows.

$$R_i^{z+1} = WC_i + B_i^r + B_i^l + \sum_{h<i \wedge \tau_h \in \Gamma_i^{\text{local}}} \left\lceil \frac{R_i^z + B_h^r}{T_h} \right\rceil \times WC_h,$$

where, B_i^r and B_i^l are the remote and the local blocking time of τ_i respectively, and Γ_i^{local} denotes the local tasks of τ_i. The iteration starts with $R_i^0 = 0$ and ends when $R_i^{z+1} = R_i^z$. τ_i is considered to be schedulable if $R_i \leqslant T_i$. This iteration can also be ended early if $R_i^{z+1} > T_i$, in that case τ_i is considered to be unschedulable.

3 Bounding the Shared Resource Requests

In this section, we exploit our task model to present tighter bounds on shared resource requests. Let $\rho(\tau_i^{j*})$ be the shared resource corresponding to τ_i^{j*}, e.g., $\rho(\tau_i^{2*})$

[③] $J_{x,*}$ is $J_{i,*}$'s local (or remote) job if it is assigned to the same (or a different) core to which $J_{i,*}$ is assigned.

is ρ_k in Fig.1. Suppose that job $J_{i,*}$ performs a maximum of $N_{i,k}$ requests to ρ_k. Let $\tau_{i,k}^{x*}$ denote the critical section where $J_{i,*}$ issues its x-th request to ρ_k, and let $S_{i,k}^x$ denote the index of the critical section corresponding to $\tau_{i,k}^{x*}$. For example in Fig.1, $N_{i,k}=2$ and $N_{i,h}=3$, the fourth critical section of τ_i is the second one to protect ρ_k, therefore $S_{i,k}^2=4$. To maintain consistency, we define $BC_i^0 = BC_i^{0*} = WC_i^0 = WC_i^{0*} = 0$.

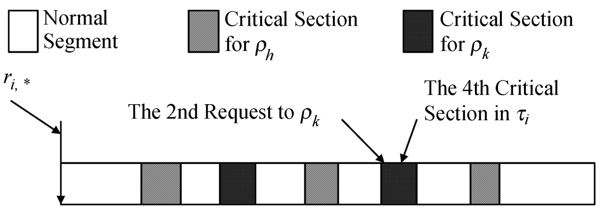

Normal Segment
Critical Section for ρ_h
Critical Section for ρ_k
$r_{i,*}$
The 2nd Request to ρ_k
The 4th Critical Section in τ_i

Fig.1. A task is modeled as an alternating sequence of normal and critical sections.

In cases when τ_i executes the critical sections protecting ρ_k for the maximum possible time, then the minimum time between $r_{i,v}$ and the instant when $J_{i,v}$ issues the x-th ($1 \leqslant x \leqslant N_{i,k}$) request to ρ_k can be bounded by $d_{i,k}^x$ as following:

$$d_{i,k}^x = \sum_{\forall j\in[0,S_{i,k}^x]} BC_i^j + \sum_{\substack{\forall j\in[0,S_{i,k}^x-1] \\ \wedge\rho(\tau_i^{j*})\neq\rho_k}} BC_i^{j*} + \sum_{\substack{\forall j\in[0,S_{i,k}^x-1] \\ \wedge\rho(\tau_i^{j*})=\rho_k}} WC_i^{j*}.$$

Similarly, the minimum time between the instant when $J_{i,v}$ issues the x-th ($1 \leqslant x \leqslant N_{i,k}$) request to ρ_k and $f_{i,v}$ can be bounded by

$$e_{i,k}^x = \sum_{\substack{\forall j\in[S_{i,k}^x,N_{i,k}] \\ \wedge\rho(\tau_i^{j*})=\rho_k}} WC_i^{j*} + \sum_{\forall j\in[S_{i,k}^x+1,S_i]} BC_i^j + \sum_{\substack{\forall j\in(S_{i,k}^x,S_i-1] \\ \wedge\rho(\tau_i^{j*})\neq\rho_k}} BC_i^{j*}.$$

Lemma 1. *Suppose τ_i executes the critical sections protecting ρ_k for the maximum possible time and a job of τ_i (e.g., $J_{i,v}$) is about to issue the x-th ($1 \leqslant x \leqslant N_{i,k}$) request to ρ_k (i.e., $\tau_{i,k}^{x*}$) at t_s, then it will take at least $\delta_{i,k}^x(n)$ for τ_i to issue the next n requests (including that corresponds to $\tau_{i,k}^{x*}$) to ρ_k.*

$$\delta_{i,k}^x(n) = d_{i,k}^{x+n-1} - d_{i,k}^x, \tag{1}$$

if $x+n-1 \leqslant N_{i,k}$; *and*

$$\delta_{i,k}^x(n) = e_{i,k}^x + T_i - R_i + d_{i,k}^{n-N_{i,k}+x-1}, \tag{2}$$

if $N_{i,k} < x+n-1 \leqslant 2N_{i,k}$; *and*

$$\delta_{i,k}^x(n) = e_{i,k}^x + (\lceil (n+x-1)/N_{i,k} \rceil - 1) \times T_i - R_i + d_{i,k}^{f(x)}, \tag{3}$$

if $x+n-1 > 2N_{i,k}$, *where*

$$f(x) = n+x-1-N_{i,k} \times (\lceil (n+x-1)/N_{i,k} \rceil - 1).$$

Proof. Suppose that τ_i has issued the n-th request to ρ_k at $t_s + \Delta t$. We prove this lemma as following.

Firstly, if $x+n-1 \leqslant N_{i,k}$, the following n requests will be issued within one job, as shown in Fig.2(a). Since $J_{i,v}$ has to execute for at least $d_{i,k}^{x+n-1} - d_{i,k}^x$ before it issues the n-th request to ρ_k, $\Delta t = d_{i,k}^{x+n-1} - d_{i,k}^x$. We prove (1).

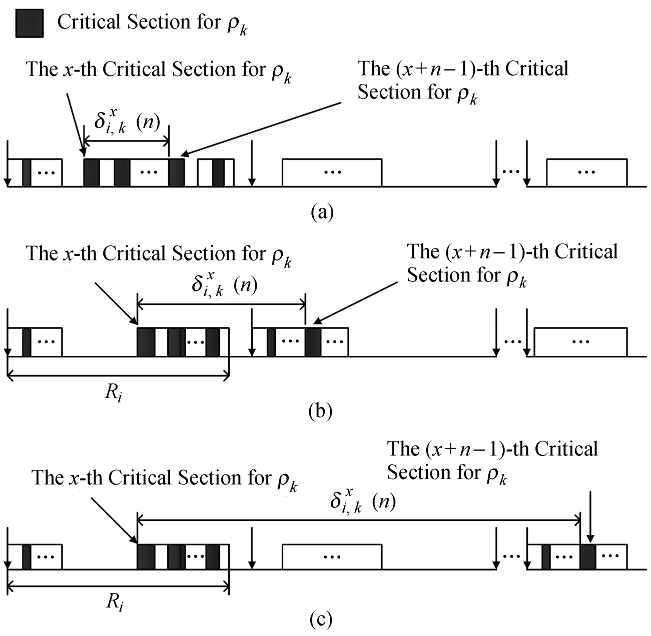

Critical Section for ρ_k
The x-th Critical Section for ρ_k
The $(x+n-1)$-th Critical Section for ρ_k
$\delta_{i,k}^x(n)$
(a)
The x-th Critical Section for ρ_k
The $(x+n-1)$-th Critical Section for ρ_k
$\delta_{i,k}^x(n)$
R_i
(b)
The $(x+n-1)$-th Critical Section for ρ_k
The x-th Critical Section for ρ_k
$\delta_{i,k}^x(n)$
R_i
(c)

Fig.2. Illustrative timeliness of τ_i. The n requests are issued (a) within one job, (b) by two successive jobs, and (c) by at least three jobs.

Secondly, if $N_{i,k} < x+n-1 \leqslant 2N_{i,k}$, the following n requests will be issued by two consecutive jobs, $J_{i,v}$ and $J_{i,v+1}$, as shown in Fig.2(b). Wherein, $J_{i,v}$ issues $N_{i,k}-x+1$ requests to ρ_k after t_s, and $J_{i,v+1}$ produces the remaining requests. Correspondingly, Δt can be divided into three sections: let Δt_1 be the time interval between t_s and $f_{i,v}$, Δt_2 be the time interval between $f_{i,v}$ and $r_{i,v+1}$, and Δt_3 be the time interval from $r_{i,v+1}$ to the instant $J_{i,v+1}$ issues its $(n-N_{i,k}+x-1)$-th request to ρ_k. To minimize the respective sections, the following conditions must be satisfied: 1) $J_{i,v}$ and

$J_{i,v+1}$ are not interrupted during Δt_1 and Δt_3 respectively; 2) the response time of $J_{i,v}$ is equal to R_i; 3) $J_{i,v+1}$ starts to execute at $r_{i,v+1}$ (when it is released).

Conditions 1) and 3) guarantee that Δt_1 and Δt_3 are minimized, wherein, $\Delta t_1 = e^x_{i,k}$, and $\Delta t_3 = d^{n-(N_{i,k}-x-1)}_{i,k}$. Condition 2) guarantees that Δt_2 is minimized, because $\Delta t_2 = T_i - (f_{i,v} - r_{i,v})$ and $R_i = \max_{\forall v}(f_{i,v} - r_{i,v})$. We prove (2) by summing up the above three items.

Thirdly, if $x + n - 1 > 2N_{i,k}$, the following n requests will be issued by at least three jobs, as shown in Fig.2(c). Let $J_{i,z}$ denote the last such job, Δt can also be divided into three sections: Δt_1^* denotes the time interval between t_s and $r_{i,v+1}$, Δt_2^* denotes the time interval between $r_{i,v+1}$ and $r_{i,z}$, and Δt_3^* denotes the time interval from $r_{i,z}$ to the instant $J_{i,z}$ issues the n-th request to ρ_k. Based on what discussed above, Δt_1^* can be minimized to $\Delta t_1 + \Delta t_2 = e^x_{i,k} + T_i - R_i$. The remaining requests involve $\lceil (n - N_{i,k} + x - 1)/N_{i,k}\rceil = \lceil (n + x - 1)/N_{i,k}\rceil - 1$ jobs, which spans $\lceil (n + x - 1)/N_{i,k}\rceil - 2$ complete periods, therefore $\Delta t_2^* = T_i \times \lceil (n + x - 1)/N_{i,k}\rceil - 2$. Finally, $n-(N_{i,k}-x+1)-N_{i,k}\times(\lceil (n+x-1)/N_{i,k}\rceil - 2) = f(x)$ requests are remained to be issued by $J_{i,z}$, therefore, Δt_3^* can be minimized to be $d^{f(x)}_{i,k}$. Summing up the above three items, we prove (3). □

Lemma 1 provides a lower bound on the time between any n requests of τ_i to ρ_k, based on which, the following theorem upper bounds the number of τ_i's requests to ρ_k during a time interval Δt.

Theorem 1. *From the instant when $J_{i,v}$'s x-th ($1 \leqslant x \leqslant N_{i,k}$) request to ρ_k is satisfied, $J_{i,v}$ (and the successive jobs of τ_i) can issue at most $\eta^x_{i,k}(\Delta t) = \max\{n \mid \delta^x_{i,k}(n) \leqslant \Delta t\}$ requests to ρ_k (include that corresponding to $\tau^{x*}_{i,k}$) within a time interval Δt.*

Proof. $\eta^x_{i,k}(\Delta t)$ is the inverse function of $\delta^x_{i,k}(n)$, therefore the proof follows directly from Lemma 1. □

Under P-FP + MPCP scheduling, higher priority remote tasks may issue several requests to a global resource ρ_k before a lower priority task acquires that resource. The following theorem provides the total execution time of a task in successive critical sections protecting a specific shared resource.

Theorem 2. *From the instant when $J_{i,v}$ has issued the x-th ($1 \leqslant x \leqslant N_{i,k}$) request to ρ_k, the total execution time of the following n, including $\tau^{x*}_{i,k}$, critical sections that protect ρ_k can be upper bounded by $\psi^x_{i,k}(n)$, wherein,*

$$\psi^x_{i,k}(n) = \sum_{\forall j\in[S^x_{i,k},S^{x+n-1}_{i,k}]\wedge\rho(\tau_i^{j*})=\rho_k} WC_i^{j*}, \qquad (4)$$

if $x + n - 1 \leqslant N_{i,k}$; and

$$\psi^x_{i,k}(n) = \sum_{\substack{\forall j\in[S^x_{i,k},N_{i,k}]\\ \wedge\rho(\tau_i^{j*})=\rho_k}} WC_i^{j*} + \sum_{\substack{\forall j\in[1,S^{n+x-1-N_{i,k}}_{i,k}]\\ \wedge\rho(\tau_i^{j*})=\rho_k}} WC_i^{j*}, \qquad (5)$$

if $N_{i,k} < x + n - 1 \leqslant 2N_{i,k}$; and

$$\psi^x_{i,k}(n) = \psi^x_{i,k}(n - N_{i,k} \times \lfloor n/N_{i,k}\rfloor) + \xi_{i,k} \times \lfloor n/N_{i,k}\rfloor, \qquad (6)$$

if $x + n - 1 > 2N_{i,k}$, where $\xi_{i,k} = \sum_{\rho(\tau_i^{j*})=\rho_k} WC_i^{j*}$.

Proof. Firstly, if $x + n - 1 \leqslant N_{i,k}$, the n requests will be issued within one job of τ_i. Thus, $\psi^x_{i,k}(n)$ is equal to the cumulative execution time of the following n critical sections for ρ_k, as provided by (4).

Secondly, if $N_{i,k} < x+n-1 \leqslant 2N_{i,k}$, the n requests will be issued by two successive jobs, $J_{i,v}$ and $J_{i,v+1}$. Wherein, $J_{i,v}$ executes, from $\tau^{x*}_{i,k}$ to $\tau^{N_{i,k}*}_{i,k}$, $N_{i,k}-x+1$ critical sections for ρ_k, and $J_{i,v+1}$ will execute, from $\tau^{1*}_{i,k}$, the remaining $n - (N_{i,k} - x + 1)$ critical sections that protect ρ_k. We prove (5).

Thirdly, if $x + n - 1 > 2N_{i,k}$, the total n requests will be issued by at least three successive jobs of τ_i. Since the cumulative execution time in any $N_{i,k}$ successive critical sections corresponding to ρ_k is equal to $\xi_{i,k} = \sum_{\rho(\tau_i^{j*})=\rho_k} WC_i^{j*}$, a total of $N_{i,k} \times \lfloor n/N_{i,k}\rfloor$ successive requests to ρ_k sum up an execution time of $\xi_{i,k} \times \lfloor n/N_{i,k}\rfloor$. Besides these $N_{i,k} \times \lfloor n/N_{i,k}\rfloor$ requests, the remaining requests to ρ_k are less than $N_{i,k}$. Thus, $x + (n - N_{i,k} \times \lfloor n/N_{i,k}\rfloor) - 1 \leqslant N_{i,k}$, or $N_{i,k} \leqslant x + (n - N_{i,k} \times \lfloor n/N_{i,k}\rfloor) - 1 \leqslant 2N_{i,k}$. As a result, (4) or (5) can be used to upper bound the execution time of the remaining requests. We prove (6). □

Combining $\eta^x_{i,k}(\Delta t)$ with $\psi^x_{i,k}(n)$, we can provide the upper bound on the total execution time of τ_i in successive critical sections corresponding to ρ_k during a time interval Δt. Based on such bound, we present a tighter worst-case blocking time analysis in the next section.

4 Improved Response Time Analysis

In this section, we analyze the response time of tasks scheduled under P-FP + MPCP scheduling. Firstly, we provide upper bounds on worst-case blockings according to the refined task model discussed in Section 3. Then we modify the framework in [9] and present an improved WCRT analysis.

4.1 Upper Bounds on Worst-Case Blockings

According to Definition 1, task blockings can be formally classified into remote and local blockings. We bound the remote blocking time caused by remote tasks with lower and higher priorities respectively, and bound the local blocking time incurred by lower priority local tasks.

4.1.1 Derivation of Remote Blocking Time

Since there is an ordering for priority ceilings of GCSs, a job can be preempted by other local jobs even when it is executing in GCSs. For example in Fig.3, $J_{2,*}$ is preempted (blocked) by a lower priority local job $J_{3,*}$ at t_1 when executing in the GCS. That is because $J_{3,*}$ has queued on ρ_h and has inherited the priority ceiling Ω_h before $J_{2,*}$ releases, it is able to preempt the execution of $J_{2,*}$ in the GCS because $\Omega_h > \Omega_k$ (both $J_{1,*}$ and $J_{3,*}$ request the global resource ρ_h, and $J_{1,*}$ has a base priority higher than that of $J_{2,*}$).

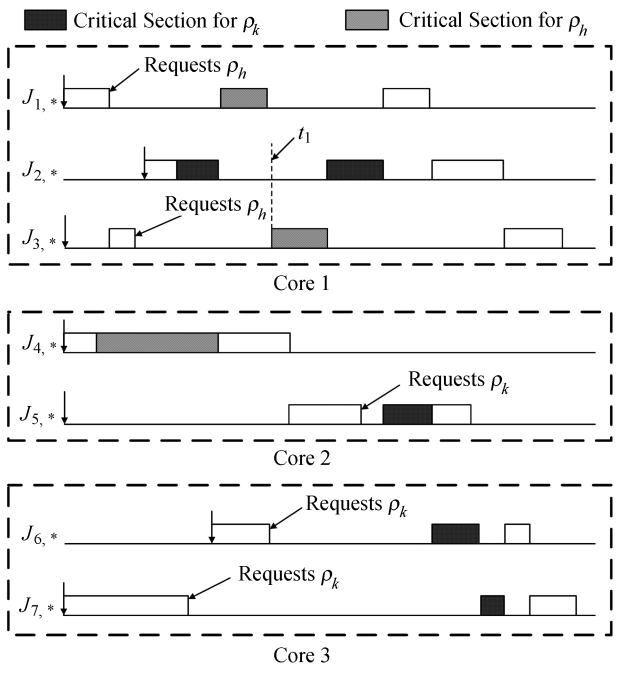


Fig.3. Illustrative example for remote blockings.

Lemma 2. *When $J_{i,*}$ is executing in the GCS protecting ρ_k, it suffers a delay for at most*

$$\varphi_{i,k} = \sum_{\tau_a \in \Gamma_i^{\mathrm{local}}} \max_{\substack{j\in[1,S_{a-1}] \\ \wedge \rho(\tau_a^{j*})=\rho_h \wedge \Omega_h > \Omega_k}} WC_a^{j*}.$$

Proof. Since $J_{i,*}$ inherits the priority ceiling Ω_k, it cannot be preempted by any other job with base priority. Suppose another local job $J_{a,*}(\tau_a \in \Gamma_i^{\mathrm{local}}$, where $\Gamma_i^{\mathrm{local}}$ denotes the local tasks of τ_i) is queuing on another global resource $\rho_h(\Omega_h > \Omega_k)$, and it acquires the lock to ρ_h before $J_{i,*}$ completes its execution in the GCS. Then $J_{a,*}$ can preempt $J_{i,*}$, because it has inherited the priority ceiling Ω_h. After $J_{a,*}$ exits the GCS, it cannot preempt $J_{i,*}$ again. That is because $J_{i,*}$ has the priority ceiling Ω_k, making it impossible for $J_{a,*}$ to be scheduled and issue another request. Therefore, $J_{i,*}$ that is executing in a GCS can be preempted by $J_{a,*}$ for at most once. Similarly, any job in GCSs protected by higher priority ceilings can potentially preempt $J_{i,*}$ for at most once. □

Once a job acquires the lock to a global resource, it occupies the global resource until it finishes the execution of the corresponding GCS. For example in Fig.3, $J_{2,*}$ occupies ρ_k during t_1 and t_2. Based on the analysis in Lemma 2, the maximum occupation time of $J_{i,*}$ on ρ_k can be computed as follows:

$$MO_{i,k} = \varphi_{i,k} + \max_{j\in[1,S_{i-1}]\wedge\rho(\tau_i^{j*})=\rho_k} WC_i^{j*}.$$

Lemma 3. *Each time $J_{i,*}$ requests the global resource ρ_k, it can be blocked by lower priority remote tasks for at most*

$$b_{i,k}^{l} = \max_{\tau_l\in\Gamma-\Gamma_i^{\mathrm{local}}\wedge l>i} MO_{l,k}.$$

Proof. Since the waiting queue on ρ_k is prioritized, none of the lower priority remote jobs waiting for ρ_k can acquire the lock before $J_{i,*}$ does. Thus, $J_{i,*}$ needs to wait for at most one lower priority remote job $J_{l,*}$ ($\tau_l \in \Gamma - \Gamma_i^{\mathrm{local}}$ and $l > i$) for at most the occupation time of $MO_{l,k}$. □

When a lower priority job $J_{i,*}$ is waiting for a global resource ρ_k, remote jobs with higher priorities may be added into the same waiting queue and inserted in front of $J_{i,*}$. As can be observed in Fig.3, $J_{7,*}$ issues a request to ρ_k before $J_{5,*}$ and $J_{6,*}$. However $J_{5,*}$ and $J_{6,*}$ acquire the lock to ρ_k in advance due to higher priorities. In addition, any high priority remote job may issue several requests to ρ_k before $J_{i,*}$ acquires the lock to ρ_k. In that case, $J_{i,*}$ has to wait until ρ_k is freed up and $J_{i,*}$ is right in the head of the waiting queue.

Lemma 4. *After $J_{i,*}$ issues a request to global resource ρ_k, it can be blocked by higher priority remote tasks for at most $b_{i,k}^h(\Delta t)$ during a time interval Δt.*

$$b_{i,k}^h(\Delta t) = \sum_{\tau_h\in\Gamma-\Gamma_i^{\mathrm{local}}\wedge h<i} \max_{x\in[1,N_{h,k}]} (\psi_{h,k}^x(\eta_{h,k}^x(\Delta t)) + \eta_{h,k}^x(\Delta t) \times \varphi_{h,k}). \quad (7)$$

Proof. The execution time of jobs of τ_h in GCSs protecting ρ_k during a time interval Δt can be formulated by $\psi_{h,k}^x(\eta_{h,k}^x(\Delta t))$. The additional interferences during the execution of those jobs in GCSs can be bounded by $\eta_{h,k}^x(\Delta t) \times \varphi_{h,k}$. Thus, the maximum occupation time of jobs of τ_h on ρ_k during a time interval of length Δt is $\max_{x\in[1,N_{h,k}]}(\psi_{h,k}^x(\eta_{h,k}^x(\Delta t)) + \eta_{h,k}^x(\Delta t) \times \varphi_{h,k})$.

Since all higher priority remote jobs can preempt $J_{i,*}$ in the global queue of ρ_k, the cumulative blocking time of $J_{i,*}$ can be safely bounded by the sum of the respective factors. □

A solution can be obtained by iteration, because both sides of (7) grow monotonically with respect to the size of Δt. Summing up $b^l_{i,k}$ and $b^h_{i,k}(\Delta t)$, the remote blocking time of $J_{i,*}$ on ρ_k can be bounded by $RB_{i,k}$ as following.

$$RB^{z+1}_{i,k} = b^l_{i,k} + b^h_{i,k}(RB^z_{i,k}). \tag{8}$$

Theorem 3. *Suppose Φ^G_i is the set of global resources requested by τ_i, then $J_{i,*}$ may incur remote blocking for at most*

$$RB_i = \sum_{\forall \rho_k \in \Phi^G_i} RB_{i,k} \times N_{i,k}. \tag{9}$$

Proof. The proof follows directly from the preceding discussion. □

4.1.2 Derivation of Local Blocking Time

A job is suspended whenever it is blocked on a global resource. Lower priority local jobs may execute and even queue on or lock global resources, and thus may result in multiple interferences to higher priority jobs. Consider the following cases.

Suppose $J_{i,*}$ is suspended on a global resource ρ_k while another lower priority local job $J_{l,*}$ is queuing up on an arbitrary global resource ρ_a. Then $J_{l,*}$ will cause $J_{i,*}$ to incur blocking when 1) $J_{l,*}$ acquires the lock to ρ_a during the time $J_{i,*}$ is executing in the GCS and $\Omega_a > \Omega_k$ ($J_{l,*}$ preempts $J_{i,*}$ immediately); 2) $J_{l,*}$ acquires the lock to ρ_a during the time $J_{i,*}$ is executing in the GCS and $\Omega_a \leqslant \Omega_k$[4] ($J_{l,*}$ will preempt $J_{i,*}$ after which completes the execution of the GCS); 3) $J_{l,*}$ acquires the lock to ρ_a when $J_{i,*}$ is executing the non-critical code ($J_{l,*}$ preempts $J_{i,*}$ immediately).

In addition, priority inversions may occur before the higher priority job issues the first request to global resources. We illustrate the phenomenon of local blocking via an example, as shown in Fig.4.

$J_{5,*}$ and $J_{6,*}$ issue requests to ρ_k and ρ_h at t_1 and t_2 respectively, and both jobs are suspended and added into the waiting queues. Then higher priority job $J_{4,*}$ releases and executes the non-critical code. $J_{5,*}$ acquires the lock to ρ_k and inherits the priority ceiling Ω_k at t_3, and it preempts $J_{4,*}$ because of higher effective priority. Similar phenomenon occurs at t_4 when $J_{6,*}$ inherits the priority ceiling Ω_h. Therefore, $J_{4,*}$ suffers local blocking before issuing requests to shared resources. $J_{4,*}$ is suspended at t_5 when trying to lock ρ_k, letting $J_{5,*}$, $J_{6,*}$, and $J_{7,*}$ execute and queue on global resources at t_6, t_7, and t_8 respectively. Subsequently, $J_{5,*}$ acquires the lock to ρ_h at t_9 and preempts $J_{4,*}$ when it is executing in the GCS (condition 1); $J_{6,*}$ acquires the lock to ρ_k which is just released by $J_{4,*}$ at t_{10}, thus $J_{6,*}$ preempts $J_{4,*}$ (condition 2); $J_{7,*}$ acquires the lock to ρ_l at t_{11} and preempts $J_{4,*}$ when it is executing in the non-critical code (condition 3).

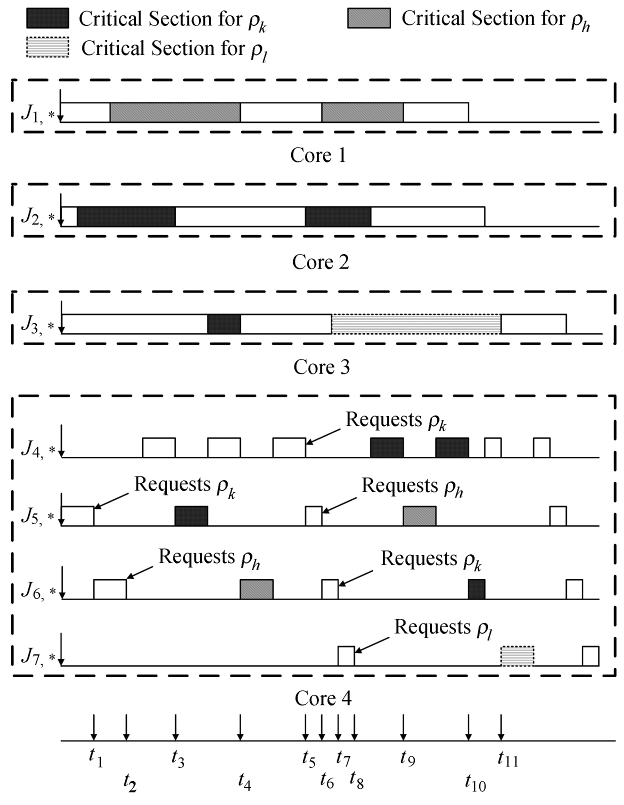


Fig.4. Illustrative sequence of tasks for local blockings.

According to what discussed above, a job $J_{i,*}$ can be blocked by any lower priority local job for at most $N_{i,G} + 1$ times, where $N_{i,G}$ is the number of GCSs of $J_{i,*}$. In that case, the lower priority job must issue at least $N_{i,G}$ requests to shared resources during the time $J_{i,*}$ is pending (plus one before $J_{i,*}$ releases). However, there is only a limited time for lower priority jobs to execute (only during the time when $J_{i,*}$ is suspended), and thus most of these blockings can be ruled out.

To bound the execution time of jobs of τ_i in successive critical sections, we slightly modify $\delta^x_{i,k}(n)$ and $\psi^x_{i,k}(n)$ to be $\delta^x_{i,*,h}(n)$ and $\psi^x_{i,*,h}(n)$ respectively. A special noteworthiness is that $\delta^x_{i,k}(n)$ and $\psi^x_{i,k}(n)$ are

[4]It is possible for $J_{l,*}$ to request ρ_a ($\rho_a = \rho_k$) when $J_{i,*}$ is waiting for the same global resource. In that case, $J_{l,*}$ will be inserted to the same queue behind $J_{i,*}$, and may acquire the lock to ρ_k and preempt $J_{i,*}$ after it releases ρ_k.

specified for certain critical sections protecting ρ_k, while the modified ones are for all critical sections that have priority ceilings higher than the base priority of τ_h (i.e., in these critical sections, τ_i has a higher effective priority than the base priority of τ_h). It can easily be shown that, $\delta^x_{i,*,h}(n)$ and $\psi^x_{i,*,h}(n)$ can be computed by the similar methods used in (1)∼(6).

Lemma 5. *$J_{i,*}$ suffers local blocking caused by jobs of $\tau_l(\tau_l \in \Gamma_i^{\text{local}}$ and $i < l)$ for at most $LB_i^l = \max_{\Omega_{\rho(\tau^{x*}_{l,*})} \geqslant \pi(\tau_i) \wedge x \in [1, S_i - 1]} \{\psi^x_{l,*,i}(n) | \psi^x_{l,*,i}(n) \geqslant \delta^x_{l,*,i}(n) - RB_i\}$, where $\pi(\tau_i)$ is the base priority of τ_i.*

Proof. Only when $J_{i,*}$ is pending can it be blocked according to Definition 1. During that time, jobs of τ_l can only be scheduled to execute non-critical sections when $J_{i,*}$ is suspended. From Theorem 3, the suspension time of $J_{i,*}$ is at most RB_i. Suppose $J_{i,*}$ is blocked by jobs of τ_l for n times, then jobs of τ_l must execute for at least $\delta^x_{l,*,i}(n)$, during which a total of $\psi^x_{l,*,i}(n)$ is the execution time of these jobs in critical sections. Thus, $J_{i,*}$ can be blocked by jobs of τ_l for at most $\psi^x_{l,*,i}(n)$. In that case, jobs of τ_l need to execute for at least $\psi^x_{l,*,i}(n) + RB_i$. Since $\delta^x_{l,*,i}(n) \leqslant \psi^x_{l,*,i}(n) + RB_i$ is always true, $\psi^x_{l,*,i}(n) \geqslant \delta^x_{l,*,i}(n) - RB_i$ holds. □

Theorem 4. *$J_{i,*}$ incurs local blocking for at most*

$$LB_i = \sum_{l > i \wedge \tau_l \in \Gamma_i^{\text{local}}} LB_i^l.$$

Proof. The proof follows directly from Lemma 5. □

4.2 Upper Bounds on Task Response Time

Based on the work in [9], the WCRT of tasks scheduled by P-FP + MPCP can be bounded by

$$R_i^{z+1} = WC_i + RB_i + LB_i + \sum_{h < i \wedge \tau_h \in \Gamma_i^{\text{local}}} \left\lceil \frac{R_i^z + RB_h}{T_h} \right\rceil \times WC_h. \quad (10)$$

Note that the calculation of R_i with (10) requires the knowledge of blocking time. However, the derivation of both RB_i and LB_i involves $\delta^x_{i,k}(n)$ and $\delta^x_{i,*,h}(n)$ respectively, which in turn requires the value of R_i, i.e., there are inter-dependences between response time and blocking time. To distinguish the response time in different scenarios, we let WR_i denote the WCRT obtained according to (10) and IR_i denote the initial value used when computing $\delta^x_{i,k}(n)$ or $\delta^x_{i,*,h}(n)$. A possible solution is as follows:

1) let $IR_i = BC_i$ in Lemma 1 (computing $\delta^x_{i,k}(n)$) when WR_i is calculated for the first time;

2) replace IR_i with WR_i that is obtained in the previous step, so as to recalculate WR_i;

3) repeat 2) until the value of WR_i becomes stable.

It is noted in (1)∼(3) that, for a given value of n, $\delta^x_{i,k}(n)$ is a non-increase function of R_i (IR_i). Since $\eta^x_{i,k}(\Delta t)$ can be considered as an inverse function of $\delta^x_{i,k}(n)$, the value of $\eta^x_{i,k}(\Delta t)$ can potentially increase with IR_i. Further, according to (7)∼ (9), RB_i increases or remains stable when IR_i increases. However, it is unclear whether LB_i is non-decreasing with the increase of IR_i. Therefore, (10) may not converge in some special cases in theory[5]. If oscillation happens in step 2), we turn to test all $R_i \in [\kappa, T_i]$ in increasing order until the first $\omega \in [\kappa, T_i]$ that satisfies (11). Then, $R_i = \omega$, otherwise we consider τ_i is unschedulable.

$$R_i = WC_i + RB_i + LB_i + \sum_{h < i \wedge \tau_h \in \Gamma_i^{\text{local}}} \left\lceil \frac{R_i + RB_h}{T_h} \right\rceil \times WC_h,$$
$$\kappa = WC_i + \sum_{h < i \wedge \tau_h \in \Gamma_i^{\text{local}}} \left\lceil \frac{\kappa}{T_h} \right\rceil \times WC_h. \quad (11)$$

5 Evaluations

In this section, we evaluate the efficiency of the proposed analysis by comparing it with the original schedulability test[5] (Original), the deferrable model-based WCRT analysis[9] (D-WCRT), and the event stream model-based WCRT analysis[12] (E-WCRT).

Firstly, we use a simple example to illustrate the improvement of the proposed analysis on worst-case blockings. Suppose five tasks, indexed in decreasing order of priorities, as shown in Table 1, run on a 2-core multicore processor. τ_1 and τ_3 are assigned on core 1, and the other three tasks are assigned on core 2. All tasks will share a single shared resource ρ_k. Suppose $\alpha = BC_i^j / WC_i^j = BC_i^{j*} / WC_i^{j*} = 0.5$, we focus on τ_2

Table 1. Task Parameters for the Illustrative Example

τ_i	Core 1		Core 2		
	τ_1	τ_3	τ_2	τ_4	τ_5
WC_i^1	7	16	12	16	234
WC_i^{1*}	3	3	3	3	\
WC_i^2	10	26	19	32	\
WC_i^{2*}	3	3	3	3	\
WC_i^3	10	19	26	33	\
WC_i^{3*}	3	3	\	3	\
WC_i^4	6	15	\	18	\
WC_i	42	85	63	108	234
T_i	120	340	300	600	650
u_i	0.3	0.25	0.21	0.18	0.36

[5]Nonetheless, we tested a total of more than 10 million randomly generated task sets in various scenarios, and (10) always converges.

and calculate its worst-case blocking time using different analysis methods. It is noted that the worst-case critical section lengths of all tasks are same in this example (i.e., equal to 3), and we use WC^* to denote such length for simplicity.

Each time τ_2 requests ρ_k, it can be blocked by lower priority remote task (i.e., τ_3) for $b^3_{2,\,k} = WC^*$. Since $RB^0_{2,\,k} = 0$, $RB^1_{2,\,k} = b^3_{2,\,k} + b^1_{2,\,k}(0)$. From Lemma 4, τ_2 can also be blocked by higher priority remote task (i.e., τ_1) for $b^1_{2,\,k}\ (\Delta t)$ during a time interval Δt. From (7), $b^1_{2,\,k}(0) = \max_{x\in[1,N_{1,k}]} \psi^x_{1,\,k}(\eta^x_{1,\,k}(0)) = WC^*$, thus $RB^1_{2,\,k} = 2\,WC^*$. Further, from Lemma 1, it will take at least $WC^* + 10 \times \alpha = 8 > 2\,WC^*$ for τ_1 to issue two requests to ρ_k, thus $b^1_{2,\,k}(2\,WC^*) = WC^*$, making $RB^2_{2,\,k} = 2\,WC^*$. Since τ_2 requests ρ_k for twice (i.e., $N_{2,\,k} = 2$), $RB_2 = 2 \times 2\,WC^* = 4\,WC^*$. On the other hand, τ_2 may experience local blocking for at most LB^4_2, according to Lemma 5. Since $\delta^x_{4,\,*,\,2}(2) = WC^* + 32 \times \alpha = 19$ and $\psi^x_{4,\,*,\,2}(2) = 2\,WC^*$, $\psi^x_{4,\,*,\,2}(2) < \delta^x_{4,\,*,\,2}(2) - RB_2$. From Lemma 5, $LB^4_2 = \max_{x\in[1,S_2-1]\wedge\Omega_{\rho(\tau^{x*}_{4,\,*})}\geqslant\pi(\tau_2)} \psi^x_{4,\,*,\,2}(1) = WC^*$. Therefore, the worst-case blocking for τ_2 is $5\,WC^*$.

While in [5], blocking terms are calculated independently. These terms include: 1) local blocking on local resources (nonexistent in this example); 2) remote blocking by lower priority tasks, that is, $b^3_{2,\,k} = WC^* \times N_{2,\,k} = 2\,WC^*$; 3) remote blocking by higher priority tasks, that is, $b^1_{2,\,k} = N_{2,\,k} \times \lceil T_2/T_1 \rceil \times WC^* = 6\,WC^*$; 4) indirect remote blocking (nonexistent in this example); 5) local blocking on global resources, that is, $LB^4_2 = \min(N_{2,\,k} + 1, 2N_{4,k}) \times WC^* = 3\,WC^*$. Therefore, the worst-case blocking for τ_2 is $11\,WC^*$.

In [9], worst-case blockings are bounded by using a WCRT-like approach. Accordingly, $rb^1_{2,\,1} = WC^* + (\lceil WC^*/T_1 \rceil + 1) \times WC^* = 3\,WC^*$, and $rb^2_{2,\,1} = rb^1_{2,\,1}$, so $rb_{2,\,1} = 3\,WC^*$ (according to (2) and (3) in [9]). Further, $rb_{2,\,2} = rb_{2,\,1}$, and the overall remote blocking is $RB_2 = rb_{2,\,1} + rb_{2,\,2} = 6\,WC^*$. Additionally, the local blocking of τ_2 is $LB_2 = S_2 \times WC^* = 3\,WC^*$. Thus, the worst-case blocking for τ_2 is $9\,WC^*$.

In [12], blocking terms are calculated independently as that in [5], and a function, $\tilde{\eta}^+_i(\Delta t) = \lceil \Delta t/d_{i,\text{srr}} \rceil$, was introduced to bound the number of requests that τ_i can issue to shared resources during a time interval Δt. $d_{i,\text{srr}}$ is defined to be the minimum distance between any two shared resource requests of τ_i. Accordingly, $b^3_{2,\,k}$ and LB^4_2 are the same as that in [5], while $b^1_{2,\,k} = \tilde{\eta}^+_1(R_2) \times WC^*$. Since R_2 is at least $\sum_{j\in[1,S_2-1]}(WC^j_2 + WC^{j*}_2) + WC^{S_i}_2 = 35$, and $d_{1,\text{srr}} = 10$, $b^1_{2,\,k} = 4\,WC^*$[6]. The worst-case blocking for τ_2 is $9\,WC^*$.

Next, we conduct schedulability experiments to illustrate the efficiency of the proposed method. Task sets with varying characteristics are generated, and are tested for schedulability (hard real-time) under P-FP + MPCP scheduling.

5.1 Experimental Setup

The ratio of schedulable task sets with varying characteristics on a platform with eight cores is evaluated. The Worst-Fit-Decreasing (WFD) heuristic is used for task partitioning. We explore factors that affect the schedulability as follows: 1) the ratio of best-case to worst-case execution time for each execution segment, i.e., $\alpha = BC^j_i/WC^j_i = BC^{j*}_i/WC^{j*}_i$, and $\alpha \in [0, 1]$; 2) the total system utilization $U = \sum^n_{i=1} u_i$ varies in increments of 0.2 from 0.2 to 8; 3) the maximum number of tasks that may require any shared resource, denoted by β[7], varies from 1 to 20; 4) the critical section length that varies from 0.05ms to 2 ms; 5) the number of critical sections per task varies from 1 to 18.

An experimental scenario consists of a permutation of the above parameters. We select a system utilization cap (in (0, 8]) in each scenario, based on which we generate tasks as follows. Task periods are generated from [10ms, 600ms] with uniform distribution, and task utilizations are selected uniformly from [0.05, 0.2]. Priorities of tasks are assigned according to the Rate Monotonic (RM) algorithm[15]. The WCET of each task τ_i is determined by the selected period and utilization, and the WCETs of the normal computation segments are randomly chosen from $[0.5 \times \sum^{S_i}_{j=1} NC^j_i/S_i$, $1.5 \times \sum^{S_i}_{j=1} NC^j_i/S_i]$. These tasks are added to the task set until the cumulative task utilization exceeds the selected system utilization cap, and in that case the last generated task is discarded. If the remaining utilization gap is within the range of values allowed for the task utilization, we generate a task that exactly fits the utilization gap and add it to the task set; otherwise, we regenerate the task set. The number of shared resource is determined by the experimental scenario. Suppose a task set contains nt tasks, each task contains y critical sections, and each shared resource can be requested by at most β tasks, then the number of the shared resources is set to $\lceil y \times nt/\beta \rceil$. A total number of 50 000 task sets are generated for each scenario, and are tested using different schedulability algorithms.

[6]As shown in the example, function $\tilde{\eta}^+_i(\Delta t)$ may be very pessimistic when $d_{i,\text{srr}}$ is small. We use $\min(\tilde{\eta}^+_i(R_j),\ \lceil T_j/T_i \rceil \times N_{i,\,k})$ in our schedulability experiments.

[7]For example, if each resource in the system can only be requested by at most w tasks, then $\beta = w$.

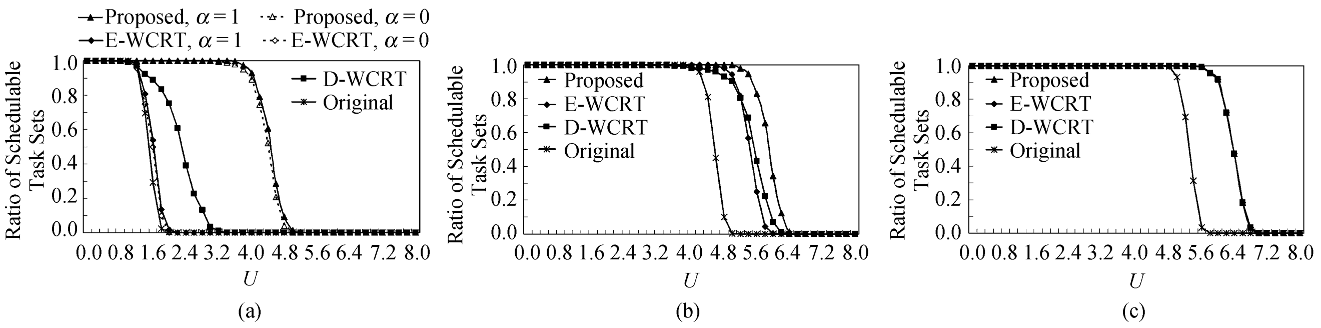


Fig.5. Schedulability with increasing U. (a) $\alpha = 1$ and 0.01, $\beta = 9$, the critical section length is 0.2 ms, and each task has 8 critical sections. (b) $\alpha = 0.5$, $\beta = 4$, the critical section length is 0.2 ms, and each task has 2 critical sections. (c) $\alpha = 0.5$, $\beta = 1$, the critical section length is 0.2 ms, and each task has 1 critical section.

5.2 Experimental Results

Firstly, we study the influence of α on the schedulability performances. Fig.5(a) shows the proposed method and E-WCRT performance slightly better when $\alpha = 1$, compared to that when $\alpha = 0.01$. That is because the worst-case blockings from both analysis approaches depend on the minimum inter-arrival time for shared resource requests, and tasks tend to issue more requests and incur more blockings within a time interval Δt when α (equivalently, the minimum inter-arrival time for shared resource requests) is small. On the other hand, D-WCRT and Original only use WCET during blocking analysis, thus the performances of both methods remain unchanged.

Both E-WCRT and D-WCRT improve upon the original analytical method, and the proposed method performs the best, as shown in Fig.5(a), Fig.5(b), and in Fig.6, E-WCRT performs better than D-WCRT when system loads are light, e.g., $U < 5$ in Fig.6(b), but it performs worse than D-WCRT otherwise. Similar trends can also be observed in Fig.6. It is also noted in Fig.5(c) that, the performances of E-WCRT, D-WCRT, and the proposed method are same when each task has only one critical section.

Fig.6 shows the influences of β, critical section length, and the number of critical sections per task on the performances of different methods. When each shared resource can be requested by more tasks, there are fewer shared resources in the system (recall that the number of shared resources is $\lceil y \times nt/\beta \rceil$ in this experiment). Consequently, tasks are more likely to block each other with bigger β. It is shown in Fig.6(a) that the schedulability performances of all methods tend to decrease with increasing β. Notably, the performance of the proposed method is seldom affected in the scenario when $\alpha = 0.5$, $U = 5$, and each task has two critical sections with lengths equal to 0.2 ms.

Fig.6(b) shows that the performances of each methods decrease uniformly with increasing critical section lengths. That is because tasks tend to suffer longer blockings, when critical section lengths are long. It can also be observed in Fig.6(b) that, the performance of the proposed method decreases slowly with increasing critical section length, while the other three methods degrade dramatically. Similar trends can also be seen in Fig.6(c) when each task has more critical sections. That is because tasks may experience more blockings when issuing more requests to shared resources.

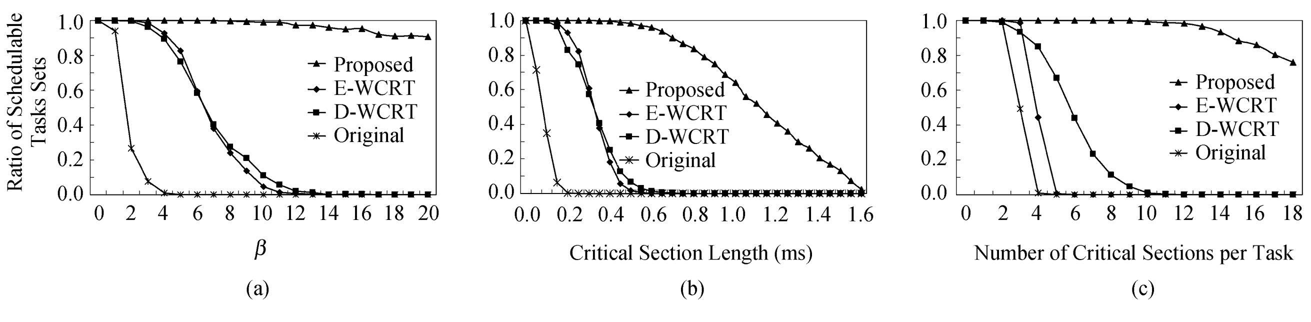


Fig.6. (a) Schedulability with increasing β, when $\alpha = 0.5$, $U = 5$, the critical section length is 0.2 ms, and each task has 2 critical sections. (b) Schedulability with increasing critical section length, when $\alpha = 0.5$, $U = 5$, $\beta = 2$, and each task has 2 critical sections. (c) Schedulability with increasing number of critical sections per task, when $\alpha = 0.5$, $U = 5$, $\beta = 2$, and the critical section length is 0.2 ms.

6 Conclusions

This paper addressed the blocking analysis for P-FP + MPCP scheduled multiprocessor real-time systems. In this paper, tasks are modeled as alternating execution segments of critical sections and non-critical sections, and each execution segment is associated with best-case and worst-case execution time (to make the model expressive, BCET can vary from zero to WCET). Based on this model, we conducted a detailed timing analysis and derived the lower bound of the inter-arrival time for shared resource requests within the same task. Furthermore, we provided a tighter upper bound, compared to existing work, on the execution time of a task in successive critical sections during an arbitrary time interval of Δt. Then, we improved the worst-case blocking analysis and presented a modified WCRT analysis based on [9]. Schedulability experiments in various scenarios show that the proposed analysis improves upon the existing approaches.

References

[1] Gracioli G, Fröhlich A, Pellizzoni R, Fischmeister S. Implementation and evaluation of global and partitioned scheduling in a real-time OS. *Real-Time Syst.*, 2013, 49(6): 669-714.

[2] Brandenburg B. Scheduling and locking in multiprocessor real-time operating systems [Ph.D. Thesis]. The University of North Carolina at Chapel Hill, 2011.

[3] Brandenburg B, Calandrino J, Block A, Leontyev H, Anderson J. Real-time synchronization on multiprocessor: To block or not to block, to suspend or spin? In *Proc. the 14th IEEE Real-Time Embedded Tec. App. Symp.*, April 2008, pp.342-353.

[4] Ras J, Cheng A. An evaluation of the dynamic and static multiprocessor priority ceiling protocol and the multiprocessor stack resource policy in an SMP systems. In *Proc. the 15th IEEE RTAS*, April 2009, pp.13-22.

[5] Rajkumar R. Real-time synchronization protocols for shared memory multiprocessors. In *Proc. the 10th Int. Conf. Dist. Computing Syst.*, May 28-Jun. 1, 1990, pp.116-123.

[6] Block A, Leontyev H, Brandenburg B, Anderson J. A flexible real-time locking protocol for multiprocessors. In *Proc. the 13th IEEE RTCSA*, August 2007, pp.47-56.

[7] Brandenburg B, Anderson J. Optimality results for multiprocessor real-time locking. In *Proc. the 31st IEEE Real-Time Syst. Symp.*, Nov. 30-Dec. 3, 2010, pp.49-60.

[8] Sha L, Rajkumar R, Lehoczky J. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Computers*, 1990, 39(9): 1175-1185.

[9] Lakshmanan K, De Niz D, Rajkumar R. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *Proc. the 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp.469-478.

[10] Bril R, Lukkien J, Verhaegh W. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *J. Real-Time Syst.*, 2009, 42(1/2/3): 63-119.

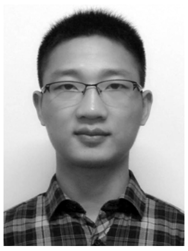
[11] Bertogna M, Baruah S. Limited preemption EDF scheduling of sporadic task systems. *IEEE Trans. Industrial Informatics*, 2010, 6(4): 579-591.

[12] Schliecker S, Negrean M, Ernst R. Response time analysis on multicore ECUs with shared resources. *IEEE Trans. Industrial Informatics*, 2009, 5(4): 402-413.

[13] Brandenburg B. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling. In *Proc. the 19th IEEE RTAS*, April 2013, pp.141-152.

[14] Ridouard F, Richard P, Cottet F. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proc. the 25th IEEE Real-Time Syst. Symp.*, December 2004, pp.47-56.

[15] Liu C, Layland J. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 1973, 20(1): 46-61.



Mao-Lin Yang is a Ph.D. candidate in the School of Information and Software Engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu. He received his B.S. degree in light chemical engineering from Tianjin University of Science and Technology in 2009, and M.S. degree in software engineering from Zhejiang University in 2011. His research interests include real-time scheduling algorithms, real-time locking protocols, and real-time operating systems. He is a student member of IEEE.

Hang Lei is a professor and Ph.D. supervisor in the School of Information and Software Engineering, UESTC, Chengdu. He received his Ph.D. degree in computer system architecture from UESTC in 1997. His research interests include embedded software technology, software systems, software reliability test and evaluation technology. He is a member of CCF.

Yong Liao is an associated professor in the School of Information and Software Engineering, UESTC, Chengdu. He received his Ph.D. degree in computer system architecture from UESTC, in 2006. His research interests include real-time scheduling and embedded real-time operating systems. He is a member of CCF.

Furkan Rabee is a Ph.D. student in the School of Computer Science and Engineering, UESTC, Chengdu. He received his B.S. degree in computer science and M.S. degree in computer network from AL-Nahrian University, in 2000 and 2008 respectively. He worked as lecturer in computer science of Kufa University in Iraq. His research interests include real-time scheduling algorithms, real-time locking protocols, and real-time operating systems.