Response Time Bounds for Event Handlers in the Priority based Functional Reactive Programming (P-FRP) Paradigm

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

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

Yuanfeng Wen  
Department of Computer Science  
University of Houston, TX, USA  
wyf@cs.uh.edu

ABSTRACT

Priority-based Functional Reactive Programming (FRP) is a new declarative approach to modeling and building reactive systems. Preempted tasks in P-FRP are aborted and have to restart when higher priority tasks have completed. Unlike the preemptive model of execution, there is no notion of critical instance in P-FRP. Determination of the actual value of Worst-Case Response Time (WCRT) in P-FRP requires evaluation of an exponential number of release scenarios. Such a ‘brute-force’ computation is expensive and impractical in several situations. In previous work, researchers have presented a polynomial time algorithm which uses fixed point iteration to compute an approximate upper bound on response time. However, this algorithm fails to converge for several task sets and hence is quite limited in use. In this paper, we present a more robust algorithm that computes response time bounds of P-FRP tasks and guarantees a result for all task sets. Experimental results using synthetic task sets of different sizes have also been presented.

Categories and Subject Descriptors

D.3.2 [Programming Languages]: Language Classifications – Applicative (functional) languages

General Terms

Algorithms, Languages

Keywords

P-FRP, WCRT, Upper bound on response time

1. INTRODUCTION

Functional Reactive Programming (FRP) [24] 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 [16], and vision [17]. 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 ([13],[22],[23]). It was shown that a variant called priority-based FRP (P-FRP) [13] combines both the semantic properties for 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 affecting the semantic soundness of the program. This way a clear separation between the semantics of the program and the responsiveness of each handler is achieved.

Ras and Cheng [19] presented the first polynomial time algorithm for computing approximate response time bounds in P-FRP. This algorithm extends the fixed point iteration algorithm developed by Audsley et al [2] by adding additional costs incurred due to abort of tasks. However, as we explain in more detail in Section 3, Ras and Cheng’s algorithm fails to converge for several task sets and therefore is an unreliable method to get guaranteed results.

In this paper, we present a new and improved algorithm for determining response time bounds for P-FRP tasks. Our algorithm runs in polynomial time and guarantees a response time bound for every task set. After presenting the notations and execution model (Section 2) of P-FRP, we analyze the limitations of Ras and Cheng’s algorithm (Section 3). We then define a lower bound on response time (Section 4) based on which we define a new algorithm to compute the...
response time upper bound (Section 5). Experimental results using our algorithm are presented (Section 6), followed by a review of related work (Section 7) and a reflection (Section 8) on our results.

2. 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.

2.1 Basic Concepts

The notation and formal definitions for important concepts 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. $\Gamma_n$ 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 $\tau_k$ and $r_k = 1 / T_k$ is the arrival rate of $\tau_k$.

• $C_k$ is the worst-case execution time for $\tau_k$.

• $R_{k,m}$ represents the release time of the $m$th job of $\tau_k$.

• $\Phi_k$ represents the release offset which is the release time of the first job of $\tau_k$. Or, $\Phi_k = R_{k,1}$. Hence, $R_{k,m} = \Phi_k + (m-1) \cdot T_k$.

• $D_k$ is the relative deadline of $\tau_k$. If some job of $\tau_k$ is released at absolute time $R_{k,m}$ then $\tau_k$ should complete processing by absolute time $R_{k,m} + D_k$, otherwise $\tau_k$ 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 execution time to arrival periods of every task. Hence, $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$.

• Interference on $\tau_i$ is the action where the execution of $\tau_i$ is interrupted by the release of a higher priority task.

• The amount of time spent in the aborted execution of a task is the abort cost.

2.2 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. Due to P-FRP’s transactional nature of execution, all tasks are assumed to run without concurrency or precedence constraints. The execution times, priority assignment and arrival periods of all tasks are assumed to be known a priori. All time attributes are composed of discrete values.

When some task is released, it enters an execution 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 execution 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. Once $\tau_i$ enters the execution queue two situations are possible. If a task of lower priority than $i$ is executing, it will be immediately preempted and $\tau_i$ will start execution. If a task of higher priority than $i$ is executing, then $\tau_i$ will wait in the queue and start execution only after the higher priority task has completed. When a task starts execution it creates a temporary copy of the state (represented by variables in register/memory) and upon completion it overwrites the original state with the temporary copy in an atomic operation. During these state copy operations, all system interrupts are disabled and no task is allowed to enter the execution queue.

3. PREVIOUS WORK

An algorithm to compute response time upper bounds for P-FRP tasks has been presented by Ras and Cheng [19]. This algorithm builds upon the fixed point iterative algorithm developed by Audsley et al [2]. The original algorithm of Audsley only accounted for interference costs, and Ras and Cheng extended it by accounting for abort costs. In [19], the WCRT for a task $\tau_i$ is defined as:

$$WCRT_i = C_i + B_{int} + I_i + \alpha_i.$$  

$B_{int}$ is a constant representing the worst-case blocking time, $I_i$ is the maximum interference cost from higher-priority tasks that a task can experience in the interval $[t, t + WCRT_i)$; $t$ is the release time of $\tau_i$ and $\alpha_i$ is its abort cost. The WCRT will be the sum of all these times, since there will be no idle period till $\tau_i$ has computed (any idle period will be taken up by aborted execution(s) of $\tau_i$).

Applying the values of $I_i$ and $\alpha_i$ as given in the paper, the complete equation of $WCRT_i$ is given as:

$$WCRT_i = P_i + B_{int} + \sum_{j \in hp_i} \left( \frac{WCRT_j}{T_j} \right) \cdot C_j$$

$$\quad \quad \quad + \sum_{j \in hp_i} \left( \frac{WCRT_j}{T_j} \right) \cdot \max_{k=i} C_k \quad \ldots(3.1)$$

$hp_i$ represents the set of tasks having higher priority than $i$. Since $WCRT_i$ appears on both sides of the equation, an iterative method first presented by Audsley et al [2] is used to compute the value of $WCRT_i$. This method uses an initial value of WCRT and iteratively converges to a solution. If $WCRT_i^n$ represents the $n^{th}$ approximate value of $WCRT_i$, and setting the blocking time to 0, eq. (3.1) can be written as:
The exact value of WCRT has been derived by evaluating all possible release scenarios of higher priority tasks. The same technique is also used for computing exact WCRT for use in experiment analysis.

4. LOWER BOUND ON RESPONSE TIME

In the preemptive model of execution, preemption induces an interference cost which is same as the execution time of the higher priority task. Audsley et al account for this interference cost in their fixed point iterative equation:

\[
WCRT_i^{n+1} = C_i + \sum_{j \in h p_i} \left( \frac{WCRT_j^n}{T_j} \right) \cdot C_j \ldots (4.1)
\]

Eq. (4.1) determines the WCRT of a task \( \tau_i \), and if \( \tau_i \) is schedulable it is guaranteed to converge to a solution as shown in [2].

In P-FRP, preemption also induces an abort cost. The minimum cost induced due to aborts is stated by the following lemma.

**Lemma 1:** For \( \Gamma_2 = \{ \tau_i, \tau_j \} \): \( i > j \), the minimum abort cost induced by \( \tau_i \) on \( \tau_j \) is 1.

**Proof.** Let \( \tau_j \) be released at time \( t \) and execute for \( h \) time units, after which a job of higher priority task \( \tau_i \) is released. If \( \tau_i \) has not completed execution, \( \tau_j \) will abort \( \tau_i \). Hence \( \tau_j \) will be aborted by any \( h < C_j \). Since we assume discrete values of time the minimum value of \( h = 1 \). This is the minimum abort cost that \( \tau_i \) can induce on \( \tau_j \).

Every preemption in P-FRP is guaranteed to increase the response time of a task \( \tau_i \) by adding both interference and abort cost. While the interference cost is always equal to the execution time of the higher priority task the abort cost can vary. If we assume minimum abort costs during each preemption, we will get the minimum value of the response time of \( \tau_i \).

The minimum abort costs can easily be accommodated in eq. (4.1) by adding 1 to the execution time of higher priority tasks. This will give us the minimum response time (also referred to as the lower bound) of \( \tau_i \). Or the minimum value of response time of \( \tau_i \) is expressed by:

\[
LBRT_i^{n+1} = C_i + \sum_{j \in h p_i} \left( \frac{LBRT_j^n}{T_j} \right) \cdot (C_j + 1) \ldots (4.2)
\]

Eq. 4.2 derives the lower bound on response time on \( \tau_i \) (represented by \( LBRT_i \)). \( LBRT_i^0 \) is set to \( C_i \). Like eq. (4.1), eq. (4.2) is guaranteed to converge if \( \tau_i \) is schedulable with minimum abort costs. If \( \tau_i \) is unschedulable with minimum abort costs, it is clearly unschedulable in its worst-case. This
iterative method converges to a solution in polynomial time as proven in [2].

5. UPPER BOUND ON RESPONSE TIME

In this section, we present an algorithm to compute the response time upper bound of P-FRP tasks. This algorithm uses the lower bound on response time computed in the previous section. It also requires an upper bound on abort costs, which is derived in the following lemma.

Lemma 2: For \( \Gamma_2 = \{ \tau_i, \tau_j \} : i > j \), the maximum abort cost induced by \( \tau_i \) on \( \tau_j \) is: \( C_i - 1 \).

Proof. Let \( \tau_j \) be released at time \( t \) and execute for \( h \) time units, after which a job of higher priority task \( \tau_i \) is released. \( \tau_j \) will abort \( \tau_i \) only if \( \tau_i \) has not completed execution. Hence, if \( h \leq C_i - 1 \), then \( \tau_j \) will abort the processing of \( \tau_i \), inducing an abort cost of \( h \). The maximum possible value of \( h \) is \( C_i - 1 \), which is the highest abort cost that \( \tau_i \) can induce on \( \tau_j \). □

A guaranteed upper bound on response time (UBRT) should never be lower than the actual WCRT of a task. Hence, to derive such a guaranteed upper bound we have to make pessimistic assumptions on the execution time of lower priority tasks. Such pessimistic assumptions will not be valid for every task set, and might lead to a UBRT value which is significantly higher than the WCRT. However, this is a trade-off that has to be made for the guarantee provided by the upper bound.

For deriving the upper bound, an important consideration is the release offset that needs to be assigned to higher priority tasks. Since the computation has to be inexpensive, evaluating all possible release scenarios is not an option. Hence, we assume that all tasks are released at the same time. As mentioned before, such a release condition is not guaranteed to lead to the WCRT of a lower priority task. However, we use this release scenario as a ‘base’ upon which pessimistic abort costs on the lower priority tasks are added. The pessimism in the abort costs will give us a value which will be more than or equal to the actual WCRT.

As shown in lemma 2, the maximum abort cost that a higher priority task \( \tau_i \) can induce on \( \tau_j \) is:

\[ C_i - 1 \]

The maximum preemption cost induced on a lower priority task \( \tau_j \) by higher priority task \( \tau_i \) is defined as the sum of interference as well as maximum abort cost:

\[ C_i + C_j - 1 \]

For computing the response time upper bound of \( \tau_j \), we make the pessimistic assumption that each job of tasks \( \tau_{j+1} \) through \( \tau_{n} \) will add maximum abort cost to the response time of \( \tau_j \). If \( \tau_j, \tau_i \in \{ \tau_{j+1}, \ldots, \tau_n \} \), then \( \tau_i \) can also preempt \( \tau_j \). For every job of a higher priority we compute the worst-case preemption costs it can cause among all the lower priority tasks. Or, if \( \tau_k \) preempts \( \tau_j \), then the worst-case preemption cost that is caused by the release of \( \tau_k \) is given by:

\[ k-1 \]

\[ C_k + ( \max_{m=1}^{k-1} C_m - 1 ) \]

\( ( \max_{m=1}^{k-1} C_m - 1 ) \) gives the maximum abort costs that can be induced among all tasks having lower priority than \( \tau_i \). The preemption costs induced on \( \tau_j \) also depends on the number of jobs of tasks \( \tau_i \) where \( i = j+1, \ldots, n \).

We compute the number of jobs of higher priority tasks based on a dynamic time interval. The initial value of this time interval is the lower bound on the response time of \( \tau_j \). The size of this interval is increased in an iterative fashion by adding pessimistic abort costs induced on \( \tau_j \) by jobs of higher priority tasks. Costs from additional jobs that were released in the increased time interval are also added in the next iteration. Since we increase the time interval by adding pessimistic abort costs to it, the value of the time interval and the number of jobs of higher priority tasks that are released in this interval, are also pessimistic.

The pseudo-code of the algorithm (referred to as UBRT-algorithm) to compute the upper bound on response time of \( \tau_j \) is given in Fig 2. We first compute the lower bound on response time of \( \tau_j \) (LBRT) and initialize UBRT to UBRT (line 2). The preemption cost from task \( \tau_{j+1} \) through \( \tau_n \) (represented by \( \tau_i \)) are computed inside the loop between lines 3-17. The number of the jobs of \( \tau_i \) in the interval \( [0, UBRT_j] \) (line 12) are calculated, and the pessimistic preemption cost induced on \( \tau_j \) by each job of \( \tau_i \) (lines 14, 15) is derived. This pessimistic cost is added to UBRT, after which we start computing the abort costs from the next higher priority task in the updated time interval \( [0, UBRT_j] \). We also add the preemption costs from new jobs of tasks \( \tau_{j+1} \) to \( \tau_{n-1} \), which

```
1: input: \( \Gamma, LBRT \)
2: UBRT \( \rightarrow LBRT \)
3: for \( a \rightarrow j+1 \) to \( n \)
4: \( Cost_{new_job} = 0 \)
5: for \( b \rightarrow j+1 \) to \( a-1 \)
6: \( jobs_b \leftarrow \frac{(UBRT_j) - (old - UBRT_j)}{T_p} \)
7: \( Cost_{new_jobs} \leftarrow Cost_{new_jobs} + jobs_b \times C_b \)
8: \( Cost_{new_jobs} \leftarrow Cost_{new_jobs} + jobs_b \times (\max_{k=1}^{C_k} - 1) \)
9: end for
10: UBRT \( \rightarrow UBRT_j \times Cost_{new_jobs} \)
11: if \( a < n+1 \)
12: \( UBRT_j \leftarrow UBRT_j + jobs_a \times C_j \)
13: \( old - UBRT_j \leftarrow UBRT_j \)
14: UBRT \( \rightarrow UBRT_j + jobs_a \times (\max_{k=1}^{C_k} - 1) \)
15: end if
16: end for
17: end for
18: return UBRT;
```

Figure 2. Pseudo-code of the UBRT-algorithm for computing upper bound on Response Time of a task \( \tau_j \) (UBRT)
are released in the additional time interval \([\text{old-UBRT}, \text{UBRT}]\) in the loop between lines 5-9. This additional cost \((\text{Cost}_{\text{newJobs}})\) is added to \(\text{UBRT}_j\) (lines 7, 8). Once the abort costs from the last higher priority task \(\tau_i\) is derived, the iterative loop exits and the final value of \(\text{UBRT}_j\) is returned.

### 5.1 Time Complexity

There are two main iterations in the UBRT-algorithm. The loop between lines 3-17 will run for \(n-j\) steps. The number of steps taken by loop between lines 5-9 will be \(n-j-1\), and this loop will be executed \(n-j\) times. The iteration between lines 5-9 executes for the maximum number of steps, hence the time complexity of this algorithm can be determined using the number of steps executed by this loop. The minimum value of \(j\) is 1 hence, the maximum number of steps for which loop between lines 5-9 can be executed is \((n-1)\cdot(n-2)\) which is asymptotically bounded by \(O(n^2)\). Hence, the time complexity of the UBRT-algorithm algorithm is bounded by \(O(n^2)\).

### 5.2 Termination Guarantee

The UBRT-Algorithm requires computation of LBRT which is guaranteed to converge to a solution if the task set is schedulable. If the task is not schedulable, a program to compute the LBRT can be easily terminated if LBRT exceeds the deadline of the task (in which case the task set is unschedulable). In the UBRT-algorithm, there is no equivalency condition like in eq. (3.1), required for terminating the computation. The algorithm executes two loops which are bounded by the number of tasks, hence irrespective of the value of \(\text{UBRT}\) the algorithm is guaranteed to terminate.

### 6. EXPERIMENTAL ANALYSIS

In the abort-restart execution paradigm of P-FRP, it is difficult to derive approximation bounds on P-FRP using formal analysis. Therefore, we have analyzed the approximation factors using experimental task sets of various sizes and different total utilization factors.

3 groups with 500 task sets in each group were randomly generated. Task sets in each group have 3, 4 and 5 tasks, and each task set in a group is unique in the sense that at least 1 task is different between any two task sets present in the group. Half the task sets in each group have total utilization factor \((U) \leq 0.5\) (lower utilization category) while the other group had \(U > 0.5\) (higher utilization category). The arrival period for each of the tasks in all the 3 groups were selected from \([40, 60]\), while the processing times were selected from \([4, 10]\). The bounds for these ranges were selected arbitrarily and were kept small to reduce the time for conducting the analysis. For each task set we determined the actual WCRT and the response time bound (UBRT) using the UBRT-Algorithm, for the lowest priority task \((\tau_1)\). Then the approximation factor for each task set was derived using the following equation:

\[
\text{Approximation factor} = \frac{\text{actual WCRT}}{\text{UBRT}}.
\]

Figs. 3(a), (b) and (c) show the approximation factors for 250 task sets in the lower utilization category for 3, 4 and 5-task sets respectively. The average approximation factor for 3, 4 and 5-task sets is \(\approx 1.9, \approx 2.7\) and \(\approx 3.5\) respectively. The minimum and maximum factors in the task sets are 1.8, 2.4, 2.9 and 2, 3.4, 3.7 respectively. Figs. 4(a), (b) and (c) shows the same results for 250 tasks in the higher utilization category. The average approximation factor for 3, 4 and 5-task sets is \(\approx 2.0, \approx 2.8\) and \(\approx 3.6\) respectively. The minimum and maximum factors in the task set are 1.6, 1.4, 2.2 and 2.2, 3.4, 4.8 respectively.

There is a noticeable tendency that the approximation factor increase with the size of the task set. This is attributed to the fact that preemption costs are proportional to the number of higher priority tasks. Since we have assumed pessimistic preemption costs, the difference between actual WCRT and one approximated by our algorithm increases for several task sets. However for some task sets especially those with higher utilization, the approximation factor is low. With higher utilization, the interference between tasks generally increases and the preemption costs are much closer to the pessimism considered in our algorithm. If the pessimism is reduced it can lead to lower approximation factors for several task sets. However, for some task sets it is also possible for the approximated value to be lower than the WCRT, in which case the algorithm fails to provide a guarantee for the computed upper bound. Hence, a high approximation factor is a trade-off that has to be made for the guarantee of upper bound provided by our algorithm.

The results highlight the challenges in deriving a single algorithm for computing tighter response time bounds for all P-FRP task sets. Based on information available \textit{a priori}, it is not possible to ascertain the exact preemption cost, and the pessimism assumed in the analysis gives approximation factors that vary significantly among different task sets.

Byun et al [3] adopt the critical section approach for CPU tasks, into a Database model. The authors have bounded the response time for a task and have considered the cost of re-execution. Since, in P-FRP a low priority task is always aborted we have to assume that every task shares a lock with each other (thereby making \(\text{xlock}(i)\) true for every high priority task). However a situation where a transaction \(m\) can abort any of the transactions in the range \([m, i+1]: m < n\) (\(m, i+1\) have higher priority than transaction \(i\), with \(n\) being the highest priority transaction), is not accounted for in this equation. Shu [20] has characterized the re-execution costs for real-time abort-oriented protocols, which are an approach where low priority transactions are aborted in favor of high priority ones, only if there is excessive blocking of higher priority transactions.
7. CONCLUSIONS

In this paper, we have introduced a polynomial time algorithm that computes an approximate upper bound on response time. As seen in experimental analysis, the quality of the bound given by this algorithm varies significantly, and it can be quite tight or loose depending on the characteristics of the task set. For task sets with loose bounds our algorithm overestimates the costs due to preemption and aborts. The exact costs of interference and aborts in P-FRP is difficult to predict based on available task parameters, and loose bounds for several task sets highlights this problem. We can reduce the pessimism in preemption costs, however doing so will make the response time computed by our algorithm lower than the WCRT for certain task sets.

Unless techniques are developed that can characterize the execution of P-FRP task sets, through which we can get a more accurate idea on preemptions, it may not be possible to derive an algorithm that gives tight bounds for all P-FRP task sets. Development of such techniques remains an open area of research in real-time systems.

8. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Award Nos. 0720856 and 1219082.

9. REFERENCES