Auto-Parallelizing Stateful Distributed Streaming Applications

Scott Schneider*, Martin Hirzel*, Bugra Gedik† and Kun-Lung Wu*

*IBM Research †Bilkent University
Big Data
Programming Model

• Streams applications
  – Described as data-flow graphs
    • An instance of a flow graph is a job in the system
  – Flow graphs consist of
    • Tuples: structured data item
    • Operators: Reusable stream analytics
    • Streams: Series of tuples with a given schema
Streaming Programming Models

**Synchronous**

- Static selectivity
  - e.g., $1 : 3$
    
    ```python
    for i in range(3):
        result = f(i)
        submit(result)
    ```
  - In general, $m : n$ where $m$ and $n$ are statically known
- Always has static schedule

**Asynchronous**

- Dynamic selectivity
  - e.g., $1 : [0,1]$
    
    ```python
    if input.value > 5:
        submit(result)
    ```
  - In general, $1 : *$
- In general, schedules cannot be static
InfoSphere Streams Runtime

SPL compiler

Streams Runtime
(Job management, Security, Continuous Resource Management)

Source PE PE PE PE PE PE Sink
x86 host x86 host x86 host x86 host x86 host
composite Main {
  type
    Entry = int32 uid, rstring server,
             rstring msg;
    Sum = uint32 uid, int32 total;
  graph
    stream<Entry> Msgs = ParSource() {
      param servers: "logs.*.com";
      partitionBy: server;
    }
    stream<Sum> Sums = Aggregate(Msgs) {
      window Msgs: tumbling, time(5),
                     partitioned;
      param partitionBy: uid;
    }
    stream<Sum> Suspects = Filter(Sums) {
      param filter: total > 100;
    }
    () as Sink = FileSink(Suspects) {
      param file: "suspects.csv";
    }
}
Overview

Compiler:
- Apply parallel transformations
- Pick routing mechanism (e.g., hash by key)
- Pick ordering mechanism (e.g., seq. numbers)

Runtime:
- Replicate segment into channels
- Add split/merge/shuffle as needed
- Enforce ordering

Stream graph description
## Transformations & Safety Conditions

<table>
<thead>
<tr>
<th>Parallelize non-source/sink</th>
<th>Parallelize sources and sinks</th>
<th>Combine parallel regions</th>
<th>Rotate merge and split</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="parallelize-non-source-sink" alt="Diagram" /></td>
<td><img src="parallelize-sources-sinks" alt="Diagram" /></td>
<td><img src="combine-parallel-regions" alt="Diagram" /></td>
<td><img src="rotate-merge-split" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- **Parallelize non-source/sink**
  - stateless or partitioned state
  - selectivity ≤ 1
  - simple chain

- **Parallelize sources and sinks**
  - stateless or partitioned state

- **Combine parallel regions**
  - stateless or
  - compatible keys
  - forwarding

- **Rotate merge and split**
  - incompatible keys
Select Parallel Segments

- Can't parallelize
  - Operators with >1 fan-in or fan-out
  - Punctuation dependency later on
- Can't add operator to parallel segment if
  - Another operator in segment has co-location constraint
  - Keys don't match
Compiler to Runtime

- Compiler
  - Graph + unexpanded parallel regions
  - Fully expanded graph
    - Runtime graph fragment
      - PE
    - Runtime graph fragment
      - PE
    - Runtime graph fragment
      - PE
### Runtime

<table>
<thead>
<tr>
<th></th>
<th>Selectivity = 1</th>
<th>Selectivity ≤ 1</th>
<th>Selectivity Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>no state</td>
<td></td>
<td></td>
<td><em>don't parallelize</em></td>
</tr>
<tr>
<td>partitioned state</td>
<td></td>
<td></td>
<td><em>don't parallelize</em></td>
</tr>
<tr>
<td>unknown state</td>
<td><em>don't parallelize</em></td>
<td><em>don't parallelize</em></td>
<td><em>don't parallelize</em></td>
</tr>
</tbody>
</table>

**Operators in parallel segments:**
- Forward seqno & pulse

**Split:**
- Insert seqno & pulse
- Routing

**Merge:**
- Apply ordering policy
- Remove seqno (if there) and drop pulse (if there)
Merger Ordering

Round-Robin

Sequence Numbers

Sequence Numbers and Pulses
Scalability

The graph shows the speedup compared to the sequential case as a function of the number of multiplications per tuple. Two lines are plotted:
- Blue line with markers: # channels = 50 (SplitShuffleMerge)
- Green line with markers: # channels = 100 (SplitMergeStateful)

The diagram includes two different configurations:
- SplitMergeStateful
- SplitShuffleMerge

The SplitMergeStateful configuration includes nodes labeled as `Src`, `Aggr`, and `Sink`. The SplitShuffleMerge configuration also includes nodes labeled as `Src`, `Aggr`, and `Sink`.
Application Kernels

![Graph showing speedup vs. 1 channel for different application kernels.]

- Network monitoring
- PageRank
- Twitter NLP
- Twitter CEP
- Finance

Speedup vs. 1 channel

Number of parallel channels

0 2 4 8 16 32

Network monitoring

ParSrc Aggr Filter<1 Aggr Filter<1 ParSink

Twitter NLP

Parse Match

Twitter CEP

Init While Sink

Chop MulAdd Add

PageRank

Vwap Project

Finance
Questions?
Backups
Compiler Changes

- Operator model
- Analyze SPL code
- Infer partition colocation
- Select parallel segments
- Fuse
- Segments
- Partitions
- Write ADL
Transport Changes

• TCPSender
  – Added the ability to send to a subset of connections on an output port

• Handshake
  – Modified to include sender identities

• TCPReciever
  – Added support for identifying which connection has delivered a tuple
Overhead
Scalability

Stateless

Stateful

Speedup compared to the sequential case

Per tuple processing cost (# of multiplications)
Scalability