Scalable Stream Processing
Surveying Storm, Samza, Spark & Flink

Wolfram Wingerath
ww@baqend.com
September 12, techcamp 2018, Hamburg
Research:
• Real-Time Databases
• Stream Processing
• NoSQL & Cloud Databases
• ...

Practice:
• Backend-as-a-Service
• Web Caching
• Real-Time Database
• ...

About me
Wolfram Wingerath
PhD Thesis & Research
Distributed Systems Engineer

University Hamburg
www.baqend.com
Outline

- **Introduction**
  - Big Data in Motion

- **System Survey**
  - Big Data + Low Latency

- **Wrap-Up**
  - Summary & Discussion

- **Future Directions**
  - Real-Time Databases

- **Big Picture**:
  - A Typical Data Pipeline
  - Processing Frameworks

- **Processing Models**:
  - Batch Processing
  - Stream Processing

- **Future Directions**
  - Real-Time Databases
IN PRACTICE

Scalable Data Processing
A Data Processing Pipeline

Today’s topic!
INTRODUCTION

Batch vs Stream Processing
Big Data Processing Frameworks

What are your options?

What to use when?

- Spark Streaming
- Google Dataflow
- HERON
- Amazon Elastic MapReduce
- STORM
- STORM Trident
- IBM InfoSphere Streams
- Flink
- APEX
- kafka streams
- samza
- hadoop
- concord
Batch Processing

„Volume“

• **Cost-effective** & Efficient
• **Easy to reason about**: operating on complete data

But:
• **High latency**: jobs periodic (e.g. during night times)

---

Persistence (e.g. HDFS)  Batch (e.g. MapReduce)  Serving (e.g. HBase)  Application
Stream Processing „Velocity“

• Low end-to-end latency
• Challenges:
  • **Long-running jobs** - no downtime allowed
  • **Asynchronism** - data may arrive delayed or out-of-order
  • **Incomplete input** - algorithms operate on partial data
  • More: fault-tolerance, state management, guarantees, ...

![Diagram of Stream Processing](image_url)
Typical Stream Operators

Examples

Filter & Transform

Filter

Map

Group

GroupByKey

Aggregates

SUM()

COUNT()

Windows

Tumbling

Sliding

https://www.infoq.com/presentations/stream-processors-databases

https://www.infoq.com/presentation/stream-processing-apache-flink
Typical Use Case
Example from Yahoo!

Input
• **Read** Ad tracking data **from Kafka**

Filter
• **Discard** useless data

Project
• **Extract** relevant fields

Group
• By Ad **campaign**

Window
• Ad views per **10-min-window**
Wrap-up
Data Processing

• Processing frameworks abstract from **scaling issues**

**Batch processing**
• easy to reason about
• extremely efficient
• huge input-output latency

**Stream processing**
• quick results
• purely incremental
• potentially complex to handle
Outline

∑  Introduction
  Big Data in Motion

System Survey
  Big Data + Low Latency
  
• System Survey:
  • Processing Model Overview
  • Storm/Trident
  • Samza
  • Spark Streaming
  • Flink

Wrap-Up
  Summary & Discussion

Future Directions
  Real-Time Databases
SURVEY

Popular Stream Processing Systems
Processing Models
Batch vs. Micro-Batch vs. Stream

- **stream**: Flink, Storm Trident, Samza
- **micro-batch**: Hadoop, Spark Streaming
- **batch**: Amazon Elastic MapReduce

- **low latency**
- **high throughput**
Storm
„Hadoop of real-time“

Overview
- **First** production-ready, well-adopted stream processor
- **Compatible**: native Java API, Thrift, distributed RPC
- **Low-level**: no primitives for joins or aggregations
- **Native stream processor**: latency < 50 ms feasible
- **Big users**: Twitter, Yahoo!, Spotify, Baidu, Alibaba, ...

History
- **2010**: developed at BackType (acquired by Twitter)
- **2011**: open-sourced
- **2014**: Apache top-level project
Dataflow

Directed Acyclic Graphs (DAG):
- **Spouts**: pull data into topology
- **Bolts**: do processing, emit data
- Asynchronous
- Lineage can be tracked for each tuple
→ At-least-once has **2x messaging overhead**
State Management

Recover State on Failure

• **In-memory** or **Redis**-backed reliable state
• *Synchronous state communication* on the critical path

→ infeasible for large state
**Back Pressure**
Throttling Ingestion on Overload

1. too many tuples → 2. tuples time out and fail → 3. tuples get replayed

**Approach**: monitoring bolts’ inbound buffer
1. Exceeding **high watermark** → throttle!
2. Falling below **low watermark** → full power!
Trident
Stateful Stream Joining on Storm

Overview:

- Abstraction layer on top of Storm
- Released in 2012 (Storm 0.8.0)
- Micro-batching
- New features:
  - High-level API: aggregations & joins
  - Strong ordering
  - Stateful exactly-once processing
    → Performance penalty
Trident
Partitioned Micro-Batching

Illustration taken from: “Storm applied”, Sean T. Allen et al.
Samza
Real-Time on Top of Kafka

Overview
- Co-developed with Kafka
  → Kappa Architecture
- **Simple**: only single-step jobs
- Local state
- Native stream processor: low latency
- **Users**: LinkedIn, Uber, Netflix, TripAdvisor, Optimizely, ...

History
- Developed at LinkedIn
- **2013**: open-source (Apache Incubator)
- **2015**: Apache top-level project

Illustration taken from: Jay Kreps, Questioning the Lambda Architecture (2014)
Dataflow
Simple By Design

- **Job**: processing step (≈ Storm bolt)
  - Robust
  - But: often several jobs
- **Task**: job instance (parallelism)
- **Message**: single data item
- **Output persisted** in Kafka
  - Easy data sharing
  - Buffering (no back pressure!)
  - But: Increased latency
- **Ordering** within partitions
- Task = Kafka partitions: not-elastic on purpose

---

Advantages of local state:

- **Buffering**
  - No back pressure
  - At-least-once delivery
  - Simple recovery

- **Fast lookups**

Illustrations taken from: Jay Kreps, *Why local state is a fundamental primitive in stream processing* (2014)
Dataflow
Example: Enriching a Clickstream

Example: the enriched clickstream is available to every team within the organization

Illustration taken from: Jay Kreps, Why local state is a fundamental primitive in stream processing (2014)
State Management
Straightforward Recovery

Spark
„MapReduce successor“

Overview

- **High-level API**: immutable collections (RDDs)
- **Community**: 1000+ contributors in 2015
- **Big users**: Amazon, eBay, Yahoo!, IBM, Baidu, ...

History

- **2009**: developed at UC Berkeley
- **2010**: open-sourced
- **2014**: Apache top-level project
Overview

- **High-level API**: DStreams (~Java 8 Streams)
- **Micro-Batching**: seconds of latency
- **Rich features**: stateful, exactly-once, elastic

History

- **2011**: start of development
- **2013**: Spark Streaming becomes part of Spark Core
Spark Streaming
Core Abstraction: DStream

Resilient Distributed Data set (RDD)
- Immutable collection & deterministic operations
- Lineage tracking:
  → state can be reproduced
  → periodic checkpoints reduce recovery time

DStream: Discretized RDD
- RDDs are processed in order: no ordering within RDD
- RDD scheduling ~50 ms → latency >100ms

Illustration taken from:
http://spark.apache.org/docs/latest/streaming-programming-guide.html#overview (2017-02-26)
Example
Counting Page Views

```javascript
pageViews = readStream("http://...", "1s")
ones = pageViews.map(event => (event.url, 1))
counts = ones.runningReduce((a, b) => a + b)
```

Flink

Overview

- **Native stream processor**: Latency <100ms feasible
- **Abstract API** for stream and batch processing, stateful, exactly-once delivery
- **Many libraries**: Table and SQL, CEP, Machine Learning, Gelly...
- **Users**: Alibaba, Ericsson, Otto Group, ResearchGate, Zalando...

History

- **2010**: start as *Stratosphere* at TU Berlin, HU Berlin, and HPI Potsdam
- **2014**: Apache Incubator, project renamed to Flink
- **2015**: Apache top-level project
Architecture

Streaming + Batch
Managed State
Streaming + Batch

- Automatic **Backups** of local state
- Stored in **RocksDB**, Savepoints written to **HDFS**

https://www.infoq.com/presentation/s/stream-processing-apache-flink
Highlight: Fault Tolerance
Distributed Snapshots

- **Ordering** within stream partitions
- **Periodic** checkpoints
- **Recovery**:
  1. *reset state* to checkpoint
  2. *replay data* from there

Outline

- **Introduction**
  Big Data in Motion

- **System Survey**
  Big Data + Low Latency

- **Wrap-Up**
  Summary & Discussion

- **Future Directions**
  Real-Time Databases

- **Discussion:**
  - Comparison Matrix
  - Other Systems
  - Takeaway
WRAP UP

Side-by-side comparison
# Comparison

<table>
<thead>
<tr>
<th></th>
<th>Storm</th>
<th>Trident</th>
<th>Samza</th>
<th>Spark Streaming</th>
<th>Flink (streaming)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strictest Guarantee</strong></td>
<td>at-least-once</td>
<td>exactly-once</td>
<td>at-least-once</td>
<td>exactly-once</td>
<td>exactly-once</td>
</tr>
<tr>
<td><strong>Achievable Latency</strong></td>
<td>&lt;&lt;100 ms</td>
<td>&lt;100 ms</td>
<td>&lt;100 ms</td>
<td>&lt;1 second</td>
<td>&lt;100 ms</td>
</tr>
<tr>
<td><strong>State Management</strong></td>
<td><img src="small" alt="Circle" /></td>
<td><img src="small" alt="Circle" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
</tr>
<tr>
<td></td>
<td>(small state)</td>
<td>(small state)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Processing Model</strong></td>
<td>one-at-a-time</td>
<td>micro-batch</td>
<td>one-at-a-time</td>
<td>micro-batch</td>
<td>one-at-a-time</td>
</tr>
<tr>
<td><strong>Backpressure</strong></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%97" alt="Cross" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no (buffering)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td><img src="%E2%9C%97" alt="Cross" /></td>
<td>between batches</td>
<td>within partitions</td>
<td>between batches</td>
<td>within partitions</td>
</tr>
<tr>
<td><strong>Elasticity</strong></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%97" alt="Cross" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
<td><img src="%E2%9C%93" alt="Checkmark" /></td>
</tr>
</tbody>
</table>
Performance
Yahoo! Benchmark

- Based on **real use case:**
  - Filter and count ad impressions
  - 10 minute windows

“**Storm [...]** and **Flink [...]** show **sub-second latencies** at relatively high throughputs with **Storm** having the **lowest 99th percentile** latency. **Spark** streaming [...] supports high throughputs, but at a relatively **higher latency.”

Other Systems

Heron
Beam
Kafka Streams
Apex
IBM InfoSphere Streams
Dataflow

And even more: Kinesis, Gearpump, MillWheel, Muppet, S4, Photon, ...
Outline

- **Introduction**
  - Big Data in Motion

- **System Survey**
  - Big Data + Low Latency

- **Wrap-Up**
  - Summary & Discussion

- **Future Directions**
  - Real-Time Databases

- **Real-Time Databases:**
  - **Why** Push-Based Database Queries?
  - **Where** Do Real-Time Databases Fit in?

- **Comparison Matrix:**
  - Meteor
  - RethinkDB
  - Parse
  - Firebase
  - Baqend
REAL-TIME DBS

Combining databases with streaming
Traditional Databases
No Request? No Data!

What’s the current state?

Query maintenance: periodic polling
→ Inefficient
→ Slow
Quick Comparison
DBMS vs. RT DB vs. DSMS vs. Stream Processing

Database Management
- static collections
- pull-based

Real-Time Data Stream Management
- evolving collections
- ephemeral streams
- push-based

Stream Processing
- persistent/ephemeral streams
- ephemeral streams
# Real-Time Databases

## In a Nutshell

<table>
<thead>
<tr>
<th></th>
<th>Meteor</th>
<th>RethinkDB</th>
<th>Parse</th>
<th>Firebase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scales with write TP</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Scales with no. of queries</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Composite queries (AND/OR)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sorted queries</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Limit</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Offset</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

- **Meteor** supports Poll-and-Diff and Oplog Tailing.
- **RethinkDB** does not support write TP, but supports other features.
- **Parse** supports all features except for composite queries.
- **Firebase** supports real-time databases, but has limitations on the number of connections.

**Note:**

- (100k connections)
- (AND in Firestore)
- (single attribute)
- (value-based)
TAKEAWAY

Trade-Offs in Stream Processing
Summary

- **Stream Processors:**
  - STORM
  - Flink
  - Samza
  - Spark Streaming

- **Real-Time Databases** integrate Storage & Streaming

- **Learn more:** slides.baqend.com
Who We Are

Our Product

**Speed Kit:**
- Accelerates *Any* Website
- Pluggable
- Easy Setup

[test.speed-kit.com](http://test.speed-kit.com)

Our Services

- Web & Data Management Workshops
- Performance Auditing
- Implementation Services

[consulting@baqend.com](mailto:consulting@baqend.com)

Wolfram Wingerath

ww@baqend.com

Distributed Systems Engineer