On the Compatibility of Exact Schedulability Tests for Global Fixed Priority Pre-emptive Scheduling with Audsley's Optimal Priority Assignment Algorithm

Robert I. Davis[1,2], Marko Bertogna[3], Vincenzo Bonifaci[4]
[1]*Real-Time Systems Research Group, Department of Computer Science, University of York, UK.*
[2]*Inria, Paris-Rocquencourt, France.*
[3]*University of Modena and* Reggio Emilia, *Italy.*
[4]*Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti" (IASI-CNR), Rome, Italy. Email: rob.davis@york.ac.uk, marko.bertogna@unimore.it, vincenzo.bonifaci@iasi.cnr.it*

Abstract

Audsley's Optimal Priority Assignment (OPA) algorithm can be applied to multiprocessor scheduling provided that three conditions hold with respect to the schedulability tests used. In this short paper, we prove that no exact test for global fixed priority pre-emptive scheduling of sporadic tasks can be compatible with Audsley's algorithm, and hence the OPA algorithm cannot be used to obtain an optimal priority assignment for such systems.

1. Introduction

Davis and Burns (2009) proved an important result about the applicability of Audsley's Optimal Priority Assignment (OPA) algorithm (Audsley; 1991, 2001). They showed that three simple Conditions are both sufficient and necessary for Audsley's algorithm to provide optimal priority assignment with respect to a given schedulability test. Davis and Burns (2011a) used the three Conditions to categorise schedulability tests for global fixed priority pre-emptive scheduling on identical multiprocessors according to their compatibility or otherwise with Audsley's OPA algorithm. They showed that the following schedulability tests are compatible with OPA:

- Deadline Analysis (DA test) of Bertogna et al. (2009).
- Improved DA-LC test (Davis and Burns, 2011a) based on the RTA-LC test – see below.
- Response Time test of Andersson and Jonsson (2000).

While the following tests are incompatible:

- Response time analysis (RTA test) of Bertogna and Cirinei (2007).
- Improved response time analysis (RTA-LC test) with limited carry-in of Guan et al. (2009).

In Theorem 5 of their paper, Davis and Burns (2011a) showed via a counter example, that any exact test for global fixed priority pre-emptive scheduling of *periodic* task sets, such as those given by Cucu and Goossens (2006, 2007) is also incompatible with Audsley's OPA algorithm. In this short paper, we extend that result, proving that any exact test for global fixed priority pre-emptive scheduling of *sporadic* task sets, such as those given by Baker and Cirinei (2007), and Bonifaci and Marchetti-Spaccamela (2012), is also incompatible with Audsley's OPA algorithm.

2. System model, terminology and notation

The system comprises a static set of n tasks that are scheduled to execute on m identical processors. Before the tasks can be scheduled, a priority assignment policy is used to assign a unique static priority i, from 1 to n (where n is the lowest priority) to each task.

We assume that each task gives rise to a potentially infinite sequence of jobs. Each job may arrive at any time once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task.

Each task τ_i is characterised by: its relative *deadline* D_i, *worst-case execution time* C_i, and minimum inter-arrival time or *period* T_i. A task's *worst-case response time* R_i is defined as the longest time from a job of the task arriving to it completing execution. We make no assumptions about the relationship between the deadlines and the periods of the tasks (i.e. task deadlines may be *arbitrary*).

At any given time, the scheduler selects the m highest priority tasks with ready jobs to execute on the m processors. (In the case of tasks with arbitrary deadlines, jobs of the same task are executed sequentially in order of arrival). The tasks are assumed to be independent and so a job of one task cannot be blocked from executing by a job of another task other than due to contention for the processors. Further, it is assumed that once a job starts to execute it will not voluntarily suspend itself. Intra-task parallelism is not permitted; hence, at any given time, each job may execute on at most one processor. As a result of pre-emption and subsequent resumption, a job may however migrate from one processor to another. The cost of pre-emption, migration, and the run-time operation of the scheduler is assumed to be subsumed into the worst-case execution time of each task.

In systems using global fixed priority scheduling, it is useful to separate the two concepts of priority assignment and schedulability testing. The priority assignment problem is one of determining the relative priority ordering of a set of tasks. Given a set of tasks with some priority ordering, then the schedulability testing problem involves determining if the tasks are all schedulable with that priority ordering.

A schedulability test S can be classified as follows. Test S is said to be *sufficient* if all of the priority ordered

sets of tasks that it deems schedulable are in fact schedulable. Similarly, test *S* is said to be *necessary* if all of the priority ordered sets of tasks that it deems unschedulable are in fact unschedulable. Finally, test *S* is referred to as *exact* if it is both sufficient and necessary.

The concept of an *optimal priority assignment policy* can be defined with respect to a schedulability test *S*:

Definition: *Optimal priority assignment policy*: A priority assignment policy *P* is referred to as *optimal* with respect to a schedulability test *S* and a given task model, if and only if every set of tasks that is compliant with the task model and is deemed schedulable by test *S* with some priority assignment policy is also deemed schedulable by test *S* using policy *P*.

We note that the above definition is applicable to both sufficient (and not necessary) schedulability tests and exact schedulability tests.

A schedulability test is said to be *OPA-compatible*, if Audsley's algorithm provides an optimal priority assignment with respect to that test.

3. OPA-Compatibility of Exact Schedulability Tests for Sporadic Tasks

Davis and Burns (2009, 2011a) showed that three simple Conditions are both sufficient and necessary for Audsley's algorithm (see Algorithm 1 below) to provide optimal priority assignment with respect to a given schedulability test *S*. In other words to show that test *S* is OPA-compatible. This is a powerful result since it enables the OPA algorithm to be applied in a wide range of scenarios, while lowering the burden of proof of optimality to one of showing compliance with the three Conditions, something that is typically easily proved or disproved.

```
for each priority level k, lowest first {
    for each unassigned task τ {
        if(τ is schedulable according to test S
            at priority k with all other
            unassigned tasks assumed to have
            higher priorities) {
            assign τ to priority k
            break (continue outer loop)
        }
    }
    return unschedulable
}
return schedulable
```

Algorithm 1: OPA Algorithm

The three Conditions are stated below. They refer to properties or attributes of the tasks which make up the task set. Task properties are referred to as *independent* if they have no dependency on the priority assigned to the task. For example in the sporadic task model used in this paper, the worst-case execution time, deadline, and minimum inter-arrival time are all independent properties of a task, while the worst-case response time depends on the task's priority and so is a *dependent* property.

Condition 1: The schedulability of a task τ_k may, according to test *S*, depend on any independent properties of tasks with priorities higher than *k*, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 2: The schedulability of a task τ_k may, according to test *S*, depend on any independent properties of tasks with priorities lower than *k*, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 3: When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test *S*, if it was previously schedulable at the lower priority. (As a corollary, the task being assigned the lower priority cannot become schedulable according to test *S*, if it was previously unschedulable at the higher priority).

Theorem 1: Any exact test for global fixed priority pre-emptive scheduling of sporadic task systems is incompatible with Audsley's OPA algorithm.

Proof: We prove the theorem via a counter example. It suffices to show that for some sporadic task set, the schedulability of the lowest priority task (according to an exact test[1]) depends on the relative priority assignment of the higher priority tasks. This shows that Condition 1 which has been shown to be a necessary condition for OPA-compatibility by Davis and Burns (2011a) does not hold.

The counter example uses a task set that was introduced by Davis and Burns (2011a), and used there to prove that any exact test for strictly periodic tasks scheduled under global fixed priority pre-emptive scheduling is not OPA-compatible.

We assume a dual processor system, and a task set with 4 tasks labelled *A*, *B*, *C*, *D*. The task parameters are as follows (worst-case execution time, deadline, minimum inter-arrival time): task *A* (1,2,3), task *B* (1,2,3), task *C* (2,4,4), task *D* (2,4,4). We prove the theorem by showing that the priority order (*A*,*B*,*C*,*D*) is schedulable, whereas the priority order (*A*,*C*,*B*,*D*) is not.

First we show that priority ordering (*A*,*C*,*B*,*D*) is not schedulable. This is trivially done by examining the schedule assuming that all tasks are released at time *t*=0 and re-released as soon as possible. In this case, task *D* misses its deadline at time *t*=4 as shown in Figure 1 below.

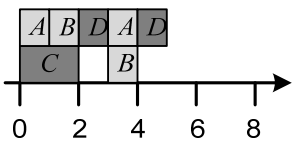

Figure 1 unschedulable priority ordering (*A*,*C*,*B*,*D*)

[1] Note all exact tests give the same result.

Now we consider the priority ordering (*A*,*B*,*C*,*D*). Since there are two processors, the two highest priority tasks, *A* and *B*, are trivially schedulable. Further, task *C* is easily seen to be schedulable, since it is schedulable with task *B* on one processor even if we assume that task A takes the whole of the other processor. (The predictability of global fixed priority pre-emptive scheduling (Ha and Liu, 1994) means that task *C* remains schedulable for execution times of task *A* less than 4). Note that task *C* is also schedulable according to the sufficient RTA test of Bertogna and Cirinei (2007)

To show that task *D* is also schedulable is more difficult; however, since the example is a relatively simple one, we can prove schedulability by an exhaustive method. We note that if there is a deadline miss, then this must necessarily occur within a busy interval. Here, a busy interval is defined as a contiguous time interval during which there is pending workload (i.e. remaining execution of a task) that was released at the start of the interval, or during the interval, but not including workload that is released at the end of the interval. Further, the start of a busy interval corresponds to a time when at least one task is released and there was no pending workload released prior to that time. Hence, by definition, no execution released before the start of a busy interval can possibly interfere with execution within that interval.

We consider all possible patterns of execution that can occur within a busy interval. To aid in the examination of these patterns, we use the notation (*v*,*w*,*x*,*y*) to indicate the release time of the first job of tasks *A*, *B*, *C*, and *D* respectively relative to the start of the busy interval. Further, we use the notation '>*x*' to mean all values greater than *x*. For example (0,1,0,2) means that task *A* and task *C* were released at the start of the interval (at *t*=0), task *B* was released at *t*=1, and task *D* at *t*=2. We systematically cover all possible distinct combinations of release times within a single busy interval. For each combination, we give the length of the busy interval. In all cases, we find that the job(s) of task *D* are schedulable. We note that as there are two processors and the parameters of tasks *A* and *B* are identical, then there is an equivalence between the schedules produced (i.e. tasks *A* and *B* are interchangeable), hence the schedule produced for release times (*v*,*w*,*x*,*y*) is the same as that for (*w*,*v*,*x*,*y*) with the task labels *A* and *B* swapped around; thus task *D* executes at identical times in the two cases. This allows us to eliminate all equivalent combinations (i.e. showing that task *D* is schedulable for (0,1,*x*,*y*) implies that it is schedulable for (1,0,*x*,*y*) etc.).

In the following table we give all of the distinct combinations (not including the equivalent cases mentioned above), with additional notes provided where appropriate. Where we indicate '*Trivial'* we mean that the busy interval ends before the first job of task *D* is actually released.

In determining the different combinations, note we only need consider initial release offsets for tasks *A* and *B* of 0, 1, or 2. This is because their minimum inter-arrival time is 3, hence any larger offset would allow an additional release at *t*=0 which is guaranteed to make the scenario harder to schedule and equates to one of the combinations listed. The combinations are grouped together where the resulting schedule within the busy interval is the same.

Combinations	**Busy interval**
(0,0,0,0), (0,0,0,1), (0,0,1,0)	3
(0,0,0,2), (0,0,2,0)	Extended cases
(0,0,>0,>0),	1 Trivial
(0,0,>2,0),	3
(0,0,0,>2),	3 Trivial
(0,1,0,0),(0,1,0,1)	4
(0,1,1,0)	3
(0,1>0,>0)	1 Trivial
(0,1,0,>1)	2 Trivial
(0,1,>1,0)	2
(0,2,0,0), (0,2,0,1)	3
(0,2,1,0)	3
(0,2,>0,>0)	1 Trivial
(0,2,0,>1)	2 Trivial
(0,2,>1,0)	2
(1,1,0,0)	3
(1,1,1,0), (1,1,2,0)	4
(1,1,0,1), (1,1,0,2)	4
(1,1,>0,>0)	0 Trivial
(1,1,>2,0)	3
(1,1,0,>2)	3 Trivial
(1,2,0,0)	3
(1,2,0,1)	4
(1,2,1,0)	4
(1,2,>0,>0)	0 Trivial
(1,2,0,>1)	2 Trivial
(1,2,>1,0)	2
(2,2,0,0)	3
(2,2,1,0)	4
(2,2,0,1)	4
(2,2,>0,>0)	0 Trivial
(2,2,0,>1)	2 Trivial
(2,2,>1,0)	2

As noted in the table, the extended cases are (0,0,0,2) and (0,0,2,0) in all other cases, the busy period ends after the time given in the table, irrespective of any valid subsequent release of another job of any of the tasks.

We now look at the extended cases. Figure 2 illustrates the schedule for (0,0,0,2) assuming that the second jobs of tasks *A* and *B* are both released at time *t*=3. Here, the busy interval extends to *t*=6. If instead the second job of task *A* or *B* (or both) is released later, then the busy interval would end at *t*=4. In all cases the job of task *D* meets its deadline (at *t*=6).

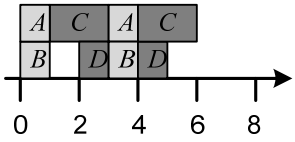

Figure 2: Extended cases: (0,0,0,2)

Similar behaviour can be observed for the schedule corresponding to (0,0,2,0), see Figure 3 below. Assuming that the second jobs of tasks *A* and *B* are both released at time *t*=3, then the first job of task *D* meets its deadline at *t*=4 and the second job easily meets its deadline at *t*=8. If the release of either (or both) of the second jobs of tasks *A* or *B* were postponed, then the busy interval would end at *t*=4. Alternatively, if release of the second job of task *D* occurred any later, the busy interval would end at *t*=5.

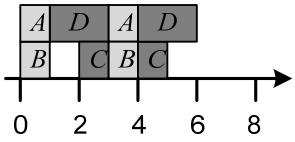

Figure 3: Extended cases: (0,0,2,0)

We have exhaustively covered every combination of release times of the set of sporadic tasks that can produce a distinct pattern or schedule within a busy interval (including the equivalent cases where the labels for tasks *A* and *B* are swapped). In all cases, task *D* was schedulable. Hence with priority ordering (*A*,*B*,*C*,*D*), the task set is schedulable. Since with priority ordering (*A*,*C*,*B*,*D*) task *D* is unschedulable, this shows that the schedulability of task *D* at the lowest priority level depends on the relative priority ordering of the higher priority tasks. This result contradicts Condition 1 which was proven necessary for OPA compatibility by Davis and Burns (2011a) □

Note we also checked schedulability of the example task set with the two priority orderings (*A*,*B*,*C*,*D*), and (*A*,*C*,*B*,*D*) using an implementation of the exact schedulability test given by Bonifaci and Marchetti-Spaccamela (2012) confirming the above results.

Intuitively, it is clear that the example shows that the OPA algorithm cannot be used, since it is impossible to correctly determine the schedulability of task *D*[2] at the lowest priority without first knowing the relative priority order of the other tasks.

4. Discussion

To prove Theorem 1 for the sporadic case, it is necessary to prove the existence of a task set τ with the following two properties:

(i) There is a task *X* in task set τ that is schedulable at the lowest priority, according to an exact test, with the *n*-1 higher priority tasks in some priority order *P*.

(ii) Task *X* in task set τ is not schedulable at the lowest priority, according to an exact test, with the *n*-1 higher priority tasks in some other priority order *Q*.

Theorem 5 of Davis and Burns (2011a) shows that there exist periodic task sets where both these properties hold. Since sporadic behaviour is a generalisation of periodic behaviour, one might assume that Theorem 1 (of this paper) follows directly from Theorem 5 of Davis and Burns (2011a). However, this is not the case.

For global fixed priority scheduling, schedulability in the sporadic case *implies* schedulability in the periodic case, since sporadic task behaviour is a generalisation of periodic task behaviour. However, unlike in the uniprocessor case (Liu and Layland 1973) the converse does not hold; even for synchronous periodic systems. Schedulability in the periodic case does *not imply* schedulability in the sporadic case. (This is demonstrated by an example below). Hence, unschedulability in the sporadic case does *not imply* unschedulability in the periodic case. The converse of course holds: unschedulability in the periodic case implies unschedulability in the sporadic case. This means that neither Theorem 1 in this paper (sporadic case) nor Theorem 5 of Davis and Burns (2011a) (periodic case) can be derived directly from the other. Considering properties (i) and (ii) stated above, if both properties hold for task set τ and task *X* according to an exact test for sporadic systems, then that implies (i) holds for any equivalent periodic system; however, it tells us nothing about whether (ii) holds in that case. Similarly, if both properties hold for task set τ and task *X* according to an exact test for periodic systems, then that implies (ii) holds for the equivalent sporadic system; but tells us nothing about whether (i) holds in that case. Thus independent proofs are needed for both the periodic case (Theorem 5 of Davis and Burns (2011a)) and the sporadic case (Theorem 1 in this paper).

We now provide a simple example which shows that there exist task sets that are schedulable under global fixed priority scheduling with synchronous periodic behaviour, that are not schedulable with sporadic behaviour.

Lauzac et al. (1998) showed that due to the so called *critical instant effect*, under global fixed priority scheduling, a task does not necessarily have its worst-case response time when released simultaneously with all higher priority tasks. This happens because simultaneous release may not be the scenario that results in all processors being occupied by higher priority tasks for the longest possible time during the interval over which the task is active.

From the example given in section 4.6.2 of the survey on multiprocessor scheduling by Davis and Burns (2011b), we now construct task set which is trivially schedulable,

[2] Or task *C*, since they have the same parameters and so are interchangeable.

with synchronous periodic behaviour, but is not schedulable with any priority ordering if it instead has sporadic behaviour. We assume a dual processor system, and a task set with 4 tasks labelled *A*, *B*, *C*, *D*. The task parameters are as follows (worst-case execution time, deadline, minimum inter-arrival time or period): task *A* (2,2,8), task *B* (2,2,8), task *C* (4,6,8), task *D* (4,6,8). As a periodic system with synchronous release of all tasks at time t=0, the task set is schedulable with priority ordering (*A*,*B*,*C*,*D*) as shown in Figure 4. However, as a sporadic system, this task set is not schedulable with any priority ordering. If either task *C* or *D* is given the highest or second highest priority, then either task *A* or *B* (whichever is given priority 3) would be unschedulable following a synchronous release of all tasks. Since tasks *A* and *B* are equivalent, as are tasks *C* and *D*, that only leaves priority ordering (*A*,*B*,*C*,*D*) as a distinct possibility (all other orderings either being unschedulable following synchronous release or equivalent to it). This priority ordering is not however schedulable if the release of task *B* is delayed until time t=2, as shown in Figure 5.

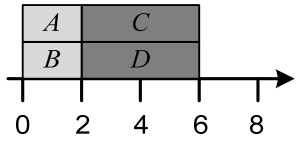

Figure 4: Periodic synchronous release

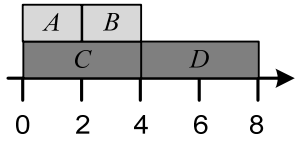

Figure 5: Sporadic asynchronous release

5. Conclusions

In this short paper, which acts as an addendum to the work of Davis and Burns (2011a), we proved in Theorem 1 that any exact test for global fixed priority pre-emptive scheduling of *sporadic* task systems, such as those given By Baker and Cirinei (2007) and Bonifaci and Marchetti-Spaccamela (2012), is incompatible with Audsley's OPA algorithm. This complements the similar result in Theorem 5 of Davis and Burns (2011a), that any exact test for global fixed priority pre-emptive scheduling of *strictly periodic* task systems, such as those given Cucu and Goossens (2006, 2007), is incompatible with Audsley's OPA algorithm.

For these exact tests, currently the only known optimal priority assignment policy involves checking all $n!$ possible priority orderings. The complexity of these optimal priority assignment problems, and thus whether more efficient priority assignment policies exist for them, remains an interesting open question.

Acknowledgements

This work was partially funded by the UK EPSRC MCC project (EP/K011626/1), and the Inria International Chair program.

References

N.C. Audsley (1991) "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Technical Report YCS 164, Dept. Computer Science, University of York, UK.

N.C. Audsley, (2001) "On priority assignment in fixed priority scheduling", Information Processing Letters, 79(1): 39-44.

B. Andersson, J. Jonsson (2000) "Some insights on fixed-priority pre-emptive non-partitioned multiprocessor scheduling". In Proceedings Real-Time Systems Symposium (RTSS) – Work-in-Progress Session.

T. P. Baker, M. Cirinei (2007) "Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks". In Proceedings of the 11th international conference on Principles of distributed systems (OPODIS).

V. Bonifaci, A. Marchetti-Spaccamela (2012) "Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems", Algorithmica, Volume 63, Issue 4, pp 763-780.

L. Cucu, J. Goossens (2006) "Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors", In Proceedings Emerging Technologies and Factory Automation, (ETFA).

L. Cucu, J. Goossens (2007) "Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary Deadline Periodic Systems ", In Proceedings Design Automation and Test in Europe (DATE), pp. 1635-1640.

M. Bertogna, M. Cirinei (2007) "Response Time Analysis for global scheduled symmetric multiprocessor platforms". proceedings Real-Time Systems Symposium (RTSS), pp. 149-158.

M. Bertogna, M. Cirinei, G. Lipari (2009) "Schedulability analysis of global scheduling algorithms on multiprocessor platforms". IEEE Transactions on parallel and distributed systems, 20(4): 553-566.

R.I. Davis, A. Burns (2009) "Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems". In proceedings 30th IEEE Real-Time Systems Symposium (RTSS'09) , pages 398-409.

R.I. Davis and A. Burns (2011a) "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems". Real-Time Systems, Vol. 47, No. 1, pp.1-40.

R.I. Davis and A. Burns (2011b) "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems."*ACM* Computing Surveys, 43, 4, Article 35.

N. Guan, M. Stigge, W.Yi, G. Yu, "New Response Time Bounds for Fixed Priority Multiprocessor Scheduling". In proceedings of the Real-Time Systems Symposium, pp. 388-397, 2009.

R. Ha, J.W-S. Liu, "Validating timing constraints in multiprocessor and distributed real-time systems". In proceedings of the International conference on Distributed Computing Systems, pp. 162–171, 1994.

S. Lauzac, R. Melhem, D. Mosse (1998) "Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor". In proceedings of the Euromicro Workshop on Real-Time Systems, pp. 188–195.

C.L. Liu, J.W. Layland (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM, 20(1): 46-61.