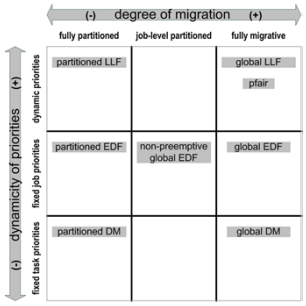
Semi-partitioned multiprocessor scheduling

Alessandra Melani

Traditional classification

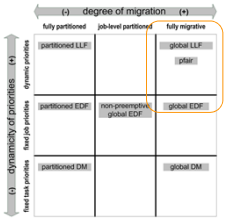
- Multiprocessor scheduling schemes, two orthogonal classification criteria [1]:
  1. **Prioritization** scheme
     - Fixed
     - Job-static
     - Job-dynamic
  2. **Migration** scheme
     - Fully partitioned
     - Migration at job boundary
     - Fully migrative



[1] Carpenter et al., "A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms", 2004

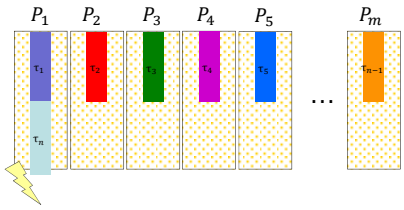
Traditional classification

- Utilization bound $UB_{\mathcal{A}}$ of algorithm $\mathcal{A}$
  - Threshold such that if $U < UB_{\mathcal{A}}$, then the system is guaranteed to be schedulable by $\mathcal{A}$
  - If $U > UB_{\mathcal{A}}$, the system is not necessarily unschedulable by $\mathcal{A}$
- High **utilization bounds** only in the top-right
  - P-fair: **100%** system utilization
- All partitioned scheduling algorithms have a utilization bound of **50%** or less



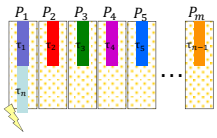
Partitioned scheduling UB

- All partitioned scheduling algorithms have a utilization bound of **50%** or less
- m processors, $n = m + 1$ tasks, with $T_i = 1$ and $C_i = 0.5 + \varepsilon$
- Since $n > m$, one processor is assigned ≥ 2 tasks, hence its utilization will be at least $1 + 2\varepsilon > 1$



Semi-partitioned scheduling

- In this system there is plenty of **idle time**, but cannot be exploited by any partitioned scheduling strategy



- By allowing task migrations, we could achieve 100% system utilization
- But, ideally, we would like to have high UB without too many preemptions, migrations and globally shared data structures $\Longrightarrow$ **semi-partitioned** scheduling
- Find a balance point between partitioned and global scheduling

Semi-partitioned scheduling

- Before run-time, we need to establish:
  - Which tasks are partitioned and which are allowed to migrate
  - Processor(s) where each task executes
  - Precise logic controlling task migrations
- By design, **partitioning is favored** and **migrations are disfavored**:
  - Produce a partitioned system, if possible
  - "Most" tasks are partitioned, "few" migrate
  - Migration limited between few processors

Approaches to semi-partitioning

1. **Slot-based / server-based** approaches
   - High-level repeating schedule for servers, mapped on the processors
   - High UBs (75%-100%), at least theoretically
2. **Timed Job migration-based** approaches
   - Migration at predetermined time offsets from task arrival
   - UBs of 72%-75% at most
   - In practice: fewer preemptions/migrations

Slot-based semi-partitioning

- Time divided into intervals called **time-slots**
- A high-level schedule is generated for one time-slot
- The pattern repeats in subsequent ones
- The time-slot on each processor is subdivided into *time reserves* (a simple form of server) for one or more tasks
- Within each reserve: EDF scheduling
- Example: **EKG-Periodic** [2]

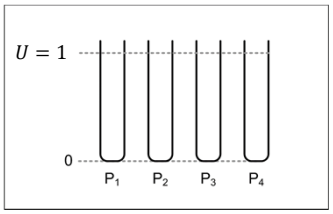
[2] B. Andersson, E. Tovar, "Multiprocessor Scheduling with Few Preemptions", RTCSA 2006

EKG-Periodic

- Stands for "**E**DF with task splitting and **K** processors in a **G**roup"
- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%
- Task assignment: processors are filled one by one, "splitting" tasks as necessary
  - At most $m - 1$ split tasks in the system
  - Split tasks use two adjacent processors each

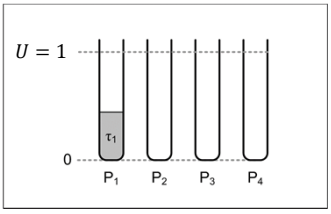
Task assignment illustrated

- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%



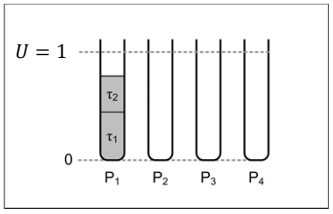
Task assignment illustrated

- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%



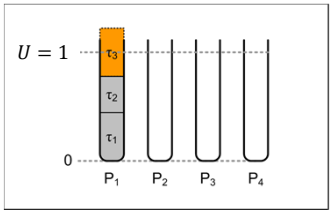
Task assignment illustrated

- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%



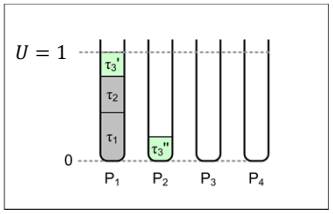
Task assignment illustrated

- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%



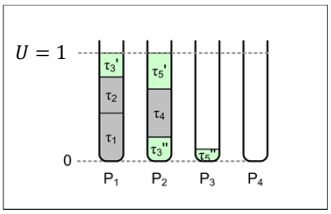
Task assignment illustrated

- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%



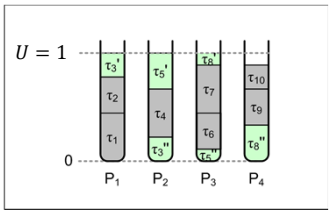
Task assignment illustrated

- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%



Task assignment illustrated

- For periodic, implicit deadline ($D = T$) tasks
- UB = 100%

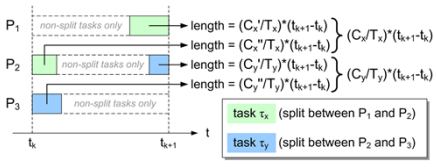


EKG-P: observations

- **Observation 1**: The deadline of a job always coincides with the arrival of the next job by the same task
  - Since tasks are periodic and implicit-deadline
- **Observation 2**: We can calculate in advance the time of the next task arrival in the system
  - Since tasks are periodic and all arrive at t=0
- **Key idea**: Between any two successive arrivals (not necessarily by the same task), split tasks execute proportionally to their utilizations
  - "Relaxed" proportional fairness ⟹ split task deadlines met
  - But: split tasks should execute on one processor at a time

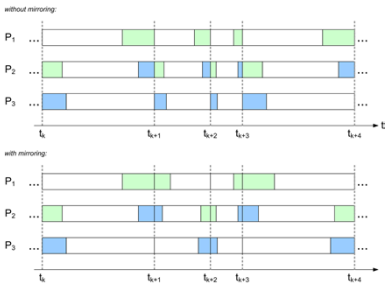
EKG-P: between successive arrivals

- Slot length equal to interval between consecutive job arrivals (not necessarily by the same task)
- At run-time, reserves for split tasks on different processors are temporally non-overlapping by design



EKG-P: the mirroring trick

- Allows saving some preemption costs



EKG: limitations

- By design, it cannot handle **sporadic** tasks
  - Time slot length computed as time interval between consecutive job arrivals
  - With sporadic arrivals, this information is unknown / unpredictable
- When two successive task arrivals occur too close in time, rapid context-switching for little execution occurs
- Both aspects remedied by EKG-Sporadic [3], at the cost of some utilization loss
  - Processors can no longer be filled up to 100%

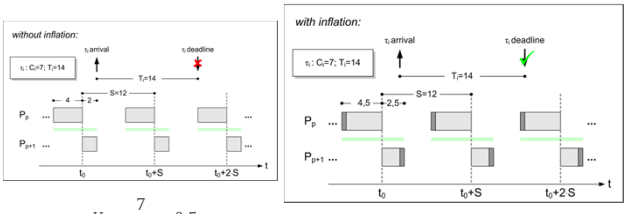
[3] B. Andersson, K. Bletsas, "Sporadic Multiprocessor Scheduling with Few Preemptions", ECRTS 2008

EKG-Sporadic

- **Fixed-length** time-slots: $S = \frac{T_{min}}{\delta}$
  - Integer parameter δ controls migration frequency
- Similar task assignment as EKG-P, with one difference:
  - **Heavy tasks**, with $U_i > SEP = 4\left(\sqrt{\delta(\delta+1)} - \delta\right) - 1$, get their own processor
  - Remaining **light tasks** assigned on next available processor whose utilization is $< SEP$
  - Each processor is filled by light tasks up to SEP; not up to 100% as before (tasks can arrive at "unfavorable" times)
- UB configurable between 65% and ~100% (at the cost of more preemptions and migrations)

Reserve inflation

- Reserves must be "inflated" to compensate for potentially unfavorable arrival phasing

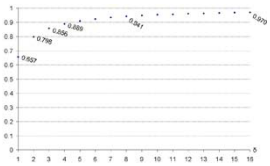


$$U_i = \frac{7}{14} = 0.5$$

τ_i gets 6 units of budget at every slot

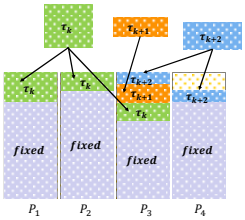
Utilization bound of EKG-S

- Reserve inflation brings a **utilization penalty**, but it is the price of flexibility to handle sporadic tasks
  - The resulting UB is SEP
  - ~65% for $\delta = 1$; ~100% for $\delta \longrightarrow +\infty$
- Utilization penalty is reduced by higher δ (shorter time slots) but at a cost of more frequent migrations



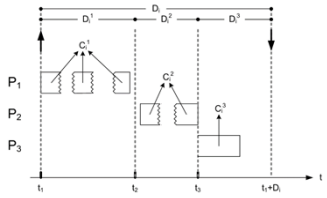
Timed job migration semi-partitioning

- Objective: fewer preemptions / migrations with respect to timeslot-based semi-partitioning
- Tasks assigned to processors according to a given heuristic
  - If remaining capacity is not enough to accept the full share of the task, the task is decided to be "**migratory**" ("**split**" into more than one processor)



Timed job migration semi-partitioning

- A "split" task starts executing on one processor and migrates to one or more other processors at pre-determined time offsets measured from its arrival
  - Always the same processors and always the same offsets, for all jobs by the same task
  - The split task can be modelled as separate sub-tasks, with **precedence constraints** and **intermediate deadlines**



Sub-task parameters

- Different heuristics can be devised for task splitting according to these principles
  - Sub-task parameters: C_i and D_i
- These heuristics, along with other aspects (e.g., bin-packing) affect the average-case performance but also the utilization bound that can be proven for the algorithm
- Examples: EDF-WM [4], C=D [5]

[4] S. Kato, N. Yamasaki, Y. Ishikawa, "Semi-partitioned scheduling of sporadic task systems on multiprocessors", ECRTS 2009

[5] A. Burns, R. Davis, P. Wang, F. Zheng, "Partitioned EDF Scheduling for Multiprocessors using a C=D scheme", RTSJ 48(1), 2011

EDF-WM

- Stands for "EDF with Windowed Migration"
  - Handles **sporadic** tasks
- Assigns tasks integrally (partitioning) by default
- It only splits a task when its partitioning fails
  - Dominates Partitioned EDF
- Attempts to split in as few pieces as possible
  - Starting from $k = 2, 3, \ldots$ until success (or failure at $k > m$)
- Defining characteristic: sub-tasks of split task have equal relative deadlines D_i/k

EDF-WM: details

- The WCET of each sub-task (except the last one) is determined according to **sensitivity analysis**
  - Maximum value that does not render the processor unschedulable, according to exact EDF schedulability test (dbf-based)
- This calculation needs to be repeated when trying a new value of k (number of pieces) because the sub-task relative deadlines change

EDF-WM: algorithm

- $k = 2$ (number of sub-tasks, initialization)
- For each processor calculate maximum execution time C for sub-task
- Consider processors by decreasing C calculated above (without loss of generality P_p to P_{p+k-1})
- Can we split τ_i in k sub-tasks ($\tau_i^{(1)}$ to $\tau_i^{(k)}$) with $D = D_i/k$ and $T = T_i$ such that
  - $C_i^{(1)}$ = max value such that P_p remains schedulable AND
  - $C_i^{(2)}$ = max value such that P_{p+1} remains schedulable AND
  - …
  - $C_i^{(k)} = C_i - \left(\sum_{j=1}^{k-1} C_i^{(j)}\right)$ and P_{p+k-1} remains schedulable?
- If yes, split like this, else increment k and repeat

C=D scheduling algorithm

- Different sub-task formation heuristic
- Each sub-task (except perhaps the last one) has its C equal to its D
  - Equivalent to running at maximum priority (non-preemptively)
- Efficient at use of remaining processor utilization
  - Provides the maximum possible time on the next processor for the task to complete the remainder of its computation time
- **Zero-laxity** sub-task runs continuously until migration

Comparison of EDF-WM and C=D

- C_i^1 = max such that its processor remains schedulable, with $D_i^1 = D_i/k$, $T_i^1 = T_i$
- …
- C_i^{k-1} = max such that its processor remains schedulable, with $D_i^{k-1} = D_i/k$, $T_i^{k-1} = T_i$
- $C_i^k = C_i - (\sum_{j=1}^{k-1} C_i^j)$, $D_i^k = D_i/k$, $T_i^k = T_i$

- C_i^1 = max such that its processor remains schedulable, with $D_i^1 = C_i^1$, $T_i^1 = T_i$
- …
- C_i^{k-1} = max such that its processor remains schedulable, with $D_i^{k-1} = C_i^{k-1}$, $T_i^{k-1} = T_i$
- $C_i^k = C_i - (\sum_{j=1}^{k-1} C_i^j)$, $D_i^k = D_i - (\sum_{j=1}^{k-1} D_i^j)$, $T_i^k = T_i$

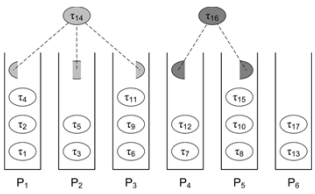
If the kth piece is not schedulable attempt splitting the task to $k+1$ pieces, with the task parameters described (respectively, under each algorithm) as above

C=D assignment splitting strategies

- Various strategies, some can be combined
  - **Continuous**: fills up a processor, then split a task between that processor and the next one
  - **Preselection**: partitioning as many tasks as possible, then split remaining tasks
  - **Interleaved**: split a task into zero-laxity sub-task and rump sub-task; throw rump sub-task back into bucket of unassigned tasks
- At most **one zero-laxity sub-task per processor**!
- Other tasks are scheduled as background workload under EDF on their respective processors

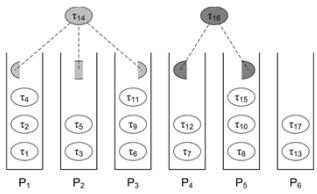
Utilization bound of C=D

- Good performance but none of those variant is easy to reason about, in terms of UB
- Simpler, "clustered" variant with at most $m/2$ split tasks has been devised for that purpose



Utilization bound of C=D

- UB of Clustered C=D is $13/18$ $(= 72.22\%)$
- Average case performance close to that of original variants
- Over all possible task assignment splitting strategies, it has been shown that the UB of C=D cannot exceed 75%



Conclusions

- **Semi-partitioned scheduling** establishes a balance point between partitioned and global scheduling
- **Time-slot based** semi-partitioning
  - Repeating schedule for servers
  - Theoretically high UBs $(75\% - 100\%)$
- **Timed-job-migration** semi-partitioning
  - Efficient to implement
  - Preemption- and migration-light
  - Competitive in terms of performance with slot-based semi-partitioning

Thank you!

Alessandra Melani
alessandra.melani@sssup.it