Partition and Compose: Parallel Complex Event Processing (Ten Years Later)

Martin Hirzel, IBM Research
Thursday, 30 June 2022
DEBS
DEBS 2012 Paper

Partition (Parallelism) and Compose (Patterns)
Feeding 2 Birds from 1 Feeder

Partition (Parallelism) and Compose (Patterns)

Speed
Ease of Coding
Stream Processing Optimizations

- **B** Batching
  - Unchanged graph
  - Stable semantics

- **P** Placement
  - Unchanged graph
  - Stable semantics

- **Ss** State sharing
  - Unchanged graph
  - Stable semantics

- **Os** Operator separation
  - Changed graph
  - Stable semantics

- **Or** Operator reordering
  - Changed graph
  - Stable semantics

- **Re** Redundancy elimination
  - Changed graph
  - Stable semantics

- **Fi** Fission
  - Changed graph
  - Unstable semantics

- **Lb** Load balancing
  - Unchanged graph
  - Stable semantics

- **As** Algorithm selection
  - Unchanged graph
  - Unstable semantics

- **Ls** Load shedding
  - Unchanged graph
  - Unstable semantics

⇒ *DEBS 2013 Tutorial and CSUR 2014 article*
# Fission for Parallelism

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
<td>Batching</td>
<td>Unchanged graph</td>
<td>Stable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>3</td>
<td>Placement</td>
<td>Unchanged graph</td>
<td>Stable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ss</td>
<td>4</td>
<td>State sharing</td>
<td>Unchanged graph</td>
<td>Stable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Os</td>
<td>5</td>
<td>Operator separation</td>
<td>Changed graph</td>
<td>Stable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Or</td>
<td>6</td>
<td>Operator reordering</td>
<td>Changed graph</td>
<td>Stable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Re</td>
<td>7</td>
<td>Redundancy elimination</td>
<td>Changed graph</td>
<td>Stable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lb</td>
<td>8</td>
<td>Load balancing</td>
<td>Unchanged graph</td>
<td>Stable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>As</td>
<td>9</td>
<td>Algorithm selection</td>
<td>Unchanged graph</td>
<td>Unstable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ls</td>
<td>10</td>
<td>Load shedding</td>
<td>Unchanged graph</td>
<td>Unstable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fi</td>
<td>11</td>
<td>Fission</td>
<td>Changed graph</td>
<td>Unstable semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

→ partitioning for stable semantics
Partition Parallelism

Up-stream operator \rightarrow \text{Simple events} \rightarrow \text{MatchRegex operator} \rightarrow \text{Composite events} \rightarrow \text{Down-stream operator}

PartitionMap
Partition Parallelism

- Up-stream operator
- Simple events
- MatchRegex operator
- Composite events
- Down-stream operator

Fission

- Up-stream operator
- Simple events
- MatchRegex (replica 0)
- MatchRegex (replica 1)
- MatchRegex (replica 2)
- Composite events
- Down-stream operator

hash-split

PartitionMap

PartitionMap

PartitionMap

PartitionMap
Scenario: Financial analysis

M-shape (double-top) stock pattern

Source: http://www.cs.cornell.edu/bigreddata/cayuga/
Regular Expression

\( \text{rise}^+ \ \text{drop}^+ \ \text{rise}^+ \ \text{drop}^* \ \text{deep} \)
M-Shape pattern in SPL

```java
stream<MatchT> Matches = MatchRegex(Quotes) {
    param
    pattern : ".\ rise+ drop+ rise+ drop* deep";
    partitionBy : symbol;
    predicates : {
        rise = price>First(price) && price>=Last(price),
        drop = price>=First(price) && price<Last(price),
        deep = price<First(price) && price<Last(price) }
    output
    Matches : symbol=symbol, seqNum=First(seqNum),
    count=Count(), maxPrice=Max(price);
}
```
Automaton

\[ \text{. rise+ drop+ rise+ drop* deep} \]
Data Structures

**:SimpleEvent**
- ts
- symbol
- price
- size
- seqNum

**:PartitionMap**
- key
- 0..*

**:PartialMatch**
- state
- aggr

**:Aggr**
- count
- price: first, last, max
- seqNum: first
Speedups

1 Machine x 8 Cores

4 Machines x 8 Cores = 32
Shuffle in twitter02 and twitter03

Source

ParseTweet (replica 0)
MatchRegex (replica 0)

ParseTweet (replica 1)
MatchRegex (replica 1)

ParseTweet (replica 2)
MatchRegex (replica 2)

Down-stream operator

Raw tweets as XML documents
Tweets as simple events
Composite events

ParseTweet (replica 2)
MatchRegex (replica 2)

Source

ParseTweet (replica 1)
MatchRegex (replica 1)

ParseTweet (replica 0)
MatchRegex (replica 0)
Productization

- In IBM Streams product since 2012
- Library operator (no core runtime change)
More Ease of Coding

Partition (Parallelism)

and

Spreadsheet (Formulas)

Speed

Ease of Coding
## Streaming Spreadsheets

**Input Trades**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sym</td>
<td>price</td>
<td>vol</td>
<td>line chart</td>
<td>calculation</td>
<td>price*vol</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ACME</td>
<td>$199.07</td>
<td>842</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ACME</td>
<td>$198.99</td>
<td>718</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ACME</td>
<td>$197.70</td>
<td>773</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ACME</td>
<td>$198.84</td>
<td>496</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ACME</td>
<td>$198.16</td>
<td>424</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ACME</td>
<td>$199.15</td>
<td>737</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ACME</td>
<td>$198.71</td>
<td>664</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ACME</td>
<td>$196.73</td>
<td>736</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>sum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>=SUM(C3:C10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Input Quotes**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sym</td>
<td>price</td>
<td>seller</td>
<td>sym</td>
<td>seller</td>
<td>vwap</td>
<td>isBargain</td>
</tr>
<tr>
<td>15</td>
<td>ACME</td>
<td>$196.96</td>
<td>Alice</td>
<td>ACME</td>
<td>Alice</td>
<td>$198.42</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>=A15</td>
<td>=C15</td>
<td>=G12/C12</td>
<td>=B15&lt;C15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Time-Based Windows

Columns

Rows

Variable-sized windows: occupy one cell, aggregate all elements
Partitioned virtual worksheet: visualize one key ("ACME"), calculate all keys.
Partition Parallelism, Again

Up-stream operator → Spreadsheet operator → Down-stream operator

Fission

Up-stream operator → Spreadsheet (replica 0) → Down-stream operator

Up-stream operator → Spreadsheet (replica 1) → Down-stream operator

Up-stream operator → Spreadsheet (replica 2) → Down-stream operator

hash-split
DEBS 2017 Paper

Speed

Ease of Coding
More Speed with Algorithms

(De-)amortize (incremental) and Aggregates (Monoids)

Speed

Ease of Coding
Algorithm Selection for Incremental Aggregation

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Graph Change</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Batching</td>
<td>Unchanged</td>
<td>Stable</td>
</tr>
<tr>
<td>P</td>
<td>Placement</td>
<td>Unchanged</td>
<td>Stable</td>
</tr>
<tr>
<td>Ss</td>
<td>State sharing</td>
<td>Unchanged</td>
<td>Stable</td>
</tr>
<tr>
<td>4</td>
<td>Operator separation</td>
<td>Changed</td>
<td>Stable</td>
</tr>
<tr>
<td>5</td>
<td>Operator reordering</td>
<td>Changed</td>
<td>Stable</td>
</tr>
<tr>
<td>6</td>
<td>Redundancy elimination</td>
<td>Changed</td>
<td>Stable</td>
</tr>
<tr>
<td>7</td>
<td>Load balancing</td>
<td>Unchanged</td>
<td>Stable</td>
</tr>
<tr>
<td>8</td>
<td>Algorithm selection</td>
<td>Unchanged</td>
<td>Unstable</td>
</tr>
<tr>
<td>9</td>
<td>Load shedding</td>
<td>Unchanged</td>
<td>Unstable</td>
</tr>
<tr>
<td>10</td>
<td>Fission</td>
<td>Changed</td>
<td>Stable</td>
</tr>
</tbody>
</table>

→ exact (not approximate) for stable semantics
Constant-Time Aggregation

Two-Stacks

time amortized \(O(1)\), space \(2N\)
Emulate 2 Stacks with 1 Queue

Two-Stacks
time amortized $O(1)$, space $2N$

(val agg)

(bottom of both stacks, boundary between stacks)
De-Amortize Flip Operation

Two-Stacks

time amortized $O(1)$, space $2N$

DABA

time worst-case $O(1)$, space $2N$

➤ DEBS 2017 paper
Go Lite on Space

Two-Stacks

time amortized $O(1)$, space $2N$

Two-Stacks Lite

time amortized $O(1)$, space $N+1$

DABA

time worst-case $O(1)$, space $2N$
# Both De-Amortized and Lite

## Two-Stacks
- **Time**: amortized O(1), space 2N

### Diagram:
- Val: c, d, e, f, g
- Agg: h, i, j, k, l, m, n

## Two-Stacks Lite
- **Time**: amortized O(1), space N+1

### Diagram:
- Queue: cd..g, defg, efg, fg, g
- Val: h, i, j, k, l, m, n

## DABA
- **Time**: worst-case O(1), space 2N

### Diagram:
- Val: c, d, e, f, g, h, i, j, k, l, m, n
- Agg: cd..l, de..l, ef..l, fg, g, h

## DABA Lite
- **Time**: worst-case O(1), space N+2

### Diagram:
- Queue: cd..l, de..l, ef..l, fg, g, h
- Val: h, i, j, k, l, m, n
- Agg: cd..l, de..l, ef..l, fg, g, h

> VLDB Journal 2021 paper
Open-Source Repository

github.com/ibm/sliding-window-aggregators
To Learn More

• Tutorial: Stream Processing Optimizations. Scott Schneider, Buğra Gedik, Martin Hirzel. DEBS 2013.


• Spreadsheets for Stream Processing with Unbounded Windows and Partitions. Martin Hirzel, Rodric Rabbah, Philippe Suter, Olivier Tardieu, Mandana Vaziri. DEBS 2016.


Thank You, Collaborators!

Thank You, DEBS Community!

• Tutorial: Stream Processing Optimizations. Scott Schneider, Buğra Gedik, Martin Hirzel. DEBS 2013.
• Spreadsheets for Stream Processing with Unbounded Windows and Partitions. Martin Hirzel, Rodric Rabbah, Philippe Suter, Olivier Tardieu, Mandana Vaziri. DEBS 2016.
Take-Home Messages

• Parallelism itself is a not a goal, but just one of several possible optimizations
• Partitioning is key to stable parallelism
• Aggregation is foundational to most event and stream programming models
Backup
Data sets ...

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Key</th>
<th># Keys</th>
<th># Events</th>
<th>Logical time</th>
</tr>
</thead>
<tbody>
<tr>
<td>finance</td>
<td>Trade</td>
<td>symbol</td>
<td>390</td>
<td>10,000,000</td>
<td>2 h 01 min</td>
</tr>
<tr>
<td>twitter</td>
<td>Tweet</td>
<td>author</td>
<td>6,142</td>
<td>200,000</td>
<td>36 h 40 min</td>
</tr>
</tbody>
</table>

```haskell
type Trade = tuple<
    timestamp ts, rstring symbol,
    uint32 price, uint32 size, uint32 seqNum>

type Tweet = tuple<
    uint64 id, timestamp ts, rstring author,
    rstring content>
```

... and benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Pattern</th>
<th>Description</th>
<th>Selectivity</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>finance0</td>
<td>largeSize priceRise+ priceDrop</td>
<td>Large trade followed by peak</td>
<td>1.71 %</td>
<td>MatchRegex only</td>
</tr>
<tr>
<td>finance1</td>
<td>. rise+ drop+ rise+ drop* deep</td>
<td>M-shape (double top)</td>
<td>0.31 %</td>
<td>MatchRegex only</td>
</tr>
<tr>
<td>finance2</td>
<td>. rise* riseEnd flat* flatEnd</td>
<td>Rise then flat with time window</td>
<td>2.72 %</td>
<td>MatchRegex only</td>
</tr>
<tr>
<td>finance3</td>
<td>divergence</td>
<td>Price substantially above VWAP</td>
<td>0.03 %</td>
<td>VWAP → MatchRegex</td>
</tr>
<tr>
<td>finance4</td>
<td>hi gap* lo</td>
<td>Max of hi smaller than min of lo</td>
<td>5.15 %</td>
<td>MinMax → MatchRegex</td>
</tr>
<tr>
<td>finance5</td>
<td>. notTooLong* largeIncrease</td>
<td>Large increase with time window</td>
<td>0.04 %</td>
<td>MatchRegex only</td>
</tr>
<tr>
<td>twitter0</td>
<td>(None)</td>
<td>Parse tweet only, no matching</td>
<td>100.00 %</td>
<td>ParseTweet only</td>
</tr>
<tr>
<td>twitter1</td>
<td>. sameTags+ sameTags5th</td>
<td>Five tweets with identical tags</td>
<td>14.07 %</td>
<td>ParseTweet → MatchRegex</td>
</tr>
<tr>
<td>twitter2</td>
<td>.+ disjointTags</td>
<td>Different first vs. last tags</td>
<td>2.15 %</td>
<td>ParseTweet → MatchRegex</td>
</tr>
</tbody>
</table>