Compositional Virtual Timeline: Verifying Dynamic-Priority Partitions with Algorithmic Temporal Isolation

Mengqi Liu\textsuperscript{1}, Zhong Shao, Hao Chen, Man-Ki Yoon\textsuperscript{2}, Jung-Eun Kim\textsuperscript{2}
Yale University

1. Mengqi Liu is now at Alibaba
2. Man-Ki Yoon and Jung-Eun Kim are now at North Carolina State University
Real-Time Systems and Temporal Isolation

Real-time systems power our world today
Real-Time Systems and Temporal Isolation

Modern systems may integrate components from various vendors

- Flight Controller — In-House
- Visual Recognition — Vendor A
- Navigation — Vendor B
- Payload Handling — Vendor C
- …..
Real-Time Systems and Temporal Isolation

Integration opens the door to security vulnerability

through real-time scheduling

- Flight Controller
- Visual Recognition (Mal)
- Navigation
- Payload Handling

---

In-House

Vendor A

Vendor B

Vendor C
Our Contributions

• **Braided virtual timeline**: a novel language-based abstraction for the formal reasoning of dynamic-priority schedulers

• A fully **verified real-time OS kernel** with budget-enforcing EDF partitions

• A mechanized proof of **temporal isolation between partitions** (no information leakage through real-time scheduling)

• Artifact available: [https://flint.cs.yale.edu/publications/compvtl.html](https://flint.cs.yale.edu/publications/compvtl.html)
Real-Time Systems and Temporal Isolation

Isolation is key in ensuring the security of modern real-time systems

- **Memory resource**: isolation through virtual memory

- **Time resource**: isolation through real-time scheduling
  - Real-time **task**: requires budget \((e)\) amount of time within every period \((p)\)
  - Real-time **partition**: hosts *a group of tasks* within the partition budget \((C)\) and period \((T)\)
Background 1: Task-Level Scheduling

Real-time periodic scheduling

- CPU time divided into units of time slots
- Period \( p \): a task repeats its execution during each period
- Budget \( e \): a task executes for (at most) \( e \) time slots during each period

A valid schedule of task
\[ \tau = (e = 3, p = 5) \]
Background 1: Hierarchical Scheduling

- CPU time is divided among partitions
- A partition’s time slots are further divided among tasks

\[ \Pi_0 = (2, 5) \]
\[ \tau_0 = (1, 6) \]
\[ \tau_1 = (2, 15) \]

\[ \Pi_1 = (2, 4) \]

Partition-level scheduling: earliest-deadline-first (EDF)

Task-level scheduling in \( \Pi_0 \)
Challenge 1: Information Flow Vulnerability

Though access to global time is prohibited, $\Pi_0$ (from Vendor A) learns about $\Pi_1$ (from Vendor B) by observing $\Pi_0$’s own tasks.

$\Pi_0 = (2, 5)$

$\tau_0 = (1, 6)$

$\tau_1 = (2, 15)$

$\Pi_1 = (2, 4)$

$\Pi_1$ is turned on

$\Pi_1$ is turned off
Background 2: Static Virtual Timeline [Liu et al POPL20]

- Characterized by a time map $\pi: \mathbb{Z} \to \mathbb{Z}$
- $\pi_i(t)$ means the amount of “available” time to $\Pi_i$ within global time duration $[0, t)$
- Can be “statically” computed from highest-priority to lowest-priority
EDF Scheduling is Compositional

N partitions are schedulable under EDF iff.

$$\sum \frac{C_i}{T_i} \leq 1$$

Fixed-priority scheduling (e.g., RMS) is NOT compositional.

N partitions are schedulable iff.

$$\forall i, k, r_i^k \leq T_i$$

where

$$r_i^{k+1} = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{r_i^k}{T_j} \right\rfloor C_j$$

$$r_i^0 = \sum_{j=1}^{i} C_j$$
Earliest Deadline First (EDF) scheduling

\[ \Pi_0 = (2, 5) \]

\[ \Pi_1 = (4, 9) \]
Earliest Deadline First (EDF) scheduling

\[ \Pi_0 = (2, 5) \]

\[ \Pi_1 = (4, 9) \]
Earliest Deadline First (EDF) scheduling

\[ \Pi_0 = (2, 5) \]

\[ \Pi_1 = (4, 9) \]
Earliest Deadline First (EDF) scheduling

\[ \Pi_0 = (2, 5) \]

\[ \Pi_1 = (4, 9) \]
Earliest Deadline First (EDF) scheduling

$\Pi_0 = (2, 5)$

$\Pi_1 = (4, 9)$
Earliest Deadline First (EDF) scheduling

\[ \Pi_0 = (2, 5) \]

\[ \Pi_1 = (4, 9) \]
Earliest Deadline First (EDF) scheduling

\[ \Pi_0 = (2, 5) \]

\[ \Pi_1 = (4, 9) \]
Challenge 2: Limitations of Static Virtual Timeline

- EDF is compositional and fully utilizes CPU time
- Virtual timeline for EDF scheduling can only be calculated dynamically
Our Approach: EDF Partitions w/ Bound Tasks

• **EDF partitions**: only considers the utilization \( \frac{C_i}{T_i} \) when adding/removing a partition. Can schedule partitions that are not schedulable under fixed-priority policy.

• **Bound tasks**: for each task in \( \Pi_i \), the task period is a multiple of \( T_i \)

• This is sufficient for ensuring temporal isolation of partitions. **BUT** we still have to formally prove it.
Verification Overview

Concrete Scheduler

Abstract Scheduler with PriQ

Abstract Scheduler with Braided Virtual Timeline

Abstract Scheduler with Oblivious Isched

Temporal Isolation of Partitions

Schedulability proof of an EDF scheduler
Concrete Scheduler

\[(t, Q_{\text{partition}}, Q_{\text{task}})\]

\[\text{lsched} \quad \text{psched}\]

Abstract Scheduler with Oblivious

\[(t, Q_{\text{partition}}, Q_{\text{task}})\]

\[\text{lsched} \quad \text{abs}_{\text{psched}}\]

Abstract Scheduler with Braided Virtual Timeline

\[(t, \pi, Q_{\text{task}})\]

\[\text{lsched} \quad \text{int}_{\text{psched}} \quad \text{abs}_{\text{psched}}\]

Schedulability proof of an EDF scheduler

Temporal Isolation of Partitions

Abstract Scheduler with Oblivious Isched

\[(t, \pi, Q_{\text{task}})\]

\[\text{obliv}_{\text{lsched}} \quad \text{abs}_{\text{psched}}\]
```c
int psched()
{
    t++;
    // refill partition budgets
    for(int i = 0; i < N; i++){
        if (t % T_i == 0){
            Q_partition[i] = C_i;
        }
    }

    // scheduling
    int pid = N;
    int min_ddl = INT_MAX;
    for(int i = 0; i < N; i++){
        if (Q_partition[i] > 0){
            int ddl = t / T_i * T_i + T_i;
            if (pid == N || ddl < min_ddl){
                pid = i;
                min_ddl = ddl;
            }
        }
    }
    if (pid < N){
        Q_partition[pid]--;
    }
    return pid;
}
```

**Partition-Level EDF Scheduler**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>The current global time</td>
</tr>
<tr>
<td>N</td>
<td>The number of partitions</td>
</tr>
<tr>
<td>T_i</td>
<td>Pre-specified period of partition Π_i</td>
</tr>
<tr>
<td>C_i</td>
<td>Pre-specified budget of partition Π_i</td>
</tr>
<tr>
<td>Q_partition[i]</td>
<td>The remaining budget of partition Π_i</td>
</tr>
</tbody>
</table>
Concrete Scheduler

\[(t, Q_{\text{partition}}, Q_{\text{task}})\]

- lsched
- psched

Abstract Scheduler with PriQ

\[(t, Q_{\text{partition}}, \text{PriQ}, Q_{\text{task}})\]

- Isched
- int_psched

Abstract Scheduler with Braided Virtual Timeline

\[(t, \pi, \text{PriQ}, Q_{\text{task}})\]

- Isched
- abs_psched

Schedulability proof of an EDF scheduler

Temporal Isolation of Partitions

Abstract Scheduler with Oblivious Isched

\[(t, \pi, \text{PriQ}, \text{It}, Q_{\text{task}})\]

- obliv_isched
- abs_psched
int int_psched(){
    t++;
    // refill partition budgets
    for(int i = 0; i < N; i++){
        if (t % T_i == 0){
            Q_partition[i] = C_i;
        }
    }

    // scheduling
    int pid = N;
    for(int p = 0; p < N; p++){
        int i = PriQ_t(p);
        if (Q_partition[i] > 0){
            pid = i;
            break;
        }
    }
    if (pid < N){
        Q_partition[pid]--;  
    }
    return pid;
}
Concrete Scheduler

Abstract Scheduler with PriQ

Abstract Scheduler with Braided Virtual Timeline

Abstract Scheduler with Oblivious lsched

Temporal Isolation of Partitions

Schedulability proof of an EDF scheduler
Further Abstraction: Braided Virtual Timeline

$$\pi_i$$  
Virtual Timeline for partition $\Pi_i$: mapping from global time to accumulative available time

$$I_i(k)$$  
The amount of temporal interference incurred on $\Pi_i$ in the $k$-th period

```c
int abs_psched() {
    t++;
    int pid = N;
    for (p = 0; p < N; p++) {
        int i = PriQ_t(p);
        if (pid == N) {
            \[ \pi_i = \lambda x. (x \geq t)? \pi_i(t - 1) + 1: \pi_i(x); \]
            if \( \pi_i(t) - \pi_i \left( \left\lfloor \frac{t}{T_i} \right\rfloor \right) < C_i \) {
                pid = i;
            }
        } else {
            // Interference incurred
            I_i \left( \left\lfloor \frac{t}{T_i} \right\rfloor \right) ++;
        }
    }
    return pid;
}
```
Contextual Refinement Proof

\[ s_1 = (t, Q) \quad \text{and} \quad s_2 = (t, \pi) \]

\[ R(s_1, s_2) \equiv \forall i, Q[i] = C_i - \min(C_i, \pi_i(t) - \pi_i(\frac{t}{T_i}T_i)) \]

\[ s_1' = (t', Q') \quad \text{and} \quad s_2' = (t', \pi') \]
Contextual Refinement Proof

\( s_1 = (t, Q) \)

\( R(s_1, s_2) \equiv \forall i, Q[i] = C_i - \min(C_i, \pi_i(t) - \pi_i(\lfloor \frac{t}{T_i} \rfloor T_i)) \)

\( s_2 = (t, \pi) \)

\( s_1' = (t', Q') \)

\( R(s_1', s_2') \)

\( s_2' = (t', \pi') \)

\( \Pi_0 = (2, 5) \quad t = 7 \)

\( \Pi_1 = (2, 4) \quad i = 0 \)

\( Q[0] = 2 - \min(2, \pi_0(7) - \pi_0(5)) \)

Partition-level scheduling: earliest-deadline-first (EDF)
Concrete Scheduler

\( (t, Q_{\text{partition}}, Q_{\text{task}}) \)

lsched

psched

Abstract Scheduler with PriQ

\( (t, Q_{\text{partition}}, \text{PriQ}, Q_{\text{task}}) \)

lsched

int\_sched

Abstract Scheduler with Braided Virtual Timeline

\( (t, \pi, \text{PriQ}, Q_{\text{task}}) \)

lsched

Abstract Scheduler with Oblivious lsched

\( (t, \pi, \text{PriQ}, \text{lt}, Q_{\text{task}}) \)

obliv\_lsched

abs\_psched

Temporal Isolation of Partitions

Schedulability proof of an EDF scheduler
Schedulability Proof of EDF

Proof goal: for each partition $\Pi_i$, the amount of available time within each period is greater than or equal to its budget.

$$\forall k \geq 0, \pi_i((k + 1)T_i) - \pi_i(kT_i) \geq C_i$$

- Abstract enough to facilitate the proof
- Proof carries down to the scheduler implementation
Schedulability Proof of EDF

Break down the proof obligation into smaller steps

<table>
<thead>
<tr>
<th>Partition</th>
<th>Budget</th>
<th>Period</th>
<th>Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>Π₀</td>
<td>10</td>
<td>40</td>
<td>25%</td>
</tr>
<tr>
<td>Π₁</td>
<td>10</td>
<td>30</td>
<td>33%</td>
</tr>
<tr>
<td>Π₂</td>
<td>20</td>
<td>50</td>
<td>40%</td>
</tr>
</tbody>
</table>

Lift each \((C_i, T_i)\) by \(\text{LCM} / T_i\)

Then enlarge a partition by \(T_i / \text{GCD}\) one at a time

Then shrink all by \(\text{LCM}/\text{GCD}\) in a single step

\{(150, 600), (200, 600), (240, 600)\} is schedulable

\=> \{(150, 600), (600, 1800), (240, 600)\} is schedulable

\=> \{(600, 2400), (600, 1800), (240, 600)\} is schedulable

\=> \{(600, 2400), (600, 1800), (1200, 3000)\} is schedulable

\=> \{(10, 40), (10, 30), (20, 50)\} is schedulable

\(\text{LCM}(40, 30, 50) = 600\)

\(\text{GCD}(40, 30, 50) = 10\)

\(\text{LCM} / \text{GCD} = 60\)
Schedulability Proof: Enlarging One Partition

- The total interference from other partitions does not increase
- Thus, $\Pi_i$ is still schedulable after enlarged by $k$
Schedulability Proof: Shrinking All Partitions

• If \( \{(kC, kT), \ldots\} \) is schedulable, \( \{(C, T), \ldots\} \) is also schedulable
Concrete Scheduler

\[(t, Q_{\text{partition}}, Q_{\text{task}})\]

\[\text{lsched}\]

\[\text{psched}\]

Abstract Scheduler with PriQ

\[(t, Q_{\text{partition}}, \text{PriQ}, Q_{\text{task}})\]

\[\text{lsched}\]

\[\text{int_psched}\]

Abstract Scheduler with Braided Virtual Timeline

\[(t, \pi, \text{PriQ}, Q_{\text{task}})\]

\[\text{lsched}\]

\[\text{abs_psched}\]

Schedulability proof of an EDF scheduler

Temporal Isolation of Partitions

Abstract Scheduler with Oblivious lsched

\[(t, \pi, \text{PriQ}, \text{lt}, Q_{\text{task}})\]

\[\text{obliv_lsched}\]

\[\text{abs_psched}\]
Local Scheduler for Tasks

```c
int lsched()
{
  for(int j = 0; j < M; j++){
    if (t % p_j == 0){
      Q_task[j] = e_j;
    }
  }

  int tid = M;
  for(int j = 0; j < M; j++){
    if(Q_task[j] > 0){
      tid = j;
      Q_task[j]--; break;
    }
  }

  return tid;
}
```

<table>
<thead>
<tr>
<th></th>
<th>The current global time</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>The number of tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pre-specified period of task $\tau_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_j</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pre-specified budget of task $\tau_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_j</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>The remaining budget of task $\tau_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_task[j]</td>
<td></td>
</tr>
</tbody>
</table>
Concrete Scheduler

\[(t, Q_{\text{partition}}, Q_{\text{task}})\]

\[\text{lsched}\]

\[\text{psched}\]

Abstract Scheduler with PriQ

\[(t, Q_{\text{partition}}, \text{PriQ}, Q_{\text{task}})\]

\[\text{lsched}\]

\[\text{int_psched}\]

Abstract Scheduler with Braided Virtual Timeline

\[(t, \pi, \text{PriQ}, Q_{\text{task}})\]

\[\text{lsched}\]

\[\text{abs_psched}\]

Schedulability proof of an EDF scheduler

Temporal Isolation of Partitions

Abstract Scheduler with Oblivious lsched

\[(t, \pi, \text{PriQ}, \text{lt}, Q_{\text{task}})\]

\[\text{obliv_lsched}\]

\[\text{abs_psched}\]
Abstraction: Oblivious Local Scheduler

```c
int obliv_lsched(){
    for(int j = 0; j < M; j++){
        if (lt % lp_j == 0){
            Q_task[j] = e_j;
        }
    }

    int tid = M;
    for(int j = 0; j < M; j++){
        if(Q_task[j] > 0){
            tid = j;
            Q_task[j]--;
            break;
        }
    }

    return tid;
}
```

<table>
<thead>
<tr>
<th>lt</th>
<th>The local time experienced by the enclosing partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp_j</td>
<td>Equals p_j multiplied by the enclosing partition’s utilization (budget / period)</td>
</tr>
</tbody>
</table>

\[
\Pi = (2, 5), \tau = (2, 15)
\]

The portion of p that is visible to \(\Pi\)
Abstract Scheduler with Oblivious lsched
(t, \(Q_{\text{partition}}\), \(Q_{\text{task}}\))

Abstract Scheduler with Braided Virtual Timeline
(t, \(\pi\), PriQ, \(Q_{\text{task}}\))

Temporal Isolation of Partitions
(t, \(\pi\), PriQ, \(l_t\), \(Q_{\text{task}}\))

Concrete Scheduler
(t, \(Q_{\text{partition}}\), \(Q_{\text{task}}\))

Schedulability proof of an EDF scheduler
Contextual Refinement Proof

\[ s_1 = (t, index, Q) \]
\[ s_2 = (lt, index, Q) \]

\[ lt = \lfloor \frac{t}{T} \rfloor C + \min(C, \pi(t) - \pi(\lfloor \frac{t}{T} \rfloor T)) \]

\[ R(s'_1, s'_2)? \]

\[ s'_1 = (t', index', Q') \]
\[ s'_2 = (lt', index', Q') \]

ret pid_1

ret pid_2
Verification Overview

Concrete Scheduler
(t, Q_{\text{partition}}, Q_{\text{task}})
lsched
psched

Abstract Scheduler with PriQ
(t, Q_{\text{partition}}, PriQ, Q_{\text{task}})
lsched
int_psched

Abstract Scheduler with Braided Virtual Timeline
(t, \pi, PriQ, Q_{\text{task}})
lsched
abs_psched

Schedulability proof of an EDF scheduler

Temporal Isolation of Partitions

Abstract Scheduler with Oblivious lsched
(t, \pi, PriQ, lt, Q_{\text{task}})
obliv_lsched
abs_psched
Challenge 1: Information Flow Vulnerability

Though access to global time is prohibited, Π₀ (from Vendor A) learns about Π₁ (from Vendor B) by observing Π₀’s own tasks.

Π₀ = (2, 5)

\[ \tau_0 = (1, 6) \]
\[ \tau_1 = (2, 15) \]

Π₁ = (2, 4)

Π₁ is turned on

π₁ is turned off

Flipped!
Evaluation: Temporal Isolation

(a) solo partition

(b) w/1 other partition

(c) w/3 other partitions

Scheduling traces of partition \( \{\tau_{4,1}, \ldots, \tau_{4,4}\} \) are equivalent when run with different other partitions.
## Proof Efforts

<table>
<thead>
<tr>
<th>Item</th>
<th>LOC in Coq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formalization of the dynamic virtual time map and related lemmas</td>
<td>2,102</td>
</tr>
<tr>
<td>The schedulability proof on top of the braided virtual timelines</td>
<td>16,170</td>
</tr>
<tr>
<td>Functional correctness proof for the partition-level EDF scheduler's C code</td>
<td>2,876</td>
</tr>
<tr>
<td>Connecting the schedulability proof with the partition-level EDF scheduler</td>
<td>4,876</td>
</tr>
<tr>
<td>Functional correctness proof for the local task scheduler's C code</td>
<td>2,963</td>
</tr>
<tr>
<td>Contextual refinement proof between the local task scheduler and its oblivious abstraction</td>
<td>7,594</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td><strong>36,581</strong></td>
</tr>
</tbody>
</table>
Conclusions

• **Braided virtual timeline**: a novel language-based abstraction for the formal reasoning of dynamic-priority schedulers

• A fully *verified real-time OS kernel* with budget-enforcing EDF partitions

• A mechanized proof of *temporal isolation between partitions* (no information leakage through real-time scheduling)

• Artifact available:  [https://flint.cs.yale.edu/publications/compvtl.html](https://flint.cs.yale.edu/publications/compvtl.html)
Thank you!