Real-Time Systems

Basic Scheduling Results for Event Driven Systems

Hermann Härtig
Outline

- mostly following Jane Liu, Real-Time Systems
- Principles
- Scheduling
- EDF and LST as dynamic scheduling methods
- Fixed Priority schedulers
- Admission based on Utilization
- Few multi-processor insights (more later)
- Anomalies
Principles

Important properties:

- scheduling decisions are triggered by events (not time instants)
- events are release, completion, blocking, unblocking of jobs
- scheduler calls, interrupts, timers, … may trigger events
- scheduling decisions are on-line
  - scheduling must be simple
- admission is on-line or off-line
- work-conserving schedulers never leave a resource idle intentionally
Restrictions of Time-Driven Systems

some restrictive assumptions of time-driven systems are relaxed:

• fixed inter-release times
  → minimum inter-release times

• fixed number of real-time tasks
  → number of real-time and non real-time tasks can vary

• a priori fairly well known parameters
  → overload, schedule non-RT in the background, …
Principles

At Admission Time:

- select scheduler (may depend on the OS)
- check if feasible schedule exists for the selected scheduler
- assign to jobs a value as a simple selection criteria: some form of priority

Scheduling / Dispatching:

- at event, select highest prioritized job
Principles

How good are schedulers?

• shorter response times
• more task sets possible
• higher utilization of resources

Optimality of schedulers (!):

• A scheduling method X is called *optimal in a class of scheduling methods*, if X produces a feasible schedule whenever there exists a scheduling method Y in this class that produces a feasible schedule.

• X is called *optimal*, if X produces a feasible schedule whenever there exists such a schedule (no matter which method produced it).
Assign priorities at time when jobs are released: “the earlier the deadline the higher the priority”

**Theorem:**
- one processor,
- jobs are preemptable,
- jobs do not contend for passive resources,
- jobs have arbitrary release times, deadlines,
- then: EDF is optimal (i.e. if there is a feasible schedule, there is also one with EDF)
Priority Assignment Following “Criticality”

The more critical a task the higher the priority (period, wcet):

\[ T_1: (2, 0.9) \quad T_2: (5, 2.3) \]

**T2** more critical than **T1**

**T1** misses deadline in Job 1 and 3, unnecessarily ...
EDF Example

**T1:** (2, 0.9)  **T2:** (5, 2.3)
[Least/Minimum] [Slack Time/Laxity] First

• Slack Time = Laxity:
  • (time to deadline - remaining execution time required to reach deadline)

• slack time: D - x - t
  • x  remaining execution time of a job
  • D  absolute deadline
  • t  current time

• priority dynamic per job (see example)
• strict version is optimal
Least Slack Time First

- scheduler checks slacks of all ready jobs and runs the job with the least slack

- two versions:
  - Strict: slacks are computed at all times
    - Each instruction (prohibitively slow)
    - Each timer “tick”
  - Non-strict: slacks are computed only at events (release, completion)
Example: Non-strict LST

Job: (release time, execution time, deadline)

$J_1 : (0, 2, 3.5)$  $J_2 : (1, 3, 5)$

$t = 0: \quad J_1$ released and scheduled

$t = 1: \quad J_2$ released;

$L(J_1) = 3.5 - 1 - 1 = 1.5; \quad L(J_2) = 5 - 3 - 1 = 1 \rightarrow J_2$ scheduled

$t = 3.5: \quad J_1$ deadline miss

EDF schedules both jobs successfully!
Example: Strict LST

Job: (release time, execution time, deadline)

\[ J_1 : (0, 2, 3.5) \quad \text{and} \quad J_2 : (1, 3, 5) \]

\begin{align*}
\text{t = 0:} & \quad J_1 \text{ released and scheduled} \\
\text{t = 1:} & \quad J_2 \text{ released;} \\
& \quad L(J_1) = 3.5 - 1 - 1 = 1.5; \quad L(J_2) = 5 - 3 - 1 = 1 \rightarrow J_2 \text{ scheduled} \\
\text{t = 1.5:} & \quad L(J_1) = 3.5 - 1 - 1.5 = 1; \quad L(J_2) = 5 - 2.5 - 1.5 = 1 \rightarrow J_1, J_2 \text{ are scheduled and executed in parallel (at half speed)} \\
\text{t = 3.5:} & \quad J_1 \text{ completes} \rightarrow J_2 \text{ continued at full speed} \\
\text{t = 5:} & \quad J_2 \text{ completes}
\end{align*}
Latest Release Time (LRT)

- **Rationale:**
  - no need to complete real-time jobs before deadline
  - use time for other activities

- **Idea:**
  - backwards scheduling
    (Deadline <-> Release, turn around precedence graph, EDF)
  - run as late as possible
  - use latest possible release times
  - optimal (analog EDF and strict LST)
EDF Optimality

- Proof: (informal)
  - assume a feasible, non EDF schedule
  - systematically transform it to an EDF schedule (3 steps)

1. \( J_k \)  
2. \( J_k \)  
3. \( J_k \)
EDF and Non-Preemptivity

- Job: (release time, execution time, deadline)

  - $J_1$: (0, 3, 10)
  - $J_2$: (2, 6, 14)
  - $J_3$: (4, 4, 12)

- EDF is not optimal if jobs are not preemptable

  - Infeasible EDF schedule:
    - $J_1$, $J_2$, $J_3$
  
  - Feasible schedule:
    - $J_1$, $J_3$, $J_2$
EDF and Multiple Processors

- Job: (release time, execution time, deadline)

  \[ J_1: (0, 1, 2) \quad J_2: (0, 1, 2) \quad J_3: (0, 5, 5) \]

  \[ J_3 \text{ missed Deadline} \]

  \[ P1 \quad J_1 \quad J_3 \]
  \[ P2 \quad J_2 \]

  \[ \begin{array}{c}
    \text{EDF} \\
    \text{feasible}
  \end{array} \]

- easy for time driven schedulers
- EDF is not optimal for multiprocessor systems
Assumptions for Next Algorithms

- Set of **periodic tasks** with these properties:
  - tasks are independent
  - one processor
  - no aperiodic tasks
  - preemptable, context switch overhead is negligibly small
  - period = minimum inter-release time
    (release times are not fixed but at least period apart)
- Since tasks are independent, tasks can be added (if admitted) and deleted at any time without causing deadline misses.
Priority-Driven Scheduling of Periodic Tasks

- To do:
  - priority assignment (off line / on line)
  - selection of next task (on line)
  - admission (required before new tasks are admitted)

- restrictions (whether they apply or not)
  - dependencies (precedence, sharing)
  - multiple processors
  - aperiodic, sporadic

- achievable resource utilization: \( U = \sum_i \frac{e_i}{P_i} \)
Rate Monotonic Scheduling

- fixed priority:
  - the shorter the period the higher the priority (rate: inverse of period)
  - example: (WCET,P); D=P

![Diagram showing T1: (1,4), T2: (2,5), T3: (5,20) with their respective periods and deadlines.](image)
Deadline Monotonic Scheduling

• fixed priority:
  • the shorter the relative deadline the higher the priority

• example: (e, D, P)

\[
\begin{array}{c}
T_1: (1, 2, 3) \\
T_2: (0.5, 1, 6)
\end{array}
\]

• Conclusion (no proof):
  RM not optimal but DM if \( D \leq P \) for all tasks
Optimality of Fixed Priority Schedulers

T: periodic tasks, independent, preemptable, one CPU

**Deadline Monotonic:**

- relative deadlines $\leq$ periods, in phase
  if there is any feasible fixed priority schedule for T, then Deadline Monotonic is feasible as well

**Rate Monotonic (RMS):**

- relative deadlines $\geq$ periods
  if there is any feasible fixed priority schedule for T, then Rate Monotonic produces a feasible as well
Admission based on Utilization

- A task \((P, e)\) requires \(e/P\) of the capacity of a processor.
- Any scheduler can admit at most up to full capacity:
  - For a task set \(T_1 \ldots T_n\): \(\sum e_i/P_i \leq m\) is a necessary but not sufficient condition for \(m\) processors.
- Can we establish a maximum bound \(X\) such that \(T_1 \ldots T_n\): \(\sum e_i/P_i \leq X\) is sufficient?

Such bounds are called *schedulable utilization* \(SU\).
- \(SU\) depends on the scheduling algorithm.
- the higher the better.
Utilization: RMS / EDF

T₁: (2, 1)  T₂: (5, 2.5)  U=1

EDF

RM

T₂ misses deadline

RMS not optimal in general
Some Schedulable Utilization (SU) Results

- independent tasks, preemptable, relative deadline = period, \( m = 1 \) processor
- \( n \) ... Number of Tasks
- EDF: SU = 1
- RMS: SU = \( n \left( 2^{1/n} - 1 \right) \) \( n \to \infty : \ln(2) \)
- RMS with harmonic periods: SU = 1
- harmonic periods (also called simply periodic):
  for all pairs of tasks \( T_i, T_j \): if \( P_i \leq P_j \) then \( P_j = n_{ij} \times P_i \)
Schedulability Test for Fixed Priority Schedulers

for task sets with $D_i \leq P_i$ (+ some more cases)

Critical Instant Analysis / Time Demand Analysis:

- critical instant for task $T_i$:

  release of jobs such that they have the maximum response time

- 1 CPU, preemptable, independent:

  Critical instant occurs when all tasks are released simultaneously.

  $\Rightarrow$ It is sufficient to check schedulability for the simultaneous release for the longest envolved period.
Non Negligible Context Switch Time

• For Job level fixed priority schedulers:
  • i.e. each job preempts at most one other job

• 2 context switches:
  • release (when it preempts other)
  • completion

• include context switch overhead in WCET:
  • \( WCET_i := WCET_{i\_original} + 2 \text{ context switches} \)
Static and Dynamic (priority)

If no new tasks arrive: static vs. dynamic priorities

- Task static: Task T does not change its priority, i.e. all jobs of T have same fixed priority
- Job static: Jobs do not change their priorities
- Job dynamic: Jobs change their priorities

Careful:

Job static is often called dynamic as well
Earliest Deadline First, priority assignment:

- fixed per job, dynamic at task level:
  - the nearer the absolute deadline of a job at release time
    the higher the priority

\[ T_1: (0.9, 2) \quad T_2: (2.3, 5) \]
EDF and Overload, examples

\[ T_1: (1, 2) \quad T_2: (3, 5) \]

\[ U=1.1 \]

\[ T_1 \text{ misses} \]

\[ T_1 \text{ and } T_2 \text{ misses} \]

No easy way to determine which jobs miss deadline
EDF and Overload, one more example

\[ T_1: (0.8, 2) \quad T_2: (4.0, 5) \quad U=1.2 \]

In fixed priority systems it is possible to predict which tasks are affected by overruns.
Scheduling Anomaly

- increasing priorities:
  - \( i < k \implies Prio(J_i) \text{ higher than } Prio(J_k) \)
- 2 processors, preemptable but not migratable

intuitive approach:
  - check for worst case (a) and best case (b) execution times and be confident ...
Scheduling Anomaly, cont

a)

- P1
  - J₁
  - J₃

- P2
  - J₂
  - J₄

b)

- P1
  - J₁

- P2
  - J₂
  - J₃
  - J₄

Cc)

- P1
  - J₁

- P2
  - J₂
  - J₄
  - J₃
  - J₄
Scheduling Anomaly on One Processor

- Job: (release time, execution time, deadline)
  - $J_1$: (0,3-4,10)
  - $J_2$: (2,6,14)
  - $J_3$: (4,4,12)

- Not preemptable

release time $J_3$  $J_3$ missed Deadline

E1 = 3

E1 = 4
Informal definition:

- Given a set of periodic tasks with known minimal and maximal execution times and a scheduling algorithm.
- A schedule produced by the scheduler when the execution time of each job has its maximum (minimum) value is called a maximum (minimum) schedule.
- An execution is called predictable, if for each actual schedule the start and completion times for each job are bound by these times in the minimum and maximal schedules.
- The execution of every job in a set of independent, preemptable jobs with fixed release times is predictable when scheduled in a priority driven manner on one processor.
Preemptive vs. Non-Preemptive Scheduling

- 2 processors,
- Tasks: notation used below: \( J_i, e_i \)
  - release time of \( J_5 \) is 4, all others 0; (!)
- static priorities, assigned such that:
  \[ i < k \implies \text{Prio}(J_i) \text{ higher than } \text{Prio}(J_k) \]
- Jobs can “migrate”
- precedence graph:
Example, executions

Preemptive execution:

- P1:
  - J1 from 0 to 3
  - J4 from 3 to 6
  - J7 from 6 to 9
  - J6 from 9 to 12
- P2:
  - J2 from 0 to 3
  - J3 from 3 to 6
  - J7 from 6 to 9
  - J5 from 9 to 12
  - J8 from 9 to 12

Non-preemptive execution:

- P1:
  - J1 from 0 to 3
  - J4 from 3 to 6
  - J5 from 6 to 9
  - J6 from 9 to 12
- P2:
  - J2 from 0 to 3
  - J3 from 3 to 6
  - J7 from 6 to 9
  - J8 from 9 to 12

Events:

- J1, 3
- J2, 1
- J3, 2
- J4, 2
- J5, 2
- J6, 4
- J7, 4
- J8, 1
Modified Example: release time of $J_5 = 0$

P1

$J_1, 3$

P2

$J_2, 1$

$J_3, 2$

$J_4, 2$

$J_5, 2$

$J_6, 4$

$J_7, 4$

$J_8, 1$

non-preemptive
Which is better?

• No general answer known!

• If jobs have same release time: preemptive is better (or equal) in a multiprocessor system if cost for preemption is ignored

• more precise: makespan is better (makespan = response time of job that completes last)

• how much better? Coffman and Garey: 2 processors:
  makespan(non-preemptive) $\leq 4/3 \times$ makespan(preemptive)
Multiple Processors

• Static vs dynamic allocation to processors
  • Partitioned: tasks are assigned to processors
  • Static: jobs are assigned to processors once
  • Dynamic: jobs “migrate”
    • example: one run queue served by all processors

• EDF not optimal
  general: “static-job” scheduling not optimal

• There are optimal “dynamic-job” schedulers
Lessons Learned

- Schedulers: static, static and dynamic (RMS, EDF, LST)

- Schedulability Analysis:
  - Critical Instant, Schedulability Utilization

- RMS and EDF are optimal under simplistic assumptions

- Anomalies