EdgeWise: A Better Stream Processing Engine for the Edge

Xinwei Fu, Talha Ghaffar, James C. Davis, Dongyoon Lee

Department of Computer Science
Edge Stream Processing

Internet of Things (IoT)
- Things, Gateways and Cloud

Edge Stream Processing
- Gateways process continuous streams of data in a timely fashion.

Things
- City
- Hospital
- Sensors
- Actuators

Gateways
- Edge Stream Processing

Cloud
- Central Analytics
Our Edge Model

Hardware
- Limited resources
- Well connected

Application
- Reasonable complex operations
- For example, FarmBeats [NSDI’17]

Things
- Sensors
- Actuators

Gateways
- Edge Stream Processing

Cloud
- Central Analytics
Edge Stream Processing Requirements

- Multiplexed - Limited resources
- Low Latency - Locality
- No Backpressure - latency and storage
- Scalable - millions of sensors

Things

- sensors
- actuators

Gateways

- Edge Stream Processing

Cloud

- Central Analytics
Dataflow Programming Model

Topology - a Directed Acyclic Graph
- source
- sink
- operation
- data flow

Deployment
- Describe # of instances for each operation

Things
- city
- hospital

Gateways
- Edge Stream Processing
- Running on gateways

Cloud
- Central Analytics
Runtime System

Stream Processing Engines (SPEs):
- Apache Storm
- Apache Flink
- Apache Heron

One-Worker-per-Operation-Architecture
- Queue and Worker thread
- Pipelined manner
- Backpressure - latency
- storage
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

- **Scalable**: ✔
- **Multiplexed**: ❌
- **Latency**: ❌
- **Backpressure**: ❌

**OWPOA SPEs**
- Cloud-class resources
- OS scheduler

**Edge**
- Limited resources
- # of workers > # of CPU cores
- Inefficiency in OS scheduler

Low input rate → Most queues are empty

High input rate → Most or all queues contain data
→ Scheduling Inefficiency
→ Backpressure
→ Latency
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

- Scalable
- Multiplexed
- Latency
- Backpressure

---

**Single core**

\[ Q_0 \rightarrow Q_1 \]

- Q0
- Q1
- OP0
- OP1

**OWPOA – Random OS Scheduler**
Problem

Existing One-Worker-per-Operation-Architecture Stream Processing Engines are not suitable for the Edge Setting!

Scalable  Multiplexed  Latency  Backpressure

Single core

Q0 -> Q1

OWPOA – Random OS Scheduler
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

- **Scalable**  
- **Multiplexed**  
- **Latency**  
- **Backpressure**

**Single core**

Q0 -> Q1

**OWPOA – Random OS Scheduler**
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

- Scalable
- Multiplexed
- Latency
- Backpressure

**Single core**

\[ Q_0 \rightarrow Q_1 \]

**OWPOA – Random OS Scheduler**
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

- Scalable    - Multiplexed    - Latency    - Backpressure

_**Single core**_

Q0 -> Q1

**OWPOA – Random OS Scheduler**
Existing One-Worker-per-Operation-Architecture Stream Processing Engines are not suitable for the Edge Setting!

- Scalable
- Multiplexed
- Latency
- Backpressure

**Single core**

Q0 -> Q1

*OWPOA – Random OS Scheduler*
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

- Scalable
- Multiplexed
- Latency
- Backpressure

**Single core**

**Q0 -> Q1**

**OWPOA – Random OS Scheduler**
Problem

Existing One-Worker-per-Operation-Architecture Stream Processing Engines are not suitable for the Edge Setting!

- **Scalable**
- **Multiplexed**
- **Latency**
- **Backpressure**

Single core

\[ Q_0 \rightarrow Q_1 \]

**OWPOA – Random OS Scheduler**
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

___Scalable   ___Multiplexed   ___Latency   ___Backpressure

**Single core**

Q0 -> Q1

**OWPOA – Random OS Scheduler**
Problem

Existing **One-Worker-per-Operation-Architecture Stream Processing Engines** are not suitable for the Edge Setting!

- **Scalable**
- **Multiplexed**
- **Latency**
- **Backpressure**

Single core

Q0 -> Q1

**OWPOA – Random OS Scheduler**

OS Scheduler doesn’t have engine-level knowledge.
Key Ideas:

- Inefficiency in OS scheduler → Engine-level scheduler
- # of workers > # of CPU cores → A fixed-sized worker pool
- Inefficiency in OS scheduler → Engine-level scheduler
**EdgeWise – Fixed-size Worker Pool**

Fixed-size Worker Pool
- # of worker = # of cores
- Support an arbitrary topology on limited resources
- Reduce overhead of contending cores

![Topology diagram](image)

**EdgeWise**

- Scalable
- Multiplexed
- Latency
- Alleviate Backpressure

![Worker pool diagram](image)
A Lost Lesson: Operation Scheduling

- Profiling-guided priority-based
- Multiple OPs with a single worker

Carney [VLDB’03]
- Min-Latency Algorithm
- Higher static priority on latter OPs

Babcock [VLDB’04]
- Min-Memory Algorithm
- Higher static priority on faster filters

We should regain the benefit of the engine-level operation scheduling!!!
EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler
- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

**Single core**

Q0 -> Q1

<table>
<thead>
<tr>
<th>Q0</th>
<th>OP0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q1</th>
<th>OP1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EdgeWise – Congestion-Aware Scheduler**

- Scalable
- Multiplexed
- Latency
- Alleviate Backpressure
EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

---

**Single core**

Q0 -> Q1

EdgeWise – Congestion-Aware Scheduler

- Scalable
- Multiplexed
- Latency
- Alleviate Backpressure
EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

• Profiling-free dynamic solution
• Balance queue sizes
• Choose the OP with the most pending data.

Single core
Q0 -> Q1

EdgeWise – Congestion-Aware Scheduler

Scalable  Multiplexed  Latency  Alleviate Backpressure
EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler
• Profiling-free dynamic solution
• Balance queue sizes
• Choose the OP with the most pending data.

**Single core**

Q0 -> Q1

Q0

Q1

OP0

OP1

Time

**EdgeWise – Congestion-Aware Scheduler**

- Scalable
- Multiplexed
- Latency
- Alleviate Backpressure
EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

Single core
Q0 -> Q1

EdgeWise – Congestion-Aware Scheduler

- Scalable
- Multiplexed
- Latency
- Alleviate Backpressure
EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler
- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

![Diagram of EdgeWise – Congestion-Aware Scheduler]

- Scalable
- Multiplexed
- Latency
- Alleviate Backpressure
Performance Analysis using Queueing Theory

Novelty:
To the best of our knowledge, we are the first to apply queueing theory to analyze the improved performance in the context of stream processing.

Conclusion 1:
Maximum end-to-end throughput depends on scheduling heavier operations proportionally more than lighter operations.

Conclusion 2:
Data waits longer in the queues of heavier operations. The growth in wait time is non-linear.

By balancing queue sizes, EdgeWise achieves Higher Throughput and Lower Latency than One-Worker-per-Operation-Architecture.

See our paper for more details.
Evaluation

**Impl:** v1.1.0

**OWPOA Baseline:**

**Experiment Setup:**
Focus on a single

**Schedulers:**
- Random
- Min-Memory
- Min-Latency

4 cores -> a pool of 4 worker threads

**Benchmarks:**
RIoTBench - a real-time IoT stream processing benchmark for Storm.

**Metrics:**
Throughput & Latency

More in the Paper.
Throughput-Latency Performance

PRED topology Throughput-Latency

- Storm
- WP+Random
- WP+MinMem
- WP+MinLat
- EdgeWise

Latency (ms)

Throughput

WP+MinLat
EdgeWise
Fine-Grained Latency Analysis

PRED Latency breakdown in *Storm*

PRED Latency breakdown in *EdgeWise*

**This is not a zero-sum game!**

**Conclusion 2:**

*Data waits longer in the queues of heavier operations. The growth in wait time is non-linear.*
Conclusion

- Study existing SPEs and discuss their limitations in the Edge
- EdgeWise
  - Fixed-size worker pool
  - Congestion-aware scheduler
  - Lost lesson of operation scheduling
- Performance analysis of the congestion-aware scheduler using Queueing Theory
- Up to 3x improvement in throughput while keeping latency low

*Sometimes the answers in system design lie not in the future but in the past.*
Backup Slides
Problem

Existing **OWPOA SPEs** are not suitable for the Edge Setting!

- Scalable
- Multiplexed
- Latency
- No Backpressure

# of instance of each operation can be assigned during the deployment

**Topology**
Performance Analysis using Queueing Theory

Novelty:
To the best of our knowledge, we are the first to apply queueing theory to analyze the improved performance in the context of stream processing.

Prior scheduling works in stream processing either provide no analysis or focus only on memory optimization.
Performance Analysis using Queueing Theory

Conclusion 1:

Maximum end-to-end throughput depends on scheduling heavier operations proportionally more than lighter operations.

\[
\begin{align*}
\lambda_i &= q_i \cdot \lambda_0 \\
\mu_i &= r_i \cdot \mu_0 \\
\rho_i &= \frac{\lambda_i}{\mu_i}
\end{align*}
\]

Scheduling weight

\[
\sum_i w_i = C
\]

Effective service rate

\[
\mu'_i = w_i \cdot \mu_i = w_i \cdot (r_i \cdot \mu_0)
\]

Stable constraint

\[
\forall i, \quad \rho'_i = \frac{\lambda_i}{\mu'_i} = \frac{q_i \cdot \lambda_0}{w_i \cdot r_i \cdot \mu_0} < 1
\]

\[
\forall i, \quad \lambda_0 < \frac{r_i}{q_i} \cdot \mu_0
\]

\[
\begin{align*}
\text{maximize} & \quad \min_i \left( w_i \cdot \frac{r_i}{q_i} \right) \\
\text{subject to} & \quad \sum_i w_i = C
\end{align*}
\]

\[
\frac{q_1 \cdot q_2 \cdot \ldots \cdot q_M}{r_1 \cdot r_2 \cdot \ldots \cdot r_M}
\]

scheduling weight \rightarrow input rate / service rate
Conclusion 2:
A data waits longer in the queues of heavier operations, and crucially the growth in wait time is non-linear.

**End-to-end latency**

\[ Latency = \sum_{i} (L_i + \text{Comm.}) \approx \sum_{i} L_i \]

**Per-operation latency**

\[ L_i = T_Q + (T_S + T_C) \approx T_Q \]

**Queueing time – waiting in the queue (using exponential distribution)**

\[ T_Q = \frac{\rho}{\mu - \lambda} = \frac{\lambda}{\mu(\mu - \lambda)} \]
Evaluation

Benchmarks:
RIoT Bench - a real-time IoT stream processing benchmark for Storm
Modification: a timer-based input generator.

Metrics:
- Throughput  
- Latency
Throughput-Latency Performance

**PRED**

![Graph showing Throughput-Latency performance for PRED with different methods and configurations.]

**TRAIN**

![Graph showing Throughput-Latency performance for TRAIN with different methods and configurations.]

**ETL**

![Graph showing Throughput-Latency performance for ETL with different methods and configurations.]

**STATS**

![Graph showing Throughput-Latency performance for STATS with different methods and configurations.]

Legend:
- Storm
- WP+Random
- WP+MinMem
- WP+MinLat
- EdgeWise
In PRED, as the input rate (throughput) increase, the coefficient of variation (CV) of operation utilization grows in Storm, but it decreases in EdgeWise.

Conclusion 1:
Maximum end-to-end throughput depends on scheduling heavier operations proportionally more than lighter operations.
Data Consumption Policy

Sensitivity study on various consumption policies with STATS topology
Performance on Distributed Edge

PRED: maximum throughput achieved with the *latency less than 100 ms*
Limitations

I/O bound computation:
• The preferred idiom is *outer I/O, inner compute*
• Worker pool size could be tuned
• I/O could be done elsewhere, like Microsoft Bosque
Icon credit for www.flaticon.com