Critique #4

- Due on 4/19 (Tuesday)

Middleware Support for Aperiodic Tasks in Distributed Real-Time Systems

Chenyang Lu
Offline Approaches to Aperiodic Tasks

- Periodic tasks + aperiodic tasks

- Problem: arrival times of aperiodic tasks are 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 → can be very pessimistic.

- Aperiodic task with a soft deadline
  - Possibly unbounded inter-arrival time
  - Maintain hard guarantees on periodic tasks
  - Reduce response time of aperiodic tasks
How about these aperiodic tasks?

- Aperiodic tasks in many distributed real-time applications
  - have hard end-to-end deadlines
  - are critical to the system

- Examples
  - Fire detection
  - Target detection
Approach to Aperiodic Tasks

- Arrival times of aperiodic tasks are unknown
  - Possibly unbounded inter-arrival time
  - Impossible to provide offline guarantees

- Aperiodic task with a soft deadline
  - Maintain hard guarantees on periodic tasks
  - Reduce response time of aperiodic tasks

- Aperiodic task with a hard deadline
  - Online guarantees through admission control
Theory vs. Middleware

- Theoretical techniques for **online** aperiodic scheduling
  - Aperiodic Utilization Bound (AUB) [Abdelzaher, 2004]
  - Aperiodic Servers
    - Polling Server [Sha 1986]
    - Deferrable Server [Strosnider 1995]
    - Priority Exchange [Lehoczyk 1987]
    - Sporadic Server [Sprunt 1989]
    - Slack Stealing [Lehoczyk 1992]

- Middleware lacks support for real-time aperiodic tasks
  - Lack scheduling mechanism
  - Lack online admission control
Contributions

- Middleware architecture within federated event channel
  - End-to-end scheduling for aperiodic & periodic tasks
  - Online admission control service

- Support and compare two alternative approaches
  - Aperiodic Utilization Bound (AUB)
  - Deferrable Server (DS)
End-to-End Tasks

Subtasks communicate through federated event service
Deferrable Server: Theory

- A periodic server executes all aperiodic tasks
- Budget: maximum time the server can run in a period
  - Bound aperiodic tasks’ impact on periodic tasks
- Algorithm
  - Server is suspended when its budget runs out
  - Server’s budget is replenished at the end of each period
- Online test: worst-case end-to-end response time ≤ deadline

- Admission test: end-to-end response time analysis
- Requires run-time scheduling mechanism
Example: Deferrable Server

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Server

- $C_s = 2$
- $T_s = 5$

**Figure 5.7** Example of a Deferrable Server scheduled by RM.
First DS implementation on top of priority-based OS (e.g., Linux, POSIX)
Server thread processes aperiodic events (2nd highest priority)
Budget manager thread (highest priority) manages the budget and controls the execution of server thread
Aperiodic Utilization Bound

- **End-to-end task** $T_i$
  - $D_i$: end-to-end deadline
  - $C_{ij}$: subtask execution time on node $j$
  - Aperiodic or periodic

- **Current job set** $S(t)$
  - Jobs that have arrived but whose **deadlines** have not expired
  - Note: A job may still be “current” after completion

- **Synthetic utilization** of processor $j$: $U_j(t) = \sum_{T_i \in S(t)} \frac{C_{ij}}{D_i}$

  **Idle resetting**: when a processor becomes idle, remove the contribution of all completed aperiodic jobs from the processor’s synthetic utilization

- **Online test**: end-to-end deadline of a task is guaranteed if

  \[
  \sum_{1 \leq j \leq 4} \frac{U_j(1 - U_j/2)}{1 - U_j} \leq 1
  \]
Admission Control Service

- Round trip delay < 1.4 ms
- Considered in schedulability analysis

**Admission Controller**

- accept task
- update util
- sched. analysis
- not accepted
- accept?
- set timer
- timer fires

**Application**

- send request
- accept
- reject task
- idle report
- release task
- task arrives

- Application Processor 1
- Application Processor 2
- Application Processor 3
Admission Control: Design Decisions

- **Critical Tasks**
  - Send an event to notify the central admission controller
  - Hold the task in a waiting queue and waits for the reply

- **Noncritical Tasks**
  - AC may eject noncritical periodic tasks when new critical ones arrive
    - (*criticality-aware admission policy*)

- **Aperiodic Tasks**
  - Update utilization when an aperiodic job is released or reaches its deadline
  - When CPU idles, idle detector reports completed aperiodic jobs to AC
    - (*resetting rule*)

- **Periodic Tasks**
  - Do not update synthetic utilization at every release or deadline.
Experimental Platforms

Task Set
• One task set contains 4 aperiodic tasks and 5 periodic tasks
• Randomly generate 60 task sets with synthetic utilization of 0.3-0.6
Comparison of AUB and DS

- Idle resetting is effective
- DS and AUB are comparable
  - But DS requires more complex run-time mechanism
Impact of Criticality

- **AUB or DS**: do not eject noncritical tasks to accept new critical tasks
- **AUB or DS with Criticality**: eject noncritical tasks to accept new critical tasks

AUB with criticality is more effective than DS for critical tasks
Conclusions

- Real-time middleware for aperiodic/period end-to-end tasks
  - Online admission control service
    - Aperiodic Utilization Bound (AUB)
    - Deferrable Server (DS)
  - Scheduling service: DS on top of priority-based OS

- Integration with TAO federated event channel

- Evaluation: efficiency
  - DS is efficient at the middleware level
  - Online admission control incurs acceptable delays

- Evaluation: admission control
  - AUB is comparable to DS with simpler run-time mechanisms
  - Criticality-aware policy accepts more critical tasks