Class Module:
Real-Time Systems

Sebastian Fischmeister
sfischme@ecemail.uwaterloo.ca

Department of Electric and Computer Engineering
University of Waterloo

November 7, 2007
Lecture Thesis:

- Understand the different notions of real-time.
- Be able to speak to communicate with real-time systems people.
- Learn a little bit about scheduling
- Learn about proofs in scheduling
Execution

Task
- A larger piece of functionality in your real-time system.
- Simple task vs complex task

Job
- A unit of work that is scheduled.
- A task consists of multiple jobs: GPS reading job, update map job
- For simple tasks, the task usually is the job.
Maximum response time = relative deadline ($D$)

$d_i = r_i + D_j$

Phase and offset are similar
Tardiness

... specifies how long the job is overdue.

\[ \text{response time} > d_{i,\text{rel}} \]

If the job completes at or before the deadline, the tardiness is zero.

Lateness is similar, but also has negative values.
Characteristics

- Missing a deadline is considered a fatal fault.
- Tardiness must always be zero!

Applications & Notes

- Safety-critical hard real-time systems
- Often, the number of allowed misses is specified via probabilities (e.g., $10^{-7}$ or $10^{-9}$).
- Burden is on the developer to proof that this is met.
Soft Real-Time Systems

Characteristics

- Missing a deadline is undesirable.
- Computed values are useful after deadline.

Applications & Notes

- Examples: online transaction systems, games, consumer electronics
- Class of systems: quality of service applications
- Used with different terminology outside the real-time community
**Usefulness Functions**

*Figure:* Examples of usefulness functions.
Is real-time = speed?

As in real-time object recognition, real-time image analysis, real-time data aggregation, real-time strategy gaming?
Processors and Resources

Processors

- They process jobs (exec. code, move data, process queries).
- They have a type and are similar if they can be interchanged.
- Multi-processor systems are declared as $P_1 \ldots P_m$.

Resources

- Jobs require resources (memory, mutex, RS232, EEPROM).
- Resources are acquired via locks ($\rightarrow$ complex tasks).
- Reusable vs. non-reusable resources
- Plentiful resource (no job is every limited by this resource).

Discussion: resource vs. processor: networks?
Real-Time Workloads

- A workload is a set of jobs which have to be processed.
- Workloads can change at run time.
- Assume parameters of workload are known for analysis.
  - a job $J$
  - feasible interval is $(r_i, d_i]$
- A priori knowledge vs. dynamic adjustment
Jitter and Job Releases

Jitter means fluctuations of a recurring event’s occurrence in time (e.g., release jitter \([r^-, r^+]\)).

Job Releases

- **Periodic**: isochronously recurring
- **Sporadic**: follows maximum inter-release time
- **Aperiodic**: arbitrary
Execution Time

... is the time it takes to finish a task without interruption.

Properties

- It can differ between runs.
- The actual execution time of \( J_i \) is not of interest.
- ... the bounds are: \([e_i^-, e_i^+]\).
- Still, finding minimum and maximum execution time is hard.
- Typically, for schedulability analysis \( e_i \) means \( e_i^+ \).
- Using only \( e_i^+ \) causes problem of underutilization.
Periodic Task Model: \( T_i = (p_i, e_i) \)

Basic Properties

- Task is characterized by its period \( p_i \) and execution time \( e_i \).
- The task is released every \( p_i \) time units.
- Periodic vs strictly periodic

Surrounding Concepts

- A task \( T_i \) can have consists of multiple jobs: \( J_{i,1} \ldots J_{i,k} \).
- Phase \( \phi_i \) is release time of first job \( r_{i,1} \).
- The hyperperiod specifies the recurrence of the set of tasks.
  \[ H = \text{lcm}(p_i) \]
  - Often the analysis only has to go up to the hyperperiod.
- Utilization of \( T_i = e_i/p_i \) (if \( d_i = p_i \))
- Often: \( d_i = p_i \) in periodic systems, but not necessarily
Precedence Constraints

...constrain the execution order based on processing requirements.

Job Precedence Constraints

- ... constrain the execution order of jobs inside tasks.
- Partial-order relation $<$ as precedence relation: $J_i < J_j < J_k$.
- $J_i$ is immediate predecessor of $J_j$.
- A job is *ready* when all constraints are satisfied.
- Usually represented by the precedence graphs $G = (\mathcal{J}, <)$. 
Precedence Constraints

Task Precedence Constraints

- ... constrain the release of tasks.
- Arbitrary complex structures and annotations:
  - AND/OR blocks, feasibility interval, conditional block

Figure: Example of task graphs.
Data Dependency

...constrains the execution order based on read/write requirements.

- They are different from processing order requirements.
- Occur in producer/consumer settings.
- Often neglected by implementing shared variables.

We’ll come back to this.
Job Preemption

...means that a task can be suspended at any time.

- **Antonym**: non-preemptable
- Often only *parts* are non-preemptable.
- Preemption mechanism requires *context switch*:
  - Save current processor state
  - Load new processor state
  - Clear pipeline
- Context-switch time is the main run-time overhead.
... provides a partial order of task importance to the system.

- Example: $\text{ABS} >_{\text{crit}} \text{stereo sound}$
- Used by admission control in overload situations
- Used for graceful degradation
Mandatory vs. Optional

Jobs

- The system still works, even if optional jobs are dumped.
- Example: evasion detection is mandatory, evasion route not.

Parts of Jobs

- The system still works, even if code parts are skipped.
- Example: optimizing routes
- → imprecise computation
Scheduler and Schedules

Scheduler

- ... follows a policy and allocates resources to jobs.
- Example: processor to execution, TDMA slot to communication
- Clairvoyant scheduler is always optimal.

Schedule

- ... is an assignment of all jobs to available resources.
- Valid schedule:
  1. Every resource is assigned to at most one job at any time.
  2. Every job is assigned at most one resource at any time.
  3. Job consumption is satisfied (ET or WCET is allocated).
  4. All precedence and resource usage constraints are satisfied.
Schedule Properties

- Feasible schedule: valid + meets job’s timing constraints.
- Scheduleable: set of jobs + execution alg. = feasible schedule
- Makespan: response time of the job that completes last

Evaluation of Scheduling Algorithms

- Does it always find a feasible schedule, if one exists?
- Maximum and average tardiness?
- Average response times, shorter makespan?
- Miss rate, loss rate, invalid rate?
- Objective depends on what is to be scheduled.
Overview of Scheduling Policies
Clock-Driven Scheduling

... uses a stored schedule and makes decisions at specified times.

- Usually clairvoyant scheduler
- Offline generated schedule
- Use timer interrupt online
- Low overhead

![Diagram of schedule with tasks](image)

**Figure:** One schedule (with gap) with $r = 0$, $e_1 = 2$, $e_2 = e_3 = 1$.

More to come later...
Round-Robin Approach

... assigns one time slice to a job at a time.

![Round-robin schedule with $r=0$, $e_1=2$, $e_2=e_3=1$.]

**Figure:** Round-robin schedule with $r=0$, $e_1=2$, $e_2=e_3=1$.

- With $n$ jobs, one job gets a slice every $n$ times slices ($\frac{1}{n}$th).
- Weighted RR assigns slots based on weights.
- Good for job pipelining, bad for precedence in general.

More to come later...
Priority-Driven Approach

... is greedy and decides based on events.

- If possible, never leaves the resource idle (locally optimal).
- Uses a priority queue (=list)
- Static vs dynamic priority assignment
- Many different forms of assigning priorities

![Precedence graph with \( \langle j_i, e_i \rangle \).]

**Figure:** Precedence graph with \( \langle j_i, e_i \rangle \).

**Example:** \( r_5 = 4 \) all other \( r = 0 \), \( J_i >_p J_k \) if \( i < k \).
Priority-Driven Approach Example

Figure: Using preemptive scheduling.

Is this the best we can do?
Priority-Driven Approach Example

Compare: preemptive \((rt_6 = 11)\) vs non-preemptive \((rt_6 = 12)\)
However, in general non-preemptive is not better!

Figure: Using non-preemptive scheduling.
Dynamic vs. Static Dispatching

**Dynamic Dispatching**

- Each job may run on any processor.
- Each job may preempt any job on any processor.
- Each job may resume on any processor (migrate).

**Static Configuration**

- Jobs are partitioned into subsystems.
- Subsystems are bound to processors.
- Reconfiguration introduces **flexibility**.

Example: $J_1$ to $J_4$ on $P_1$ and all remaining on $P_2$.

Discussion of recent results.
Consistency of Observed Timings

Effective Release Time ($r^*$)

- ... fixes the problem of: $r_2 > r_1$ and $J_2 < J_1$.
- No predecessor $\rightarrow r^* = r$
- Otherwise, $r_i^* = \max(r_j^* \cup r_i)$ for all $J_j < J_i$.

Example:

\[ \begin{align*}
J_1(2, 10) & \quad J_3(1, 12) & \quad J_4(4, 9) & \quad J_6(0, 20) \\
J_2(0, 7) & \quad J_5(1, 8) & \quad J_7(6, 21) \\
\end{align*} \]

\[ r_3^* = 2, \quad r_6^* = 4 \ldots \]
Consistency of Observed Timings

Effective Relative Deadline ($D^*$)

- ... fixes the problem of: $D_1 > D_2$ and $J_2 > J_1$.
- No successor $\rightarrow D^* = D$
- Otherwise, $D^*_i = \min(D^*_j \cup D_i)$ for all $J_j > J_i$.

Example:

\[ D^*_4 = 9, \quad D^*_3 = 12 \ldots \]
What are \( r^\star \) and \( D^\star \) good for?

... can ignore precedence constraints in schedulability check.

**Do we need more accuracy?**

E.g., \( r^\star = \max((r_j^\star + e_j^+) \cup r_i) \)?

On a one-processor system with preemptable jobs: no!

From now on \( r \) means \( r^\star \).
Earliest Deadline First (EDF)

Basics

- Priority-driven approach
- Ready queue is sorted: $\text{prio}_i > \text{prio}_j$, iff $d_i < d_j$.
- EDF is optimal with preemptive simple tasks.
- Optimal means EDF produces a feasible schedule, if one exists.

Example:

$J_1, d_1 = 8$  \hspace{1cm}  $J_2, d_2 = 5$

\begin{center}
\begin{tikzpicture}
\draw[->, thick] (0,0) -- (7,0) node[below] {$t$};
\draw[->, thick] (0,-1) -- (0,3) node[left] {$\uparrow$};
\fill[blue!30] (0,0) rectangle (3,1) node[midway, above] {$J_1$};
\fill[green!30] (3,0) rectangle (4.5,1) node[midway, above] {$J_2$};
\fill[blue!30] (4.5,0) rectangle (6,1) node[midway, above] {$J_1...$};
\node[below] at (3,-.5) {$J_1$ preempt!};
\node[below] at (4.5,-.5) {$f_2!$};
\node[below] at (6,-.5) {$J_1$ resumes};
\end{tikzpicture}
\end{center}
Proof of Optimality for EDF

**Theorem**
When preemption is allowed and jobs do not contend for resources, the EDF algorithm can produce a feasible schedule of a set $\mathcal{J}$ of jobs with arbitrary release times and deadlines on a processor if and only if $\mathcal{J}$ has feasible schedules.

Proof intuition: Any feasible schedule of $\mathcal{J}$ can be transformed into an EDF schedule.
Proof of Optimality for EDF

Basis

- Parts of $J_i$ and $J_k$ are scheduled in intervals $I_i$ and $I_k$.
- $d_i > d_k$ but $I_1$ is earlier than $I_2$

Case 1

- $r_k$ later than end of $I_1$
- $\rightarrow$ adheres to EDF, because $J_k$ cannot be scheduled
Proof of Optimality for EDF

Case 2

- $r_k$ is before end of $I_1$
- solution: swap $J_i$ and $J_k$ and fit parts into $I_1$ and $I_2$
Proof of Optimality for EDF

Filling Gaps

- Schedule might still have gaps
- Solution: swap idle time with already released tasks

\[ \rightarrow \text{as long as there is a feasible schedule, this algorithm generates the 'EDF version' of this schedule.} \]
Latest Release Time (LRT)

Basics

- Finishing sooner doesn’t matter, if only deadlines matter.
- Swaps deadlines and release time in sched. decision (=rev. EDF).
- Can leave system idle → not a priority-driven algorithm.
- Optimal similar to EDF.

\[ \begin{align*}
J_1, & \ 3 (0, 6) \\
J_2, & \ 2 (5, 8) \\
J_3, & \ 2 (2, 7)
\end{align*} \]
Least Slack Time First (LST)

Basics
- ... schedules: the smaller the slack, the higher the priority.
- Priority-driven approach
- LST is optimal

Example:

J_i, e (r, D]

J_1, 3 (0, 6]   J_2, 2 (5, 9]   J_3, 2 (2, 6]

slack(J_1) = 3
slack(J_3) = 2

J_1 resumes
Comparison EDF, LRT, LST

Earliest Deadline First

- Widespreadly known
- Robust
- Simple to implement

Latest Release Time

- Good to fit in soft real-time tasks.
- Requires precise execution time analysis!

Least Slack Time First

- Requires knowledge of slack
Counter Examples To Optimality: Non-preemptability

Example

\( \mathcal{J} = \{ J_1 = \langle 0, 3, 10 \rangle, J_2 = \langle 2, 6, 14 \rangle, J_3 = \langle 4, 4, 12 \rangle \} \) with \( J_i = \langle r, e, D \rangle \) using EDF.

Figure: EDF fails with non-preemptive tasks.

Figure: A feasible schedule exists!
Counter Examples To Optimality: Multi-Processor

Example

\[ \mathcal{J} = \{J_1 = \langle 0, 1, 1 \rangle, J_2 = \langle 0, 1, 2 \rangle, J_3 = \langle 0, 5, 5 \rangle \} \] with \( J_i = \langle r, e, D \rangle \) using EDF on a 2-processor machine.

Figure: EDF fails on 2-processor system.

Figure: A feasible schedule exists!
Anomalies of Priority-Driven Systems

Example specification

- Given $J_1$ to $J_4$ with $\text{prio}_i > \text{prio}_j$ where $i < j$.
- System is dynamic (=common priority queue).
- Preemption but no job migration.

<table>
<thead>
<tr>
<th></th>
<th>$r_i$</th>
<th>$d_i$</th>
<th>$[e_i^-, e_i^+]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>0</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>$J_2$</td>
<td>0</td>
<td>10</td>
<td>[2, 6]</td>
</tr>
<tr>
<td>$J_3$</td>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>$J_4$</td>
<td>0</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>
Anomalies of Priority-Driven Systems

\[ e_2 = 6 \text{ (similar with } e_2 = 5) \]

\[ P_1 \]
\[ J_1 \quad J_3 \]

\[ P_2 \]
\[ J_2 \quad J_4 \]

6 \quad 16
Anomalies of Priority-Driven Systems

\[ e_2 = 2 \]

\[ P_1 \quad J_1 \]

\[ P_2 \quad J_2 \quad J_4 \quad J_3 \quad J_4 \]

\[ 2 \quad 2 \quad 20 \]
Anomalies of Priority-Driven Systems

\[ e_2 = 3 \]

\[ P_1 \]

\[ J_1 \]

\[ P_2 \]

\[ J_2 \]
\[ J_4 \]
\[ J_3 \]
\[ J_4 \]

3 21
Anomalies of Priority-Driven Systems

\[ e_2 = 5 \]

- \( P_1 \): 
  - \( J_1 \)
  - \( J_3 \)

- \( P_2 \): 
  - \( J_2 \)
  - \( J_4 \)

Timing: 
- \( 0 \) to \( 5 \)
- \( 5 \) to \( 15 \)
Summary of Anomaly Example

\begin{tabular}{|c|c|}
\hline
$e_2$ & $rt_4$ \\
\hline
2 & 20 \\
3 & 21 \\
4 & 15 \\
5 & 15 \\
6 & 16 \\
\hline
\end{tabular}

Take-away message

- Scheduling anomalies make validation hard.
- Anomalies exist even on single processor systems.

So, is this also true for preemptive uni-processor systems?
Uni-Processor Anomalies Are Non-Existent

... assuming indep. and preemptive jobs with prio.-driven algorithm.

Predictability of schedules

- Actual schedule uses $e$
- Maximal schedule uses only $e_i^+$
- Minimal schedule uses only $e_i^-$
- Schedule is \textit{predictable} if no anomalies in between.

Predictability of jobs

- Use the maximal and minimal schedule for $s^+$ and $s^-$, resp.
- Start-time predictable, if $s_i^- < s_i < s_i^+$
- Completion-time predictable, if $f_i^- < f_i < f_i^+$
- Job is \textit{predictable} if both apply.
Proof for Non-Existence

Theorem
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.

Proof

- Assume: $J_1, J_2, ..., J_{i-1}$ have higher priority than $J_i$ and are predictable. We show by contradiction that $J_i$ (next priority level) is also predictable.

- Suppose that $s_i^- < s_i < s_i^+$ is not true.

- Consider case $s_i > s_i^+$:
  - By $s_i^+$ every higher priority has finished (max. schedule).
  - If $s_i > s_i^+$, then either idle $[s_i^+, s_i]$ or lower priority job runs.
  - Both cannot happen, according to priority-driven algorithm.

Similar argument for case $s_i < s_i^-$.
Proof for Non-Existence (contd.)

Proof contd.

- Suppose $F_i^- < f_i, f_i^+$ is not true.
- Consider case $f_i > f_i^+$:
  - $[s_i, f_i^+]$ is longer than $[s_i^+, f_i^+]$
  - If $f_i > f_i^+$, then (a) gaps or (b) lower priority job runs.
  - Both cannot happen! (priority-driven algorithm)
- Similar argument for case $f_i < f_i^-$ with minimal schedule.
- $\rightarrow J_i$ is start and completion-predictable!

Discussion

- We can ignore variations in execution times (!!)
- Validate using maximal & minimal schedule only
- Independence is hard to achieve in complex systems
- What about conventional 'non-tasks' such as interrupts?
Validation Algorithm

... determines whether all jobs meet their deadlines.

Properties

- Correctness: correct, if no false positives.
- Execution time: some run in constant time.
- Robustness: violating some assumptions still retains correctness.
- Accuracy: accurate, if no false negatives.
Offline vs Online Scheduling

Offline scheduling (vs. online)

- Inflexible for run-time changes
- Near optimal use of resource (esp. multi-resource)
- No run-time overhead for complex scheduling decisions

Online scheduling is only possibility for unpredictable workloads.

Caveat: No optimal online schedule for this case!
No Optimal Online Scheduling

... assuming unpredictable workloads with non-preemptable jobs.

Example

$J_1$ with $e_i = 1$ and $D_1 = 1$ arrives at $t = 0$. What to do?

Figure: Case 1: Immediately?

Figure: Case 1: Later?
Overload

... occurs when the workload cannot be scheduled (clairvoyant).

- Clairvoyant scheduler knows everything (=optimal).
- Overload implies discarding jobs.

Performance of handling overloads

- Value of job $val_i = \begin{cases} 
  e & \text{if } f_i < D_i, \\
  0 & \text{otherwise}. 
\end{cases}$

- Value of schedule $val_s$ is the sum of all jobs in the sequence.
- Optimality $val_o = \text{always produce schedule of maximum possible value.}$
- Competitive factor $c = val_s/val_o$
EDF/LST and Overload

EDF and LST have a competitive factor of 0!

Example

![Diagram showing two tasks J1 and J2 with competitive factor calculation]

Competitive factor

- Optimal value = $e_1$, EDF value = $e_2$
- $c = \frac{e_1}{e_2}$, with infinitely small $e_2$: $c = 0$. 
Theoretic Limit on Competitive Factor

**Theorem**
No online scheduling algorithm can achieve a competitive factor greater than 0.25 when the system is overloaded.

**Setup**
- Adversary offers two jobs: major jobs $J_i$ and assoc. jobs $J_i^a$.
- Jobs have $e = D$ (→ do it now or discard it)
- Assoc. jobs have $e = \epsilon$
- Number of $J_i^a$ for each $J_i$ depend on scheduling decisions.
Theoretic Limit on Competitive Factor

Adversary slightly overload the system: \( r_i = r_{i-1} + e_{i-1} - \epsilon \)

Discard case
At \( r_i \) the scheduler discards \( J_{i-1} \) and starts \( J_i \):

- \( J_{i+1} \) is released with \( r_{i+1} \) computed as \( r_i \).
- Discards until \( J_{\text{max}} \), and total value is \( e_{\text{max}} \).
- Clairvoyant would schedule all \( J^a_i \) instead and \( J_{\text{max}} \).

Completion case
The scheduler decides to complete \( J_i \) and then execute only \( J^a_{i+1} \):

- The adversary stops releasing further \( J^a_{i+1} \).
- Adversary pics \( r_{i+2} \) so that \( J_{i+1} \) could have been executed.
- The total value of the schedule will be \( e_i + \epsilon \).
- Clairvoyant would schedule all \( J_i \); \( \sum_{k=0}^{i} e_k i < \text{max} \).
Theoretic Limit on Competitive Factor

Filling in the values

- $e_0 = 1, \ i > 0 : e_i = c \cdot e_{i-1} - \sum_{k=0}^{i} e_k$
- competitive factor is $\begin{cases} \frac{e_{\text{max}}}{\sum_{k=0}^{\text{max}} e_k} & \text{discard case} \\ \frac{1}{c} & \text{complete } J_1 \text{ and then } J^a \end{cases}$
- Case 2 is always greater if $i \geq 4$.
- For 4, it’s $\frac{1}{4} = 0.25$. 
Theoretic Limit on $c$ (Example)

The scheduler executes $J_2$ to completion.

Schedule value is 8.

Clairvoyant schedule value is 32.
The End?

Hokus Pokus?