A Unified Runtime Framework for Weakly-hard Real-time Systems

Hyunjong Choi, Hyoseung Kim
I. Introduction & Related Work

Weakly-hard real-time systems

- Improve resource usage efficiency
  - Tolerable to some deadline misses w/o affecting functional correctness

\[(m, K)\]: at most \(m\) jobs can miss their deadlines among any \(K\) consecutive jobs

- Various assumptions on handling of deadline-missed jobs

<table>
<thead>
<tr>
<th>Handling scheme</th>
<th>Prior work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job abort</td>
<td>Goossens (RTN, 2008), Koren (RTSS, 1995), Ramanathan (1999)</td>
</tr>
<tr>
<td>Delayed completion</td>
<td>Hammadeh (ECRTS 2017), Sun (TECS, 2017)</td>
</tr>
<tr>
<td>Job pre-skip</td>
<td>Koren (RTSS, 1995), Ramanathan (1999)</td>
</tr>
</tbody>
</table>

< Weakly-hard studies based on job handlings >

No prior work of comparative analysis among various handling schemes

Handling of deadline-missed jobs

- Four handling schemes

**Job abort**

- Terminal immediately
- No effect on the next released job
- Drawback: implementation cost (rollback: system-level vs. **task-level**)  

**Delayed completion**

- Run until a job completes
- Can improve quality of service of a system
- Drawback: no merits of weakly-hard concept in overloaded situations

**Job pre-skip**

- Determine at a job release time
- **Online** (slack time) and offline (predetermined patterns)
- Drawback: runtime overhead (slack) and underutilization

**Job post-skip**

- Run until a job completes, but discard the next released job
- Drawback: **degradation** of quality of service of a system
Runtime framework

- Job abort
  - Rollback mechanism (task-level)
  
  Step 1. Store a checkpoint
  Step 2. Notify a deadline miss to the user space
  Step 3. Recover from the checkpoint

- Delayed completion
  - Put in sleep mode when the latest released job is completed

- Job pre-skip
  - Online vs offline

Additional sequences based on handling schemes
Computational overheads

- **Experimental setup**
  - Linux kernel running on Raspberry Pi 3 (Quad Cortex A53 @ 1.2GHz)

- **Four major sequences that can cause extra runtime overhead**
  - `sigsetjmp (job abort)`, `siglongjmp (job abort)`, slack (job pre-skip), pattern (job pre-skip)

Acceptably small in $\mu$s units, compared to periods of tasks denoted in ms
Conclusion & Future work

**Conclusion**
- Proposed a unified runtime framework for multiple deadline-miss handling schemes in weakly-hard real-time systems
- Applicable to other OSs using fixed-priority preemptive schedulers
- Different results (violation of the constraints, utilization) observed depending on the handling scheme for the same taskset

**Future work**
- Will use for the issues that have not studied much in weakly-hard context (e.g., inter-task dependency, shared resources, multicore systems, and contention in cache and main memory)
Thank you

A Unified Runtime Framework for Weakly-hard Real-time Systems

Hyunjong Choi, Hyoseung Kim

Q & A
Runtime mechanism

- A fundamental runtime mechanism for periodic task execution

Legend:  
- ▼ System call  
- ▲ Awake a task  
- ↩ Sequence in user-space  
- ► Execution of a task

**User-space**
- Create a thread
- Call to register as a real-time task (SYSCALL)

**Kernel-space**
- **Initializer**
  - Assign a real time priority
  - Set a task as a real-time task
  - Register a timer
- **Scheduler**
  - Trace tasks on the resource (profiling)

**Periodic task**
- Begin
- Repeat
- Job complete (SYSCALL)

**Complete sequence**
- - Put a task sleep
- - Record a complete time (profiling)
- - Wait until the next timer interrupt

**Additional sequences based on handling schemes**
Job abort scheme

- Employed *task-level rollback* approach

```c
// Kernel signal handler
signl Handler {
    siglongjmp(sigjmp_buf);
}

// Periodic task
While (1) {
    sigsetjmp(sigjmp_buf);
}
```

**Step 1. Store a checkpoint**

**Step 2. Notify deadline miss**

**Step 3. Recover from the checkpoint**

- PC & SP rollback,
- User-space
- Kernel-space
Case study

- Select a taskset given in the study
  - RM-RTO\(^\dagger\) algorithm

<table>
<thead>
<tr>
<th>Tasks</th>
<th>T [ms]</th>
<th>C [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>19</td>
<td>5</td>
</tr>
</tbody>
</table>

< Taskset 2 with skip parameter\(*\) of 2 >

\(^\dagger\) RM-RTO stands for Rate Monotonic Red Task Only

\(^\dagger\) G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In RTSS, 1995

\(*\) Tolerance of a task to missing deadlines

\(\varphi\) A task experiences more than \(m\) deadline misses in a window of \(K\) jobs.