Critique #4

- Due on 4/20 (Tuesday)

Real-Time Scheduling
Multiprocessor and Distributed Systems

Chenyang Lu
Multiprocessor System

- Tight coupling among processors.
- Communicate through shared memory and on-board bus.
- Scheduled by a common scheduler/OS.
  - Global scheduling
  - Partitioned scheduling
- States of all processors available to each other.
Distributed System

- Loose coupling among processors
  - Each processor has its own scheduler
  - Costly to acquire states of other processors

- Wide range of systems
  - Processor boards mounted on a VME bus
  - Automobile: 100s processors connected through Control Area Networks (CANs)
  - Air traffic control system on a wide area network
End-to-End Task Model

- An (end-to-end) task is composed of multiple subtasks running on multiple processors
  - Message, event, remote method invocation

- Task = a chain/tree/graph of subtasks
  - $T_i = \{T_{i,1}, T_{i,2}, \ldots, T_{i,n(i)}\}$
  - $n(i)$: the number of subtasks of $T_i$
  - Precedence constraint: Job $J_{i,j}$ cannot be released until $J_{i,j-1}$ finishes.

```
T_1 ─ T_{11} ─ T_{12} ─ T_{13} ─ T_2 ─ T_3
P_1  ─ P_2  ─ P_3
```

Chenyang Lu
An End-to-End Task on Event Service

- Dependency implemented through events
- Event Channel (EC) dispatches events according to their priorities.
- Gateway forwards events between processors.

Chenyang Lu
A task is subject to an end-to-end deadline.

Does not care about the response time of a subtask.

How to guarantee end-to-end deadlines in distributed systems?
End-to-End Scheduling

1. Task allocation
2. Synchronization protocol
3. Subdeadline assignment
4. Schedulability analysis
Task Allocation

- Map tasks to processors

- Strategies
  - Offline → static allocation
  - Online
    - Allocate a task when it arrives
    - Re-allocate (migrate) a task after it starts

- NP-hard → heuristics needed
Bin Packing

- Pack subtasks to bins (processors) with limited capacity
  - Size of a subtask $T_{i,j}$: $u_{i,j} = C_{i,j}/P_i$
  - Capacity of each bin: utilization bound

- Goal: minimize #bins subject to capacity constraints
  - Ignore communication cost
  - Assume every subtask is periodic
Bin Packing: 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 $S_m$ ($1 \leq m \leq k$) without exceeding its capacity
    - allocate $T_{i,j} \rightarrow S_m$
  - else
    - add a new processor $S_{k+1}$ and allocate $T_{i,j} \rightarrow S_{k+1}$. 
First-Fit Performance

- #Processor needed: $\frac{m}{m_{\text{min}}} \to 1.7$ as $m_{\text{min}} \to \infty$
  - $m$: #processor needed under First-Fit
  - $m_0$: minimum #processor needed

- First-Fit can always find a feasible allocation on $m$ processors if total subtask utilization $\leq m(2^{1/2}-1) = 0.414m$
  - Assuming identical processors
Minimize Communication Cost

- Inter-subtask communication introduces overhead & delay

- Minimize communication subject to processor capacity constraints
  - Partition subtasks into groups
  - Allocate groups to processors
End-to-End Scheduling

1. Task allocation
2. **Synchronization protocol**
3. Subdeadline assignment
4. Schedulability analysis
Synchronization Requirements

- Allow schedulability analysis
- Bounded worst-case response time
- Low overhead
- Low 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 to analyze schedulability
  - High-priority tasks arrive early $\rightarrow$ long worst-case response time for low-priority tasks
  - Jitter can accumulate over multiple hops
Greedy Protocol Example

Phase of $T_3$ misses deadline
Critique of Greedy Protocol

- Low overhead
- Low average response time
- High jitter
- Difficult to analyze schedulability
- Long worst-case response time
Modified Phase-Modification Protocol (MPMP)

- Enforce periodic release based on the worst-case response times of preceding subtasks.

- Every job $J_{i,j;k}$ is released at time

$$\phi_i + (k-1)P_i + \sum_{l=1}^{j-1} R_{i,l}$$

- $R_{i,l}$: worst case response time of $T_{il}$

- Require upper bounds on the response times of all subtasks.

- In addition, a subtask cannot be released until it receives a sync message its predecessor.
On P1

On P2

Release delayed

Phase of $T_3$
Properties of MPMP

- Enable schedulability analysis
- Bounded worst-case response time
- Low jitter
- Does not require global clock synchronization

- Require worst-case response times of all subtasks
- Long average response time
Release Guard

- if CPU never idles since releasing $J_{i,j;k}$, release $J_{i,j;k+1}$ when
  - it receives a sync message from $J_{i,j;k}$, or
  - at time $r_{i,j;k-1} + P_i$
whichever is later  periodic release based on local knowledge

- else, release $J_{i,j;k+1}$ when
  - it receives a sync message from $J_{i,j;k}$, or
  - processor becomes idle idle-resetting → improve response time
whichever is later

- Improve average response time without affecting schedulability
On P1

On P2

\[ g_{2,2} = 0 \]

\[ g_{2,2} = 4 + 6 = 10 \]

\[ g_{2,2} = 9 \]

\[ g_{2,2} = 9 + 6 = 15 \]

Phase of \( T_3 \)
Non-Assumptions

✓ Do **not** require worst-case response times of all subtasks
✓ Do **not** require global clock synchronization
✓ Work best for loosely coupled system!
Properties of Release Guard

- Enable schedulability analysis
- Bounded worst-case response time
- Does not require global knowledge or clock synchronization
- Low jitter (if idle-resetting is not used)
- Low average response time (if idle-resetting is used)
RG: Middleware Implementation

If current time is earlier than the release guard $T_g$

- EC I/O thread buffers the event in the release guard queue;
- At $T_g$, RG thread removes the buffered event from queue and inserts it into a dispatching lane

---

**EC I/O Thread**

**RG Thread**

**Dispatching Thread**

---

### Score Board: Sync Protocols

<table>
<thead>
<tr>
<th>Schedulability</th>
<th>WCRT</th>
<th>ART</th>
<th>Global State</th>
<th>Jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Greedy</strong></td>
<td>Hard</td>
<td>H</td>
<td>L</td>
<td>N</td>
</tr>
<tr>
<td><strong>MPMP</strong></td>
<td>Y</td>
<td>L</td>
<td>H</td>
<td>Y</td>
</tr>
<tr>
<td><strong>RG</strong></td>
<td>Y</td>
<td>L</td>
<td>M/H</td>
<td>N</td>
</tr>
</tbody>
</table>

- if information about all tasks are available *a priori*
  - use RG or MPMP
- else
  - use RG
End-to-End Scheduling

1. Task allocation
2. Synchronization protocol
3. **Subdeadline assignment**
4. Schedulability analysis
Subdeadline Assignment

- Subdeadline $\rightarrow$ priority $\rightarrow$ response time

- Optimal subdeadline assignment is NP-hard
  - Offline: heuristic search
  - Online: simpler heuristics
Subdeadline Assignment

- **Notations**
  - Relative deadline $D_i$ of task $T_i$
  - Relative subdeadline $D_{ij}$ of subtask $T_{ij}$ ($1 \leq j \leq n(i)$)

- **Ultimate Deadline (UD):** $D_{ij} = D_i$
  - But some subtasks must finish earlier than the end-to-end deadline!
More Heuristics

• Proportional Deadline (PD):

\[ D_{ij} = D_i \frac{C_{ij}}{\sum_{k=1}^{n(i)} C_{ik}} \]

– Assign slack proportionally to execution time

• Normalized Proportional Deadline

\[ D_{ij} = D_i \frac{C_{ij} U(V_{i,j})}{\sum_{k=1}^{n(i)} (C_{ik} U(V_{i,k}))} \]

– Assign more slack to subtasks on busier processors
End-to-End Scheduling

1. Task allocation
2. Synchronization protocol
3. Subdeadline assignment
4. Schedulability analysis
End-to-End Response Time Analysis

With release guard or modified phase modification protocol, the worst-case end-to-end response time of a task $T_i$ is the sum of the worst-case response times of its subtasks $T_{ik} (1 \leq k \leq n_i)$ where $n_i$ is the number of subtasks of $T_i$.

$$R_i = \sum_{k=1}^{n_i} R_{ik}$$