From a Calculus to an Execution Environment for Stream Processing

Robert Soulé  Martin Hirzel  Buğra Gedik  Robert Grimm
Cornell University  IBM Research  Bilkent University  New York University

DEBS 2012
… to an Execution Environment

Source languages

- CQL (StreamSQL)
- Sawzall (MapReduce)
- StreamIt (SDF)

Optimizations

- Fusion (merge ops)
- Fission (replicate ops)
- Placement (assign hosts)

River (execution environment)

System S (platform)

Benefits of execution environment:
- Language portability
- Optimization reuse
From a Calculus …

• Calculus = formal language + semantics
  – Stream calculus, Soulé et al. [ESOP’10]

• Graph language:
  – Stream operators with functions \( F \)
  – Queues \( Q \)
  – Variables \( V \)

• Semantics:
  – Small-step
  – Operational
  – Sequence of “operator firings”

\[
F \vdash <Q_1, V_1> \\
\rightarrow_b <Q_2, V_2> \\
\rightarrow_b^* \ldots
\]
Benefits of Calculus: Translation Correctness Proofs

\[ F_s, P_s, I_s \xrightarrow{s} O_s \]

\[ F_b, P_b, I_b \xrightarrow{b} O_b \]

Input  Execute  Output

Translate

\[ \text{Translate} \]

\[ \text{Input} \]

\[ \text{Execute} \]

\[ \text{Output} \]
## From Abstractions to the Real World

<table>
<thead>
<tr>
<th><strong>Brooklet calculus</strong></th>
<th><strong>River execution environment</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence of atomic steps</td>
<td>Operators execute concurrently</td>
</tr>
<tr>
<td>Pure functions, state threaded through invocations</td>
<td>Stateful functions, protected with automatic locking</td>
</tr>
<tr>
<td>Non-deterministic execution</td>
<td>Restricted execution: bounded queues and back-pressure</td>
</tr>
<tr>
<td>Opaque functions</td>
<td>Function implementations</td>
</tr>
<tr>
<td>No physical platform, independent from runtime</td>
<td>Abstract representation of platform, e.g. placement</td>
</tr>
<tr>
<td>Finite execution</td>
<td>Indefinite execution</td>
</tr>
</tbody>
</table>
Concurrent Execution

Case 1: No Shared State

- Brooklet operators fire one at a time
- River operators fire concurrently
- For both, data must be available
Concurrent Execution
Case 2: With Shared State

- Locks form equivalence classes over shared variables
- Every shared variable is protected by one lock
- Shared variables in the same class protected by same lock
- Locks acquired/released in standard order

Minimal locking
Naïve approach:
block when output queue is full

Deadlock!

\[ o_2 \text{ waits b/c output Q is full} \]
\[ o_3 \text{ waits b/c } o_2 \text{ locked w} \]
Restricted Execution
Safe Back-Pressure

- Our approach: only block on output queue when not holding locks on variables
Applications of an Execution Environment

• Easier to develop source languages
  – Implementation language
  – Language modules
  – Operator templates

• Possible to reuse optimizations
  – Annotations provide additional information between source and intermediate language
logs : \{\text{origin} : \text{string}; \text{target} : \text{string}\} \text{ stream};

\text{hits} : \{\text{origin} : \text{string}; \text{count} : \text{int}\} \text{ stream} =
\text{select} \text{istream(}\text{origin}, \text{count(}\text{origin})\text{)}
\text{ from logs[range 300]}
\text{ where origin} \neq \text{target}

\text{Expose operators, communication, and state}

Pre-existing operator templates

Bag.filter (fun x -> #expr)

Bag.filter (fun x -> origin \neq target)
Translation Support: Pluggable Compiler Modules

```
select istream(*)
from quotes[now], history
where quotes.ask<=history.low
and quotes.ticker=history.ticker
```

\[ \text{CQL} = \text{SQL} + \text{Streaming} + \text{Expressions} \]
Optimization Support: Extensible Annotations

Source language

River (execution environment)

System S (platform)

Optimizer

Needs to know:
- Safety
- Profitability

Establishes by construction, e.g., Sawzall reducers commute

Establishes, e.g., available resources

13
# Optimization Support: Current Annotations

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Fuse(ID)</td>
<td>Fuse operators with same ID in the same process</td>
<td>Fusion</td>
</tr>
<tr>
<td>@Parallel()</td>
<td>Perform fission on an operator</td>
<td>Fission</td>
</tr>
<tr>
<td>@Commutative()</td>
<td>An operator’s function is commutative</td>
<td>Fission</td>
</tr>
<tr>
<td>@Keys($k_1,\ldots,k_n$)</td>
<td>An operator’s state is partitionable by fields $k_1,\ldots,k_n$</td>
<td>Fission</td>
</tr>
<tr>
<td>@Group(ID)</td>
<td>Place operators with same ID on the same machine</td>
<td>Placement</td>
</tr>
</tbody>
</table>
Evaluation

• Four benchmark applications
  – CQL linear road
  – StreamIt FM radio
  – Sawzall web log analyzer (batch)
  – CQL web log analyzer (continuous)

• Three optimizations
  – Placement
  – Fission
  – Fusion
Distributed Linear Road
(simplified version from Arasu/Babu/Widom [VLDBJ’06])

First distributed CQL implementation
CQL: Placement, Fusion, Fission

- Placement + Fusion → 4x speedup on 4 machines
- Fission → 2x speedup on 16 machines
- Insufficient work per operator
StreamIt: Placement

- Optimization reuse $\rightarrow$ 1.8x speedup on 4 machines
Sawzall (MapReduce on River)  
Fission + Fusion

- Same fission optimizer for Sawzall as for CQL
- 8.92x speedup on 16 machines, 14.80x on 64 cores
- With fusion, 50.32x on 64 cores
Related Work

Stream processing

CQL
Arasu et al. [VLDB J.’06]

SVM
Labonte et al. [PACT’04]

Execution environment

P-Code
Nelson [CC’79]

Translators from languages to IL

This paper
Conclusions

• River, execution environment for streaming
• Semantics specified by formal calculus
  – Brooklet, Soulé et al. [ESOP’10]
• 3 source languages, 3 optimizations
  – First distributed CQL
  – Language compiler module reuse
  – Optimization enabled by annotations
• Encourages innovation in stream processing
• http://www.cs.nyu.edu/brooklet/