Precedence-aware Automated Competitive Analysis of Real-time Scheduling

Andreas Pavlogiannis Nico Schaumberger Ulrich Schmid
Krishnendu Chatterjee

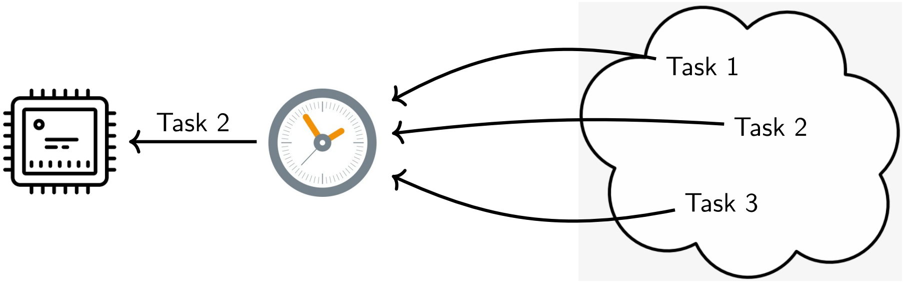
August 28, 2020

AARHUS UNIVERSITY

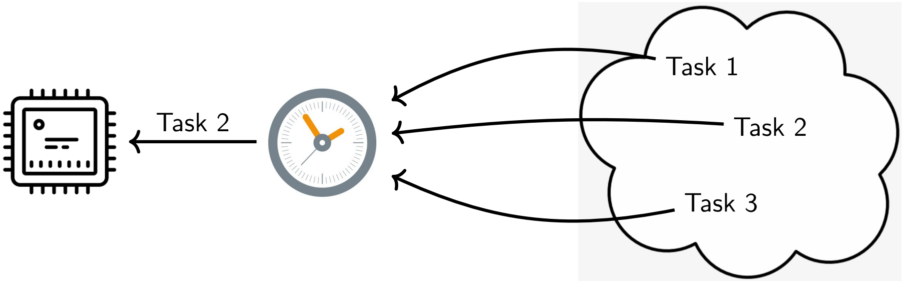
TU WIEN

IST AUSTRIA
Institute of Science and Technology

Real-time Scheduling

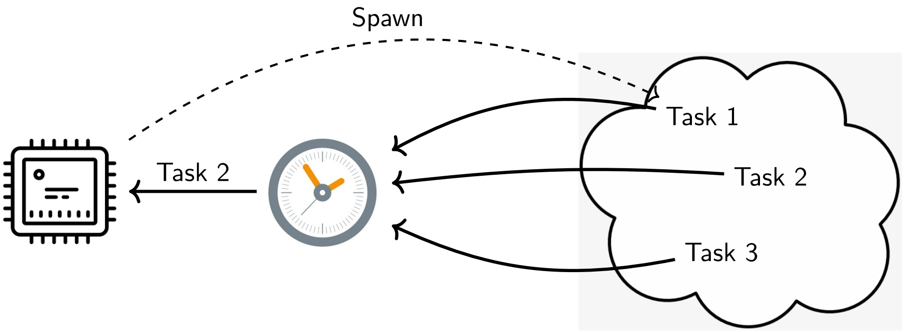


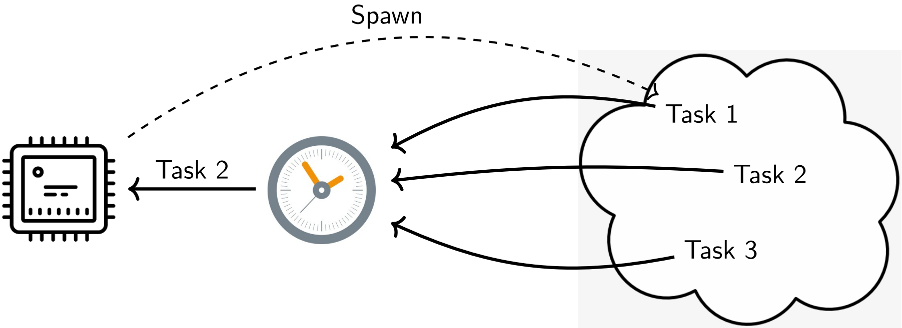
Real-time Scheduling



- How well is scheduler handling requests? $\rightarrow$ Competitive analysis
- Usually done manually, some recent efforts in automated techniques

Scheduling Dynamic Tasks





Challenge

How to we compute competitiveness automatically?

This paper

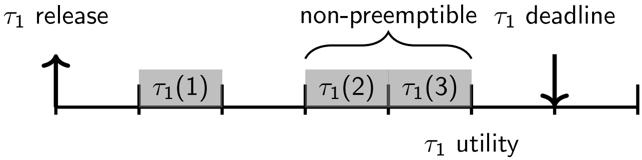
A framework for push-button computation of competitiveness in precedence-aware scheduling

Scheduling Model

- Discrete time
- Input: a collection of tasks $\mathsf{Ts} = \{\tau_1, \ldots, \tau_n\}$
- Each $\tau_i = (\mathsf{Et}_i, \mathsf{Dl}_i, \mathsf{Ut}_i, \mathsf{Np}_i)$
  - $\mathsf{Et}_i \in \mathbb{N}^+$ is the WCET
  - $\mathsf{Dl}_i \in \mathbb{N}$ is the relative deadline
  - $\mathsf{Ut}_i \in \mathbb{N}$ is the utility value
  - $\mathsf{Np}_i = \{[l_1, l_2], \ldots, [l_{2k-1}, l_{2k}]\}$ are no-preemption intervals

- In each round, various job instances of tasks are released
- A scheduler decides which released task takes the processor for a single time unit
- Preemption is allowed given the no-preemption interval of each task
- If a job is completed within its deadline, it contributes a utility to the system

Scheduling Model



Static vs Dynamic Tasks

Static

- Tasks are independent
- Every release of a task has the same characteristics (e.g., utility)

Dynamic

- Tasks are dependent
- E.g., completion of a task can cause the release of another

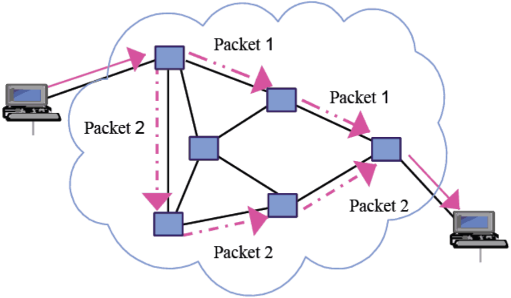
1. Pairing Precedences

- Precursor tasks $\tau_1, \ldots, \tau_k$
- Time window $[t_1, t_2]$
- Dependent task τ, modified τ'

Semantics

- If all precursor tasks $\tau_1, \ldots, \tau_k$ are completed now, if dependent task τ is released between t_1 and t_2 slots, it is modified to τ'

1. Pairing Precedences (Example)



Packet Switching

- A packet consists of a header fragment τ_h and a data fragment τ_d
- Serving the data contributes to utility iff the header has been completed
- Pairing precedence: completing τ_h modifies the utility of the next release of τ_d to non-zero

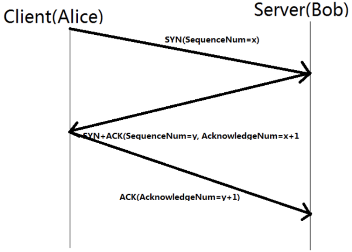
2. Follower Precedences

- Precursor tasks $\tau_1, \ldots, \tau_k$
- Time window $[t_1, t_2]$
- Dependent task τ

Semantics

- If all precursor tasks $\tau_1, \ldots, \tau_k$ are completed now, the dependent task τ must be released between t_1 and t_2 slots
- When τ is released, the precedence resets

2. Follower Precedences (Example)



Handshake Protocol

- Payload message τ_p and ack message τ_a
- Ack is sent only if the payload message has been completed
- Follower precedence: completing τ_p releases τ_a

Competitiveness (static)

- Given a job sequence σ
- Take $\mathrm{Ut}_{\mathcal{A}}(\sigma(k))$ be the utility of scheduler $\mathcal{A}$ in the first k slots
- The goal of $\mathcal{A}$ is to maximize $\mathrm{Ut}_{\mathcal{A}}(\sigma(k))$

Competitiveness (static)

- Given a job sequence σ
- Take $\text{Ut}_{\mathcal{A}}(\sigma(k))$ be the utility of scheduler $\mathcal{A}$ in the first k slots
- The goal of $\mathcal{A}$ is to maximize $\text{Ut}_{\mathcal{A}}(\sigma(k))$

How good is the *worst-case* performance of an online scheduler?

- Utility can be 0 if no tasks are ever released
- Non-informative

Competitiveness (static)

- Given a job sequence σ
- Take $\mathsf{Ut}_{\mathcal{A}}(\sigma(k))$ be the utility of scheduler $\mathcal{A}$ in the first k slots
- The goal of $\mathcal{A}$ is to maximize $\mathsf{Ut}_{\mathcal{A}}(\sigma(k))$

How good is the *worst-case* performance of an online scheduler?

- Utility can be 0 if no tasks are ever released
- Non-informative

Traditionally, captured by the competitive ratio

$$\mathcal{CR}(\mathcal{A}) = \inf_{\mathcal{B},\sigma} \liminf_{k\to\infty} \frac{1 + \mathsf{Ut}_{\mathcal{A}}(\sigma(k))}{1 + \mathsf{Ut}_{\mathcal{B}}(\sigma(k))}$$

- "The smallest ratio of the utility of $\mathcal{A}$ over the utility of an offline scheduler $\mathcal{B}$"
- Note: $\mathcal{A}$ and $\mathcal{B}$ operate on the same sequence σ

Competitiveness (dynamic)

Problem

With *dynamic* task releases (precedences) the job sequences of online and offline schedulers might diverge!

E.g., a follower task is only seen by the scheduler that completes the precursor tasks

Competitiveness (dynamic)

Problem

With *dynamic* task releases (precedences) the job sequences of online and offline schedulers might diverge!

E.g., a follower task is only seen by the scheduler that completes the precursor tasks

This Paper

How do we

1. define, and
2. automatically compute

the competitive ratio in the presence of precedences?

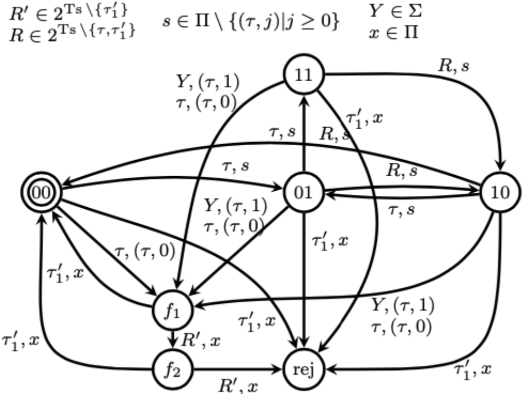
Safety Monitors

Observation: Precedences can be monitored in finite-state

A precedence C can be formally specified by a *safety monitor* $\mathcal{S}^{\mathsf{C}}$

- Input alphabet is $\Theta = \Sigma \times \Pi$
  - Σ is the set of possible tasks released in each step
  - Π is the set of possible scheduling decisions on previously released and non-completed tasks
  - If C is not satisfied by the environment, $\mathcal{S}^{\mathsf{C}}$ enters a special *reject state*

Pairing precedence: Completion of τ modifies the next release of τ_1 in the interval $[1, 2]$ to τ_1'



Global Safety Monitor

- $\overline{\mathcal{S}}$ is global safety monitor tracking all precedences
- Given a scheduler $\mathcal{A}$, $\mathcal{A}[\sigma]$ is the schedule on job sequence σ
- Write $\sigma \models \mathcal{A}, \overline{\mathcal{S}}$ to denote that $\overline{\mathcal{S}}$ accepts $(\sigma, \mathcal{A}[\sigma])$
- I.e., σ satisfies the precedences of $\overline{\mathcal{S}}$ for the schedule produced by $\mathcal{A}$

Job Sequence Compatibility

Split the taskset Ts into

- Ts_b is the *baseline* taskset
  - Contains independent tasks
- Ts_f is the *follower* taskset
  - Contains tasks that can be released only as a consequence of a follower precedence
- Ts_p is the *pairing* taskset
  - Contains the paired version τ' of each task τ that is a consequence to a pairing precedence
  - A grounding function f maps τ' to τ

Job Sequence Compatibility

Split the taskset Ts into

- Ts_b is the *baseline* taskset
  - Contains independent tasks
- Ts_f is the *follower* taskset
  - Contains tasks that can be released only as a consequence of a follower precedence
- Ts_p is the *pairing* taskset
  - Contains the paired version τ' of each task τ that is a consequence to a pairing precedence
  - A grounding function f maps τ' to τ

Compatible Sequences

Two job sequences σ_1 and σ_2 are *compatible* $\sigma_1 \bowtie \sigma_2$ iff

$$\left(\sigma_1^\ell \cup f(\sigma_1^\ell)\right) \cap \mathsf{Ts}_b = \left(\sigma_2^\ell \cup f(\sigma_2^\ell)\right) \cap \mathsf{Ts}_b$$

"At every slot ℓ, baseline tasks and groundings of pairing tasks should coincide in σ_1, σ_2"

Putting it all Together

Competitive ratio under precedences

$$\mathcal{CR}(\mathcal{A}) = \inf_{\substack{\mathcal{B},\sigma_{\mathcal{A}},\sigma_{\mathcal{B}}:\\ \sigma_{\mathcal{A}} \bowtie \sigma_{\mathcal{B}}\\ \sigma_{\mathcal{A}} \models \mathcal{A},\overline{\mathcal{S}}\\ \sigma_{\mathcal{B}} \models \mathcal{B},\overline{\mathcal{S}}}} \liminf_{k\to\infty} \frac{1+\mathsf{Ut}_{\mathcal{A}}(\sigma_{\mathcal{A}}(k))}{1+\mathsf{Ut}_{\mathcal{B}}(\sigma_{\mathcal{B}}(k))}$$

- We have defined competitiveness under precedences
- How to compute it automatically given an online scheduler and a taskset?

Schedulers as Labeled Transition Systems

- Let Dl_{max} be the maximum deadline
- No need to remember task releases more than Dl_{max} slots ago
- Finite state!

Schedulers as Labeled Transition Systems

- Let $\mathsf{Dl}_{\mathsf{max}}$ be the maximum deadline
- No need to remember task releases more than $\mathsf{Dl}_{\mathsf{max}}$ slots ago
- Finite state!

<u>Online scheduler</u>

- Represented as a deterministic labeled-transition system
- Input alphabet is $\Theta = \Sigma$, the set of possible tasks released in each step
- Output alphabet is $\Xi = \Pi$, is the set of possible scheduling choices for tasks released in the last $\mathsf{Dl}_{\mathsf{max}}$ slots

Schedulers as Labeled Transition Systems

- Let Dl_{max} be the maximum deadline
- No need to remember task releases more than Dl_{max} slots ago
- Finite state!

Online scheduler

- Represented as a deterministic labeled-transition system
- Input alphabet is $\Theta = \Sigma$, the set of possible tasks released in each step
- Output alphabet is $\Xi = \Pi$, is the set of possible scheduling choices for tasks released in the last Dl_{max} slots

Offline scheduler

- Offline scheduler $\rightarrow$ Online, but *non-deterministic*
- Represented as a finite-state labeled transition system

Main Contribution

- Schedulers, precedences, job sequence compatibility, all represented as finite state automata
- Take their Cartesian product $\mathcal{P}$
- Competitive ratio reduces to the minimum mean cycle problem on the state space of $\mathcal{P}$

Main Contribution

- Schedulers, precedences, job sequence compatibility, all represented as finite state automata
- Take their Cartesian product $\mathcal{P}$
- Competitive ratio reduces to the minimum mean cycle problem on the state space of $\mathcal{P}$

Theorem

The competitive ratio $\mathcal{CR}(\mathcal{A})$ can be computed in $O((n \cdot m) \cdot \log(n \cdot \mathrm{Ut}_{\mathrm{max}}))$ time, where

- *n is the number of states in $\mathcal{P}$*
- *m is the number of transitions in $\mathcal{P}$*
- *$\mathrm{Ut}_{\mathrm{max}}$ is the maximum utility of all tasks*

Main Contribution

- Schedulers, precedences, job sequence compatibility, all represented as finite state automata
- Take their Cartesian product $\mathcal{P}$
- Competitive ratio reduces to the minimum mean cycle problem on the state space of $\mathcal{P}$

Theorem

The competitive ratio $\mathcal{CR}(\mathcal{A})$ can be computed in $O((n \cdot m) \cdot \log(n \cdot \mathrm{Ut}_{\max}))$ time, where

- *n is the number of states in $\mathcal{P}$*
- *m is the number of transitions in $\mathcal{P}$*
- $\mathrm{Ut}_{\max}$ *is the maximum utility of all tasks*

In the paper: A parallel algorithm (CUDA)

Experiments

Example Scheduling Scenarios with Precedences

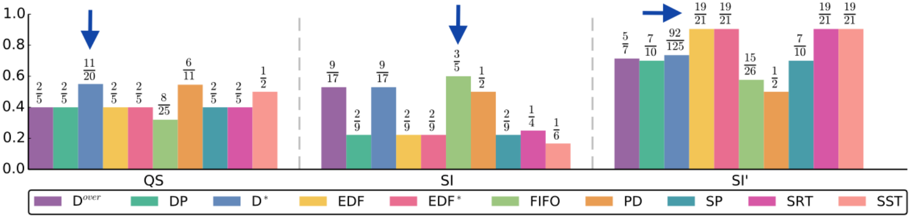
1. Packet Switching (PS)
   - Packet consists of a header τ_h, data τ_d
   - Pairing precedence: τ_d has positive utility only if paired with τ_h
2. Handshake Protocols (HP)
   - Handshake consists of a payload message τ_p and an acknowledgment τ_a
   - Follower precedence: τ_a released iff τ_p is completed
3. Sporadic Interrupts (SI)
   - Periodic worker τ_w, interrupt τ_i
   - Pairing precedence: workload and utility of τ_w depend on preceding interrupt
4. Query Scheduling (QS)
   - Completing query τ_1 releases τ_3
   - Completing τ_1 and either τ_2 or τ_3 releases τ_4
   - Follower precedence: τ_3 released iff τ_1 is completed
   - Follower precedence: τ_4 released iff τ_1 and either τ_2 or τ_3 are completed
   - Pairing precedence: zeros the utility of one τ_4 release when all τ_1, τ_2, τ_3 are completed

10 Schedulers Tested

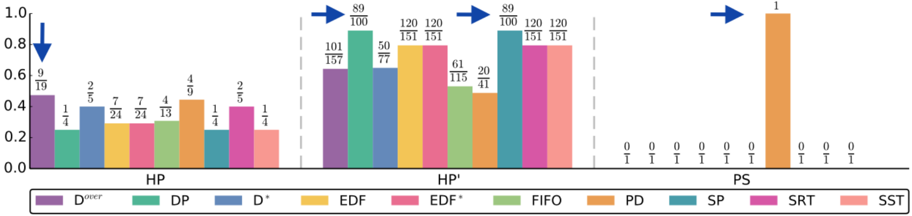
1. Earliest Deadline First (EDF)
2. Earliest Deadline First (EDF*)
3. First-in First-out (FIFO)
4. Static Priorities (SP)
5. Dynamic Priorities (DP)

1. Smallest Remaining Time (SRT)
2. Profit Density (PD)
3. Smallest Slack Time (SST)
4. D-over (D^{over})
5. D-star (D^*)

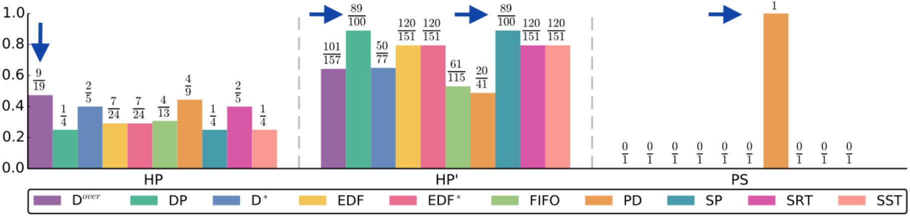
Competitive Ratios (1)



Competitive Ratios (2)



Competitive Ratios (2)



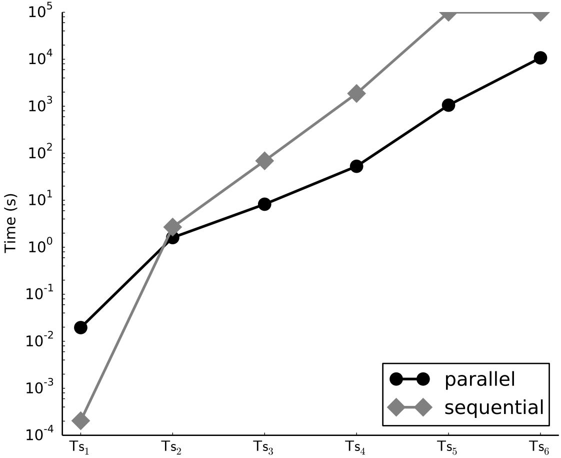
- Competitive ratio varies drastically per scheduler/taskset
- Very hard to predict/analyze by hand

Effect of Parallelism

- 3072 cores
- How much speedup?

Effect of Parallelism

- 3072 cores
- How much speedup?



Conclusion

1. Competitiveness characterizes a real-time scheduler's performance
2. Automated techniques for competitiveness can be very instrumental
3. Research in fairly early stages

Conclusion

1. Competitiveness characterizes a real-time scheduler's performance
2. Automated techniques for competitiveness can be very instrumental
3. Research in fairly early stages

<u>This work</u>

1. A framework for *formal*, *automated* competitive analysis
2. Uniprocessor, firm deadlines
3. *Precedences* capture dynamic interaction between tasks
4. Parallel implementation based on CUDA
5. Results show competitiveness is very intricate in presence of precedences
6. Tool support is instrumental