End-to-End Scheduling Framework

1. Task allocation: bind tasks to processors
2. Synchronization protocol: enforce precedence constraints
3. Subdeadline assignment
4. Schedulability analysis

Task Allocation

Bin-Packing Heuristics: First-Fit

- Subtasks assigned in arbitrary order
- To allocate a new subtask \( T_{i,j} \)
  - If \( T_{i,j} \) can be added to an existing processor \( P_k \) (\( 1 \leq k \leq m \)) without exceeding its capacity, allocate \( T_{i,j} \) to \( P_k \)
  - Else add a new processor \( P_{k+1} \) and allocate \( T_{i,j} \) to it.

Performance limit of First-Fit

- Number of processors needed: \( m/m_0 \rightarrow 1.7 \) as \( m_0 \rightarrow \infty \)
  - \( m \): number of processors needed under First-Fit
  - \( m_0 \): minimum number of processors needed
- First-Fit can always find a feasible allocation on \( m \) processors if total subtask utilization is no greater than \( m(2^{1/2}-1) \approx 0.414m \)
  - Assuming fixed-priority scheduling, identical processors

Task Allocation to Minimize Communication Cost

- Inter-subtask communication can introduce overhead and delay
  - E.g., Remote method invocation is more expensive and slower than local invocations
- Goal: minimize communication cost subject to processor capacity constraints
- Two steps
  - Partition subtasks into groups
  - Allocate groups to processors

Synchronization Protocols

- Requirements
  - Correct: Enforce precedence constraints
  - Allow accurate schedulability analysis
  - Low worst-case response time
  - Low overhead
  - Reduce jitter
  - Low average response time
**Greedy Protocol**

- Release job $J_{i,j;k}$ as soon as $J_{i,j-1;k}$ is completed
- Subtasks may not be periodic under a greedy protocol
  - Difficult for schedulability analysis
  - Higher-priority tasks arrive early → high worst-case response time for lower-priority tasks
  - Jitter can accumulate over multiple hops

**Critique on Greedy Protocol**

- Correctness
- Allow schedulability analysis
- Worst-case response time
- Overhead
- Jitter
- Low average response time

**Phase-Modification Protocol (PMP)**

- Idea: Enforce periodic release based on worst-case response time
- Every job $J_{i,j;k}$ is released at time $s_i + (k - 1)p_i + \sum_{j=1}^{k-1} W_{i,j}$
  - $s_i$: start time of job $J_{i,1}$
  - $w_i$: worst case response time of $T_i$

**Critique on PMP**

- Incorrect if tasks have release jitter or overrun
- Allow schedulability analysis
- Low worst-case response time
- Overhead:
  - No explicit synchronization
  - Depend on global clock synchronization
  - Low jitter
- High average response time

**Modified PMP**

- Same as PMP except
  - A subtask cannot be released unless its predecessor has been completed
- Assumptions
  - Require upper bound on the response time of all subtasks
  - Require global clock

**Critique on MPMP**

- Correct
- Allow accurate schedulability analysis
- Low worst-case response time
- Overhead:
  - require explicit synchronization
  - Does not require global clock sync
  - Low jitter
- High average response time
Release Guard

- If processor never idles since last release time $r_{ik+1;j-1}$:
  - release $J_{ik+1;j}$ either when it receives a sync message from $J_{ik;j}$ or at time $r_{ik+1;j-1} + p_i$, whichever is later.
- Else:
  - release $J_{ik+1;j}$ when receiving a sync message or when processor becomes idles, whichever is later.
  - Improve average response time without affecting schedulability at the cost of jitter.

RG Assumptions

- Assumptions
  - Does not require upper bound on the response time of all subtasks.
  - Does not require global clock synchronization.
  - Work best for loosely coupled system.

Critique on Release Guard

- Correct
- Allow accurate schedulability analysis
- Low worst-case response time
- Overhead
  - require explicit synchronization
  - Does not require global clock synchronization
  - Low jitter (if rule 2 is not used)
- Improved average response time (if rule 2 is used)

Score Board: Sync protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Correct</th>
<th>Sched</th>
<th>WCRT</th>
<th>Avg RT</th>
<th>Global Inf.</th>
<th>Jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>Y</td>
<td>N</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>PMP</td>
<td>N</td>
<td>Y</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>MFMP</td>
<td>Y</td>
<td>Y</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>RG</td>
<td>Y</td>
<td>Y</td>
<td>L</td>
<td>M/H</td>
<td>L</td>
<td>H/L</td>
</tr>
</tbody>
</table>

- Use MPMP or RG if
  - Information about all tasks are available a priori
  - System has global clock sync
  - Otherwise only RG can be used

Subdeadline Assignment Heuristics

- Notations
  - (Relative) deadline $d_i$ of task $T_i$.
  - (Relative) subdeadline $d_{ij}$ of subtask $T_{ij} (1 \leq j \leq n(j))$.
  - Slack of subtask $T_{ij}$: $s_{ij} = s - d_{ij}$.
- Ultimate Deadline (UD): $d_{ij} = d_i$.
- Effective Deadline (ED): $d_{ij} = d_i - \sum_{k=j}^{n} d_{ik} - \sum_{k=j}^{n} s_k$.
  - Assign all slack to 1st subtask.

Subdeadline Assignment

- Subdeadline $\Rightarrow$ priorities under EDF & DM $\Rightarrow$ response times.
- Optimal subdeadline assignment is NP-hard
  - Offline: heuristic search algorithms
  - Online: simpler heuristics.
More Heuristics

- Proportional Deadline (PD):
  \[ d_y = d_j \frac{e_{ij}}{\sum_{k} e_{ik}} \]
  - Assign slack proportionally to execution time
- Normalized Proportional Deadline
  \[ d_y = d_j \frac{e_{ij} u(V_{ij})}{\sum_{k} (e_{ik} u(V_{ik}))} \]
  - Assign more slack to subtasks on busier processors

Scheduling

- Single processor
  - Periodic tasks
    - Fixed priority vs. dynamic priority
    - Deadline vs. period
    - Resource contention
  - Aperiodic + periodic tasks
  - Distributed systems

Scheduling Aperiodic Tasks

- Hybrid task set: periodic tasks + aperiodic tasks
- Problem: Arrival time is unknown
- Sporadic task with a hard deadline
  - Inter-arrival time must be lower bounded
  - Schedulability analysis: treated as a periodic task with period = minimum inter-arrival time
- Aperiodic task with a soft deadline
  - Possibly unbounded inter-arrival time
  - Goals:
    - Maintain hard guarantees on periodic tasks
    - Reduce response time of aperiodic tasks

Background Scheduling

- Treat aperiodic tasks as lowest-priority tasks
- Advantages
  - Simple
  - Aperiodic tasks have no impact on the schedulability of periodic tasks
- Disadvantage
  - Aperiodic tasks have very long response times when the utilization of periodic tasks is high
  - Acceptable only if
    - System is not busy
    - Aperiodic tasks can tolerate long delays

Polling Server

- Polling server (PS): a periodic task used to serve aperiodic requests
  - Period: \( p_1 \)
  - Capacity: \( c_1 \)
- Rules
  - Released periodically with period \( p_1 \)
  - Serves any pending aperiodic requests
  - Suspends itself if
    - it has used up its capacity, or
    - no aperiodic request is pending
  - Server capacity is replenished to \( c_1 \) in the next period

Schedulability

- The aperiodic requests have the same impact on periodic tasks as a periodic task.
  - \( n \) tasks with \( m \) PS: \( U_p + U_a \leq U_b(n+m) \)
  - Can have multiple PS (with different periods) for different aperiodic requests
- Disadvantage: If an aperiodic request "misses" the execution of PS, it has to wait till the next period \( \rightarrow \) long response time.
Deferrable Server (DS)
- Unlike PS, DS preserves unused capacity until the end of the current period
- Better response to aperiodic requests
- However, DS’ impact on periodic tasks is different from an periodic task

Utilization Bound with DS
- Under RMS \( U_b = U_s + n \left( \frac{U_s + 2}{2U_s + 1} \right)^{1/n} - 1 \)
- As \( n \to \infty \) \( U_b = U_s + \ln \left( \frac{U_s + 2}{2U_s + 1} \right) \)
- When \( U_s = 0.186 \), min \( U_b = 0.652 \)
- System is schedulable if \( U_s \leq \ln \left( \frac{U_s + 2}{2U_s + 1} \right) \)

Pointers
- Class hand-out
- Rate Monotonic
- EDF
- General
  - Real-Time Systems, Jane Liu.