A Response-Time Analysis for Non-Preemptive Job Sets under Global Scheduling

Mitra Nasri  Geoffrey Nelissen  Björn B. Brandenburg
Our work in a nutshell

- Multiprocessor platform
- Per job best-case and worst-case response time (BCRT and WCRT)
- Global job-level fixed-priority (JLFP) work-conserving scheduling policies
- Non-preemptive job sets
- Hard or soft timing constraints
- Release jitter
- Execution time variation
- Bounded jitter
- Bounded variation
- Non-deterministic release time

Applicable to:
- Irregular release patterns
- Bursty releases
- And periodic tasks with/without offset

Global EDF
Global fixed priority
Global RM
...
Why non-preemptive scheduling for multiprocessor platforms?

- Preserves data affinity of local **caches**
- Makes **synchronization** easy (e.g., resolves lock-holder preemption problem)
- Reduces **context switches** and scheduling overheads

- Reduces the worst-case execution times (WCET)
- Improves the accuracy of estimating the WCET by simplifying the execution

**Hence, it can be used to make multiprocessor platforms more (time) predictable**
State of the art on global non-preemptive scheduling

<table>
<thead>
<tr>
<th>Finite job sets*</th>
<th>Periodic tasks with offset</th>
<th>Synchronous periodic tasks</th>
<th>Sporadic tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>open problem</td>
<td>open problem</td>
<td>open problem</td>
<td>Exact analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Open problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sufficient analyses</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Global fixed-priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[Baruah06, Guan08, Guan11, Lee14, Lee17]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Global EDF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[Baruah06, Guan08]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• General work-conserving policy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[Baruah06, Guan08]</td>
</tr>
</tbody>
</table>

We derive a response-time bound for these cases

Applicable to:
- Irregular release patterns
- Bursty releases
- Frame-based tasks
- ...

* In finite job sets, each job is known by its release time, release jitter, best-case and worst-case execution times, and deadline
JLFP: job-level fixed-priority
A response-time analysis for a wide class of global scheduling policies based on searching the space of possible schedules

We use and extend the notion of schedule-abstraction graphs [RTSS’17]
(recently introduced to analyze uniprocessor non-preemptive scheduling)

Schedule-Abstraction Graphs
(definition, usage, and construction)
Key challenges in the schedulability analysis of job sets (with non-deterministic parameters)

Since there is **no periodicity assumption** about job releases, finding a **worst-case scenario** is fundamentally hard.

Naively enumerating all possible **combinations of release times** and **execution times** (a.k.a. execution scenarios) is **not practical**.

---

What is a schedule-abstraction graph?

“schedule-abstraction graph” [RTSS’17] is a technique that allows us to aggregate “similar” schedules while searching for all possible schedules.

Hence, it reduces the search space.

What is a schedule-abstraction graph?

A path in the graph represents an ordered set of dispatched jobs.

**Initial state:**
No job has been dispatched.

**Final state:**
Every path includes all jobs.

What is a schedule-abstraction graph?

A **path** in the graph represents an ordered set of dispatched jobs.

A **vertex** abstracts a system state.
An **edge** abstracts a dispatched job.

A path in the graph represents an ordered set of dispatched jobs. A vertex abstracts a system state. An edge abstracts a dispatched job.

A vertex abstracts a system state (before dispatching $J_3$). A vertex abstracts a system state (after dispatching $J_3$).

$J_3$ finishes any time during [5, 10].

What is a schedule-abstraction graph?

A path in the graph represents an ordered set of dispatched jobs.

A vertex abstracts a system state. An edge abstracts a dispatched job.

A state represents the finish-time interval of any path reaching that state.

\[ J_2 \text{ finishes in } [12, 25] \]

\[ J_1 \text{ finishes in } [10, 15] \]

\[ v_p: [10, 25] \]

- Processor is certainly available after time 25
- Processor is certainly busy before time 10

How to use a schedule-abstraction graph?

The **worst-case (best-case) response time** of a job $J_i$ is its largest (smallest) finish time among **all edges whose label is $J_i$**.

**Example for job $J_2$**

$J_2$ finishes in $[3, 10]$  
$J_2$ finishes in $[12, 25]$  
$J_2$ finishes in $[20, 24]$  
$J_2$ finishes in $[14, 18]$  
$J_2$ finishes in $[16, 28]$

**BCRT** = 3  
**WCRT** = 28

How to build a schedule-abstraction graphs?

[RTSS’17] used a breadth-first strategy

Repeat until every path includes all jobs
1. Find the shortest path
2. For each not-dispatched job that can be dispatched after the path:
   2.1. Expand (add a new vertex)
   2.2. Merge (if possible, merge the new vertex with an existing vertex)

Initial state

Overview of the solution

**Goal:** define and build a schedule-abstraction graph for global scheduling policies

**SYSTEM ABSTRACTION**
(What is the system state? What is on the edges?)

**EXPANSION RULES**
(How to select jobs that can be dispatched “next” by the scheduling policy at any state?)

**MERGING RULES**
(When and how to merge two states?)

In the talk

Our prior work in [RTSS’17] was for uniprocessor system

Its state definition and expansion and merging rules are not applicable to multiprocessor scheduling

Definition of state

\[ v_i = \begin{cases} \varphi_1 : [EFT_1, LFT_1] \\ \varphi_2 : [EFT_2, LFT_2] \\ \vdots \\ \varphi_m : [EFT_m, LFT_m] \end{cases} \]

Example:

\[ \nu_p : \begin{cases} \varphi_1 : [10, 20] \\ \varphi_2 : [30, 40] \end{cases} \]

Core \( \varphi_1 \) is certainly not available before time 10

Core \( \varphi_1 \) is possibly available from time 10

Core \( \varphi_1 \) is certainly available from time 20

The earliest finish time of the job running on this core

The latest finish time of the job running on this core

One interval for each of the \( m \) cores

The earliest finish time of the job running on this core
Definition of expansion rules (for global multiprocessor scheduling)

Rule 1: work-conserving scheduler
If at time \( t \) there is a certainly released job and a certainly available core, a job will be dispatched at time \( t \).

Rule 2: job-level fixed-priority scheduler
A lower priority job cannot be dispatched as soon as a higher-priority job is certainly released and not yet scheduled.

(eligible jobs)
Which jobs **may possibly** be dispatched “next” on each of the cores?

What is the **new state**?
Finding “eligible” jobs

For each not-scheduled job $J_i$ on each core $\varphi_k$

1. Find the **earliest start time** (EST) of $J_i$ on $\varphi_k$

2. Find the **latest start time** (LST) of $J_i$ on any core for a **work-conserving** and **JLFP policy**

3. If $\text{EST} \leq \text{LST}$ then add an edge for job $J_i$ dispatched on core $\varphi_k$

**Example:** is $J_{\text{low}}$ eligible on each core $\varphi_1$?

---

**Table:**

<table>
<thead>
<tr>
<th>Job</th>
<th>Release time</th>
<th>Deadline</th>
<th>Execution time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min  Max</td>
<td></td>
<td>Min Max</td>
<td></td>
</tr>
<tr>
<td>$J_{\text{low}}$</td>
<td>5 15</td>
<td>50</td>
<td>2 15</td>
<td>low</td>
</tr>
<tr>
<td>$J_{\text{high}}$</td>
<td>12 20</td>
<td>45</td>
<td>1 10</td>
<td>high</td>
</tr>
</tbody>
</table>

---

**Diagram:**

- **EST**: Earliest start time
- **time**: Time axis
- **$\nu_p$**: Core $\varphi_1$ and $\varphi_2$
- **$J_{\text{low}}$** and **$J_{\text{high}}$**: Jobs
- **$J_{\text{low}}$**: Release time 5, 15; Deadline 50; Execution time 2, 15; Priority low
- **$J_{\text{high}}$**: Release time 12, 20; Deadline 45; Execution time 1, 10; Priority high
Finding “eligible” jobs

For each not-scheduled job $J_i$ on each core $\varphi_k$

1. Find the **earliest start time** (EST) of $J_i$ on $\varphi_k$

2. Find the **latest start time** (LST) of $J_i$ on any core for a **work-conserving** and **JLFP policy**

3. If $\text{EST} \leq \text{LST}$ then add an edge for job $J_i$ dispatched on core $\varphi_k$

**Example:** is $J_{low}$ eligible on each core $\varphi_1$?

Merging rules and other details in the paper...

<table>
<thead>
<tr>
<th>Job</th>
<th>Release time</th>
<th>Deadline</th>
<th>Execution time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>$J_{low}$</td>
<td>5</td>
<td>15</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>$J_{high}$</td>
<td>12</td>
<td>20</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
Empirical Evaluation
Main questions

How much the proposed analysis improves schedulability over the state of the art?

Which state of the art?
- For most cases that we cover, there is no prior test.
- So we compare against sporadic tests

Does the proposed analysis scale (in terms of runtime) to practical workload sizes?
Evaluation setup

Baseline tests (designed for sporadic tasks)
- Baruah-EDF [Baruah’06] for Global-EDF
- Guan-Test1-WC [Guan’11] for general work-conserving scheduling policies
- Guan-Test2-FP [Guan’11] for Global-FP
- Lee-FP [Lee’17] for Global-FP

We used rate-monotonic priorities for all fixed-priority policies

Periodic task set generation
- Periods randomly chosen from $[10000, 100000] \mu s$ with log-uniform distribution
- Utilizations are obtained from RandFixSum
- Release jitter options: {no jitter, small jitter of $100 \mu s$}
- BCET = $0.1 \cdot WCET$
- A task set with more than 100000 jobs per hyperperiod is discarded

Experiment platform
- Intel Xeon E7-8857 v2 processor
- 3 GHz clock speed and 1.5 TiB RAM

Schedulability improvements

10 tasks, 4 cores, varying utilization

More than 60 percentage point improvement in detecting schedulable task sets
Schedulability improvements

10 tasks, $U = 2.8$, varying number of cores

More than 70 percentage point improvement in detecting schedulable task sets
More than 43 percentage point improvement in detecting schedulable task sets.
Runtime of the analysis

- 10 tasks, 4 cores, varying utilization
- 4 cores, U = 2.8, varying number of tasks
- 10 tasks, U = 2.8, varying number of cores

- Experiment performed on Intel Xeon E7-8857 v2 processor 3 GHz clock speed and 1.5 TiB RAM
- A single-threaded implementation
Runtime of the analysis

The analysis has acceptable runtime for small- and medium-sized workloads.
Conclusions and future directions
Summary

Goal

A response time analysis for non-preemptive job sets scheduled by global JLFP policies

Solution

We introduced a schedule-abstraction graph for global multiprocessor scheduling

What did we get?

Up to 70 percentage point improvement in schedulability ratio (w.r.t. the baseline analyses for sporadic tasks)
Road map and future directions

- Other scheduling policies (e.g., dynamic job-priority policies)
- Conditional or dynamic precedence constraints
- Shared resources
- Cache-related preemption delay
- Integrating with safety and reliability analysis (accounting for fault tolerance methods and failures)

Supporting precedence constraints (under submission)
This work
Our analysis

- Multiprocessor platform
- Hard or soft timing constraints
- Release jitter
- Non-preemptive jobs

Per job **best-case and worst-case response time (BCRT and WCRT)**

Global job-level fixed-priority work-conserving scheduling policies

Execution time variation

- Mitra Nasri
- Geoffrey Nelissen
- Björn B. Brandenburg

Thank you