Robust Priority Assignment for Fixed Priority Real-Time Systems

Robert Davis and Alan Burns

Real-Time Systems Research Group

University of York, UK

Outline

- Introduction
  - Background, Previous Research, Motivation
- Robust Priority Assignment (RPA)
  - Definition, Algorithm, Theorems, Examples
- Mixed Systems
  - Two classes of tasks, Robust partial ordering, Improving algorithm efficiency
- Summary of contributions

Background

- **Fixed priority scheduling**
  - Widely used in embedded real-time systems:
    - Electronic Control Units (ECUs) & communications networks in automobiles,
    - Industrial process control,
    - Digital set-top boxes,
    - Medical systems,
    - Mobile phones,
    - Space systems

- Common problem faced by engineers:
  - **How to assign priorities, so that the system will meet its time constraints?**
  - Previous research: Answers this question for well defined (restrictive) system models

Definition

- **Optimal Priority Assignment**

For a given system model, a priority assignment policy P is referred to as **optimal** if there are no systems, compliant with the system model, that are schedulable using another priority assignment policy that are not also schedulable using policy P.

An optimal priority assignment policy can schedule any system that can be scheduled using any other priority assignment

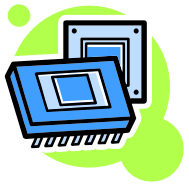
Previous Research

- **Priority Assignment**
  - Rate Monotonic – Serlin 1972 [1], Liu & Layland 1973 [2]
    - Optimal for D=T
  - Deadline Monotonic – Leung & Whitehead 1982 [4]
    - Optimal for D≤T, Not optimal for tasks with offsets
  - Arbitrary Deadlines (D>T) – Lehoczcky et al. 1990 [5]
    - Deadline Monotonic not optimal
  - Optimal Priority Assignment algorithm – Audsley 1991 [6]
    - Optimal for tasks with offsets, D>T etc.
  - Non-pre-emptive scheduling – George et al 1996 [7]
    - Deadline Monotonic not optimal, Audsley's algorithm optimal
  - Blocking according to SRP – Audsley & Bletsas 2006 [10]
    - Audsley's algorithm remains optimal
  - Tasksets with jitter – Zuhily & Burns 2007 [9]
    - "Deadline minus Jitter" Monotonic optimal

Motivation

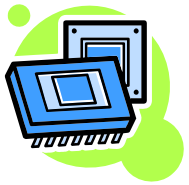
- **Commercial Real-Time Systems**
  - Seldom if ever fully compliant with the system models used in research
  - Tasks subject to all manner of additional interference:
    - Interrupts, occurring in bursts, at ill-defined rates, using more execution time than expected
    - Ill-defined RTOS overheads
    - Tasks overrunning their execution time budgets
    - Ill-defined critical sections with interrupts and task switches disabled, possibly due to the behaviour of the RTOS
    - Cycle stealing by peripheral devices (DMA)
    - Errors causing recovery mechanisms to execute
- **This research**
  - Considers systems subject to **additional interference**
  - Seek to find the **robust priority ordering**, that is able to tolerate the maximum amount of additional interference

System Model



- **Single processor**
  - Static set of *n* tasks τ_i
  - Fixed Priority Scheduling
- **Task scheduling**
  - Pre-emptive
  - Non-pre-emptive
  - Co-operative
- **Task parameters**
  - Periodic or sporadic: minimum inter-arrival time T
  - Deadline D≤T, D>T (arbitrary), or before completion
  - Worst-case execution time C
  - Release jitter, from notional arrival to being ready to execute

System Model



- **Blocking**
  - Access to resources / critical sections according to the Stack Resource Policy (SRP) - Baker 1991 [11]
- **Transactions**
  - Groups of tasks related by offsets
- **No voluntary suspension**

Additional Interference

- **Very general model of additional interference**
- Additional Interference function $E(\alpha, w, i)$
  - α scaling factor – used to model variability
  - w time window – over which interference occurs
  - i priority level – at or below which the interference impinges on task response times
- Require that $E(\alpha, w, i)$ is a monotonic non-decreasing function of its parameters
  - In practice most sources of interference are
    - Greater in longer intervals of time than in shorter ones
    - Affect lower priorities if they also affect higher priorities
    - Guaranteed to be monotonic in α as this is the scaling factor

Definition

- **Robust Priority Assignment**

  (with an additional interference function $E(\alpha, w, i)$)

  For a given system model and additional interference function, a priority assignment policy P is referred to as **robust** if there are no systems, compliant with the system model, that are schedulable and can tolerate additional interference characterized by a scaling factor α using another priority assignment policy Q that are not also schedulable and can tolerate additional interference characterized by the same or larger scaling factor using priority assignment policy P.

  Of all feasible priority assignments, the robust priority assignment tolerates the most additional interference (largest α)

Robust Priority Assignment

- **Robust Priority Assignment Algorithm**
  - Based on Audsley's optimal priority assignment algorithm
  - Applicable to analysable system models (where schedulability can be determined) and the following conditions hold:
    1. Response time of a task may be dependent on:
       the set of higher priority tasks but not their priority order
       the set of lower priority tasks but not their priority order
    2. If the priorities of two tasks are swapped:
       the response time of the task being assigned the higher priority cannot increase.
       the response time of the task being assigned the lower priority cannot decrease.
  - As additional interference $E(\alpha, w, i)$ is monotonically non-decreasing in its parameters, the above conditions also hold when additional interference is considered

Robust Priority Assignment

- **Robust Priority Assignment (RPA) Algorithm**

```
for each priority level i, lowest first
{
    for each unassigned task τ
    {
       binary search for the largest value of α for
       which task τ is schedulable at priority i
    }
    if no tasks are schedulable at priority i
       return unschedulable
    else
       assign the schedulable task that tolerates the
       max α at priority i to priority i
}
return schedulable
```

Robust Priority Assignment

- **Robust Priority Assignment (RPA) Algorithm**
  - Computes
    - Maximum additional interference tolerated by each task at its assigned priority level
    - Maximum additional interference tolerated by the system as a whole
      - = minimum additional interference tolerated by any task
  - Priority ordering generated is:
    - **Optimal** (Theorem 1)
      - Proof by equivalence with Audsley's Optimal Priority Assignment Algorithm
    - **Robust** (Theorem 2)
      - Proof by contradiction…see paper

Robust Priority Assignment

- Example 1: **Non-pre-emptive scheduling**
  - Additional interference from single invocation of an interrupt handler with unknown execution time
  - Additional interference $E(\alpha, w, i) = \alpha$

Task	C	T	D
τ_A	125	450	450
τ_B	125	550	550
τ_C	65	600	600
τ_D	125	1000	1000
τ_E	125	2000	2000

Robust Priority Assignment

- Computed values of α

Priority	Task				
	τ_A	τ_B	τ_C	τ_D	τ_E
5	NS	NS	NS	120	**354**
4	NS	NS	NS	**120**	-
3	10	**110**	74	-	-
2	135	-	**199**	-	-
1	**200**	-	-	-	-

- Robust priority ordering
  - Tolerates infrequent interrupts of up to **110** time units
- Deadline monotonic: neither optimal nor robust
  - Tolerates infrequent interrupts of up to **74** time units

Robust Priority Assignment

- Example 2: **Pre-emptive scheduling, D >T**

Task	C	D	T
τ_A	42	118	100
τ_B	52	154	140

  - Schedulable with priority orderings (τ_A, τ_B) and (τ_B, τ_A) with no additional interference

Robust Priority Assignment

- Case 1: $E(\alpha, w, i) = \alpha \left\lceil \frac{w}{100} \right\rceil$
  - (τ_A, τ_B) tolerates α = (58, **9**)
  - (τ_B, τ_A) tolerates α = (51, **10**) **Robust ordering**
- Case 2: $E(\alpha, w, i) = \alpha \left\lceil \frac{w}{200} \right\rceil$
  - (τ_A, τ_B) tolerates α = (76, **18**) **Robust ordering**
  - (τ_B, τ_A) tolerates α = (96, **15**)
- Case 3: $E(\alpha, w, i) = \alpha \left(\left\lceil \frac{w}{100} \right\rceil K + \left\lceil \frac{w}{200} \right\rceil L \right)$
  - Robust ordering depends on specific values of K and L
  - K=1, L=0: equivalent to Case 1: (τ_B, τ_A) is the **Robust ordering**
  - K=0, L=1: equivalent to Case 2: (τ_A, τ_B) is the **Robust ordering**

Robust Priority Assignment

- Key Result #1 (if somewhat negative)

In general, the Robust priority ordering can only be found if the form of the additional interference function is well defined (only α unknown).

But more to follow about specific system models…

Mixed Systems

- **Mixed systems: two subsets of tasks**
  - "D-J Monotonic tasks"
    - Satisfy the restrictions where "Deadline minus Jitter" monotonic priority ordering is known to be optimal
    - Pre-emptable, D≤T, jitter, resource access according to SRP, no transactions / offsets
  - "Non D-J Monotonic tasks"
    - Don't satisfy the restrictions where "Deadline minus Jitter" monotonic priority ordering is known to be optimal
    - Pre-emptable with D>T, non-pre-emptable, co-operative scheduling with non-pre-emtable final section, transactions, non-zero offsets

Mixed Systems

- Key Result #2

  (Theorems 3, 4 & 5)

  For a mixed system, where a feasible priority ordering exists, there exists a Robust Priority Ordering with the D-J monotonic tasks in "Deadline minus Jitter" monotonic partial order

  - This is the case irrespective of task execution times, and irrespective of the form of the additional interference function*

  *provided only that the additional interference function is monotonic in its parameters

Mixed Systems

- **Use previous result to improve efficiency of**
  - Optimal Priority Assignment Algorithm [Audsley 1991]
  - Robust Priority Assignment Algorithm

- Key point:
  - Of all the D-J monotonic tasks, the one with the largest value of "Deadline minus Jitter" is always the one that can tolerate the most additional interference at a given priority level
  - Therefore, only one D-J monotonic task need be checked at each priority level – the one with the largest value of "Deadline minus Jitter" of all unassigned tasks

Improving algorithm efficiency

- **Efficiency: RPA and Audsley's algorithm**
  - *n* tasks, *m* of which are D-J monotonic.
  - Algorithm complexity:
    - In the worst-case,
      - D-J monotonic tasks assigned first

        $$m(n-m+1)$$
      - *Non D-J monotonic tasks assigned last*

        $$+ (n-m)(n-m+1)/2$$
      - Total

        $$= (n(n+1)-m(m-1))/2$$
      - Reduces to

        $$n(n+1)/2 \quad \text{if } m = 0$$

Improving algorithm efficiency

- Example: 50 tasks:
  - $n(n+1)/2$ = 1275 computations
  - 4 tasks in a transaction, 46 D-J monotonic tasks
    => 240 computations, factor of 5 improvement

Number of non D-J Monotonic Tasks	Number of Computations	Improvement factor
1	99	12.9
2	147	8.7
3	194	6.6
4	**240**	**5.3**
5	285	4.5
10	495	2.6
25	975	1.3

Simple Systems

- Key result #3

  (Theorem 6)

  For systems where all tasks comply with the D-J Monotonic system model, "Deadline minus Jitter" monotonic priority assignment is the Robust Priority Assignment policy

  - This is the case irrespective of task execution times, and irrespective of the form of the additional interference function*

  *provided that the additional interference function is monotonic in its parameters

Simple Systems

- Implications of Theorem 6
  - **For simple commercial real-time systems**
    - using fixed priority pre-emptive scheduling,
    - D≤T, release jitter, resource access according to SRP, no transactions / offsets
    - BUT subject to ill-defined additional interference
      - Interrupts, RTOS overheads, cycle stealing, budget overruns etc.

"Deadline minus Jitter" monotonic priority assignment is the most robust priority ordering to use

Contribution

- Concept of Robust Priority Ordering
  - Tolerates the most additional interference of any priority ordering
- Robust Priority Assignment Algorithm
  - Finds the Robust Priority Ordering for a wide range of system models
- Key results

1. General case: **Robust Priority Ordering depends upon the exact form of the additional interference function**
2. Mixed systems (Some D-J monotonic tasks): **Robust Priority Ordering always has D-J monotonic tasks in "Deadline minus Jitter" monotonic partial order***
3. Simple Systems (All D-J monotonic tasks): **"Deadline minus Jitter" monotonic ordering is the Robust Priority Ordering***

*irrespective of the form of the additional interference function or task execution times

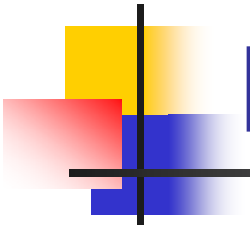
Conclusions

- This research has helped provide a more definitive answers to the engineer's questions:

  Question:*"What priority ordering should I use?"*
  Answer: *"Robust Priority Ordering"*

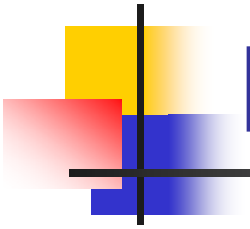
  Question: "How do I find that?"
  Answer: "For simple system models, its Deadline minus Jitter' monotonic priority ordering, otherwise use the Robust Priority Assignment Algorithm."

- Acknowledgements
  - Research partially funded by: EU frescor project



Robust Priority Assignment

Questions?



Robust Priority Assignment

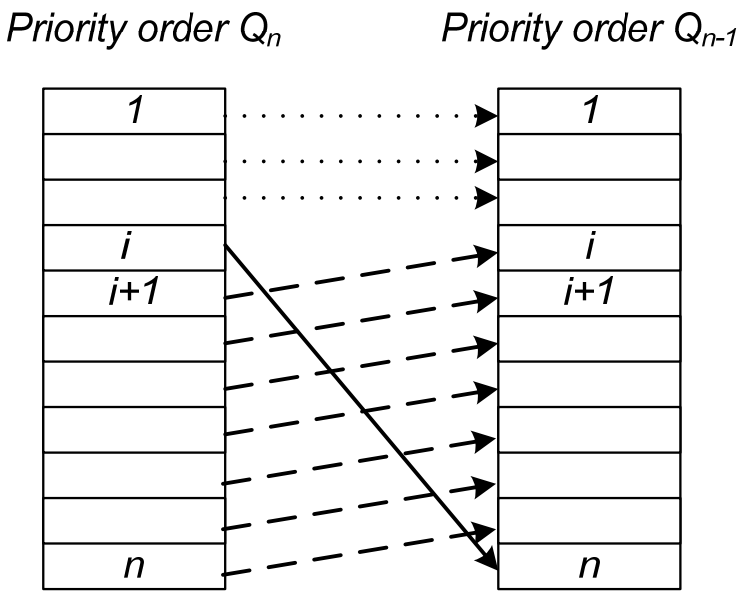
The End

Proof of Robust Ordering

- **Proof by contradiction**
  - Assume an alternative priority ordering Q_n exists that tolerates greater additional interference than priority ordering P (generated by the RPA algorithm)
  - n-1 transformations
    - Q_n into Q_{n-1}, Q_{n-2}... Q_1 = P
  - Each transformation will not decrease the additional interference tolerated => contradiction

Proof of Robust Ordering

- **First step**:
  - Select the task in Q_n that is at priority n in P (The RPA algorithm ordering)
  - Shift that task (from priority i) to priority n
- Q_{n-1} can tolerate at least as much additional interference as Q_n
  - Higher priority than i – same
  - The task at priority i in Q_n can tolerate at least as much additional interference at priority n as the task at priority n (first iteration of the RPA algorithm)
  - Tasks at priorities i+1..n in Q_n shifted up one in priority – so can tolerate at least as much additional interference



Proof of Robust Ordering

- **k^{th} step:**
- At each step k = n down to 1:
  - Select the task in Q_k that is at priority k in P (The RPA algorithm ordering)
  - Shift that task (from priority i) to priority k
- Q_{k-1} can tolerate at least as much additional interference as Q_k
  - Higher priority than i – same
  - Lower priority than k – same
  - Task at priority i in Q_k can tolerate at least as much additional interference at priority k as the task at priority k
  - Tasks at priorities i+1..k in Q_k shifted up one in priority – so can tolerate at least as much additional interference
- n-1 steps to reach robust ordering P
  - No decrease in additional interference tolerated □