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
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 $J_{i,j-1}$ has been completed.
Event Service

- TAO: Open-source Real-Time CORBA middleware.
- Event Channel (EC) dispatches events according to their priorities.
- Gateway forwards events between processors.
End-to-End Deadline

- 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 $P_l$ ($1 \leq l \leq k$) without exceeding its capacity, allocate $T_{i,j} \rightarrow P_l$
  - Else, add a new processor $P_{k+1}$ and allocate $T_{i,j} \rightarrow P_{k+1}$.
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 cost 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

- 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 lower-priority tasks
  - Jitter can accumulate over multiple hops
Greedy Protocol Example

On P1

On P2

On P3

Phase of T_3

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
Phase-Modification Protocol (PMP)

- Enforce periodic release based on the worst-case response times of preceding subtasks.
- Every job $J_{i,j;k}$ is released at time

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

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

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

- Modified PMP (MPMP): Same as PMP except a subtask cannot be released unless its predecessor has been completed.
MPMP Illustrated

T₁

2 4 6 8 10 12

T₂,₁

2 4 6 8 10 12

T₂,₂

2 4 6 8 10 12

T₃

2 4 6 8 10 12

Phase of T₃

Synch signal delayed

Chenyang Lu
Critique of MPMP

- Enable schedulability analysis
- Bounded worst-case response time
- Low jitter
- Does not require global clock synchronization
  - Indicate “ahead time” in sync message
- Require upper bounds on the 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

d else, release $J_{i;j;k+1}$ when

- receiving a sync message from $J_{i;j;k}$, or
- when processor becomes idle,

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 $T_3$

Idle time detected
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 clock synchronization
- Low jitter (if idle rule is not used)
- Improved average response time (if idle rule 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

---


Chenyang Lu
### Score Board: Sync Protocols

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