

SPRINT: Extending RUN to Schedule Sporadic Tasks

Andrea Baldovin, Geoffrey Nelissen,
Tullio Vardanega, Eduardo Tovar

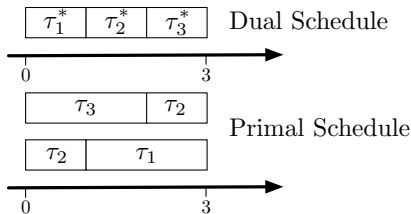
22nd International Conference on Real-Time Networks and Systems
Versailles, October 10th, 2014



Outline

- 1 Recalling RUN
- 2 Introducing SPRINT
 - The intuition behind
 - Budget computation
 - Server priorities
 - Example
- 3 Simulation results
- 4 Future work

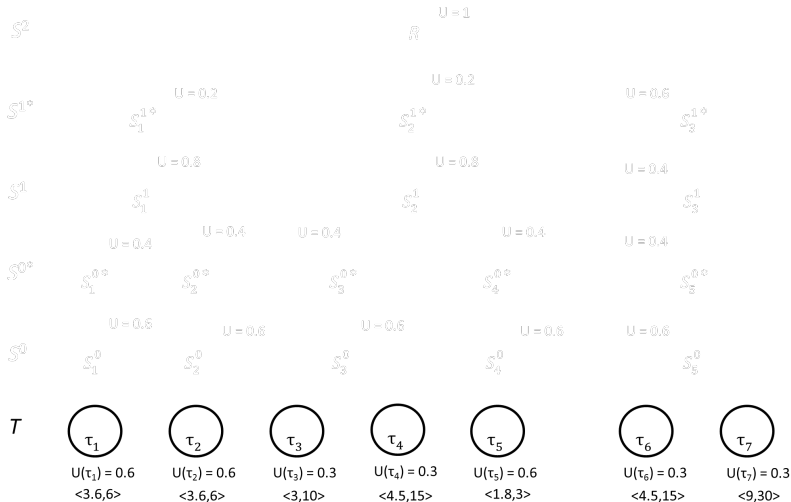
- Optimal, semi-partitioned scheduling algorithm for multicore
- Consists of 2 phases
 - Offline
 - * Builds a reduction tree by dual scheduling and bin-packing



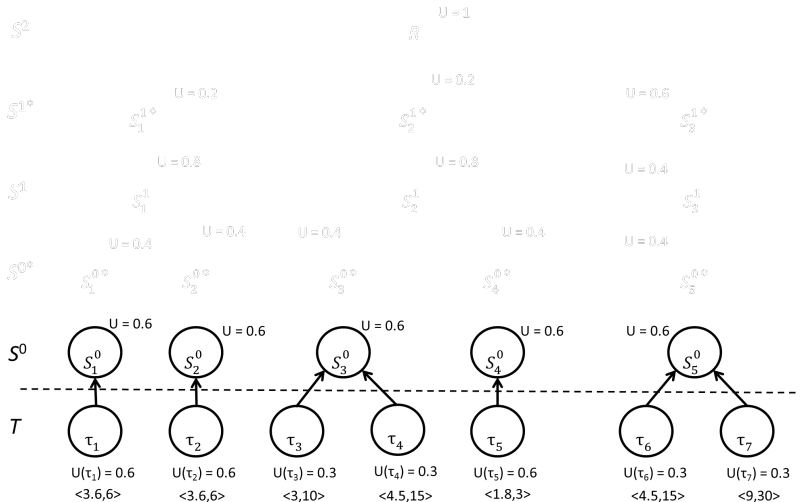
$$U(\mathcal{T}^*) = n - U(\mathcal{T})$$

- Online
 - * Solves each single-core scheduling problem with an optimal algorithm (EDF)
- Scheduling m tasks on n processors
 - k scheduling problems on 1 processor

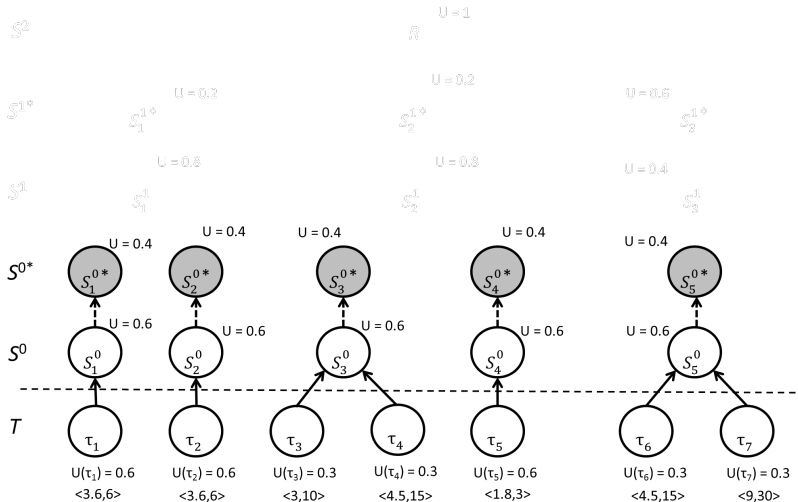
RUN - Example (Offline phase)



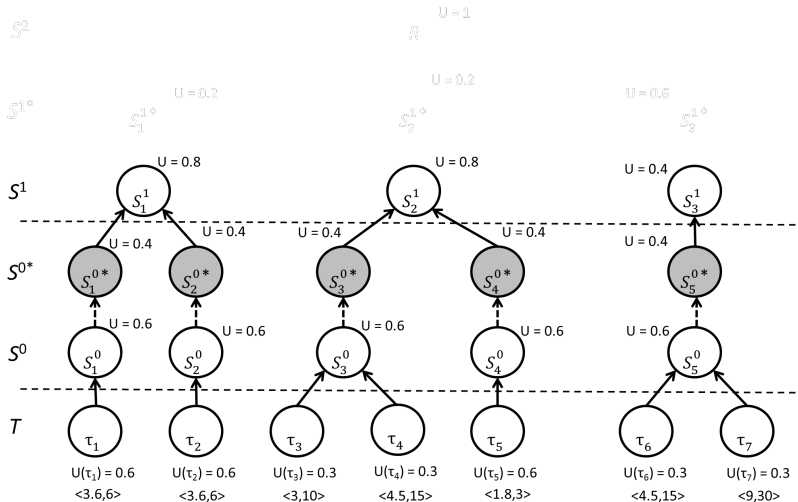
RUN - Example (Offline phase)



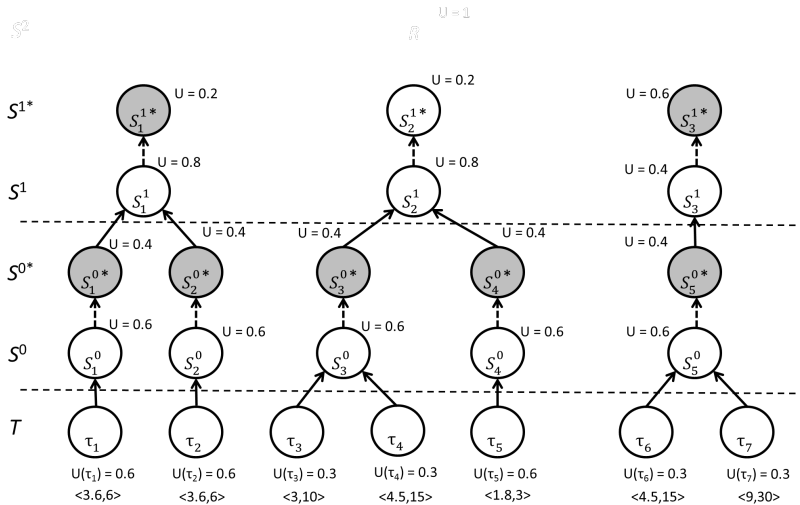
RUN - Example (Offline phase)



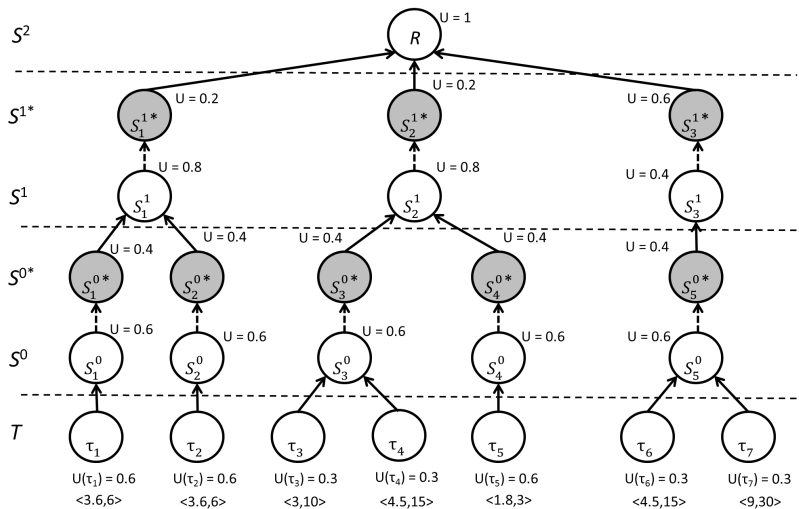
RUN - Example (Offline phase)



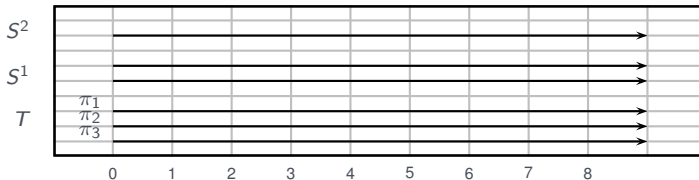
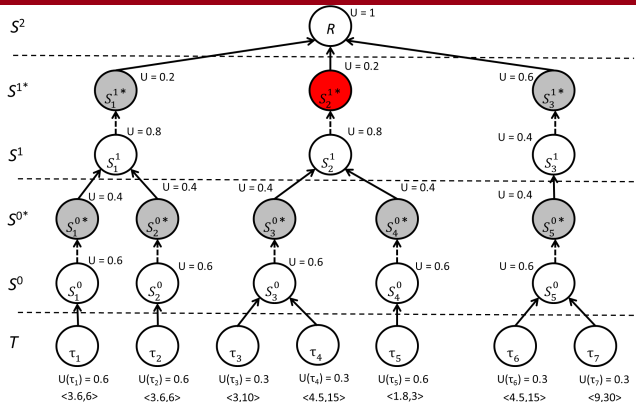
RUN - Example (Offline phase)



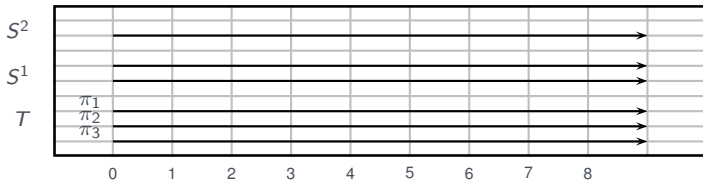
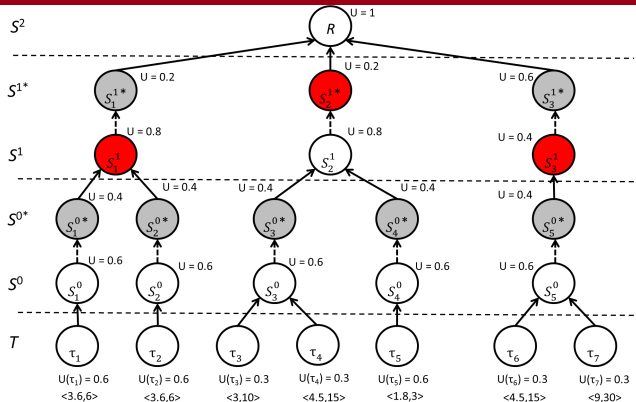
RUN - Example (Offline phase)



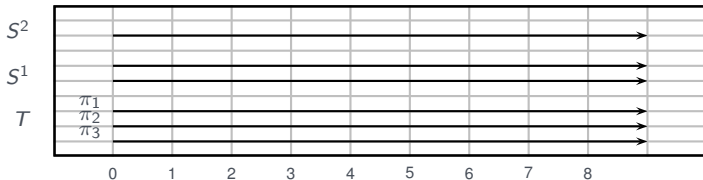
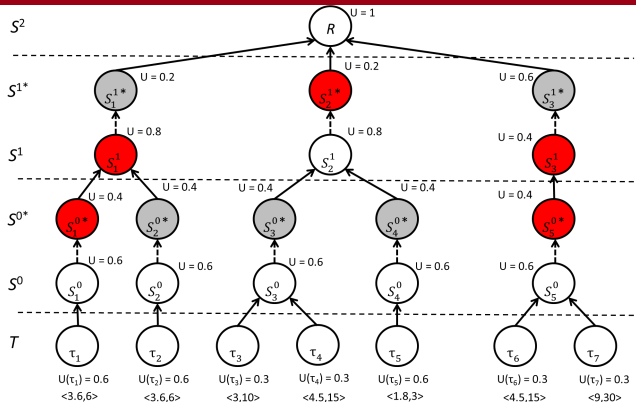
RUN - Example (Online phase)



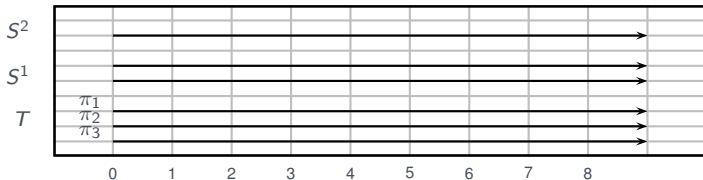
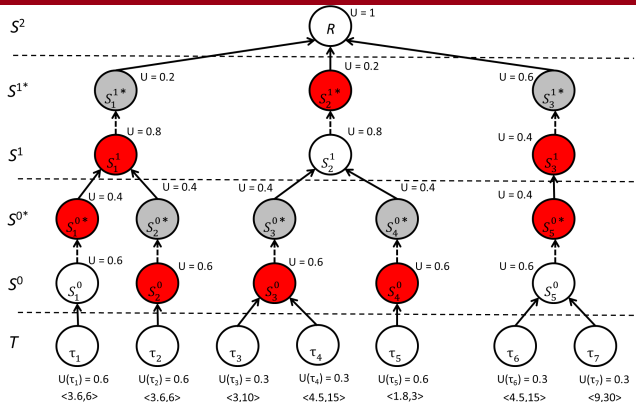
RUN - Example (Online phase)



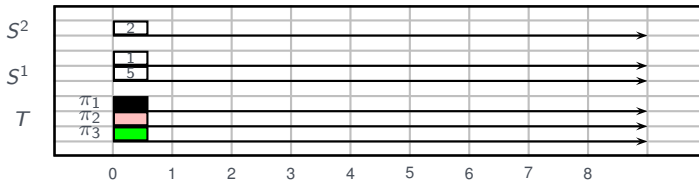
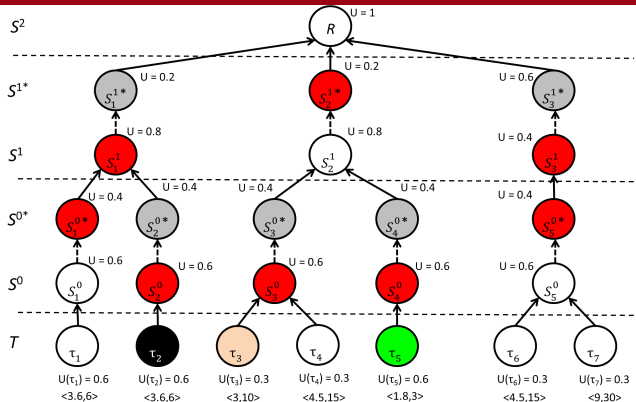
RUN - Example (Online phase)



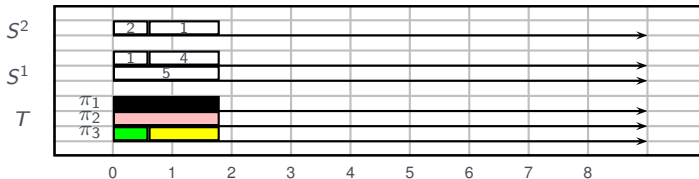
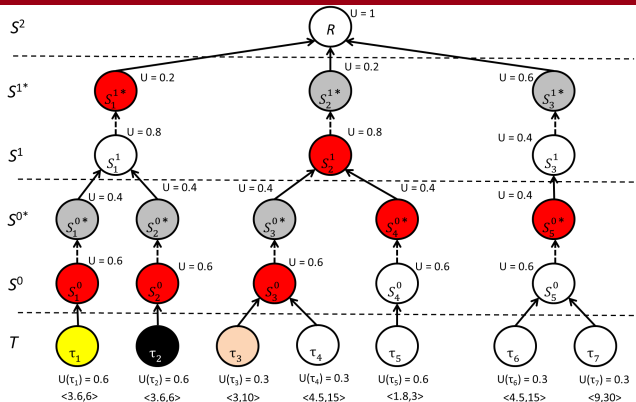
RUN - Example (Online phase)



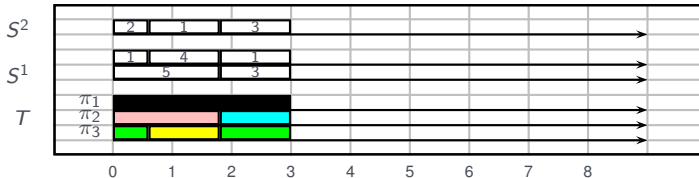
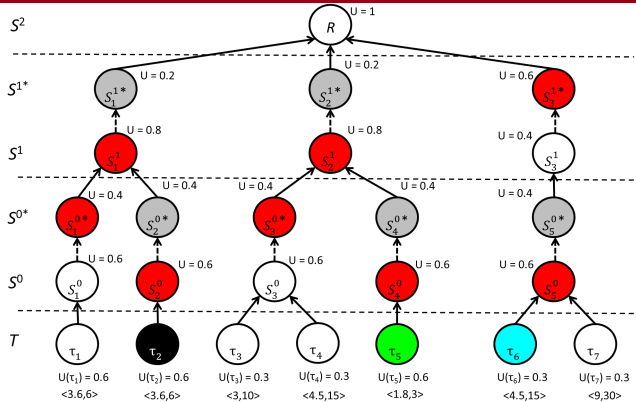
RUN - Example (Online phase)



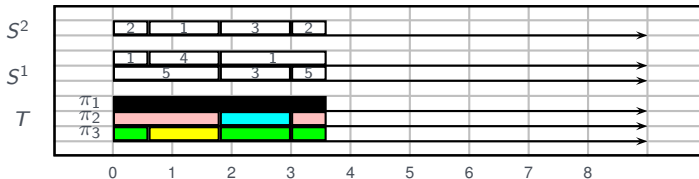
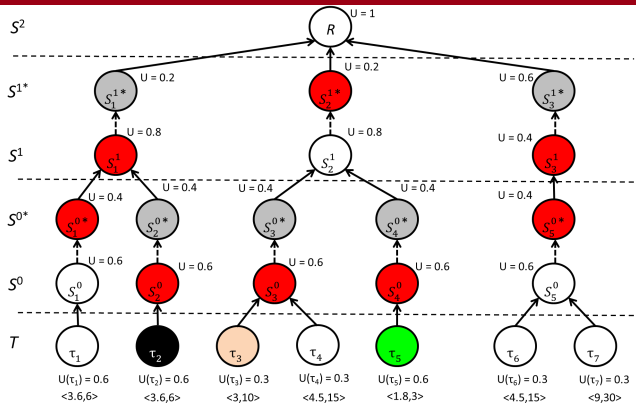
RUN - Example (Online phase)



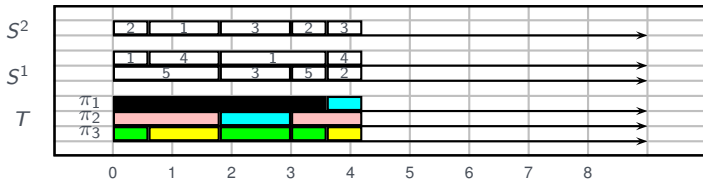
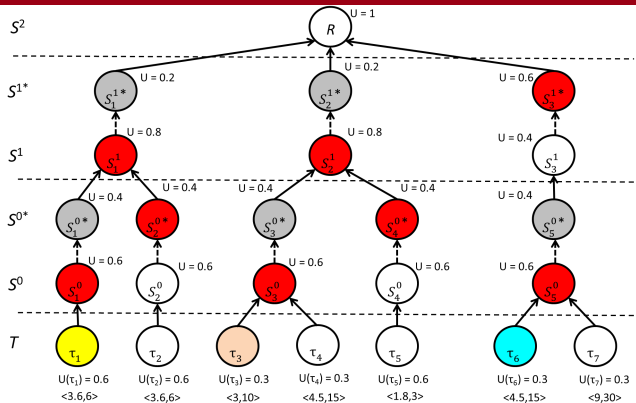
RUN - Example (Online phase)



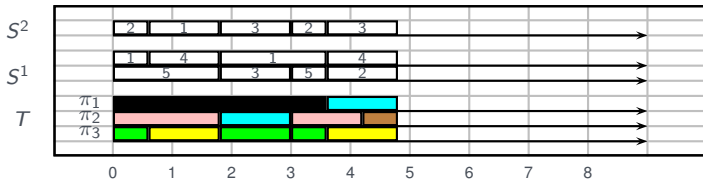
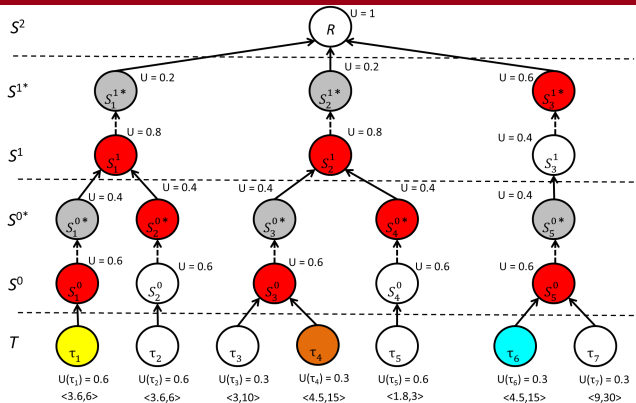
RUN - Example (Online phase)



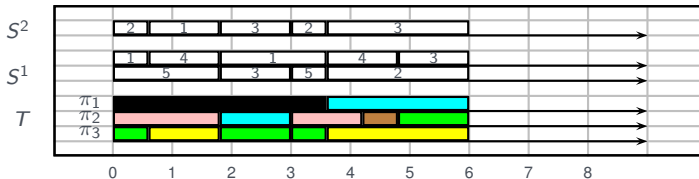
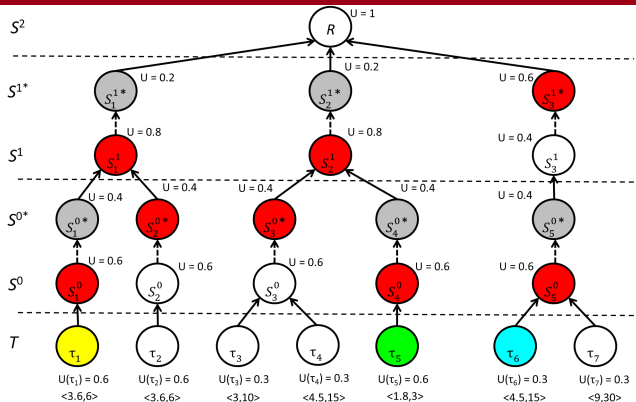
RUN - Example (Online phase)



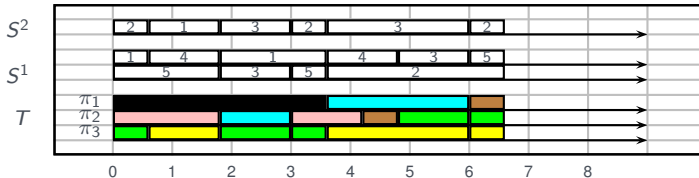
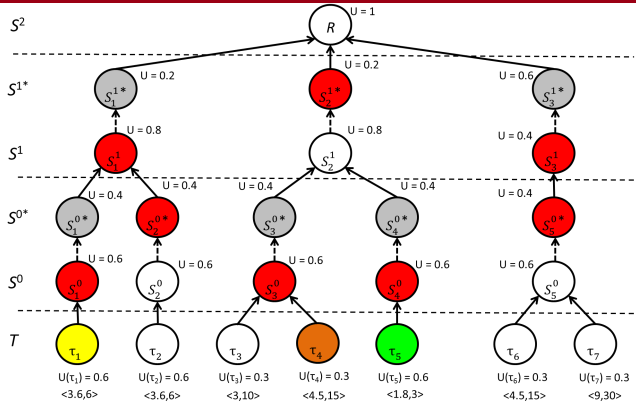
RUN - Example (Online phase)



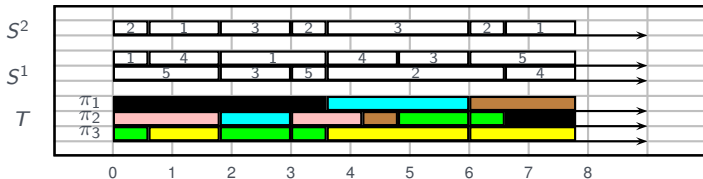
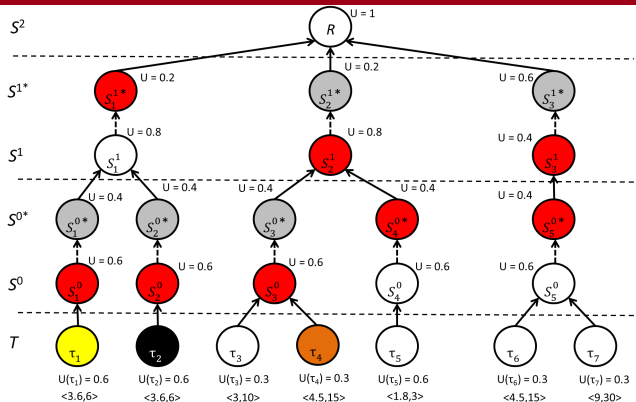
RUN - Example (Online phase)



RUN - Example (Online phase)



RUN - Example (Online phase)



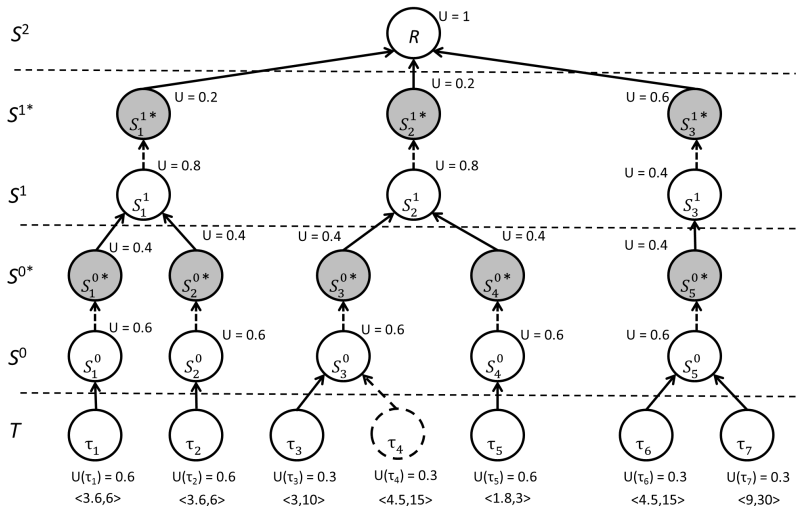
SPRINT - Intuition

■ RUN only fits for periodic task sets

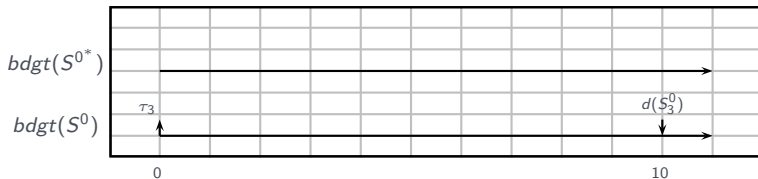
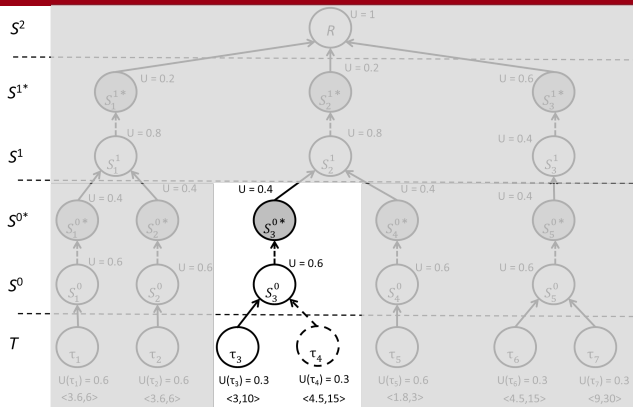
- Job release events at level 0 only occur at multiples of tasks' activations
- However, artificial workload may be present in the system to reach full utilization assumed by the packing process
- Let's take advantage of this time for trying to accommodate sporadic activations instead
- In this scenario, job releases may occur at any point in time
- Recompute server execution budgets accordingly to account for future sporadic releases

SPRINT - Reduction at level 0

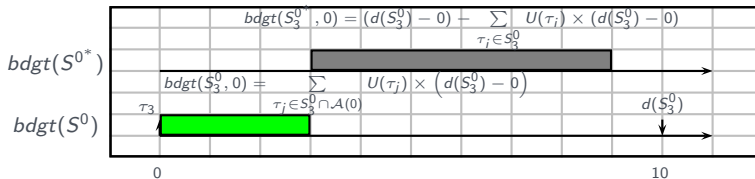
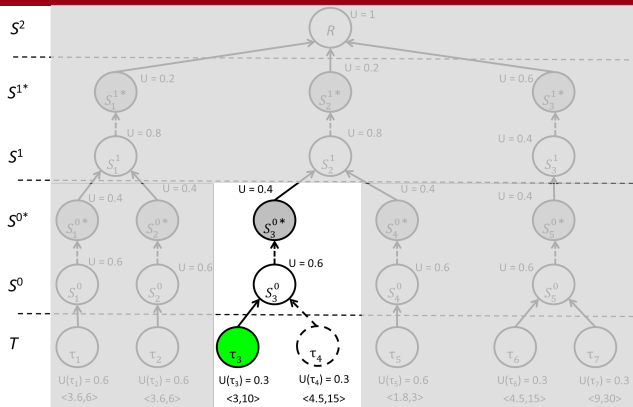
- At level 0, adjusting budgets of servers is all we need



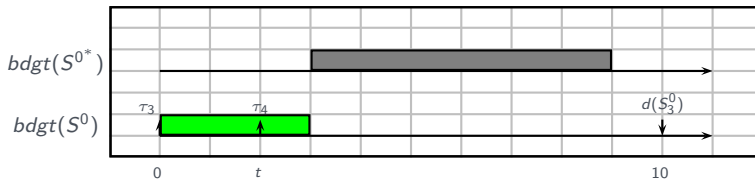
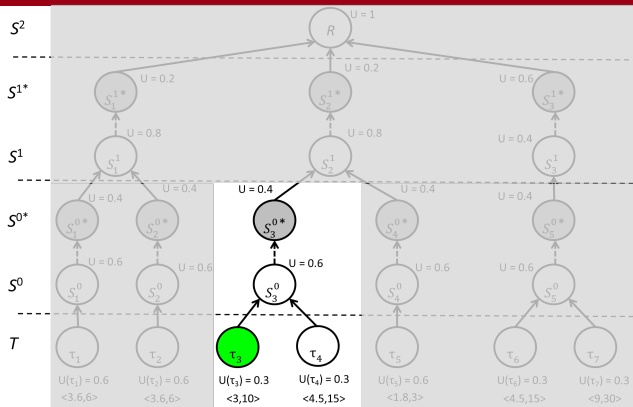
SPRINT - Reduction at level 0



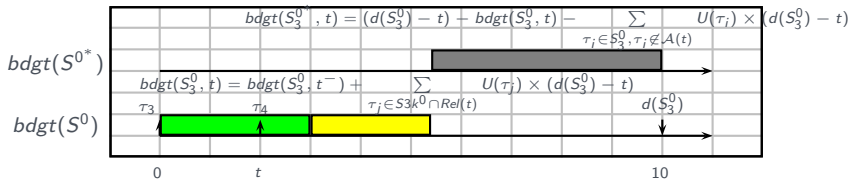
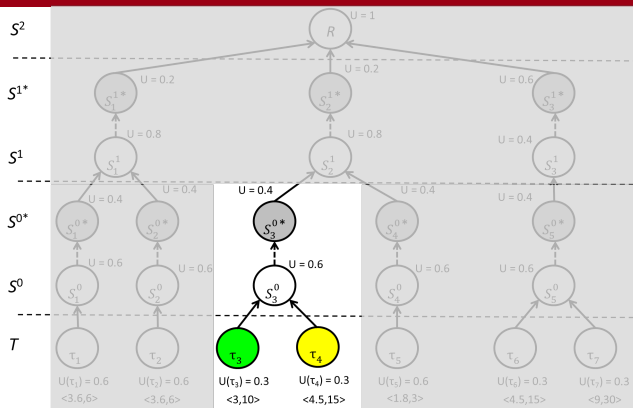
SPRINT - Reduction at level 0



SPRINT - Reduction at level 0

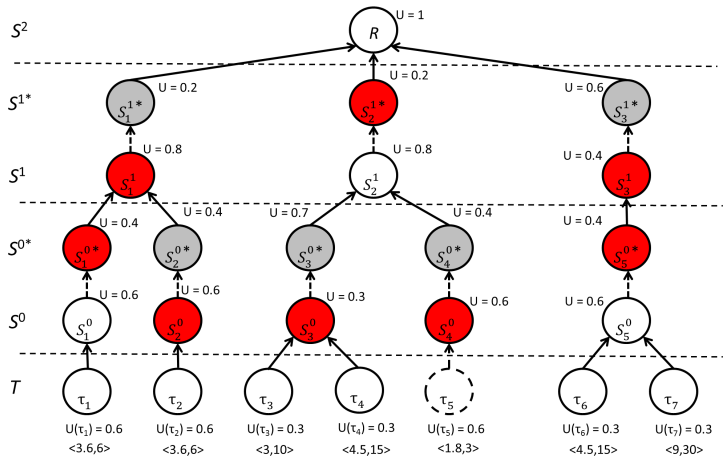


SPRINT - Reduction at level 0



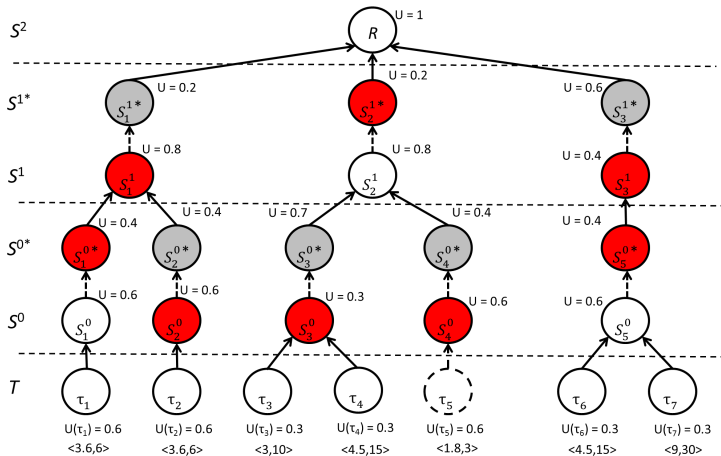
Server priorities

- But that's not enough when the full tree is considered



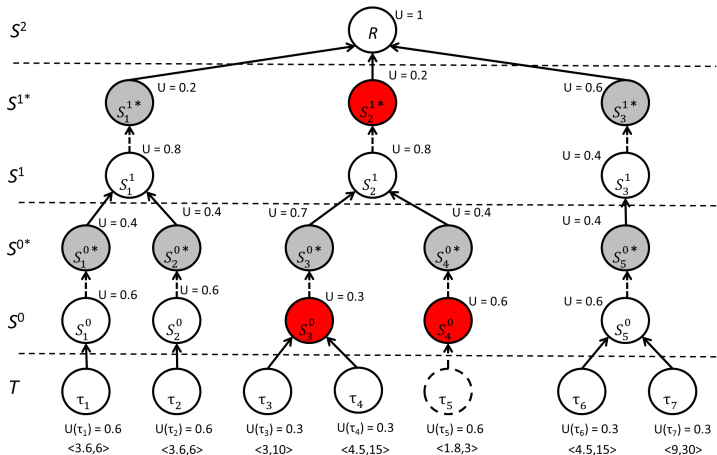
Server priorities

- We need a mechanism to select the “right” branch of the tree



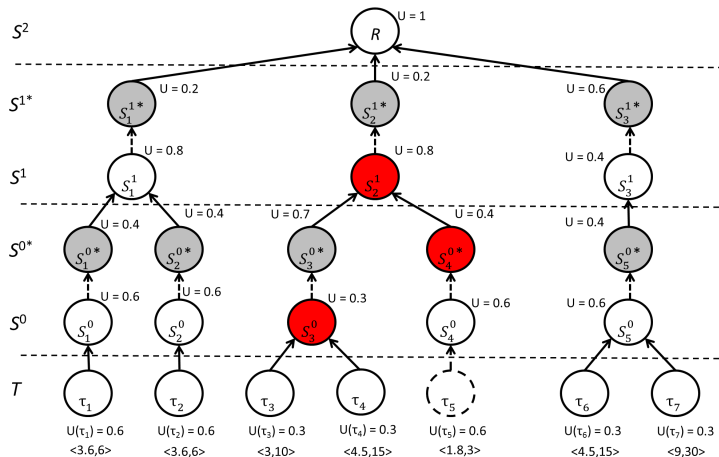
Server priorities

- Observation 1: S_i^{1*} executes \iff all $S_k^0 \in S_i^1$ execute



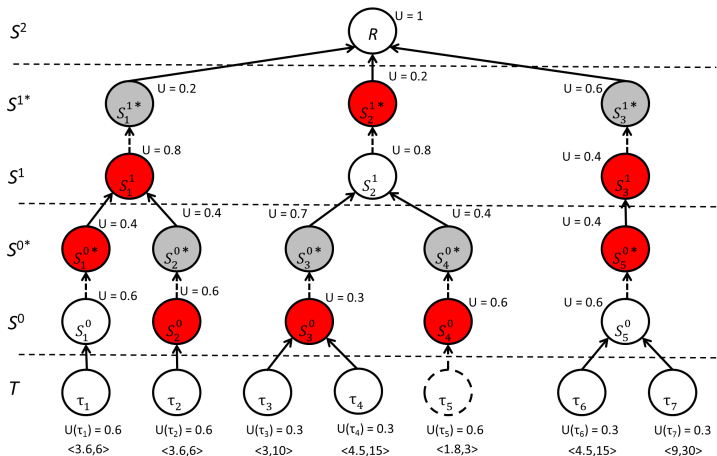
Server priorities

- Observation 2: S_i^{1*} does not execute \iff all $S_k^0 \in S_i^1$ but one execute

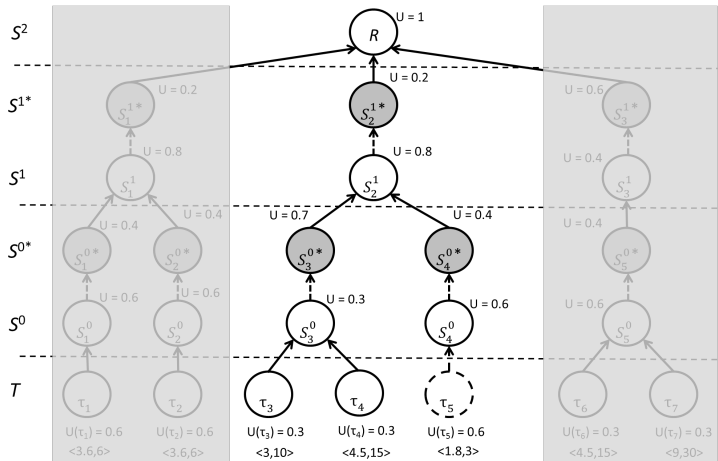


Server priorities

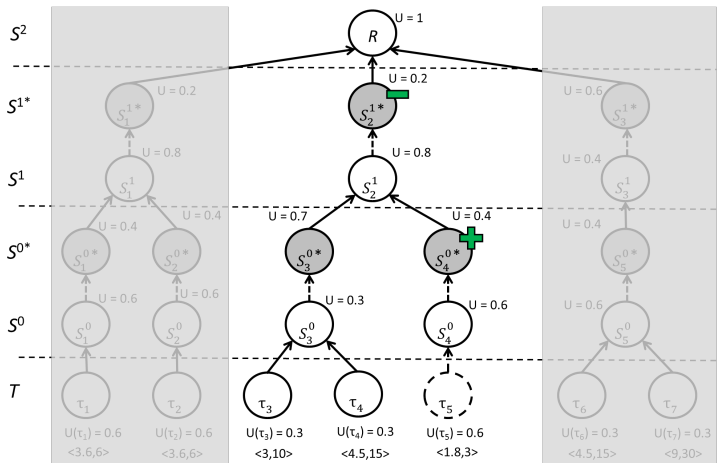
- Inference: Therefore, we don't need to execute S_i^{1*} if some sporadic task is packed below it



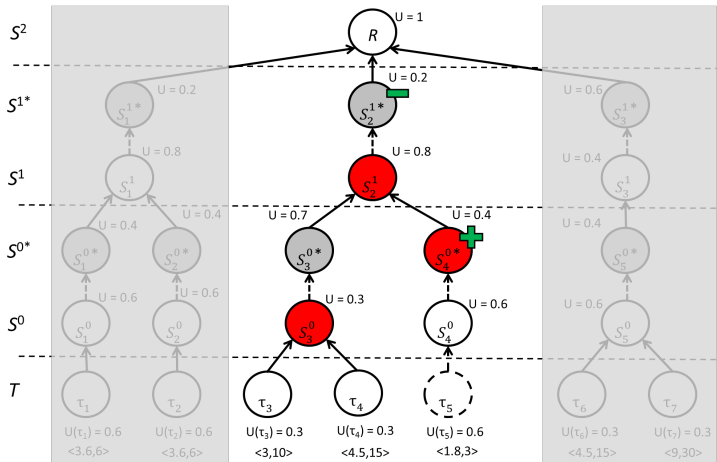
Server priorities



Server priorities



Server priorities

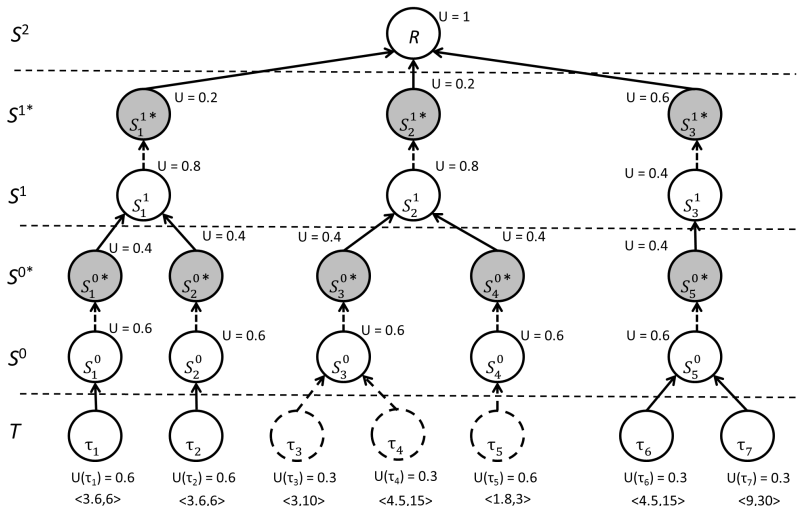


SPRINT - Reduction at level 1

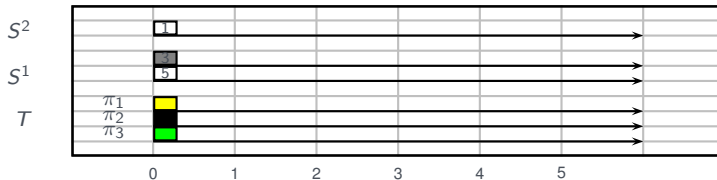
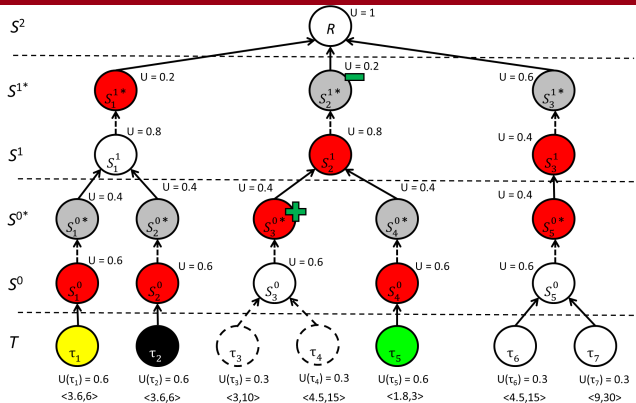
- So, how do we compute the budget of servers at level 1?
 - Requires reasoning on the dual problem
 - Target 1: Providing enough execution time for job completion
 - Target 1: Ensuring that sporadic releases do not interfere with nominal execution
 - Still, it's a matter of computing the demand from lower-level server
 - Now taking into account possible sporadic activations in the future
- Our solution is tailored to level 1, but does not scale up
 - That's where optimality is lost
- Mathematical formulation in the paper

SPRINT - Example

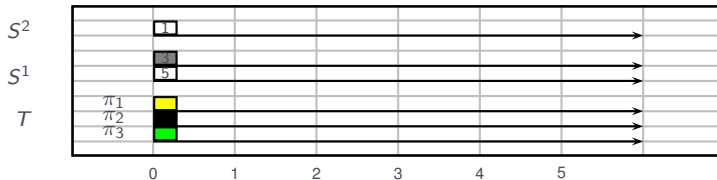
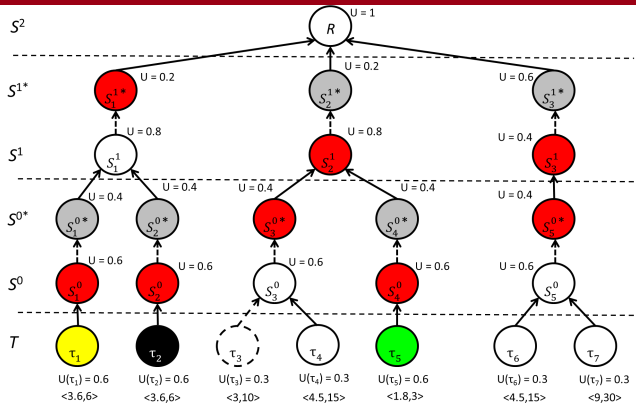
- τ_3 releases jobs at $\{3, \dots\}$, τ_4 at $\{0.3, \dots\}$, τ_5 at $\{0, 3.5, \dots\}$



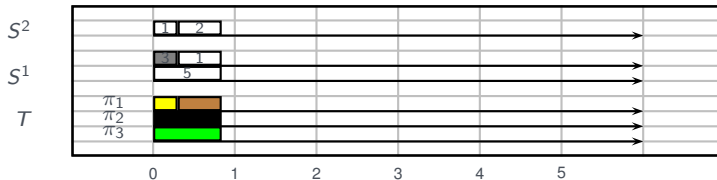
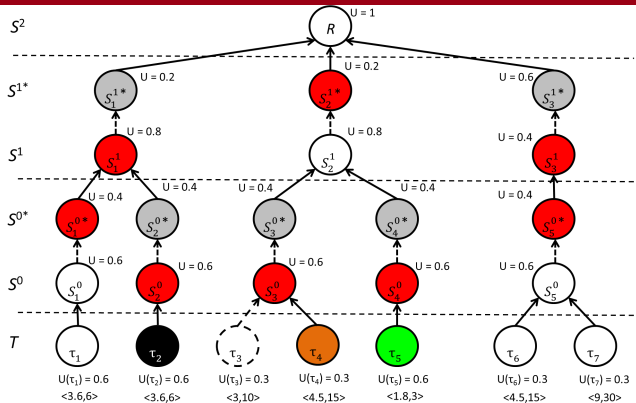
SPRINT - Example



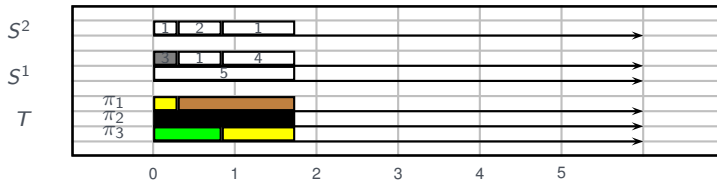
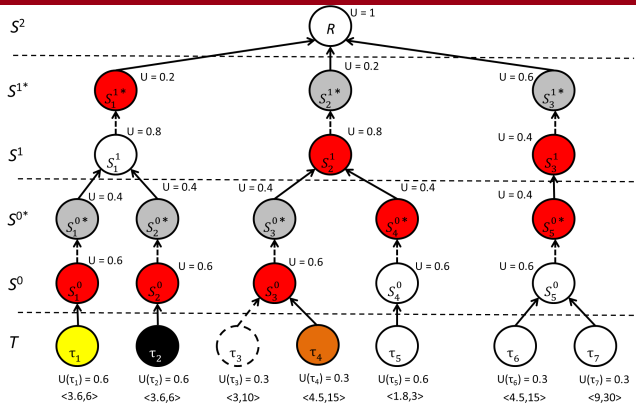
SPRINT - Example



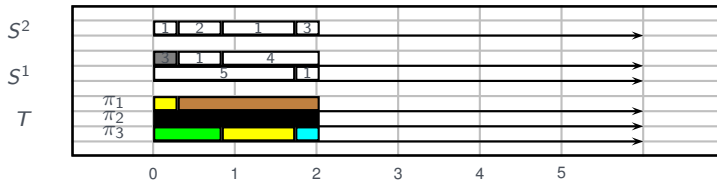
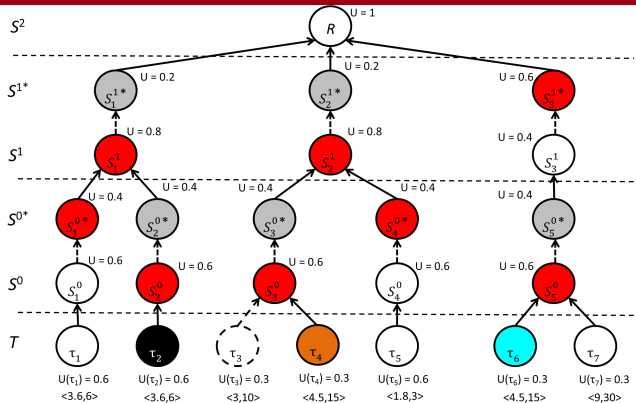
SPRINT - Example



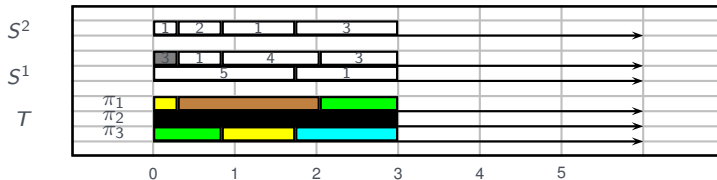
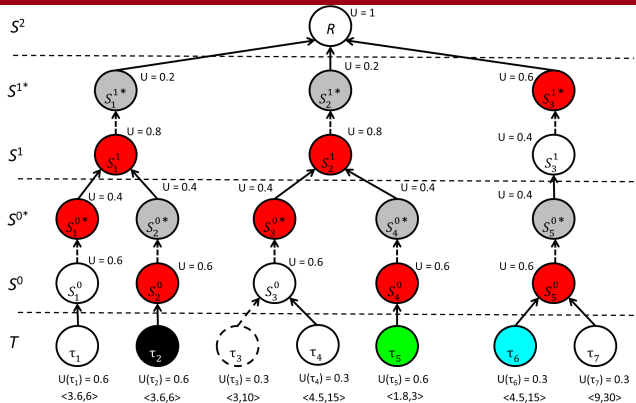
SPRINT - Example



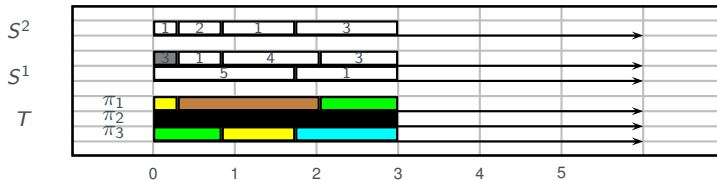
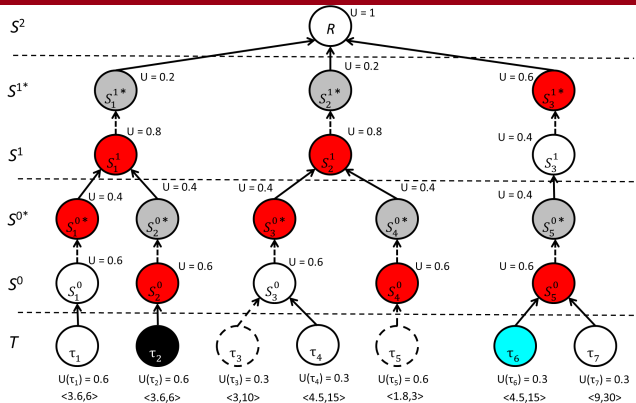
SPRINT - Example



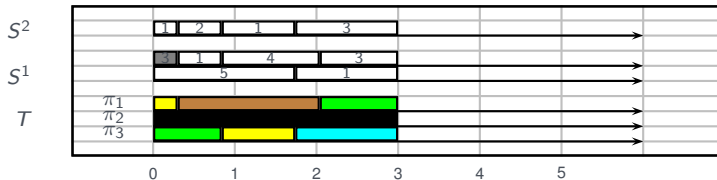
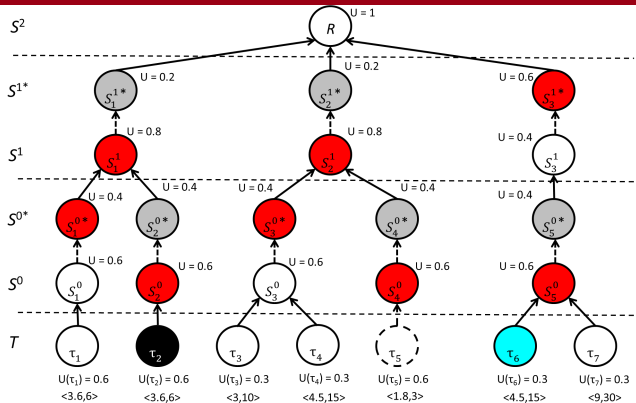
SPRINT - Example



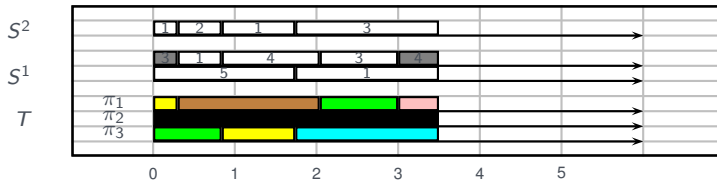
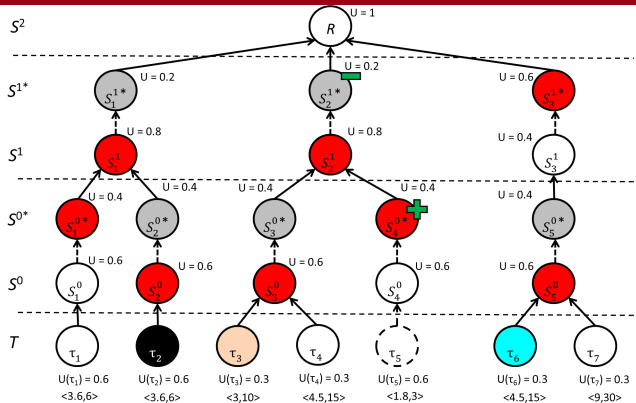
SPRINT - Example



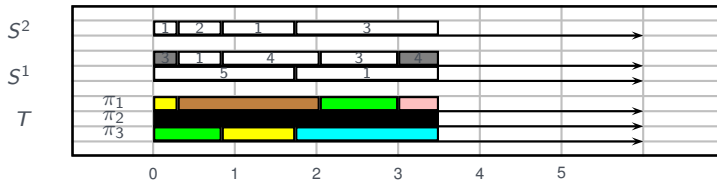
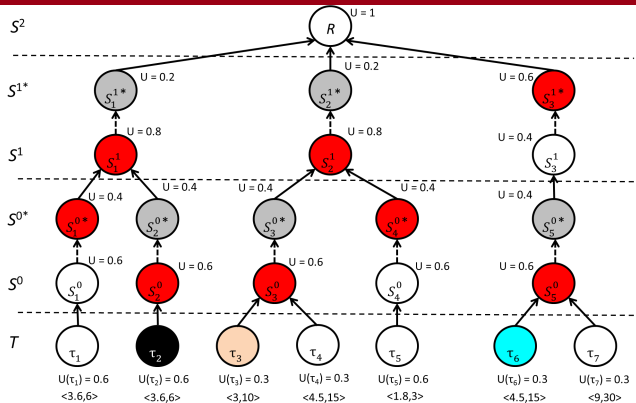
SPRINT - Example



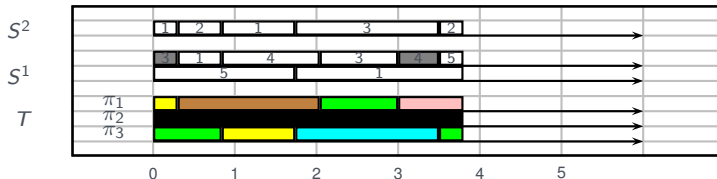
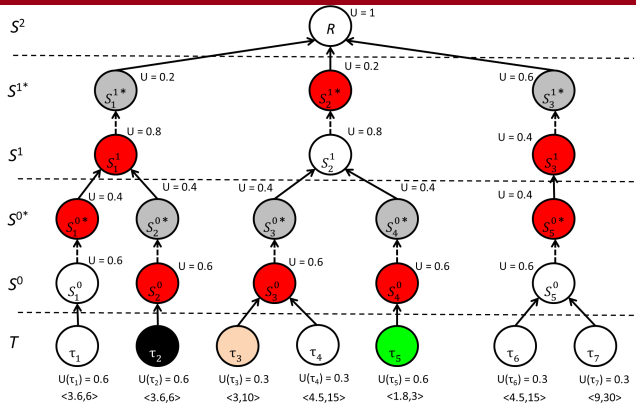
SPRINT - Example



SPRINT - Example

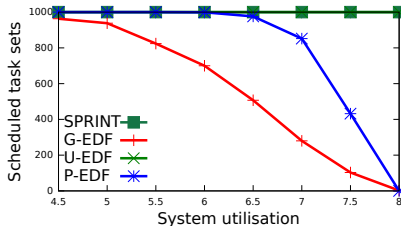
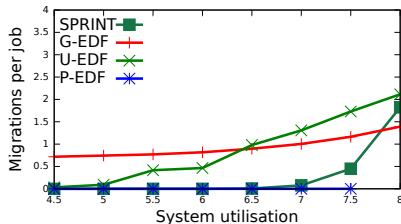
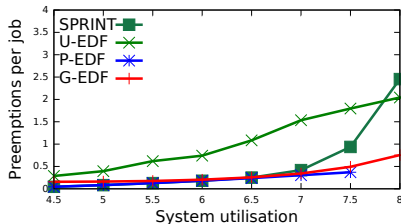


SPRINT - Example



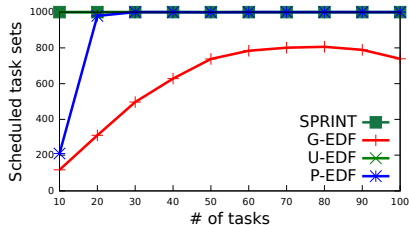
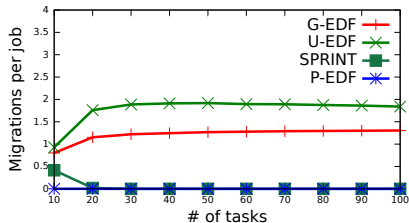
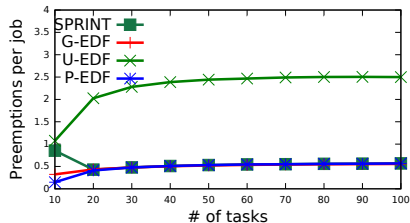
SPRINT - Simulation results /1

■ Increasing system utilisation



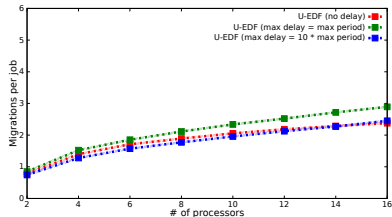
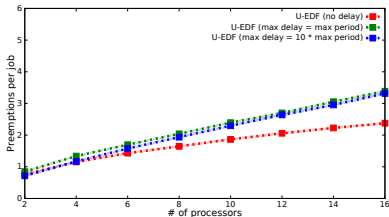
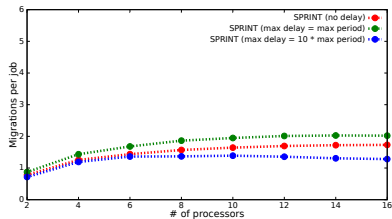
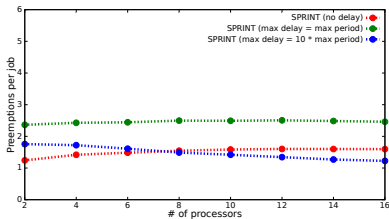
SPRINT - Simulation results /2

■ Increasing number of tasks

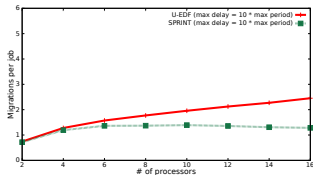
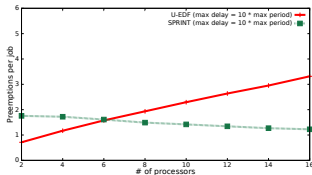
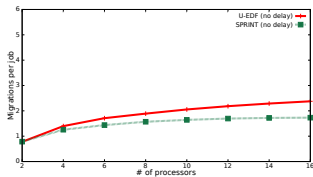
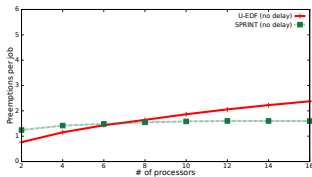
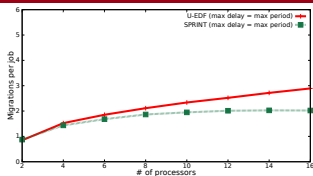
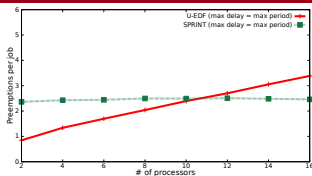


SPRINT - Simulation results /3

■ Increasing number of processors



SPRINT - Simulation results /3



Future work

- Optimality needs further investigation
 - Reasoning on level 1 is not easy, finding a general rule even less
 - Although not optimal, SPRINT schedules the *vast majority* of task sets
- Comparison needed on a real implementation
- Comparison needed against QPS
 - RUN better than QPS according to QPS paper
 - Preliminary result on a real implementation: it may depend on the specific task set