Scheduling hard real-time systems: a review

By A. Burns
Software Engineering Journal
May. 1991

Nashville – TN - USA
Summary

- Introduction
  - Concepts
  - Deadlines, Systems, Characteristics..
  - Static and Dynamic Algorithms
  - Preemptive and Non-Preemptive

- Uniprocessor Systems
  - Without blocking
    - Independent periodic process
    - Transient overloads & Period Transformation
    - Independent aperiodic processes
  - With Blocking
    - Priority Inversion & Prevention

- Multiprocessor Systems
  - Aperiodic processes
  - Remote blocking & Transient Overloads
  - Distributed Systems & Communication Delays & RPC

- Conclusions & Major Contributions

- Attachment 1
  - Scheduling Ada Tasks
Introduction
Introduction

- Hard real-time systems
  - Crucial deadlines
    - Not meet the deadlines $\rightarrow$ error
- Application domain
  - Air-traffic control
  - Process control
  - On-board systems
- Not meet the deadline
  - Benign
    - Utility zero but do not damage to the system
  - Damage
    - Potential to be catastrophic
Introduction

- Hard deadline
  - Missed deadline is benign

"Utility" refers to the contribution of this event to the system objectives.
Introduction

- Safety-Critical System
  - Failure becomes more severe as time passes the deadline

![Diagram showing utility and damage over time with start time and deadline](image)

Fig. 2 A safety-critical system
Introduction

- Soft Deadline
  - Missed deadline does not compromise the integrity of the system
    - No damage.

Fig. 3  A soft deadline
Introduction

- Hybrid System
  - Each deadline indicates a level of utility of the process.
  - In the figure: D1: Maximum Utility, D2: Least positive contribution, D3: Damage will occur.

![Diagram showing utility over time with three main points: start time, D1, D2, and D3.](image)

**Fig. 4** A hybrid system
Deadline Characteristics

- **Periodic**
  - Execute on a regular basis
  - Characteristics
    - Period
    - Deadline (often is equal to the period)
    - Execution time (per period)

- **Aperiodic**
  - Activation is a random event, usually triggered by an action external to the environment. **It will be defined as “sporadic” if there is a minimum delay between the events.**
  - Characteristics
    - Usually can be characterized by some kind of random distribution (for example: Poisson Distribution)
    - Execution time
Static and Dynamic Algorithms

- Static
  - Prior knowledge of process characteristics are required
    - Schedules for each process is defined in advance
      - Little runtime overhead
    - Is also said to be “OFFLINE” if ALL scheduling decisions are made before the running of the system.
      - A table is generated with all scheduling decisions

- Dynamic
  - Process schedule is determined in run-time
    - More flexible system that can react to different levels of activity
    - More runtime overhead
    - Appropriated to soft systems
    - Useful to form error recovery procedures for missed deadlines
Preemptive and Non-Preemptive

- Preemptive
  - A process can be suspended so it can be restarted later
  - Usually occurs when a higher priority process becomes runnable.
    (for example, due to the result of a interruption request)

<table>
<thead>
<tr>
<th>Preemptive</th>
<th>RTOS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISR</td>
</tr>
<tr>
<td></td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td>P1</td>
</tr>
</tbody>
</table>

- Non-preemptive
  - The process will not be arbitrary suspended by other process and will continue until resume it’s operation

- Hybrid
  - A scheduler can be preemptive, but allow the process continue executing after be interrupted. *(deferred preemption)*
    - Process can define non-preemptable sections in the code
Problem Space

Uniprocessor & Multiprocessor
Uniprocessor & Multiprocessor

- Uniprocessor
  - One processor system

- Multiprocessor (several different architectures and organizations)
  - Distributed systems
Uniprocessor Systems

Without Blocking
Uniprocessors Systems – No Blocking

- Considerations
  - Single scheduler and one process executing at a time.
  - Processes are independent of each other.
- Independent periodic processes
  - Rate monotonic
    - Process are allocated a priority based on their period
      - The shorter the period, the higher the priority
    - Low runtime overhead
    - Average and worst case execution time can be used to evaluate the schedulability.
- For an arbitrary collection of Random process
  - If schedulability is guaranteed, the utilization: ~88%
  - Worst-case utilization ~69% (periods are relative prime)
  - If process are close to harmonic, the utilization: ~100%
Uniprocessors Systems – No Blocking

- Independent periodic processes
  - Earliest Deadline
    - Processes will be selected according to their deadline.
    - The process closest to its deadline will be the next to be executed.
  - Least slack time
    - Each process will have its slack time calculated, and the one with smallest slack, will be schedule to execute.

  \[ \text{Slack Time} = T - C \]

\[ T = T_{\text{deadline}} - t_0 \]
\[ C = \text{Process Computational Time} \]
\[ T \] “remaining time to the deadline”
\[ t_0 \] : current time
\[ T_{\text{deadline}} \] : deadline of the process

Uniprocessors Systems – No Blocking

Independent periodic processes

- Transient Overloads
  - Situation on which the system is not able to meet all deadlines, this might occur in a situation like:
    - Tasks are scheduled according the worst-case that were too “optimistic” or the hardware failed to perform as anticipated.

- Rate Monotonic x Transient Overloads
  - The overload situation is not adequately address
    - The process priority is based on the “period” and not in how critical the task it is.
      - For example: The task with the longest period might be the critical one, and will be scheduled with the lowest priority.
      - As the hardware might underperform, the lowest priority task will be the most likely one to miss the deadlines.
Uniprocessors Systems – No Blocking

Independent periodic processes

- Solving Transient Overloads
  - Period Transformation applied to Rate Monotonic
    - Priority of the process should also be an accurate statement of its “relative” importance
  - The new process can be obtained:
    - Adding delays into the body of the code (ex: 2 delays)
    - Instructing the runtime system to schedule it as several process (ex: 3)
Uniprocessors Systems – No Blocking

- Independent Aperiodic processes
  - Earliest deadline scheduling
    - Remains optimal to deal with both periodic and aperiodic processes. This is guaranteed if the aperiodic processes are “sporadic”.
  - Dynamic Approach
    - Is required if they are not sporadic
- Rate Monotonic & Aperiodic Processes
  - Periodic process is used to provide service to aperiodic process
    - The aperiodic process will only be handled when the “service process” is executing
      - Problem: aperiodic process may not be available when the service is executing, or it might concentrate the arrivals when the service process is not executing.
    - *Bandwidth-preserving* algorithms are used
      - A “ticket” abstraction is used. Spare resources of time are converted to “tickets”. The aperiodic process arrival will be treated if there are “tickets” available in the service process.
Uniprocessor Systems

With Blocking
Considerations
- Single scheduler and one process executing at a time.
- **Processes interact with each other.**

Problem
- The interaction between process complicates the calculation of the execution time.
- It has been shown that calculate the schedulability in processes that use semaphores is a NP-hard problem.
- The focus is to guarantee the feasibility test, that will ensure the schedulability of system.

Priority Inversion
- A higher priority process can be blocked by lower priority processes an unbounded number of times

>> Priority Inversion Example >>
Uniprocessors Systems – Blocking

- Priority Inversion

LABROSE, J.J.; HALL, I.; uC/OS-III The Real Time Kernel. Micrium Press, Weston, FL.
Uniprocessors Systems – Blocking

- Prevention of Priority Inversion
  - No process is allowed to access the critical section if there is any possibility of a higher priority process being blocked.
  - Feasibility
    - All process must be periodic
    - All maximum execution time must be known
- Problem
  - Enforce the introduction of idle time
    - Worst case utilization is approximately 50%
- Priority inheritance
  - The low priority process that is locking the semaphore required by a higher priority process, will have it’s priority raised. This will guarantee that the process will not be preempted by a “medium” priority task.
    - In the last example: Task L will have its priority raised to the priority of Task H, preventing the execution of Task M.
Uniprocessors Systems – Blocking

- **Ceiling Protocol**
  - Avoid “chains” of blocking
    - Prevent deadlocks and transient blockings
  - The protocol guarantee that a high-priority process can be blocked at MOST ONCE during its execution. (per activation)
    - If a process P1 blocks a higher priority process P2, then no other semaphore that could block P2 is allowed to be locked.
  - Major benefit
    - High-priority tasks can only be blocked once (per activation) by any lower priority task.
  - Cost
    - More processes will experience block.
- **Ceiling Priority**
  - Once a semaphore is locked, the task will immediately receive the priority stored in the semaphore. The semaphore will hold the ceiling level (value of the higher priority task that use this semaphore).
Multiprocessor Systems
## Multiprocessor Systems

- **The Problematic**
  - Uniprocessor algorithms are not directly applicable.
  - Optimal scheduling of multiprocessor is NP-hard.

### Rate Monotonic

P1, P2; Deadline: 50 units Execution\cycle: 25 units

P3; Deadline: 100 units Execution\cycle: 52 units

*P3 will miss its deadline.

Avg. Processor utilization: 76%
Multiprocessor Systems

- Allocation of Periodic Processes
  - Better to statically allocate periodic processes
    - Avoid migration of process
      - Downgrade performance
  - Rate Monotonic Algorithm
    - Can be used to test the schedulability in “each” processor, allocating process which are harmonically related in the same processor.

- Allocation of Aperiodic Processes
  - Statically allocation of process can be used
    - No benefit is gained from the spare capacity in a processor when the other is in transient overload.
  - Dynamic Approach is proposed
    - Periodic process are statically allocated, but aperiodic process can migrate between processors
Multitasking Systems

- Allocation of Aperiodic Processes
  - Dynamic Approach (Described in terms of distributed systems)
    1. Each aperiodic process arrives at some node in the network
    2. The node check if this process can be scheduled together with the current load.
       1. If yes, the process is said to be guaranteed by this node
       2. If no, an alternative node is searched
          1. The search uses a table that check the status (load) of the other processors of the network
    3. Processor is moved to a new node, where the probability of been guaranteed is high. If it end up not been scheduled, it repeats the search process, until it gets executed.

- Node Search
  - Receiver initiated
    - Node asks for processes to migrate to it
  - Sender initiated
    - Node keep record of workloads of other nodes.
Multiprocessor Systems

- Allocation of Aperiodic Processes
  
  Status: Not Guaranteed

Periodic process:

Aperiodic process:

(!) Process migration overhead can be reduced by keeping copies of the processes at nodes more likely to have a lower load.
Multiprocessor Systems

- Remote Blocking
  - Occurs when a process use remote resources that are shared by remote process
  - Minimization Criteria
    - Process should not use remote resources
      - Minimization
        - Allocating the process so that those that share a critical region reside on the same processor. (Reduce interprocessor communication)
    - Remote blocking should only be a function of remote critical sections, not process execution time (due to remote preemption).
      - Minimization
        - Critical section can be made non preemptable
        - Critical section can have its priority raised to be higher than the priority of all other local process.
    - Remote blocking usually can not be eliminated, but effort to minimization should be done.
Multithreaded Systems

- Transient Overloads
  - Already noticed that static allocation can’t deal with transient overloads
  - Each processor has to deal with its transient overload
    - Missed deadlines should correspond to the less important processes.
  - Techniques indicated in the last slides
    - Aperiodic deadlines can be missed, rather than periodic ones
    - Aperiodic deadlines are not missed according to its importance

- Possible Protocol to Avoid Aperiodic Deadlines Missed
  1) Aperiodic and Periodic process are scheduled
     1) A transient overload analysis is evaluated: If the event occurs or is predicted
     2) A set of process that will miss their deadlines are migrated. (May include both aperiodic\periodic process)
  3) This new set is isolated in a processor and priorities are assigned using Rate Monotonic and Period Transformation.
Multiprocessor Systems

- Distributed Systems
  - Static allocation is usually the method considered “more” appropriated.
  - Strategies presented later to minimize the Remote Blocking are widely used.
  - Algorithms for suboptimal allocation or periodic process are used
    - Simulated Annealing (example…)
      - Energy function is based on
        - Schedullability\processor
        - Memory utilization\processor
        - Replicated process (for fault tolerance)
        - Constrained process (process that should be allocated to a single processor)
      - Network traffic.
- Process Migration
  - To reduce the time penalty of process migration, a copy of hard deadline processes are maintained in each processor.
    - This property can also be used to reconfiguration after failure
Multiprocessor Systems

- Distributed Systems
  - Communication Delays
    - Typically is configured as a Local Area Network (LAN)
  - Delays are significant
    - Must be taken into account into the analysis
    - Delays must be bounded
      - It’s a problem because most commercial networks are non deterministic, or use FIFO queuing (or have only a small range of priorities available).
        - This gives rise to priority inversion and prohibits the calculation of useful worst case analysis.
- Remote Procedure Calls
  - A process can be distributed between more than one processor. The process will be held in a specific processor, but the thread control (inner to the process) will be passed to other processor.
    - It’s a process partition. (Essentially a process will be able to call “functions” in a server process providing the service)
  - Remote blocking will inevitably occur, and must be minimized.
Conclusions
Conclusions

- Complexity of general scheduling has been seen to be NP-hard
  - Suboptimal schemes must be used.
- Feasibility of the scheduling can be guaranteed according the following aspects of realistic hard real-time systems:
  1) Guarantee periodic\non-periodic hard real time process on the same processor
  2) Utilization of spare time by non-critical process
  3) Initial static allocation of processes
  4) Process migration due to environment conditions or overload.
- The following problems must be addressed
  1) Guaranteeing hard deadlines constraints to be used. (based on worst-case execution times, and arrival times)
  2) CPU Utilization must be monitored, to deal with spare time.
  3) Generalized scheduler’s must be able to deal with: periods that are not equal to deadline and process with multiple deadlines in a single execution.
  4) Tasks in hard real-time systems are unlikely to be independent
  5) Fault-tolerant programming techniques require consideration.
Major Contributions

- Scheduling Algorithms for different real-time systems are reviewed
  - For both periodic and aperiodic tasks and application domain
- Hazards that affects the scheduling
  - Transient overloads
  - Priority Inversion
  - Remote blocking
- Techniques to eliminate\prevent\minimize hazards
  - Period transformation
  - Priority inheritance
  - Ceiling protocol \ Ceiling priority
  - Process migration \ Process copies
  - Process grouping (by harmonic deadlines)

! As this is a “paper” none of the techniques are discussed in “detail”, but the paper properly summarize the main aspects of each solution.
Thanks.....

Paper Presentation
Scheduling hard real-time systems: a review

Short Bio:

Professor Alan Burns a member of the Department of Computer Science, University of York. He is a member of ARTIST - the EU Centre of Excellence in Real-Time and Embedded Systems. In 2009 Professor Burns was elected a Fellow of the Royal Academy of Engineering. In 2012 he was elected a Fellow of the IEEE.

http://www-users.cs.york.ac.uk/~burns/
Attachment 1
Scheduling Ada Tasks
Scheduling Ada Tasks

- Ada Importance to Real Time Systems
  - Widely used in embedded computer systems and for real time systems
  - Strong built-in support for concurrency
- Concerns about ADA
  - Specification of period activity and deadline information’s
    - Does not directly provide these features.
  - Priority model defined in (Ada LRM 1983)
    - Priorities are static
    - FIFO queues are defined for entries
  - Programmers explore alternatives to avoid these limitations. (like implement their own structures to deal with the priorities)
- Protected records in Ada 9X
  - New revision of the language (1993) is expected to eliminate the problems listed before.
  - A monitor-like passive synchronization primitive is to be added
    - Protected record will encapsulate critical regions