Robust Guaranteeable Management of System Resources

Scott Brandt

and Scott Banachowski, Tim Bisson, Richard Golding, Tim Kaldewey, Caixue Lin, Suruchi Malatpure, Carlos Maltzahn, Roberto Pineiro, Anna Povzner, Ted Wong, and Joel Wu

1 June 2007
We need resource guarantees

• *Many* applications need resource guarantees
  • CPU: Audio, video, soft devices, data acquisition, control systems, GPS, image processing, scientific visualization
  • I/O: Data capture, capping background activities (recovery, backup, etc.), any time-constrained processing involving lots of data
  • Network: VOIP, streaming multimedia, I/O
  • Even so-called best-effort applications have constraints

• The increasing power of computer hardware has made such applications feasible on general-purpose systems, but resource management techniques predate them
Existing systems provide poor solutions

• Most systems provide one type of processing
  • Best-effort
    • no guarantees → over-provision
    • result: poor utilization, still no guarantees
  • Real-time
    • guaranteed scheduling → over-constrain
    • result: a priori knowledge, poor utilization, all-or-nothing (i.e. no graceful degradation)
• Hierarchical hybrids get the worst of both
  • BE w/fixed-priority: poor utilization, expert knowledge, poor responsiveness
  • RT w/BE in background: poor utilization, expert knowledge, poor responsiveness
Achieving robust guaranteeable resources

- Goal: Unified resource management algorithms capable of providing
  - Good performance
  - Arbitrarily hard or soft performance guarantees with
    - Arbitrary resource allocations
    - Arbitrary timing granularity
  - Complete isolation between processes/sets of processes
  - All resources: CPU, disk, network, memory
Example: CPU scheduling (1/3)

• Video player needing 10 ms of CPU per frame, decoding and displaying 24 frames/second
  = 240 ms per second
  = 24% of the CPU

< 24%  →  missed deadlines
> 24%  →  wasted resources
== 24%  →  good, but
Example: CPU scheduling (2/3)

• Video player needing 10 ms of CPU per frame, decoding and displaying 24 frames/second
  = 24% of the CPU
  = 10 ms per 41.6 ms

240 ms every second → jitter
1 ms every 4.16 ms → overhead
10 ms every 41.6 ms → perfect
Example: I/O scheduling (3/3)

- Video player needing 10 ms of CPU per frame, decoding and displaying 24 frames/second
  = 24% of the CPU
  = 10 ms every 41.6 ms
  + one frame of I/O every 41.6 ms
Observation

• Resource management consists of two distinct decisions
  • Resource Allocation: How much resources to allocate?
  • Dispatching: When to provide the allocated resources?

• Most resource managers integrate their management or punt on one or both
  • Best-effort, proportional-share, real-time
Separating them is powerful!

- Separately managing resource allocation and dispatching gives direct control over the amount and timing of resource management.
- Enables direct support of all types of timeliness needs.
The resource allocation/dispatching (RAD) scheduling model

- Rate
- Share of resources
- Series of jobs w/ budgets and deadlines
- Times at which allocation must equal reservation

Process

Rate

Deadlines

Dispatcher
Supporting different timeliness requirements with RAD

- **Hard Real-time**
  - Rate-based
    - Period WCET
  - Soft Real-time
    - Rate Bounds
    - Period ACET
  - Best-effort
    - Priority

- **Scheduling Policy**
  - Rate
  - Deadlines

- **Runtime System**
  - Scheduling Mechanism
  - Set of jobs w/ budgets and deadlines

- **Dispatcher**
  - $P_i$
The Rate-Based Earliest Deadline (RBED) CPU scheduler

- Processes are assigned rates
  - Based on priorities, processing characteristics, availability, etc.
  - $\sum$ rates $\leq$ 100%
- Jobs are formed with w/budget (how much) and deadline (when)
  - Deadlines based on processing characteristics
  - budget = rate * relative deadline
- Jobs are dispatched with Earliest Deadline First (EDF)
  - Guarantees deadlines up to 100% utilization
  - Timers enforce budget
## Assigning rates and deadlines

<table>
<thead>
<tr>
<th>Process Type</th>
<th>Rate</th>
<th>Deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRT</td>
<td>WCET/period</td>
<td>Period</td>
</tr>
<tr>
<td>RB</td>
<td>Rate</td>
<td>Based on rate bounds (e.g., buffering capacity)</td>
</tr>
<tr>
<td>SRT</td>
<td>Based on needs and availability</td>
<td>WCET/Rate or determined via adaptation</td>
</tr>
<tr>
<td>BE</td>
<td>Based on availability and priority</td>
<td>Based on responsiveness needs</td>
</tr>
</tbody>
</table>
### Examples

<table>
<thead>
<tr>
<th><strong>Process</strong></th>
<th><strong>Example</strong></th>
<th><strong>Rate</strong></th>
<th><strong>Deadlines</strong></th>
<th><strong>Jobs</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Budget</strong></td>
</tr>
<tr>
<td>HRT</td>
<td>Flight Control</td>
<td>20%</td>
<td>1 s</td>
<td>200 ms</td>
</tr>
<tr>
<td>RB</td>
<td>Audio</td>
<td>20%</td>
<td>10 s of buffer</td>
<td>20%</td>
</tr>
<tr>
<td>SRT</td>
<td>Video</td>
<td>20%</td>
<td>41.6 ms</td>
<td>≤ 20%</td>
</tr>
<tr>
<td>I/O-bound BE</td>
<td>Editor</td>
<td>(w_i U_{BE}) (w_i = \frac{\text{pri}_i}{\sum \text{pri}_j})</td>
<td>100 ms</td>
<td>100 ms * (\frac{1}{n} * U_{BE})</td>
</tr>
<tr>
<td>CPU-bound BE</td>
<td>Compiler</td>
<td>(w_i U_{BE}) (w_i = \frac{\text{pri}_i}{\sum \text{pri}_j})</td>
<td>1 s</td>
<td>1 s * (\frac{1}{n} * U_{BE})</td>
</tr>
</tbody>
</table>
RBED Task model

- Processes have a guaranteed rate of execution $r$
  - Reserved via admission control or assigned dynamically such that $\sum r \leq 100$
- Processes are expressed as a series of jobs $j_i$
  - = a unit of work within a process
- Each job has a deadline $d_i$
  - $\Delta_i = d_i - \text{now}$ ($\Delta_i$ is the relative deadline)
- Each job has a budget $b_i = r\Delta_i$
RBED Theory 1 (at job release)

- RBED guarantees that all jobs $j_i$ will receive their budget $b_i = r\Delta_i$ by time $d_i$

  ➡ All processes will receive their assigned rate

  ➡ The actual rate will exactly equal the assigned rate at the processes’ deadlines

- And
  - A process’ rate can be increased or decreased arbitrarily (within available resources)
  - A job’s deadline $d_i$ can be given any value (within limits due to overhead considerations)
RBED Theory II (at any time)

• Rate changes
  + The rate of a process can be increased at any time (within available resources)
  - The rate of a process can be decreased at any time, down to the rate corresponding to the current job’s resource usage

• Deadline changes
  + The deadline of a job can be increased at any time
  - The deadline of a job can be reduced at any time, down to the time corresponding to its resource usage

• Corollaries:
  ~ The deadline of a job can be increased at any time (without changing budget) [deadline increase, rate decrease]
  ~ The deadline of a job can be decreased at any time (without changing budget), down to the current time (as long as resource are available for the rate change) [deadline increase, rate decrease]
What do we have

• A scheduling framework capable of guaranteeing
  • Rate of resource allocation
  • Timing of resource delivery
  • Isolation
• With arbitrarily hard or soft applications
• Based on proven real-time principles
• Applications
  • CPU, disk, network, ...
  • Real-time systems, best-effort systems, *everything*
  • Embedded systems, desktop systems, servers, VMs
Talks today

• 11:00–11:20 Scott Brandt
  Robust Guaranteeable Management of System Resources

• 11:20–11:35 Jaeheon Yi and Suresh Iyer
  Integrated Scheduling in an Embedded System

• 11:35–11:55 Theodore Wong
  Utilization As An Efficient Guaranteeable Metric of Disk Performance

• 11:55–12:15 Anna Povzner
  Fahrrad: An I/O Scheduler for Efficient Guaranteed Disk Performance

• 12:15–12:30 Roberto Pineiro
  Caching for Improved Storage QoS Guarantees