Multi-core real-time scheduling

Introduction: problem definition and classification
Some anomalies of multiprocessor scheduling
Model and assumptions
Extension of uni-processor scheduling strategies
Pfair approaches
Introduction

- Mono-processor scheduling: one-dimension problem
  - **Temporal** organization
    - When to start, interrupt, resume every task?

- Multi-processor (multi-core) scheduling: two-dimension problem
  - **Temporal** organization +
  - **Spatial** organization
    - On which processor execute every task?
Classification

- **Partitioned** scheduling
  - Each of the two dimensions is dealt with separately
- **Global** scheduling
  - Temporal and spatial dimensions are dealt with jointly
- **Semi-partitioned** scheduling
  - Hybrid

Classification: partitioned scheduling

- Each of the two dimensions is dealt with separately
  - Spatial organization: the n tasks are partitioned onto the m cores. No task migration at run-time
  - Temporal organization: Mono-processor scheduling is used on each core
Classification: partitioned scheduling

Two points of view

- Number of processors to be determined: optimization problem (bin-packing problem)
  - Bin = task, size = utilization (or other expression obtained from the task temporal parameters)
  - Boxes = processors, size = ability to host tasks
- Fixed number of processors: search problem (knapsack problem)

Both problems are NP-hard

Optimal mono-processor scheduling strategies: XX
- RM, DM
- EDF, LLF (see uni-processor scheduling chapter)

Bin-packing heuristics: YY
- FF: First-Fit
- BF: Best-Fit
- WF: Worst-Fit, NF: Next-Fit
- FFD, BFD, WFD: First/Best/Worst-Fit Decreasing

Partitioning algorithms XX-YY
Classification: partitioned scheduling

- **Benefits**
  - Implementation: local schedulers are independent
  - No migration costs
  - Direct reuse of mono-processor schedulability tests
  - Isolation between processors in case of overload

- **Limits**
  - Rigid: suited to static configurations
  - NP-hard task partitioning
  - Largest utilization bound for any partitioning algorithm [Andersson, 2001] \( \frac{m+1}{2} \)
    
    \[ \text{(m+1 tasks of execution time 1+} \varepsilon \text{ and period 2)} \]

Classification: global scheduling

- **Temporal and spatial dimensions are dealt with jointly**
  - Global unique scheduler and run queue
  - At each scheduling point, the scheduler decides when and where schedule at most m tasks
  - Task migration allowed
Classification: global scheduling

- **Benefits**
  - Suited to dynamic configurations
  - Dominates all other scheduling policies
    - (if unconstrained migrations + dyn. priorities – see later)
  - Optimal schedulers exist
  - Overloads/underloads spread on all processors

- **Drawbacks**
  - System overheads: migrations, mutual exclusion for sharing the run queue

---

Classification: global scheduling

- *(Preemptive)* global RM/DM/EDF: definition
  - Task priorities assigned according to RM/DM/EDF
  - Scheduling algorithm: the m higher priority tasks are executed on the m processors
Classification: semi-partitioned scheduling

- Partitioned scheduling as far as possible
- Some statically determined tasks may migrate
  - Constraint: migrating tasks (T4 on the example) must execute on a single processor at a time

Terminology

- A task set is **schedulable** if there exists a scheduling policy such that all deadlines are met
- A task set is **schedulable by a scheduling policy** if under that scheduling policy all deadlines are met
- A scheduling policy is **optimal** if it is able to correctly schedule all schedulable task sets
  - Different from the optimality defined before
- **Utilization bound** of a scheduling policy: utilization $U_{\text{lim}}$ below which all task sets meet their deadline
Terminology

- Priorities
  - Fixed per task (FTP)
  - Fixed per job (FJP)
  - Dynamic per job (DJP)

Overview of global scheduling policies

- Assumptions
  - Tasks
    - Periodic tasks (Pi)
    - Implicit deadlines (Di=Pi)
    - Synchronous tasks (Oi=0 for all i)
    - Independent tasks
    - A single job of a task can be active at a time
  - Architecture
    - Identical processors
    - System costs are neglected (preemption, migration, scheduling policy)
Scheduling anomalies (1/3)

- Dhall's effect [Dhall & Liu, 1978]
  - Periodic task sets with utilization close to 1 are unschedulable using global RM / EDF
  - $n = m+1$, $P_i = 1$, $C_i = 2\varepsilon$, $u_i=2\varepsilon$ for all $1 \leq i \leq m$
  - $P_{m+1}=1+\varepsilon$, $C_{m+1}=1$, $u_{m+1}=1/(1+\varepsilon)$
  - Task $m+1$ misses its deadline although $U$ very close to 1

\[
\begin{align*}
T_1 & \quad 0 \quad 2\varepsilon \\
T_m & \quad 1 \\
T_{m+1} & \quad 1+\varepsilon
\end{align*}
\]

Scheduling anomalies (2/3)

- Period increase for periodic tasks and fixed priorities [Anderson, 2003]
  - $n = 3$, $m=2$, $(P_1= 3, C_1=2)$, $(P_2=4,C_2=2)$, $(P_3=12,C_3=7)$
  - Schedulable under global RM
  - If $P_1$ is increased to $P_1=4$ and priorities stay the same, $T_3$ misses its deadline
Scheduling anomalies (2/3)

- \((P_1= 3, C_1=2), (P_2=4, C_2=2), (P_3=12, C_3=7)\)

- \((P_1= 4, C_1=2), (P_2=4, C_2=2), (P_3=12, C_3=7)\)

\[\text{Critical instant not necessarily the simultaneous release of higher priority tasks}\]

- \(n=3, m=2\)
- \((P_1=2, C_1=1), (P_2=3, C_2=2), (P_3=4, C_3=2)\)
- Under RM scheduling
  - Response time of \(T_3\) higher at time 4 than at time 0
General properties of multiprocessor scheduling (1/2)

- Exact schedulability condition
  - \( U \leq m \) and \( u_{\text{max}} \leq 1 \)
  - \( U \) = total utilization
  - \( U_{\text{max}} \) = maximum utilization
  - Does not tell for which scheduling algorithm!

- Schedule is cyclic on the hyperperiod \( H \) (PPCM\((P_i)\)) for:
  - Deterministic
  - Without memory scheduling algorithms

General properties of multiprocessor scheduling (2/2)

- Theorem [Srinavasan & Baruah, 2002]
  - Non existence of FJP (FJP+FTP) scheduling with utilization bound strictly larger than \( (m+1)/2 \) for implicit deadline periodic task sets
Global multiprocessor scheduling: detailed outline

- Transposition of uni-processor algorithms
- Extensions of uni-processor algorithms
  - US (Utilization Threshold)
  - EDF(k)
  - ZL (Zero Laxity)
- Pfair approaches (Proportional Fair)

Transposition of uni-processor algorithms (1/2)

- Main algorithms
  - RM (Rate Monotonic) ➔ G-RM, Global RM
  - EDF (Earliest Deadline First) ➔ G-EDF, Global EDF
- Not optimal anymore
- Sufficient schedulability tests (depend on $u_{\text{max}}$)

<table>
<thead>
<tr>
<th>G-RM</th>
<th>G-EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_{\text{max}} \leq m/(3m-2)$ and $U \leq m^2/(3m-2)$</td>
<td>$u_{\text{max}} \leq m/(2m+1)$ and $U \leq m^2/(2m+2)$</td>
</tr>
<tr>
<td>$u_{\text{max}} \leq 1/3$ and $U \leq m/3$</td>
<td>$u_{\text{max}} \leq 1/2$ and $U \leq (m+1)/2$</td>
</tr>
<tr>
<td>$U \leq m/2 + (1-u_{\text{max}}) + u_{\text{max}}$</td>
<td>$U \leq m - (m-1) u_{\text{max}}$</td>
</tr>
</tbody>
</table>
Transposition of uni-processor algorithms (2/2)

Extensions of global RM/EDF: US (Utilization Threshold) policies

- Priority assignment depend on an utilization threshold $\xi$
  - If $u_i > \xi$, then $T_i$ is assigned maximal priority
  - Else, $T_i$’s priority assigned as in original algorithm (RM/EDF)
  - Arbitrary deterministic tie resolution

- Remarks
  - Still non optimal,
  - Outperforms the base policy
  - Defies Dhall’s effect
Extensions of global RM/EDF: US (Utilization Threshold) policies

- Example: RM-US[$\xi=1/2$]

<table>
<thead>
<tr>
<th></th>
<th>Ci</th>
<th>Pi</th>
<th>Ui</th>
<th>Prio</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>4</td>
<td>10</td>
<td>2/5</td>
<td>2</td>
</tr>
<tr>
<td>T2</td>
<td>3</td>
<td>10</td>
<td>3/10</td>
<td>2</td>
</tr>
<tr>
<td>T3</td>
<td>8</td>
<td>12</td>
<td>2/3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>T4</td>
<td>5</td>
<td>12</td>
<td>5/12</td>
<td>1</td>
</tr>
<tr>
<td>T5</td>
<td>7</td>
<td>12</td>
<td>7/12</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

Extensions of global RM/EDF: US (Utilization Threshold) policies

- Utilization bounds

<table>
<thead>
<tr>
<th>RM-US</th>
<th>EDF-US</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\xi=m/(3m-2)$</td>
<td>$\xi=m/(2m-1)$</td>
</tr>
<tr>
<td>$U \leq m^2/(3m-2)$</td>
<td>$U \leq m^2/(2m-1)$</td>
</tr>
<tr>
<td>$\xi=1/3$</td>
<td>$\xi=1/2$</td>
</tr>
<tr>
<td>$U \leq (m+1)/3$</td>
<td>$U \leq (m+1)/2$</td>
</tr>
</tbody>
</table>

- Remarks
  - Utilization bounds do not depend on $u_{\text{max}}$ anymore
  - EDF-US[1/2] attains the best utilization bound possible for FJP
Extensions of global RM/EDF: EDF(k)

- Task indices by decreasing utilization
  - \( u_i \geq u_{i+1} \) for all \( i \) in \([1,n]\)
- Priority assignment depends on a threshold on task index
  - \( i < k \), then maximum priority
  - Else, priority assignment according to original algorithm

Example, EDF(4)

<table>
<thead>
<tr>
<th>Ci</th>
<th>Pi</th>
<th>Ui</th>
<th>Prio</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>4</td>
<td>10</td>
<td>2/5</td>
</tr>
<tr>
<td>T2</td>
<td>3</td>
<td>10</td>
<td>3/10</td>
</tr>
<tr>
<td>T3</td>
<td>8</td>
<td>12</td>
<td>2/3</td>
</tr>
<tr>
<td>T4</td>
<td>5</td>
<td>12</td>
<td>5/12</td>
</tr>
<tr>
<td>T5</td>
<td>7</td>
<td>12</td>
<td>7/12</td>
</tr>
</tbody>
</table>
Extensions of global RM/EDF: EDF(k)

- Sufficient schedulability test
  \[ m \geq (k - 1) - \left( \sum_{i=k+1}^{n} u_i \right) \frac{n}{1 - u_k} \]
  - \( k_{\text{min}} \) = value minimizing right side of the equation
  - With \( k = k_{\text{min}} \), utilization bound of \( (m+1)/2 \) (best possible for FJP)
  - Comparison with EDF[1/2]
    - Same utilization bound
    - EDF(\( k_{\text{min}} \)) dominates EDF[1/2]

Extensions of global RM/EDF: ZL (Zero Laxity) policies

- XX-ZL: apply policy XX until Zero Laxity
  - Maximal priority when laxity reaches zero (regardless of the currently running job), original priority assignment for the others
  - In category DJP (dynamic job scheduling)
- Utilization bound: \( (m+1)/2 \)
- Dominates G-EDF
Extensions of global RM/EDF: ZL (Zero Laxity) policies

- Example: m=3, m=2; all Pi to 2, all Ci to 2
  - G-EDF: T3 misses its deadline
  - EDZL: OK

Pfair algorithms

- Principle
- Construction of a Pfair schedule
- Pfair scheduling policies
Pfair algorithms: principle

- Pfair: “Proportionate Fair”
  - [Baruah et al, 1996]
  - Allocate time slots to tasks as close as possible to a “fluid” system, proportional to their utilization factor

- Example
  - $C_1=C_2=3$, $P_1=P_2=6$ ($u_1=u_2=1/2$)
  - Each task will be “approximately” allocated 1 slot out of 2 (whatever the processor)

- Lag function: difference between real and fluid execution
  - Discrete time, successive time slots $[t, t+1[$
  - Weight of a task: $\omega_i = u_i$

- Lag
  \[
  \text{lag}(T_i, t) = \omega_i t - \sum_{u=0}^{t-1} S(T_i, u)
  \]
  - First term: fluid execution
  - Second term: real execution, with $S(T_i, u)=1$ if $T_i$ executed in slot $u$, else 0

- Pfair schedule: for all time $t$, lag in interval $]-1,1[$
Pfair algorithms: principle

- Example

![Diagram showing execution domain of Pfair]

Property

- If a Pfair schedule exists, deadlines are met

Exact test of existence of a Pfair schedule

\[
\sum_{i=1}^{n} u_i \leq m
\]

- Full processor utilization!
Pfair algorithms: construction of a Pfair schedule

- Divide tasks in unity-length sub-tasks
  - Pfair condition: each subtask $j$ executes in a time window between a pseudo-arrival and a pseudo-deadline

  - Pseudo-arrival: $r(T_i^j) = \left\lfloor \frac{j}{\omega_i} \right\rfloor$

  - Pseudo-deadline: $d(T_i^j) = \left\lceil \frac{j}{\omega_i} \right\rceil$

---

Example (to be fixed)
Pfair algorithms: scheduling algorithms

- EPDF (Earliest Pseudo-Deadline First)
  - Apply EDF to pseudo-deadlines
  - Optimal only for $m=2$ (2 processors)

- PF, PD, $PD^2$
  - EPDF with non-arbitrary tie breaking rules in case of identical pseudo-deadlines
  - All of them are optimal
  - Most efficient one: $PD^2$

- Ongoing works
  - Reduce numbers of context switches and migrations while maintaining optimality

Conclusion

- Multi-processor scheduling is an active research area

- Ongoing works
  - Global multi-core scheduling
  - Semi-partitioned scheduling
  - Determining upper bounds of practical factors (preemption, migration, …)
  - Implementation in real-time operating systems