Real-Time Scheduling for Multi-Processors Systems

Frank Singhoff
Laurent Pautet
strec.wp.mines-telecom.fr

Version 1.0

Architecture Issues
No Mono-Processor Architecture Anymore

- Historically … mono-processors
  - platform = a dedicated processor, a clock and a common memory …
  - predictable (cache and pipeline inhibited)
  - no longer common technology, limited performance
- Trends … multi-processors
  - Use COTS (not dedicated) processors (FAA, 2011).
  - Shared resources => +interferences ; -predictable
  - More powerful, but less predictable
    (cannot inhibit interconnection bus)

Architecture Issues
Interferences with Multi-Processors

- Let’s have task T_1 (resp T_2) running on core C_1 (resp C_2) ; C_1 and C_2 share a common cache L_2 or an interconnection bus
- T_1 and T_2 are functionally independent ... but finally dependent because of shared hardware resources inducing interferences
- A task can be delayed due to contention / interference on shared hardware
- This can be an even more important problem in multi-processors than in mono-processor

Multi-Processors Architecture
Processors

- Identical processors: processors all executing the same units of work during the same units of time
- Uniform processors: processor j with speed s_j executes $s_j.t$ units of work for t units of time.
- Heterogeneous processors: processor j executes $s_{i,\,j}.\ t$ units of work of job i for t units of time.
- Heterogenous processors : no shared memory, nor migration (a distributed system)

Multi-Processors Architecture Scheduling

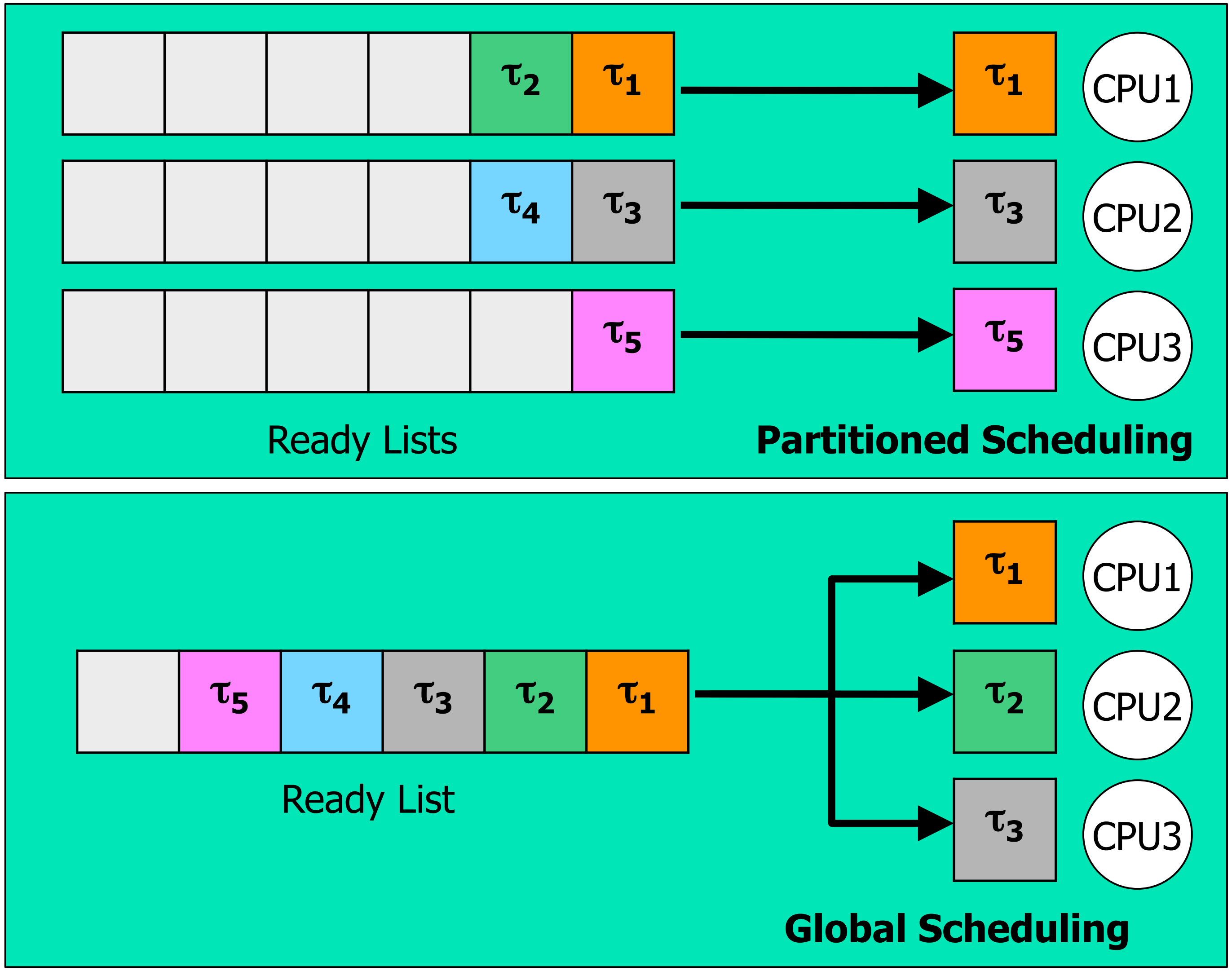
- Mono-Processor scheduling : 1 problem
  - Time Allocation – when to execute a task
- Multi-Processors scheduling : 2 problems
  - Processor Allocation – where to execute a task
  - Time Allocation – when to execute a task

As a consequence, most results from
mono-processor real-time scheduling theory
are no longer true for
multi-processors real-time scheduling theory

Multi-Processors Scheduling
Different Approaches

- Partitioned scheduling (offline processor allocation)
  - Handle separately processor and time allocations
    - Map all tasks on processors
    - Schedule tasks on each processor.
  - Possible end-to-end delay verification
- Global scheduling (online processor allocation)
  - Handle globally processor and time allocations
  - Pick a task from a global ready list
  - Map it on one of the idle processors
- Hybrid scheduling (mixed approach)
  - Offline allocation of tasks to Virtual Processes (servers)
  - Online scheduling of Virtual Processes (and tasks as well)

Scheduling Approaches
Partitioned Scheduling Approach



Partitioned Scheduling
Task Assignment

- How to statically assign tasks to processors
- Bin-packing problem: minimize the number of bags to pack bins of different volumes
- NP-hard problem => partitioning heuristics
  - Different parameters:
    - Processors (identical or not), tasks (periods, budgets), etc.
    - Task communications, shared resources, etc.
  - Different objective functions:
    - Minimize processors, communications, latencies, etc.
- Difficult to compare heuristics
  - Especially when the final objective is actually schedulability

Partitioned Scheduling
Assignment and Scheduling Variants

- Sort tasks before packing
  - Ascending/descending order of utilization/period
- Select a mono-processor scheduling
  - RM or DM, EDF or LLF
  - Schedulability test to allocate a task to a processor
- Select a bin-packing heuristic
  - First-Fit, Next-Fit, Worst-Fit or Best-Fit

Partitioned Scheduling
Rate-Monotonic Next-Fit

- List tasks in ascending order of their utilization/period.
- Processor p=0
- For task t=0 to n
  - Assign task t to processor p if the feasibility test is met (eg: $U \leq 0.69$ or response time computation)
  - Stop when no processor found
  - Loop to next processor p = (p+1) mod m

Partitioned Scheduling Limitations

- Partitioned Scheduling cannot be optimal
- m processors
- (m+1) tasks of parameters (C, T), $C=T/2+\varepsilon$
- Exercice : Prove that for periodic tasksets with implicit deadlines, the largest worst-case utilization bound for any partitioning algorithm is $(m+1)/2$.

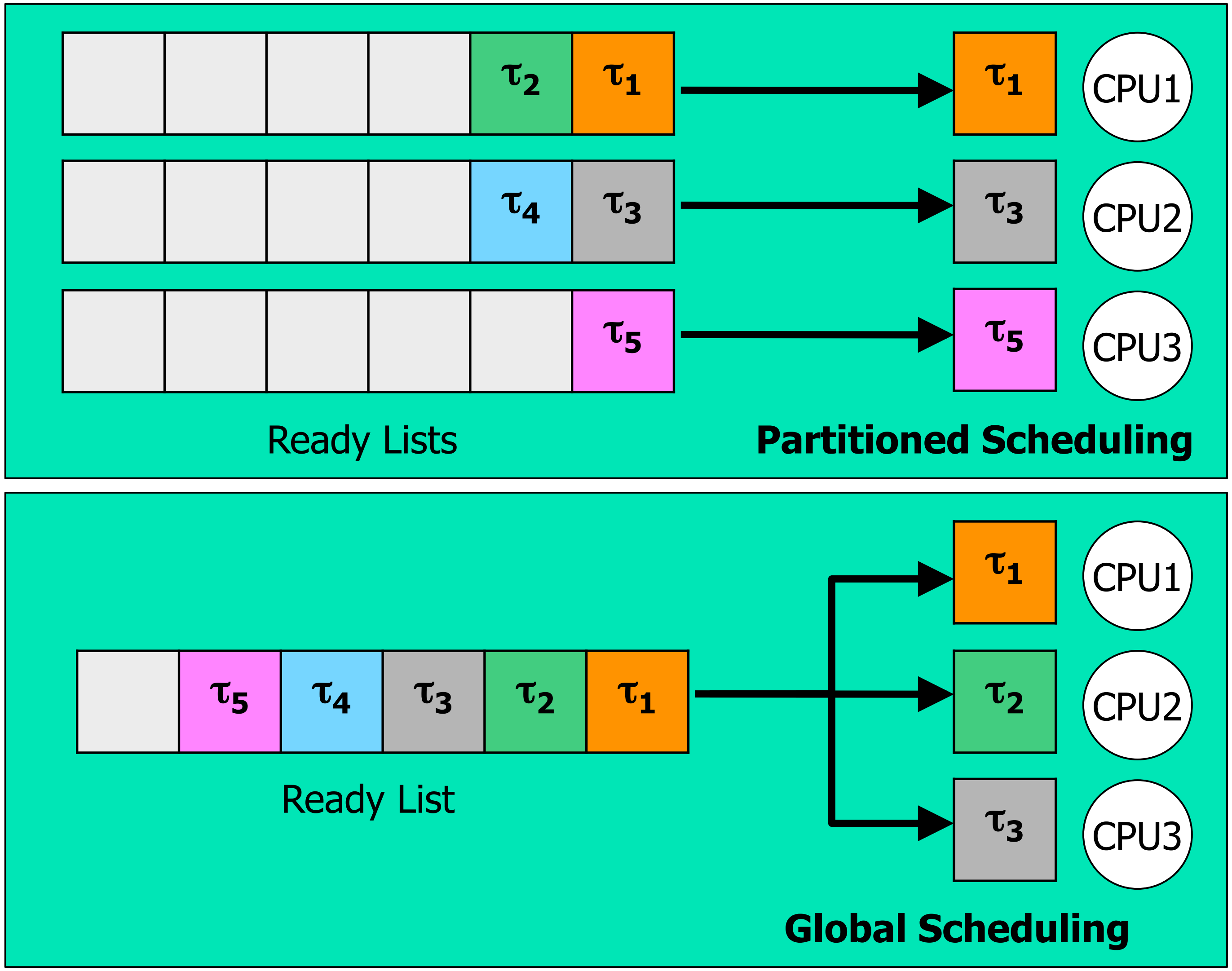
Partitioned Scheduling
Pros and Cons

- Pros
  - Better suitability for heterogeneous systems
  - Inherit from mature mono-processor scheduling
  - Time and space isolation (major safety property)
    - Failures / anomalies limited to one processor
- Cons
  - 2 problems both being NP-hard
    - Processor allocation (mapping)
    - Time allocation (scheduling)
  - Less optimal use of resources (idle processors)

Partitioned Scheduling
Other resources (memory, bus, …)

- Similar benefits/limitations for other resources
  - resource partitioning and
  - resource sharing
- Resource partitioning : great predictability … but resources less efficiently used
- Global resource sharing: poor predictability ... but resources more efficiently used
- Example: partitioned cache vs shared cache
  - Partition too small: time to reload data
  - Partition too large: waste of resource

Scheduling Approaches
Global Scheduling



Global Scheduling Pros and Cons

- Pros
  - Optimal scheduling exist
  - Better suited for homogeneous multi-core architectures
  - Better resource optimization : busy cores, less preemptions ... but migrations
- Cons
  - Not well suited at all to heterogeneous systems
  - More recent and less numerous results of scheduling theory
  - … for simple architectures and task models

Global scheduling Sharing resources

- A global scheduler deals with two problems:
  - When and how to assign task / job priorities.
  - Choose a processor on which to run the task.
- Sharing time
  - Preemption (same as mono-processor)
  - A job starts its execution in a time interval and ends in another time interval
- Sharing processors
  - Migration
  - A job starts its execution on a processor and ends on another processor

Global Scheduling
Migrations and Priorities

Migration strategies

- No task migration: All its jobs are assigned to a given processor => partitioning
- Task migration: Jobs can start executing on different processors but complete on their selected processor
- Job migration: A job can migrate during its execution.

Priority assignments

- Fixed priority associated to a task (eg: RM).
- Fixed priority associated to a job (eg: EDF).
- Dynamic priority associated to a job (ex: LLF).

Global Scheduling
Two general approaches

Mono-processor based global scheduling:

- Global RM, Global DM, Global EDF, Global LLF, ...
  - Variants depending on migration level (task or job)
- Globally apply a mono-processor scheduling strategy on all processors. Assign the *m* highest priority tasks or jobs to the *m* processors at any time.
- Task or job preemption when all processors are busy

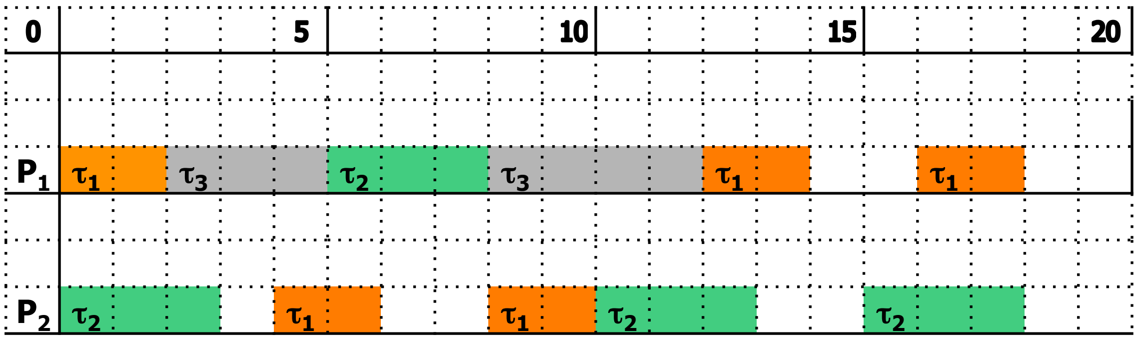
New algorithms: PFair, RUN, ...

Different and fewer results and properties compared to mono-processor scheduling

Mono-Processor based Global Scheduling
Different response times

- Use of Global Deadline Monotonic scheduling
- Priority assignment: $\tau_1 > \tau_2 > \tau_3$
- Tasks can migrate, jobs cannot

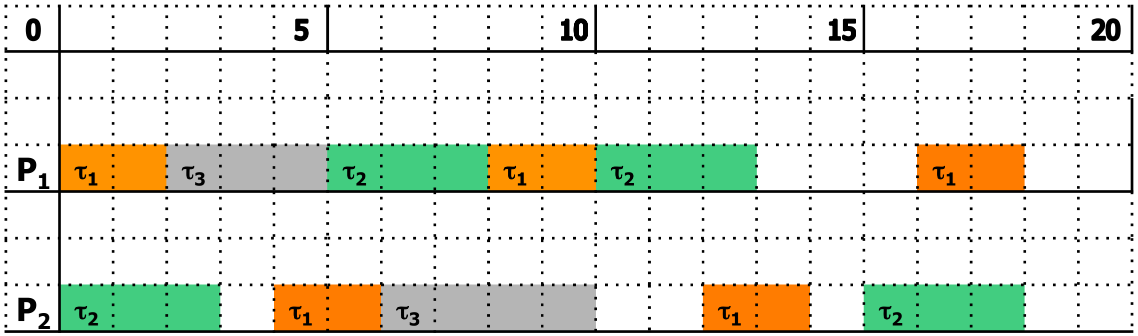
	C	T	D
τ_1	2	4	4
τ_2	3	5	5
τ_3	7	20	20



Mono-Processor based Global Scheduling Different response times

- Use of Global Deadline Monotonic scheduling
- Priority assignment: $\tau_1 > \tau_2 > \tau_3$
- Jobs can migrate
- Not the same response time for τ_3

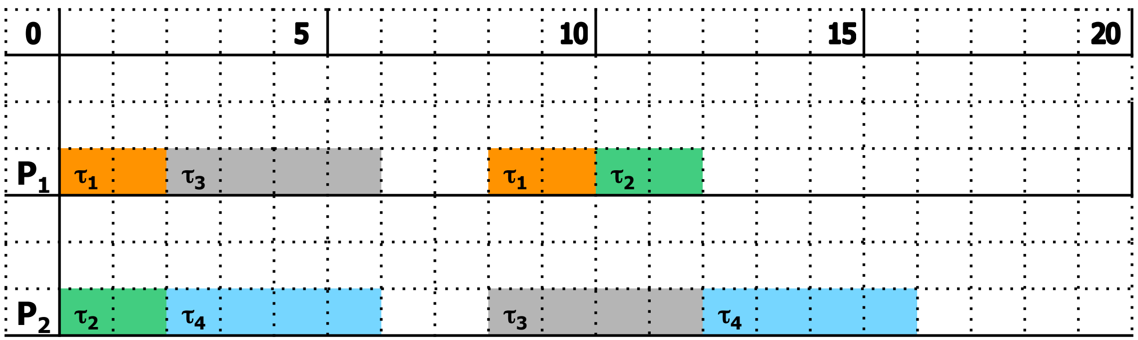
	C	T	D
τ_1	2	4	4
τ_2	3	5	5
τ_3	7	20	20



Mono-Processor based Global Scheduling No Critical Instant

- In a mono-processor, the critical instant is the worst case scenario for periodic tasks
- All tasks are released at the same instant
- Used to compute the worst response time
- But not the worst scenario in multi-processors.
- Here, R4=8 but with critical instant R4=6

	C	D	T
τ_1	2	2	8
τ_2	2	4	10
τ_3	4	6	8
τ_4	4	8	8



Mono-Processor based Global Scheduling Different feasibility interval

- In mono-processor, the feasibility interval is used to check schedulability of independent asynchronous / synchronous periodic tasks, $\forall i: D_i \leq P_i$ with a fixed priority scheduling $[0, 2 * LCM(\forall i: P_i) + \max(\forall i: S_i)]$
- In multi-processors, a similar result: $[0, LCM(\forall i: P_i)]$ but for a set of independent synchronous periodic tasks only

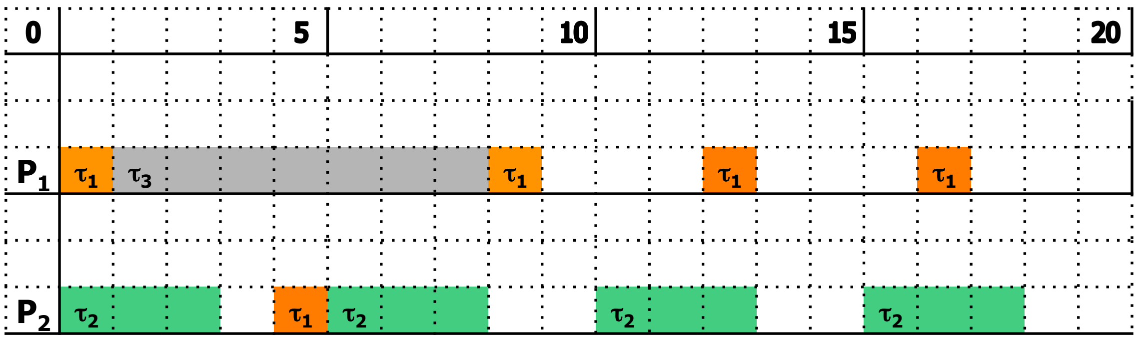
Mono-Processor based Global Scheduling
Scheduling anomalies

- Anomaly: intuitively positive change in a schedulable set of tasks that leads to a non-schedulable set of tasks
- In mono-processor, when a tasks set is schedulable, it is still schedulable if we lower its utilisation (reduce C_i or increase T_i)
- In multi-processor, this is no longer true

Mono-Processor based Global Scheduling
Scheduling anomalies

- Use of Global Deadline Monotonic Scheduling
- Jobs can migrate
- $U_1=1/4$
- Tasks set is schedulable

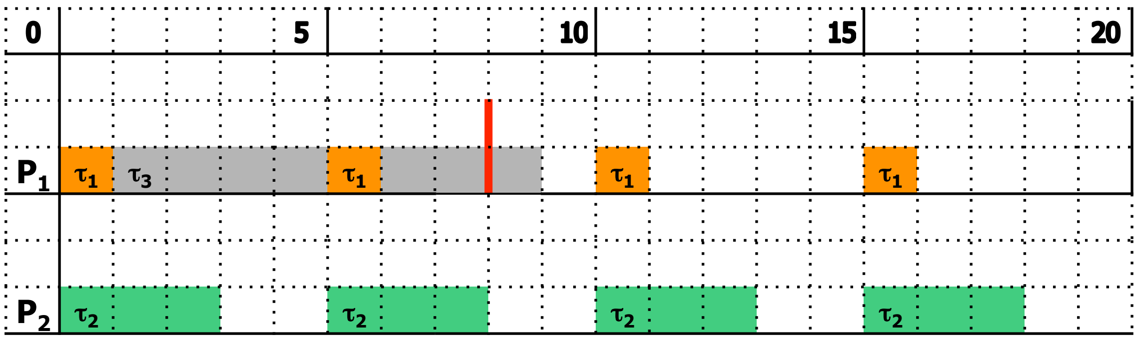
	C	T	D
τ_1	1	4	2
τ_2	3	5	3
τ_3	7	20	8



Mono-Processor based Global Scheduling
Scheduling anomalies

- Use of Global Deadline Monotonic Scheduling
- Task τ_1 has a larger period
- Task set with a lower utilisation (1/4 -> 1/5)
- Tasks set is non-schedulable ($R_3=9 > D_3$)

	C	T	D
τ_1	1	5	2
τ_2	3	5	3
τ_3	7	20	8



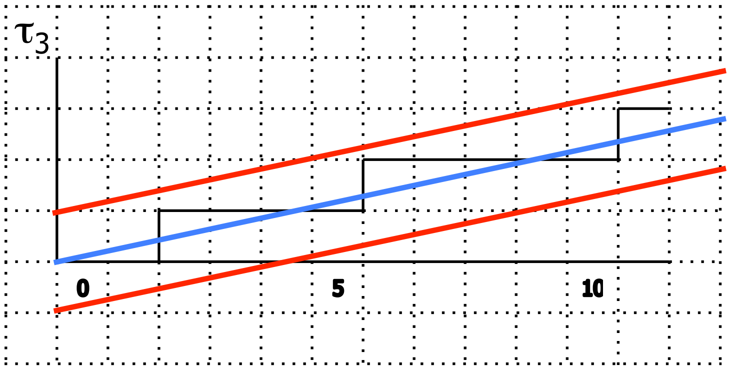
Mono-Processor based Global Scheduling Limitations

- m processors
- (m+1) tasks of parameters (C, T), C=T/2+ε
- Exercice : Prove that the maximum utiliza-tion bound for any global fixed job priority algorithm is $(m+1)/2$.
- Global LLF (dynamic priority per job) > Global EDF (static priority per job)

Global Scheduling
Pfair Algorithms : Principles

- The proportion of time units allocated at instant t to a task must remain as close as possible to its utilisation
- Optimal algorithm for identical processors and synchronous deadline implicit periodic tasks
- Lots of preemptions and migrations

	C	T
τ_1	1	2
τ_2	1	3
τ_3	2	9



Global Scheduling
Pfair Algorithms : Modelling

- Execute tasks at a constant rate (fluid model) such as $\forall i$: workload(τ_i, t) = t * C_i / T_i
- Can be approximated by sched(τ_i, t) where sched(τ_i, t) = 1 when τ_i is scheduled in interval [t, t + 1[, sched(τ_i, t) = 0 otherwise
- A schedule is said to be Pfair if and only if
  lag(τ_i, t) = workload(τ_i, t) – $\Sigma_{k \leq t}$ sched(τ_i, k)
  where $\forall i$, $\forall t$: $-1 \leq$ lag(τ_i, t) ≤ 1
- A Pfair scheduling is feasible on m processors as long as $U \leq m$ (full utilization !)

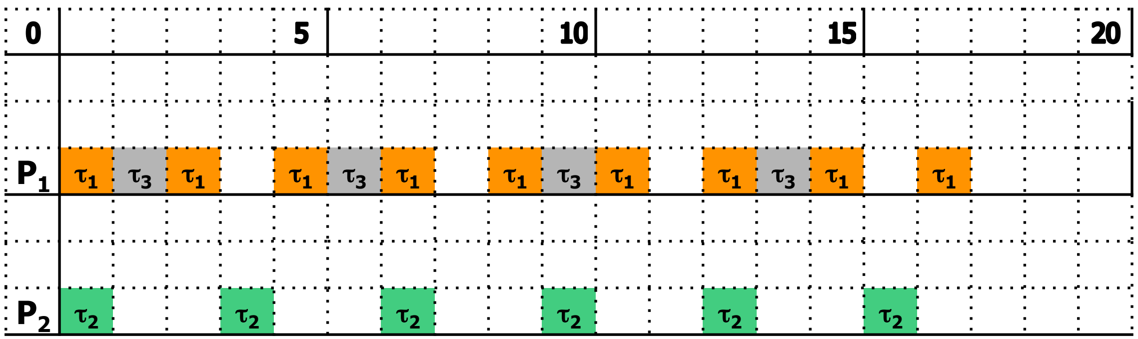
Global Scheduling
Pfair Algorithms : Implementation

- Split each task i into C_i subtasks (1 time unit)
- Assign a pseudo deadline $d(\tau_i, j)$ and a pseudo release $r(\tau_i, j)$ to subtask j in $[1..C_i]$:
  - $d(\tau_i, j) = \lceil j * Ti / Ci \rceil$
  - $r(\tau_i, j) = \lfloor (j - 1) * Ti / Ci \rfloor$
- Schedule subtask j according to $d(\tau_i, j)$ (EDF)
- Improve Pfair with non-arbitrary tie breaks to reduce context switches and migrations in case of identical pseudo-deadlines

Global Scheduling
Pfair Algorithms : Example

- $r(\tau_i, j) = \lfloor (j-1) * T_i/C_i \rfloor$ and $d(\tau_i, j) = \lceil j * T_i/C_i \rceil$
- $r(\tau_1,1) = 0$; $d(\tau_1,1) = 2$; $U_1=1/2$
- $r(\tau_2,1) = 0$; $d(\tau_2,1) = 3$; $U_2=1/3$
- $r(\tau_3,1) = 0$; $d(\tau_3,1) = 5$; $U_3=2/9$
- $r(\tau_3,2) = 4$; $d(\tau_3,2) = 9$; $U_3=2/9$

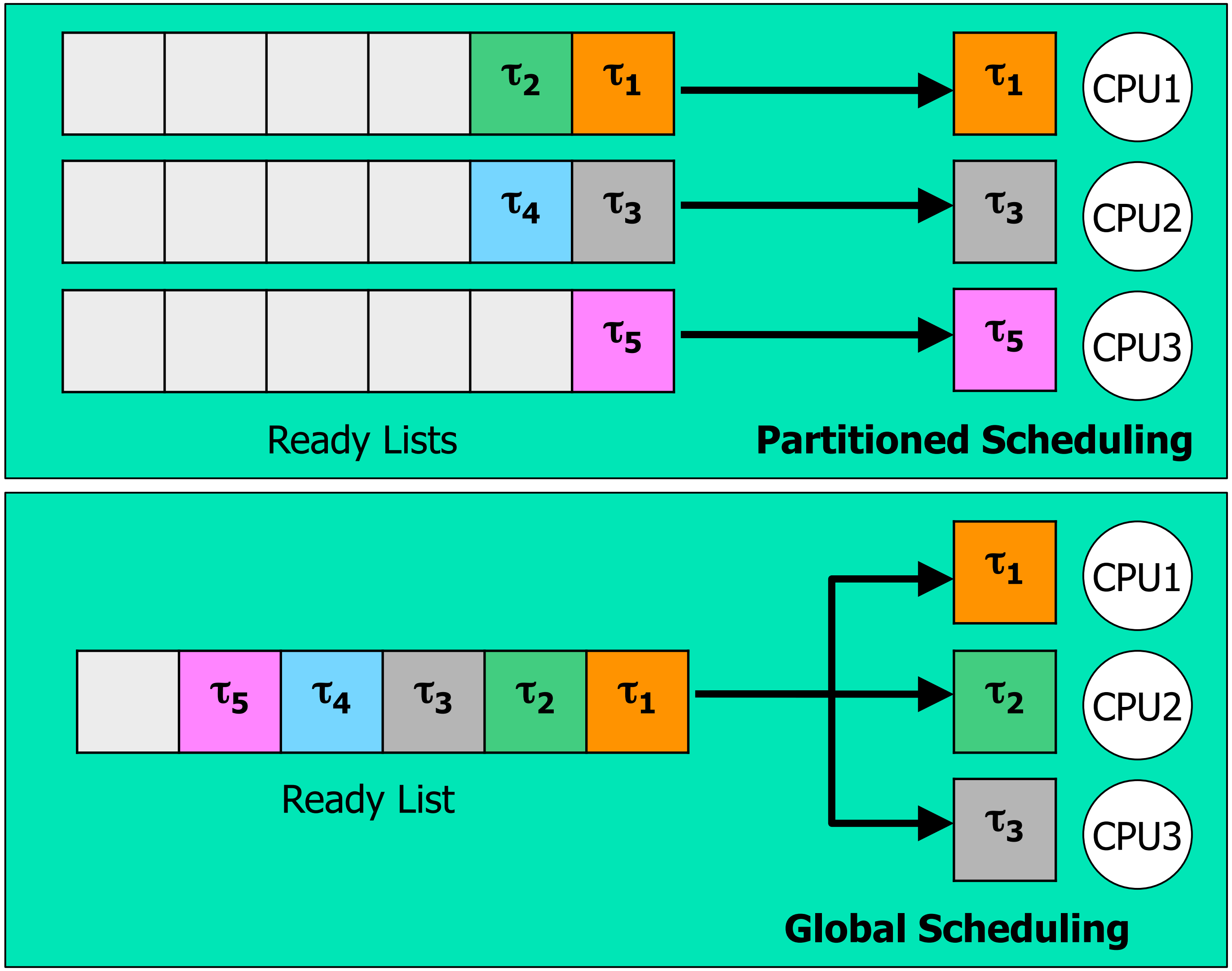
	C	T
τ_1	1	2
τ_2	1	3
τ_3	2	9



Global Scheduling Conclusions

- Global multi-processor scheduling has different properties compared to mono-processor scheduling (optimality, critical instant, feasibility interval, anomalies, ...).
- Additional parameters : migration, task / processor assignment, ...
- We limited architecture to identical processors, without shared resources
- We have limited task model to a simplified task one
- We have not discussed dependencies between tasks (shared resources, precedence constraints), nor communications.

Scheduling Approaches
Hybrid Scheduling



Hybrid Scheduling Principles

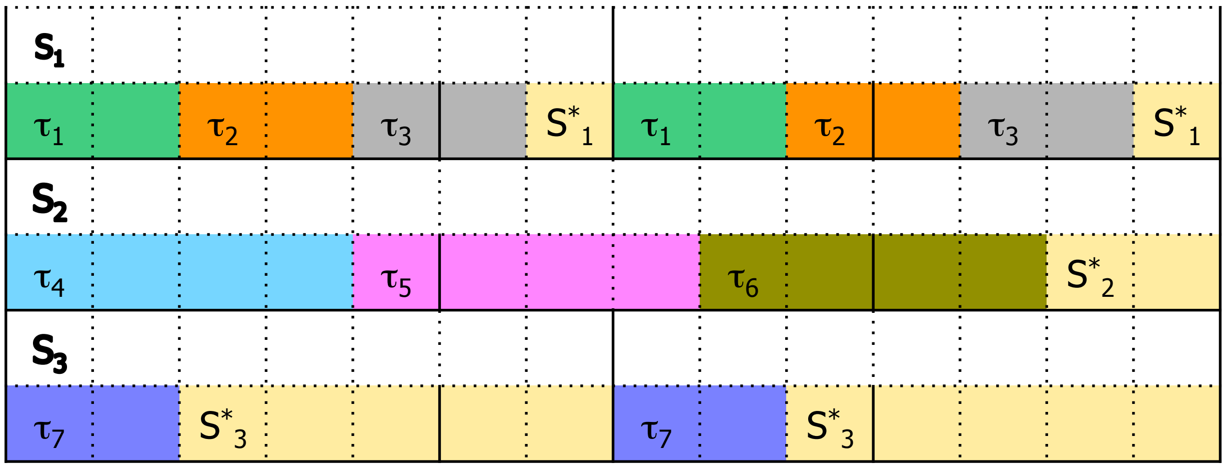
- A mixed solution between partitioned (offline) and global scheduling (online)
- Example: RUN (Reduction to Uniprocessor)
  - Optimal, less preemptions compared to PFair
  - Offline: build a reduction tree (PACK & DUAL steps)
    1. PACK tasks on a min nbr of virtual processors/servers
    2. Stop when schedule on a single processor/server
    3. Define idle time of processors/servers as DUAL idle tasks
    4. Loop to step 1
  - Online: schedule reduction tree (schedule tasks / processors in a virtual processor using EDF)

RUN
Offline : PACK + DUAL (first layer)

- Pack tasks on a minimum number of virtual processors (servers) S_1 to S_3. Use First-Fit.
- So, we cannot merge 2 virtual processors (VP)
- 3 idle time intervals : S^*_1 to S^*_3

U=2

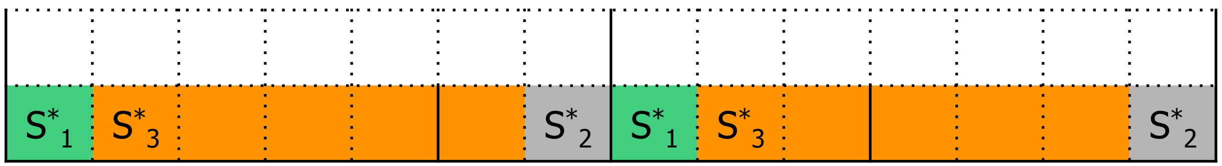
	C	T
τ_1	2	7
τ_2	2	7
τ_3	2	7
τ_4	4	14
τ_5	4	14
τ_6	4	14
τ_7	2	7



RUN
Offline : DUAL + PACK (second layer)

- Define S^*_1 to S^*_3 as (dual) tasks
- They model the idle time left on VPs
- Pack and schedule S^*_1 to S^*_3 on 1 VP
  - This new VP schedules « idle tasks » : we free a processor as the idle time is packed on 1 processor
- While #processors > 1, loop DUAL+PACK steps

	C	T
S^*_1	1	7
S^*_2	2	14
S^*_3	5	7

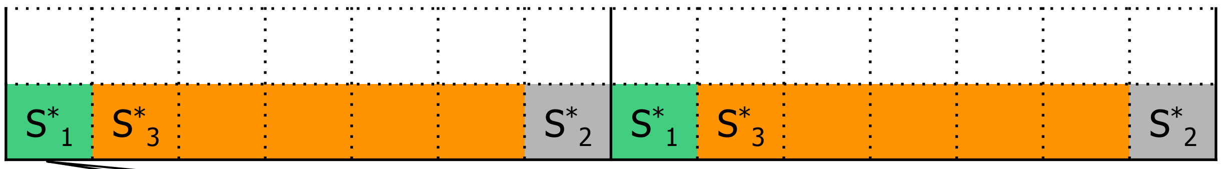


RUN
Online: Schedule Reduction Tree

- We have a tree of servers (or a hierarchy of servers) that schedules tasks and servers
  - We start scheduling the root server of the tree
  - When we schedule a dual server, we do not schedule its tasks or servers. We schedule the remaining tasks or servers applying EDF.
  - When we schedule a primary server, we do schedule its tasks or servers applying EDF
- In the example, we start executing S^*_1. Thus, we do not execute S_1 but S_2 ou S_3. Applying EDF, S_2 will execute τ_1 and S_3 will execute τ_1

RUN
Online: Scheduling previous example

	C	T
S^*_1	1	7
S^*_2	2	14
S^*_3	5	7

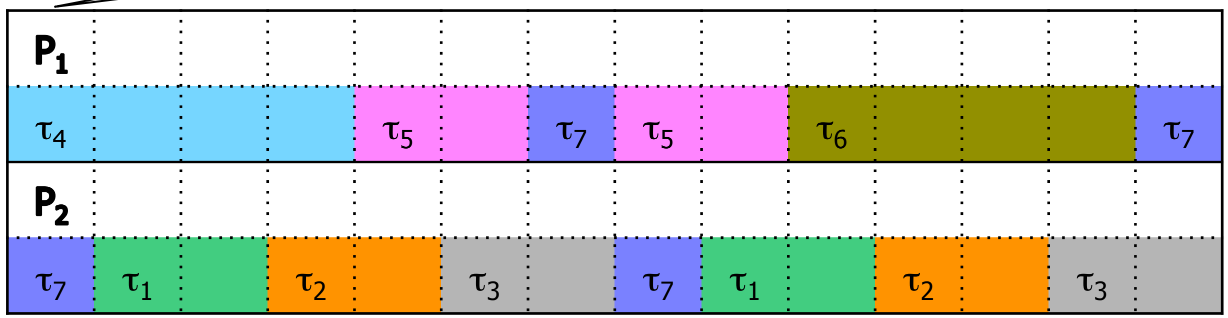


Schedule S^*_1 => schedule all but S1,
=> schedule S_2 or S_3

Schedule T_4, T_5 or T_6 on P_1 and T_7 on P_2 both using EDF

U=2

	C	T
τ_1	2	7
τ_2	2	7
τ_3	2	7
τ_4	4	14
τ_5	4	14
τ_6	4	14
τ_7	2	7



Real-Time Scheduling for Distributed Systems

- Tasks exchange messages

1. Tasks are dependant and assigned to procs
   - The task input is the output of its predecessors
2. Model and schedule messages as tasks

Non-preemptive task	**Message**
(Mono) Processor	Communication medium
Capacity / Budget	Communication delay (buffer, access, propagation)

3. Schedule messages on bus or network
   - Use non-preemptive tasks scheduling

Distributed Real-Time Scheduling
Step 1: Dependant Tasks on Mono-Processors

- Dependant tasks on a mono-processor
  - Modify task parameters to have independant tasks
  - $A^*_i = \max(A_i, \max_{j \text{ in } pred(i)} A^*_j + C_j)$
  - $D^*_i = \min(D_i, \min_{j \text{ in } succ(i)} D^*_j - C_j)$
- For a static priority scheduling, give higher priority to predecessors than to task (DMS)
  - We can compute response time
- For a dynamic priority scheduling, use new deadlines (EDF)

Distributed Real-Time Scheduling

Step 2 : Dependant tasks on Distributed Systems

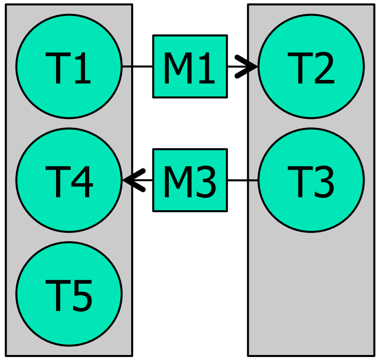
- Holistic Method
  - Compute response time with jitter …
  - defined as the max response time of predecessors
- Iterative method (as for mono-processors)
- For task with an fixed priority scheduling
  - $R_i^{n+1} = J_i + C_i + \Sigma_{k \text{ in hp on proc (i)}} C_k * \lceil (J_k + R_i^n) / T_k \rceil$
- For message
  - $R_i = J_i + M_i$

Distributed Real-Time Scheduling

Step 3 : Message Scheduling on (CAN) Bus

- Messages modeled as non-preemptive tasks
- Compute response time for static priority scheduling of non-preemptive tasks
- $R^i_{n+1} = J_i + C_i + \Sigma_{k \text{ in } hp(i)} C_k * \lceil (J_k + R^i_n) / T_k \rceil + \max_{l \text{ in } lp(i)} (C_l)$
  - The last term represents the blocking time induced by a lower priority non-preemptive task

Distributed Real-Time Systems



	T	C	Pri
T1	100	4	HI
T2	100	3	ME
T3	60	2	HI
T4	60	5	ME
T5	90	3	LO
M1	100	6	LO
M3	60	1	HI

Step 1 : T_1 (resp T_3) has higher priority than successor T_2 (resp T_4)
Priorities are computed with DMS and D_1 (resp D_3) < D_2 (resp D_4)

Step 2 : $R_i^{n+1} = J_i + C_i + \Sigma_{k \text{ in pred}(i)} C_k * \lceil (J_k + R_i^n) / T_k \rceil$

	M1	M3	T1	T2	T3	T4	T5
J	0	0	0	0	0	0	0
R	6	1	4	5	2	9	12

	M1	M3	T1	T2	T3	T4	T5
J	**4**	**2**	0	**6**	0	**1**	0
R	**10**	**3**	4	**11**	2	**10**	12

	M1	M3	T1	T2	T3	T4	T5
J	4	2	0	**10**	0	**3**	0
R	10	3	4	**15**	2	**12**	12

Step 3 : M_1 and M_3 are schedulable on network (trivial)

Conclusions

- Less mono-processors, more multi-processors or heterogeneous systems on the market
- Very active research domain to design new scheduling approaches
- Less predictive processors on the market ; approximate WCET due to many interferences
- Define modes and change mode when overloaded
- The low criticality mode includes all the tasks
- The high criticality one only high criticality tasks
- Active research domain : mixed criticality Systems