DRIZZLE:
FAST AND ADAPTABLE STREAM PROCESSING AT SCALE

Shivaram Venkataraman, Aurojit Panda, Kay Ousterhout, Michael Armbrust, Ali Ghodsi, Michael Franklin, Benjamin Recht, Ion Stoica
STREAMING WORKLOADS
STREAMING TRENDS: LOW LATENCY

Results power decisions by machines

Credit card fraud
↓ Disable account

Suspicious user logins
↓ Ask security questions

Slow video load
↓ Direct user to new CDN
STREAMING REQUIREMENTS: HIGH THROUGHPUT

Disable stolen accounts

Detect suspicious logins

Dynamically adjust application behavior

As many as 10s of millions of updates per second

Need a distributed system
DISTRIBUTED EXECUTION MODELS
Group by user, run anomaly detection
Group by user, run anomaly detection

Mutable local state

Low latency output
Group by user, run anomaly detection

**Execution Models: Continuous Operators**

- Systems:
  - Naiad
  - Flink
  - Google MillWheel

- Streaming DBs:
  - Borealis, Flux etc

- Mutable local state

- Low latency output
Group by user, run anomaly detection

Tasks output state on completion

Output at task granularity

EXECUTION MODELS: MICRO-BATCH

Micro-batch
**EXECUTION MODELS:** MICRO-BATCH

- Group by user, run anomaly detection
- Dynamic task scheduling
  - Adaptability
  - Straggler mitigation
  - Elasticity
  - Fault tolerance
- Output at task granularity
  - Tasks output state on completion
- Tasks output state on completion
FAILURE RECOVERY
FAILURE RECOVERY: CONTINUOUS OPERATORS

Chandy Lamport Async Checkpoint

Checkpointed state
FAILURE RECOVERY: MICRO-BATCH

Task output is periodically checkpointed

Task boundaries capture task interactions!
Failure recovery: Micro-batch

- Task output is periodically checkpointed.
- Parallelize replay.
- Replay tasks from failed machine.
**EXECUTION MODELS**

**Continuous operators**
- Static scheduling
- Inflexible
- Slow failover
- **Low latency**

**Micro-batch**
- **Scheduling granularity**
  - Dynamic scheduling
  - Adaptable
  - Parallel recovery
  - Straggler mitigation
- **Processing granularity**
  - Higher latency
**EXECUTION MODELS**

**Continuous operators**
- Static scheduling
- **✓ Low latency**

**Drizzle**
- **✓ Dynamic scheduling** (coarse granularity)
- **✓ Low latency**
- (fine-grained processing)

**Micro-batch**
- **✓ Dynamic scheduling** (coarse granularity)
- Higher latency
- (coarse-grained processing)
Inside the scheduler:

1. Decide how to assign tasks to machines
   - data locality
   - fair sharing

2. Serialize and send tasks
Cluster: 4 core, r3.xlarge machines
Workload: Sum of 10k numbers per-core

SCHEDULING OVERHEADS

Median-task time breakdown

- Compute + Data Transfer
- Task Fetch
- Scheduler Delay

Cluster: 4 core, r3.xlarge machines
Workload: Sum of 10k numbers per-core
inside the scheduler

(1) Decide how to assign tasks to machines
   - data locality
   - fair sharing

(2) Serialize and send tasks

Reuse scheduling decisions!
DRIZZLE

Goal:
remove frequent scheduler interaction

(1) Pre-schedule reduce tasks

(2) Group schedule micro-batches
Goal: Remove scheduler involvement for reduce tasks

(1) **Pre-schedule** reduce tasks
Goal: Remove scheduler involvement for reduce tasks

(1) **Pre-schedule** reduce tasks
COORDINATING SHUFFLES: EXISTING SYSTEMS

Metadata describes shuffle data location

Data fetched from remote machines
COORDINATING SHUFFLES: PRE-SCHEDULING

1. Pre-schedule reducers
2. Mappers get metadata
3. Mappers trigger reducers
DRIZZLE

Goal:
wait to return to scheduler

(1) Pre-schedule reduce tasks

(2) Group schedule micro-batches
GROUP SCHEDULING

Group of 2

Schedule group of micro-batches at once

Fault tolerance, scheduling at group boundaries
**MICRO-BENCHMARK: 2-STAGES**

100 iterations – Breakdown of pre-scheduling, group-scheduling

- **Baseline**
- **Only Pre-Scheduling**
- **Drizzle-10**
- **Drizzle-100**

In the paper: group size auto-tuning
EVALUATION

Continuous operators

Static scheduling

1. Latency?
   - Low latency

Drizzle

- Dynamic scheduling (coarse granularity)
  - Low latency (fine-grained processing)

Micro-batch

- Dynamic scheduling (coarse granularity)

2. Adaptability?
   - (coarse-grained processing)
EVALUATION: LATENCY

Yahoo! Streaming Benchmark

Input: JSON events of ad-clicks
Compute: Number of clicks per campaign
Window: Update every 10s

Comparing Spark 2.0, Flink 1.1.1, Drizzle
128 Amazon EC2 r3.xlarge instances
STREAMING BENCHMARK - PERFORMANCE

Yahoo Streaming Benchmark: 20M JSON Ad-events / second, 128 machines

Event Latency: Difference between window end, processing end

Event Latency (ms)

- Spark
- Drizzle
- Flink
ADAPTABILITY: FAULT TOLERANCE

Yahoo Streaming Benchmark: 20M JSON Ad-events / second, 128 machines
Inject machine failure at 240 seconds
EXECUTION MODELS

Continuous operators

- Static scheduling
- Low latency

Drizzle

- Dynamic scheduling (coarse-granularity)
- Low latency (fine-grained processing)
- Optimization of batches

Micro-batch

- Dynamic scheduling
- Higher latency
- Optimization of batches
Optimize execution of each micro-batch by pushing down aggregation

Yahoo Streaming Benchmark: 20M JSON Ad-events / second, 128 machines

INTRA-BATCH QUERY OPTIMIZATION
# EVALUATION

<table>
<thead>
<tr>
<th>End-to-end Latency</th>
<th>Yahoo Streaming Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tolerance</td>
<td></td>
</tr>
<tr>
<td>Query optimization</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Throughput</th>
<th>Synthetic micro-benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elasticity</td>
<td>Video Analytics</td>
</tr>
<tr>
<td>Group-size tuning</td>
<td>Shivaram’s Thesis: Iterative ML Algorithms</td>
</tr>
</tbody>
</table>
CONCLUSION

Continuous operators

Static scheduling
- Low latency

Drizzle
- Dynamic scheduling (coarse granularity)
- Low latency (fine-grained processing)
- Optimization of batches

Source code: https://github.com/amplab/drizzle-spark

Micro-batch
- Dynamic scheduling (coarse granularity)
- Higher latency (coarse-grained processing)
- Optimization of batches

Shivaram is answering questions on sli.do