Haren: A Framework for Ad-Hoc Thread Scheduling Policies for Data Streaming Applications

Dimitris Palyvos-Giannas, Vincenzo Gulisano, Marina Papatriantafilou

13th International Conference on Distributed and Event-Based Systems
June 24-28, 2019, Darmstadt
Stream Processing Basics

- **Data Source**
- **Tuple**
- **Stream**
- **Operator**
- **Stream Processing Engine (SPE)**

**Performance Metrics**
- Throughput
- Latency
- CPU Utilization
- Memory Utilization
Resource Scheduling

Input

A → B → C → D

E → F

SPE Instance (Process) 1
A B C D

SPE Instance (Process) 2
E F
Thread Scheduling inside SPE Instances

- Operators are executed by **CPU threads**.
- Usually **one dedicated thread per operator**.
- What if **#CPUs < #Operators**?
- **Operating System scheduler** allocates CPU…
- …but it has **no knowledge of streaming goals**!

- **Alternative**: **Application-level thread scheduling**
- Can optimize for specific **performance goals**!
- For a (short) time interval, **two questions**:
  1. How to **assign** operators to threads (inter-thread)?
  2. What is the **priority** of operators for each thread (intra-thread)?

  **Two scheduling functions** – almost any policy!
Custom Thread Scheduling

But there are obstacles…

• **Low-level programming details.**
  • Difficult to program.
  • Difficult to ensure efficiency and correctness.

• Schedulers **programmed to specific SPE.**
  • Reinventing the wheel - cannot reuse code.
  • Difficult to port scheduling policies to other SPEs.

---

**Which operator to execute?**

**It depends! Custom scheduling = more choice!**

Minimize queue sizes

Minimize latency

---

Most SPEs avoid custom thread scheduling!
Haren: A Scheduling Framework for Streaming

Haren hides the complexity of custom scheduling: User only programs high-level scheduling logic!

**Goal 1** Compact interface that allows implementing arbitrary scheduling policies.

**Goal 2** Configuration of both inter-thread and intra-thread rules.

**Goal 3** Parallelization of scheduling computation when possible.

Reusable scheduling policies in different SPEs!
Stream Processing & Scheduling

Haren Framework Overview

Haren Implementation

Evaluation & Conclusions
Haren Overview

- Orchestrates operator execution using a group of Processing Threads (PTs).
- Remains SPE-agnostic by using the features abstraction.
  - Retrieves necessary features of the operators from the SPE through a well-defined interface.
  - A feature is any value that characterizes an operator, its streams or its tuples.
  - Example features: cost, input stream size, …
  - Maintains a table of operator features.
- Applies high-level user-defined scheduling functions to the features to take scheduling decisions.
Feature Categories

**Static** (e.g., operator type) 
*Remain constant*

**Dynamic** (e.g., cost, input stream size) 
*Can change over time*

**Independent** (e.g., cost) 
*Can only change upon execution of Op*

**Dependent** (e.g., input stream size) 
*Can change upon execution of some other operator ≠ Op*
Inter & Intra Thread Scheduling Functions

1. Inter-thread: How to **assign** operators to threads?
2. Intra-thread: How to compute the **priority** of operators in each thread?
Haren Processing Thread (PT) Behavior

- PTs *execute* operators most of the time ($T_E$).
- Dynamic nature of stream processing $\rightarrow$ features & priorities change over time.
- PTs periodically switch to *scheduling*, updating features and scheduling decisions ($T_S$).
- Fine-grained control over scheduling overhead by tuning the scheduling period $P$. 
Stream Processing & Scheduling

Haren Framework Overview

Haren Implementation

Evaluation & Conclusions
Execution Task $T_E$

Processing Thread

**Main Loop**

while running:
  while elapsed_time < scheduling_period:
    starting from the beginning of assigned pick first operator that can run
    (has input > 0 and output capacity > 0)
    if found operator that can run:
      process max $b$ tuples
    if no operator can run:
      back-off (sleep)
  goto scheduling task $T_S$
Scheduling Task $T_S$

- **Decides schedule** for the next $T_E$.
- Computes a **new assigned array** for each PT where operators sorted on priority.
- Most steps are executed in **parallel** by all PTs.
- Few **sequential** steps for PTs to synchronize and agree on scheduling decisions.
**Processing Thread**

Concurrent

```python
for op in Executed
    update independent features of op
    mark op & dependent operators
```

<table>
<thead>
<tr>
<th>Operators</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F1</td>
</tr>
<tr>
<td>B</td>
<td>F2</td>
</tr>
<tr>
<td>C</td>
<td>F3</td>
</tr>
<tr>
<td>D</td>
<td>F4</td>
</tr>
<tr>
<td>E</td>
<td>F5</td>
</tr>
</tbody>
</table>

**Marked Table (Bool)**

(needed for next step)
**T_S**: Dependent Features Update

**Thread t'**

Sequential

```python
for op in All_Operators
    if op is marked
        update dependent features of op
```

Haren only updates features that:
- Have (potentially) changed.
- Are used by the scheduling functions $f, g$.

### Feature Table

<table>
<thead>
<tr>
<th>Operators</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Marked Table

- Processing Threads (PTs)
- Execution Task $T_E$
- Scheduling Task $T_S$
- Independent Features Update
- Dependent Features Update
- Operator Assignment ($f$)
- Priority Computation ($g$)
- Operator Sorting
- Execution Task $T_E$
**Thread t’**

Sequential

\[
\text{for } op \text{ in All Operators} \\
\text{threadID} = f(op) \\
\text{append op to assigned[threadID]}
\]

**Feature Table**

<table>
<thead>
<tr>
<th>Operators</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Assigned Operators per PT**

- PT1
- PT2
- PT3
- PT4

**Processing Threads (PTs)**

- Execution Task \(T_E\)
- Scheduling Task \(T_S\)

- Independent Features Update
- Dependent Features Update
- Operator Assignment \(f\)
- Priority Computation \(g\)
- Operator Sorting

- Execution Task \(T_E\)
**T_S: Priority Computation & Sorting**

**Processing Thread**  

Concurrent

```python
for op in assigned
    priority[op] = g(op)
sort assigned on priority
```

**Assigned (after previous step)**  

```
E  C  A  B  D
```

**Assigned (after priority computation)**  

```
E  C  A  B  D
```

**Assigned (after sort)**  

```
D  E  B  C  A
```

**Processing Threads (PTs)**

- **Execution Task T_E**
- **Scheduling Task T_S**
  - Independent Features Update
  - Dependent Features Update
  - Operator Assignment ($f$)
  - Priority Computation ($g$)
  - Operator Sorting

- **Execution Task T_E**
Evaluation

1. **Performance comparison** of dedicated threads (OS) vs Haren policies.
2. **Scheduling overhead** evaluation.
3. **Multi-Class** scheduling.

Evaluation setup:

- Queries → **chains of operators**.
- **Varying cost** and **selectivity** for each query.
- **Varying parallelism** (#queries).
- **Odroid-XU4** devices.
  - Samsung Exynos5422 Cortex-A15 2Ghz and Cortex-A7 Octa core CPU, 2 GB RAM
  - Resource constrained → Custom scheduling even more important.
- Java Haren implementation.
  - Integrated with the lightweight Liebre SPE ([https://github.com/vincenzo-gulisano/Liebre](https://github.com/vincenzo-gulisano/Liebre))
Evaluation 1: Performance Comparison

Operating System Scheduling (Dedicated Threads)
Unaware of streaming goals

Highest Rate
Optimize mean latency

First-Come-First-Serve
Optimize maximum latency

Chain
Optimize total queue sizes

CPU, Memory and more in the paper...
Evaluation 2: Scheduling Overheads

<table>
<thead>
<tr>
<th>Parallelism</th>
<th>% Time Spent</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2%</td>
</tr>
<tr>
<td>20</td>
<td>2%</td>
</tr>
</tbody>
</table>

**Tasks:**
- Execution Task $T_E$
- Scheduling Task $T_S$

**Features:**
- Independent Features Update
- Dependent Features Update
- Operator Assignment $(\phi)$
- Priority Computation $(\eta)$
- Operator Sorting

**Processing Threads (PTs):**
- Coord (Sequential Part)
Evaluation 3: Multi-Class Scheduling

Haren Scheduling Policy
1. **Prioritize** High Queries over Low Queries
2. Optimize **Max Latency** for High Queries
3. Optimize **Mean Latency** for Low queries

More graphs in the paper!
Conclusions

• **Haren** is an all-purpose framework for scheduling in streaming.
• Easy definition of ad-hoc thread scheduling policies.
• Expressive and efficient, can outperform dedicated threads approach.
• **Parallelizes** scheduling computations.

Dimitris Palyvos-Giannas
palyvos@chalmers.se