Robust and Energy-Efficient Real-Time Systems

Lecture 9: Multiprocessors in real-time systems

(Based on material kindly provided by Jan Jonsson)
Why multiprocessors?

- High throughput
  - Parallel execution of tasks (actions/threads/processes)
  - Parallelization of task algorithms (explicit/implicit)
- Price/performance ratio is decreasing
  - Multicore CPUs are already commonplace
- Push-pull effect:
  - New applications push future computer performance
  - New computer platforms pull new applications
- Reliability
  - Faulty CPU (just) increases response times
Multiprocessor scheduling

How are tasks assigned to processors?

- **Static assignment**
  - The processor(s) used for executing a task are determined before system is put in mission (*off-line*).
  - Algorithms: partitioned scheduling

- **Dynamic assignment**
  - The processor(s) used for executing a task are determined during system operation *on-line*.
  - Algorithms: global scheduling
Multiprocessor scheduling

How are tasks allowed to migrate?

- Partitioned scheduling
  - A task must *always execute on the same processor*
  - Equivalent to multiple uniprocessor systems!

- Global scheduling
  - A task is allowed to *execute on an arbitrary processor* (sometimes even after being preempted)
Partitioned scheduling

General characteristics:

- Each processor has its own queue for ready tasks
- Tasks are organized in groups, and each task group is assigned to a specific processor
- When selected for execution, a task can only be dispatched to its assigned processor
Partitioned scheduling

**Advantages:**

- Mature scheduling framework
  - Uniprocessor scheduling applicable for each CPU
  - Uniprocessor resource-management can be used
- Partitioning of tasks is done ahead of execution, and may be automated

**Disadvantages:**

- Cannot exploit all unused execution time
  - Surplus capacity cannot be shared among processors
  - Will suffer from overly-pessimistic WCET derivation
Partitioned scheduling

Major disadvantage:

The partitioning problem itself – i.e., splitting a task set into $m$ groups that are each schedulable on a uniprocessor, if this is at all possible – is NP-complete!

- Consequence: there cannot exist an efficient partitioning algorithm that is optimal (unless P = NP)
- For practical partitioned systems, heuristic algorithms must be used
  - Bin-packing, branch-and-bound, simulated annealing, ...
Bin-packing algorithms

- Basic idea: The problem concerns packing objects of varying sizes in bins with the objective of minimizing number of used bins...

- Application to multiprocessor systems:
  - Bins = processors, objects = tasks, size = utilization $U_i$
  - The decision whether a processor is "full" or not is derived from a utilization-based feasibility test

- Assumptions:
  - Independent, periodic tasks
  - Preemptive, uniprocessor scheduling (RM)
A bin-packing partitioning algorithm

Rate-Monotonic-First-Fit (RMFF): (Dhall and Liu, 1978)

- Let the processors be indexed as \( N_1, N_2, \ldots \)
- Assign the tasks in the order of increasing periods (RM)
- For each task \( \tau_i \), pick the first previously-used \( N_j \) such that \( \tau_i \) can be feasibly scheduled together with all tasks that have already been assigned to \( N_j \) (according to the utilization-based RM-feasibility test)
- If no such \( N_j \) exists, assign \( \tau_i \) to the next unused processor
A bin-packing partitioning algorithm

The utilization bound $U_{RMFF}$ for a system with $m$ processors using the RMFF scheduling policy is

$$m(2^{1/2} - 1) \leq U_{RMFF} \leq (m + 1)/(1 + 2^{1/(m+1)})$$

(Oh & Baker, 1998)

Note: \(2^{1/2} - 1 \approx 0.41\)

Thus: task sets whose utilization do not exceed \(\approx 41\%\) of the total processor capacity are always RMFF-schedulable.
Branch-and-bound algorithms

Basic idea:

- A set of solutions to a given problem is organized in a search tree
- A vertex in the search tree corresponds to a specific solution structure
- A goal vertex corresponds to a complete solution to the problem and is located at a leaf of the tree
- The root vertex corresponds to an initial solution
- The search for a solution starts with the root vertex
- Search objective is to find a goal vertex that optimizes a given cost (performance measure)
Branch-and-bound algorithms

Basic idea:

- For each vertex, a set of child vertices is generated by modifying the structure of the current vertex (branching)
- To check if a tree branch may lead to an acceptable solution, a lower-bound function is applied to each of the child vertices
- If a child vertex looks promising, it will be further investigated
- If a child vertex will only lead to inferior solutions, that entire branch is pruned (bounding)
Branch-and-bound algorithms

Application to multiprocessor scheduling:

- The search tree = all possible task-to-processor assignments
- A vertex = partial or complete assignment
- The root vertex = an empty schedule
- A goal vertex = a complete schedule
- Lower-bound function checks whether a child vertex contains a feasible schedule
- Generation of a child vertex = adding one of the ready tasks to the schedule in the current vertex
Global scheduling

General characteristics:

- All ready tasks are kept in a common global queue
- When selected for execution, a task can be dispatched to an arbitrary processor, even after being preempted
- Task execution is assumed to be greedy:
  - If higher-priority tasks occupy all processors, a lower-priority task cannot grab a processor until the execution of a higher-priority task is complete
Global scheduling

Advantages:
- Supported by most multiprocessor operating systems
- Effective utilization of processing resources
  - Unused processor time can easily be reclaimed

Disadvantages:
- Weak theoretical framework
  - Few results from the uniprocessor case can be used
- Poor resource utilization for hard timing constraints
  - No more than 50% resource utilization guaranteed
- Suffers from several scheduling anomalies
  - Sensitive to period adjustments
Global scheduling

- The problem of deciding if a task set is schedulable on \( m \) processors with respect to global scheduling is \textbf{NP-complete}.

- The "\textit{root of all evil}" in global scheduling (Liu, 1969):

  "\textit{The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.}"

- Consequence: we're in deep trouble!
Weak theoretical framework

Results for RM, DM and EDF do not just carry over:

- Dhall’s effect (Dhall & Liu, 1978):
  - Some low-utilization task sets can be unschedulable regardless of how many processors are used

- Dependence on relative priority ordering:
  - Changing relative priority among higher-priority tasks may affect schedulability for a lower-priority task

- Hard-to-find critical instant:
  - A critical instant does not always occur when a task arrives at the same time as all higher-priority tasks
Dhall's effect

Greedy RM or EDF ($m=3$):

$C_1 = 2\varepsilon, T_1 = 1$
$C_2 = 2\varepsilon, T_2 = 1$
$C_3 = 2\varepsilon, T_3 = 1$
$C_4 = 1, T_4 = 1+\varepsilon$

Deadline miss
Dhall's effect

Greedy RM or EDF ($m=3$):

\begin{align*}
C_1 &= 2\epsilon, \quad T_1 = 1 \\
C_2 &= 2\epsilon, \quad T_2 = 1 \\
C_3 &= 2\epsilon, \quad T_3 = 1 \\
C_4 &= 1, \quad T_4 = 1 + \epsilon
\end{align*}

\[
U_{global} = m \frac{2\epsilon}{1} + \frac{1}{1 + \epsilon}
\]

Total multiprocessor utilization $U_{global}$ can be arbitrarily close to 1 no matter how many processors used!
Avoiding Dhall's effect

Priority based on utilization \((m=3)\):

- \(C_1 = 2\varepsilon, T_1 = 1\)
- \(C_2 = 2\varepsilon, T_2 = 1\)
- \(C_3 = 2\varepsilon, T_3 = 1\)
- \(C_4 = 1, T_4 = 1+\varepsilon\)
A new assignment scheme

- Algorithm **RM-US**[$m/(3m-2)$] (Andersson, Baruah & Jonsson, 2001) assigns *static* priorities to tasks according to the following rule:

  - If $U_i > m/(3m-2)$ then $\tau_i$ has the *highest* priority (ties broken arbitrarily)
  - If $U_i \leq m/(3m-2)$ then $\tau_i$ has **RM** priority

<table>
<thead>
<tr>
<th>$m$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m/(3m-2)$</td>
<td>1</td>
<td>0.5</td>
<td>0.43</td>
<td>0.4</td>
<td>0.38</td>
<td>0.37</td>
<td>0.33</td>
</tr>
</tbody>
</table>
RM-US[m/(3m-2)] example

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>0.143</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>0.45</td>
</tr>
<tr>
<td>11</td>
<td>22</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Assume 3 processors

$m = 3$ means breakpoint is $\approx 0.43$

Highest priority: $\tau_3$ and $\tau_4$ (arbitrarily ordered)
Thereafter: $\tau_1$, $\tau_2$ and $\tau_5$ (in decreasing order)
Feasibility

- Processor utilization-based test for RM-US\[m/(3m-2)\]
  (Andersson, Baruah & Jonsson 2001):

  \[
  U_{RM-US} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \frac{m^2}{3m - 2}
  \]

  (A sufficient condition)

- Note: \(U_{RM-US}\) reaches \(m/3\) as \(m\) grows towards infinity

- Thus any task set can be feasibly scheduled by RM-US\[m/(3m-2)\] given sufficiently many processors

- I.e., RM-US\[m/(3m-2)\] avoids Dhall's effect
A fundamental limit

- The utilization guarantee bound for any static-priority multiprocessor scheduling algorithm cannot be higher than $1/2$ of the capacity of the processors (Andersson, Jonsson & Baruah 2001)

- This applies for all types of static-priority scheduling (partitioned or global, greedy or not)

- Hence, we can never expect to utilize more than half the processing capacity if hard timing constraints exist

- The most resource-efficient multiprocessor real-time systems are therefore ones with a mix of soft and hard constraints
Scheduling anomalies

- Scheduling anomaly: A *seemingly positive change* in the system (reducing load or adding resources) causes a non-intuitive *decrease in performance*

- State-of-the-art (*uniprocessor systems*):
  - Anomalies only found for *non-preemptive scheduling* (Mok, 2000)

- State-of-the-art (*multiprocessor systems*):
  - Many anomalies for non-preemptive scheduling...
  - *Execution-time-based & period-based* anomalies also for *preemptive scheduling*!
Scheduling anomalies

Execution-time-based anomalies: (Ha & Liu, 1994)

- Assumptions:
  - Preemptive multiprocessor scheduling
  - Independent tasks
  - Restricted migration (individual task instances cannot migrate)
  - Fixed execution times

- Task response times may increase as a result of:
  - Reducing task execution times!
Scheduling anomalies

Period-based anomalies: (Andersson & Jonsson, 2000)

- Assumptions:
  - Preemptive multiprocessor scheduling
  - Independent tasks
  - Full migration
  - Fixed execution times

- A task’s response time may increase as a result of:
  - Increasing the period of a higher-priority task
  - Increasing the period of the task itself

- Note: increasing the periods is commonly used to reduce the load in feedback-control systems!
State-of-the-art in global scheduling

- Static priorities:
  - RM-US\(\frac{m}{3m-2}\) circumvents Dhall’s effect, has a utilization guarantee bound of \(\frac{m}{3m-2} \geq 33.3\%\).
  - Generalized by Baker (2003) to DM

- Dynamic priorities:
  - EDF-US\(\frac{m}{2m-1}\) (Srinivasan & Baruah, 2002), a utilization guarantee bound of \(\frac{m}{2m-1} \geq 50\%\).

- Optimal multiprocessor scheduling:
  - Using so called p-fair scheduling and dynamic priorities it is possible to achieve 100% resource utilization on a multiprocessor.