Contributions to Stream Processing

An Analysis of Existing Contributions

January 10, 2019

Nathan Woods

Gianforte School of Computing
Montana State University

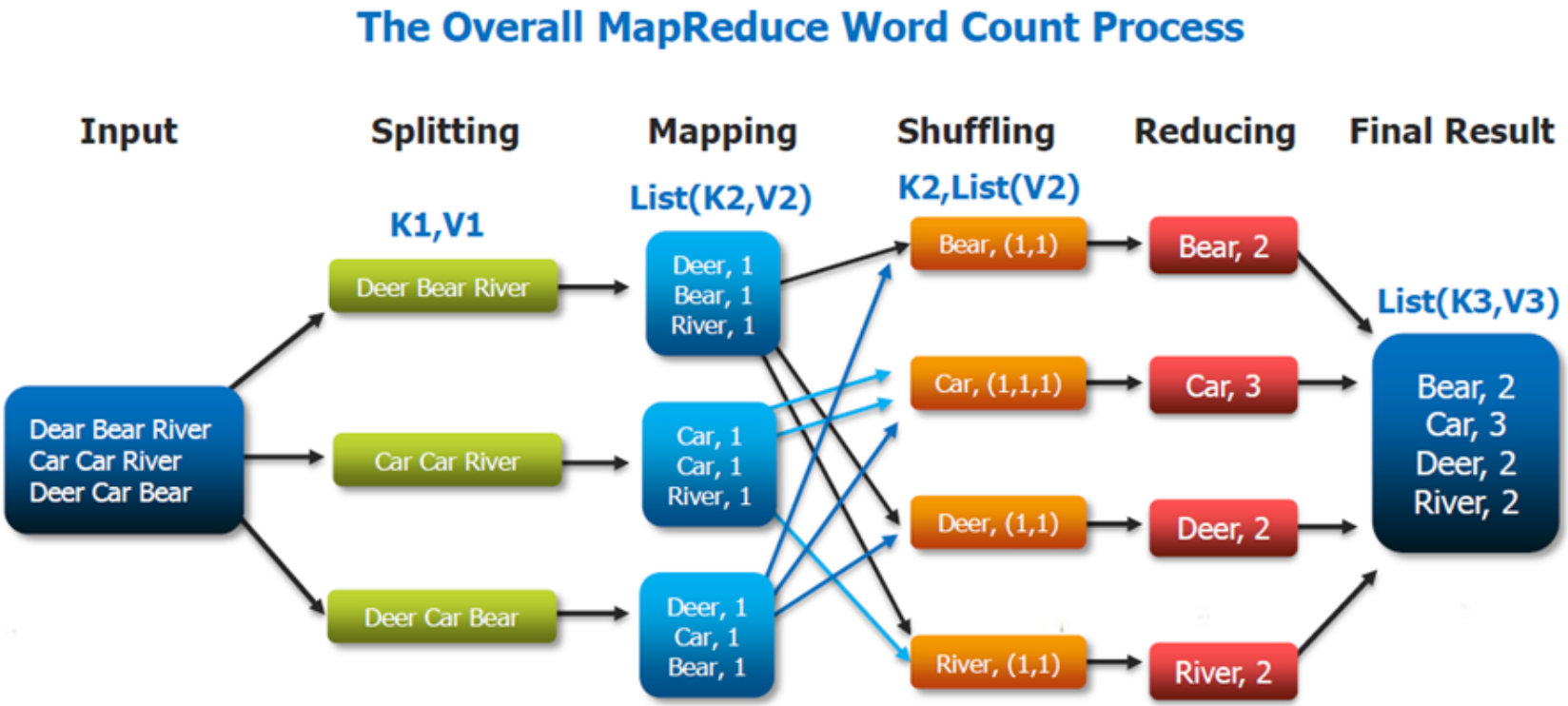
MONTANA STATE UNIVERSITY

Introduction

- 2,744,774 emails per second
- 8,262 tweets per second
- 70,634 searches per second
- 64,201 GB/s total internet traffic

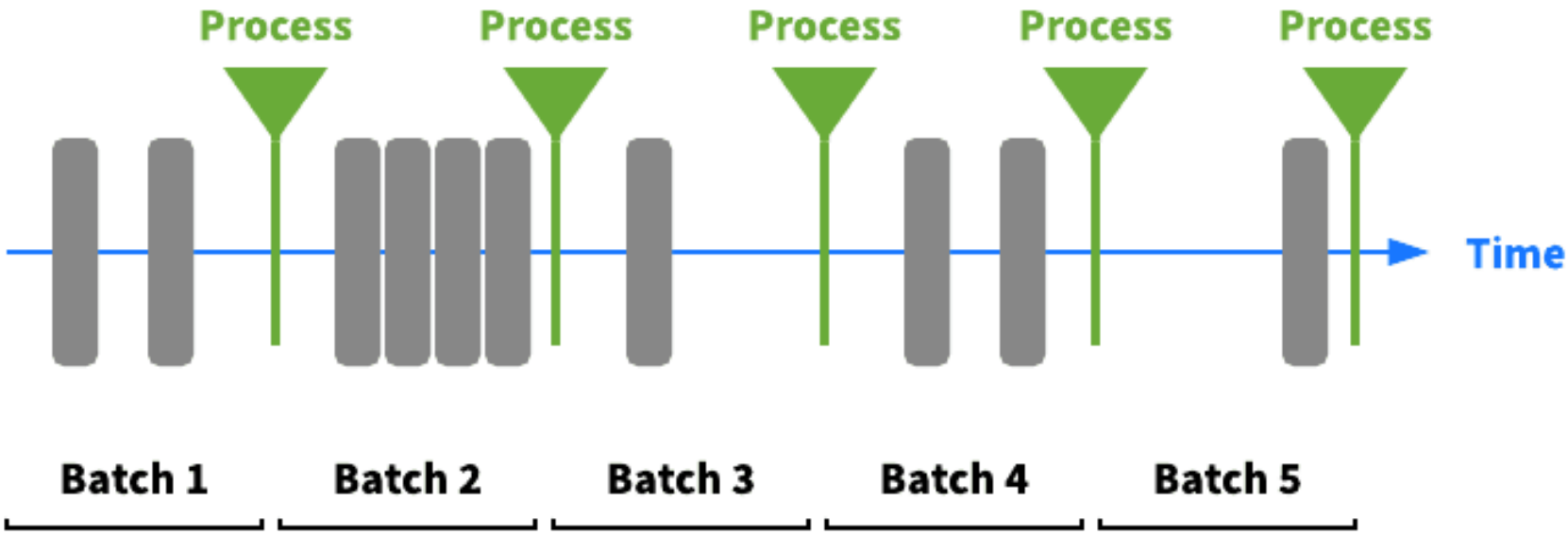
`http://www.internetlivestats.com/`

MapReduce



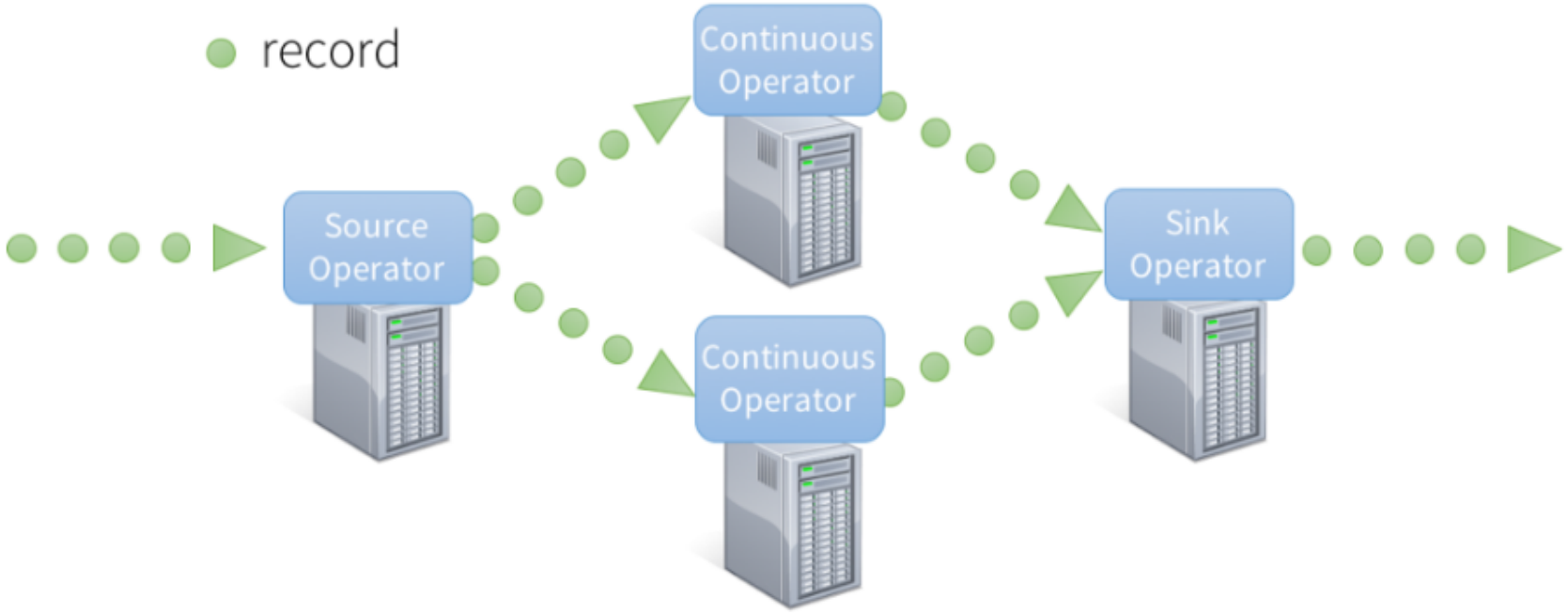
https://i.stack.imgur.com/199Q1.png

Micro-Batching



https://streaml.io/media/img/batch-processing.png

Continuous Operators



https://pangbw.files.wordpress.com/2017/04/15.png

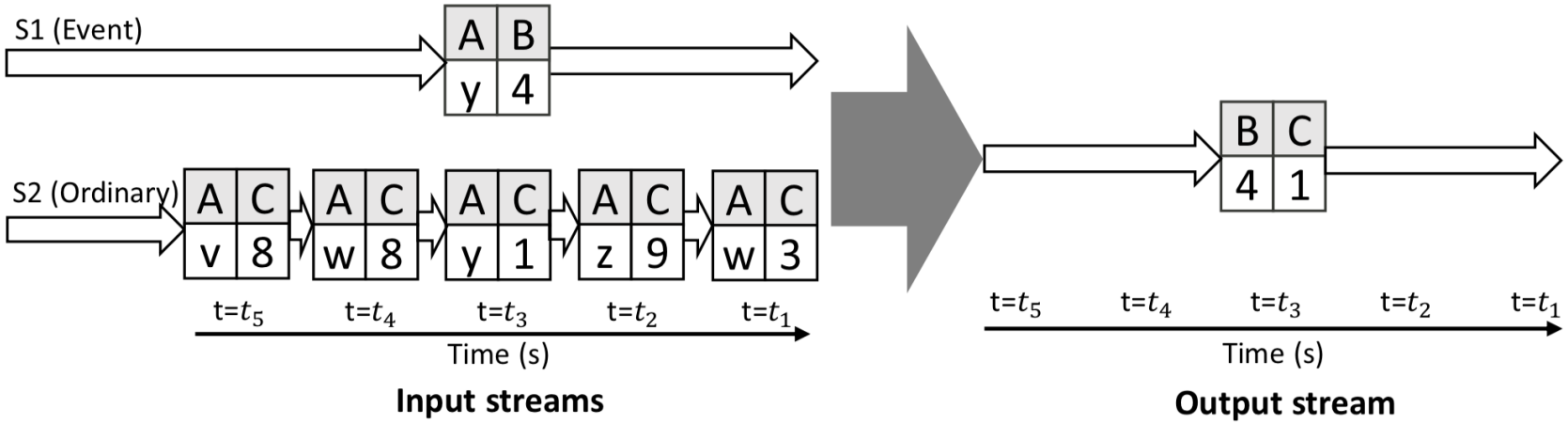
Outline

- Introduction
- Contributions
  - Smart Windows
  - Drizzle
  - Spade
  - Real-Time Analytics
  - Consistency
- Synthesis

Smart Windows

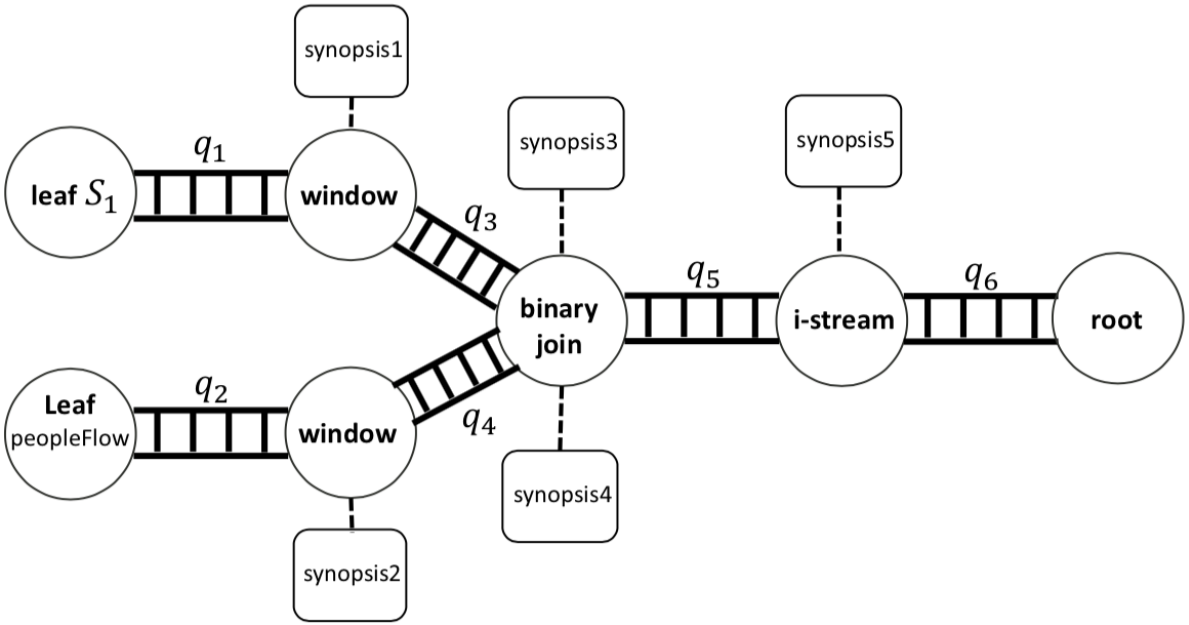
S.A.Shaikh, Y. Watanabe, Y. Wang, and H. Kitagawa
Smart Query Execution for Event-driven Stream Processing
Multimedia Big Data (BigMM), 2016 IEEE Second International Conference on. IEEE, 2016, pp. 97-104

Event Driven Queries



- **Event-driven query:** Generates results after specified events.
- **Event stream:** Above specified event streams.
- **Active:** When event stream window is not empty.
- **Active Duration:** Equivalent to the time-based window size.

Event Windows



- **Tuple-based Window:** Given integer n, return the n most recent tuples from stream S.
- **Time-based Window:** Given τ, at any time t return the tuples with timestamps between $t - \tau$ and t from stream S.
- **Incremental Computation:** Annotate events with "+" or "-" to signal events being added or removed from the window.

Smart Windows

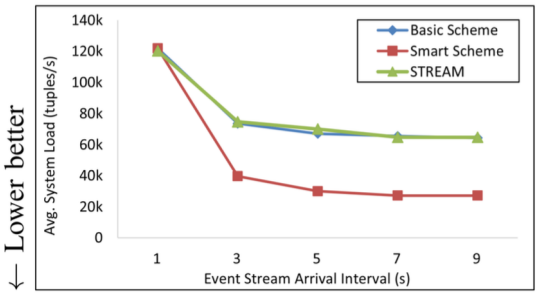
Algorithm 1 Smart Window (W_o): When new tuple arrives

1: **for each** arrival of ordinary stream tuple $e \in Q$ at timestamp t **do**
2: **if** isActive(Q) **then**
3: Insert e in the output part and send $\langle e, t, + \rangle$ downstream
4: **else**
5: Buffer e in the suspended part
6: **end if**
7: **if** # of elements $\in W_o >$ size of W_o **then**
8: Find e'; $\{e'$: oldest element in $W_o\}$
9: **if** $e' \in$ suspended part **then**
10: delete e'
11: **else**
12: delete e' and send $\langle e', t, - \rangle$ downstream
13: **end if**
14: **end if**
15: **end for**

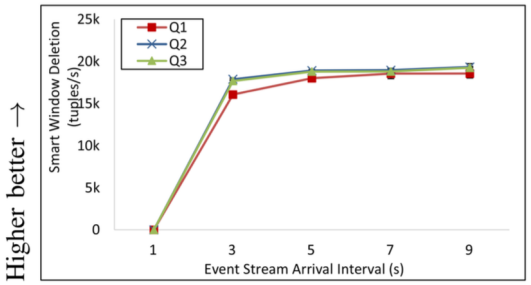
Testing

`JsSpinner`, 13k LOC C++, 2.66 GHz CPU with 4GB RAM

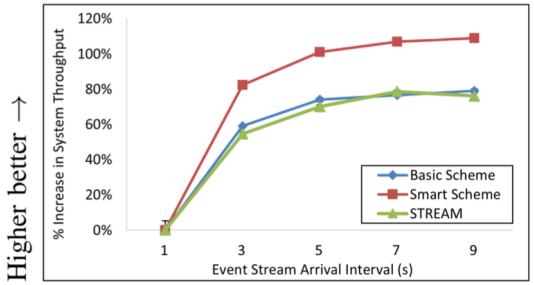
System Load: Number of tuples processed by all operators
Throughput: Number of input tuples processed



(a) System Load



(b) Event Deletions



(c) System Throughput

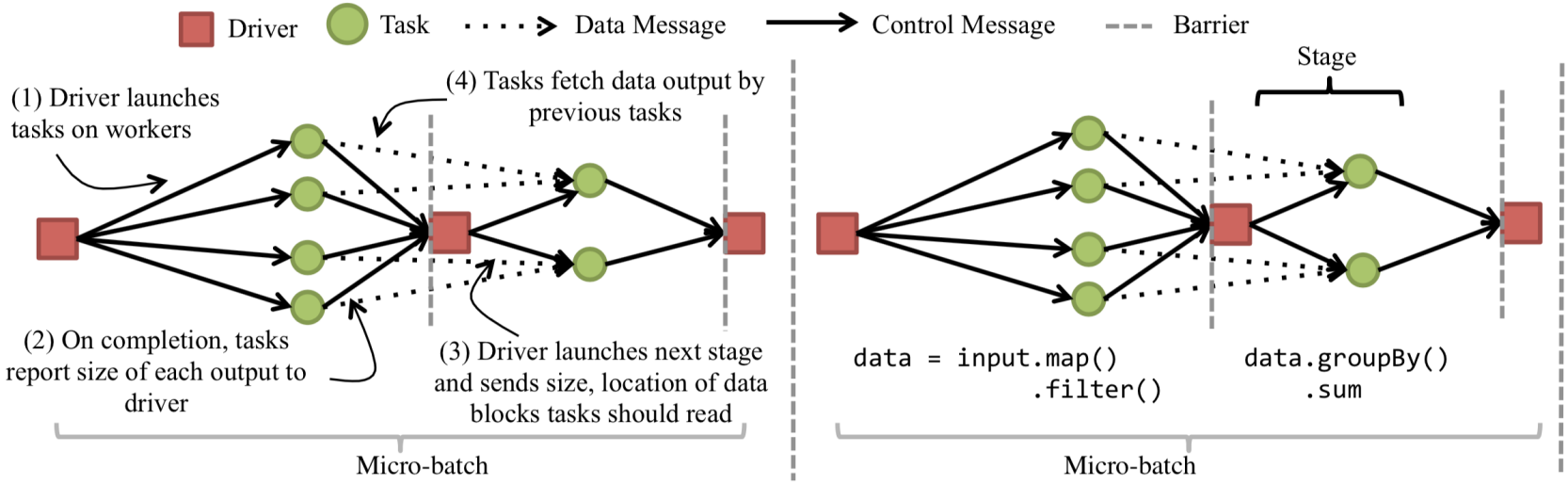
Summary

- Smart Windows reduce Processing Load
- Small scale testing
- Detect conditional queries
- Expand concept to include conditional joins

Drizzle

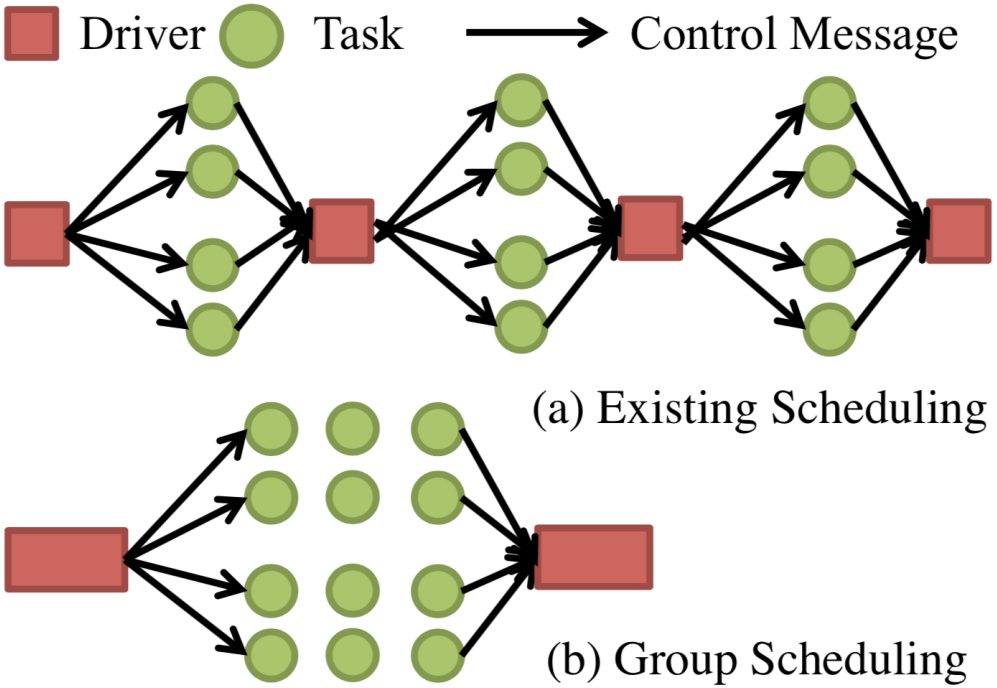
S.Venkataraman, A.Panda, K.Ousterhout, M.Armbrust, A.Ghodsi, M.R.Franklin, B.Recht, and I.Stoica
Drizzle: Fast and Adaptable Stream Processing at Scale
Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2017, pp. 374-389

Stream Processing Failures



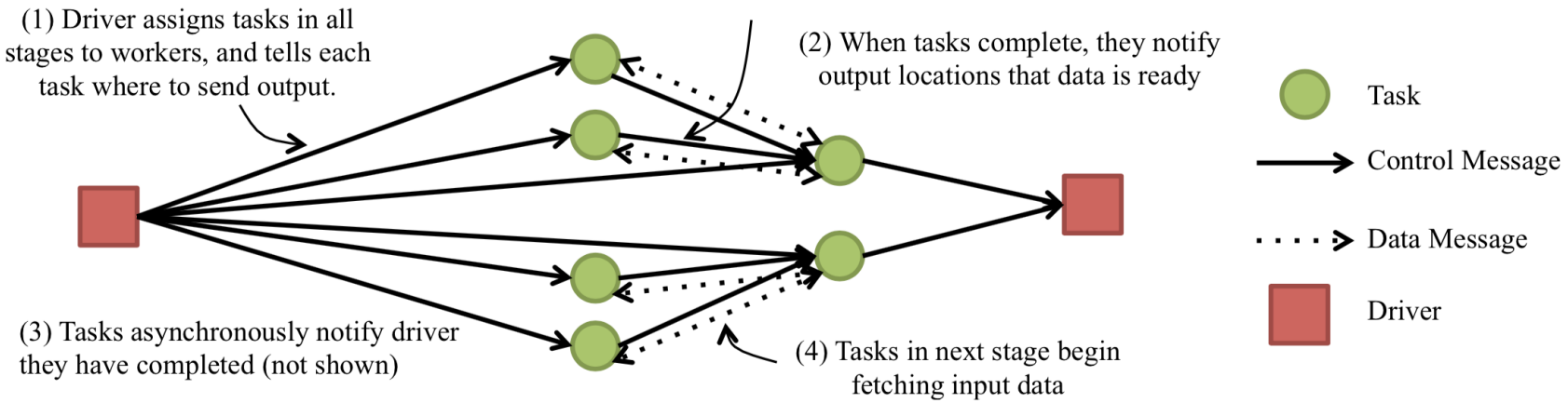
- **Failures:** Communication, Process, Hardware
- **Micro-Batch Recovery:** Task Boundaries, Parallel Retry
- **Continuous Recovery:** Replay from Checkpoints

Group Scheduling



Same scheduling instructions across multiple micro-batches.

Pre-Scheduling Shuffles



Scheduling dormant downstream tasks first.
Upstream can send data directly to consumer.

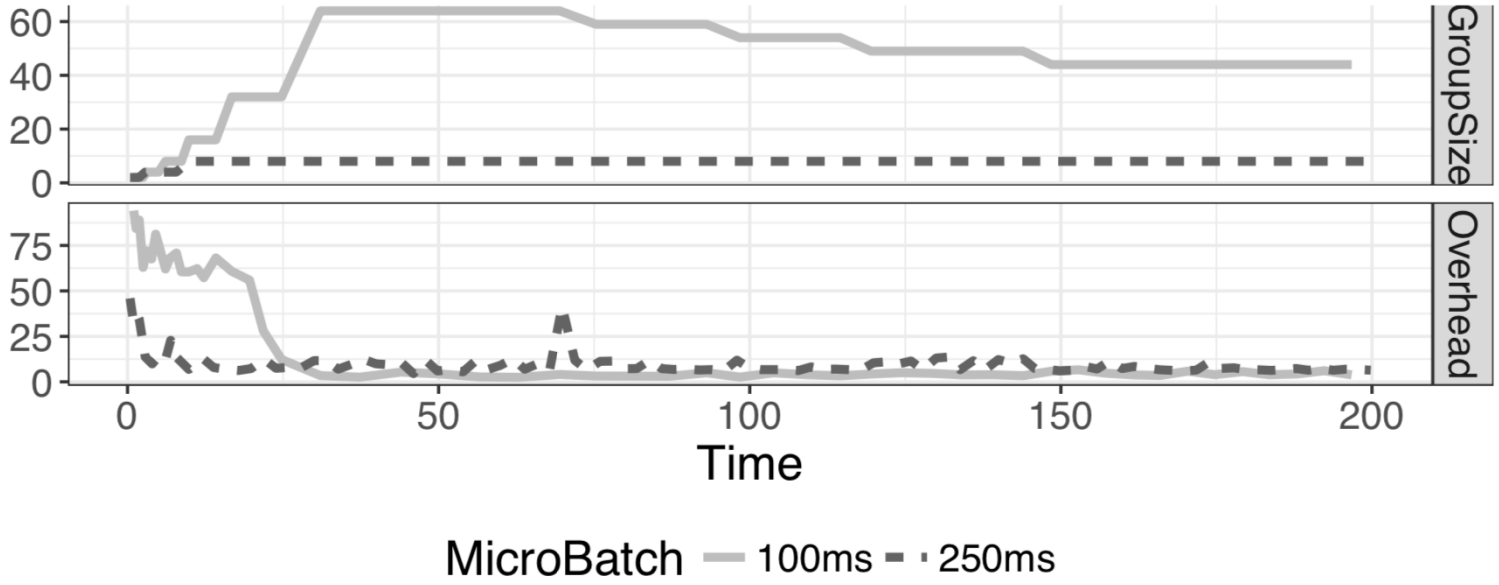
Adaptability

Fault Tolerance: Heartbeats on workers, parallel retries, Checkpoints

Elasticity: At group boundaries, queries for available resources, adjusts as required

Group Size: both properties require tuning of group or batch sizing

Tune group size based on TCP congestion control



Data-Plane Changes

Within Batch

Vectorized CPU Operations

Minimize trafic with partial merges

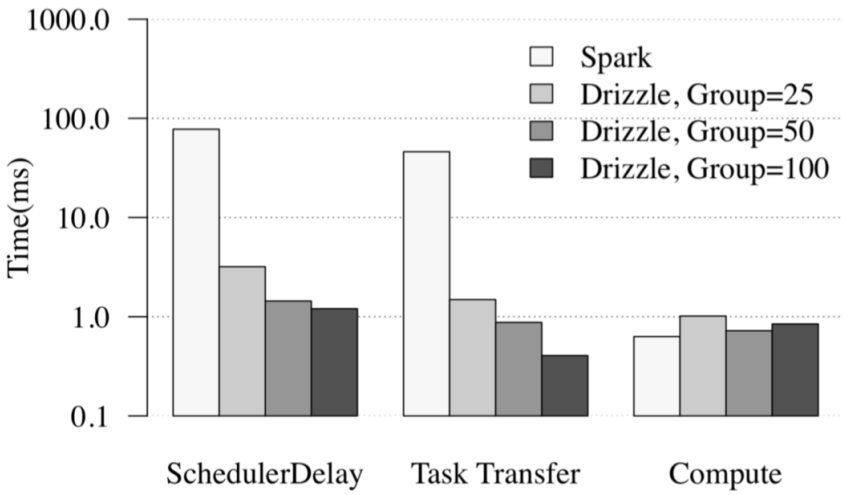
Across Batches

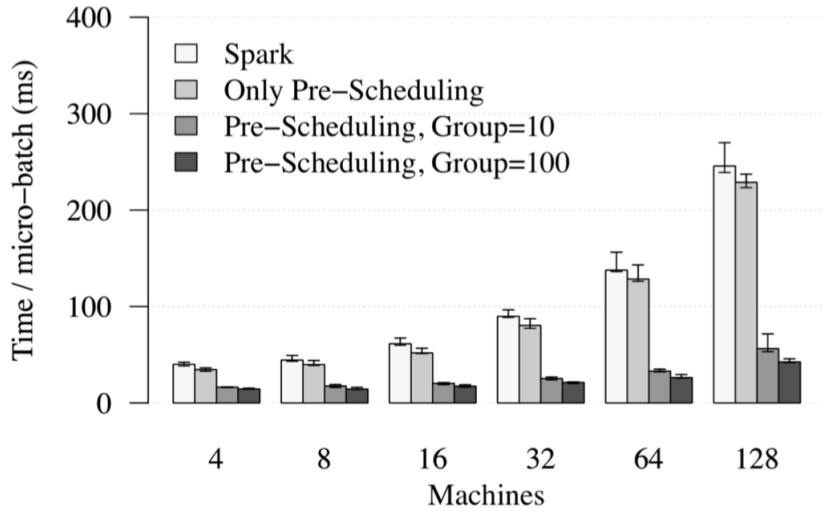
Metrics to measure query performance

Reuse results across different queries

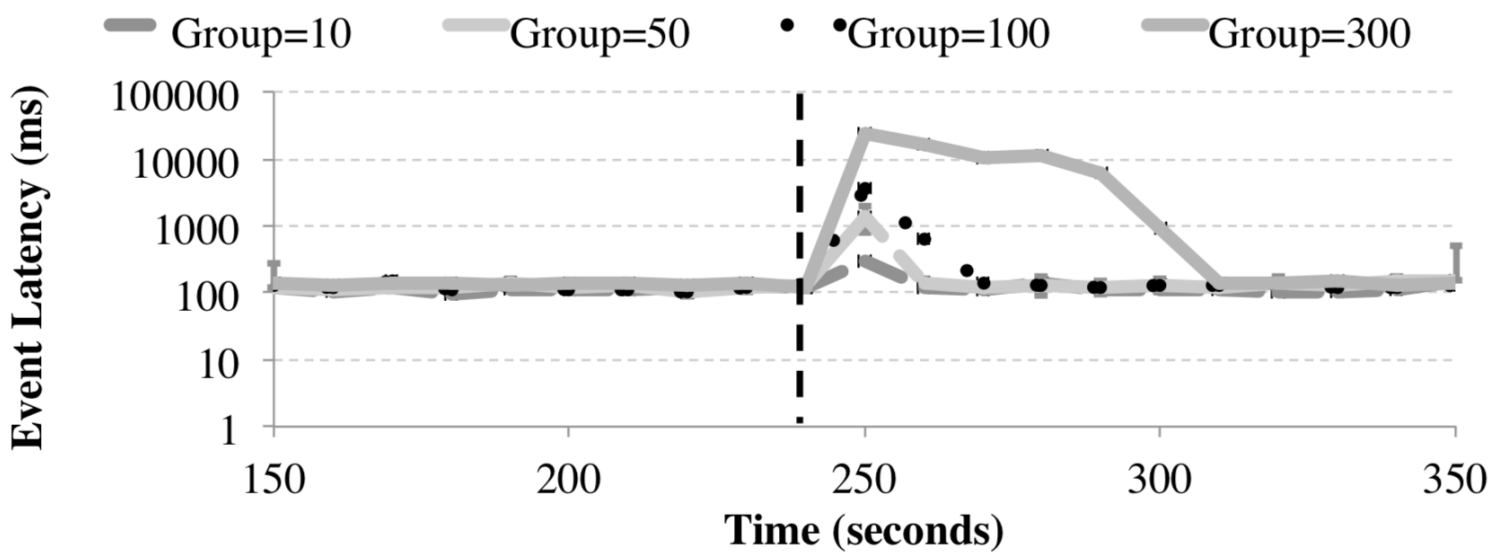
Testing

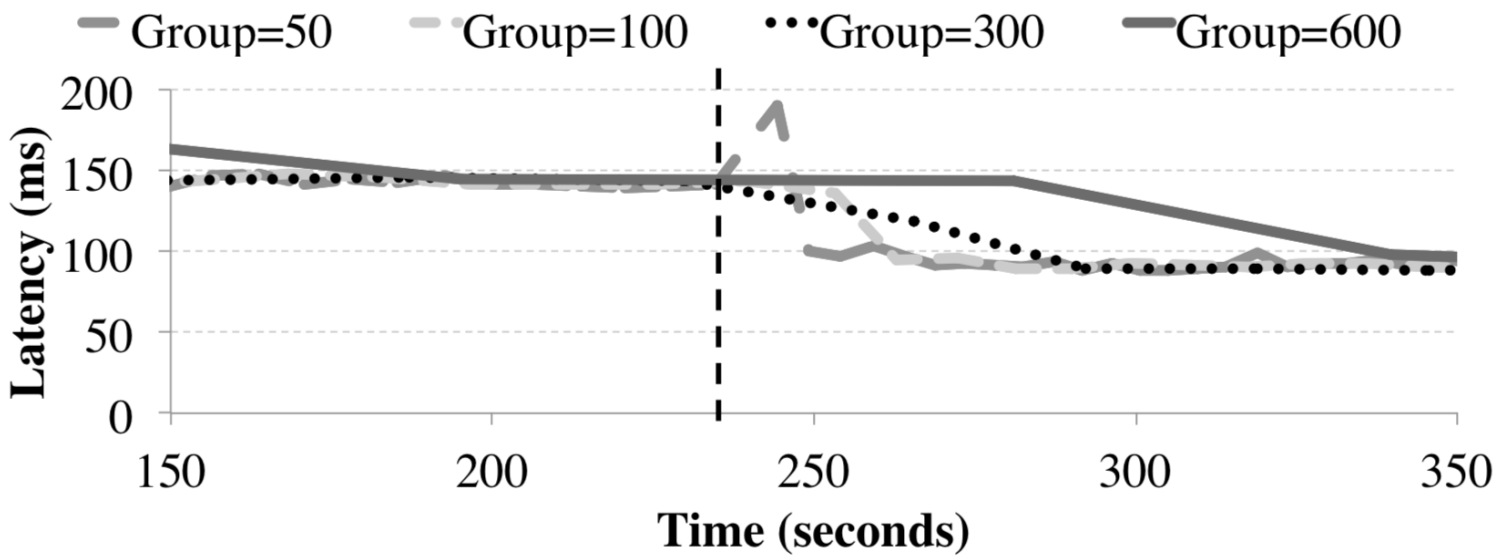
Apache Spark Extension, 4k LOC Skala, 128 `r3.xlarge`
(4 Core, 30.5GB RAM, 80GB SSD)
Tests include JVM warmup time





Testing Continued





Summary

- Low Latency Scheduling
- Data-Level Optimizations
- Included JVM warmup time
- Multiple node failures?

Spade

B.Gedik, H.Andrade, K.-L. Wu, P.S.Yu, and M.Doo
Spade: The System S Declarative Stream Processing Engine
Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008, pp. 1123-1134

Stream Processing Ecosystem

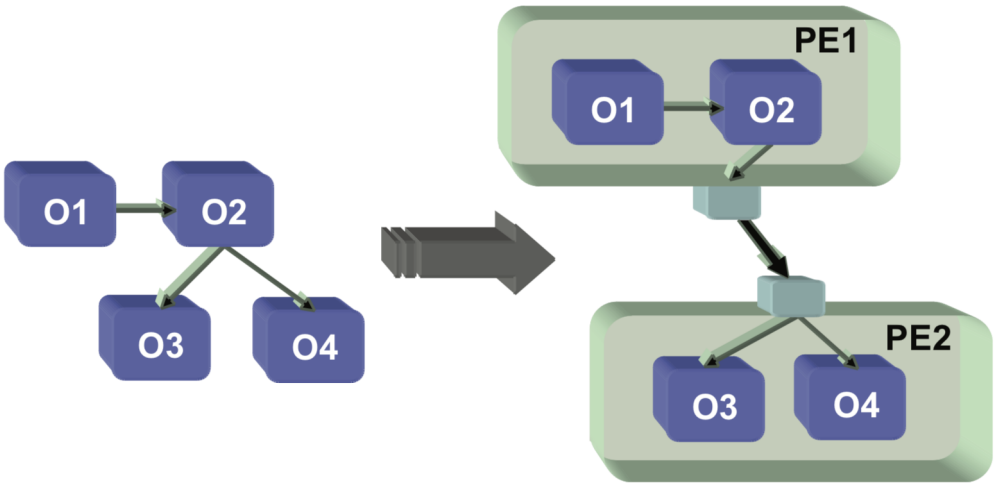
Spade

- Declarative Programming Language
- Fundamental Unit is a Stream
- Deploys Programs to System S
- Includes Compilers, Optimizers and Generators (UDOPS)
- Differences: Pre-grouping, Vectorized Operations, Edge Adapters, Windowing Schemes

System S

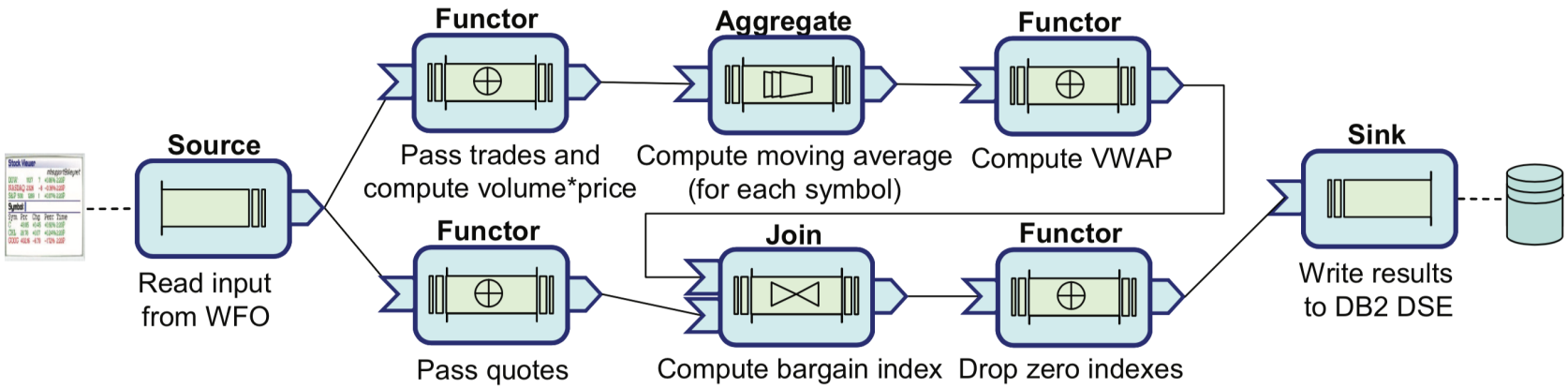
- Large-scale distributed data stream processing middleware
- Distributes jobs across a cluster (DAGs and PEs)
- Handles Reliablity, Scheduling and Placement Optimization, Distributed Job Management, Storage Services, Security, etc

Operator Fusion



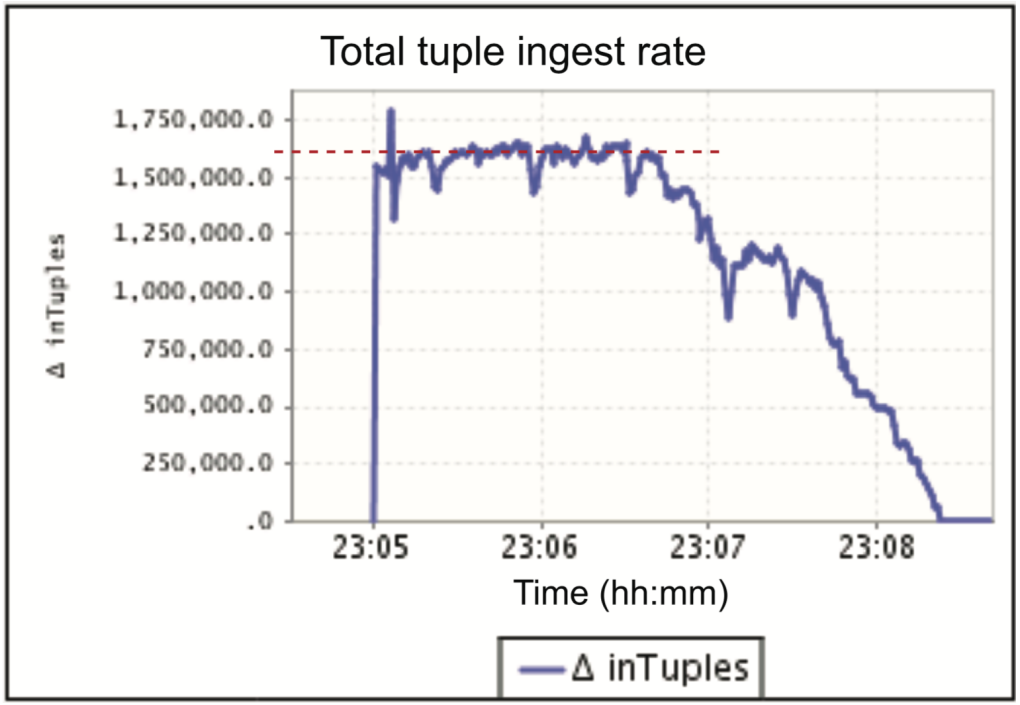
Combine Input Queues for a PE

Example Application



Testing

22 days of stock data, 20GB of data, 16 nodes in test cluster
1.6 Million tuples/second, 3.5 minutes



Summary

- Stream Processing Ecosystem (2 years before Spark)
- Small test set of 20GB
- System could report under-performing queries for review

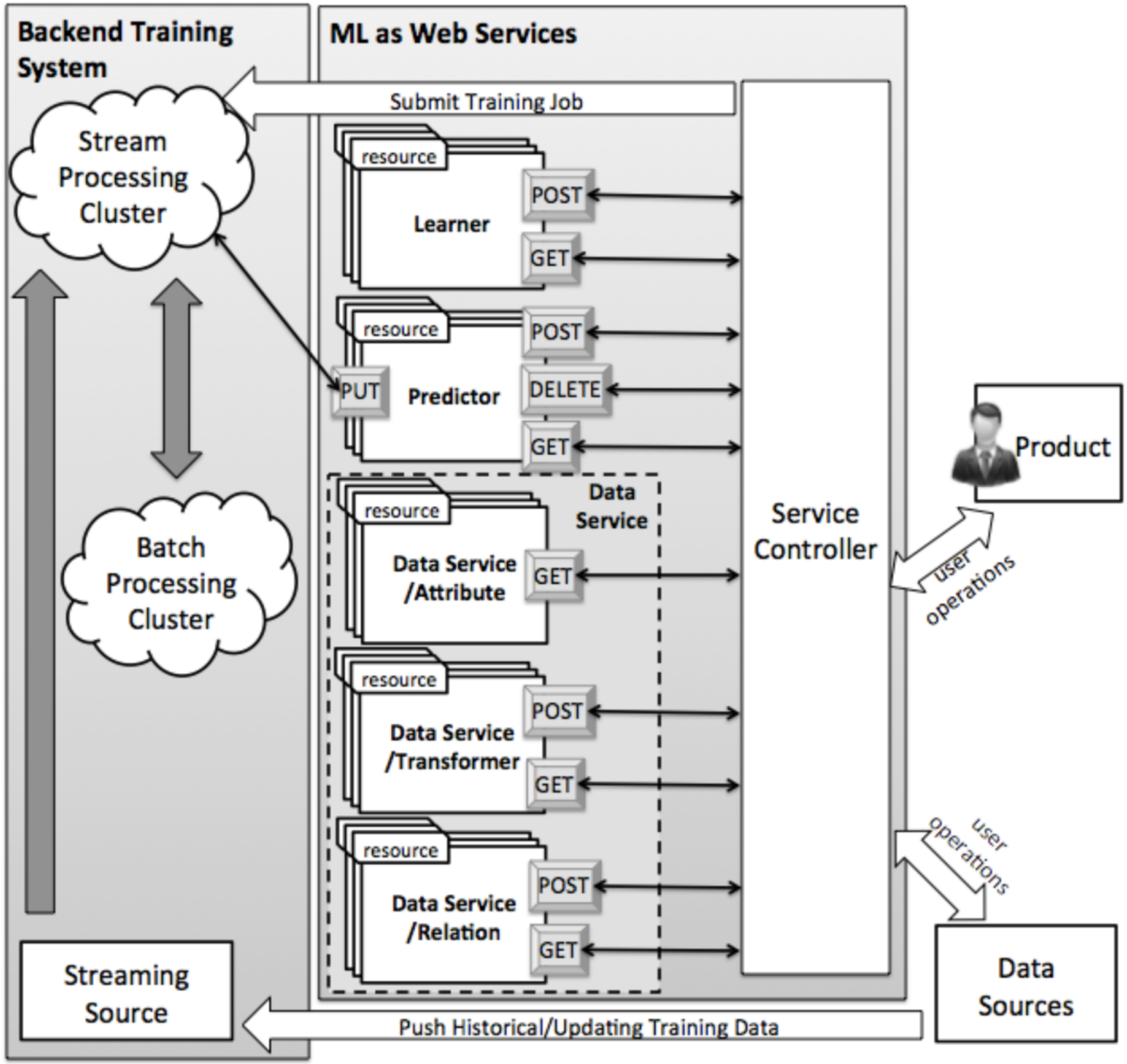
Real-Time Analytics

D.Xu, D.Wu, X.Xu, L.Zhu, and L.Bass
Making Real Time Data Analytics Available as a Service
Quality of Software Architectures (QoSA), 2015 11th International ACM SIGSOFT Conference on. IEEE, 2015, pp. 73-82

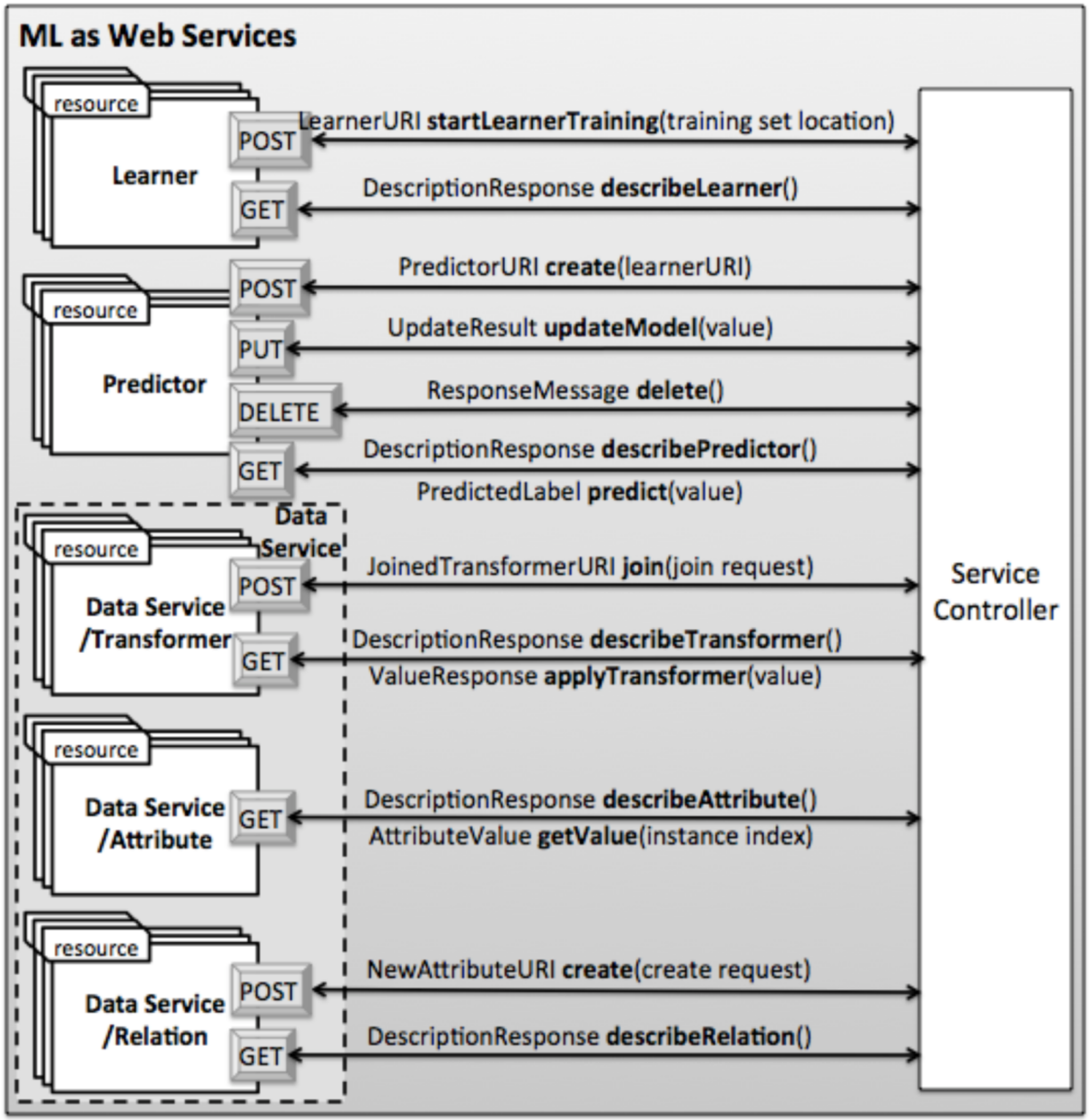
Data Analytics

- Scaling data processing to meet demand
- ML models to update with live data
- Combining big data processing with ML training

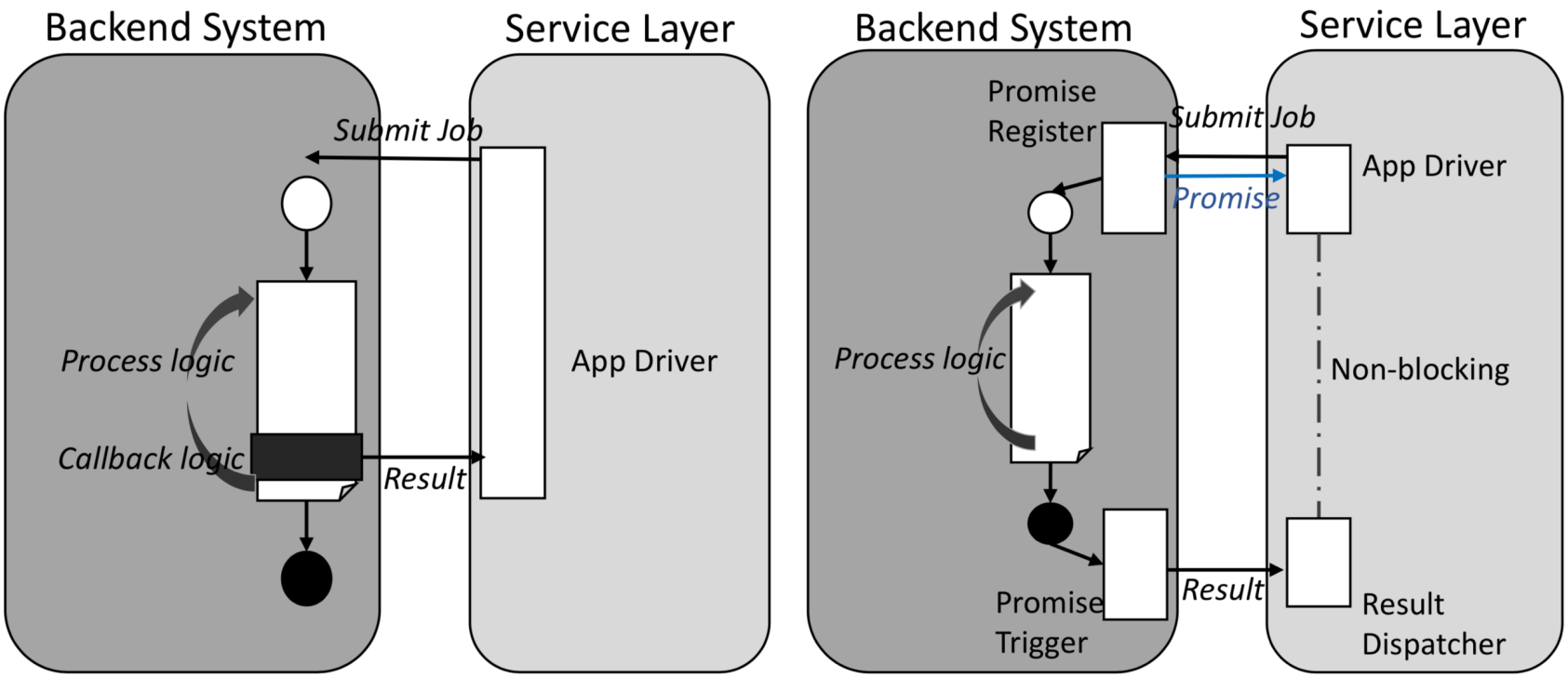
Components


Backend Training System
ML as Web Services
Submit Training Job
Stream Processing Cluster
Batch Processing Cluster
Streaming Source
resource
Learner
POST
GET
resource
PUT
Predictor
POST
DELETE
GET
Data Service
resource
Data Service /Attribute
GET
resource
Data Service /Transformer
POST
GET
resource
Data Service /Relation
POST
GET
Service Controller
Product
user operations
user operations
Data Sources
Push Historical/Updating Training Data

ML API

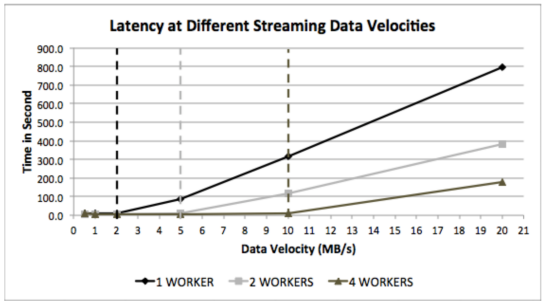

ML as Web Services
resource
Learner
POST
GET
LearnerURI startLearnerTraining(training set location)
DescriptionResponse describeLearner()
resource
Predictor
POST
PUT
DELETE
GET
PredictorURI create(learnerURI)
UpdateResult updateModel(value)
ResponseMessage delete()
DescriptionResponse describePredictor()
PredictedLabel predict(value)
Data Service
resource
Data Service /Transformer
POST
GET
JoinedTransformerURI join(join request)
DescriptionResponse describeTransformer()
ValueResponse applyTransformer(value)
resource
Data Service /Attribute
GET
DescriptionResponse describeAttribute()
AttributeValue getValue(instance index)
resource
Data Service /Relation
POST
GET
NewAttributeURI create(create request)
DescriptionResponse describeRelation()
Service Controller

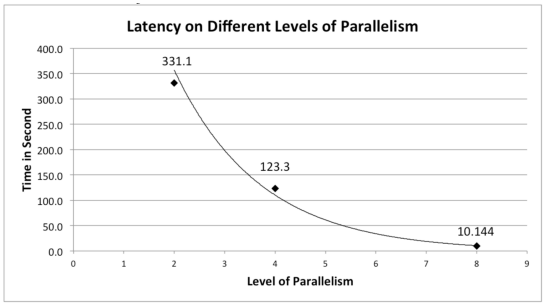
Processing Integration

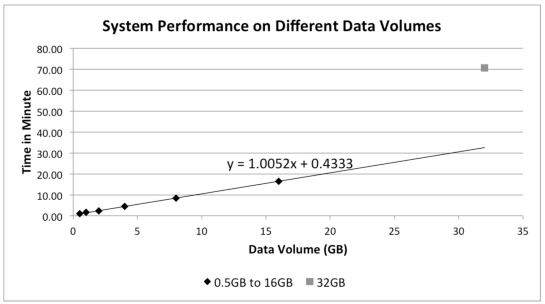


Testing

MLLib + Spark, 9 `t2.medium` (2 core, 4GB memory)
Simulated bitrates with file uploads every 250ms for 200s
2MB/s = 500KB every 250ms







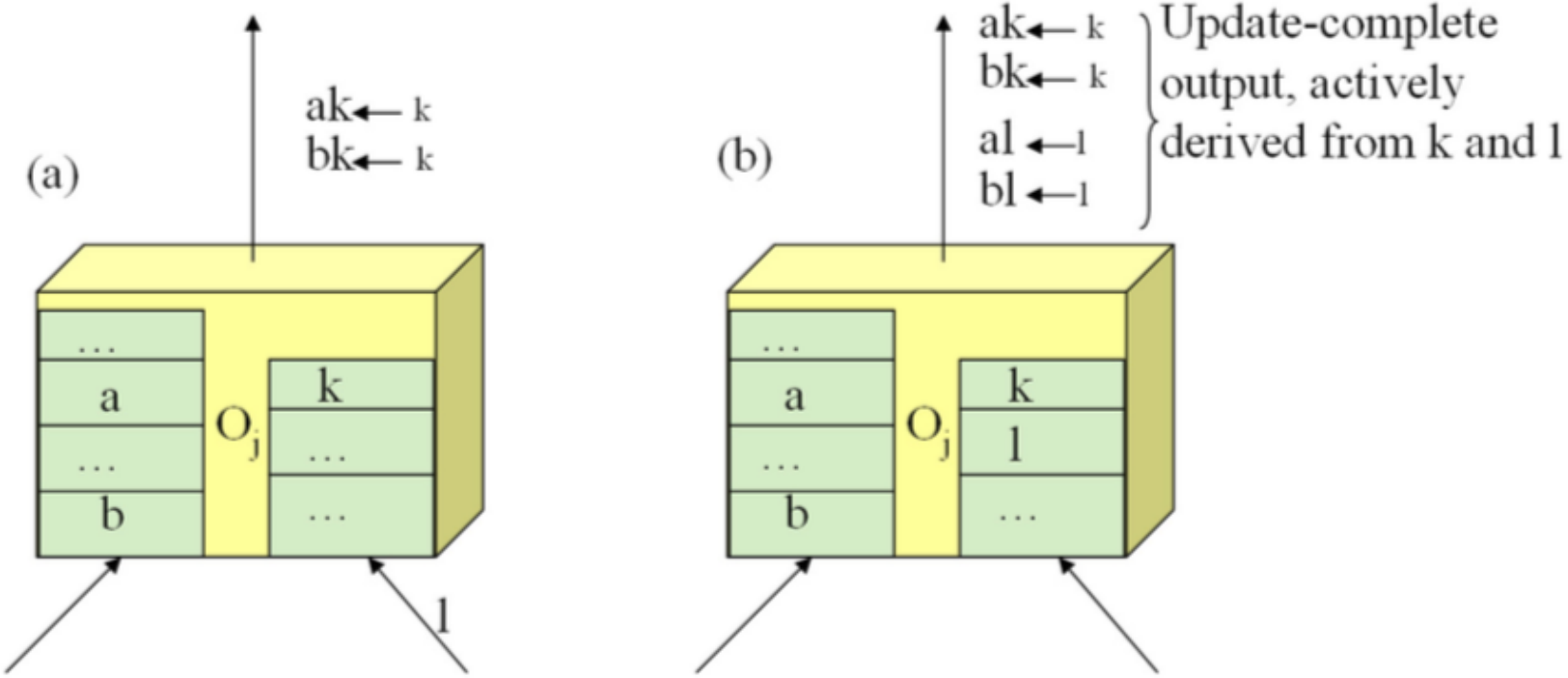
Summary

- Wrapped ML behind a REST API
- Adapted Stream Processing Systems to Support ML + API
- Small testing bit-rates
- Initialize state with parallel processing
  Update models with incremental changes

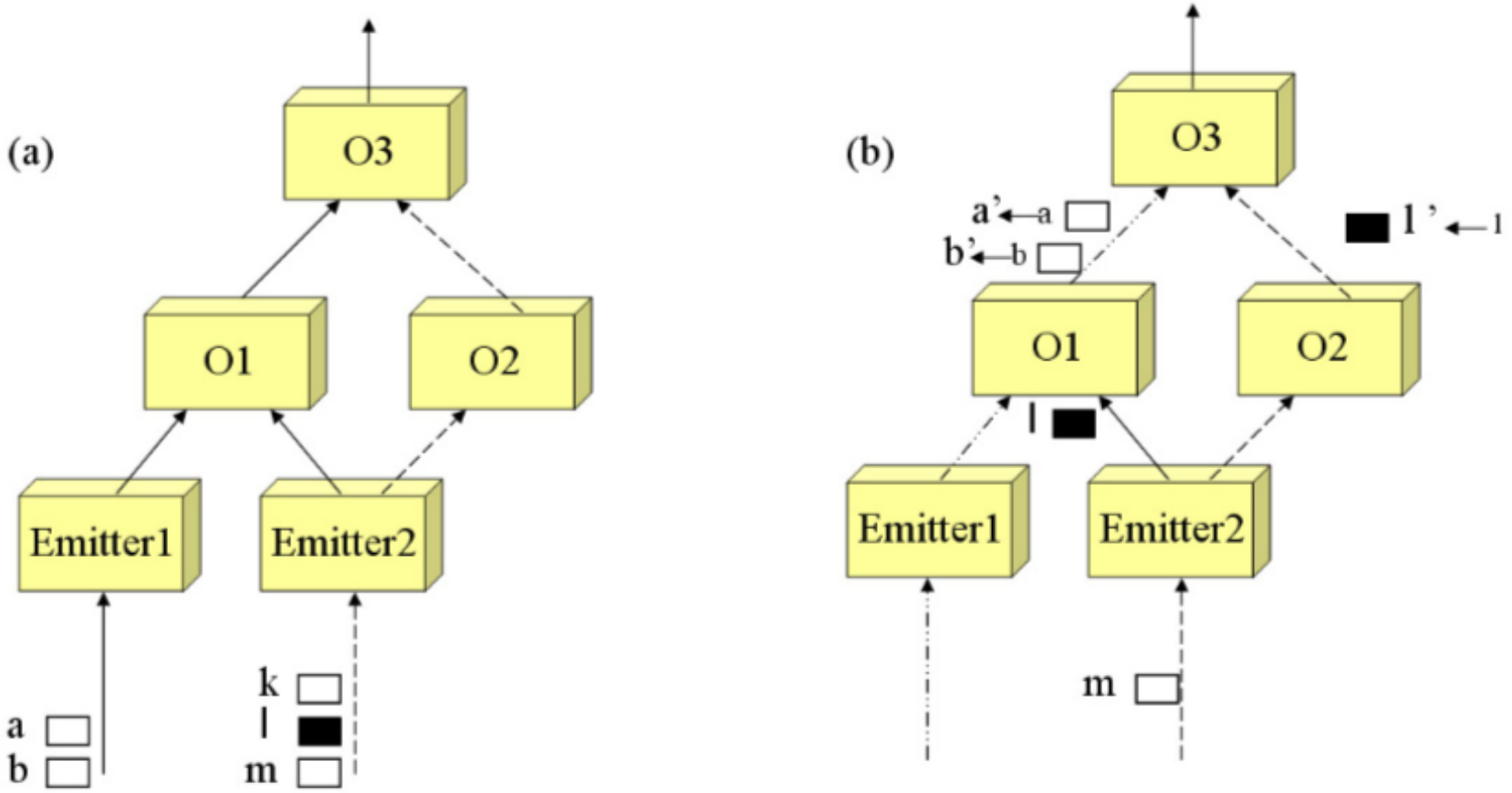
Consistency

G.A.Mihaila, I.Stanoi, and C.A.Lang
Anomaly-Free Incremental Output in Stream Processing
Proceedings of the 17th ACM Conference on Information and knowledge management. ACM, 2008, pp. 359-368

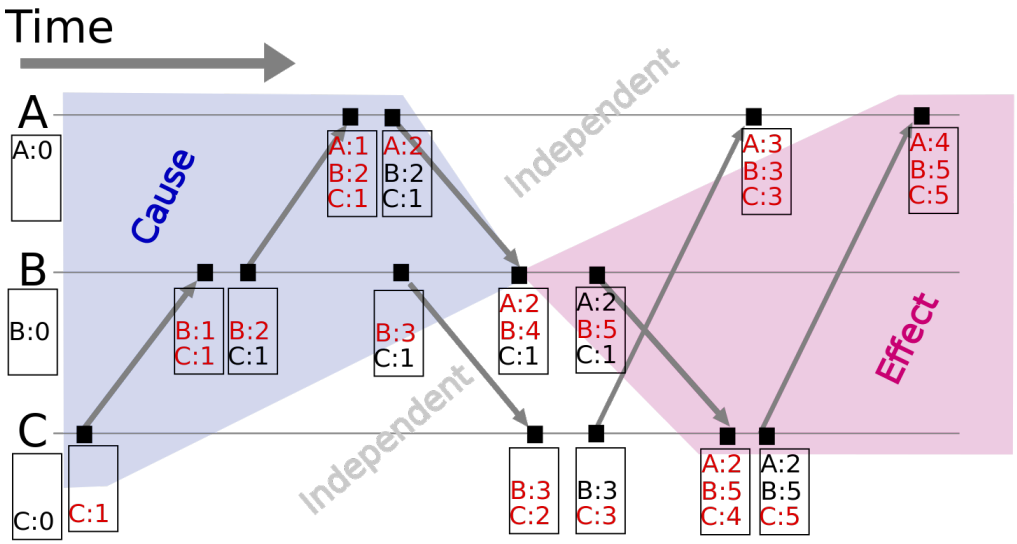
Data Inconsistency



Data Inconsistency Continued



Heeding

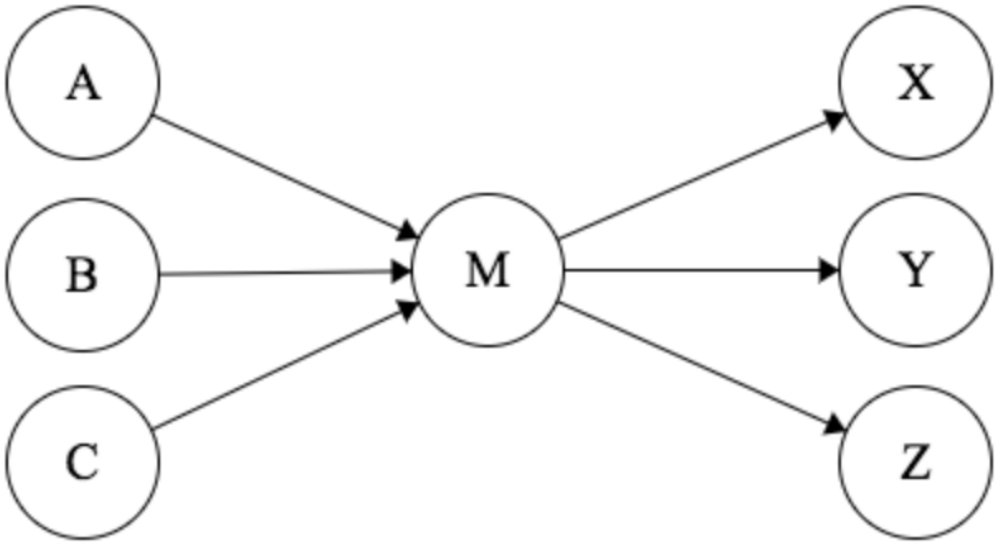


https://en.wikipedia.org/wiki/Vector_clock

Also labels active contributor to an event

Wait-and-See, Passive Consistency

Periodic Draining

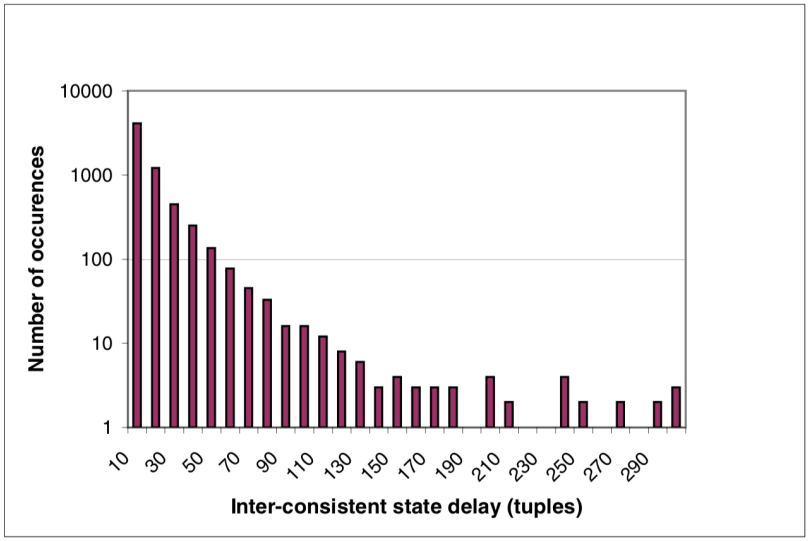


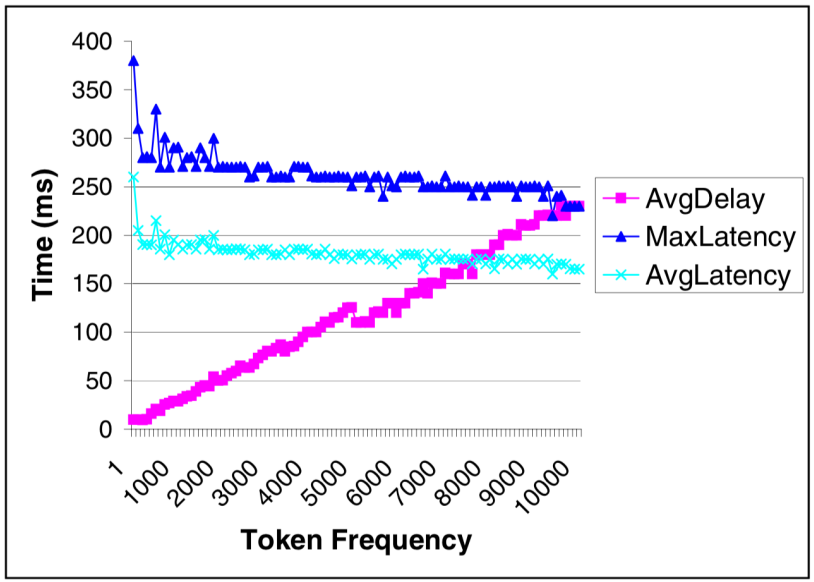
http://madebyevan.com/fsm/

Tokens injected every δT
Active Consistency

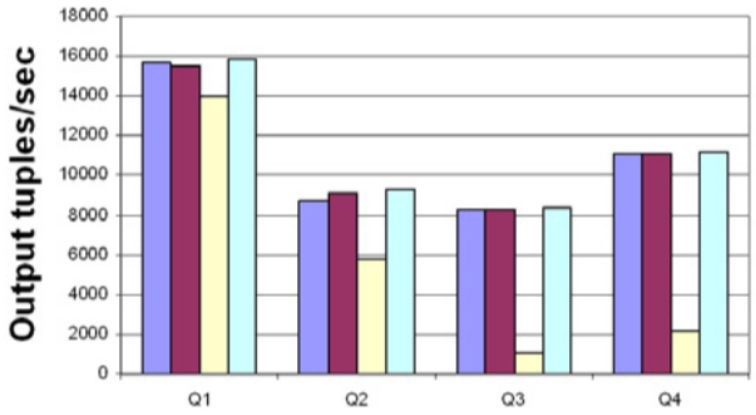
Testing

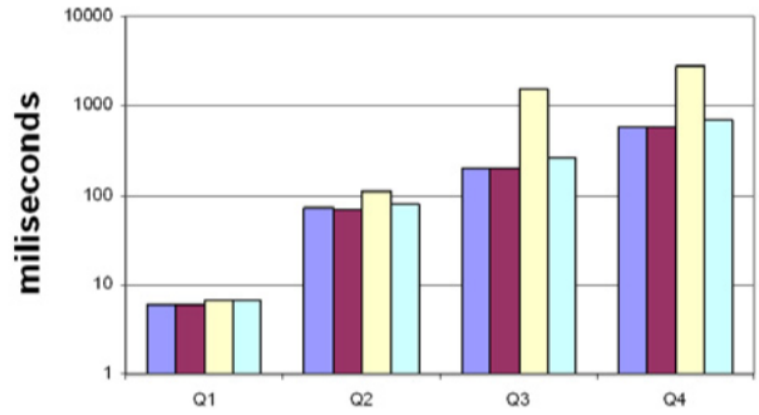
Existing WhiteWater System, Single machine, one operator/time
Round-Robin execution, Join Keys $[1, 5000]$

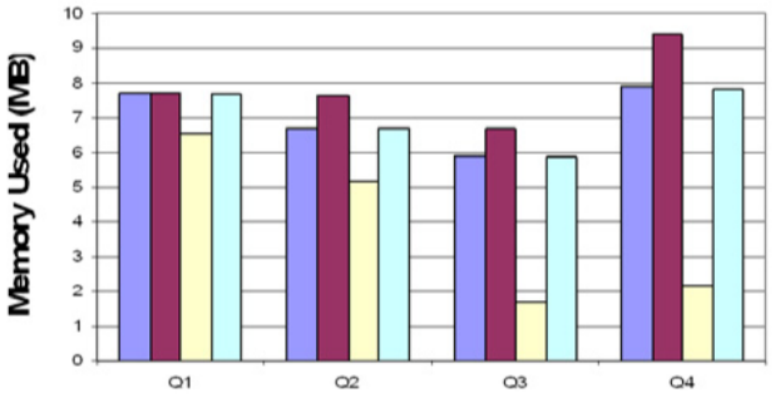


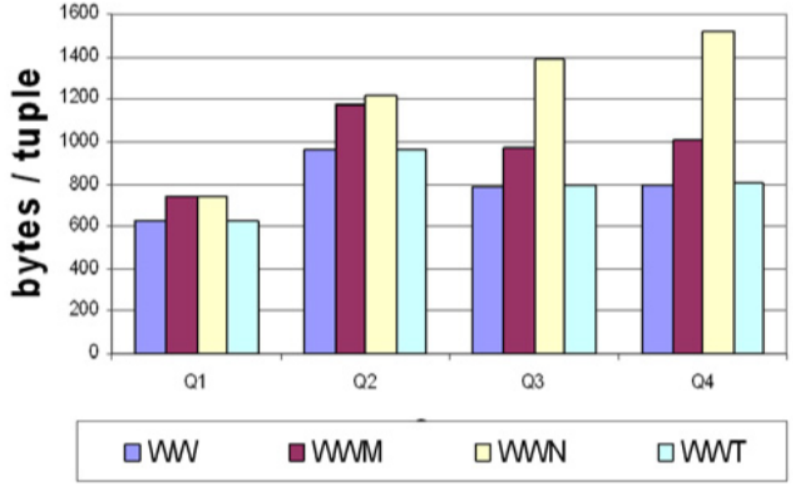


Testing


Output tuples/sec
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
Q1
Q2
Q3
Q4


miliseconds
10000
1000
100
10
1
Q1
Q2
Q3
Q4


Memory Used (MB)
10
9
8
7
6
5
4
3
2
1
0
Q1
Q2
Q3
Q4


bytes / tuple
1600
1400
1200
1000
800
600
400
200
0
Q1
Q2
Q3
Q4
WW
WWM
WWN
WWT

Summary

- Ensure Consistent Results in Stream Processing Engines
- Rather simple testing scenarios
- Integrate consistency with recovery

Open Research Questions

More Large-Scale Testing
Integrating Features
Recovery after large failures
Multiple Cluster processing

Thank You

Questions?

Slides available at `https://bign8.info/msu/qual.pdf`