# Schedulability Analysis for the Abort-and-Restart (AR) Model

H.C. Wong
Real-Time Systems Research Group,
Department of Computer Science,
University of York, UK.
hw638@york.ac.uk

A. Burns
Real-Time Systems Research Group,
Department of Computer Science,
University of York, UK.
alan.burns@york.ac.uk

## 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. Two sufficient but tractable tests are developed and are 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 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 assignment. 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 cannot 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.

*RTNS 2014* , October 8 - 10 2014, Versailles, France
Copyright 2014 ACM 978-1-4503-2727-5/14/10 ...$15.00.
http://dx.doi.org/10.1145/2659787.2659813.

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 [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 Motivation for the AR model

Nowadays, computers have more power of execution than before. In concurrent programming, sometimes programmers consider how to enhance the correctness of programs rather than reduce the overhead. For a real-time system, it is more complicated because of timing constraints and priorities. A concurrency control mechanism for a system is important because it affects the correctness and the schedulability.

The AR model provides strong correctness guarantees on dealing with shared resource. And it also supports FRP which has been used for the domains of computer animation, computer vision, robotics and control systems [9]. Original FRP cannot be used for real-time systems but P-FRP has rectified this. Hence the AR model allows P-FRP to be used for real-time systems.

A real-time database system can be simply defined as a database system with timing constraints [12]. The system receives a high demand of requests and the responses are required to be sent out before their deadlines. A transactional memory can handle shared resource in a convenient way, and the AR scheme can be applied to it. For example, all transactions must be consistent and up-to-date for the stock market. The AR model has the properties of atomic execution and preemption. A transaction will not conflict with another transaction, and an urgent transaction can be execute as soon as possible.

Real-time Java is designed to allow programmers to develop a real-time application using the Java language. Preemptible Atomic Regions (PAR) is a new concurrency control abstraction for real-time systems [11]. The basic notions of the AR model and the PAR model are similar but PAR makes a log of shared resource and then the state of resource will be rolled back if the task is preempted. The paper [11] introduced the example of PAR for real-time Java. In other words, the AR model has been implemented in a common programming language.

## 1.2 Contributions and Organisation

This paper builds on the preliminary work in this area published in JRWRTC [18]. An analysis for the AR model is presented. 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 two new schedulability tests for the AR model. These tests are sufficient for the model but are open to exact analysis. In this paper, the tighter schedulability test is termed the *Multi-bag* approach. A final contribution is to address priority assignment for the AR model. General priority assignment schemes such as rate and deadline 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-Utilisation Monotonic* (EUM).

The rest of the paper is organised 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 identification, schedulability test and priority assignment. Section 3 is our analysis for the AR model. It consists of critical instant and schedulability tests. Section 4 introduces a new priority assignment for the AR model, and it includes pseudo code. Section 5 discusses our experiments. Finally Section 6 states our conclusions.

## 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.

$\tau_i$ any given task in the system.

$C_i$ the worst-case execution time for $\tau_i$ (also referred to as WCET).

$T_i$ period for $\tau_i$.

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

$P_i$ priority for $\tau_i$.

$D_i$ deadline for $\tau_i$.

$R_i$ response time for $\tau_i$.

$\alpha_i$ the maximum abort cost for $\tau_i$ (see equation 1).

$\tilde{C}_j^i$ the new value for the WCET of $\tau_j$, the biggest abort cost is picked between $\tau_j$ and $\tau_i$ (see equation 3).

$hp_{(i)}$ higher than the priority of $\tau_i$.

$hep_{(i)}$ higher or equal to the priority of $\tau_i$.

$lp_{(j)}$ lower than the priority of $\tau_j$.

$E_j(R_i)$ the number of release of $\tau_j$ within $R_i$.

$M_{i,j}$ a bag for $\tau_j$ when scheduling $\tau_i$.

$\gamma_{i,j}$ the total abort cost for $\tau_j$ when scheduling $\tau_i$.

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 [13, 14] 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. $\tau_1$ is the highest priority task and has 3 ticks for WCET (Worst-case execution time). Task $\tau_2$ has 4 ticks for WCET.

**Table 1: An example task-set.**

| Task | Period | WCET | Release offset | Priority |
|---|---|---|---|---|
| $\tau_1$ | 12 | 3 | 3 | 1 |
| $\tau_2$ | 15 | 4 | 0 | 2 |

In Figure 1, $\tau_2$ is released at 0 and executes until time 3, because of the arrival of $\tau_1$, $\tau_2$ is aborted at 3. $\tau_1$ finishes its job at 6 and $\tau_2$ is restarted as a new job so the spent time between 0 and 3 is wasted.

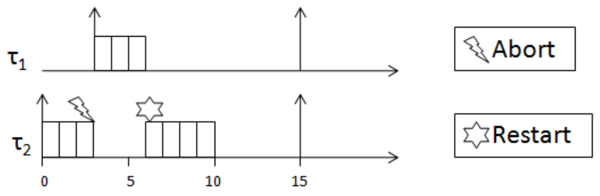

**Figure 1: An example task-set.**

## 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 $\tau_1$ 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. Although atomic, copy-and-restore cannot be undertaken instantaneously. Hence a high priority task cannot abort a lower priority task while it is restoring state; the higher priority task must block lending to a blocking term in the analysis. For ease of presentation however this term is omitted from the scheduling equations presented in this paper.

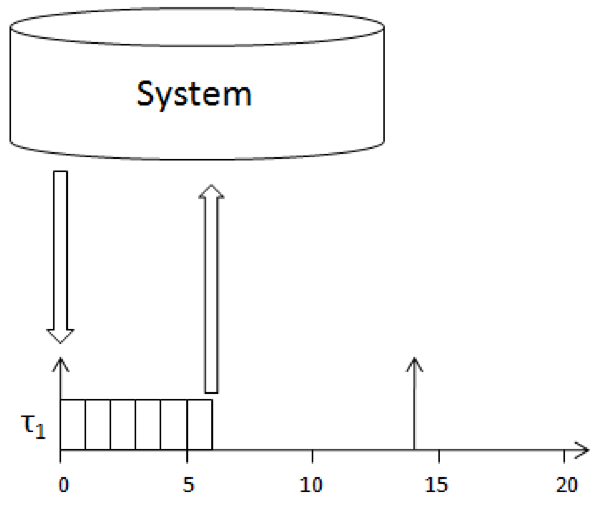

**Figure 2: Copy-and-Restore Operation.**

## 2.3 Related Research

In the paper of Ras and Cheng [13], the authors state that the critical instant argument from Liu and Layland [10] may not apply fully to the AR 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 illustrates this if $\tau_1$ and $\tau_2$ are released together then $R_2 = 6$. Figure 1 shows clearly $R_2 \geq 10$.

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:

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

$\alpha_i$ is the maximum abort cost for $\tau_i$ 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 $\tau_i$, then multiplies this by the value of $C_k$ which is the maximum worst-case execution time between $\tau_i$ and the higher 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:

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

Table 2 is a task-set given from paper [4]. $\tau_3$ is the highest priority task and $\tau_1$ is the lowest priority task. They applied

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

| Task | Period | WCET |
|---|---|---|
| $\tau_1$ | 40 | 3 |
| $\tau_2$ | 12 | 4 |
| $\tau_3$ | 9 | 3 |

Equation (2) to the task-set and the calculation looks as below for $\tau_3$:

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

The task-set is deemed unschedulable.

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

It was noted [3] 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 assignment policy is introduced by Belwal and Cheng [3], namely: *Utilisation Monotonic* (UM) priority assignment in which a higher priority is assigned to a task which has a higher utilisation. 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 (for their analysis).

## 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 an 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 $\tau_1$ and the lowest priority task is $\tau_n$. Task $\tau_1$ has N - 1 choices of lower priority tasks to abort in each of their cases; $\tau_2$ has N - 2 choices of lower priority tasks to abort. This continues until $\tau_{n-1}$ which has only one choice to abort. Finally, $\tau_n$ 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 $\tau_1$ is a sporadic task and has the highest priority. It has a minimum inter-arrival time, 8. Other tasks are periodic tasks.

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

| Task | Period | WCET | Priority |
|---|---|---|---|
| $\tau_1$ | 8 | 1 | 1 |
| $\tau_2$ | 20 | 2 | 2 |
| $\tau_3$ | 40 | 4 | 3 |

In Figure 3, the response time of $\tau_3$ is 16 when the second job of $\tau_1$ is released with the minimum inter-arrival time, 8.

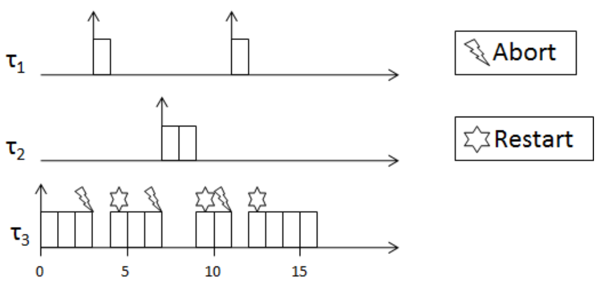

**Figure 3: A time chart.**

If, however, the second job of $\tau_1$ is released 1 tick later, the response time of $\tau_3$ 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 $\tau_n$ (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 $\tau_i$ will abort (just before it completes) a task with a lower priority than $\tau_i$ but with the maximum execution time of all lower priority tasks. Let the aborted task be $\tau_a$, 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 $\tau_i$ 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 $\tau_n$) to $\tau_i$ having an execution time of $C_i + C_a$ at priority $P_i$. Let $\tilde{C_i^n} = C_i + C_a$. The original task-set with computation times $C_i$ is transposed into a task-set with $\tilde{C_i^n}$. This is now a conventional task-set, so the critical instant is when there is a synchronous release. (The maximum interference on $\tau_n$ 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 the 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 $\tilde{C_j^i}$ for $\tau_j$ is combined by $C_j$ and $C_k$:

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

where $\tilde{C_j^i}$ is the new value for the WCET of $\tau_j$, $C_j$ is the original WCET of $\tau_j$ and $C_k$ is the biggest execution time of a task with priority between $\tau_i$ and $\tau_j$ but $\tau_j$ is not included. The response time analysis applies to $\tau_i$. Note that in general the $\tilde{C_j^i}$ values will depend on the task under investigation.

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 $\tau_4$ is being computed.

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

| Task | Period | C | $\tilde{C_j^4}$ | Priority |
|---|---|---|---|---|
| $\tau_1$ | 28 | 2 | 7(2+5) | 1 |
| $\tau_2$ | 120 | 3 | 8(3+5) | 2 |
| $\tau_3$ | 140 | 4 | 9(4+5) | 3 |
| $\tau_4$ | 200 | 5 | 5(5+0) | 4 |

The $\tilde{C_j^4}$ values are computed by Equation (3). In this example we consider the response time for $\tau_4$ so $i = 4$. For $\tilde{C_1^4}$, j is 1 and $C_k$ is higher than or equal to $\tau_4$ but lower than $\tau_1$. The calculation is $\tilde{C_1^4} = C_1 + C_4$, so the result of $\tilde{C_1^4}$ is $2 + 5 = 7$.

For $\tilde{C_4^4}$, i and j are 4. $C_k$ is higher than or equal to $\tau_4$ but lower than $\tau_4$ so no task is matched, so the result of $\tilde{C_4^4}$ is $5 + 0 = 5$. After all the $\tilde{C_j^4}$ values had been calculated, we use $\tilde{C_j^i}$ instead of $C$ in the response time analysis; that is:

$$R_4 = \tilde{C_4^4} + \sum_{\forall j \in hp_4} \left\lceil \frac{R_4}{T_j} \right\rceil \cdot \tilde{C_j^4} \tag{4}$$

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

1. $R_4^1 = 5 + (\lceil \frac{5}{28} \rceil \cdot 7 + \lceil \frac{5}{120} \rceil \cdot 8 + \lceil \frac{5}{140} \rceil \cdot 9) = 29$
2. $R_4^2 = 5 + (\lceil \frac{29}{28} \rceil \cdot 7 + \lceil \frac{29}{120} \rceil \cdot 8 + \lceil \frac{29}{140} \rceil \cdot 9) = 36$

3. $R_4^3 = 5 + (\lceil \frac{36}{28} \rceil \cdot 7 + \lceil \frac{36}{120} \rceil \cdot 8 + \lceil \frac{36}{140} \rceil \cdot 9) = 36$

To compare the result with the equation of Ras and Cheng [13] (given in Section 2.3). Their calculation would be:

1. $R_4^1 = 5 + (\lceil \frac{5}{28} \rceil \cdot 2 + \lceil \frac{5}{120} \rceil \cdot 3 + \lceil \frac{5}{140} \rceil \cdot 4) + \lceil \frac{5}{28} \rceil \cdot 5 + \lceil \frac{5}{120} \rceil \cdot 5 + \lceil \frac{5}{140} \rceil \cdot 5 = 29$
2. $R_4^2 = 5 + (\lceil \frac{29}{28} \rceil \cdot 2 + \lceil \frac{29}{120} \rceil \cdot 3 + \lceil \frac{29}{140} \rceil \cdot 4) + \lceil \frac{29}{28} \rceil \cdot 5 + \lceil \frac{29}{120} \rceil \cdot 5 + \lceil \frac{29}{140} \rceil \cdot 5 = 36$
3. $R_4^3 = 5 + (\lceil \frac{36}{28} \rceil \cdot 2 + \lceil \frac{36}{120} \rceil \cdot 3 + \lceil \frac{36}{140} \rceil \cdot 4) + \lceil \frac{36}{28} \rceil \cdot 5 + \lceil \frac{36}{120} \rceil \cdot 5 + \lceil \frac{36}{140} \rceil \cdot 5 = 36$

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

To compute the worst-case response time for $\tau_3$ requires the $\tilde{C}_j^3$ values to be recomputed (as show in Table 5).

**Table 5: $\tilde{C}_j^3$ values for $\tau_3$**

| Task | Period | C | $\tilde{C}_j^3$ | Priority |
|---|---|---|---|---|
| $\tau_1$ | 28 | 2 | 6(2+4) | 1 |
| $\tau_2$ | 120 | 3 | 7(3+4) | 2 |
| $\tau_3$ | 140 | 4 | 4(4+0) | 3 |

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

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

PROOF. We rephrase Equation (2) as below:

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

and simplify:

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

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 $\tau_j$, so we rephrase it again.

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

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

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \tilde{C}_j^i \quad (8)$$

Finally, $\tilde{C}_j^i$ 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.

## 3.3 Tighter analysis

The approach of $\tilde{C}_j^i$ is sometimes too pessimistic because higher priority tasks cannot always abort the lower priority task with the biggest execution time on each release. In this case, the higher priority task aborts the task with second biggest execution time. The multi-set approach [1] from CRPD[1] has a similar property to the AR model.

In the AR model, task $\tau_j$ can abort any task $\tau_k \in hep_{(i)} \bigcap lp_{(j)}$ up to $E_j(R_i)$ times, where $E_j(R_i)$ is the number of release of $\tau_j$ within the response time of $\tau_i$. Equation (9) shows all tasks $k \in hep_{(i)} \bigcap lp_{(j)}$ are found and put the values (abort cost) into the set $M_{i,j}$. $C_k$ is the abort cost for $\tau_k$. $E_j(R_k)E_k(R_i)$ is the number of that $\tau_j$ aborts $\tau_k$ within the response time of $\tau_i$.

$$M_{i,j} = \bigcup_{k \in hep_{(i)} \bigcap lp_{(j)}} \left( \bigcup_{E_j(R_k)E_k(R_i)} C_k \right) \quad (9)$$

The total abort cost $\gamma_{i,j}$ of $\tau_j$ for $\tau_i$ is calculated by adding the $E_j(R_i)$ largest values in $M_{i,j}$. The equation is shown below,

$$\gamma_{i,j} = \sum_{l=1}^{E_j(R_i)} M_{i,j}^l \quad (10)$$

For the response time analysis, the total abort cost ,$\gamma_{i,j}$, is added to the interference from the above higher priority tasks. Note that $\gamma_{i,j}$ is re-computed on each iteration.

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left( \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \gamma_{i,j} \right) \quad (11)$$

In other words, each task $\tau_j$ has a bag $M_{i,j}$ to contain a series of abort cost $C_k$ from tasks $\tau_k \in hep_{(i)} \bigcap lp_{(j)}$, with the number is decided by $E_j(R_k)$. A task-set has a number of bags for each task. Therefore, we call this approach, the *multi-bag*, in this paper.

## 3.4 Example

To illustrate, consider the example task set in Table 6.

| Task | T=D | C |
|---|---|---|
| $\tau_1$ | 25 | 3 |
| $\tau_2$ | 35 | 10 |
| $\tau_3$ | 45 | 3 |

**Table 6: An example task-set. ($\tau_1$ has the highest priority.)**

Using the approach of $\tilde{C}_j^i$, the task-set is not schedulable. The values for $\tilde{C}_1^3, \tilde{C}_2^3, \tilde{C}_3^3$ are 13, 13 and 3. The calculation is shown below,

1. $R_3^0 = 3$
2. $R_3^1 = 3 + (\lceil \frac{3}{25} \rceil \cdot 13 + \lceil \frac{3}{35} \rceil \cdot 13) = 29$
3. $R_3^2 = 3 + (\lceil \frac{29}{25} \rceil \cdot 13 + \lceil \frac{29}{35} \rceil \cdot 13) = 42$

[1]Cache Related Pre-emption Delay anlysis.

4. $R_3^3 = 3 + (\lceil \frac{42}{25} \rceil \cdot 13 + \lceil \frac{42}{35} \rceil \cdot 13) = 55$

.

It ends here because the response time is bigger than the deadline.

Using the multi-bag approach, the task-set is schedulable. The calculation is shown below,

1. $R_1^0 = 3$

1. $R_2^0 = 10$
2. $R_2^1 = 10 + (\lceil \frac{10}{25} \rceil \cdot 3 + \gamma_{2,1}^{l=1}\{10\}) = 23$
3. $R_2^2 = 10 + (\lceil \frac{23}{25} \rceil \cdot 3 + \gamma_{2,1}^{l=1}\{10\}) = 23$

1. $R_3^0 = 3$
2. $R_3^1 = 3 + (\lceil \frac{3}{25} \rceil \cdot 3 + \gamma_{3,1}^{l=1}\{10,3\}) + (\lceil \frac{3}{35} \rceil \cdot 10 + \gamma_{3,2}^{l=1}\{3\}) = 29$
3. $R_3^2 = 3 + (\lceil \frac{29}{25} \rceil \cdot 3 + \gamma_{3,1}^{l=2}\{10,3,3\}) + (\lceil \frac{29}{35} \rceil \cdot 10 + \gamma_{3,2}^{l=1}\{3\}) = 35$
4. $R_3^3 = 3 + (\lceil \frac{35}{25} \rceil \cdot 3 + \gamma_{3,1}^{l=2}\{10,3,3\}) + (\lceil \frac{35}{35} \rceil \cdot 10 + \gamma_{3,2}^{l=1}\{3\}) = 35.$

The second release of $\tau_1$ aborts $\tau_3$ because $\tau_2$ is finished and no job is waiting for execution. The multi-bag approach demonstrates an improvement but it is still only a sufficient test.

## 4. PRIORITY ASSIGNMENT SCHEMES

In the section on related research, Rate Monotonic (RM) and Utilisation 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[2]. An inspection of Equation (3) shows that the minimum execution times (the $\tilde{C}_j^i$ values) are obtained when priority is ordered by execution time. Although this does not necessarily minimise utilisation, it may furnish an effective priority assignment scheme.

For many scheduling problems, Audsley's Algorithm furnishes an optimal priority assignment; i.e. the algorithm can find a schedulable priority ordering if such an ordering exists [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 permitted).

**Table 7: The response time of $\tau_4$ is 23.**

| Task | Period | WCET | $\tilde{C}_j^4$ | Priority |
|---|---|---|---|---|
| $\tau_1$ | 100 | 5 | 9(5+4) | 1 |
| $\tau_2$ | 120 | 4 | 7(4+3) | 2 |
| $\tau_3$ | 140 | 3 | 5(3+2) | 3 |
| $\tau_4$ | 200 | 2 | 2(2+0) | 4 |

In Table 7 $\tau_4$ is the lowest priority task and its response time is 23 (by Equation (4)). After $\tau_2$ and $\tau_3$ (higher priority) swapped their priorities, the response time for $\tau_4$ is changed to 24 as shown in Table 8.

[2]Tasks with the same execution time are ordered by deadline (higher priority to shorter deadline). If they also have the same deadline then the shortest period is used to break the tie.

**Table 8: The response time of $\tau_4$ is 24.**

| Task | Period | C | $\tilde{C}_j^4$ | Priority |
|---|---|---|---|---|
| $\tau_1$ | 100 | 5 | 9(5+4) | 1 |
| $\tau_3$ | 140 | 3 | 7(3+4) | 2 |
| $\tau_2$ | 120 | 4 | 6(4+2) | 3 |
| $\tau_4$ | 200 | 2 | 2(2+0) | 4 |

### 4.1 New Algorithm

The Exhaustive 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, and vice versa. In a later section, the experiments show that UM and EM have similar results. If a new 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 cannot be scheduled then try to find a higher priority task which has less utilisation. 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.

An example of the use of the algorithm is given in Table 9. Again deadline is equal to period; R is response time. Note only $C$ values are given in the table, the necessary $\tilde{C}_j^i$ values are dependent on which task is actually been tested, they must be re-computed for each task.

**Table 9: An example task-set fails in EM ordering.**

| Task | Period | C | U | Priority | R |
|---|---|---|---|---|---|
| $\tau_1$ | 60 | 6 | 0.1 | 1 | 6 |
| $\tau_2$ | 50 | 5 | 0.1 | 2 | 16 |
| $\tau_3$ | 32 | 4 | 0.125 | 3 | 24 |
| $\tau_4$ | 25 | 3 | 0.12 | 4 | 30 (X) |
| $\tau_5$ | 100 | 2 | 0.02 | 5 | |

The task-set is initially ordered by the EM algorithm. The schedulability test begins from the top. $\tau_1$, $\tau_2$ and $\tau_3$ meet their deadlines. A missed deadline occurs at $\tau_4$ so the algorithm searches for a less utilisation task from $\tau_3$ to $\tau_1$. The utilisation of $\tau_2$ is 0.1 which is less than $\tau_4$, $\tau_2$ shifts down below $\tau_4$. The priority of $\tau_3$ shifts up to 2. The priority of $\tau_4$ shifts up to 3. The priority of $\tau_2$ changes to 4.

In Table 10, the task-set has had its priorities changed and is schedulable after the shifting. By the nature of shifting down less utilisation 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-Utilisation Monotonic (EUM) priority assignment scheme.

### 4.2 Time complexity

**Table 10: The task-set is scheduled by EUM algorithm.**

| Task | Period | C | U | Priority | R |
|---|---|---|---|---|---|
| $\tau_1$ | 60 | 6 | 0.1 | 1 | 6 |
| $\tau_3$ | 32 | 4 | 0.125 | 2 | 14 |
| $\tau_4$ | 25 | 3 | 0.12 | 3 | 20 |
| $\tau_2$ | 50 | 5 | 0.1 | 4 | 50 |
| $\tau_5$ | 100 | 2 | 0.02 | 5 | 88 |

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, but 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 $\tau_1$ is the highest priority task and the lowest priority task is $\tau_N$. The algorithm tests each task by the priority ordering from high to low. In the first recursion, it takes N tests from $\tau_1$ to $\tau_N$ and it fails at $\tau_N$. 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. It is the same as the partial sums of the series $1 + 2 + .. + N$. The equation can be represented as below:

$$\sum_{k=1}^{N} k = \frac{N(N+1)}{2} \qquad (12)$$

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 Exhaustive Search (ES) it is a tractable task.

## 5. EXPERIMENTAL EVALUATION

The experiments are separated into two parts. First, the EUM algorithm is compared with other algorithms (DM, UM, EM and ES). Secondly, the multi-bag approach is compared with the $\tilde{C}$ approach using different priority assignments. All experiments used the same parameters but different priority assignments. The parameters are: Deadline is equal to or less than period. All tasks are periodic. A set of N utilisation values $U_i$ was generated by the UUniFast Algorithm [7]. Task periods were generated between 500 and 5000 according to a log-uniform distribution[3]. And the computed value $T_i$ is assigned to $\tau_i$. Task execution times are: $C_i = U_i \cdot T_i$. Utilisation for task-sets are ranged between 10% and 70%. Each utilisation 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, 8 and 15. 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 Utilisation rate and the Y-axis is the Schedulability rate, i.e. the percentage of task sets that were deemed schedulable.

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

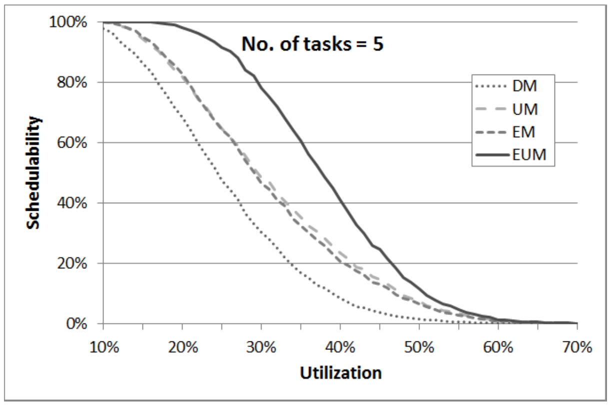

**Figure 4: The number of tasks is 5.**

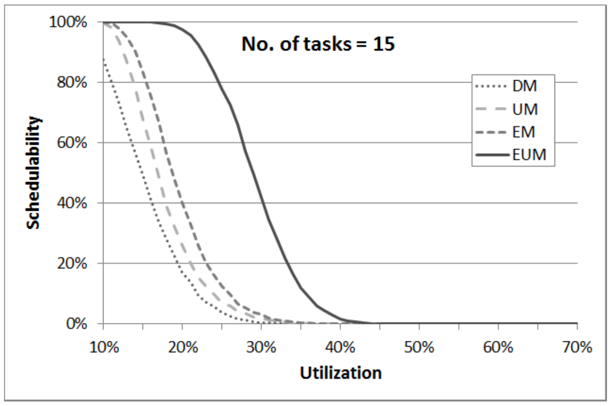

**Figure 5: The number of tasks is 15.**

In Figure 4 the number of tasks is 5. We observe that DM 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 5 the number of tasks is 15. Again DM is the worst; EM is better than UM and EUM is the best. Results for larger value of N are similar (but are 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 6 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). Although not exact, the performance of EUM for $N = 8$ leads to a reasonable conclusion that EUM is an effective and near optimal priority ordering for the AR model.

For a further analysis, Figures 7 and 8 show two examples when deadline is less than period. In Figure 7, the number of tasks is 20 and deadline is 80% of period. The schedulability of using the UM ordering is getting worse at 18% utilisation rate. In Figure 8, the number of tasks is also 20 but deadline is 50% of period. It shows that DM is much better than both UM and EM. EUM is still far better than the other priority

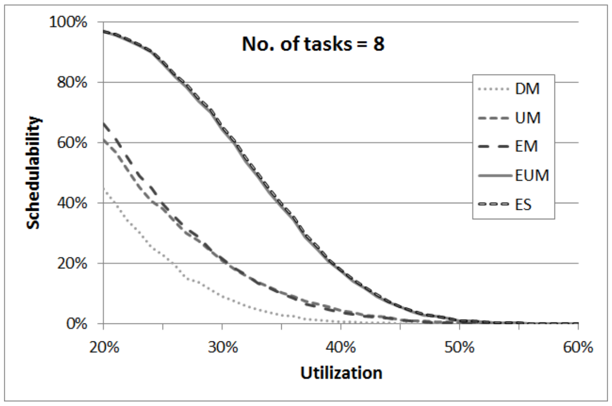

**Figure 6: The number of tasks is 8.**

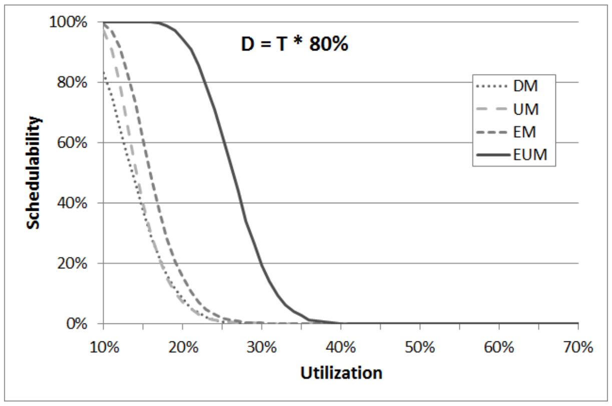

**Figure 7: The number of tasks is 20 with D = T * 80%.**

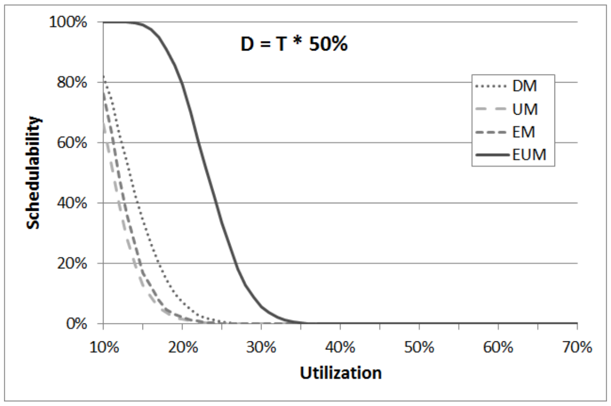

**Figure 8: The number of tasks is 20 with D = T * 50%.**

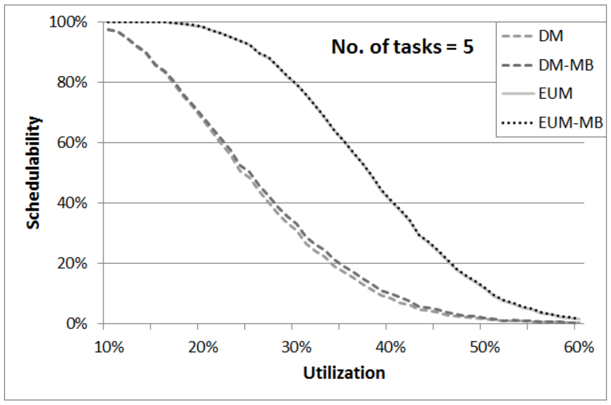

**Figure 9: The number of tasks is 5.**

assignments.

In the second experiment, the multi-bag approach is compared with the $\tilde{C}$ approach using DM, EUM and ES priority assignments. UM and EM are skipped because EUM dominates them.

In Figure 9 the number of tasks is 5. The DM and EUM lines are using the $\tilde{C}$ approach, and the DM-MB and EUM-MB lines are using the multi-bag approach. From the results, DM-MB is improved cleary but EUM-MB is hard to see the difference (actually improved in number).

In Figure 10 the number of tasks is 15. It has similar patten but the improvement of DM-MB is reduced. The EUM and EUM-MB lines are also hard to see the difference (but still improved in number).

In Figure 11 the number of tasks is 8 but only 1000 tasksets. The multi-bag approach takes more computation time than the $\tilde{C}$ approach so the lines are not as smooth as other figures. ES-MB is an optimal priority assignment with the multi-bag approach. DM-MB is far away from ES-MB as expected, and the result shows that the EUM priority assignment is still close to ES, which is optimal.

## 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 executes.

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 and a tighter sufficient test have 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.

## 7. REFERENCES

[1] S. Altmeyer, R. Davis, and C. Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5):499–526, 2012.

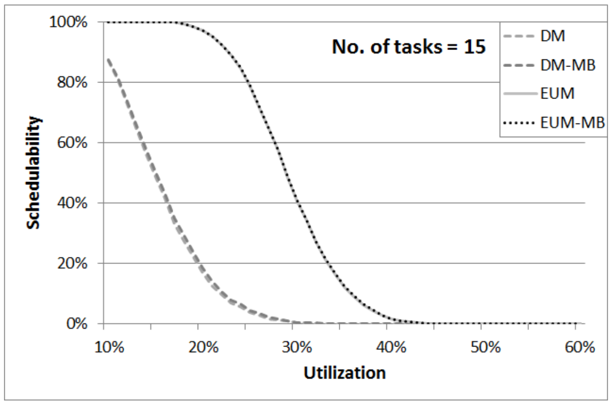

**Figure 10: The number of tasks is 15.**

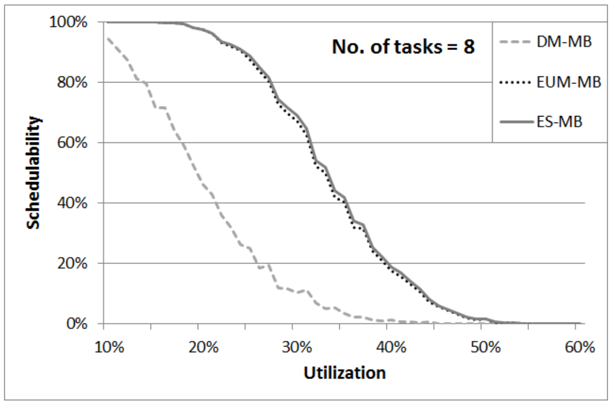

**Figure 11: The number of tasks is 8 (only 1000 task-sets).**

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

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

[4] C. Belwal and A. Cheng. Determining Actual Response Time in P-FRP. In R. Rocha and J. 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. 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. 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. 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] R. Kaiabachev, W. Taha, and A. Zhu. E-FRP with priorities. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, EMSOFT '07, pages 221–230, New York, NY, USA, 2007. ACM.

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

[11] J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin, and J. Vitek. Preemptible atomic regions for real-time Java. In *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, pages 10 pp.–71, 2005.

[12] O. Ulusoy and G. Belford. Real-time transaction scheduling in database systems. *Information Systems*, 18(8):559 – 580, 1993.

[13] J. Ras and A. 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. 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. In *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, 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.

[18] H. Wong and A. Burns. Improved priority assignment for the abort-and-restart (ar) model. In *7th Junior Researcher Workshop on Real-Time Computing (JRWRTC)*, 2013.