Flumina: Correct Distribution of Stateful Streaming Computations

Konstantinos Kallas*; Filip Nikšić*; Caleb Stanford*; Rajeev Alur

UCSD -- February 2020
Motivation
Stream Processing

Compared to batch processing:

- Low response times
- Can support larger datasets
- More natural for some applications
Many applications

- Video Streaming
- Medical Devices
- Drones
- More Video Streaming
Solutions for Distributed Stream Processing

- Dataflow or SQL
- Great Performance
- High-level
- Support many computations
  - map
  - filter
  - keyBy
What about this?

Looks distributable

- Reduced network load
- Lower latency
- Privacy
Main idea:
View streams as partial orders
Conceptual model

- Specification
- Target Topology
- Optimizer
- Distribution Plan
- Distributor
- Implementation
Example

\[
\begin{align*}
\text{state} & := \text{int} // \text{max temp so far} \\
\text{temp}_e & := \langle \text{temp, int} \rangle \\
\text{update}_\text{temp} :: \text{temp}_e \to \text{state} \to \text{state} \\
\text{update}_\text{temp} <\text{temp, Val}> \text{OldMax} & := \\
& \quad \text{return max(OldMax, Val)} \\
\text{max}_e & := \langle \text{max_temp} \rangle \\
\text{update}_\text{max} :: \text{max}_e \to \text{state} \to \text{state} \\
\text{update}_\text{max} <\text{max_temp}> \text{OldMax} & := \\
& \quad \text{output}(<\text{day_max_temp}, \text{OldMax}); \\
& \quad \text{return 0}
\end{align*}
\]
Dependency relation

< max_temp > events depend on < temp, V > events

< max_temp > events depend on < max_temp > events

Independent events can be processed concurrently
Distribution model

Worker 1
Handles: < temp, ... >,
< max_temp >

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >

Communication

“more” dependent

“less” dependent

Leaves process events independently
Processing dependent events

Worker 1
Handles: < temp, ... >,
< max_temp >

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >
Processing dependent events

Worker 1
Handles:
< temp, ... >,
< max_temp >

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >
Processing dependent events

Worker 1
Handles:
< temp, ... >,
< max_temp >

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >
Fork - Join

// State
state := int // max temp so far

// Events
temp_e := <temp, int>
max_e := <max_temp>

update_temp :: temp_e -> state -> state
update_temp <temp, Val> OldMax :=
    return max(OldMax, Val)

update_max :: max_e -> state -> state
update_max <max_temp> OldMax :=
    output(<day_max_temp, OldMax>);
    return 0

fork :: state -> (state * state)
fork Max :=
    return (Max, Max)

join :: (state * state) -> state
join Max1 Max2 :=
    return max(Max1, Max2)
Fork - Join

Worker 1
Handles: < temp, ... >,
< max_temp >

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >

Worker 6
Handles: < temp, ... >

Worker 7
Handles: < temp, ... >
Correctness

Sequential

\[ u_t \rightarrow u_t \rightarrow u_t \rightarrow u_t \rightarrow u_m \]

Distributed

\[ u_t \rightarrow f \rightarrow u_t \rightarrow j \rightarrow u_m \]

\[ u_t \rightarrow f \rightarrow u_t \rightarrow j \rightarrow u_m \]

\[ u_m : \text{update\_max} \]
\[ u_t : \text{update\_temp} \]
\[ f : \text{fork} \]
\[ j : \text{join} \]
Correctness

Sequential

Distributed

u_t: update_temp
f: fork
j: join

u_m: update_max
Correctness

Sequential

Distributed

\( u_t \rightarrow u_t \rightarrow u_t \rightarrow u_t \rightarrow u_m \)

\( u_t \rightarrow f \rightarrow u_t \rightarrow j \rightarrow u_m \)

\( u_m : \text{update\_max} \)
\( u_t : \text{update\_temp} \)
\( f : \text{fork} \)
\( j : \text{join} \)
Correctness

Sequential

Distributed

u_m : update_max
u_t : update_temp
f : fork
j : join
Correctness

Sequential

1. $u_t$
2. $u_t$
3. $u_t$
4. $u_t$
5. $u_m$

Distributed

1. $u_t$
2. $f$
3. $u_t$
4. $j$
5. $u_m$

Symbols:
- $u_m$: update_max
- $u_t$: update_temp
- $f$: fork
- $j$: join
Small Recap

- Streams are partial orders
- Dependency relation encapsulates ordering requirements
- Forks-joins as distribution primitives
- Provably correct distribution
Automated Distribution
Automated Distribution

Worker 1
Handles: < temp, ... >, < max_temp >

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >

Sensor 1
Stream 1

Sensor 2
Stream 2

Sensor 3
Stream 3

Coordinator
Stream 4
Evaluation
Implementation

Public on github: https://github.com/angelhof/fluming
Setup

Single machine with 18 cores
Microbenchmarks

state := int // max temp so far
temp_e := <temp, int>

update_temp :: temp_e -> state -> state
update_temp <temp, Val> OldMax :=
    return max(OldMax, Val)

max_e := <max_temp>

update_max :: max_e -> state -> state
update_max <max_temp> OldMax :=
    output(<day_max_temp, OldMax>);
    return 0
Optimizer Comparison -- Latency

Latency Comparison

4 sensors and 1 central node
4 input temp streams (1 per sensor)

sequential: 1 centralized worker
greedy hybrid: 5 workers in central node
greedy: 5 workers (1 per node)
Flumina vs Flink -- Scaling

28 sensor nodes
Latency + Throughput
1 central node and varying sensors
1 temp input stream per sensor

2 sensor nodes
Case studies

- Distributed Outlier Detection
  - Sequential: 700 LoC
  - Distributed: + 50 LoC
  - Performance similar to original paper

- Energy Management
  - Sequential: 200 LoC
  - Distributed: + 60 LoC
  - Network Load: 350MB out of 29GB

Goal: Evaluate usability in complex applications
Conclusion
Conclusions

- Programming model
  - View input streams as partial orders
  - This enables correct distribution of more streaming computations
- Distributed computations in Flumina
  - Requires small effort to specify
  - Can be implemented efficiently in an automatic way
Future Work

- Verification of Flumina code
- Synthesis of fork-join pairs
- Online re-distribution
- High level query language
- Privacy

Public on github: https://github.com/angelhof/flumina