Improved Priority Assignment for the Abort-and-Restart (AR) Model

H.C. Wong and A. Burns
Department of Computer Science,
University of York, UK.

February 1, 2013

Abstract

This paper addresses the scheduling of systems that implement the abort and restart (AR) model. The AR model requires that preempted tasks are aborted. As a result high priority tasks run quickly and shared resources need not be protected (as tasks only work on copies of these resources). However there is significant wastage as low priority tasks may be subject to a series of aborts. We show that exact analysis of the AR model is intractable. A sufficient but tractable test is developed and is used to address the priority assignment issue. Again an optimal tractable algorithm is not available. The paper develops a priority assignment heuristic than is demonstrated to perform better than existing schemes.

1 Introduction

Abort-and-Restart (AR) is a scheme to support Priority-based Functional Reactive Programming (P-FRP). P-FRP has been introduced as a new functional programming scheme [3] for real-time systems. It combines the property of stateless execution from Functional Reactive Programming (FRP) [17], and supports priority assignments. Stateless execution means that tasks execute independently and no resource is locked by any task. To achieve this property of P-FRP, higher priority tasks can preempt lower priority tasks and the lower priority tasks are aborted and restarted after the higher priority tasks have finished execution. In the classical preemptive model, the lower priority tasks continue their execution but it is different for P-FRP; the lower priority tasks restart as new. Abort-and-Restart is the key operation for P-FRP so we call it the Abort-and-Restart (AR) model in this paper.

In the AR model, tasks can not access resources directly. Rather, tasks make copies of the resource at the beginning of their execution. The updated data is then copied back into the system once the tasks have completed their execution. In some situations, higher priority tasks preempt lower priority tasks. Once the higher priority tasks have completed execution, the lower priority tasks are aborted and restarted. The operation of abort-and-restart is to delete the old copy of the resource, and take a new copy from the system.

The classical preemptive model must deal with the problem of resource sharing. These problems can bring serious consequences. They may lead to inaccurate data, misses deadlines or deadlock. To cater for these problems various forms of priority inheritance and priority ceiling protocols have been developed[9, 10, 12, 15, 16]. One advantage of the AR model is that it does not face these problems because tasks do not access resources directly. The disadvantage is that aborted tasks delete the old copy of the resource and restart as new, hence the time spent before preemption is wasted. In this paper, we call this wasted time, the *abort cost*.

1.1 Contributions

This paper presents an analysis for the AR model. It first confirms that an exact analysis is not feasible as the critical instance cannot, in general, be identified in polynomial time. In the classical preemptive model, the critical instant is when all tasks are released at the same time. But this is not the case for the AR model. The second contribution is to develop a new schedulability test for the AR model. This test is sufficient for the model but is open to exact analysis. A final contribution is to address priority assignment for the AR model. General priority assignment schemes such as

rate monotonic, are not optimal for the AR model or the developed test. In this paper, we evaluate a number of existing priority assignment schemes and provide an improved (though still not optimal) priority assignment scheme which is termed *Execution-time-toward-Utilization Monotonic* (EUM).

1.2 Organization

The rest of the paper is organized as follows. Section 2 shows the system model and the related work. It explains the AR model and the related work introduces the previous research for critical instant, schedulability test and priority assignment. Section 3 is our analysis for the AR model. It consists of critical instant and schedulability test. Section 4 introduces a new priority assignment for the AR model, and it includes a pseudo code. Section 5 discusses our experiments. Section 6 states our conclusions and future work.

2 System Model and Related Work

In this paper, we consider the static priority scheduling of a set of sporadic tasks on a single processor. Each task consists of a potentially unbounded sequence of jobs.

The notations and formal definitions used in this paper are listed as follows:

N the number of tasks.

τ_i any given task in the system.

C_i the worst-case execution time for τ_i. (also reference to as WCET)

T_i period for τ_i.

U the total utilization of a task-set, $U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i}$.

P_i priority for τ_i.

D_i deadline for τ_i.

R_i response time for τ_i.

α_i the maximum abort cost for τ_i (see equation 1).

B_i blocking time for τ_i.

In general we allow $D_i \leq T_i$, although previous work and many of the examples in this paper have $D_i = T_i$.

2.1 The Abort-and-Restart Model

The Abort-and-Restart (AR) model [14, 13] is an implementation scheme for P-FRP. The classical preemptive model does not fit with P-FRP although it is similar to the AR model except for the operation of abort-and-restart. In the classical preemptive model, preempted tasks continue their job once higher priority tasks completed execution. The key concept of the AR model is that lower priority tasks are preempted and aborted by releases of higher priority tasks. Once the higher priority tasks have completed, the lower priority task are restarted as new.

Consider Table 1, there is a 2-task task-set. τ_1 is the highest priority task and has 3 ticks for WCET (Worst-case execution time). Task τ_2 has 4 ticks for WCET.

Task	Period	WCET	release offset	Priority
τ_1	12	3	3	1
τ_2	15	4	0	2

Table 1: An example task-set. (τ_1 has the highest priority.)

In Figure 1, τ_2 is released at 0 and executes until time 3, because of the arrival of τ_1, τ_2 is aborted at 3. τ_1 finishes its job at 6 and τ_2 is restarted as a new job so the spent time between 0 and 3 is wasted.

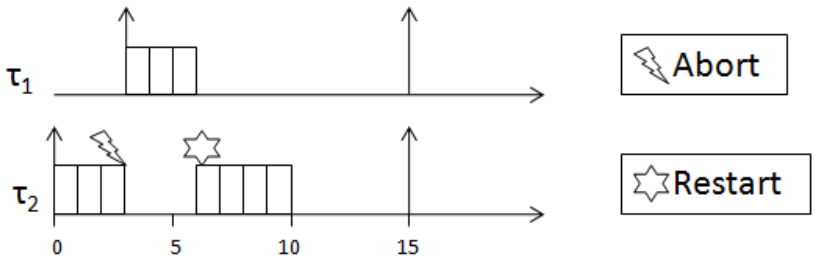

Figure 1: An example task-set.

2.2 Copy-and-Restore Operation

The Copy-and-Restore operation [3, 4, 5, 6] occurs when tasks begin or restart execution, they get a copy of the current state from the system. We call the copy *scratch state*, which is actually a set of data which will be used during the execution of the task. Tasks only change their copy so no tasks lock the

data resource. If higher priority tasks arrive, the lower priority task discards their copy. Once the higher priority tasks have completed execution, the lower priority tasks are aborted and restart. When a task has finished, the copy is restored into the system as an atomic action; this is illustrated in Figure 2 where τ_1 starts at time 0 and copies a set of data from the system. After six ticks, its job is done and then it restores the updated data into the system.

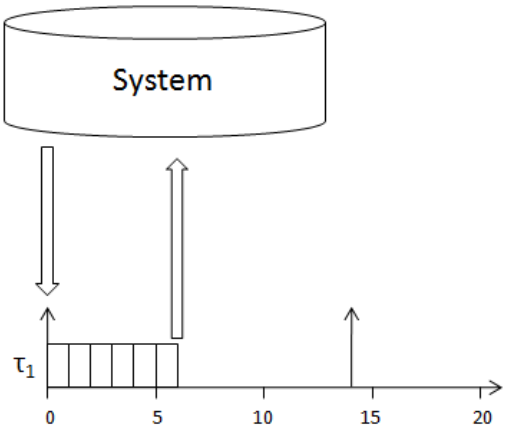

Figure 2: Copy-and-Restore Operation.

2.3 Related Research

In the paper of Ras and Cheng [13], the authors state that the critical instant argument from Liu and Layland [11] may not apply fully to the ANR model. In another paper from Belwal and Cheng [4], the authors also realised that a synchronous release of tasks does not lead to the worst-case response time. The simple example in Table 1 and Figure 1 illustrate this if τ_1 and τ_2 are released together then $R_2 = 6$. Figure 1 shows clearly $\tau_2 \geq 10$.

Ras and Cheng [13] also state that standard response time analysis is not applicable for the AR model, and assert that the abort cost can be computed by the following equation:

$$\alpha_i = \sum_{j=i+1}^{N} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \max_{k=i}^{j-1} C_k \tag{1}$$

α_i is the maximum abort cost for τ_i because the worst case is when a higher priority task aborts the lower priority task which has the biggest worst-case execution time. Equation (1) uses the number of releases for a task, which has a higher priority than τ_i, then multiplies this by the value of

C_k which is the maximum worst-case execution time between τ_i and highest priority task.

The central idea of their analysis is that the response time for the AR model can be computed by the combination of standard response time analysis and Equation (1). The new equation is as follows:

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + \alpha_i \tag{2}$$

Task	Period	WCET
τ_1	40	3
τ_2	12	4
τ_3	9	3

Table 2: A task-set is given from the paper[4].

Table 2 is a task-set given from paper [4]. τ_3 is the highest priority task and τ_1 is the lowest priority task. They applied equation (2) to the task-set and the calculation looks as below for τ_3:

1. $R_1^1 = 3 + (\left\lceil \frac{3}{9} \right\rceil \cdot 3 + \left\lceil \frac{3}{12} \right\rceil \cdot 4) + \left\lceil \frac{3}{9} \right\rceil \cdot 3 + \left\lceil \frac{3}{12} \right\rceil \cdot 4 = 17$
2. $R_1^2 = 3 + (\left\lceil \frac{17}{9} \right\rceil \cdot 3 + \left\lceil \frac{17}{12} \right\rceil \cdot 4) + \left\lceil \frac{17}{9} \right\rceil \cdot 3 + \left\lceil \frac{17}{12} \right\rceil \cdot 4 = 31$
3. $R_1^3 = 3 + (\left\lceil \frac{31}{9} \right\rceil \cdot 3 + \left\lceil \frac{31}{12} \right\rceil \cdot 4) + \left\lceil \frac{31}{9} \right\rceil \cdot 3 + \left\lceil \frac{31}{12} \right\rceil \cdot 4 = 51$

The task-set is deemed unschedulable.

In Section 3.2 we will derived an equivalent but more intuitive schedulability test for the AR model.

It was noted above that Rate Monotonic (RM) priority assignment is optimal for implicit deadline tasks in the standard model but is not optimal in the AR model An alternative assignmeent policy is introduced by Belwal and Cheng [3], namely: *Utilization Monotonic* (UM) priority assignment in which a higher priority is assigned to a task which has a higher utilization. Belwal and Cheng [3] state that it furnishes a better schedulability rate than RM. They also note that when RM and UM give the same ordering of priorities then that order is optimal.

3 New Analysis

In this section we derive a new sufficient test of schedulability for the AR model. But first we explain why the method cannot be exact.

3.1 Critical Instant for the AR model

First we consider periodic tasks and then sporadic. In the AR model, a critical instant occurs when a higher priority task aborts a lower priority, because the abort cost is added to the response time. For 2-task task-set, there is only one case where the highest priority task aborts the lowest priority task. This was illustrated in earlier example (in Table 1 and Figure 1). For 3-task task-set, there are two cases as the highest priority task can abort either of the two lower priority tasks. To generalise:

Lemma 3.1. *A task-set with N periodic tasks under the AR model has at least (N-1)! abort combinations.*

Proof. Consider a pure periodic task-set $\Gamma_N = \{\tau_1, \tau_2, ..., \tau_n\}$ and all tasks only release once. The highest priority task is τ_1 and the lowest priority task is τ_n. Task τ_1 has N - 1 choices of lower priority tasks to abort in each of their cases; τ_2 has N - 2 choices of lower priority tasks to abort. This continues until τ_{n-1} which has only one choice to abort. Finally, τ_n has zero choices because there is no lower priority task. When higher priority tasks are released more than once, the number of choices for those tasks are increased. The number of abort combinations is therefore at least $(N-1) * (N-2) * ... * 1$, which is (N-1)!. □

There is no information within the task set that would indicate which set of abort combination could give rise to the worst-case response times. Hence they all need to be checked for exact analysis. For sporadic tasks:

Lemma 3.2. *A sporadic task with a later release may bring a longer response time.*

Proof. In general, a sporadic task with its maximum arrival rate delivers the worst-case response time. Lemma 3.2 can be proved by showing a counter example. In Table 3, there is a 3-task task-set. Task τ_1 is a sporadic task and has the highest priority. It has a minimum inter-arrival time, 8. Other tasks are periodic tasks.

In Figure 3, the response time of τ_3 is 16 when the second job of τ_1 is released with the minimum inter-arrival time, 8.

Task	Period	WCET	Priority
τ_1	8	1	1
τ_2	20	2	2
τ_3	40	4	3

Table 3: A task-set with a sporadic task.

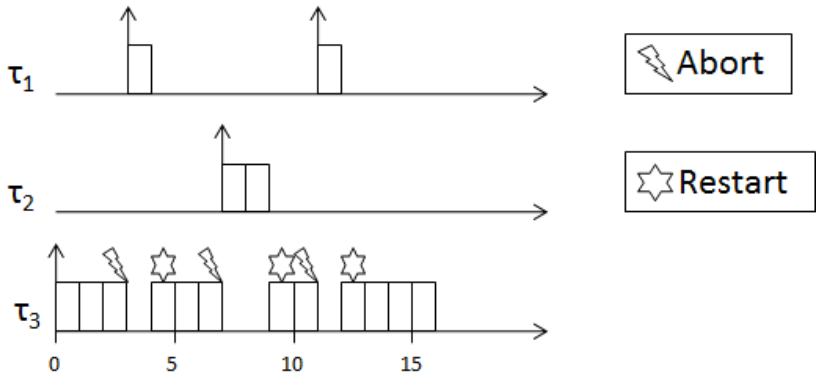

Figure 3: A time chart.

If, however, the second job of τ_1 is released 1 tick later, the response time of τ_3 will be 17. In this condition, a sporadic task with a later release may bring a longer response time. □

For a set of sporadic tasks exact analysis would require all possible released times to be checked.

Theorem 3.3. *Finding the critical instant for the AR model with periodic and sporadic tasks is intractable.*

Proof. Lemma 3.1 shows that there is at least $(N-1)!$ abort combinations for N periodic tasks, and all of which must be checked for the worst-case to be found. For sporadic tasks all possible release-time over a series of releases must be checked to determinate the worst-case impact of the sporadic task. These two properties in isolation and together show that this is an intractable number of release conditions to check in order to define the critical instant. □

In real-time scheduling, a schedulability test cannot be exact (sufficient and necessary) if the critical instant cannot be found in polynomial time.

3.2 New Formulation for schedulability tests

In the last section we showed that an exact analysis for the AR model is not possible. In this section we derive a sufficient test that is itself tractable. Hence we have traded sufficiency with tractability. We believe this new test is more intuitive than those previously published,

Given a priority assignment, the worst-case response time of task τ_n (priority P_n) will depend only on the behaviour of tasks of priority greater than P_n. Consider the interference caused by a single release of task $\tau_i (P_i > P_n)$. In the worst-case τ_i will abort (just before it completes) a task with a lower priority than τ_i but with the maximum execution time of all lower priority tasks. Let the aborted task be τ_a, so $P_i > P_a \geq P_n$ and $C_a = \max_{\forall j \in hep_n \bigcap lp_i} C_j$.

The impact of τ_i will therefore be, in the worst-case, C_i at priority P_i and C_a at priority P_a. As $P_a \geq P_n$ this is equivalent (for τ_n) to τ_i having an execution time of $C_i + C_a$ at priority P_i. Let $\beta_i = C_i + C_a$. The original task-set with computation times C_i is transposed into a task-set with β_i. This is now a conventional task-set, so the critical instant is when there is a synchronous release. (The maximum interference on τ_n must occur when all higher priority tasks arrive at their maximum rate, initially at the same time, and all have their maximum impact.)

The worst-case for the AR model is that any higher priority task aborts a lower priority task which has a biggest possible worst-case execution time, and that this abort occurs just before the aborted task would actually complete. By this process, a new value β_j for τ_j is combined by C_j and C_k:

$$\beta_j = C_j + \max_{\forall k \in hep_i \bigcap lp_j} C_k \tag{3}$$

where β_j is the new value for the WCET of τ_j, C_j is the original WCET of τ_j and C_k is the bigger execution time of a task with priority between τ_i and τ_j but τ_j is not included. The response time analysis applies to τ_i. Note that in general the β_j values will depend on the task under investigation.

Task	Period	C	β	Priority
τ_1	28	2	7(2+5)	1
τ_2	120	3	8(3+5)	2
τ_3	140	4	9(4+5)	3
τ_4	200	5	5(5+0)	4

Table 4: An example with new WCET for 4-task task-set.

In Table 4, there is an example task-set. Deadline is equal to period and the time unit is a tick. The highest priority is 1. The response time of task τ_4 is being computed.

The β values are computed by Equation (3). In this example we consider the response time for τ_4 so $i = 4$. For β_1, j is 1 and C_k is higher than or equal to τ_4 but lower than τ_1. The calculation is $\beta_1 = C_1 + C_4$, so the result of β_1 is $2 + 5 = 7$.

For β_4, i and j are 4. C_k is higher than or equal to τ_4 but lower than τ_4 so no task is matched, so the result of β_4 is $5+0 = 5$. After all the β values had been calculated, we use β instead of C in the response time analysis; that is:

$$R_4 = \beta_4 + \sum_{\forall j \in hp_4} \left\lceil \frac{R_4}{T_j} \right\rceil \cdot \beta_j \tag{4}$$

This is solved in the usual way by forming a recurrence relationship. The calculations are as follows:

1. $RT_4^1 = 5 + (\left\lceil \frac{5}{28} \right\rceil \cdot 7 + \left\lceil \frac{5}{120} \right\rceil \cdot 8 + \left\lceil \frac{5}{140} \right\rceil \cdot 9) = 29$
2. $RT_4^2 = 5 + (\left\lceil \frac{29}{28} \right\rceil \cdot 7 + \left\lceil \frac{29}{120} \right\rceil \cdot 8 + \left\lceil \frac{29}{140} \right\rceil \cdot 9) = 36$
3. $RT_4^3 = 5 + (\left\lceil \frac{36}{28} \right\rceil \cdot 7 + \left\lceil \frac{36}{120} \right\rceil \cdot 8 + \left\lceil \frac{36}{140} \right\rceil \cdot 9) = 36$

To compare the result with the equation of Ras and Cheng [13]. Their calculation would be:

1. $RT_4^1 = 5+(\left\lceil \frac{5}{28} \right\rceil \cdot 2+\left\lceil \frac{5}{120} \right\rceil \cdot 3+\left\lceil \frac{5}{140} \right\rceil \cdot 4)+\left\lceil \frac{5}{28} \right\rceil \cdot 5+\left\lceil \frac{5}{120} \right\rceil \cdot 5+\left\lceil \frac{5}{140} \right\rceil \cdot 5 = 29$
2. $RT_4^2 = 5+(\left\lceil \frac{29}{28} \right\rceil \cdot 2+\left\lceil \frac{29}{120} \right\rceil \cdot 3+\left\lceil \frac{29}{140} \right\rceil \cdot 4)+\left\lceil \frac{29}{28} \right\rceil \cdot 5+\left\lceil \frac{29}{120} \right\rceil \cdot 5+\left\lceil \frac{29}{140} \right\rceil \cdot 5 = 36$
3. $RT_4^3 = 5+(\left\lceil \frac{36}{28} \right\rceil \cdot 2+\left\lceil \frac{36}{120} \right\rceil \cdot 3+\left\lceil \frac{36}{140} \right\rceil \cdot 4)+\left\lceil \frac{36}{28} \right\rceil \cdot 5+\left\lceil \frac{36}{120} \right\rceil \cdot 5+\left\lceil \frac{36}{140} \right\rceil \cdot 5 = 36$

The results are the same but Equation (4) clearly involves less computation.

To compute the worst-case response time for τ_3 requires the β values to be recomputed (as show in Table 5).

The test derived above whilst more intuitive and more efficiently solved is nevertheless equivalent to take previous published.

Theorem 3.4. *Equations (2) and (4) are equivalent.*

Task	Period	C	β	Priority
τ_1	28	2	6(2+4)	1
τ_2	120	3	7(3+4)	2
τ_3	140	4	4(4+0)	3

Table 5: β values for τ_3

Proof. We rephrase Equation (2) as below:

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \max_{k=i}^{j-1} C_k \tag{5}$$

and simplify:

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + \max_{k=i}^{j-1} C_k \tag{6}$$

both $\max_{k=i}^{j-1} C_k$ and $\max_{\forall k \in hep_i \bigcap lp_j} C_k$ are to pick a bigger WCET task with priority is higher or equal to $\tau_(i)$ and lower than τ_j, so we rephrase it again.

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + \max_{\forall k \in hep_i \bigcap lp_j} C_k \tag{7}$$

Equation (3) replaces into Equation (7) as below:

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \beta_j \tag{8}$$

Finally, β_i replaces to C_i using Equation (3). □

As equation (2) was previously proved to be sufficient for the AR model [13] it follows that equation (4) is similarly sufficient.

Although the equations are equivalent, Equation (4) is in the standard form for response time analysis and is therefore amenable to the many ways that have been found to efficiently solve this form of analysis[8]. It is also in a form that allows the issue of priority assignment to be addressed.

4 Priority assignment schemes

In the section on related research, Rate Monotonic (RM) and Utilization Monotonic (UM) have been introduced as possible priority assignment schemes

for the AR model. Here, we introduce another priority assignment called Execution-time Monotonic (EM) which assigns a higher priority to a task which has a bigger worst case execution time. An inspection of Equation (3) shows that the minimum execution times (the β values) are obtained when priority is order by execution time. Although this does not necessarily minimise utilisation, it may furnish an effective priority assignment scheme.

For many scheduling problems, Audsley Algorithm furnishes an optimal priority assignment; i.e. the algorithm can find a schedulable priority ordering if such an ordering exists[1, 2]. Unfortunately one of the prerequisites for Audsley's Algortihm does not hold. Specifically the response time of a task depends not only on the set of higher priority tasks but also on their relative order (which is not allowed).

Task	Period	WCET	β	Priority
τ_1	100	5	9(5+4)	1
τ_2	120	4	7(4+3)	2
τ_3	140	3	5(3+2)	3
τ_4	200	2	2(2+0)	4

Table 6: The response time of τ_4 is 23.

In Table 6 τ_4 is the lowest priority task and its response time is 23. After τ_2 and τ_3 (higher priority) swapped their priorities, the response time for τ_4 is changed to 24 as shown in Table 7.

Task	Period	C	β	Priority
τ_1	100	5	9(5+4)	1
τ_3	140	3	7(3+4)	2
τ_2	120	4	6(4+2)	3
τ_4	200	2	2(2+0)	4

Table 7: The response time of τ_4 is 24.

4.1 New Algorithm

The Exhaust Search (ES) Algorithm is optimal for any model but the complexity is the factorial of the number of tasks. Therefore, it is not applicable in general but it can validate other algorithms for small values of N. By comparison with ES, both UM and EM are not optimal. Sometimes, there are

more than one schedulable orderings for a task-set. Some tasks are scheduled by EM but not UM, vice versa. Their relationship is shown in Figure 4.

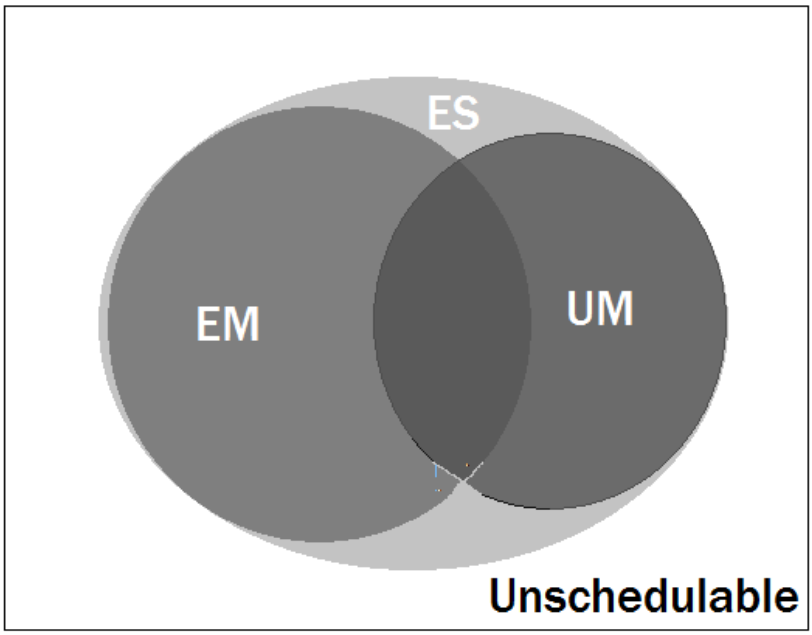

Figure 4: Relationships between UM, EM and ES.

These circles represent task-sets that are scheduled by the labeled algorithms. White space are task-sets that are unscheduled by any algorithm. ES covers both UM and EM because it is optimal. UM and EM are overlapped because some task-sets are scheduled by both of them. In a later section, the experiments show that UM and EM have similar results. If an algorithm dominates both UM and EM, it will offer a better schedulability rate.

We derive a new algorithm that starts with EM ordering and tests the tasks in priority order starting with the highest priority task. If any task can not be scheduled then try to find a higher priority task which has less utilization. The ordering begins from the failed task to the top. If a task is found then shift down the higher priority task below the lower priority task. If no task is found, the task-set is deemed to be not schedulable. Pseudo code of the new algorithm is given in Algorithm 1. The explanations of functions are list below:

- sortByEM(ts) = do a EM priority assignment for task-set ts
- st(i) = Schedulability Test for task i

- u(i) = get utiliation of task i
- d(i) = get deadline of task i
- move(x, y) = move task x below task y (top is higher priority).

```
sortByEM(taskset);
for i in 2 .. N do
    if st(i) = true then
        continue;
    else
        for j in i-1 .. 1 do
            if u(j) < u(i)||(u(j) = u(i)&d(j) > d(i)) then
                found = true;
                move(j,i);
                i=j-1;
            else
                found = false;
            end
            if not found then
                return fail;
            end
        end
    end
end
return pass;
```

Algorithm 1: A pseudo code of the new algorithm.

An example of the use of the algorithm is given in Table 8. Again deadline is equal to period; RT is response time. Note only C values are given in the table, the necessary β values are dependent on which task is actually been tested, they must be re-computed for each task.

Task	Period	C	U	Priority	RT
τ_1	60	6	0.1	1	6
τ_2	50	5	0.1	2	16
τ_3	32	4	0.125	3	24
τ_4	25	3	0.12	4	30 (X)
τ_5	100	2	0.02	5	

Table 8: An example task-set fails in EM ordering.

The task-set is initially ordered by the EM algorithm. The schedulability test begins from the top. τ_1, τ_2 and τ_3 meet their deadlines. A missed deadline occurs at τ_4 so the algorithm searches for a less utilization task from τ_3 to τ_1. The utilization of τ_2 is 0.1 which is less than τ_4, τ_2 shifts down below τ_4. The priority of τ_3 shifts up to 2. The priority of τ_4 shifts up to 3. The priority of τ_2 changes to 4.

Task	Period	C	U	Priority	RT
τ_1	60	6	0.1	1	6
τ_3	32	4	0.125	2	14
τ_4	25	3	0.12	3	20
τ_2	50	5	0.1	4	50
τ_5	100	2	0.02	5	88

Table 9: The task-set is scheduled by EUM algorithm.

In Table 9, the task-set has had its priorities changed and is schedulable after the shifting. By the nature of shifting down less utilization tasks to the bottom, UM ordering is the worst-case. The algorithm is intuitively a set of transformations starting at EM and moving towards UM. It dominates both EM and UM. We name it, the Execution-time-toward-Utilization Monotonic (EUM). A set diagram for EUM is shown in Figure 5.

4.2 Time complexity

EUM priority assignment starts with EM ordering and the worst-case is when a task-set can only be scheduled by UM ordering (or is not schedulable at all, by only fails at the last task). The lower priority tasks shift up with higher priority level one by one until the task-set is in UM ordering. After each shifting, a schedulability test for the shifted task is undertaken. In the analysis of time complexity, we count each schedulability test rather than the computational complexity. For instance, a N-task task-set starts with EM ordering and the task-set is only scheduled by UM ordering which is completely opposite to EM. Task τ_1 is the highest priority task and the lowest priority task is τ_N. The algorithm tests each task by the priority ordering from high to low. In the first recursion, it takes N tests from τ_1 to τ_N and it fails at τ_N. According to the nature of the algorithm, a failed task shifts up if a higher priority task has less utilization. The current status of the task-set is in reverse order of UM. In other words, τ_N will shift up and test again one by one, i.e. τ_N to τ_{N-1}, τ_{N-1} to τ_{N-2} and so on. For the recursion of τ_N, the

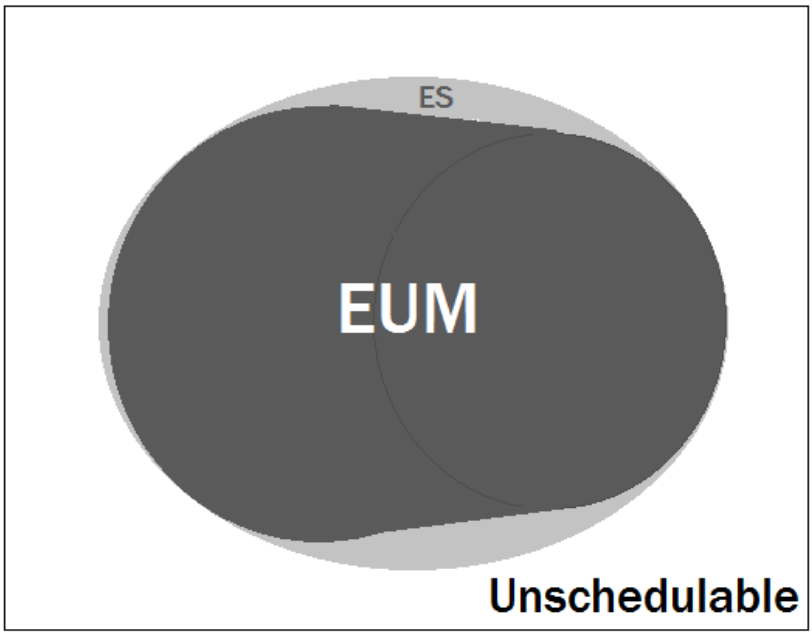

Figure 5: A set diagram for EUM.

number of tests is N-1. The number of tests for τ_{N-1} is N-2. The number of tests for τ_2 is 1, and τ_1 is 0. The equation can be represented as below:

$$\sum_{k=1}^{N-1} k = \frac{N(N-1))}{2} \tag{9}$$

so the time complexity of EUM priority assignment is $\frac{N(N-1))}{2}$ in worst-case. Clearly, EUM dominates EM and UM because the algorithm starts with EM ordering and ends at UM ordering in the worst-case steps. Unlike ES it is a tracable task.

5 Experimental Evaluation

The experiments are separated into two parts. First, the EUM algorithm is compared with the non-optimal group (RM, UM and EM). Secondly, the EUM algorithm is compared with ES, the optimal algorithm for the largest feasible value for N (8). All experiments used the same parameters but different priority assignments. The parameters are:

- Deadline is equal to period.

- All tasks are periodic.
- A set of N utilization values U_i was generated by the UUniFast Algorithm [7].
- Task periods were generated between 500 and 5000 according to a log-uniform distribution[1]. And the value T_i assigns to τ_i.
- Task execution times are: $C_i = U_i \cdot T_i$
- Utilization for task-sets are ranged between 10% and 70%.
- Each utilization rate generates 10000 different task-sets, i.e. $U = 10\%$ generates 10000 task-sets, $U = 11\%$ generates another 10000 task-sets, and so on.
- The numbers of task for the non-optimal group are 5, 10, 15 and 20. A maximum of 8 task is all that can be accomplished by ES. The final experiment is therefore restricted to just 8 tasks.

For all diagrams, the X-axis is Utilization rate and the Y-axis is the Schedulability rate, i.e. the percentage of task sets that were deemed schedulable.

In Figure 6 the number of tasks is 5. We observe that RM has the worst schedulability, UM and EM are quite similar before $U = 27\%$. After that, UM is better than EM. EUM is of course always better than the others.

In Figure 7 the number of tasks is 10; RM is still the worst and EM is better than UM.

In Figures 8 and 9 the numbers of tasks is 15 and 20. Again RM is the worst; EM is better than UM and EUM is the best. The two diagrams have a similar pattern. Results for larger value of N are similar (but not included).

For the final comparing experiment of EUM and ES, ES is the factorial of the number of task so we picked the number of task as large as possible. In Figure 10 the number of tasks is 8. The diagram shows the result that EUM is very close to ES. Indeed it is impossible to distinguish between them in the diagram. Nevertheless EUM is not optimal, the figure contains in total 410,000 task sets of which ES deemed 137,366 schedulable and EUM (136,712), a difference of 654 (i.e. schedulable by ES but not by EUM). Finally, Tables 10 and 11 show an example task-set which is schedulable by ES but not by EUM.

[1]The log-uniform distribution of a variable x is such that ln(x) has a uniform distribution.

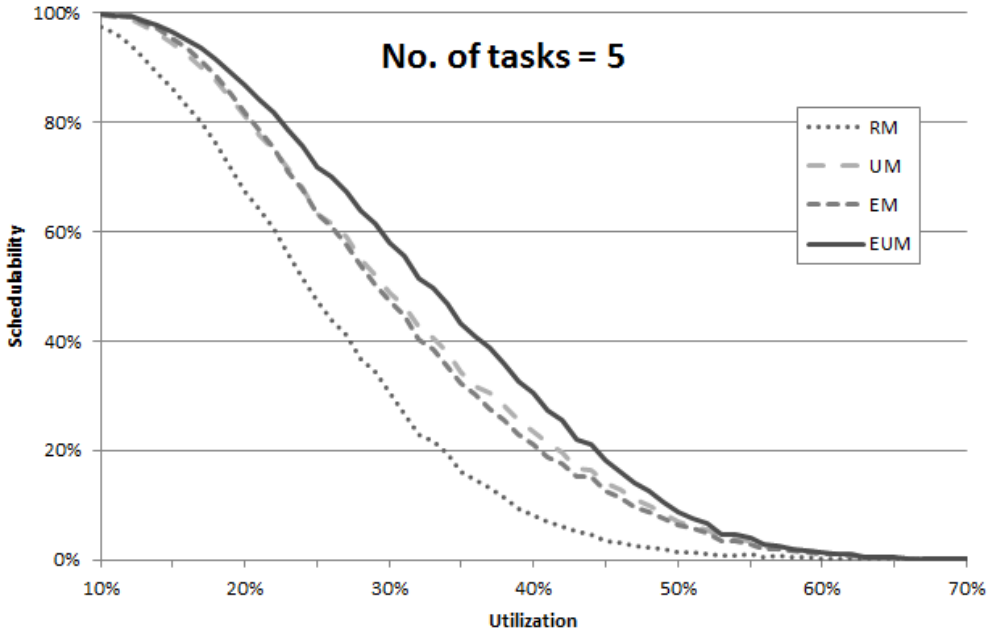

Figure 6: The number of tasks is 5.

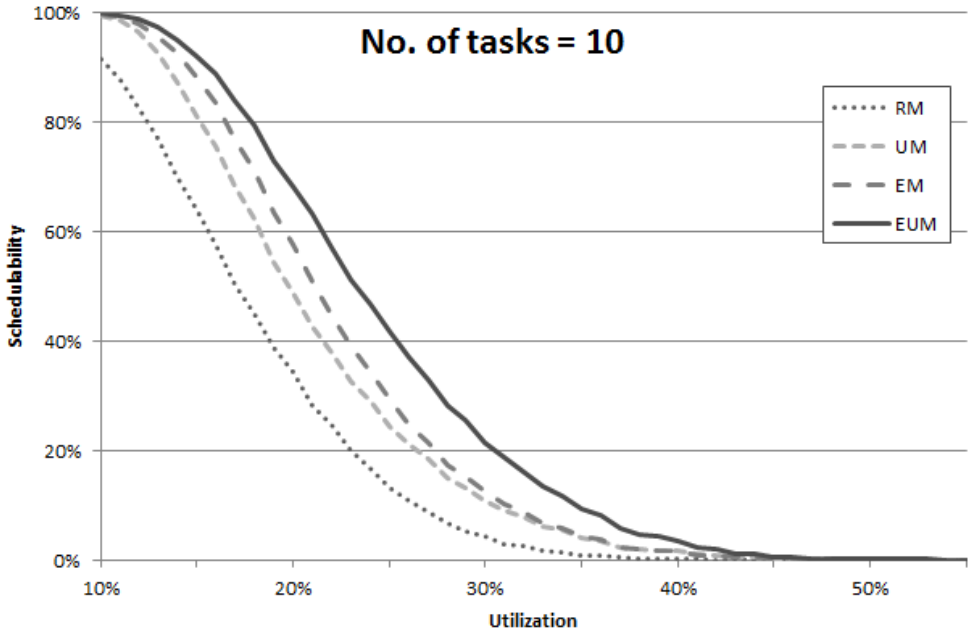

Figure 7: The number of tasks is 10.

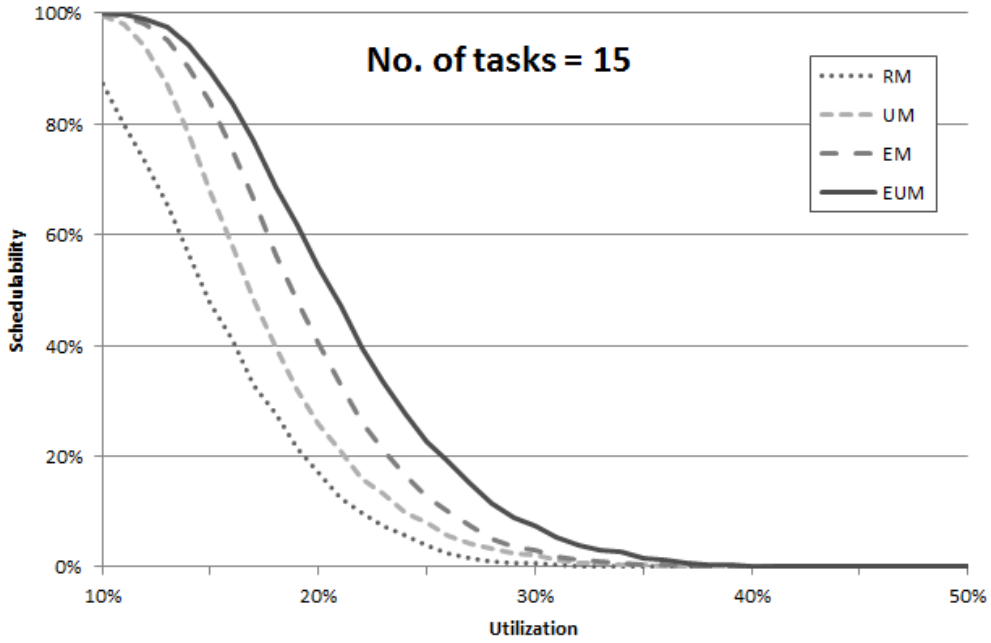

Figure 8: The number of tasks is 15.

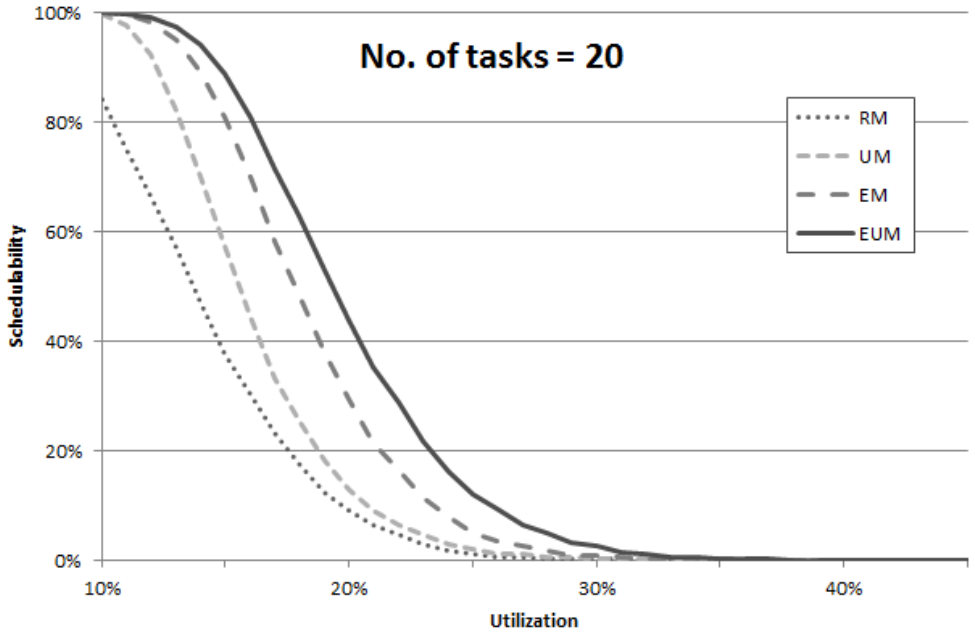

Figure 9: The number of tasks is 20.

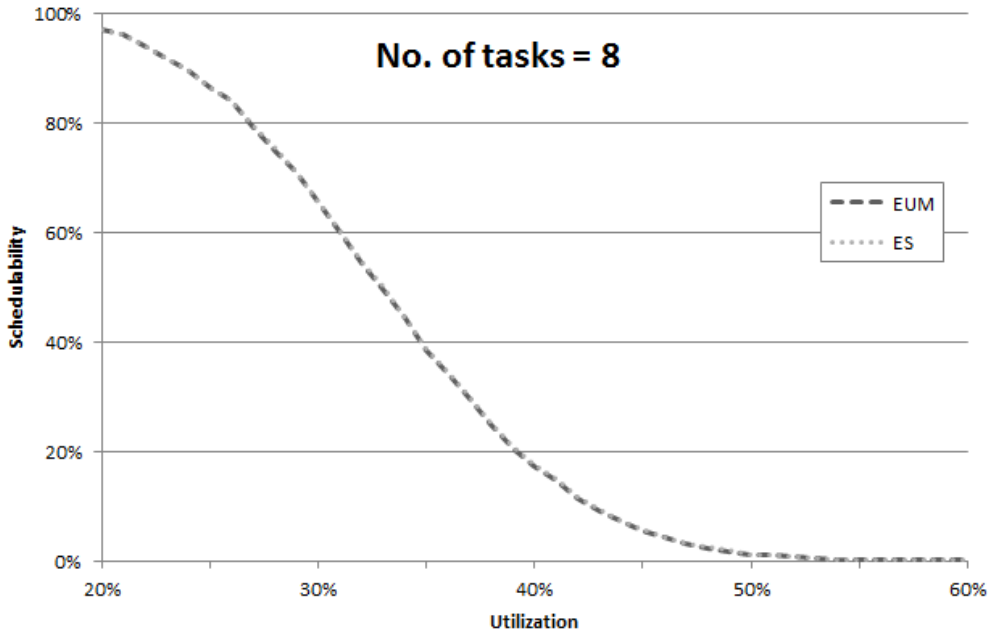

Figure 10: The number of tasks is 8.

Task	Period	C	U	Priority	RT
τ_3	1430	179	0.125	1	179
τ_6	1035	90	0.087	2	359
τ_2	656	49	0.075	3	457
τ_5	1269	27	0.021	4	511
τ_7	1925	131	0.068	5	X
τ_4	2579	31	0.012	6	X
τ_1	2688	8	0.003	7	X
τ_8	1042	7	0.007	8	X

Table 10: A task-set deemed not schedulable by EUM algorithm.

Task	Period	C	U	Priority	RT
τ_7	1925	131	0.068	1	131
τ_3	1430	179	0.125	2	489
τ_2	656	49	0.075	3	587
τ_6	1035	90	0.087	4	947
τ_8	1042	7	0.007	5	961
τ_5	1269	27	0.021	6	1035
τ_4	2579	31	0.012	7	1264
τ_1	2688	8	0.003	8	1746

Table 11: The task-set is schedulable by ES algorithm.

6 Conclusion

The AR model has been proposed as a means of implementing priority-based functional reactive programming. Any released task, if it has a higher priority than the current running task, will abort that task. It can therefore immediately make progress. As a consequence the aborted task must re-start its execution when it is next executed.

We have confirmed that the AR model is intractable, in the sense that exact analysis is not possible due to the number of cases that need to be investigated in order to identify the worst-case release conditions (the critical instant). Nevertheless a tractable sufficient test has been developed that allow the issue of priority ordering to be addressed.

Unfortunately optimal priority ordering is also problematic with the AR model. Deadline (or Rate) monotonic ordering is demonstrably not optimal. Also the optimal Audsley's algorithm is not applicable. We have however developed a heuristic (called EUM) that performs well and has only N^2 complexity (for N tasks). On small sized systems ($N = 8$) EUM performs almost identically to an optimal scheme (using exhaustive search). For larger numbers of N (where exhaustive search is infeasible) it performs better than previous published approaches.

References

[1] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Report YCS 164, University of York, 1991.

[2] N.C. Audsley. On Priority Assignment in Fixed Priority Scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[3] C. Belwal and A.M.K. Cheng. On Priority Assignment in P-FRP. *RTAS*, pages 45–48, 2010.

[4] C. Belwal and A.M.K. Cheng. Determining Actual Response Time in P-FRP. In Ricardo Rocha and John Launchbury, editors, *Practical Aspects of Declarative Languages*, volume 6539 of *Lecture Notes in Computer Science*, pages 250–264. Springer Berlin/Heidelberg, 2011.

[5] C. Belwal and A.M.K. Cheng. Determining Actual Response Time in P-FRP Using Idle-Period Game Board. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, 0:136–143, 2011.

[6] C. Belwal and A.M.K. Cheng. Feasibility Interval for the Transactional Event Handlers of P-FRP. In *Proc. of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM, pages 966–973, Washington, DC, USA, 2011.

[7] E. Bini and G. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, 30:129–154, 2005.

[8] R.I. Davis, A. Zabos, and A. Burns. Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems. *IEEE Transactions on Computers*, 57(9):1261–1276, 2008.

[9] S. Kim, S. Hong, and T.-H. Kim. Integrating real-time synchronization schemes into preemption threshold scheduling. In *Object-Oriented Real-Time Distributed Computing, (ISORC)*, pages 145–152, 2002.

[10] S. Kim, S. Hong, and T.-H. Kim. Perfecting preemption threshold scheduling for object-oriented real-time system design: From the perspective of real-time synchronization. In *Proc. of the joint conference on Languages, compilers and tools for embedded systems: software and compilers for embedded systems*, LCTES/SCOPES, pages 223–232, New York, NY, USA, 2002. ACM.

[11] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, 1973.

[12] J. Ras and A.M.K Cheng. An Evaluation of the Dynamic and Static Multiprocessor Priority Ceiling Protocol and the Multiprocessor Stack

Resource Policy in an SMP System. In *Real-Time and Embedded Technology and Applications Symposium*, pages 13–22, 2009.

[13] J. Ras and A.M.K Cheng. Response Time Analysis for the Abort-and-Restart Task Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm. In *Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 305–314, 2009.

[14] J. Ras and A.M.K. Cheng. Response Time Analysis of the Abort-and-Restart Model under Symmetric Multiprocessing. In *Computer and Information Technology (CIT)*, pages 1954–1961, 2010.

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

[16] H. Takada and K. Sakamura. Real-Time Synchronization Protocols with Abortable Critical Sections. pages 44–52, 1994.

[17] Z. Wan and P. Hudak. Functional reactive programming from first principles. In *Proc. of the ACM SIGPLAN*, PLDI, pages 242–252. ACM, 2000.