CHALMERS

Pfair scheduling

Presented by: Björn Andersson
Department of Computer Engineering
ba@ce.chalmers.se

CHALMERS

proportionate

P fair scheduling

Presented by: Björn Andersson
Department of Computer Engineering
ba@ce.chalmers.se

CHALMERS

Why is it interesting?

Real-time scheduling algorithms based on proportionate fairness offers:

- ”fairness for free”

  pfairness satisfied $\Rightarrow$ deadlines are met for the periodic scheduling problem
- optimal (uni- and multiprocessor)
- low output jitter
- peaceful coexistence of real-time and non-real-time tasks

CHALMERS

Outline

Preliminaries
- Problem statement,
- Concepts and system model
- The idea of proportionate progress

Results
- Existence of a pfair scheduling algorithm
- The algorithm PF

CHALMERS

Problem statement

Schedule on a multiprocessor a set of periodically arriving hard real-time tasks with constant execution time in order to meet deadlines

CHALMERS

Concepts and system model

- A task τ_i
- A task set $\tau = \{\tau_1,\tau_2,\ldots,\tau_n\}$
- Period T_i and execution time C_i. $(0<C_i/T_i<1)$
- Relative deadline D_i. $D_i = T_i$ (periodic scheduling problem)
- Hard deadlines
- A task arrives at t=0 for the first time (synchronous task set)
- $L=\text{lcm}(T_1,T_2,\ldots,T_n)$
  Example: lcm(2,3,6)=6, because
  2*k1 = 6
  3*k2 = 6
  6*k3 = 6

CHALMERS

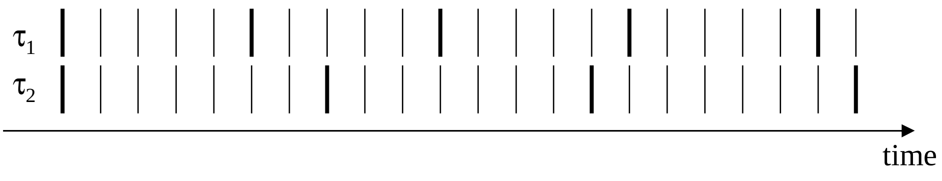
Concepts and system model

- m identical processors
- $\sum_{i=\{1,2,\ldots,n\}} C_i/T_i \leq m$
- A task can be preempted. No preemption cost.
- A task can migrate. No migration cost.
- Quantum based: $T_i \in \mathbf{Z}^+, C_i \in \mathbf{Z}^+$, scheduling decisions can only occur at integers
- A task must execute during a whole time slot or not execute in that time slot at all.

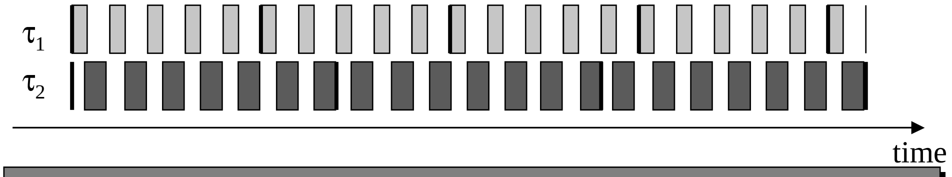
CHALMERS

The idea of proportionate progress

Problem: schedule $\tau = \{(T_1 = 5, C_1=2), (T_2 = 7, C_2=4)\}$ on 1 processor



One solution: schedule a task τ_i to execute C_i/T_i every time unit

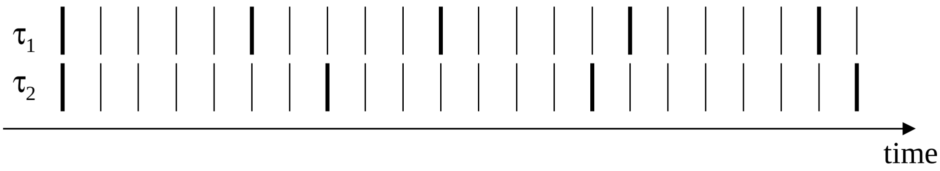


C_i/T_i every time unit $\Rightarrow$ $(C_i/T_i)*T_i$ every T_i time unit $\Rightarrow$ C_i every T_i time unit $\Rightarrow$ deadlines are met in periodic scheduling

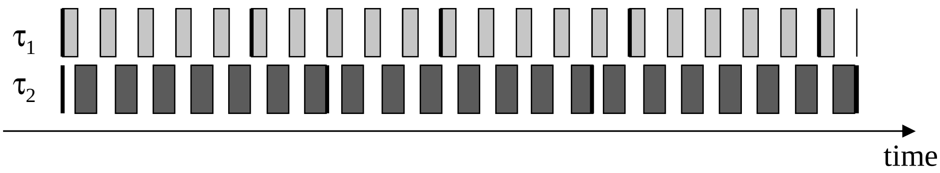
CHALMERS

The idea of proportionate progress

Problem: schedule τ = {(T_1 = 5,C_1=2), (T_2 = 7,C_2=4)} on 1 processor



One solution: schedule a task τ_i to execute C_i/T_i every time unit



Is there a schedule that only switches tasks at integer boundaries?

CHALMERS

The idea of proportionate progress

Definition

$lag(\tau_i,t) = t \bullet (C_i/T_i) - allocated(\tau_i,t)$

- $lag(\tau_i,t)$: error
- $t \bullet (C_i/T_i)$: Should have executed during [0,t)
- $allocated(\tau_i,t)$: Actually did execute during [0,t)

Consequence

τ_i executes => $lag(\tau_i)$ decreses by $1-C_i/T_i$

τ_i does not execute => $lag(\tau_i)$ increases by C_i/T_i

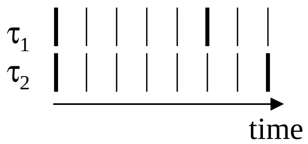
Goal

Find an algorithm that minimizes $\max_{t,\tau} |lag(\tau_i,t)|$

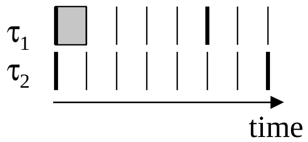
CHALMERS

The idea of proportionate progress

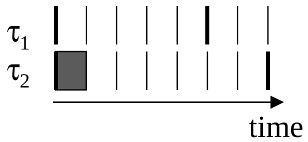
Problem: schedule $\tau = \{(T_1 = 5, C_1 = 2), (T_2 = 7, C_2 = 4)\}$ on 1 processor
switch tasks only at integer boundaries



No task executes in [0,1) =>
$lag(\tau_1,1) = 1*(2/5)-0 \neq 0$
$lag(\tau_2,1) = 1*(4/7)-0 \neq 0$



τ_1 executes in [0,1) =>
$lag(\tau_1,1) = 1*(2/5)-1 \neq 0$
$lag(\tau_2,1) = 1*(4/7)-0 \neq 0$



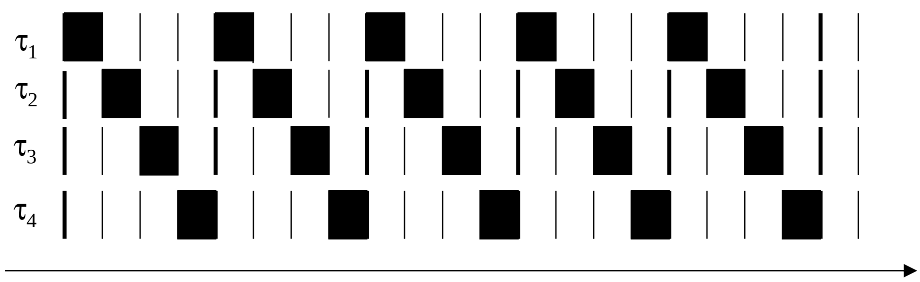
τ_2 executes in [0,1) =>
$lag(\tau_1,1) = 1*(2/5)-0 \neq 0$
$lag(\tau_2,1) = 1*(4/7)-1 \neq 0$

$lag(\tau_1,1)=0$ is impossible

CHALMERS

The idea of proportionate progress: How far from zero can the lag be?

Problem: schedule $\tau = \{(T_1 = 4, C_1 = 1), (T_2 = 4, C_2 = 1), (T_3 = 4, C_3 = 1),$
$(T_4 = 4, C_4 = 1)\}$ on 1 processor,



$lag(\tau_1,1) = 1*(1/4) - 1 = -3/4$

$lag(\tau_4,3) = 3*(1/4) - 0 = 3/4$

$-1 < lag(\tau_i,t) < 1$ seems to be the worst-case lag

CHALMERS

The idea of proportionate progress: Pfairness

Definition

A schedule is pfair iff:
forall τ_i and forall t: $-1 < lag(\tau_i,t) < 1$

Consequence

If a schedule is pfair then the schedule solves periodic scheduling.

CHALMERS

The idea of proportionate progress: Pfairness

Proof

A schedule S is pfair
$\Rightarrow -1 < lag(\tau_i,t) < 1$
$\Rightarrow -1 < lag(\tau_i,k*T_i) < 1$
$\Rightarrow -1 < k*T_i*(C_i/T_i) – allocated(\tau_i,k*T_i) < 1$
$\Rightarrow -1 < k*C_i – allocated(\tau_i,k*T_i) < 1$
$\Rightarrow k*C_i – allocated(\tau_i,k*T_i) = 0$
$\Rightarrow allocated(\tau_i,k*T_i) = k*C_i$
$\Rightarrow$ $allocated(\tau_i,(k+1)*T_i) - allocated(\tau_i,k*T_i) = C_i$
$\Rightarrow \tau_i$ executed C_i time units during $[k*T_i,k*T_i+T_i]$
$\Rightarrow$ τ_i meets every deadline in periodic scheduling

CHALMERS

Outline

Results

- Existence of a pfair scheduling algorithm
- The algorithm PF

CHALMERS

Results: existence of a pfair scheduling algorithm

Want to show

$\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) \leq m \Rightarrow$ a pfair schedule exists

Idea

1. $\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) = m \Rightarrow$ a pfair schedule exists
2. If $\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) < m$ then add a dummy task such that $\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i)$ becomes m.

CHALMERS

Results: existence of a pfair scheduling algorithm

Want to show

$\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) = m$ $\Rightarrow$ a pfair schedule exists

Idea

$\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) = m \Rightarrow$ a pfair schedule exists during [0,L) $\Rightarrow$ a pfair schedule exists

CHALMERS

Results: existence of a pfair scheduling algorithm

Want to show

$\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) = m$ $\Rightarrow$ a pfair schedule exists during [0,L)

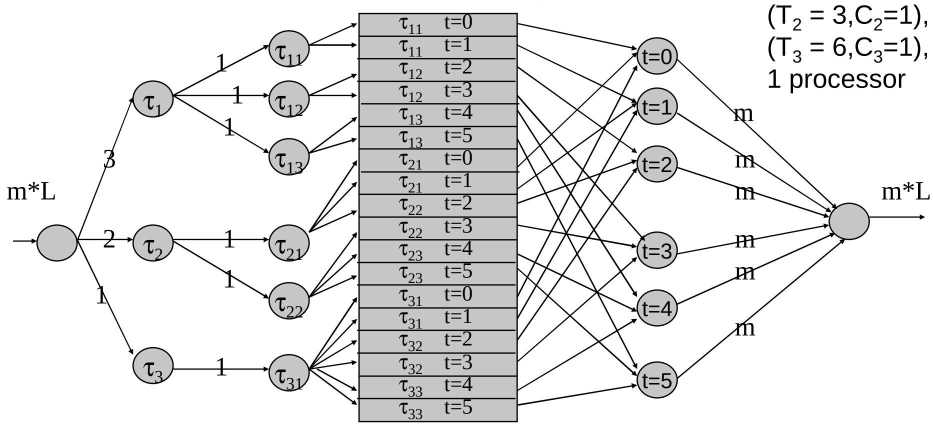
Idea

$\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) = m \Rightarrow$ an integer solution to the network flow exists $\Rightarrow$ a pfair schedule exists during [0,L)

CHALMERS

Results: existence of a pfair scheduling algorithm

(T_1 = 2,C_1=1),
(T_2 = 3,C_2=1),
(T_3 = 6,C_3=1),
1 processor



An integer solution to the network flow exists
⇒ a pfair schedule exists during [0,L)

CHALMERS

Results: existence of a pfair scheduling algorithm

Want to show

$\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) = m$ ⇒ an integer solution to the network flow

Idea

$\sum_{i=\{1,2,\ldots,n\}} (C_i/T_i) = m$ ⇒ a fractional solution to the network flow ⇒ an integer solution to the network flow

CHALMERS

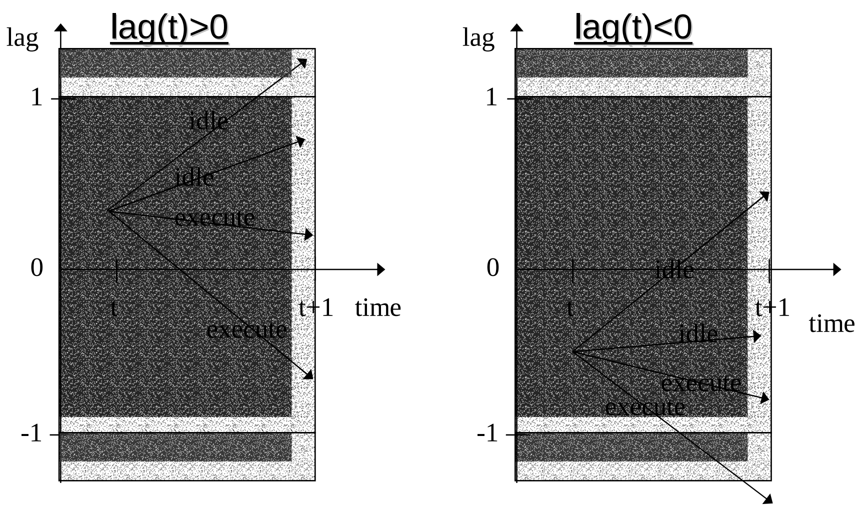
Results: The algorithm PF

The algorithm PF assigns priorities to tasks at every time slot. (dynamic priority)

Theorem:
The schedule generated by algorithm PF is pfair.
Proof:
Baruah et al. Algorithmica'96

CHALMERS

Results: The algorithm PF



CHALMERS

Results: The algorithm PF

- Execute all *urgent* tasks.
  A task τ_i is urgent at time t if
  lag(τ_i,t)>0 and
  lag(τ_i, t+1)≥0 if τ_i executes.
- Do not execute *tnegru* tasks.
  A task τ_i is tnegru at time t if
  lag(τ_i,t) < 0 and
  lag(τ_i,t)≤0 if τ_i does not execute.
- For the other tasks, execute the task that have the least t>now such that lag(τ_i,t)>0.

CHALMERS

Improvements

A new task model: intra-sporadic task

- Intra-sporadic task ⇒ a pseudotask can arrive whenever the previous pseudotask has completed
- Intra-sporadic tasks can be used to schedule sporadic and asynchronous task sets
- PD^2 preserves optimality for these problems
- For details see: Srinivasan and Anderson, STOC'2002

CHALMERS

Conclusion

The notion of pfairness can be used to design optimal real-time scheduling algorithms for multiprocessors.