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

H.C. Wong and Alan Burns
Real-Time Systems Research Group, Department of Computer Science, University of York, UK.
{hw638, 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. 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 that 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 [2] for real-time systems. It combines the property of atomic execution from Functional Reactive Programming (FRP) [11], and supports priority assignments. To achieve this property of P-FRP, preempted tasks are aborted and the tasks restart as new once the higher priority tasks are completed. We call it the Abort-and-Restart (AR) model in this paper.

Various forms of priority inheritance and priority ceiling protocols have been developed [10] to deal with the problems of shared resources. The AR model does not face the problems because tasks do not access resources directly. But 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

The AR model provides strong correctness guarantees on dealing with shared resources, and it also supports FRP which has been used for the domains of computer animation, computer vision, robotics and control systems [6]. Original FRP cannot be used for real-time systems but P-FRP has rectified this. Preemptible Atomic Regions (PAR) is a new concurrency control abstraction for real-time systems [8]. The basic notions of the AR model and the PAR model are similar. In other words, the AR model has been implemented in a common programming language.

1.2 Contributions

This paper presents an analysis for the AR model. It first confirms that an exact analysis is not tractable as the critical instance cannot, in general, be identified in polynomial time. The second contribution is to develop a new schedulability test for the AR model. A final contribution is to address priority assignment. General priority assignment policies 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 policies and provide an improved (though still not optimal) policy called *Execution-time-toward-Utilisation Monotonic* (EUM).

2 System Model and Related Work

We consider the static priority scheduling of a set of sporadic tasks on a single processor. Each task gives rise to a potentially unbounded sequence of jobs. The notations and formal definitions used are as follows: N is the number of tasks. τ_i, any given task in the system. C_i worst-case execution time (WCET), T_i period, D_i deadline, P_i priority, R_i worst-case response time, U_i utilisation of task τ_i. U is the total utilisation of all the tasks in the task-set. α_i is the maximum abort cost for τ_i (see equation 1). $\tilde{C}_i^n$ is the new value for the WCET of τ_i, the biggest abort cost is picked between τ_i and τ_n (see equation 3). 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

In the Abort-and-Restart (AR) model [9], lower priority tasks are preempted and aborted by releases of higher priority tasks. Once the higher priority tasks have completed, the lower priority tasks are restarted as new.

Table 1. An example task-set.

Task	Period	WCET	Release offset	Priority
τ_1	12	3	3	H
τ_2	15	4	0	L

In Table 1 and 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 time 3. τ_1 finishes its job at time 6 and τ_2 is restarted as a new job so the spent time between 0 and 3 is wasted.

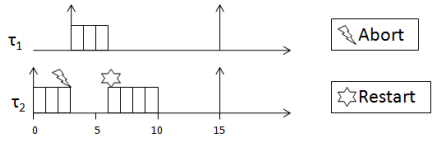

Figure 1. An example task-set.

2.2 Copy-and-Restore Operation

When tasks begin or restart execution, they get a copy (*scratch state*) of the current state from the system [3]. Tasks only modify the copy so no tasks lock the data resource. The copy will be discarded when a higher priority task is released. Once the higher priority tasks have completed execution, the lower priority tasks

are aborted and restarted. When a task has finished, the copy is restored into the system as an atomic action. 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 leading to a blocking term in the analysis. For ease of presentation this term is omitted from the scheduling equations given in this paper.

2.3 Related Research

Ras and Cheng [3] state that the critical instant argument from Liu and Layland [7] may not apply fully to the AR model. Table 1 and Figure 1 illustrate this if τ_1 and τ_2 are released together then $R_2 = 7$. Figure 1 shows clearly that $R_2 \geq 10$.

Ras and Cheng [9] 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 equations:

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

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

In Section 3.2 we will derived an equivalent but more intuitive schedulability test for the AR model. Belwal and Cheng [2] noted that Rate Monotonic (RM) priority assignment is not optimal in the AR model. They introduced an alternative policy called *Utilisation Monotonic* (UM) priority assignment in which a higher priority is assigned to a task which has higher utilisation, and showed that it provides better schedulability 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 task, because the abort cost is added to the response time. For a 2-task task-set, only the highest priority task aborts the lowest priority task as illustrated in Table 1 and Figure 1. For a 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 released once. The highest priority task is τ_1 and the lowest priority task is τ_n. Each task τ_i has $N - i$ choices of lower priority tasks to abort. 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)!. □

As 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, 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 2, there is a three 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.

Table 2. A task-set with a sporadic task.

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

In Figure 2, the response time of τ_3 is 16 when the second job of τ_1 is released with the minimum inter-arrival time, 8.

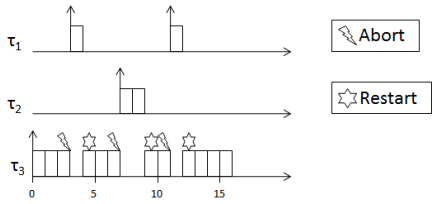

Figure 2. A time chart.

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

For a set of sporadic tasks exact analysis would require all possible release 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, all of which must be checked for the worst-case to be found. For sporadic tasks all possible release times 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. □

A exact schedulability test cannot be tractable if the critical instant cannot be found in polynomial time.

3.2 New Formulation for schedulability tests

In this section we derive a sufficient test that is tractable. Hence we have traded necessity for 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 τ_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 $\tilde{C}_i^n = C_i + C_a$. 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 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 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 τ_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 τ_j, C_j is the original WCET of τ_j and C_k is the biggest 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 $\tilde{C}_j^i$ values will depend on the task under investigation.

In Table 3, there is an example implicit-deadline task-set. The highest priority is 1. The response time of task τ_4 is being computed.

Table 3. An example with new WCET for 4-task task-set.

Task	Period	C	$\tilde{C}_i^4$	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

The $\tilde{C}_i^4$ values are computed by Equation (3). In this example we consider the response time for τ_4 so $i = 4$. For $\tilde{C}_1^4$, j is 1 and C_k is higher than or equal to τ_4 but lower than τ_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 τ_4 but lower than τ_4 so no task is matched, so the result of $\tilde{C}_4^4$ is $5+0 = 5$. After all the $\tilde{C}_i^4$ values have been calculated, we use $\tilde{C}_i^n$ 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 result is $R_4 = 36$, which is the same as that obtained via the equation of Ras and Cheng [9]. In fact, the test derived above, i.e. (4), while more intuitive and more efficiently solved is nevertheless equivalent to that given in [9].

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

Proof. We rephrase (2) as follows:

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

Both $\max_{k=i}^{j-1} C_k$ and $\max_{\forall k \in hep_i \bigcap lp_j} C_k$ pick a bigger WCET task with a priority that is higher or equal to $\tau_(i)$ and lower than τ_j, so we simply to obatin:

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot (C_j + \max_{\forall k \in hep_i \bigcap lp_j} C_k) \tag{6}$$

Equation (3) replaces into (6) as follows:

$$R_i = C_i + \sum_{\forall j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \tilde{C}_j^n \tag{7}$$

□

As (2) was previously proved to be sufficient for the AR model [9] it follows that (4) is similarly sufficient.

Although the equations are equivalent, (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 [5]. It is also in a form that allows the issue of priority assignment to be addressed.

4 Priority assignment schemes

Here, we introduce a priority assignment policy called Execution-time Monotonic (EM) which assigns a higher priority to a task which has a bigger worst case execution time [1]. An inspection of (3) shows that the minimum execution times (the $\tilde{C}_i^n$ values) are obtained when priority is ordered by execution time. Although this does not necessarily minimise utilisation, it may provide an effective priority assignment policy. Audsley's Algorithm provides optimal priority assignments[1] but it does not hold for the AR model since 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).

4.1 New Algorithm

Exhaustive Search (ES) of all possible priority assignments is optimal for any model but it is not tractable. We used it to validate other policies for small values of N. In a later section, the experiments show that UM and EM have similar results, and they do not dominate each other. 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 can not 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.

Table 4 shows that the task-set is not schedulable at τ_4. Again deadline is equal to period; RT is response time. Note only C values are given in the table, the necessary $\tilde{C}_i^n$ values are dependent on which task is actually being tested, they must be re-computed for each task.

Table 4. An example task-set fails in EM ordering.

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	

τ_2 has less utilisation than τ_4 so we shift τ_2 down below τ_4. The new ordering is τ_1,τ_3,τ_4,τ_2 and τ_5, then the task-set is schedulable. We call this policy Execution-time-toward-Utilisation Monotonic (EUM) priority assignment.

[1] With ties broken arbitrarily

4.2 Time complexity

To analyse the complexity of the EUM policy, we count each single task schedulability test required (each test is itself of pseudo-polynomial complexity). In the worst-case, an N-task task-set starts with EM ordering and the task-set is only scheduled by UM ordering which is the completely opposite to EM. It is easy to see that in this case, $2N-1$ schedulability tests are required before the task that starts out at priority N is placed at priority 1, and that a further $2(N-1)-1$ tests are needed before the next task (that started at priority $N-1$) is placed at priority 2. Overall, the number of single task schedulability tests required to transform EM ordering into UM ordering is given by:

$$\sum_{k=1}^{N-1}(2k-1) = N^2 \tag{8}$$

So the complexity of EUM priority assignment is $O(N^2)$ single task schedulability tests. EUM dominates EM and UM because the EUM algorithm starts with EM ordering and ends at UM ordering in the worst-case; however, unlike Exhaustive Search (ES) it is a tractable priority assignment policy.

5 Experimental Evaluation

The experiments compare different priority assignments (DM, UM, EM, ES and EUM) for the AR model. The parameters are: Deadline is equal to period. All tasks are periodic. A set of N utilisation values U_i was generated by the UUniFast Algorithm [4]. Task periods were generated between 500 and 5000 according to a log-uniform distribution[2]. Task execution times are: $C_i = U_i \cdot T_i$ Utilisation for task-sets are ranged between 20% and 60% in steps of 1%. 10000 task-sets were generated for each utilisation level. The number of tasks in each task-set was 8, as this is the maximum that could be handled by Exhaustive Search (ES). Other experiments were also performed for larger task sets (up to 20 tasks) for the heuristic policies, but are not shown due to space limitations.

In Figure 3 the X-axis is Utilisation and the Y-axis is the Schedulability rate, i.e. the percentage of task sets that were deemed schedulable. DM has the worst schedulability, UM and EM are quite similar, while EUM is the best of the heuristics and very close to optimal for task sets of size 8. Indeed it is impossible to distinguish between them. 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). Note that EUM explored a maximum of $N(N-1)/2 = 28$ different priority orderings in this case, whereas ES explored a maximum of $N! = 40320$. 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, at least for relatively small task sets.

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.

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

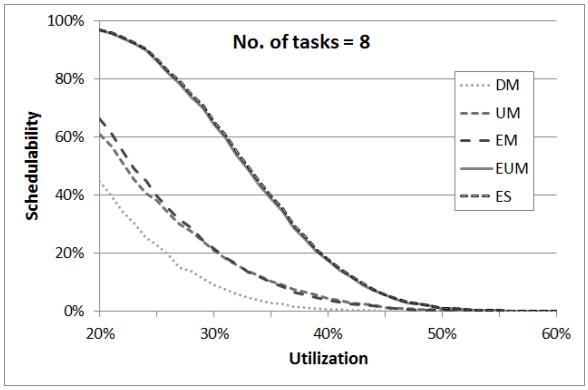

Figure 3. The number of tasks is 8.

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 allows 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 $O(N^2)$ complexity in terms of the number of single task schedulability tests required. 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. On Priority Assignment in Fixed Priority Scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

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

[3] 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.

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

[5] 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.

[6] 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. ACM, 2007.

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

[8] 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.

[9] 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.

[10] 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.

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