Release Offset Bounds for Response Time Analysis of P-FRP using Exhaustive Enumeration

Chaitanya Belwal
Dept. of Computer Science,
University of Houston, USA
cbelwal@cs.uh.edu

Albert M.K. Cheng
Dept. of Computer Science,
University of Houston, USA
cheng@cs.uh.edu

Walid Taha
School of Info. Sc., Computer and Electrical Engineering
Halmstad University,Sweden
Walid.Taha@hh.se

Abstract - **Functional Reactive Programming (FRP) is a declarative approach to modeling and building reactive systems. Priority-based FRP (P-FRP) is a formalism of FRP that guarantees real-time response. Unlike the classical preemptive model[1] of real-time systems, preempted tasks in P-FRP are aborted and have to restart when higher priority tasks have completed. Due to this abort-restart of nature of preemption, there is no single critical instant of release that leads to Worst-Case Response Time (WCRT) of lower priority P-FRP tasks. At this time, the only method for determining the WCRT is through an exhaustive enumeration of all release offsets of higher priority tasks between the release and deadline of the lower priority task. This makes the computational cost of WCRT dependent on the deadline of a task, and when such deadlines are large the computational costs of this technique make it infeasible even for small task sets. In this paper, we show that the release offsets of higher priority tasks have a lower and upper bound and present techniques to derive these bounds. By enumerating only those release offsets while lie within our derived bounds the number of release scenarios that have to be enumerated is significantly reduced. This leads to lower computational costs and makes determination of the WCRT in P-FRP a practically feasible proposition**.

Keywords – Functional programming, real-time system, schedulability analysis, response time, exhaustive enumeration

I. INTRODUCTION

Functional Reactive Programming (FRP) [25] is a declarative programming language for modeling and implementing reactive systems. It has been used for a wide range of applications, notably, graphics [9], robotics [18], and vision [19]. 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 [11], 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 ([15],[23],[24]). It was shown that a variant called priority-based FRP (P-FRP) [15] combines both the semantic properties of FRP, guarantees resource boundedness, and supports assigning different priorities to different events. In P-FRP, higher priority events can preempt lower-priority ones. However, to maintain guarantees of type safety and stateless execution, the functional programming paradigm requires the execution of a function to continue uninterrupted. To comply with this requirement, as well as allow preemption of lower priority events, P-FRP implements a transactional model of execution. Using only a copy of the state during event execution and atomically committing these changes at the end of the event handler (or *task*), P-FRP ensures that handling an event is an "all or nothing" proposition. This preserves 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. Thus, a clear separation between the semantics of the program and the responsiveness of each handler is achieved.

The transactional execution model used in P-FRP is not new, and several such models have been presented in the past. These are the transactional memory systems [12], lock-free execution for critical sections [1] and preemptable atomic regions in Java [17]. While several response time studies of this model are available ([1],[10], [17]), they only provide basic schedulability analysis by modifying existing methods developed for the preemptive model.

A fundamental assumption made in most response-time studies for the preemptive model is that the amount of processor time that a task will take to complete execution cannot be more than its worst-case execution time (WCET). However, since a preempted task is aborted and eventually restarted, the amount of processor time taken by P-FRP tasks to complete execution can be larger than their fixed WCET and therefore, not known *a priori*. Due to this there is no single *critical instant* [16] of release of higher priority tasks that lead to the Worst-Case Response Time (WCRT) for a lower priority P-FRP task (see example in section 2.2.2).

Currently, the only way of determining WCRT in P-FRP is to compute the response time under all possible release scenarios of higher priority tasks. The response time of a lower priority task τ_j depends upon job releases of higher priority tasks between the release and deadline of τ_j. Hence to determine the WCRT of task τ_j which has a relative deadline of D_j and is released at time t, we have to compute the response time of τ_j under all possible release scenarios of higher priority tasks in the interval $[t,t+D_j)$. This requires enumerating every possible combination of release offsets (release time of the 1st job of a task), where the release offset values lie in the range $[t,t+D_j)$. The release offset of tasks in

[*] This work is supported in part by U.S. National Science Foundation under Award nos. 0720856 and 0720857

[1] 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

a combination do not have to be unique since it is possible for multiple tasks to be released simultaneously.

For a task set of size n, the total number of enumerations whose response time has to be evaluated is:

$$(D_j - t + t+1)^{n-j} = (D_j+1)^{|HP|} \qquad \ldots(1.1)$$

where $|HP|$ denotes the total number of tasks present in the set having higher priority than τ_j. It is clear that the number of enumerations and hence the computational cost, is dependent on the deadline of τ_j as well as the size of the task set. In this paper, we show that to compute the WCRT of τ_j there exist lower and upper bounds on the release offset of higher priority tasks. Due to the existence of these bounds, enumeration of all release offset in the range $[t,t+D_i)$ is not required resulting in significant computational savings. The previous works in P-FRP only deal with computing approximate values of WCRT ([15],[21]) or computing response time for a given release scenario ([3],[4]). None of the prior works deal with methods to compute exact WCRT.

Note that it may be possible to reduce the number of higher priority tasks whose release offsets have to be considered. This will lower the number of required enumerations even further (by lowering the value of $|HP|$ in eq. (1.1)). However, we are not aware of any such solution and to the best of our knowledge this remains an open problem till this date.

A. Contributions

This paper presents techniques for determining the lower and upper bound on release offset of higher priority tasks for computation of WCRT in P-FRP. After presenting the execution model (Section 2) we highlight important schedulability characteristics of P-FRP (Section 3). Using the schedulability characteristics presented, we show that release offsets of higher priority tasks can be bounded, and derive methods and an algorithm to compute these release offset bounds (Section 4). Experimental results which highlight the significant computational savings made possible by our method are presented (Section 5). We conclude with a review of related work (Section 6) and a reflection on our results (Section 7).

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 dual concept of arrival rate, and their processing time; the concept of a time interval and release offset therein. For our task model all these are assumed to be known *a priori*. 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 $\tau_k \in \Gamma_n$ is the positive integer k, where a higher number implies higher priority
- 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 **worst-case execution time** for τ_k
- $t_{copy}(k)$ is the time taken to make a **copy** of the state before τ_k starts execution (see section 2.2.1)
- $t_{restore}(k)$ is the time taken to **restore** the state after τ_k has completed execution (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, $P_k = t_{copy}(k) + C_k + t_{restore}(k)$
- $R_{k,m}$ 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, $R_{k,m} = \Phi_k + (m-1)\cdot T_k$
- $[t_1, t_2)$ represents the **time interval** such that: $\forall t \in [t_1, t_2)$, $t_1 \le t < t_2 \wedge t_1 \ne t_2$
- D_k is the **relative deadline** of τ_k. If some job of τ_k is released at absolute time $R_{k,m}$ then τ_k should complete processing by absolute time $R_{k,m} + D_k$, otherwise τ_k will have a **deadline miss.** In this paper, $D_k = T_k$
- The **total utilization factor** (U) of a task set is the sum of ratios of processing time to arrival periods of every task. Hence, $U = \sum_{i=1}^{n} \frac{P_i}{T_i}$
- **Interference** on τ_k is the action where the processing of τ_k is interrupted by the release of a higher priority task

B. Execution Model and Assumptions

In this study, all tasks are assumed to execute in a uniprocessor system. When job of a higher priority task is released, it can immediately preempt an executing lower priority task, and changes made by the lower priority task are rolled back. The lower priority task will be restarted when the higher priority task has completed processing. All tasks are assumed to run without precedence constraints.

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. All time properties have discrete integer values.

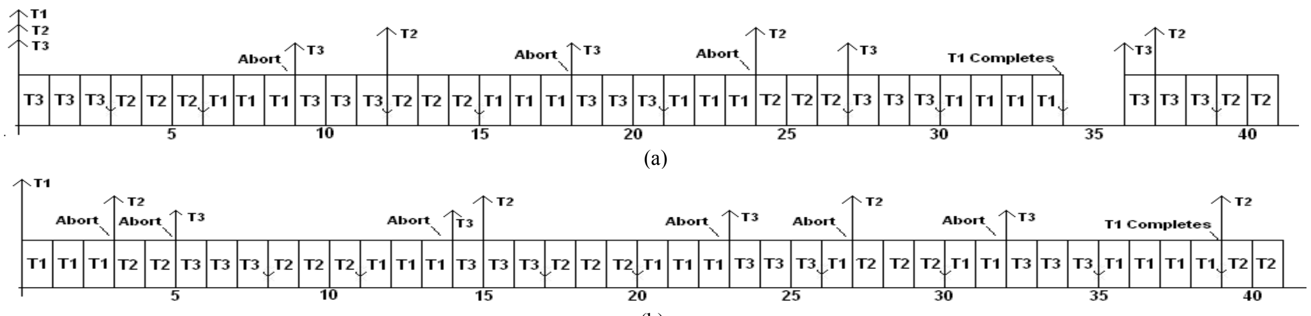


Figure 1. (a) When tasks are released synchronously, τ_1 completes processing at time 34. T1, T2 and T3 represent tasks τ_1, τ_2 and τ_3 respectively. The 'Abort' tag shows the time when τ_1 or τ_2 are aborted by higher priority tasks The total abort costs induced on the 1st job of τ_1 is 9 (b) When τ_3 is released at time 5 and τ_2 is released at time 3, τ_1 completes processing at time 39. The total abort costs induced on the 1st job of τ_1 is 12

Once τ_i enters the queue two situations are possible. If a task of lower priority than i is being processed, it will be immediately preempted and τ_i will start processing. If a task of higher priority than 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 explained in the following paragraph.

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 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 its processing time, P_k.

Since the abort-restart semantic has sufficient complexity in itself, at this time we do not consider situations where higher priority tasks do not preempt lower priority tasks. Hence, for the methods presented in this paper, the values of $t_{copy}(k)$ and $t_{restore}(k)$ for all tasks are kept same and equal to a single discrete time unit of processing. Hence,

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

Note, that copy and restore operations are only a fraction of the execution time of the task [15], hence this assumption is reasonable. However for greater precision, in future work we will develop methods where the values of $t_{restore}(k)$ and $t_{copy}(k)$ are variable.

2. Critical Instant in P-FRP

In response time analysis for fixed-priority scheduling a *critical-instant* of release is assumed. Critical instant is the time at which task releases lead to the WCRT of the task under study. In their seminal work, Liu and Layland [16] showed that in fixed-priority scheduling for the preemptive model, the critical-instant occurs when a lower priority task is released at the same time as all higher priority tasks. In P-FRP, same release time of tasks does not lead to the WCRT for all cases. Consider the following example:

$\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$, and the arrival time period between jobs of tasks τ_1, τ_2 and τ_3 are 45, 12 and 9 with their processing times being 4, 3 and 3 respectively. If the release of all tasks is at t=0 in [0, 40), τ_1 completes processing at t=34 as shown in *Fig. 1(a)*. If τ_3 is released at t=5 and τ_2 is released at t=3, then τ_1 completes processing at t=39 (*Fig. 1(b)*). Hence, the notion of critical instance in P-FRP is clearly not the simultaneous release of all tasks.

III. SCHEDULABILITY CHARACTERISTICS

In this section, we formally state two schedulability characteristics of the P-FRP execution model based on which we derive a necessary schedulability test. Before that, we introduce the important definitions of abort and interference cost.

Definition: In the preemptive model of execution, if a higher priority τ_i interferes with the execution of a lower priority 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 3.1: *If a P-FRP task set is schedulable, tasks will be able to complete processing between successive jobs of any other task present in the set.*

Proof. Consider a task set with only two tasks: $\Gamma_2 = \{\tau_i, \tau_j\}$ and $i > j$. Assume all tasks are released simultaneously (A similar analysis holds true for an asynchronous release). There are two possible cases.

Case 1: $T_i \leq T_j$

Since, both tasks are released simultaneously, τ_i will complete first. τ_j will start processing when τ_i has completed which is at time $0+P_i$. The next job of τ_i will be released 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 the 2nd job of τ_i which is released at time T_i. After the 2nd job of τ_i has completed, τ_j will get another time period of length $T_i - P_i$ to complete processing. 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 processing and the task set will be unschedulable. Hence, if Γ_2 is schedulable, τ_j has to complete processing between successive jobs of τ_i. Or,

$$P_j \le T_i - P_i \qquad \ldots (3.1)$$
$$\Rightarrow \; P_i \le T_i - P_j.$$

Since, $T_i < T_j$:

$$P_i \le T_j - P_j \qquad \ldots (3.2)$$

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

If both tasks are released simultaneously, then τ_i will complete first. τ_j will start processing when τ_i has completed which is at time $0+P_i$. The next job of τ_j will be released at time T_j, before the 2nd job of τ_i is released. The time left for processing the 1st job of τ_j is $T_j - P_i$. If τ_j is unable to complete processing within this time it will miss its deadline. Hence if Γ_2 is schedulable, τ_j has to complete processing between its two successive jobs after accounting for the processing of τ_i. Therefore,

$$P_j \le T_j - P_i \qquad \ldots (3.3)$$
$$\Rightarrow \; P_i \le T_j - P_j \qquad \ldots (3.4)$$

In eq. (3.3) since $T_i > T_j$,

$$P_j \le T_i - P_i \qquad \ldots (3.5)$$

Eqs. (3.1), (3.2), (3.4) and (3.5) show that if Γ_2 is schedulable, each task will be able to complete processing between successive jobs of the other task.

If, $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$, we can do the above analysis for each unique pair of two tasks. Start with the pair $\{\tau_1, \tau_2\}$, then $\{\tau_1, \tau_3\}$ through $\{\tau_1, \tau_n\}$. Then the pairs $\{\tau_2, \tau_3\}$ through $\{\tau_2, \tau_n\}$ will be analyzed. This way, by applying the above method for each unique pair we can show that if Γ_n is schedulable, each task in Γ_n will be able to complete processing between jobs of other tasks present in the set. □

Lemma 3.2 ([5], 3.1): *If Γ_n is schedulable, then the combined utilization factor (U) of all tasks in Γ_n will be less than or equal to unity. Or, if,* $U = \sum_{i=1}^{n} \frac{P_i}{T_i}$ *, then* $U \le 1$.

Definition: Lemmas 3.1 and 3.2 define necessary schedulability conditions for P-FRP tasks. These conditions are necessary but not sufficient since the satisfaction of these conditions alone does not guarantee the schedulability of a task set. The satisfiability of these two conditions is termed as the **P-FRP necessary schedulability test** in the rest of this paper.

Lemma 3.3: *For $\Gamma_2 = \{\tau_i, \tau_j\}$: $i > j$, the maximum abort cost induced by τ_i on τ_j is: $t_{copy}(j) + C_j$.*

Proof. Let τ_j be released at time t and execute for h time units, after which a job of the higher priority task τ_i is released. τ_i will abort τ_j only if τ_j has not started to restore its state. Hence, if $h \le t_{copy}(j) + C_j$, then τ_i will abort the processing of τ_j, inducing an abort cost of h. The maximum possible value of h is $t_{copy}(j) + C_j$, which is the highest abort cost that τ_i can induce on τ_j. □

IV. DETERMINING WORST CASE RESPONSE TIME

To determine the WCRT in a P-FRP n-task set, all possible release offset scenarios of higher priority tasks have to be analyzed for their response times. If we have to determine the WCRT of some task τ_j in Γ_n which is assumed to be released at time 0, this implies computing response time for every unique permutation where the release offsets of tasks τ_n to τ_{j+1} can range from 0 to T_j. As shown in Section 1, a total of $(T_j+1)^{n-j}$ cases have to be analyzed. If we bound the values of release offsets we only need to analyze the release offset permutations within the bounds and not those in the interval $[0, T_j]$. Even though the computationally complexity of this solution is still exponential to the size of the task set, the number of computations will be considerably less relative to analyzing all release offsets in the interval $[0, T_j]$. The following theorems establish these bounds.

Theorem 4.1: *Let $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$. The release offsets of tasks $\tau_{j+1} \ldots \tau_n$ which lead to the worst-case response time of τ_j, are guaranteed to be more than or equal to the worst-case abort costs that can be induced on τ_j.*

Proof. Let $\Phi_{j+1}, \ldots, \Phi_n$ be the release offsets of tasks that lead to the WCRT of τ_j, represented by $WCRT_j$. Let τ_{min} represent the task with the minimum release offset, represented by Φ_{min}. Or, $\Phi_{min} = minimum\{\Phi_{j+1}, \ldots, \Phi_n\}$.

From lemma 3.3, the maximum abort cost that can be induced on τ_j is: $t_{copy}(j) + C_j$. There can be two possible cases:

<u>Case 1</u>: $\Phi_{min} < t_{copy}(j) + C_j$

In this case, τ_{min} aborts the processing of τ_j. Let RT_j represent the response time of τ_j in this release scenario. If the release offset of tasks $\tau_{j+1} \ldots \tau_n$ is increased by an equal amount h: $\Phi_{min} + h \le (t_{copy}(j) + C_j)$, then the absolute response time of all jobs of tasks $\tau_{j+1} \ldots, \tau_n$ will increase equally by h. The response time of τ_j will also increase by at least h to: $RT_j + h$. Hence, the response of time of τ_j will increase with any increase in Φ_{min} by value h (provided, $\Phi_{min} + h \le t_{copy}(j) + C_j$).

However, if h is such that: $\Phi_{min} + h > t_{copy}(j) + C_j$, then τ_j will complete processing before any of the higher priority tasks are released, and the response time of τ_j will be P_j. Therefore, to increase the response time of τ_j, h cannot be more than $(t_{copy}(j) + C_j) - \Phi_{min}$. If $max\text{-}\Phi_{min}$ represents the maximum possible release offset at which τ_{min} can be released such as to increase the response time of τ_j, then:

$$\begin{aligned} max\text{-}\Phi_{min} &= \Phi_{min} + (t_{copy}(j) + C_j) - \Phi_{min} \\ &= t_{copy}(j) + C_j. \end{aligned}$$

Case 2: $\Phi_{min} > t_{copy}(j) + C_j$

In this case, τ_j will complete processing before any of the higher priority tasks are even released, making the response time of τ_j as P_j. This release scenario will not lead to the WCRT for τ_j.

Clearly, to induce the WCRT on τ_j the release offset of tasks $\tau_{j+1} \dots \tau_n$ will be greater than or equal to $t_{copy}(j) + C_j$. This is the **lower bound** of release offsets. □

Lemma 4.2: *Let $\Gamma_n = \{\tau_1, \tau_2, \dots, \tau_n\}$. The worst-case response time of task τ_j is guaranteed to occur after all tasks having higher priority than j have been released.*

Proof. Let τ_k represent a task having higher priority than τ_j. Or,

$$\tau_k \in \{\tau_{j+1}, \dots, \tau_n\}.$$

Let $WCRT_j$ represent the WCRT of τ_j after all tasks in $\{\{\tau_{j+1}, \dots, \tau_n\} - \tau_k\}$ have been released.

τ_j will start to restore its state at time $WCRT_j - t_{restore}(j)$. However, if τ_k is released at time $WCRT_j - h$, where $0 < h \leq : t_{copy}(j) + C_j$, then it will abort τ_j. The new WCRT of τ_j will be atleast:

$$WCRT_j - h + P_k + P_j.$$

This illustrates the fact that if all higher priority tasks have not been released, the response time of τ_j can be increased by releasing these higher priority tasks at times which force τ_j to abort. Hence, all tasks having a higher priority than j need to be released before the WCRT of τ_j can occur. □

Theorem 4.3: *For $\Gamma_n = \{\tau_1, \tau_2, \dots, \tau_n\}$, the release offset values of tasks $\tau_{j+1} \dots \tau_n$, which lead to the worst-case response time of τ_j, have an upper bound.*

Proof. Consider $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$. We have to derive the WCRT of τ_1. From lemma 4.2 we know that the release offset of all higher priority tasks have to be considered in this upper bound. From theorem 4.1, the release offset of tasks τ_2 and τ_3 will be more than or equal to $t_{copy}(1) + C_1$. Let the 1st job of τ_3 be released at time: $t_{copy}(1) + C_1$. This job will complete processing at time $t_{copy}(1) + C_1 + P_3$. Let τ_1 restart processing after τ_3 has completed and continue till time:

$$t_{copy}(1) + C_1 + P_3 + t_{copy}(1) + C_1.$$

Since, the P-FRP necessary schedulability test (lemmas 3.1, 3.2) is satisfied, $P_3 + P_1 \leq T_1$. Hence, another job of τ_3 will not be released till:

$$t_{copy}(1) + C_1 + P_3 + P_1.$$

If no other task is released then τ_1 will be able to complete processing at time $t_{copy}(1) + C_1 + P_3 + P_1$. To prevent this, the 1st job of τ_2 is released at time:

$$t_{copy}(1) + C_1 + P_3 + t_{copy}(1) + C_1.$$

In this case, the release offset upper bound is $t_{copy}(1) + C_1 + P_3 + t_{copy}(1) + C_1$. All tasks have to be released by this time to prevent τ_1 from completing processing.

Similarly, when $\Gamma_n = \{\tau_1, \tau_2, \dots, \tau_n\}$ there will exist a release offset upper bound by which all higher priority tasks are released to cause delay in the response time of a lower priority task. □

A. Release Offset Upper Bound

Theorem 4.3 proves that an upper bound on release offset exists. In this section, we present a method to derive this upper bound. An intuitive way to compute this bound is to release the highest priority task first, followed by other tasks in priority order, at intervals which induce maximum abort cost on the lower priority task τ_j. However, this is not guaranteed to be the true upper bound. Consider a task set $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$. If, $P_2 > P_3$ then:

$$t_{copy}(1) + C_1 + P_2 + t_{copy}(1) + C_1 > t_{copy}(1) + C_1 + P_3 + t_{copy}(1) + C_1$$

in which case the upper bound will be reached when τ_2 is released first at time: $t_{copy}(1) + C_1$, followed by τ_3 at time $t_{copy}(1) + C_1 + P_2 + t_{copy}(1) + C_1$. The release offset upper bound will be will be atleast:

$$t_{copy}(1) + C_1 + P_2 + t_{copy}(1) + C_1.$$

However, if $P_2 < P_3$ the release offset upper bound will be atleast $t_{copy}(1) + C_1 + P_3 + t_{copy}(1) + C_1$.

If there are more than 2 higher priority tasks the computation for the upper bound gets even more complex. Consider finding the release offset upper bound when $\Gamma_n = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. As before, lets release the 1st job of τ_4 at time $t_{copy}(1) + C_1$ followed by the 1st job of τ_3 at time:

$$t_{copy}(1) + C_1 + P_4 + t_{copy}(1) + C_1.$$

1st job of τ_2 will be released at:

$$t_{copy}(1) + C_1 + P_4 + t_{copy}(1) + C_1 + P_3 + t_{copy}(1) + C_1 \quad \dots(4.1)$$

However, even if the P-FRP necessary schedulability test is satisfied it is possible for the 1st job of τ_3 to be aborted by the 2nd job of τ_4 at:

$$t_{copy}(1) + C_1 + P_4 + t_{copy}(1) + C_1 + h \; (0 < h \leq t_{copy}(3) + C_3).$$

In this case τ_3 will resume processing at:

$$t_{copy}(1)+C_1+P_4+t_{copy}(1)+C_1+h+P_4$$

and complete at:

$$t_{copy}(1)+C_1+P_4+t_{copy}(1)+C_1+h+P_4+P_3$$

τ_2 can be released after allowing τ_1 to be processed again for its maximum abort time. Hence, τ_2 will be released at:

$$t_{copy}(1)+C_1+P_4+t_{copy}(1)+C_1+h+P_4+P_3+t_{copy}(1)+C_1 \ldots (4.2)$$

Clearly, expression (4.2) > expression (4.1). Hence, the abort of the 1st job of τ_3 by the 2nd job of τ_4, allows us to delay the release of τ_2, by $h + P_4$ time.

The abort of tasks having higher priority than τ_1 by even higher priority tasks complicates the computation of the upper bound on release offset. For example, if τ_3 is released first followed by τ_4, it could be possible that the 2nd job of τ_3 induces an even higher abort cost on the 1st job of τ_4, further delaying the release of task τ_2.

Unfortunately, knowledge on the *release order* of higher priority tasks which lead to an upper bound on release offset cannot be ascertained by the knowledge of arrival periods and processing times alone. We have to evaluate all possible permutations of release orders and use the maximum value among these as the upper bound on release offsets. If the WCRT of τ_j has to be derived, we will have to evaluate ${}^{n-j}P_{n-j} = (n-j)!$ release orderings. For each release order, we add the maximum abort cost $t_{copy}(j) + C_j$, between releases of the 1st jobs of all higher priority tasks. If the 2nd job of some higher priority task is released in this period, we accordingly increment the upper bound.

Pseudo-code for the mechanism of computing the release offset upper bound is given in algorithm 4.1. By processing one time tick at a time, this algorithm computes the upper bound under every release order permutation. After the $(n-j)$Q permutations of release orders are generated (line 5), the algorithm executes a loop (lines 6-27) for each release order. The release offset of each task at the kth index of the order is computed in a loop (lines 9-24). Starting with a time tick of 0, the algorithm simulates the execution of τ_j or higher priority tasks for every time tick till the upper bound is found. The function 'PutTasksinQueue(*tick*)' (line 12) is a function that inserts higher priority tasks, which have been previously released, in the processing queue. If the queue in empty, implying that no previously released higher priority tasks is available for processing, τ_j will be processed (line 19), and if τ_j has been processed for max_{abort} time the release offset of higher priority task τ_k, which will abort τ_j, is assigned a value (line 21). If some higher priority tasks is released before τ_j has been processed for max_{abort} time, the number of ticks τ_j has been processed for is reset to 0 (line 14), signifying that it has to be processed again for max_{abort} time before a higher priority task can be released. The offset bound for each release order is compared against the current value of $offset_{ub}$ (line 25), to insure that it has the maximum value among all the permutations. It might be possible that some tasks could never be released due to high arrival rate of some higher priority task. To prevent the loop from going infinitely, we abort the computation, if $offset_{ub}$ exceeds $T_j - P_j$ (line 26) since in this case, τ_j is unschedubale.

Algorithm 4.1

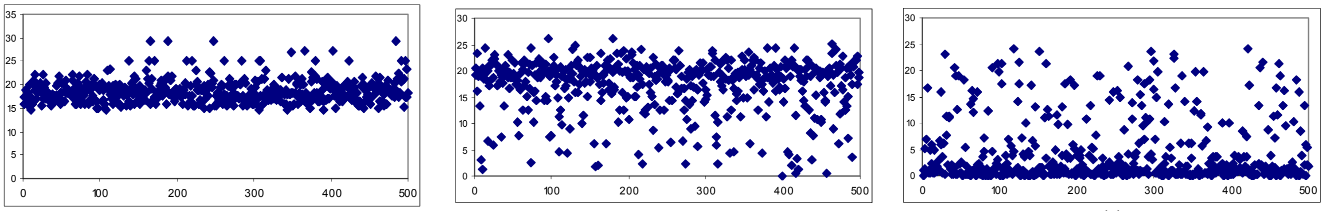
```
1.  input: $\Gamma_n$, $j$
2.  output: $offset_{ub}$, upper bound on release offset
3.  $offset_{ub} \leftarrow 0$
4.  $max_{abort} \leftarrow C_j + t_{copy}(j)$
5.  Compute $(n-j)!$ permutation of release orders
6.  for each $(n-j)!$ possible release orders
7.    $\forall(k=n-j,\ldots,n)$, set $\Phi_k$ to 0
8.    $k \leftarrow 1$ : $tick \leftarrow 0$
9.    for each task $\tau_k$ in $k^{th}$ index of the permutation
10.       loop while $(\Phi_k=0)$
11.          $tick \leftarrow tick + 1$
12.          PutTasksInQueue($tick$)
13.          if Queue Not Empty
14.            $time_j \leftarrow 0$
15.            $\tau_i \leftarrow$ highest priority task in Queue
16.            if($\tau_i$ has been processed for $P_i$ time)
17.                    remove $\tau_i$ from Queue
18.          else
19.            $time_j \leftarrow time_j + 1$
20.            if($time_j = max_{abort}$)
21.                $\Phi_k \leftarrow tick$
22.          end if
23.        end loop
24.     end for
25.     if $(tick > offset_{ub})$ $offset_{ub} \leftarrow tick$
26.     if$(tick > T_j - P_j)$ $offset_{ub} \leftarrow T_j$
27. end for
28. return $offset_{ub}$
```

Time Complexity: In the worst-case, j=1 and all release order permutations will result in the maximum upper bound of T_1. The loop inside lines 10-23 will run for T_1 time and the loop in lines 9-24 will run $(n-1)$ times. The outer loop in lines 6-27 will run for $(n-1)!$ times. Hence, the worst-case complexity of this algorithm is bounded by:

$$O((n-1)! \cdot (n-1) \cdot T_1).$$

B. Computing the WCRT

To determine the WCRT of τ_j, all possible combinations of release offset of higher priority tasks in the range [$offset_{lb}$, $offset_{ub}$] will be enumerated and analyzed for their response time. The total number of cases that will be analyzed this way is:



(a) (b) (c)

Figure 2. % $(offset_{ub}–offset_{lb}+1)^{n-1}$ / $(WCRT_1+1)^{n-1}$ (a) 3-task sets (b) 4-task sets (c) 5-task sets. Each discrete value in the X-axis represents a task set

$(offset_{ub} – offset_{lb}+ 1)^{n-j}$.

where, $offset_{ub}$ and $offset_{lb}$ represent the upper and lower bounds on release offsets, respectively. Using this method, the number of cases whose response time is computed will be less than when the response time is computed for all cases in the interval $[0, T_j]$. This method does require an additional computation for the upper bound, however the overhead of this upper bound is very low relative to the overall computation of WCRT. Note that if $offset_{ub} > T_j$, there will be a release scenario in which τ_j will be unschedulable and hence, evaluation of all release permutations is not required leading to additional computational savings.

Example: Consider the task set specified in Section 2.2.2. In this example,

$offset_{lb} = 4 – 1 = 3$ (from theorem 4.1).

the maximum release offset of τ_3 when τ_2 is released first : 9
the maximum release offset of τ_2 when τ_3 is released first : 9
hence, $offset_{ub} = 9$.

We will evaluate the response time of all possible release offset permutations in the period [3,9]. After this evaluation is performed, the WCRT of τ_1 is computed as 39 and occurs when τ_2, τ_3 are released at times 3 and 5 respectively. These release offsets are clearly within the derived bounds. The number of cases that were evaluated is:

$(offset_{ub} – offset_{lb}+ 1)^{n-j}$
$= (9 – 3 + 1)^2 = 49$.

If we evaluate all release offset permutation in the interval $[0, T_j)$ the number of cases that need to be evaluated are:

$(T_j+1)^2 = (46)^2 = 2116$.

Number of cases evaluated by bounding the offsets is only ≈ 2.3% of the cases evaluated in $[0, T_j)$.

V. EXPERIMENTAL ANALYSIS

To ascertain the computational savings, we present an experimental analysis of our method. 3 groups, with 500 task sets, each having 3,4, and 5 tasks were randomly generated. Each task set in a group is unique in the sense that atleast 1 task is different between any two task sets present in a group. Every task set satisfied the P-FRP necessary schedulability test. The arrival period for each of the tasks in all the 3 groups were selected from the range [40,60], while the processing times were selected from [4,10]. These ranges were arbitrarily selected and kept small to reduce the time for our analysis. For each task set we determined the WCRT of the lower priority task (τ_1) by enumerating release offset within the bounded range as well as in the range $[0, T_1)$. If a task set was found unschedulable in its worst-case, it was discarded and a new one generated in its place. The response time for a task under a given release scenario is computed using the method given in [3] (This method is more efficient than a time accurate simulation).

Figs. 2(a), *(b)* and *(c)* show the results for 500 tasks sets which have 3,4 and 5-task sets respectively. Analyzing all release combination of higher priority tasks in the range $[0, T_1)$, and analyzing those in the range $[offset_{lb}, offset_{ub}]$ yields the same value of WCRT, which proves the correctness of our approach. Though we have analyzed $(T_1+1)^{n-1}$ permutations for the range $[0, T_1)$, this number is dependent on the arrival period of τ_1 which can be significantly higher than the WCRT for some task sets. Higher values of T_1 can show biased results in favor of our method.

To remove this bias, we compare against the number of release offsets enumerated till the WCRT of τ_1. Since the task set is schedulable, T_1 has to be greater than or equal to the WCRT of τ_1 ($WCRT_1$). Hence at the minimum, all $(WCRT_j)^{n-1}$ release permutations in the range $[0, WCRT_1)$ have to be analyzed for their response time. *Fig. 2(a)* shows the value of $(offset_{ub} – offset_{lb} + 1)^{n-1}$ as a % of $(WCRT_j)^{n-1}$ for 3-task sets. For all the 500 task sets with 3 tasks each, the number of cases evaluated by bounding the offsets were in the range [64,324], versus [361, 1936] for $(WCRT_j)^{n-1}$ cases. Percentage wise, cases evaluated with bounded offsets are only 15% - 30% of the cases evaluated with offsets in the range $[0, WCRT_1)$.

Figs. 2(b) and *2(c)* show the same set of results for task sets with 4 tasks and 5 tasks. For task sets with 4 tasks, the number of cases evaluated by bounding the offsets are in the range [4096, 42875], versus [19683, 40707584] for $(WCRT_j)^{n-1}$ cases, while for task sets with 5 tasks the same numbers are [279841, 59969536] and [1679616, 149000000000]. Percentage wise, cases evaluated when offsets are bounded are 0.1% - 26% (for 4 tasks) and 0.001% - 24% (for 5 tasks). The results clearly show that the computation of $WCRT_j$ using exhaustive enumeration within offset bounds, requires considerably less number of computations than enumerations within the range $[0,T_j)$.

VI. RELATED WORK

Related work includes response time analysis for P-FRP as well as for the preemptive and other abort-restart execution models. In [15], simple response time analysis has been done for P-FRP by assuming very low execution times of tasks. Ras and Cheng [21] present an algorithm that approximates an upper bound on WCRT by extending Audsley's algorithm [2] for preemptive systems. In ([3],[4]) Belwal and Cheng present an efficient method for determining the exact response time for a given release scenario of higher priority tasks in P-FRP.

Response time analysis for the preemptive model was first studied by Joseph and Pandya [14], and fixed priority scheduling was independently studied by Audsley et al [2]. Transactional memory systems have been described by Herlihy and Moss [12]. Response time analysis for transaction memory using dynamic scheduling for multiprocessor systems has been done by Fahmy et al [10]. In Manson et al [17], an atomic execution of the critical section has been implemented in Java, and response time analysis using fixed priority scheduling has been presented. However, the response time so derived does not account for complete re-execution of critical sections. Davis and Burns [8] derive upper bounds on response time for fixed priority scheduling. Jeffay et al [13] have taken a detailed look at the problem of scheduling periodic or sporadic tasks without preemption and inserted idle time.

Anderson et al [1] do response analysis of the lock-free mechanism. Lock-free is a mechanism to access shared resources among tasks such that none of them have to block and wait for the resource to become available. It is a mechanism to avoid priority inversion [22]. Comparisons between transaction memory based systems and lock-free execution and benefits of the former have been shown in Herlihy and Moss [12].

VII. CONCLUSIONS AND FUTURE WORK

In P-FRP, due to abort of preempted tasks there is no single critical instance of release that can be used to compute the WCRT. Till now, to compute the exact WCRT of a lower priority task $\tau_j \in \Gamma_n$, all release offset combinations of tasks τ_{j+1} to τ_n in the period $[0,T_j)$ had to be enumerated and analyzed for their response time. Due to the large number of cases that have to be evaluated, this method is computationally expensive and impractical. In this paper, we have presented an efficient technique for determining WCRT of P-FRP tasks using enumeration of release scenarios within a small bounded range. Techniques to compute the bounded range have been presented. Experimental results show that this approach considerably reduces the total number of release scenarios that need to be evaluated for their response time.

However, the method of determining WCRT by enumerating release scenarios is still a 'brute-force' approach, and the number of computations is an exponential power of task set size. Therefore, the number of cases to be evaluated can still be extremely high for larger task sets. Unfortunately at this time, such a method remains the only viable approach for determining exact value of WCRT in P-FRP. Till a polynomial time method is developed, our technique remains the most efficient way to compute WCRT in P-FRP.

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, A. Burns, M. Richardson, K. Tindell, A. Wellings. "Applying new scheduling theory to static priority preemptive scheduling". *Software Engineering Journal 8(5), pp. 284-292*, 1993

[3] C. Belwal, A.M.K. Cheng. "Determining Actual Response Time in P-FRP". *Practical Aspects of Declarative Languages (PADL)'11,*2011

[4] C. Belwal, A.M.K. Cheng. "Determining Actual Response Time in P-FRP using idle-period game board". *IEEE ISORC'11,*2011

[5] C. Belwal, A.M.K. Cheng, "On Priority Assignment in P-FRP". *RTAS'2010 Work-in-Progress session*, 2010

[6] G. Bernat. "Response Time Analysis of Asynchronous Real-Time Systems". *Real-Time Syst. 25, pp. 131-156*, 2003

[7] A.M.K. Cheng, Jim Ras, ``Response Time Analysis of the Abort-and-Restart Model under Symmetric Multiprocessing''. *ICESS-'10*, 2010

[8] R.I. Davis, A.Burns. "Response Time Upper Bounds for Fixed Priority Real-Time Systems". *RTSS'08*, *pp.407-418*, 2008

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

[10] 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

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

[12] 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

[13] K. Jeffay, D. F. Stanat, C. U. Martel. "On Non-preemptive scheduling of Periodic and Sporadic tasks". *IEEE Symposium on Real-time Systems*, *pp.129-139* , 1991

[14] M. Joseph, P. Pandya. "Finding Response Times in a Real-Time System". *BCS Computer Journal (Vol. 29, No. 5), pp. 390-395*, 1986

[15] R. Kaiabachev, W. Taha, A. Zhu. "E-FRP with Priorities". *EMSOFT'07 , pp. 221-230* , 2007

[16] 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

[17] 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

[18] J. Peterson, G. D. Hager, P. Hudak. "A Language for Declarative Robotic Programming". *ICRA'99, IEEE*, 1999

[19] J. Peterson, P.Hudak, A.Reid, G. D. Hager. "FVision: A Declarative Language for Visual Tracking". In *PADL'01,* 2001

[20] M. F. Ringenburg, D. Grossman. "AtomCaml: first-class atomicity via rollback". *ACM ICFP'05*, *pp. 92-104*, 2005

[21] J. Ras, A.M.K. Cheng, "Response Time Analysis for the Abort-and-Restart Task Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm", *RTCSA'09*, 2009

[22] 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

[23] Z. Wan, W. Taha, P. Hudak. "Real - time FRP". *ICFP'01*, *pp. 146-156*, ACM Press ,2001

[24] Z. Wan, W. Taha, P. Hudak. "Task driven FRP". *PADL'02*, 2002

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