Programmable Temporal Isolation through Variable-Bandwidth Servers

Silviu S. Craciunas

Department of Computer Sciences
University of Salzburg

joint work with
C. Kirsch, H. Payer, H. Röck, and A. Sokolova
• action is a piece of code
• process is a sequence of actions
schedule the processes so that each of their actions maintains its response time
Goal

- solve the scheduling problem
  (temporal isolation)
- change execution speed of processes
  (programmable)
- solve admission problem
  (changeable set of processes)
Goal

- solve the scheduling problem
  - temporal isolation
- change execution speed of processes
  - programmable
- solve admission problem
  - changeable set of processes

Solvable with variable bandwidth servers (VBS)

Results:
- a constant-time scheduling algorithm
- a constant-time admission test
Resources and VBS

- VBS is determined by a bandwidth cap \((u)\)
- VBS processes dynamically adjust speed (resource)
  \[
  \frac{\lambda_1}{\pi_1} \leq u \quad \text{and} \quad \frac{\lambda_2}{\pi_2} \leq u
  \]
- Generalization of constant bandwidth servers (CBS)
  [Abeni and Buttazzo 2004]
One process on a VBS

process

load

action $\alpha_1$

response time

load

action $\alpha_2$

response time

process running on a VBS

$\lambda_1$

$\pi_1$

response time under VBS

$\lambda_2$

$\pi_2$

response time under VBS
VBS

- process running on a VBS
- arrival
- time
- termination
- response time under VBS
- new action
- dispatch
- release
- blocked
- ready
- running
- next action
- preemption due to limit
- preemption due to release
multiple processes are EDF-scheduled
Scheduling result and bounds

Processes $P_1, P_2, ..., P_n$ on VBSs $u_1, u_2, ..., u_n$, are schedulable if $\sum u_i \leq 1$

For any action $\alpha$ on a resource $(\lambda, \pi)$ we have

- upper response time bound: $\lceil \frac{\text{load}}{\lambda} \rceil \pi + \pi - 1$
- lower response time bound: $\lfloor \frac{\text{load}}{\lambda} \rfloor \pi$

jitter: $\pi - 1$
Programmable temporal isolation

the “speed“ of an action is programmable
(influencing response time and jitter)

smaller $\pi \Rightarrow$

+ smaller jitter
+ VBS response time closer to „ideal“ response time
- higher administrative overhead

(more scheduler invocations)

Finding the right $\lambda, \pi$ is difficult.
Real-world example

```java
loop {
    sensor_data = read(sensors);
    actuator_data = compute(sensor_data);
    write(actuator_data);
    log(actuator_data);
    update_internal_state();
}

} until (done);
```

- **低延迟** (low latency)
- **低要求** (less stringent)
- **控制循环周期** (control-loop period)
Real-world example

```
loop {
    sensor_data = read(sensors);
    actuator_data = compute(sensor_data);
    write(actuator_data);
    log(actuator_data);
    update_internal_state();
}

} until (done);
```

Different throughput and latency requirements for different portions of code
Implementation

- constant-time scheduling algorithm
- different queue management plugins
  
  (lists, arrays, matrices, trees)

trade off time and space complexity

<table>
<thead>
<tr>
<th></th>
<th>list</th>
<th>array</th>
<th>matrix/tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>$O(n^2)$</td>
<td>$O(\log(t)+n\log(t))$</td>
<td>$\Theta(t)$</td>
</tr>
<tr>
<td>space</td>
<td>$\Theta(n)$</td>
<td>$\Theta(t+n)$</td>
<td>$O(t^2+n)$</td>
</tr>
</tbody>
</table>

$n$ - number of processes  
$t$ - number of time instants
Results

scheduler overhead

Maximum duration (μs) for increasing number of processes

break-even point

Standard deviation for increasing number of processes
scheduler overhead (list)

20 scheduler runs had 100μs duration
Results

scheduler overhead (array)
scheduler overhead (matrix)
Results

space complexity

memory usage

1GB
100MB
5MB
100KB
5KB
matrix
tree
array
list

KB

2^0
2^5
2^10
2^15
2^20

time instants (t)

750 processes

750 processes
Results

bare-metal experiment

Graph showing response time (ms) against number of processes, with CPU utilization (%) on the y-axis.
Results

bare-metal experiment

- in theory response time jitter is one period
- in practice the jitter may be more than one period but still bounded

- scheduler
- overhead accounting

Graph showing response time (ms) on the y-axis and number of processes on the x-axis.

Data points for response time and number of processes are plotted.

CPU utilization is also shown on the graph.
Conclusion

VBS scheduling enables:

- temporal isolation
- trading off throughput and latency
- controlling the response-time jitter of individual process actions
- trading off space and time complexity of the scheduling overhead

http://tiptoe.cs.uni-salzburg.at/
VBS scheduling enables:

- temporal isolation
- trading off throughput and latency
- controlling the response-time jitter of individual process actions
- trading off space and time complexity of the scheduling overhead

Thank you!

http://tiptoe.cs.uni-salzburg.at/
• CBS does not allow changing of the period and limit
• RBED and VBS differ on the level of abstractions provided
Release strategy

late release

arrival

response time under VBS

time

termination

early release

arrival

response time under VBS

time

termination
Logical response time jitter

Process running on VBS

arrival

actual termination

logical termination

jitter

$2(\pi - 1)$