Inovace tohoto kurzu byla v roce 2011/12 podpořena projektem
CZ.2.17/3.1.00/33274 financovaným Evropským sociálním fondem
a Magistrátem hl. m. Prahy.

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Embedded and Real-Time Systems

Multi-Processor Scheduling

Acknowledgement

- This lecture has been almost entirely built based on an excellent overview paper
  - Robert I. Davis, Alan Burns: “A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems”

Multi-processor scheduling

- Much harder than for uniprocessor
  - *“Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. 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.”*

C. L. Liu: “Scheduling algorithms for multiprocessors in a hard real-time environment”

Prerequisitities and definitions

- We will assume:
  - Homogeneous processors – all the processors are identical

- Migration
  - No migration
    - Each task is allocated to a processor without possibility of further migration
  - Task-level migration
    - Jobs of a task may execute on different processors; each job can only execute on a single processor
  - Job-level migration
    - A single job can migrate to and execute on different processors; parallel execution is however not permitted

Definitions

- Scheduling algorithms
  - Partitioned – no migration permitted
  - Global – task or job level migration permitted

- Deadlines
  - Implicit deadlines (deadline equals to period)
  - Constrained deadlines (deadline is less than or equal to period)
  - Arbitrary deadlines (less than, equal to, or greater than period)

Theoretical results

- There is a $O(N^3)$ algorithm that is able to determine an optimal multiprocessor schedule for any arbitrary set of completely determined jobs where all of the arrival and execution times are known a priori.
  - N … number of jobs
- No online optimal algorithms exists for sporadic tasks with constrained or arbitrary deadlines
  - Knowledge of arrival times is necessary
- In general for offline algorithms, then necessary and sufficient condition is $u_{sum} \leq m$
  - u_{sum} … taskset utilization
  - m … number of processors

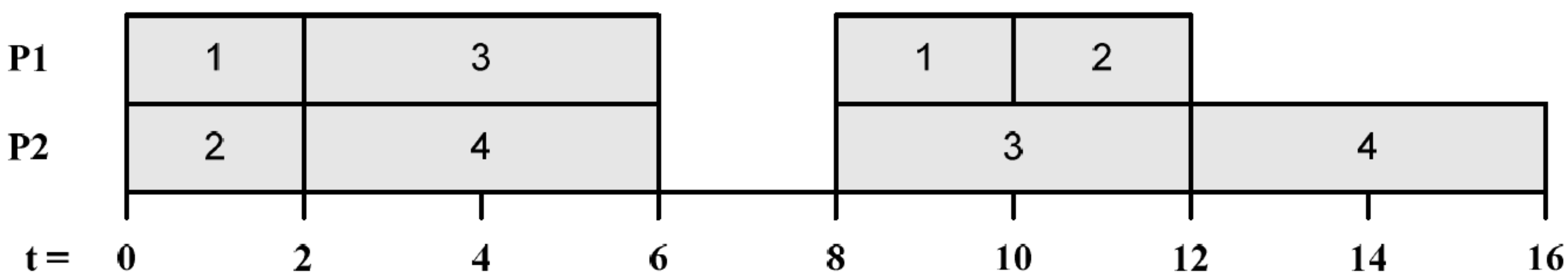
Maximum utilization bounds

- For tasks with implicit deadlines, maximum utilization bounds are
  - Global (job-level migration), dynamic priority … m
  - All other classes … $(m+1)/2$

  - Equation $(m+1)/2$ holds because $m+1$ tasks with execution time $1+\epsilon$ and a period of 2 cannot be scheduled on m processors regardless of theallocation algorithm used.

Critical instant effect

- Under global fixed task priority scheduling, a task does not necessarily have its worst-case response time when released simultaneously with all higher priority tasks
  - This is a fundamental difference to uniprocessor scheduling



  - The critical instant in the example above does not yield the longest schedule

Partitioned scheduling

- Division of tasks to processors
  - Uniprocessor scheduling on each processor
- Advantages compared to global scheduling
  - Task overruns affect only tasks on one processor
  - No migration cost
  - Separate run-queues (less overhead compared to manipulating one global queue)

Partitioned scheduling

- The allocation of tasks to processors is NP-Hard
  - Bin packing problem
- Allocation heuristics
  - First-Fit (FF), Next-Fit (NF), Best-Fit (BF), Worst-Fit (WF), etc.
  - Combining algorithms with heuristics (e.g. RMNF)

Algorithm	Approximation Ratio ($\Re_A$)
RMNF	2.67
RMFF	2.33
RMBF	2.33
RM-FFDU	5/3
FFDUF	2
RMST	$1/(1-u_{\max})$
RMGT	7/4
RMMatching	3/2
EDF-FF	1.7
EDF-BF	1.7

- Approximation ratio shows asymptotic relation in required number of processors w.r.t to optimal case

Global scheduling

- Global queue, job migrations

- Advantages
  - Typically fewer context switches – scheduler preempts a task only when there are no idle processors
  - Spare capacity created when a task executes for less than its WCET can be used by all tasks (not just those on one processor)
  - When task overruns, there is arguably lower probability that it will cause deadline misses
  - More appropriate for open systems (non need for task allocation when task set changes)

Dhall's effect

- Theoretically, global scheduling suffers from Dhall's effect

- Consider the following task set:

$$\tau = \{\tau_1 = (2\epsilon, 1), \tau_2 = (2\epsilon, 1), \ldots, \tau_m = (2\epsilon, 1), \tau_{m+1} = (1, 1+\epsilon)\}$$

  - m … number of processors

- Under online global scheduling (e.g. EDF or RM), the utilization bound is $1 + \epsilon$
  - For example if priorities are assigned by RM or EDF, task τ_{m+1} misses the deadline regardless the number of processors

Global scheduling algorithms

- Fixed job priority scheduling
  - Global EDF
  - EDF-US[x]
    - EDF + tasks with utilization higher than x get the highest priority (ties broken arbitrarily)
  - EDF(k)
    - EDF + k tasks with highest utilization get the highest priority
- Fixed task priority scheduling
  - Global RM, Global DM, RM-US
- Dynamic priority scheduling
  - Fluid algorithms (Pfair, LLREF, ...)
  - Ensure fairness and thus yield optimality
  - High overhead

Pfair algorithm

- Proportional fairness (P-fairness)
  - Each task τ_i is assigned resources in proportion to its weight $W_i = C_i / T_i$ hence it progresses proportionately
  - At every time t, task τ_i must have been scheduled either $\lfloor W_i \times t \rfloor$ or $\lceil W_i \times t \rceil$ time units
  - Preemption is assumed to only occur at integral time units
  - Task model is periodic with implicit deadlines.

Hybrid approaches

- Global scheduling can have large overhead
- Migration of tasks may be also very costly

- It is possible to group small numbers of processors and allocate tasks to them
  - aka clustering