Real-time Scheduling

How well is the scheduler handling requests?

→ Competitive analysis

Usually done manually, some recent efforts in automated techniques
Real-time Scheduling

- How well is scheduler handling requests? → Competitive analysis
- Usually done manually, some recent efforts in automated techniques
Scheduling Dynamic Tasks

Challenge: How do we compute competitiveness automatically?
Challenge

How to we compute competitiveness automatically?
A framework for push-button computation of competitiveness in precedence-aware scheduling
Scheduling Model

- Discrete time
- Input: a collection of tasks $T_s = \{\tau_1, \ldots, \tau_n\}
- Each $\tau_i = (E_i, D_i, U_i, Np_i)$
  - $E_i \in \mathbb{N}^+$ is the WCET
  - $D_i \in \mathbb{N}$ is the relative deadline
  - $U_i \in \mathbb{N}$ is the utility value
  - $Np_i = \{[l_1, l_2], \ldots, [l_{2k-1}, l_{2k}]\}$ 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

\( \tau_1 \) release

\( \tau_1(1) \)

non-preemptible \( \tau_1 \) deadline

\( \tau_1(2) \) \( \tau_1(3) \)

\( \tau_1 \) utility
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 $\tau$, modified $\tau'$

Semantics

- If all precursor tasks $\tau_1, \ldots, \tau_k$ are completed now, if dependent task $\tau$ is released between $t_1$ and $t_2$ slots, it is modified to $\tau'$
1. Pairing Precedences (Example)

Packet Switching

- A packet consists of a header fragment $\tau_h$ and a data fragment $\tau_d$.
- Serving the data contributes to utility iff the header has been completed.
- Pairing precedence: completing $\tau_h$ modifies the utility of the next release of $\tau_d$ to non-zero.
2. Follower Precedences

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

**Semantics**

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

- Payload message $\tau_p$ and ack message $\tau_a$
- Ack is sent only if the payload message has been completed
- Follower precedence: completing $\tau_p$ releases $\tau_a$
Competitiveness (static)

- Given a job sequence $\sigma$
- Take $U_{t_A}(\sigma(k))$ be the utility of scheduler $A$ in the first $k$ slots
- The goal of $A$ is to maximize $U_{t_A}(\sigma(k))$
Competitiveness (static)

- Given a job sequence $\sigma$
- Take $U_tA(\sigma(k))$ be the utility of scheduler $A$ in the first $k$ slots
- The goal of $A$ is to maximize $U_tA(\sigma(k))$

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 $\sigma$
- Take $U_{A}(\sigma(k))$ be the utility of scheduler $A$ in the first $k$ slots
- The goal of $A$ is to maximize $U_{A}(\sigma(k))$

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**

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

- “The smallest ratio of the utility of $A$ over the utility of an offline scheduler $B$”
- Note: $A$ and $B$ operate on the same sequence $\sigma$
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?
Observation: Precedences can be monitored in finite-state

A precedence $C$ can be formally specified by a safety monitor $S^C$

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

Pairing precedence: Completion of $\tau$ modifies the next release of $\tau_1$ in the interval $[1, 2]$ to $\tau_1'$.
$\overline{S}$ is global safety monitor tracking all precedences

Given a scheduler $\mathcal{A}$, $\mathcal{A}[\sigma]$ is the schedule on job sequence $\sigma$.

Write $\sigma \models \mathcal{A}, \overline{S}$ to denote that $\overline{S}$ accepts $(\sigma, \mathcal{A}[\sigma])$.

I.e., $\sigma$ satisfies the precedences of $\overline{S}$ 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 $\tau'$ of each task $\tau$ that is a consequence to a pairing precedence
  - A grounding function $f$ maps $\tau'$ to $\tau$

"At every slot $\ell$, baseline tasks and groundings of pairing tasks should coincide in $\sigma_1, \sigma_2$"
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 $\tau'$ of each task $\tau$ that is a consequence to a pairing precedence
  - A grounding function $f$ maps $\tau'$ to $\tau$

Compatible Sequences

Two job sequences $\sigma_1$ and $\sigma_2$ are compatible $\sigma_1 \trianglerighteq \sigma_2$ iff

\[(\sigma_1^\ell \cup f(\sigma_1^\ell)) \cap Ts_b = (\sigma_2^\ell \cup f(\sigma_2^\ell)) \cap Ts_b\]

“At every slot $\ell$, baseline tasks and groundings of pairing tasks should coincide in $\sigma_1, \sigma_2$”
Putting it all Together

Competitive ratio under precedences

\[
CR(A) = \inf_{B, \sigma_A, \sigma_B: \sigma_A \preceq \sigma_B, \sigma_A \models A, \overline{S}, \sigma_B \models B, \overline{S}} \liminf_{k \to \infty} \frac{1 + Ut_A(\sigma_A(k))}{1 + Ut_B(\sigma_B(k))}
\]
We have defined competitiveness under precedences
How to compute it automatically given an online scheduler and a taskset?
Schedulers as Labeled Transition Systems

- Let $D_{\text{max}}$ be the maximum deadline
- No need to remember task releases more than $D_{\text{max}}$ slots ago
- Finite state!
Schedulers as Labeled Transition Systems

- Let $D_{\text{max}}$ be the maximum deadline
- No need to remember task releases more than $D_{\text{max}}$ 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 $D_{\text{max}}$ slots
Schedulers as Labeled Transition Systems

- Let $D_{\text{max}}$ be the maximum deadline
- No need to remember task releases more than $D_{\text{max}}$ 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 $D_{\text{max}}$ slots

**Offline scheduler**

- Offline scheduler → 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 $P$
- Competitive ratio reduces to the minimum mean cycle problem on the state space of $P$

Theorem

The competitive ratio $\text{CR}(A)$ can be computed in $O((n \cdot m) \cdot \log(n \cdot U_{\text{max}}))$ time, where $n$ is the number of states in $P$, $m$ is the number of transitions in $P$, and $U_{\text{max}}$ is the maximum utility of all tasks.

In the paper: A parallel algorithm (CUDA)
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 $CR(A)$ can be computed in $O((n \cdot m) \cdot \log(n \cdot Ut_{\text{max}}))$ time, where

- $n$ is the number of states in $\mathcal{P}$
- $m$ is the number of transitions in $\mathcal{P}$
- $Ut_{\text{max}}$ 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 $CR(A)$ can be computed in $O((n \cdot m) \cdot \log(n \cdot Ut_{\text{max}}))$ time, where

- $n$ is the number of states in $\mathcal{P}$
- $m$ is the number of transitions in $\mathcal{P}$
- $Ut_{\text{max}}$ is the maximum utility of all tasks

In the paper: A parallel algorithm (CUDA)
Experiments
Example Scheduling Scenarios with Precedences

Packet Switching (PS)
- Packet consists of a header $\tau_h$, data $\tau_d$
- Pairing precedence: $\tau_d$ has positive utility only if paired with $\tau_h$

Handshake Protocols (HP)
- Handshake consists of a payload message $\tau_p$ and an acknowledgment $\tau_a$
- Follower precedence: $\tau_a$ released iff $\tau_p$ is completed

Sporadic Interrupts (SI)
- Periodic worker $\tau_w$, interrupt $\tau_i$
- Pairing precedence: workload and utility of $\tau_w$ depend on preceding interrupt

Query Scheduling (QS)
- Completing query $\tau_1$ releases $\tau_3$
- Completing $\tau_1$ and either $\tau_2$ or $\tau_3$ releases $\tau_4$
- Follower precedence: $\tau_3$ released iff $\tau_1$ is completed
- Follower precedence: $\tau_4$ released iff $\tau_1$ and either $\tau_2$ or $\tau_3$ are completed
- Pairing precedence: zeros the utility of one $\tau_4$ release when all $\tau_1, \tau_2, \tau_3$ 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)

Very hard to predict/analyze by hand

Precedence-aware Automated Competitive Analysis of Real-time Scheduling
- 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?

![Graph showing the effect of parallelism with time on the y-axis and Ts1 to Ts6 on the x-axis.](graph.png)
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

This work

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