Flumina: Correct Distribution of Stateful Streaming Computations

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

NTUA PL Seminar -- December 2019
Motivation
Stream Processing

Compared to batch processing:

- Low response times
- Can support larger datasets
- More natural for some applications
Let there be ...
Edge computing

Big Node

Node

Node

Privacy

Client

Response time
Writing distributed code is hard :’(

Event stream

Node

code 
{
...
}

Big

Node

code 
{
...
}

Node

code 
{
...
}

Node

code 
{
...
}

Node

code 
{
...
}

Event stream

BIG

Event stream
Existing Stream Processing Solutions

Flink

Apache Storm

Apache Spark

Heron
Problems of existing solutions

- Computation has to be tuned depending on
  - performance requirements
  - underlying computational resources
  - knowledge about data (input rates, locality)
- No formal definition of correctness
Main idea:
Independent events can be processed concurrently with minimal communication
Conceptual model
Example

```
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
```
Dependency relation

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

< max_temp > events depend on < max_temp > events
Distribution 🎄 model

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

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >
Distribution model

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

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >
Distribution model

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

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >
Worker 1
Handles: < temp, ... >, < max_temp >

Worker 2
Handles: < temp, ... >

Worker 3
Handles: < temp, ... >

Worker 4
Handles: < temp, ... >

Worker 5
Handles: < temp, ... >

Distribution model
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, ... >
Fork-join

Sequential

Distributed

u_m : update_max
u_t : update_temp
f : fork
j : join
Fork-join

Sequential

Distributed

$u_t$ : update_temp

$u_m$ : update_max

$f$ : fork

$j$ : join
Fork-join

Sequential

Distributed

u_m: update_max
u_t: update_temp
f: fork
j: join
Fork-join

Sequential

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

Distributed

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

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

u_m : update_max
u_t : update_temp
f : fork
j : join
Fork-join

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. $u_m$
6. $u_t$
7. $u_t$

$u_m : update\_max$
$u_t : update\_temp$
$f : fork$
$j : join$
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
Setup

Single node 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
Microbenchmarks

![Graph showing latency over time for different strategies: sequential, greedy hybrid, and greedy.]
Flumina vs Flink -- Scaling
Case studies

- Distributed Outlier Detection
  - Sequential: 700 LoC
  - Distributed: + 50 LoC
- Energy Management
  - Sequential: 200 LoC
  - Distributed: + 60 LoC
Conclusion
Future Work

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