Optimal Priority Assignments in P-FRP*

Chaitanya Belwal, Albert M.K. Cheng

Computer Science Department
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

UH-CS-11-03
April 27, 2011

Keywords: Real-time Systems, Priority Assignment, Schedulability Analysis, Functional Programming

Abstract

Priority-based Functional Reactive Programming (P-FRP) has been recently introduced as a new functional programming formalism for real-time systems. P-FRP allows static priority assignment and guarantees real-time response by preempting lower priority tasks. Due to the state-less nature of functional programs, preempted tasks in P-FRP are aborted and restarted after the higher priority tasks have completed execution. In the classical preemptive model† of real-time systems, it has been demonstrated that for fixed priority scheduling, if tasks are schedulable with any priority assignment, they are also schedulable by the rate-monotonic (RM) priority assignment, making this priority assignment optimal for all task sets in the preemptive model. However, the RM priority assignment is not optimal in P-FRP, and it has been unknown if an optimal fixed priority assignment can even exist for such an execution model. In this paper, we formally present the priority assignment characteristics of P-FRP and show that based on task periods, either a combined utilization and rate-monotonic, or only the rate-monotonic priority assignments is optimal for a system with two tasks. Using this result, we formally prove the limitation of optimal priority assignments in a P-FRP system having more than two tasks where, unless arrival periods are integer multiples of each other, no single priority assignment exists which is optimal for all task sets. Experimental results using task sets of different sizes are also presented.

† In this paper the classical preemptive model refers to a real-time system in which tasks can be preempted by higher priority tasks and can resume execution from the point they were preempted

* This work is supported in part by U.S. National Science Foundation under Award no. 0720856

Optimal Priority Assignments in P-FRP*

Chaitanya Belwal and Albert M.K. Cheng
Department of Computer Science,
University of Houston, TX, USA

Abstract

Priority-based Functional Reactive Programming (P-FRP) has been recently introduced as a new functional programming formalism for real-time systems. P-FRP allows static priority assignment and guarantees real-time response by preempting lower priority tasks. Due to the state-less nature of functional programs, preempted tasks in P-FRP are aborted and restarted after the higher priority tasks have completed execution. In the classical preemptive model‡ of real-time systems, it has been demonstrated that for fixed priority scheduling, if tasks are schedulable with any priority assignment, they are also schedulable by the rate-monotonic (RM) priority assignment, making this priority assignment optimal for all task sets in the preemptive model. However, the RM priority assignment is not optimal in P-FRP, and it has been unknown if an optimal fixed priority assignment can even exist for such an execution model. In this paper, we formally present the priority assignment characteristics of P-FRP and show that based on task periods, either a combined utilization and rate-monotonic, or only the rate-monotonic priority assignments is optimal for a system with two tasks. Using this result, we formally prove the limitation of optimal priority assignments in a P-FRP system having more than two tasks where, unless arrival periods are integer multiples of each other, no single priority assignment exists which is optimal for all task sets. Experimental results using task sets of different sizes are also presented.

Index Terms

Real-time Systems, Priority Assignment, Schedulability Analysis, Functional Programming

I. Introduction

Functional Reactive Programming (FRP) [28] is a declarative programming language for the modeling and implementation of reactive systems. It has been used for a wide range of applications, notably, graphics [9], robotics [20], and vision [21]. FRP elegantly captures continuous and discrete aspects of a hybrid system using the notions of behavior and event, respectively. Because this language is developed as an embedded language in Haskell [15], it benefits from the wealth of abstractions provided in this language. Unfortunately, Haskell provides no real-time guarantees, and therefore, neither does FRP.

To address this limitation, resource-bounded variants of FRP were studied [16],[26],[27]. Recently, it was shown that a variant called priority-based FRP (P-FRP) [16] combines both the semantic properties for FRP, guarantees resource boundedness, and supports the assignment of different priorities to different events. In P-FRP, higher priority events can preempt lower-priority ones. However, a requirement [24] in the functional programming model is that the state of the system cannot be changed, and no function can have side effects. To maintain this guarantee of stateless execution, the functional programming paradigm requires the execution of an event handler (or *task*) to be atomic in nature. To comply with this requirement, as well as allow preemption of lower priority events, P-FRP implements a transactional model of execution. By using only a copy of the state

‡ In this paper the classical preemptive model refers to a real-time system in which tasks can be preempted by higher priority tasks and can resume execution from the point they were preempted

* This work is supported in part by U.S. National Science Foundation under Award no. 0720856

during event processing and atomically committing these changes at the end of the event handler, a multi-version commit model of execution is implemented. This ensures that handling an event is an “all or nothing” proposition, and ensures the atomicity of handling an event. This is shown to preserve the easily understandable semantics of the FRP, and provides a programming model where response times to different events can be tweaked by the programmer without ever affecting the semantic soundness of the program.

Functional programming offers several benefits over the imperative programming style used in C++, Java, Ada etc. It allows the programmer to intuitively describe safety critical behaviors of the system, lowering the chance of introducing bugs in the design phase, while its stateless nature of execution does not require use of synchronization primitives, reducing the complexity of programming. While several variants of functional languages are being used in embedded systems, like Erlang [10] for mission critical telecommunication equipment and Atom [13] for controlling hybrid vehicles, their use in practical real-time and embedded systems is still quite limited. Apart from a steep learning curve, lack of understanding of their space and real-time temporal properties has been cited [12] as one of the reasons inhibiting a wider industry adoption of functional languages.

For fixed-priority scheduling in real-time systems, there are essentially two areas of work; schedulability analysis and priority assignment. For a given priority assignment, schedulability analysis determines if tasks in the system can complete execution before their respective deadlines. Hence, priority assignment has a direct impact on the schedulability of a given task set. A task set which is known to be schedulable, is also guaranteed to be schedulable in an *optimal* priority assignment. Knowledge of an optimal priority assignment for a task set gives system designers valuable insight on schedulable assignments of task priorities in fixed priority systems. An optimal priority assignment also serves as a schedulability test, since if a task set is not schedulable in its optimal priority assignment it is guaranteed to be not schedulable in any priority assignment.

In their seminal work, Liu and Layland [18] showed that the rate-monotonic (RM) priority assignment is an optimal priority assignment for the classical preemptive model of execution. Furthermore, Leung and Whitehead [17] showed that if task deadlines are same as task arrival periods, then the optimality for RM priority assignment is valid only when tasks are released at the same time (*synchronously*). However, the optimality of the RM priority assignment does not hold true for P-FRP. This is due to the abort nature of preemption, where the actual execution time taken by tasks can be higher than their *a priori* known worst-case execution times. Hence, a relevant question that arises for real-time researchers is, can an optimal priority assignment even exist for such an execution model ? And under what constraints can such an optimal priority assignment be applicable.

An answer to this question has benefits to real-time research which extend beyond the functional programming model we have studied. Over the past several years, researchers have looked at the abort-restart model as a promising method to avoid concurrency control and resource sharing conflicts in the preemptive execution model. This has resulted in response-time studies of lock-free semantics [1], preemptable critical sections in Java [19], and lately, transactional memory systems [14]. While each of these systems have their unique execution semantics, work on priority assignment for P-FRP can be extended to these systems. In future work, we will be using results of this work to derive optimal priority assignment in the lock-free execution model. Previous works on other abort-restart models [1],[6],[14],[19] and P-FRP [16],[22] have only handled the problem of response time analysis.

A. Contributions

This paper presents a formal study on P-FRP, and finds special groups of task sets which have the same optimal priority assignment under a synchronous release of tasks. We also analyze those groups of task sets where no single priority assignment which is optimal for sets can exist. For such task sets, analyzing all possible combinations of task priorities is the only option.

We first present schedulability (Section 3) and priority assignment (Section 4) characteristics of P-FRP and show that between 2 P-FRP tasks, the rate-monotonic priority assignment is optimal only for those task sets where one task period is more than or equal to double of the other. When this condition is not met, a single priority assignment is optimal only if it has both utilization-monotonic (UM) and rate-monotonic (RM) characteristics. We conclude that if a P-FRP task set with 2 tasks is schedulable, it is guaranteed to be also schedulable under a utilization or rate-monotonic priority assignment (Section 5). We then look at systems with n tasks ($n > 2$) and show that the RM priority assignment is optimal for only that group of task sets where task periods are integer multiples of each other. For other cases, we use the results derived for 2 task systems and prove that no priority assignment can exist which is optimal for all task sets (Section 6) . Results generated from simulation of experimental task sets validate our theorems, and also show that even though no single priority assignment is optimal for n-task sets, several task sets are schedulable under a UM or RM priority assignment (Section 7). We conclude by reviewing related work (Section 8) and a reflection on our results (Section 9).

II. Basic Concepts and Execution Model

In this section, we introduce the basic concepts and the notation used to denote these concepts in the rest of the paper. In addition, we review the P-FRP execution model and assumptions made in this study.

A. Basic Concepts

Essential concepts for P-FRP are tasks and their associated priority, their associated time period and the concept of arrival rate and their processing time. Included also is the concept of a time interval and task jobs therein. The notation and formal definitions for these concepts as well as a few others used in the paper are as follows:

- Let task set $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n periodic tasks. Γ_n is also referred to as an ***n*-task set**
- The **priority** of a $\tau_k \in \Gamma_n$ is the integer pr_k. If $pr_i > pr_k$ then τ_i has a higher priority than τ_k. Each task is associated with a unique priority number
- T_k is the **arrival time period** between two successive jobs of τ_k and $r_k = 1 / T_k$ is the **arrival rate** of τ_k
- C_k is the fixed **worst-case execution time** (WCET) for τ_k
- $t_{copy}(k)$ is the time taken to make a **copy** of the state before τ_k starts processing (see section 2.2.1)
- $t_{restore}(k)$ is the time taken to **restore** the state after τ_k has completed processing (see section 2.2.1)
- P_k is the **processing time** for τ_k. Processing of a task includes execution as well as copy and restore operations. Hence, $P_k = t_{copy}(k) + C_k + t_{restore}(k)$
- An **absolute time** t or **time** t is the time elapsed since, the real-time system was started. The real-time system is assumed to have started at absolute time 0
- $[t_1,t_2)$ represents a time interval such that: $\forall t \in [t_1,t_2)$, $t_1 \le t < t_2 \wedge t_1 \neq t_2$, t_1 and t_2 are absolute times
- $R_{k,m}$ represents the **release time** of the m^{th} job of τ_k
- Φ_k represents the **release offset** which is the release time of the first job of τ_k. Or, $\Phi_k = R_{k,1}$. Hence, $R_{k,m} = \Phi_k + (m-1)\cdot T_k$
- D_k is the **relative deadline** of τ_k. If some job of τ_k is released at time $R_{k,m}$,then τ_k should complete processing by time $R_{k,m} + D_k$, otherwise τ_k will have a deadline miss. For this study, $D_k = T_k$
- The **utilization ratio** of a task τ_k (U_k), is the ratio of its processing time to its arrival time period. $U_k = \frac{P_k}{T_k}$
- The **total utilization factor** (U) of a task set is the sum of ratios of processing time to arrival periods of every task. Hence, $U = \sum_{i=1}^{n} \frac{P_i}{T_i}$
- A **feasibility interval** is the time interval $[t_H, t_H + H)$ such that if all tasks are schedulable in $[t_H, t_H + H)$ then the tasks will also be schedulable in the time interval $[0, Z)$: $Z \to \infty$. H is the length of the feasibility interval and t_H is its start time
- **Interference** on τ_k is the action where the processing of τ_k is interrupted by the release of a higher priority task
- A **rate-monotonic (RM)** priority assignment is one where priorities are assigned to tasks based on their arrival rates. The task with the highest arrival rate has the highest priority
- A **utilization-monotonic (UM)** is one where priorities are assigned to tasks based on their utilization ratios. The task with the highest utilization ratio has the highest priority

B. Execution Model and Assumptions

For this study, all tasks are assumed to execute in a uniprocessor system and have no precedence constraints. When a job of a higher priority task is released, it can immediately preempt a lower priority task, and changes made by the lower priority task are rolled back. The lower priority task will be restarted after the higher priority task has completed processing. When some task is released, it enters a processing queue which is arranged by priority order such that all arriving higher priority tasks are moved to the head of the queue. The length of the queue is bounded and no two instances of the same task can be present in the queue at the same time. This requires a task to complete processing before the release of its next job. To maintain this requirement, we assume a *hard* real-time system with task deadline equal to the time period between jobs. Hence, $\forall \tau_k \in \Gamma_n, D_k = T_k$.

A task set is schedulable in some time interval only if no task in the set has a deadline miss. Every job of task τ_k is assumed to execute for its worst-case execution time, hence the processing times for all jobs of τ_k (P_k) is considered the same.

Once a task τ_i enters the processing queue, two situations are possible. If a task of lower priority than τ_i is being processed, it will be immediately preempted and τ_i will start processing. If a task of higher priority than τ_i is being processed, then τ_i will wait in the queue and start processing only after the higher priority task has completed. An exception to the immediate preemption is made during *copy* and *restore* operations, which is explained in the following section.

1)Copy and Restore Operations

In P-FRP, when a task starts processing it creates a 'scratch' state, which is a *copy* of the current state of the system. Changes made during the processing of this task are maintained inside such a state. When the task has completed, the 'scratch' state is *restored* into the final state in an atomic operation. Therefore, during the restoration and copy operations the task being processed cannot be preempted by higher priority tasks. If the task is preempted after copy, but before the restore operation, the scratch state is simply discarded. The time to discard the state of an aborted task is minimal and has been ignored in this study. The context-switch between tasks only involves a state copy operation for the task that will be commencing processing. The time taken for copy ($t_{copy}(k)$) and restore ($t_{restore}(k)$) operations of τ_k is part of the processing time of the task, P_k.

In this study, the values of $t_{copy}(k)$ and $t_{restore}(k)$ for all tasks are assumed to be the same and equal to a single time unit of execution. Hence,

$\forall j,k \in \Gamma_n,\ t_{copy}(k)=t_{restore}(k) \text{ and } t_{copy}(j) = t_{copy}(k),$

$t_{copy}(k),\ t_{restore}(k) = 1.$

This is a reasonable assumption, since copy and restore operations are only a fraction of total processing time. Though most of our results can be extended to cases where $t_{copy}(k)$ and $t_{restore}(k)$ can be more than unity, we intend to address this problem in a separate paper.

In this work, we look at priority assignment strategies only for a synchronous release of P-FRP tasks. Therefore, the release time of the first job of all tasks is considered as time 0. Hence,

$\forall \tau_k \in \Gamma_n, \Phi_k = 0.$

III. Schedulability Characteristics in P-FRP

In this section, we present some important schedulability characteristics of P-FRP tasks, based on which we define a necessary schedulability test.

Lemma 3.1. *The total utilization factor of a schedulable P-FRP task set will always be less than or equal to* 1.

Proof. Consider $\Gamma_n = \{\tau_1, \tau_2,\ldots, \tau_n\}$. Let us assume $U > 1$. Let $L = \text{LCM}(T_1 \ldots T_n)$. We take a simple case where all tasks are released at the same time, and there is no interference (hence, no aborts). In the interval $[0,L)$ the total processing time from each of the tasks is: $\frac{P_1}{T_1}\cdot L + \frac{P_2}{T_2}\cdot L \ldots + \frac{P_n}{T_n}\cdot L.$

Let Z denotes the total processor idle period (when no task of Γ_n is being processed) in $[0,L)$. If Z=0, then there is no idle time and some task in Γ_n is always being processed. If $Z > 0$, then no task of Γ_n was processed for total Z time in the interval $[0,L)$. Hence,

$$\frac{P_1}{T_1}\cdot L + \frac{P_2}{T_2}\cdot L \ldots + \frac{P_n}{T_n}\cdot L + Z = L \Rightarrow U + \frac{Z}{L} = 1.$$

Since $U > 1$ and $Z \geq 0$, $\Rightarrow U + \frac{Z}{L} > 1$,Therefore the assumption $U > 1$ is wrong. Hence, $U \leq 1$. □

Lemma 3.2. *If* Γ_n *is schedulable, then any task present in* Γ_n *will be able to complete processing between any two consecutive jobs of every other task present in the set.*

Proof. Assume $\Gamma_2 = \{\tau_i, \tau_j\}$. Let $pr_i > pr_j$. If both tasks are released synchronously then τ_i will complete first. Task j will start processing when τ_i has completed which is at time $0+P_i$. The next job of τ_i will take place at time T_i. The time left for processing τ_j is $T_i - P_i$. If τ_j is unable to complete processing within this time it will be aborted by 2nd job of τ_i which is released at time T_i. After the 2nd job of τ_I has completed processing, τ_j will get another time period of length $T_i–P_i$ to complete. The abort/restart cycle of τ_j will continue for every job of τ_i. If $P_j > T_i - P_i$, τ_j will never be able to complete and the task set will be unschedulable. Hence to be schedulable, τ_j will be able to complete processing between successive jobs of τ_i.

Now, the first job of τ_j is released at time 0 and the second will be released at time T_j. If tasks τ_i and τ_j are released synchronously then τ_i will complete first since it has the higher priority. This leaves a maximum of $T_j - P_i$ time for τ_j to complete processing. Since τ_j requires a contiguous period of minimum P_j length to complete processing, for the task set to be schedulable:

$P_j \leq T_j - P_i \Rightarrow P_i \leq T_j - P_j$.

Since $T_j - P_j$ is the time remaining to execute τ_i between successive jobs of τ_j, τ_i will complete processing between successive jobs of τ_j, otherwise τ_j will be aborted by jobs of τ_i and will never be able to complete processing.

If $\Gamma_n = \{\tau_1, \tau_2 \ldots,\tau_n\}$ we can do the above analysis for each unique pair $\{\tau_i, \tau_j\},\tau_i, \tau_j \in \Gamma_n$ to show that if Γ_n is schedulable, each task in P-FRP task set will be able to complete processing between successive jobs of other tasks present in the set. □

Definition: Lemmas 3.1 and 3.2 define conditions which will always be satisfied by any schedulable P-FRP task set. However, the satisfaction of conditions specified in lemmas 3.1 and 3.2 alone does not guarantee the schedulability of the task set, since, the schedulability also depends on the priority assignment and execution pattern of tasks. Therefore, these schedulability conditions are necessary but not sufficient. The verification of conditions specified in lemmas 3.1 and 3.2 is termed as the **P-FRP schedulability test** in the rest of this paper.

IV. Characteristics of Priority Assignment in P-FRP

In this section, we define characteristics for priority assignment in P-FRP. We show that the rate-monotonic priority assignment is not optimal for P-FRP, and prove that a task set schedulable in P-FRP will always be schedulable in the preemptive model. We also compute the costs induced due to interference and abort in P-FRP, and introduce the concept of intermediate release points (IRPs), which characterize the release time of the higher priority tasks. These IRPs are classified into *abort* and *delay* types, and we prove that only abort IRP affect the schedulability in P-FRP. Two important observations are defined while important results are derived in theorems 4.11 and 4.13. Relevant definitions and examples are also given at various places in this section.

From this section onwards, any general P-FRP task set Γ_n is assumed to satisfy the P-FRP schedulability test. The variable L represents the least-common-multiple (LCM) of the task periods. The feasibility interval for a synchronous release in P-FRP, as given in [3] is $[0,L)$.

Lemma 4.1. *In P-FRP, the rate-monotonic priority assignment is not an optimal priority assignment with synchronous release of tasks.*

Proof. If we can give a P-FRP task set which is not schedulable using the RM priority assignment, but is schedulable by a priority assignment which is not RM, it is sufficient to prove this lemma. Consider the following task set:

Task	*pr*	*P*	*T*	*U*
τ_1	1	7	15	0.46
τ_2	2	3	12	0.25

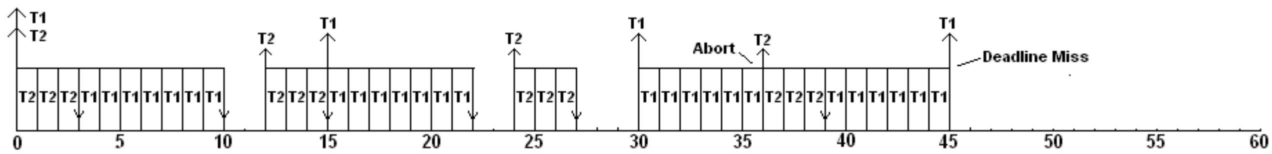

Figure 1(a): Deadline miss for the 2nd job of τ_1 under RM priority assignment. T1, T2, T3 represent tasks τ_1, τ_2 , τ_3 respectively

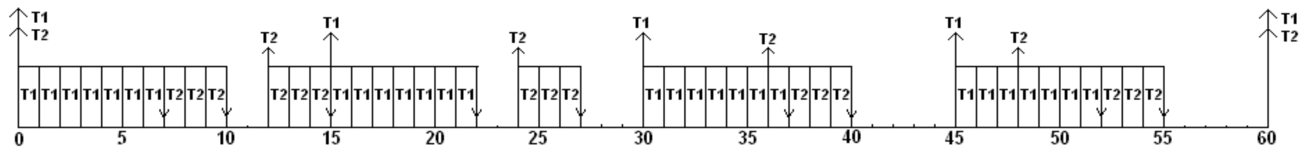

Figure 1(b): Task set is schedulable in the feasibility interval of [0,60) under a non-RM priority assignment

The priority assignment is RM-based with τ_2 having the highest arrival rate hence, the highest priority. In this scheduling policy, the first job of τ_1 is unable to complete processing before its second job at time 45 (*Figure 1(a)*). If the priority order is changed, as shown below:

Task	*pr*	*P*	*T*	*U*
τ_1	2	7	15	0.46
τ_2	1	3	12	0.25

Then jobs of all tasks will be able to complete processing in the feasibility interval [0,60) of this task set (*Figure 1(b)*). □

Lemma 4.2. *If a task set is schedulable for some priority assignment in the classical preemptive model, then it is not guaranteed to be schedulable for the same priority assignment in P-FRP.*

Proof. If we can show that a task set is unschedulable for some priority assignment in P-FRP, but schedulable in the classical model it will be sufficient to prove this lemma.

Consider the task set used in lemma 4.1:

Task	*pr*	*P*	*T*
τ_1	1	7	15
τ_2	2	3	12

This priority assignment is rate-monotonic and schedulable in the classical model. As we have already shown, this priority assignment is not schedulable in P-FRP. □

Lemma 4.3. *If a task set is schedulable for some priority assignment in P-FRP, then it will also be schedulable for the same priority assignment in the classical preemptive model.*

Proof. The response time of the highest priority task in P-FRP and the classical model will be the same. Higher priority tasks can cause interference in the processing of lower priority tasks. In P-FRP, this interference leads to abort which puts another cost on the processing time of lower priority tasks. There are two possible situations:

<u>No interference from higher priority tasks</u>: The difference in response time between P-FRP and classical model is created by abort of lower priority tasks, which is caused by interference from tasks of higher priority. Hence, if there is no interference, there will be no aborts and response time for all tasks in P-FRP and the classical model will be same. Hence, if the task set is schedulable in P-FRP, it will also be schedulable in the classical model.

Interference from higher priority tasks: Consider, $\Gamma_2 = \{\tau_i, \tau_j\}$ and $pr_i > pr_j$:

Let τ_j be released at time t_a and execute for h time units: $t_{copy}(j) \le h \le t_{copy}(j) + C_j$, after which it is aborted by the release of a job of τ_i. The selected range of h allows τ_i to be released after the copy, but before the restore operations of τ_j. τ_j will re-start processing after τ_i has completed at time: t_a+h+P_i. τ_j will take another P_j time units to complete processing and will finish at time $t_a+h+P_i+P_j$. Since τ_j was released at time t_a, its response time is: $h+P_i+P_j$.

If tasks are processed in the preemptive model, the response time of τ_j will be $h+P_i+P_j–h = P_i+P_j$. Hence, after interference from higher priority tasks, the response time of lower priority tasks in P-FRP will always be more than the response time in the preemptive model. Hence, if a task is schedulable in P-FRP, it will also be schedulable in the classical model. □

Definition. In the preemptive model of execution, if a higher priority task τ_i interferes with the execution of a lower priority task τ_j, then τ_i will preempt τ_j. The response time of τ_j will be delayed by time taken to process τ_i, which is P_i. This is referred to as the **interference cost.** In the P-FRP execution model, preempted tasks are also aborted. The amount of time spent in aborted processing is called the **abort cost.** Hence, in P-FRP, interference induces both an interference and abort cost on the response time of a preempted lower priority task.

Lemma 4.4(a). *For two tasks τ_i and τ_j, if $T_i < T_j$, then in the time interval $[0,L)$ there will be at least one job of τ_i that is released strictly between any two successive jobs of τ_j.*

Proof. The time difference between the releases of any two jobs of τ_j is T_j.

Number of jobs of τ_i between any two jobs of $\tau_j = \left\lfloor \frac{T_j}{T_i} \right\rfloor$. Since, $T_j > T_i$ at least one job of τ_i will be released between any two jobs of τ_j. □

Example: In *Figure 2*, $T_2 < T_1$. The 2nd job of τ_2 is released between the 1st and 2nd jobs of τ_1, 3rd job of τ_2 is released between 2nd and 3rd jobs of τ_1 and so on.

Lemma 4.4(b). *For two tasks τ_i and τ_j, if $T_i < T_j$. In the time interval $[0,L)$, with the exception of the 1st and last jobs, every job of τ_j will be released between two successive jobs of τ_i.*

Proof. Let $L = p \cdot T_i = q \cdot T_j$, $p > q$.

Jobs of τ_j will be released at times: 0, T_j, $2 \cdot T_j$, $3 \cdot T_j$ …, $q \cdot T_j$.
Jobs of τ_i will be released at times: 0, T_i, $2 \cdot T_i$, $3 \cdot T_i$ …, $p \cdot T_j$.

Since, $T_j \neq f \cdot T_i$, $T_i < T_j < 2 \cdot T_i$
Similarly, $2 \cdot T_i < 2 \cdot T_j < 3 \cdot T_i$ *or* $3 \cdot T_i < 2 \cdot T_j < 4 \cdot T_i$

It is easy to see, that for any m^{th} job of T_j:
$a \cdot T_i < (m–1) \cdot T_j < (a+1) \cdot T_i$, $0 < a < p$ and $1 < m \le q$. □

Example: In *Figure 2*, $T_2 < T_1$. The 2nd job of τ_1 is released between the 2nd and 3rd job of τ_2, 3rd job of τ_1 is released between 3rd and 4th jobs of τ_2 and so on.

Definition. A relative time instance where the p^{th} job of a higher priority task τ_i, is released (time $R_{i,p}$) between job m (time $R_{j,m}$) and job m+1 (time $R_{j,m+1}$) of a lower priority τ_j is termed as an **intermediate release point (IRP),** provided $R_{i,p-1} \neq R_{j,m}$. The length of the IRP is relative to the release time of the m^{th} job of τ_j. If multiple jobs of higher priority task τ_i are released in the interval $[R_{j,m}, R_{j,m+1})$, then only the first job of τ_i that is released in this interval will be considered as an IRP. The **intermediate release point set** $\psi_{IRP}(i,j)$, contains the length of all those IRPs where τ_i is released between jobs of τ_j, for the priority assignment $pr_i > pr_j$ in the time interval $[0,L)$. $|\psi_{IRP}(i,j)|$ represents the number of elements in the set $\psi_{IRP}(i,j)$. The value of an IRP refers to its length. Since, $R_{i,p-1} \neq R_{j,m}$ an

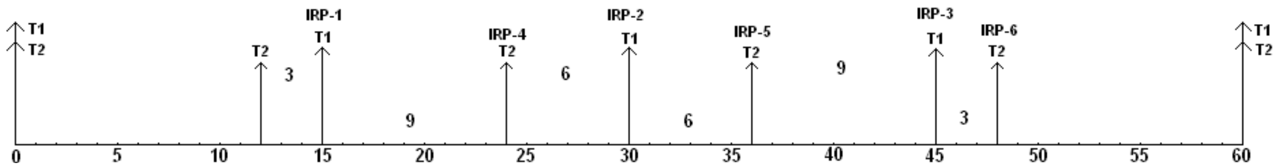

Figure 2: Intermediate release points in the task set example used in lemma 4.1. In this example $\psi_{IRP}(1,2) = \{ 3,6,9 \}$ represented by IRP-1, IRP-2 and IRP-3. $\psi_{IRP}(2,1) = \{ 9,6,3 \}$ represented by IRP-4, IRP-5 and IRP-6. When $pr_2 > pr_1$, IRP-5 and IRP-6 can abort τ_1, hence $\psi_{AIRP}(2,1) = \{3,6\}$, and $max_{AIRP}(2,1) = 6$. When $pr_1 > pr_2$, IRP-2 and IRP-5 cause delay in the start of τ_2, hence $\psi_{DIRP}(1,2) = \{6,9\}$ while $\psi_{AIRP}(1,2) = \varnothing$ and $max_{AIRP}(1,2) = 0$

IRP will always have non-zero values. *Figure 2* shows the IRPs for different priority assignments for our sample 2-task set.

Lemma 4.5. *For two tasks τ_i and τ_j, the maximum value of intermediate release point, for any priority assignment cannot exceed the value of the lowest arrival period. Or:*

$$\text{if, } h \in \psi_{IRP}(i,j) \text{ or } h \in \psi_{IRP}(j,i) \text{ then } h < \text{minimum}(T_i, T_j).$$

Proof. Let $\min(T_i, T_j) = T_i$ and $h \in \psi_{IRP}(i,j)$.

In lemma 4.4(a), we have seen that at least one job of τ_i is released between any two jobs of τ_j. However only the first job is counted in the set $\psi_{IRP}(i,j)$ as per the IRP definition.

Let the p^{th} job of τ_i be released at time t_a between the m^{th} and $(m+1)^{th}$ jobs of τ_j, creating an IRP of length h. Therefore, the m^{th} job of τ_j is released at time $t_a - h$. If $h > T_i$, then the $(p-1)^{th}$ job of τ_i will be released after m^{th} job and the $(p-1)^{th}$ job will be the IRP, making $h < T_i$. If $h = T_i$ then it means that both τ_i and τ_j are released at the same time in which case the p^{th} job of τ_i is not an IRP as per definition.

Hence, if $h \in \psi_{IRP}(i,j)$ then $h < T_i$.

Now, let $h \in \psi_{IRP}(j,i)$. In lemma 4.4(b), we have shown that a job of τ_j will always be released between successive jobs of τ_i. Assume that the p^{th} job of τ_i is released at time t_a, and the m^{th} job of τ_j is released at time $t_a + h$. If $h > T_i$, then the m^{th} job of τ_j will be an IRP for the $(p+1)^{th}$ job of τ_i. If $h = T_i$, then the m^{th} job of τ_j is not an IRP as per definition. If $h < T_i$ then the m^{th} job of τ_j will be an IRP for the p^{th} job of τ_i

Hence, if $h \in \psi_{IRP}(i,j)$ then $h < T_i$. □

Lemma 4.6. *For two tasks τ_i and τ_j, if $pr_i > pr_j$, and an intermediate release point lies in the range, $[\varepsilon, t_{copy}(j)+C_j]$ or $[\varepsilon, P_j - t_{restore}(j)]$, $\varepsilon > 0$, then τ_i will induce an abort cost on the response time of τ_j.*

Proof. Assume some job of τ_j be released at time t_a, and a job of τ_i is released at time t_a+h: $h \in [\varepsilon, t_{copy}(j)+C_j]$, $\varepsilon > 0$. Since $h < P_j$, $h \in \psi_{IRP}(i,j)$. Even if τ_i is released at time ε: $\varepsilon < t_{copy}(j)$, it will induce a minimum abort cost of $t_{copy}(j)$ on τ_j. If $h > t_{copy}(j)+C_j$ then τ_j will be in the restoration phase and cannot be aborted by τ_i. Hence, every $h: h \in \psi_{IRP}(i,j)$ and $h \in [\varepsilon, t_{copy}(j)+C_j]$ will induce an abort cost on the response time of τ_j.

Since, $t_{copy}(j)+C_j = P_j - t_{restore}(j)$, $h \in [\varepsilon, P_j - t_{restore}(j)]$ will also induce abort cost on τ_j. □

Definition. An IRP is an **abort intermediate release point (AIRP)** if it adds abort costs to the response time of a lower priority task. The **abort intermediate release point set** $\psi_{AIRP}(i,j)$ of two tasks τ_i and τ_j, contains only those IRPs in the time interval $[0,L)$, with the priority assignment $pr_i > pr_j$, which can induce an abort cost on the response time of τ_j. Or,

$$\forall h \in \psi_{AIRP}(i,j),\ h \in \psi_{IRP}(i,j) \wedge h \in [\varepsilon, t_{copy}(j)+C_j],\ \varepsilon > 0.$$

Based on the above definition it is clear that,

$$\psi_{AIRP}(i,j) \subset \psi_{IRP}(i,j).$$

The **maximum abort intermediate release point** ($max_{AIRP}(i,j)$) is the maximum value in the set $\psi_{AIRP}(i,j)$:

$$max_{AIRP}(i,j) = \text{maximum}\ \{\psi_{AIRP}(i,j)\}.$$

From lemma 4.6, the upper bound on $max_{AIRP}(i,j)$ is easily derived to be: $t_{copy}(j)+C_j$.

Example: In *Figure 2*, $\psi_{AIRP}(2,1) = \{3,6\}$ and $max_{AIRP}(i,j) = 6$, while $\psi_{AIRP}(1,2) = \emptyset$.

Lemma 4.7. *For two tasks τ_i and τ_j, if $\alpha = T_j - P_j - P_i$, then any intermediate release point in $\psi_{IRP}(i,j)$, that lies in the range $[P_j+\alpha+1, T_j]$ will cause a delay in the start of some job of τ_j.*

Proof. Let the m^{th} job of τ_j be released at time t_a. Assume $|\psi_{IRP}(i,j)| > 0$ and $pr_i > pr_j$.

For some $h \in \psi_{IRP}(i,j)$, let the p^{th} job of τ_i be released at time t_a+h and let τ_i start processing at time $t_a+h+\beta$, $\beta \geq 0$. β accounts for any blocking τ_i will experience if it is released during the state copy or restoration phase of τ_j. τ_i will complete processing at time $t_a+h+\beta+P_i$.

The p^{th} job of τ_i will delay the start of the $(m+1)^{th}$ job of τ_j only if:

$$t_a+h+\beta+P_i > t_a + T_j \Rightarrow h+\beta+P_i > T_j \quad \ldots (4.7.1)$$

from the definition of α: $P_i + P_j + \alpha = T_j$,
substituting this value of T_j in eq. 4.7.1 :

$$h+\beta > P_j + \alpha \quad \ldots (4.7.2)$$

if, $h \leq t_{copy}(j)+C_j$ then $\beta = 0$ and $h+\beta < P_j + \alpha$

hence, to satisfy eq. (4.7.2), $h > t_{copy}(j)+C_j$.

Let us take a look when:

$$t_{copy}(j)+C_j < h \leq t_{copy}(j)+C_j + t_{restore}(j)$$

for every, $h= t_{copy}(j)+C_j + \delta$,

$$\beta = t_{restore}(j) - \delta,\ \delta > 0$$

Therefore, $h+\beta = t_{copy}(j)+C_j +\delta + t_{restore}(j) - \delta = P_j$.

In this case, eq. (4.7.2) will not be satisfied. To satisfy eq. (4.7.2):

$$h > t_{copy}(j)+C_j + t_{restore}(j) \Rightarrow \beta = 0.$$

Since, $\beta = 0$, in eq. (4.7.2): $h > P_j + \alpha$.

Or, the minimum possible value of $h = P_j+\alpha+1$.

If $h > T_j$, then τ_i will be released after the $(m+1)^{th}$ job of τ_j has started processing. In this case, τ_i will not cause a delay in the start of τ_j, but can cause it to abort. Therefore, the maximum possible value of h which can delay the start of $(m+1)^{th}$ job of τ_j is T_j. Hence, if $h \in [P_j+\alpha+1, T_j]$, the release of a job of τ_i at time $t_a + h$ is guaranteed to delay the execution of τ_j. □

Definition. An IRP is a **delay intermediate release point (DIRP)** if it causes a delay in the start time of the lower priority task. The **delay intermediate release point set** $\psi_{DIRP}(i,j)$ contains only those IRPs of τ_i in the time interval $[0,L)$ and the priority assignment $pr_i > pr_j$, which can delay the start time of τ_j. Or,

$\forall h \in \psi_{DIRP}(i,j) : h \in \psi_{IRP}(i,j) \wedge h \in [P_j+\alpha+1, T_j],\ \alpha=T_j-P_j-P_i$
Based on the above definition it is clear that,

$$\psi_{DIRP}(i,j) \subset \psi_{IRP}(i,j).$$

<u>Example</u>**:** In *Figure 2,* $\psi_{DIRP}(1,2) = \{6,9\}$, while $\psi_{DIRP}(2,1) = \varnothing$.

Lemma 4.8. *The sets* $\psi_{AIRP}(i,j)$ *and* $\psi_{DIRP}(i,j)$ *are mutually exclusive.*

Proof. For, any $h_1 \in \psi_{AIRP}(i,j)$, $h_1 \in [\varepsilon, t_{copy}(j)+C_j]$, $\varepsilon > 0$
For, any $h_2 \in \psi_{DIRP}(i,j)$, $h_2 \in [P_j+\alpha+1, T_j]$, $\alpha = T_j - P_j - P_i$

Since, $t_{copy}(j)+C_j < P_j$
if , $h \in [\varepsilon, t_{copy}(j)+C_j]$ then $h \notin [P_j+\alpha+1, T_j]$.

Or ,
$\forall h_1 \in \psi_{AIRP}(i,j)$, $h_1 \notin \psi_{DIRP}(i,j)$ and,
$\forall h_2 \in \psi_{DIRP}(i,j)$, $h_2 \notin \psi_{AIRP}(i,j)$.

Therefore, $\psi_{DIRP}(i,j) \cap \psi_{AIRP}(i,j) = \varnothing$. □

Theorem 4.9. *Delay intermediate release points do not affect the schedulability of a task set. Or, if* $\Gamma_2 = \{\tau_i, \tau_j\}$ *is schedulable when* $\psi_{DIRP}(i,j) = 0$*, then it is guaranteed to be schedulable when* $\psi_{DIRP}(i,j) \neq 0$*.*

Proof. Let $h \in \psi_{DIRP}(i,j) \Rightarrow h \in [P_j+\alpha+1, T_j]$ and the m^{th} job of τ_j is released at absolute time t_a. The m^{th} job of τ_j will complete processing at time $t_a + P_j$. Let the p^{th} job of τ_i be released at time t_a+h such that is causes a delay of time β in the start of the $(m+1)^{th}$ job of τ_j. Assume the $(m+1)^{th}$ job of τ_j is unable to complete processing before the release of its $(m+2)^{th}$ job.

The p^{th} job of τ_i will complete processing by time t_a+h+P_i. Hence, $\beta = t_a+h+P_i - (t_a+T_j)$. Since, $h \in [P_j+\alpha+1, T_j]$ the maximum possible value of $h = T_j$, in which case:
$\beta = t_a+T_j+P_i - (t_a+T_j) = P_i$.
This is the upper bound of β.

After the p^{th} job of τ_i has finished processing, the $(m+1)^{th}$ job of τ_j will start and complete processing at time $t_a+T_j+\beta+P_j$. The $(p+1)^{th}$ job of τ_i will not be released till time $R_{i,p+1} = t_a+h+T_i$.

Since the basic schedulability test is satisfied, we know:
$T_i - P_i \geq P_j \Rightarrow (t_a+h) - (t_a+h) + T_i - P_i \geq P_j$
$\Rightarrow R_{i,p+1} - (t_a+h+P_i) \geq P_j$.

No job of τ_i will be released in the interval $[t_a+h+P_i, R_{i,p+1})$ therefore, the $(m+1)^{th}$ job of τ_j will not experience any interference from τ_i in this interval. To be unschedulable, the $(m+2)^{th}$ job of τ_j should be released before the $(m+1)^{th}$ job has completed processing. Or,
$t_a + 2 \cdot T_j < t_a+T_j+\beta+P_j$
$\Rightarrow \beta > T_j - P_j$ … (4.9.1)

If eq. (4.9.1) is satisfied then the $(m+1)^{th}$ job of τ_j will be unschedulable. However, from the basic schedulability test we know: $P_i \leq T_j - P_j$. Since the upper bound on β is P_i,
$\Rightarrow \beta \leq T_j - P_j$.

Clearly, eq. (4.9.1) can never be satisfied implying that a value of β that can make τ_j unschedulable does not exist.

Hence, any $h \in \psi_{DIRP}(i,j)$ will not affect the schedulability of Γ_2. □

Lemma 4.10. *For two tasks* τ_i *and* τ_j*, if an IRP of length h is present in the set* $\psi_{IRP}(i,j)$*, then an IRP of the same length is also present in the set* $\psi_{IRP}(j,i)$*. Or,*
If, $h \in \psi_{IRP}(i,j)$ *then* $h \in \psi_{IRP}(j,i)$*.*

Proof. Consider $\Gamma_2=\{\tau_i, \tau_j\}:pr_i > pr_j$ and $T_i < T_j$ and $h\in \psi_{IRP}(i,j)$. If two jobs of τ_j are released at times 0 and t_a, then two jobs of τ_j will also be released at times $L - t_a$ and L.

Since, $h\in \psi_{IRP}(i,j)$, jobs of τ_i will be released between jobs of τ_j, at times:

$0, t_a+h-T_i, t_a+h,\ldots, L-(t_a+h), L-(t_a+h-T_i), L$

From lemma 4.4, $h < T_i$; therefore,

$L-(t_a+h) < L-t_a < L-(t_a+h-T_i)$.

Hence, when $pr_j > pr_i$, the job of τ_j releasing at time $L - t_a$ will be an IRP between jobs of τ_i which are released at times $L-(t_a+h)$ and $L-(t_a+h-T_i)$. The length of this IRP will be:

$L- t_a-(L-(t_a+h)) = h$.

Therefore, $h\in \psi_{IRP}(j,i)$.

Similarly for $T_i > T_j$, if $h\in \psi_{IRP}(j,i)$:

$L-(t_a+h) < L-t_a < L-(t_a+h-T_j)$.

Hence, when $pr_i > pr_j$, the job of τ_i that is released at time $L - t_a$ will be an IRP between the jobs of τ_j which are released at times $L-(t_a+h)$ and $L-(t_a+h-T_j)$.

Therefore, $h\in \psi_{IRP}(j,i)$.

Similarly we can show that for any $h\in \psi_{IRP}(i,j)$, $h\in \psi_{IRP}(j,i)$. □

<u>Example</u>: In *Figure 2*, IRP-1 between jobs of τ_2 is of length 3. IRP-6 between jobs of τ_1 is also of length 3. It is clearly seen that for every IRP present between jobs of τ_2, IRP's of the same length are present between jobs of τ_1.

Theorem 4.11. *For two tasks τ_i and τ_j, the IRP set for priority assignment $pr_i > pr_j$ is same as the IRP set for priority assignment $pr_j > pr_i$. Or, $\psi_{IRP}(i,j) = \psi_{IRP}(j,i)$.*

Proof. From theorem 4.10 we know that if some $h\in \psi_{IRP}(i,j)$, then $h\in \psi_{IRP}(j,i)$. Similarly, it is easy to show that for any $h\in \psi_{IRP}(j,i)$, then $h\in \psi_{IRP}(i,j)$. Therefore, we have,

$\forall h_1\in \psi_{IRP}(i,j), h_1\in \psi_{IRP}(j,i)$ and,

$\forall h_2\in \psi_{IRP}(i,j), h_2\in \psi_{IRP}(j,i)$

$\Rightarrow \quad \psi_{IRP}(i,j)= \psi_{IRP}(j,i)$. □

<u>Example</u>: In *Figure 2*, $\psi_{IRP}(1,2) = \{3,6,9\}$ while $\psi_{IRP}(2,1) = \{9,6,3\}$. Clearly, $\psi_{IRP}(1,2) = \psi_{IRP}(2,1)$.

Observation 4.12(a). *For two tasks τ_i and τ_j, if $T_i < T_j < 2\cdot T_i$ and $gcd(T_i, T_j) =1$, then $\psi_{IRP}(i,j) = \{1,2\ldots, T_i-1\}$.*

Observation 4.12(b). *For two tasks τ_i and τ_j, if $T_i < T_j < 2\cdot T_i$ and $gcd(T_i, T_j) =m$: $m >1$, then $\psi_{IRP}(i,j) = \{m, 2\cdot m\ldots, T_i - m\}$.*

<u>Example</u>: In our sample task set, $T_i = 12$ and $T_j = 15$. $gcd(12,15) = 3$. Based on observation 4.12(b):

$\psi_{IRP}(i,j) = \{3, 2\cdot 3, (12-3)\} = \{3,6,9\}$. These IRP's are represented by IRP-1, IRP-2 and IRP-3 in *figure 2*.

Theorem 4.13. *For two tasks τ_i and τ_j, if $T_i < T_j < 2\cdot T_i$: $T_j = T_i + y$ and $P_i < P_j$: $P_i / T_i > P_j / T_j$, then the difference between maximum abort IRPs for priority assignments $pr_i > pr_j$ and $pr_j > pr_i$ should be less than or equal to y: Or,*

$max_{AIRP}(i,j) - max_{AIRP}(j,i) \leq y$.

Proof. Let $P_j = P_i + x$, $x \geq 1$.

Since, $P_i / T_i > (P_i + x) / (T_i + y)$

$\Rightarrow \quad x < y \cdot P_i / T_i \Rightarrow x < y.$

Let us consider two cases based on the greatest common divisor (*gcd*) of T_i and T_j:

Case 1: $gcd(T_i, T_j) = 1$

From observation 4.12(a), the following IRPs will be present: 1, 2 …. , $T_i - 1$.

Since IRPs for this case are in increments of 1, it is guaranteed that there will an IRP of length = $P_i - t_{restore}(i)$. As per lemma 4.6, an IRP with this value is the maximum IRP that can abort τ_i.

Therefore, $\quad max_{AIRP}(j,i) = P_i - t_{restore}(i).$

The maximum possible value of $max_{AIRP}(i,j)$ *is:* $P_j - t_{restore}(j)$.
Therefore,

$$max_{AIRP}(i,j) - max_{AIRP}(j,i) \le P_j - t_{restore}(j) - P_i + t_{restore}(i)$$

since, $t_{restore}(j) = t_{restore}(i)$

$\Rightarrow max_{AIRP}(i,j) - max_{AIRP}(j,i) \le P_j - P_i$

$\Rightarrow max_{AIRP}(i,j) - max_{AIRP}(j,i) \le x.$

Since, $x < y$

$$max_{AIRP}(i,j) - max_{AIRP}(j,i) < y.$$

Case 2: $gcd(T_i, T_j) = m$: $m > 1$

As per observation 4.12(b) the following IRP's will be present $\psi_{IRP}(i,j) = \{m, 2 \cdot m \ldots, T_i - m\}$.

Let, $T_i = a \cdot m$ and $T_j = b \cdot m$: $b > a$. We will look at two different cases.

Case 2.1: $P_i \ge m$

Since, the P-FRP schedulability test is satisfied, we know: $T_i - P_i \ge P_j$.

Since, $P_i \ge m \Rightarrow T_i - P_i \le T_i - m$

$\Rightarrow \quad P_j \le T_i - m$ (since, $T_i - P_i \ge P_j$)

$\Rightarrow \quad P_j \le m \cdot (a-1) \qquad \ldots (4.13.1)$

length of maximum possible IRP = $T_i - m = (a-1) \cdot m$.

Even if P_j is at its highest possible value of $m \cdot (a-1)$ (from eq. 4.13.1), the maximum possible IRP will not be able to abort τ_j. Hence, the maximum value of IRP which can abort τ_j when $P_j = m \cdot (a-1)$, is the next lower IRP:

$$m \cdot (a-1) - m = m \cdot (a-2).$$

Therefore, maximum possible value of $max_{AIRP}(i,j) = m \cdot (a-2)$.
Minimum possible value of $max_{AIRP}(j,i) = 0$ (when, $P_i = m$).

The term $max_{AIRP}(i,j) - max_{AIRP}(j,i)$ is maximized when $max_{AIRP}(i, j)$ has its highest possible value and $max_{AIRP}(j, i)$ its lowest. Or,

$$max_{AIRP}(i, j) - max_{AIRP}(j, i) = m \cdot (a-2) - 0 = m \cdot (a-2).$$

Now, $x = P_j - P_i$. Using values of $P_j = m \cdot (a-1)$ and $P_i = m$ which maximize the expression $max_{AIRP}(i, j) - max_{AIRP}(j, i)$, we get:

$$x = m \cdot (a-1) - m = m \cdot (a-2).$$

Therefore, $max_{AIRP}(i, j) - max_{AIRP}(j, i) = x.$

Since, $x < y$

$max_{AIRP}(i, j) - max_{AIRP}(j, i) < y$.

Case 2.2: $P_i < m$

In this case, τ_i cannot be aborted by any job of τ_j. Hence, $max_{AIRP}(j,i) = 0$.

Let $P_i = \delta$, $2 < \delta < m$.

Since, $P_i / T_i > P_j / T_j \Rightarrow \delta / a{\cdot}m > P_j / b{\cdot}m$

$\Rightarrow \quad P_j < \delta{\cdot}b/ a$. … (4.13.2)

Case 2.2.1: $b/a \leq b - a$

We know, $max_{AIRP}(i,j) < P_j$

$\Rightarrow max_{AIRP}(i,j) < \delta{\cdot}b/ a$ (from eq. 4.13.2).

Therefore,

$max_{AIRP}(i,j) - max_{AIRP}(j,i) < \delta{\cdot}b/a$ (since, $max_{AIRP}(j,i) = 0$)

since, $\delta < m$,

$max_{AIRP}(i,j) - max_{AIRP}(j,i) < m{\cdot}b/a$

since, $b/a \leq b - a$

$\Rightarrow m{\cdot}b/a \leq m{\cdot}b - m{\cdot}a$

$\Rightarrow max_{AIRP}(i,j) - max_{AIRP}(j,i) < m{\cdot}b - m{\cdot}a$

$\Rightarrow max_{AIRP}(i,j) - max_{AIRP}(j,i) < y$.

Case 2.2.2: $b/a > b - a$

Since, $T_j < 2{\cdot}T_i \Rightarrow b < 2{\cdot}a \Rightarrow b/a < 2$.

Being subject to the restriction $b/a < 2$, the condition, $b/a > b - a$ will never be satisfied for any $b > a+1$. Hence, this condition is only valid when $b = a+1$.

Since, $y = T_j - T_i$

$\Rightarrow \quad y = b{\cdot}m - a{\cdot}m = (a+1){\cdot}m - a{\cdot}m = m$.

As, $P_i = \delta$ and $x < y{\cdot}P_i / T_i$

$\Rightarrow \quad x < m{\cdot}\delta/a{\cdot}m$

also, $P_j = P_i + \delta$ and $x < \delta/a$

$\Rightarrow \quad P_j = \delta + x \Rightarrow P_j < \delta + \delta/a$.

The value of P_j will be maximized when $a = 1$. Or, $P_j < 2{\cdot}\delta$. Since, $\delta < m \Rightarrow \quad P_j < 2{\cdot}m$.

Hence, the only IRP *in* ψ_{IRP}(i,j) that can abort τ_j when P_j is maximum, is the IRP of length m. Or, for any value of P_j : $max_{AIRP}(i,j) \leq m$.

Therefore,

$max_{AIRP}(i,j) - max_{AIRP}(j,i) \leq m - 0$

$\Rightarrow \quad max_{AIRP}(i,j) - max_{AIRP}(j,i) \leq m$.

Since, $y = m$,

$max_{AIRP}(i,j) - max_{AIRP}(j,i) \leq y$. □

This theorem identifies an important property when :

$T_i < T_j < 2{\cdot}T_i$ and $P_i < P_j$: $P_i / T_i > P_j / T_j$,

and is used in the derivation of optimal priority assignment for 2-task sets in theorem 5.6(a).

V. Priority Assignment in 2-Task Sets

In this section, we evaluate priority assignment strategies for a P-FRP task set having 2 tasks. We show that the RM priority assignment is always optimal when one arrival period is more than double of the other. For task sets where arrival periods do not share this relationship, we derive conditions which determine schedulability under different priority assignments. Finally, we show that if a general 2-task P-FRP task set is known to be schedulable, then it will also be schedulable in either the UM or RM priority assignments.

Theorem 5.1. *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_j = f{\cdot}T_i$: $f \in Z+$; $f \geq 2$, then Γ_2 is schedulable under any priority assignment.*

Proof. Since, the P-FRP schedulability test is satisfied, we know:

$$P_i + P_j \leq T_i \Rightarrow \quad P_i + P_j \leq T_j / 2.$$

In this case, $L = f{\cdot}T_i$, therefore in the interval $[0,L)$ jobs of task τ_j will be released at times 0 and L, while jobs of τ_i will be released at 0, T_i, $2{\cdot}T_i$… L. Only the 1st job of τ_j has to be processed in the interval $[0,L)$. Let us take a look at two possible cases.

<u>**Case 1:**</u> $pr_i > pr_j$

The first job of τ_i will start and complete processing by time P_i, followed by the 1st job of τ_j. Since, as per the P-FRP schedulability test $P_i + P_j \leq T_i$, the 1st job of τ_j will be processed before the release of the 2nd job of τ_i. Hence, Γ_2 will be schedulable.

<u>**Case 2:**</u> $pr_j > pr_i$

The first job of τ_j will start and complete processing by time P_j, followed by the 1st job of τ_i. Since, $P_i + P_j \leq T_i$, the 1st job of τ_i will be processed before the release of its 2nd job. Hence, Γ_2 will be schedulable.

Clearly, all task sets $\Gamma_2=\{\tau_i, \tau_j\}$ where $T_j = f{\cdot}T_i$: $f \in Z+$; $f \geq 2$ will be schedulable under any priority assignment. □

Theorem 5.2(a). *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_j > 2{\cdot}T_i$, and $T_j \neq f{\cdot}T_i$: $f \in \mathbb{Z}^+$; $f > 2$ then Γ_2 will always be schedulable under the priority assignment $pr_i > pr_j$.*

Proof. Let the m^{th} job of τ_j be released at time t_a, when τ_i is not running. At time t_a+h: $t_{copy}(j) \leq h \leq t_{copy}(j) + C_j$, τ_j is aborted by the release of the p^{th} job of τ_i. τ_i will finish processing at time t_a+h+P_i, after which τ_j will re-start. The $(p+1)^{th}$ job of τ_i will be released at time t_a+h+T_i, while the $(m+1)^{th}$ job of τ_j is released at time $t_a + T_j$.

Since, $P_j < T_i$ and $h < P_j$

$\Rightarrow \quad h < T_i$

also, $T_i + T_i < T_j$, therefore,

$$h+T_i < T_j, \text{ or } t_a+h+T_i < t_a + T_j.$$

Hence, the $(p+1)^{th}$ job of τ_i will be released before the $(m+1)^{th}$ job of τ_j. From the P-FRP schedulability test we know:

$$P_i + P_j \leq T_i.$$

Both the p^{th} job of τ_i which starts processing at time t_a+h, and the m^{th} job of τ_j which re-starts processing at time t_a+h+P_i, will complete before the release of the $(p+1)^{th}$ job of τ_i. Hence, even if τ_i induces a maximum abort of $t_{copy}(j) + C_j$ on τ_i, τ_j will be schedulable.

If $h > t_{copy}(j) + C_j$, then τ_j will complete processing before the start of p^{th} job of τ_i, and both tasks will be schedulable. Hence, Γ_2 will always be schedulable if $T_j > 2{\cdot}T_i$ and $pr_i > pr_j$. □

Theorem 5.2(b). *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_j > 2{\cdot}T_i$ and $T_j \neq f{\cdot}T_i$: $f \in \mathbb{Z}^+$; $f > 2$,then the task set will be schedulable for the priority assignment $pr_j > pr_i$ only if, $T_i \geq P_i + P_j + max_{AIRP}(j,i)$.*

Proof. Since, the P-FRP schedulability test is satisfied we know:

$$T_i \geq P_i + P_j.$$

Assume that the p^{th} job of τ_i is released at time t_a, when τ_j is not running. At time t_a+h: $t_{copy}(i) \le h \le max_{AIRP}(j,i)$, τ_i is aborted by the m^{th} job of τ_j. τ_j will finish processing at time t_a+h+P_j after which τ_i will re-start processing and will complete at time $t_a+h+P_j+P_i$.

The $(m+1)^{th}$ job of τ_j is released at time t_a+h+T_j and the $(p+1)^{th}$ job of τ_i is released at time t_a+T_i. Since, $T_i \le T_j/2$

$\Rightarrow \quad t_a+h+T_j > t_a+T_i$.

Therefore, the $(p+1)^{th}$ job of τ_i will be released before the $(m+1)^{th}$ job of τ_j. To be schedulable the p^{th} job of τ_i will have to complete processing before the release of its $(p+1)^{th}$ job. Or

$t_a+h+P_i+P_j \le t_a+T_i$.

The maximum value of h in this case is: $h = max_{AIRP}(j,i)$.

Hence,

$t_a+max_{AIRP}(j,i)+P_j+P_i \le t_a+T_i$

$\Rightarrow \quad T_i \ge P_i + P_j + max_{AIRP}(j,i)$.

If the above inequality is satisfied then Γ_2 is guaranteed to be schedulable. □

Corollary 5.2.1. *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_j > 2\cdot T_i$, then the task set is guaranteed to be schedulable for the priority assignment $pr_j > pr_i$,only if, $T_i \ge P_i + P_j + t_{copy}(i)+C_i$.*

Proof. From theorem 5.2 we know that for Γ_2 to be schedulable:

$T_i \ge P_i + P_j + max_{AIRP}(j,i)$.

The maximum possible value of $max_{AIRP}(j,i)$ is $t_{copy}(i)+C_j$.

Hence, if

$T_i \ge P_i + P_j + t_{copy}(i) + C_i$,

then Γ_2 is guaranteed to be always schedulable. □

Theorem 5.3. *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_j > 2\cdot T_i$,then the rate-monotonic priority assignment is optimal.*

Proof. The rate-monotonic priority assignment is $pr_i > pr_j$. As shown in theorem 5.2(a), Γ_2 will always be schedulable in this priority assignment. Corollary 5.2.1 shows that Γ_2 is schedulable in the non-RM priority assignment $pr_j > pr_i$, only if certain conditions are met.

Since Γ_2 is schedulable in the RM priority assignment $pr_i > pr_j$ without pre-conditions, RM is the optimal priority assignment. □

Theorem 5.4. *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_i < T_j < 2\cdot T_i$, then the task set will only be schedulable in the priority assignment $pr_i > pr_j$ if, $T_j \ge P_i + P_j + max_{AIRP}(i,j)$.*

Proof. Since, the P-FRP schedulability test is satisfied we know:

$T_i - P_i \ge P_j$ and $T_j - P_j \ge P_i$.

Let the m^{th} job of τ_j be released at time t_a when τ_i is not running. At time t_a+h: $t_{copy}(j) \le h \le max_{AIRP}(i,j)$, τ_j is aborted by the p^{th} job of τ_i. τ_i will finish at time t_a+h+P_i after which τ_j will re-start processing and will complete at time $t_a+h+P_i+P_j$. The $(p+1)^{th}$ job of τ_i will be released at time t_a+h+T_i.

Since,

$T_i - P_i \ge P_j$

$\Rightarrow t_a+h+T_i \ge t_a+h+P_i+P_j$.

Hence, the p^{th} job of τ_i and the m^{th} job of τ_j will complete before the $(p+1)^{th}$ job of τ_i is released. Now the $(m+1)^{th}$ job of τ_j is released at time t_a+T_j. If this $(m+1)^{th}$ job is released before the m^{th} job has completed processing, τ_j will have a deadline miss. Or,

if, $t_a + T_j < t_a+h+P_i+P_j$, then τ_j is unschedulable.

Therefore, for τ_j to be schedulable: $t_a + T_j \geq t_a+h+P_i+P_j$.
$\Rightarrow \quad T_j \geq h+P_i+P_j$.

The upper bound of h is the maximum possible abort cost that can be induced on τ_j. Hence,
$h = max_{AIRP}(i,j)$.

Therefore, $T_j \geq P_i + P_j + max_{AIRP}(i,j)$. □

Corollary 5.4.1. *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_i < T_j < 2\cdot T_i$ then the task set is guaranteed to be schedulable in the priority assignment $pr_i > pr_j$, only if: $T_j \geq P_i + P_j + t_{copy}(j)+C_j$.*

Proof. From theorem 5.3 we know that for Γ_2 to be schedulable:
$T_i \geq P_i + P_j + max_{AIRP}(i,j)$.
The maximum possible value of $max_{AIRP}(j,i)$ is $t_{copy}(j)+C_j$.

Hence, if,
$T_i \geq P_i + P_j + t_{copy}(j) + C_j$,
then Γ_2 is guaranteed to be always schedulable under $pr_i > pr_j$. □

Theorem 5.5. *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_i < T_j < 2\cdot T_i$, then the task set will be schedulable in the priority assignment $pr_j > pr_i$, only if: $T_i \geq P_i + P_j + max_{AIRP}(j,i)$.*

Proof. Since, the P-FRP schedulability test is satisfied we know:
$T_i - P_i \geq P_j$ and $T_j - P_j \geq P_i$.

Assume that the p^{th} job of τ_i is released at time t_a, when τ_j is not running. At time t_a+h: $t_{copy}(i) \leq h \leq max_{AIRP}(j,i)$, τ_i is aborted by the m^{th} job of τ_j. τ_j will finish processing at time t_a+h+P_j after which τ_i will re-start and complete at time $t_a+h+P_j+P_i$. The $(m+1)^{th}$ job of τ_j will be released at time t_a+h+T_j. Since, $T_j - P_j \geq P_i$:

$t_a+h+T_j \geq t_a+h+P_j+P_i$.

Hence, the m^{th} job of τ_j and the p^{th} job of τ_i will complete before the $(m+1)^{th}$ job of τ_j is released. Now the $(p+1)^{th}$ job of τ_i is released at time t_a+T_i. If this $(p+1)^{th}$ job is released before the p^{th} job has completed processing τ_i will have a deadline miss. Or,

If, $t_a + T_i < t_a+h+P_j+P_i$, τ_i is unschedulable.

Therefore, to be schedulable: $t_a + T_i \geq t_a+h+P_j+P_i$

The maximum value of h is the maximum abort cost on τ_i. Hence, $h = max_{AIRP}(j,i)$. Therefore, for guaranteed schedulability of Γ_2 in $pr_j > pr_i$:
$T_i \geq P_i + P_j + max_{AIRP}(j,i)$. □

Corollary 5.5.1. *For $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_i < T_j < 2\cdot T_i$ then the task set is guaranteed to be always schedulable in the priority assignment $pr_j > pr_i$, only if: $T_i \geq P_i + P_j + t_{copy}(i)+C_j$.*

Proof. From theorem 5.5, we know that for Γ_2 to be schedulable:

$T_i \geq P_i + P_j + max_{AIRP}(i,j)$.

The maximum possible value of $max_{AIRP}(j,i)$ is $t_{copy}(j)+C_j$.
Hence, if,

$T_i \geq P_i + P_j + t_{copy}(j)+C_j$,

then Γ_2 is guaranteed to be always schedulable. □

Theorem 5.6(a). *Let $\Gamma_2=\{\tau_i, \tau_j\}$, $T_i < T_j < 2\cdot T_i$ and P_i and P_j have values such that $P_i / T_i > P_j / T_j$. If Γ_2 is schedulable in the priority assignment $pr_j > pr_i$, it is guaranteed to be schedulable in the priority assignment $pr_i > pr_j$.*

Proof. Let us look at all possible cases based on the relation between processing times of tasks τ_i and τ_j:

<u>Case 1</u>: $P_i \geq P_j$

Since, $T_i < T_j \Rightarrow P_i / T_i > P_j / T_j$.

Let Γ_2 be schedulable under: $pr_j > pr_i$.

From theorem 5.5 this implies:

$T_i \geq P_i + P_j + max_{AIRP}(j,i)$ … (5.6.1)

Since, $P_i \geq P_j$ and $t_{restore}(j) = t_{restore}(i)$ and $t_{copy}(i) = t_{copy}(j)$

$\Rightarrow max_{AIRP}(i,j) \leq t_{copy}(j)+C_j < t_{copy}(i)+C_i$.

From theorem 4.11 we know,

$\psi_{IRP}(i,j)= \psi_{IRP}(j,i)$

$\Rightarrow max_{AIRP}(i,j) \in \psi_{AIRP}(j,i)$

$\Rightarrow max_{AIRP}(i,j) \leq max_{AIRP}(j,i)$.

Therefore, in eq. 5.6.1,

$T_i \geq P_i + P_j + max_{AIRP}(i,j)$

Since, $T_j > T_i$,

$\Rightarrow T_j > P_i + P_j + max_{AIRP}(i,j)$ …(5.6.2)

If eq. (5.6.2) is satisfied, then as per theorem 5.4, Γ_2 will be schedulable under $pr_i > pr_j$.

Hence, if Γ_2 is schedulable in $pr_j > pr_i$, it is guaranteed to be schedulable in $pr_i > pr_j$.

<u>Case 2</u>: $P_i < P_j$: $P_i / T_i > P_j / T_j$

Let Γ_2 be schedulable under: $pr_j > pr_i$.

From theorem 5.4 this implies:

$T_i \geq P_i + P_j + max_{AIRP}(j,i)$ … (5.6.3)

Let $T_j = T_i + y$: $y \geq 1$

In eq. (5.6.3):

$T_i + y \geq P_i + P_j + max_{AIRP}(j,i) + y$

$\Rightarrow T_j \geq P_i + P_j + max_{AIRP}(j,i) + y$ … (5.6.4)

From theorem 4.13 we know:

$max_{AIRP}(i,j) – max_{AIRP}(j,i) \leq y$.

Therefore, in eq. (5.6.4),

$T_j \geq P_i + P_j + max_{AIRP}(j,i) + max_{AIRP}(i,j) – max_{AIRP}(j,i)$

$\Rightarrow T_j \geq P_i + P_j + max_{AIRP}(i,j)$ … (5.6.5)

As per theorem 5.4, if eq. (5.6.5) is satisfied then Γ_2 will be schedulable under $pr_i > pr_j$.

Hence, if Γ_2 is schedulable in $pr_j > pr_i$, it is guaranteed to be schedulable in $pr_i > pr_j$. □

Theorem 5.6(b). *Let $\Gamma_2=\{\tau_i, \tau_j\}$, if $T_i < T_j < 2\cdot T_i$ and P_i and P_j have values such that $P_i / T_i > P_j / T_j$. If, Γ_2 is schedulable in the priority assignment $pr_i > pr_j$, it can be unschedulable in the priority assignment $pr_j > pr_i$.*

Proof. If we can show one example which satisfies the conditions $T_i < T_j < 2\cdot T_i$, $P_i / T_i > P_j / T_j$ and is schedulable only under the priority assignment $pr_i > pr_j$, it is sufficient to prove this theorem.

Consider the following task set:

Task	*pr*	*P*	*T*	*U*
τ_1	1	3	12	0.25
τ_2	2	6	10	0.60

Here, $P_2 / T_2 > P_1 / T_1$. If we analyze the execution of this task set in its feasibility interval of [0,60), it is schedulable in the priority assignment $pr_2 > pr_1$. However in $pr_1 > pr_2$, the 2[nd] job of τ_2 has a deadline miss at time 20. An execution table for tasks τ_1 and τ_2 is available in appendix 1-A. □

Theorem 5.7. *For $\Gamma_2=\{\tau_i, \tau_j\}$, and $T_i < T_j < 2\cdot T_i$, if a priority assignment exists which is both utilization and rate-monotonic then this priority assignment is optimal for Γ_2.*

Proof. The RM-priority assignment is $pr_i > pr_j$. If $P_i / T_i > P_j / T_j$,then the UM priority assignment is also $pr_i > pr_j$. In theorem 5.6(a) we have seen that if these conditions are satisfied and Γ_2 is schedulable in $pr_j > pr_i$, it is guaranteed to be schedulable in $pr_i > pr_j$.

In theorem 5.6(b) we have seen that if Γ_2 is schedulable in $pr_i > pr_j$, it can be unschedulable in $pr_j > pr_i$. Hence, $pr_i > pr_j$ is the optimal priority assignment when this priority assignment is both UM and RM. □

Definition. A priority assignment which is both utilization and rate monotonic is henceforth referred to as the **U-RM priority assignment**.

Theorem 5.8. *For a 2-task set $\Gamma_2=\{\tau_i, \tau_j\}$, let $T_i < T_j < 2\cdot T_i$. If a U-RM priority assignment does not exist for Γ_2, then there is no single priority assignment which is optimal for all 2-task sets $\Gamma_2=\{\tau_i, \tau_j\}$ where $T_i < T_j < 2\cdot T_i$.*

Proof. A U-RM priority assignment will not exist for Γ_2 only when $P_i < P_j$ and $P_i / T_i < P_j / T_j$.

In this case the UM priority assignment is $pr_j > pr_i$ and the RM priority assignment is $pr_i > pr_j$. If we can show one example which is schedulable only in $pr_j > pr_i$, and a second example which is schedulable only in $pr_i > pr_j$ it is sufficient to prove this theorem.

Consider the task set used in lemma 4.1:

Task	*pr*	*P*	*T*	*U*
τ_1	1	7	15	0.46
τ_2	2	3	12	0.25

The RM-priority assignment is $pr_2 > pr_1$, while the UM-assignment is $pr_1 > pr_2$. As shown in lemma 4.1, this task set is schedulable only in $pr_1 > pr_2$.

Now, consider the following task set:

Task	*pr*	*P*	*T*	*U*
τ_1	1	6	15	0.40
τ_2	2	4	12	0.33

The RM-priority assignment is $pr_2 > pr_1$, while the UM-assignment is $pr_1 > pr_2$. If we analyze the execution of this task set in its feasibility interval of [0,60), it is schedulable in the priority assignment $pr_2 > pr_1$, but the 2nd job of τ_2 has a deadline miss at time 24, in $pr_1 > pr_2$. An execution table for τ_1 and τ_2 is available in appendix 1-B. □

Theorem 5.9. *If a 2-task set* $\Gamma_2=\{\tau_i, \tau_j\}$ *is known to be schedulable, then* Γ_2 *is guaranteed to be also schedulable in a rate-monotonic or utilization-monotonic priority assignments.*

Proof. Let's assume, $T_i < T_j$. There can be two possible cases.

Case 1: $2 \cdot T_i \leq T_j$

With this condition, Γ_2 is guaranteed to be always schedulable under the priority assignment $pr_i > pr_j$ as shown in theorems 5.1 and 5.3. $pr_i > pr_j$ is a rate-monotonic priority assignment.

Case 2: $T_i < T_j < 2 \cdot T_i$

With this condition there are two possible cases.

Case 2.1: $P_i / T_i > P_j / T_j$

The priority assignment $pr_i > pr_j$ is both a utilization and rate-monotonic priority assignment, and if Γ_2 is schedulable, it is guaranteed to be schedulable under this priority assignment, as shown in theorem 5.7.

Case 2.2: $P_i / T_i < P_j / T_j$

The utilization-monotonic priority assignment is $pr_j > pr_i$ and the rate-monotonic priority assignment is $pr_i > pr_j$. These are the only two priority assignments possible for this task set. Hence, if Γ_2 is schedulable, it is either under a utilization or rate-monotonic, or both, priority assignments. □

VI. Priority Assignment in *n*-Task Sets

In this section, we prove that for P-FRP task sets having n tasks ($n > 2$), a single priority assignment which is optimal exists only when task periods are integer multiples of each other. We also show that for other n-task sets, no priority assignment exists which is optimal for all task sets.

Theorem 6.1. *For,* $\Gamma_n=\{\tau_1, \tau_2 \ldots, \tau_n\}$*:* $n > 2$*, and task periods have the following relationships:* $T_i = f \cdot T_{i+1}$ *,* $i=1,n-1$*:* $f \geq 2$*;* $f \in \mathbb{Z}^+$*, then the rate-monotonic priority assignment is optimal.*

Proof. Consider $\Gamma_3 = \{\tau_i, \tau_j, \tau_k\}$. *Let ,*

$T_j = f \cdot T_i$, $T_k = g \cdot f \cdot T_i$, $g \geq 2$, $f \geq 2$; $f,g \in \mathbb{Z}^+$.

The rate-monotonic priority assignment is $pr_i > pr_j > pr_j$.

$L = g \cdot f \cdot T_i$, and there will be only one job of τ_k that has to be processed in $[0,L)$. Since the P-FRP schedulability test is satisfied we know:

$P_i + P_j \leq T_i$ and $P_i + P_k \leq T_i$.

The 1st job of τ_i will finish at time P_i and the 1st job of τ_j will be processed next as per the RM priority assignment. The 1st job of τ_j will finish at time $P_i + P_j$, after which the 2nd job of τ_i is released at time T_i. The second job of τ_j will not be released at least till time $2 \cdot T_i$, hence after the 2nd job of τ_i has completed, the single job of τ_k will complete processing. Jobs of two tasks τ_i and τ_j that will be released in the interval $[2 \cdot T_i, L)$ will be able to complete processing as per theorem 5.1.

This same analysis can be easily extended (by considering each pair of tasks) to any general task set, Γ_n which has tasks whose periods are integer multiples of each other. Since, such task sets will always be schedulable in the rate-monotonic priority assignment, this priority assignment is optimal. □

Lemma 6.2. *For, $\Gamma_n=\{\tau_1, \tau_2 \ldots, \tau_n\}$: $n > 2$, if task periods have the following relationships: $T_i > 2 \cdot T_{i+1}$ and $T_i \neq f \cdot T_{i+1}$, $i=1,n-1$: $f \geq 2$; $f \in \mathbb{Z}^+$,then the rate monotonic priority assignment is not optimal.*

Proof. If we can show one schedulable task set with this condition, which is not schedulable under the rate-monotonic priority assignment, it is sufficient to prove this lemma. Consider the task set Γ_3:

Task	*pr*	*P*	*T*	*U*
τ_1	1	8	60	0.13
τ_2	2	6	25	0.24
τ_3	3	3	12	0.25

The priority assignment $pr_3 > pr_2 > pr_1$ is RM. If we analyze the execution of this task set in its feasibility interval of [0, 300) with the RM priority assignment, the 4^{th} job of τ_1 has a deadline miss at time 240. If the priority assignment is changed to $pr_2 > pr_3 > pr_1$, jobs of all tasks are able to complete in the feasibility interval. An execution table for this task set is available in appendix 2-A. □

Lemma 6.3. *For, $\Gamma_n=\{\tau_1, \tau_2 \ldots, \tau_n\}$: $n > 2$, let task periods have the following relationships: $\forall \tau_i, \tau_j \in \Gamma_n$, $T_j < 2 \cdot T_i$. If a U-RM priority assignment exists for Γ_n, then it is not guaranteed to be the optimal priority assignment.*

Proof. If we can show one schedulable task set which is not schedulable in the U-RM priority assignment, it is sufficient to prove this lemma. Consider the task set Γ_3:

Task	*pr*	*P*	*T*	*U*
τ_1	1	3	16	0.18
τ_2	2	4	14	0.28
τ_3	3	4	12	0.33

The priority assignment $pr_3 > pr_2 > pr_1$ is both UM and RM. If we analyze the execution of this task set in its feasibility interval of [0,336), the 19^{th} job of τ_1 has a deadline miss at time 304. If the priority assignment is changed to $pr_1 > pr_3 > pr_2$, jobs of all tasks are able to complete in the feasibility interval. An execution table for this task set is available in appendix 2-B. □

Lemma 6.4. *Two tasks when executed independently will be schedulable in the same priority assignment, in which they were schedulable when part of a larger task set. Or, if $\Gamma_n=\{\tau_1,.., \tau_i,.. \tau_j, \ldots, \tau_n\}$ is schedulable in the priority assignment :*

$pr_1 > pr_2 > \ldots > pr_j \ldots > pr_i > \ldots > pr_n$

then $\Gamma_2 = \{\tau_i, \tau_j\}$ will also be schedulable in the priority assignment $pr_j > pr_i$.

Proof. Consider $\Gamma_n=\{\tau_1, \tau_2 \ldots, \tau_n\}$: $n > 2$.
Let Γ_n be schedulable under some priority assignment. Let this priority assignment be:

$pr_1 > \ldots > pr_j > \ldots > pr_i > .. > pr_{k-1} > pr_k$, where $\tau_i, \tau_j \in \Gamma_n$.

Let us remove the highest priority task τ_k: $\tau_k \neq \tau_i, \tau_j$ from Γ_n and, $\Gamma_{n-1} = \Gamma_n - \tau_k$.

As the priority assignment is same, the removal of the highest priority task can only lead to lesser chances of interference and hence, lower abort costs on all lower priority tasks. Therefore, if Γ_n is schedulable, Γ_{n-1} will also be schedulable.

Similarly, if we remove the highest priority task τ_k in Γ_{n-1} ($\neq \tau_i, \tau_j$), we get Γ_{n-2} which will also be schedulable in the same priority assignment. This way we can remove all higher priority tasks without affecting the schedulability of the task set. Once all tasks having higher priority than τ_i and τ_j are removed, let us remove all tasks having lower priority than τ_i and τ_j. Since lower priority tasks do not have any impact on the response time of higher priority tasks, their removal will not affect the schedulability of the task set. After removal of these lower and higher priority tasks we get the task set Γ_2={τ_i, τ_j}, which clearly is schedulable in the same priority assignment as was in Γ_n, $pr_j > pr_i$.

Similarly, we can see that any two tasks taken from Γ_n, will be schedulable in the same priority assignment as was in the Γ_n. □

Theorem 6.5. *For a task set* $\Gamma_n=\{\tau_1, \tau_2 \ldots ,\tau_n\}$*: $n > 2$, where task periods have the following relationships:* $T_i > f\cdot T_{i+1}$, $i=1,n-1$: $f \geq 2$; $f \in \mathbb{Z}^+$ *, then no priority assignment exists which is optimal for every* Γ_n.

Proof. Let us assume Γ_n is schedulable and an optimal priority assignment exists for this case. Let this priority assignment be:

$pr_1 > pr_2 > \ldots > pr_j \ldots > pr_i > \ldots > pr_n$.

Let Γ_2 ={τ_i, τ_j}: $\tau_i, \tau_j \in \Gamma_n$. From lemma 6.4 we know that if Γ_n is guaranteed to be schedulable in its optimal priority assignment, Γ_2 is also guaranteed to be schedulable in the priority assignment:

$pr_j > pr_i$.

This implies that $pr_j > pr_i$ is an optimal priority assignment for Γ_2. From lemma 6.2 we know the optimal priority assignment for Γ_n cannot be rate-monotonic, therefore $pr_j > pr_i$ is a non-RM priority assignment.

However, from theorem 5.3 we know that Γ_2 is guaranteed to be schedulable only under the RM priority assignment.

Since, the existence of an optimal priority assignment for Γ_n requires a guarantee from Γ_2 to be always schedulable under a non-RM priority assignment, we have a contradiction.

Hence, no priority assignment exists which is optimal for all Γ_n. □

Theorem 6.6. *For,* $\Gamma_n=\{\tau_1, \tau_2 \ldots ,\tau_n\}$*: $n > 2$, where task periods have the following relationships:* $\forall \tau_i, \tau_j \in \Gamma_n$, $T_j < 2\cdot T_i$, *no priority assignment exists which is optimal for all* Γ_n.

Proof. Let us assume Γ_n is schedulable, and an optimal priority assignment exists for this case. Let this priority assignment be:

$pr_1 > pr_2 > \ldots > pr_j \ldots > pr_i > \ldots > pr_n$.

Let Γ_2 ={τ_i, τ_j}: $\tau_i, \tau_j \in \Gamma_n$. From lemma 6.4 we know that if Γ_n is guaranteed to be schedulable in its optimal priority assignment, Γ_2 is also guaranteed to be schedulable in the same priority assignment:

$pr_j > pr_i$.

This implies that $pr_j > pr_i$ is an optimal priority assignment for Γ_2. There can be two possible cases.

Case 1: A U-RM priority assignment exists for Γ_n

From lemma 6.3 we know that the optimal priority assignment Γ_n cannot be U-RM, therefore $pr_j > pr_i$ is a priority assignment which is non-U-RM.

However, from theorem 5.7 we know that Γ_2 is guaranteed to be schedulable only under the U-RM priority assignment.

Since, the existence of an optimal priority assignment for Γ_n requires a guarantee from Γ_2 to be always schedulable under a non-U-RM priority assignment, we have a contradiction. Hence, no optimal priority assignment can exist for Γ_n in this case.

Case 2: A U-RM priority assignment does not exist for Γ_n

From theorem 5.7 we know that no optimal priority assignment can exist for Γ_2 in this case.

Since, the existence of an optimal priority assignment for Γ_n requires a guarantee from Γ_2 to also have an optimal priority assignment, we have a contradiction. Hence, no optimal priority assignment can exist for Γ_n in this case. □

n	U_{min}	U_{max}	**only UM**	**only RM**	**Non UM/RM**	**Both UM/RM**	**U-RM Exists**	**U-RM Schedulable**
2	0.10	0.50	0	0	0	500	314	314
2	0.51	1.0	2	20	0	478	364	364
3	0.10	0.50	1	0	0	499	191	191
3	0.51	1.0	84	21	26	369	175	165
4	0.10	1.0	150	16	74	260	107	94
5	0.10	1.0	120	32	244	104	82	40

Table 1: Task sets schedulable under different priority assignments

VII. Experimental Validation

We present an experimental validation of the results derived for tasks sets with 2 and more than 2 tasks, using randomly generated task sets in groups having 2,3,4 and 5 tasks. For task sets with 2 and 3 tasks we have further divided them into 2 categories, based on their utilization factors. Each group has 500 tasks, and task sets in each group are unique in the sense that at least 1 task is different between any two task sets present in a group.

The arrival periods and processing times of tasks are selected from sample ranges of [12, 32] and [3, 10], respectively. The values of these ranges were arbitrarily selected and kept low to reduce the time taken to perform the evaluation. As per our initial assumptions, $t_{copy}(k)$ and $t_{restore}(k)$ are set to 1. The task periods in every task set are selected such that no two tasks in a set have the same period.

For task sets with 2 tasks we simulated the execution of every task set in its feasibility interval using the $pr_i > pr_j$ and $pr_j > pr_i$ priority assignments. For task sets having 3,4, and 5 tasks we simulated the execution in each of the possible 3!, 4! and 5! priority assignments. If any task set was unschedulable in all the priority assignments, the task set was regenerated. Hence, each of the tasks sets we have simulated are schedulable in at least one priority assignment.

In *Table 1* we show the schedulability of task sets with different sizes and utilization factors in the UM or RM priority assignments. The number of task sets which are schedulable only by the UM priority assignment are classified under 'only-UM', while those schedulable only under RM are given under 'non-RM'. Task sets not schedulable under both the UM and RM priority assignments are given under 'non-RM/UM', while those schedulable by both UM and RM is given under 'Both UM/RM'. 'U-RM Exists' shows the number of tasks sets for which a U-RM priority assignment exists and the number of tasks schedulable in the U-RM priority assignment is given under 'U-RM Schedulable'. Clearly, every task set in 'U-RM Exists' will also be in 'Both UM/RM'.

When n=2, all task sets for which a U-RM priority assignment exists are schedulable under U-RM, which validates theorem 5.7. Also all the task sets are schedulable by a priority assignment which is either RM or UM, validating theorem 5.9. In cases where the U-RM priority assignment does not exist, some task sets are schedulable by only the RM or UM priority assignments, showing that no single optimal priority assignment exists in this case, as proved in theorem 5.8.

For n=3, while several task sets are schedulable by both UM and RM priority assignment, some are schedulable by a priority assignment which is non-UM/RM. Also for the higher utilization group, the 175 task sets for which a U-RM priority assignment exists, only 165 are schedulable in this priority assignment. Clearly, no single priority assignment is optimal for all 3-task sets which agrees with theorems 6.5 and 6.6. However, it is interesting that in the lower utilization group all task sets are schedulable by UM/RM and all 191 task sets for which a U-RM priority assignment exists, are also schedulable in this priority assignment.

The results for 4 and 5 task sets also agree with results derived for n-task sets in section 6. It is noteworthy that, even though no single optimal priority assignment exists, several n-task sets ($n > 2$) are still schedulable by UM or RM priority assignments, with UM clearly outperforming RM. As a general recommendation to engineers, for determining priority assignments under which an n-task set will be schedulable, first the schedulability in UM and RM priority assignments should be evaluated followed by rest of the ($n! - 2$) priority assignments.

VIII. Related Work

Response time analysis of P-FRP was first studied by Kaibachev et al [16], who derive response time bounds by placing restrictions on execution times of higher priority tasks. Ras and Cheng [22], have presented response time analysis and have compared the performance of P-FRP execution with priority inversion strategies. Both [16], [22] do not discuss priority assignment strategies for P-FRP. Response time analysis of transactional memory systems [14] has been done by Fahmy et al [11] while Manson et al [19] study response time of atomic processing of critical sections in Java. Anderson et al [1] have presented response time analysis of the lock-free execution. Lock-free is a mechanism to avoid priority inversion [23], the implementation of which is via an unconditional loop that terminates when the necessary updates to the shared resource are complete. Sivasankaran et al [25], have discussed priority assignment in real-time active databases. They have defined polices for parent, immediate and deferred transactions. The focus in this paper has been on dynamic priority assignment, which makes this unsuitable for our need as we are concerned with fixed assignment policies.

Notable work on fixed priority assignment strategies for the preemptive model have been done by Audsley [2] and Davis and Burns [7],[8]. In [2], an offline polynomial time algorithm that uses a transformation function to change the priorities of tasks, is presented. This paper also identifies minimum number of priority levels required for each task. This work has been extended in [5] to derive priority assignment in the presence of blocking. In [7], the concept of a 'robust' priority ordering for is introduced.

IX. Conclusion and Future Work

We have studied priority assignments in P-FRP and shown that unlike the classical model, a single priority assignment is not universally optimal, even for 2-task sets. However, for 2-task sets we have proven that if a single priority assignment is both UM and RM, then it is guaranteed to be the optimal priority assignment. It has also been proven that every schedulable 2-task will also be schedulable by a UM or RM priority assignments.

In [18], Liu and Layland also present their initial results using 2-task sets and then scale these methods for n-task sets in a fairly straight-forward way. Unfortunately, the execution model of P-FRP does not have this simplicity, and the optimality of UM or RM priority assignment do not hold true when there are more than 2 tasks. However, for n-task sets where task periods are double, or more than double or each other the RM priority assignment is optimal. For n-task sets where task periods do not share this relationship we prove that no single optimal priority assignment can exist. An algorithm based approach which evaluates all the possible $n!$ priority assignments is the only way to determine priority assignments under which such an n-task set is schedulable. It should however be noted, that a high percentage of P-FRP tasks sets having more than two tasks are schedulable by the UM/RM priority assignment, hence, in several situations analyzing all the $n!$ possible priority assignments might not be required.

While the classical preemptive model is well understood and several mature studies have been done over the past several years, the abort-restart model has not been thoroughly researched. Our work has characterized the execution of P-FRP, hence, the abort-restart model, using the concept of intermediate release points (IRP), abort-IRP and delay-IRP. Introducing such concepts is important because of the additional cost of abort which is dependent solely on the release time of jobs of higher priority tasks. The abort cost introduces a dynamic nature to the execution of tasks in P-FRP, analysis of which cannot be done by the variety of existing methods.

Our work gives system designers/engineers an important insight on the schedulability characteristics of a system implemented using P-FRP. This work will guide system designers on tweaking task parameters which enhance the schedulability of the task sets, as well as help them identify priority assignments where the task set will be unschedulable. In future work, we will enhance this study by considering more practical values of $t_{copy}(k)$ and $t_{restore}(k)$ for a task τ_k, and by determining if the optimal priority assignments derived in this paper, also hold true when tasks are released asynchronously.

References

[1] J. H. Anderson, S. Ramamurthy, K. Jeffay. "Real-time computing with Lock free Shared Objects". *ACM Transactions on Comp.Sys. 5(6), pp. 388-395*, 1997

[2] N. Audsley. “On priority-assignment in fixed priority scheduling”. *Information Processing Letters Volume 79, pp. 39-44*, 2001

[3] C. Belwal, A.M.K. Cheng. “On the Feasibility Interval for P-FRP”. Manuscript under review, *http://www2.cs.uh.edu/~cbelwal/FeasibilityInterval_PFRP.pdf*, 2011

[4] C. Belwal, A.M.K. Cheng. “On Priority Assignment in P-FRP ”. *RTAS’10 Work-In-Progress Session*, 2010

[5] K. Bletsas, N. Audsley. “Optimal priority assignment in the presence of blocking”. *Inf. Process. Lett.* 99, *3 (Aug. 2006), 83-86*, 2006

[6] J. Byun, A. Burns, A. Wellings. “A Worst-Case Behavior Analysis for Hard Real-time transactions”. *Workshop on Real-time Databases*, 1996

[7] R.I. Davis, A.Burns. “Robust Priority Assignment for Fixed Priority Real-Time Systems”. *RTSS’07*, *pp. 3-14*, 2007

[8] R.I. Davis, A. Burns. “Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems”. *RTSS’09*, *pp. 398-409*, 2009

[9] C. Elliott, P. Hudak. “Functional reactive animation”. *ICFP’97,pp. 263-273,* 1997

[10] Erlang, *http://www.erlang.org*

[11] S.F. Fahmy, B. Ravindran, E.D. Jensen. “Response time analysis of software transactional memory-based distributed real-time systems”. *ACM SAC Operating Systems*, 2009

[12] K. Hammond. “Chapter 1 – Is it Time for Real-Time Functional Programming”. *Trends in Functional Programming Volume 4 – Editor Stephen Gilmore, Intellect Ltd.*, 2005

[13] T. Hawkins. “Controlling Hybrid Vehicles with Haskell”. *Commerical Uses of Functional Languages (CUSP)’08*, 2008

[14] M. Herlihy, J.E.B. Moss. “Transactional memory: architectural support for lock-free data structures”. *ACM SIGARCH Computer Architecture New (Col. 21, Issue 2),pp. 289-300,* 1993

[15] Haskell, *http://www.haskell.org*

[16] R. Kaiabachev, W. Taha, A. Zhu. “E-FRP with Priorities”. *EMSOFT’07*, *pp. 221-230* , 2007

[17] J.Y.T. Leung, J. Whitehead. “On the complexity of fixed-priority scheduling of periodic, real-time tasks”. *Performance Evaluation (Netherlands) 2(4), pp. 237-250*, 1982

[18] C. L. Liu, L. W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”. *Journal of the ACM (Volume 20 Issue 1), pp. 46 - 61* , 1973

[19] J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin, J. Vitek. “Preemptible Atomic Regions for Real-Time Java”. *RTSS’05*, *pp.62-71*, 2005

[20] J. Peterson, G. D. Hager, P. Hudak. “A Language for Declarative Robotic Programming”. *ICRA’99, IEEE*, 1999

[21] J. Peterson, P.Hudak, A.Reid, G. D. Hager. “FVision: A Declarative Language for Visual Tracking”. *Symposium on Practical Aspects of Declarative Languages (PADL’01)*, 2001

[22] J. Ras, A. Cheng. “Response Time Analysis for the Abort-and-Restart Task Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm”. *RTCSA’09*, 2009

[23] L. Sha, R. Rajkumar, J. P. Lehoczky. “Priority Inheritance Protocols: An approach to Real Time Synchronization”. *Transactions on Computers Volume 39, Issue 9, pp.1175 – 1185*, 1990

[24] M. Swaine. “It's Time to Get Good at Functional Programming”. *Dr. Dobbs Journal, http://www.drdobbs.com, Dec‘ 2008*, 2008

[25] R. Sivasankaran, J. Stankovic, D. Towsley, B. Purimetla, K. Ramamritham. “Priority assignment in real-time active databases”. *The VLDB Journal* 5, 1, 019-034. 1996.

[26] Z. Wan, W. Taha, P. Hudak. “Real-time FRP”. *ICFP’01, pp. 146-156, ACM Press* ,2001

[27] Z. Wan, W. Taha, P. Hudak. “Task driven FRP”. *PADL’02.* 2002

[28] Z. Wan, P. Hudak. “Functional reactive programming from first principles”. *ACM SIGPLAN Conference on Programming Language Design and Implementation,pp.242-252*,2000

Appendix: 1-A

Task Set:

Task	*Pr*	*P*	*T*	*U*
τ_1	1	3	12	0.25
τ_2	2	6	10	0.6

U_RM Priority Assignment: $pr_2 > pr_1$; non-U_RM is $pr_1 > pr_2$

Time	Release	U_RM	Non-U_RM	Time	Release	U_RM	Non-U_RM
0	T1, T2	T2	T1	**31**		T2	
1		T2	T1	**32**		T2	
2		T2	T1	**33**		T2	
3		T2	T2	**34**		T2	
4		T2	T2	**35**		T2	
5		T2	T2	**36**	T1	T1	
6		T1	T2	**37**		T1	
7		T1	T2	**38**		T1	
8		T1	T2	**39**		*	
9		*	*	**40**	T2	T2	
10	T2	T2	T2	**41**		T2	
11		T2	T2	**42**		T2	
12	T1	T2	T1	**43**		T2	
13		T2	T1	**44**		T2	
14		T2	T1	**45**		T2	
15		T2	T2	**46**		*	
16		T1	T2	**47**		*	
17		T1	T2	**48**	T1	T1	
18		T1	T2	**49**		T1	
19		*	T2	**50**	T2	T2	
20	T2	T2	**Deadline Miss**	**51**		T2	
21		T2		**52**		T2	
22		T2		**53**		T2	
23		T2		**54**		T2	
24	T1	T2		**55**		T2	
25		T2		**56**		T1	
26		T1		**57**		T1	
27		T1		**58**		T1	
28		T1		**59**		*	
29		*					
30	T2	T2					

Appendix: 1-B

Task Set:

Task	*Pr*	*P*	*T*	*U*
τ_1	1	6	15	0.4
τ_2	2	4	12	0.33

RM Priority Assignment: $pr_2 > pr_1$; UM is $pr_1 > pr_2$

Time	**Release**	**RM**	**UM**	**Time**	**Release**	**RM**	**UM**
0	T1, T2	T2	T1	**31**		T1	
1		T2	T1	**32**		T1	
2		T2	T1	**33**		T1	
3		T2	T1	**34**		T1	
4		T1	T1	**35**		T1	
5		T1	T1	**36**	T2	T2	
6		T1	T2	**37**		T2	
7		T1	T2	**38**		T2	
8		T1	T2	**39**		T2	
9		T1	T2	**40**	T1		
10				**41**			
11				**42**			
12	T2	T2	T2	**43**			
13		T2	T2	**44**			
14		T2	T2	**45**		T1	
15	T1	T2	T1	**46**		T1	
16		T1	T1	**47**		T1	
17		T1	T1	**48**	T2	T2	
18		T1	T1	**49**		T2	
19		T1	T1	**50**	T1	T2	
20		T1	T1	**51**		T2	
21		T1	T2	**52**		T1	
22			T2	**53**		T1	
23			T2	**54**		T1	
24	T2	T2	**Deadline Miss**	**55**		T1	
25		T2		**56**		T1	
26		T2		**57**		T1	
27		T2		**58**			
28				**59**			
29							
30	T1	T1					

Appendix: 2-A

Task Set:

Task	*Pr*	*P*	*T*	*U*
τ_1	1	8	60	0.13
τ_2	2	6	25	0.24
τ_3	3	3	12	0.25

RM Priority Assignment: $pr_3 > pr_2 > pr_1$; non-RM is $pr_2 > pr_3 > pr_1$

Time	Release	Non-RM	RM
0	T1,T2,T3	T2	T3
1		T2	T3
2		T2	T3
3		T2	T2
4		T2	T2
5		T2	T2
6		T3	T2
7		T3	T2
8		T3	T2
9		T1	T1
10		T1	T1
11		T1	T1
12	T3	T3	T3
13		T3	T3
14		T3	T3
15		T1	T1
16		T1	T1
17		T1	T1
18		T1	T1
19		T1	T1
20		T1	T1
21		T1	T1
22		T1	T1
23			
24	T3	T3	T3
25	T2	T2	T3
26		T2	T3
27		T2	T2
28		T2	T2
29		T2	T2
30		T2	T2
31		T3	T2
32		T3	T2
33		T3	
34			
35			
36	T3	T3	T3
37		T3	T3
38		T3	T3
39			
40			
41			
42			
43			
44			
45			
46			
47			
48	T3	T3	T3
49		T3	T3
50	T2	T2	T3
51		T2	T2
52		T2	T2
53		T2	T2
54		T2	T2
55		T2	T2

Time	Release	Non-RM	RM
56		T3	T2
57		T3	
58		T3	
59			
60	T1,T3	T3	T3
61		T3	T3
62		T3	T3
63		T1	T1
64		T1	T1
65		T1	T1
66		T1	T1
67		T1	T1
68		T1	T1
69		T1	T1
70		T1	T1
71			
72	T3	T3	T3
73		T3	T3
74		T3	T3
75	T2	T2	T2
76		T2	T2
77		T2	T2
78		T2	T2
79		T2	T2
80		T2	T2
81			
82			
83			
84	T3	T3	T3
85		T3	T3
86		T3	T3
87			
88			
89			
90			
91			
92			
93			
94			
95			
96	T3	T3	T3
97		T3	T3
98		T3	T3
99			
100	T2	T2	T2
101		T2	T2
102		T2	T2
103		T2	T2
104		T2	T2
105		T2	T2
106			
107			
108	T3	T3	T3
109		T3	T3
110		T3	T3

Time	Release	Non-RM	RM
111			
112			
113			
114			
115			
116			
117			
118			
119			
120	T1,T3	T3	T3
121		T3	T3
122		T3	T3
123		T1	T1
124		T1	T1
125	T2	T2	T2
126		T2	T2
127		T2	T2
128		T2	T2
129		T2	T2
130		T2	T2
131		T1	T1
132	T3	T3	T3
133		T3	T3
134		T3	T3
135		T1	T1
136		T1	T1
137		T1	T1
138		T1	T1
139		T1	T1
140		T1	T1
141		T1	T1
142		T1	T1
143			
144	T3	T3	T3
145		T3	T3
146		T3	T3
147			
148			
149			
150	T2	T2	T2
151		T2	T2
152		T2	T2
153		T2	T2
154		T2	T2
155		T2	T2
156	T3	T3	T3
157		T3	T3
158		T3	T3
159			
160			
161			
162			
163			
164			
165			

Time	Release	RM	non-RM	Time	Release	RM	non-RM	Time	Release	RM	non-RM
166				221			T1	276	T3	T2	
167				222			T1	277		T2	
168	T3	T3	T3	223			T1	278		T2	
169		T3	T3	224			T1	279		T2	
170		T3	T3	225	T2	T2	T2	280		T2	
171				226		T2	T2	281		T3	
172				227		T2	T2	282		T3	
173				228	T3	T2	T3	283		T3	
174				229		T2	T3	284			
175	T2	T2	T2	230		T2	T3	285			
176		T2	T2	231		T3	T2	286			
177		T2	T2	232		T3	T2	287			
178		T2	T2	233		T3	T2	288	T3	T3	
179		T2	T2	234			T2	289		T3	
180	T1,T3	T2	T3	235			T2	290		T3	
181		T3	T3	236			T2	291			
182		T3	T3	237			T1	292			
183		T3	T2	238			T1	293			
184		T1	T2	239			T1	294			
185		T1	T2	240	T1,T3	T3	**Deadline Miss**	295			
186		T1	T2	241		T3		296			
187		T1	T2	242		T3		297			
188		T1	T2	243		T1		298			
189		T1	T1	244		T1		299			
190		T1	T1	245		T1					
191		T1	T1	246		T1					
192	T3	T3	T3	247		T1					
193		T3	T3	248		T1					
194		T3	T3	249		T1					
195			T1	250	T2	T2					
196			T1	251		T2					
197			T1	252	T3	T2					
198			T1	253		T2					
199			T1	254		T2					
200	T2	T2	T2	255		T2					
201		T2	T2	256		T3					
202		T2	T2	257		T3					
203		T2	T2	258		T3					
204	T3	T2	T3	259		T1					
205		T2	T3	260		T1					
206		T3	T3	261		T1					
207		T3	T2	262		T1					
208		T3	T2	263		T1					
209			T2	264	T3	T3					
210			T2	265		T3					
211			T2	266		T3					
212			T2	267		T1					
213			T1	268		T1					
214			T1	269		T1					
215			T1	270		T1					
216	T3	T3	T3	271		T1					
217		T3	T3	272		T1					
218		T3	T3	273		T1					
219			T1	274		T1					
220			T1	275	T2	T2					

Appendix: 2-B

Task Set:

Task	*Pr*	*P*	*T*	*U*
τ_1	1	3	16	0.18
τ_2	2	4	14	0.28
τ_3	3	4	12	0.33

U_RM Priority assignment: $pr_3 > pr_2 > pr_1$; non-U_RM is $pr_1 > pr_3 > pr_2$

Time	**Release**	**non-U_RM**	**U_RM**	**Time**	**Release**	**non-U_RM**	**U_RM**	**Time**	**Release**	**non-U_RM**	**U_RM**
0	T1,T2,T3	T1	T3	**55**				**111**		T3	T3
1		T1	T3	**56**	T2	T2	T2	**112**	T1,T2	T1	T2
2		T1	T3	**57**		T2	T2	**113**		T1	T2
3		T3	T3	**58**		T2	T2	**114**		T1	T2
4		T3	T2	**59**		T2	T2	**115**		T2	T2
5		T3	T2	**60**	T3	T3	T3	**116**		T2	T1
6		T3	T2	**61**		T3	T3	**117**		T2	T1
7		T2	T2	**62**		T3	T3	**118**		T2	T1
8		T2	T1	**63**		T3	T3	**119**			
9		T2	T1	**64**	T1	T1	T1	**120**	T3	T3	T3
10		T2	T1	**65**		T1	T1	**121**		T3	T3
11				**66**		T1	T1	**122**		T3	T3
12	T3	T3	T3	**67**				**123**		T3	T3
13		T3	T3	**68**				**124**			
14	T2	T3	T3	**69**				**125**			
15		T3	T3	**70**	T2	T2	T2	**126**	T2	T2	T2
16	T1	T1	T2	**71**		T2	T2	**127**		T2	T2
17		T1	T2	**72**	T3	T3	T3	**128**	T1	T1	T2
18		T1	T2	**73**		T3	T3	**129**		T1	T2
19		T2	T2	**74**		T3	T3	**130**		T1	T1
20		T2	T1	**75**		T3	T3	**131**		T2	T1
21		T2	T1	**76**		T2	T2	**132**	T3	T3	T3
22		T2	T1	**77**		T2	T2	**133**		T3	T3
23				**78**		T2	T2	**134**		T3	T3
24	T3	T3	T3	**79**		T2	T2	**135**		T3	T3
25		T3	T3	**80**	T1	T1	T1	**136**		T2	T1
26		T3	T3	**81**		T1	T1	**137**		T2	T1
27		T3	T3	**82**		T1	T1	**138**		T2	T1
28	T2	T2	T2	**83**				**139**		T2	
29		T2	T2	**84**	T2,T3	T3	T3	**140**	T2	T2	T2
30		T2	T2	**85**		T3	T3	**141**		T2	T2
31		T2	T2	**86**		T3	T3	**142**		T2	T2
32	T1	T1	T1	**87**		T3	T3	**143**		T2	T2
33		T1	T1	**88**		T2	T2	**144**	T1,T3	T1	T3
34		T1	T1	**89**		T2	T2	**145**		T1	T3
35				**90**		T2	T2	**146**		T1	T3
36	T3	T3	T3	**91**		T2	T2	**147**		T3	T3
37		T3	T3	**92**				**148**		T3	T1
38		T3	T3	**93**				**149**		T3	T1
39		T3	T3	**94**				**150**		T3	T1
40				**95**				**151**			
41				**96**	T1,T3	T1	T3	**152**			
42	T2	T2	T2	**97**		T1	T3	**153**			
43		T2	T2	**98**	T2	T1	T3	**154**	T2	T2	T2
44		T2	T2	**99**		T3	T3	**155**		T2	T2
45		T2	T2	**100**		T3	T2	**156**	T3	T3	T3
46				**101**		T3	T2	**157**		T3	T3
47				**102**		T3	T2	**158**		T3	T3
48	T1,T3	T1	T3	**103**		T2	T2	**159**		T3	T3
49		T1	T3	**104**		T2	T1	**160**	T1	T1	T2
50		T1	T3	**105**		T2	T1	**161**		T1	T2
51		T3	T3	**106**		T2	T1	**162**		T1	T2
52		T3	T1	**107**				**163**		T2	T2
53		T3	T1	**108**	T3	T3	T3	**164**		T2	T1
54		T3	T1	**109**		T3	T3	**165**		T2	T1

Time	**Release**	**U_RM**	**non-**	**Time**	**Release**	**U_RM**	**non-U_RM**	**Time**	**Release**	**U_RM**	**non-U_RM**
166		T2	T1	**223**				**280**	T2	T2	T2
167				**224**	T1,T2	T1	T2	**281**		T2	T2
168	T2, T3	T3	T3	**225**		T1	T2	**282**		T2	T2
169		T3	T3	**226**		T1	T2	**283**		T2	T2
170		T3	T3	**227**		T2	T2	**284**			
176	T1	T1	T1	**233**		T2	T1	**290**		T1	T3
177		T1	T1	**234**		T2	T1	**291**		T3	T3
178		T1	T1	**235**		T2		**292**		T3	T1
179				**236**				**293**		T3	T1
180	T3	T3	T3	**237**				**294**	T2	T3	T2
181		T3	T3	**238**	T2	T2	T2	**295**		T2	T2
182	T2	T3	T3	**239**		T2	T2	**296**		T2	T2
183		T3	T3	**240**	T1,T3	T1	T3	**297**		T2	T2
184		T2	T2	**241**		T1	T3	**298**		T2	T1
185		T2	T2	**242**		T1	T3	**299**			T1
186		T2	T2	**243**		T3	T3	**300**	T3	T3	T3
187		T2	T2	**244**		T3	T2	**301**		T3	T3
188				**245**		T3	T2	**302**		T3	T3
189				**246**		T3	T2	**303**		T3	T3
190				**247**		T2	T2	**304**	T1	T1	**Deadline Miss**
191				**248**		T2	T1	**305**		T1	
192	T1,T3	T1	T3	**249**		T2	T1	**306**		T1	
193		T1	T3	**250**		T2	T1	**307**			
194		T1	T3	**251**				**308**	T2	T2	
195		T3	T3	**252**	T2,T3	T3	T3	**309**		T2	
196	T2	T3	T2	**253**		T3	T3	**310**		T2	
197		T3	T2	**254**		T3	T3	**311**		T2	
198		T3	T2	**255**		T3	T3	**312**	T3	T3	
199		T2	T2	**256**	T1	T1	T2	**313**		T3	
200		T2	T1	**257**		T1	T2	**314**		T3	
201		T2	T1	**258**		T1	T2	**315**		T3	
202		T2	T1	**259**		T2	T2	**316**			
203				**260**		T2	T1	**317**			
204	T3	T3	T3	**261**		T2	T1	**318**			
205		T3	T3	**262**		T2	T1	**319**			
206		T3	T3	**263**				**320**	T1	T1	
207		T3	T3	**264**	T3	T3	T3	**321**		T1	
208	T1	T1	T1	**265**		T3	T3	**322**	T2	T1	
209		T1	T1	**266**	T2	T3	T3	**323**		T2	
210	T2	T1	T2	**267**		T3	T3	**324**	T3	T3	
211		T2	T2	**268**		T2	T2	**325**		T3	
212		T2	T2	**269**		T2	T2	**326**		T3	
213		T2	T2	**270**		T2	T2	**327**		T3	
214		T2	T1	**271**		T2	T2	**328**		T2	
215			T1	**272**	T1	T1	T1	**329**		T2	
216	T3	T3	T3	**273**		T1	T1	**330**		T2	
217		T3	T3	**274**		T1	T1	**331**		T2	
218		T3	T3	**275**				**332**			
219		T3	T3	**276**	T3	T3	T3	**333**			
220			T1	**277**		T3	T3	**334**			
221			T1	**278**		T3	T3	**335**			
222			T1	**279**		T3	T3				