Stream processing everywhere—what to use?

Jim Scott – Director, Enterprise Strategy & Architecture

@kingmesal #StrataHadoop
The Landscape

• Stream Processing is Fundamentally Simple
  – Inputs -> Outputs
  – But it is WAY more complicated than this…

• Optimization can be complicated

• This space is very confused
  – Performance of different options is dependent upon source

• Lots of misinformation
  – e.g. performance comparisons that are not apples-to-apples
Semantics

• There are three general categories of delivery patterns:
  – At-most-once: messages may be lost. This is usually the least desirable outcome.
  – At-least-once: messages may be redelivered (no loss, but duplicates). This is good enough for many use cases.
  – Exactly-once: each message is delivered once and only once (no loss, no duplicates). This is a desirable feature although difficult to guarantee in all cases.
Today’s Options – Apache Style

- Samza
- Storm
- Spark Streaming
Apache Samza
Apache Samza

- Originally developed in LinkedIn (Chris Riccomini/Jay Kreps), now ASF top-level project
- Distributed stream processing framework (YARN/Kafka)

http://samza.apache.org/
Concepts

- Streams & Partitions
- Jobs & Tasks
- Dataflow Graphs
- Containers

samza.apache.org/learn/documentation/latest/introduction/concepts.html
Streams & Partitions

- **Stream**: immutable messages
- **Each stream comprises one or more partitions**
- **Partition**: totally ordered sequence of messages
Jobs & Tasks

- **Job**: logical unit of stream processing, a collection of tasks
- **Task**: unit of parallelism
  - in the context of a job, each task consumes data from one partition
  - processes messages from each of its input partitions sequentially
Dataflow Graphs

• **Dataflow graph**: logical, directed graph of jobs
• Jobs in DG are decoupled
  – can be developed independently
  – don’t impact up/downstream jobs
• DG can contain cycles
Samza Architecture

- **Processing layer** → Samza API
- **Pluggable execution layer**
  (default: YARN)
- **Pluggable streaming layer**
  (default: Kafka)

samza.apache.org/learn/documentation/latest/introduction/architecture.html
Samza Execution: Containers

• Partitions and tasks are both logical units of parallelism

• Containers are the unit of physical parallelism, essentially a Unix process (or Linux cgroup)
Samza Resources

- http://www.infoq.com/articles/linkedin-samza

Kudos to Chris Riccomini and Martin Kleppmann for their invaluable support concerning Samza!
Apache Storm
Apache Storm

- Originally developed by Nathan Marz at Backtype/Twitter, now ASF top-level project
- Distributed, fault-tolerant stream-processing platform

http://storm.apache.org/
Concepts

• Tuples and Streams
• Spouts, Bolts, Topologies
• Tasks and Workers
• Stream Grouping

storm.apache.org/documentation/Tutorial.html
Tuples and Streams

- **Tuple**: ordered list of elements
- **Stream**: unbounded sequence of tuples

```
<table>
<thead>
<tr>
<th>Tuple</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(b20ea50, <a href="mailto:nathan@nathanmarz.com">nathan@nathanmarz.com</a>)</td>
<td>(064874b, <a href="mailto:andy.feng@gmail.com">andy.feng@gmail.com</a>)</td>
</tr>
</tbody>
</table>
```

stream
Spouts

• The sources of streams

• Can talk with
  – Queues (Kafka, Kestrel, etc.)
  – Web logs
  – API calls
  – Filesystem (MapR-FS / HDFS)
  – Etc.
Bolts

• Process tuples and create new streams

• Implement business logic via …
  – Transform
  – Filter
  – Aggregate
  – Join
  – Access datastores & DBs
  – Access APIs (e.g., geo location look-up)
Topologies

- Directed graph of spouts and bolts
Stream Grouping

- **Shuffle grouping**: tuples are randomly distributed across all of the tasks running the bolt.

- **Fields grouping**: groups tuples by specific name field and routes to the same task.
Tasks and Workers

- **Task**: each spout/bolt executes as many threads of execution across the cluster

- **Worker**: a physical JVM that executes a subset of all the tasks for the topology
Trident—the ‘Cascading’ of Storm

- High-level abstraction processing library on top of Storm
- Rich API with joins, aggregations, grouping, etc.
- Provides stateful, exactly-once processing primitives

storm.apache.org/documentation/Trident-tutorial.html
Execution

- **Nimbus**
- **Zookeeper**
- **Supervisor**
- **Worker**

Diagram showing the components of the execution environment:
- **Master node**
  - Nimbus
  - 0MQ/Netty
  - Supervisor
  - Workers
- **Worker node**
  - Supervisor
  - Workers
Storm Resources

- https://www.udacity.com/course/ud381
- http://www.manning.com/sallen/
- https://github.com/tdunning/storm-counts
Apache Spark
Apache Spark

Continued innovation bringing new functionality, such as:

- **Tachyon** (Shared RDDs, off-heap solution)
- **BlinkDB** (approximate queries)
- **SparkR** (R wrapper for Spark)

Spark SQL
(SQL/HQL)

Spark Streaming
(stream processing)

MLlib
(machine learning)

GraphX
(graph processing)

Spark (core execution engine—RDDs)

Mesos

Standalone

YARN

file system (local, MapR-FS, HDFS, S3) or data store (HBase, Elasticsearch, etc.)

http://spark.apache.org/
Sweet spot …

• Iterative Algorithms
  – machine learning
  – graph processing beyond DAG

• Interactive Data Mining

• Streaming Applications
Interfacing to permanent storage

• Local Files
  – file:///opt/httpd/logs/access_log

• Object Stores (e.g. Amazon S3)

• MapR-FS, HDFS
  – text files, sequence files, any other Hadoop InputFormat

• Key-Value datastores (e.g. Apache HBase)

• Elasticsearch
Cluster Managers

YARN

Standalone
Resilient Distributed Datasets (RDD)

- **RDD**: core abstraction of Spark execution engine
- Collections of elements that can be operated on in parallel
- Persistent in memory between operations

RDD Operations

• Lazy evaluation is key to Spark

• Transformations
  – Creation of a new dataset from an existing: `map`, `filter`, `distinct`, `union`, `sample`, `groupByKey`, `join`, etc.

• Actions
  – Return a value after running a computation: `collect`, `count`, `first`, `takeSample`, `foreach`, etc.
Spark Streaming

- High-level language operators for streaming data
- Fault-tolerant semantics
- Support for merging streaming data with historical data

spark.apache.org/docs/latest/streaming-programming-guide.html
Spark Streaming

Run a streaming computation as a series of small, deterministic batch jobs.

- Chop up live stream into batches of $X$ seconds (DStream)
- Spark treats each batch of data as RDDs and processes them using RDD ops
- Finally, processed results of the RDD operations are returned in batches
Spark Streaming

Run a streaming computation as a series of small, deterministic batch jobs.

• Batch sizes as low as ½ second, latency of about 1 second
• Potential for combining batch processing and streaming processing in the same system
Spark Streaming: Transformations

- Stateless transformations
- Stateful transformations
  - checkpointing
  - windowed transformations
    - window duration
    - sliding duration

spark.apache.org/docs/latest/streaming-programming-guide.html#transformations-on-dstreams
Spark Streaming: Execution
Spark Resources

- [http://databricks.com/spark-training-resources](http://databricks.com/spark-training-resources)
- [http://oreilly.com/go/sparkcert](http://oreilly.com/go/sparkcert)
- [http://spark-stack.org](http://spark-stack.org)
- [https://www.mapr.com/blog/getting-started-spark-mapr-sandbox](https://www.mapr.com/blog/getting-started-spark-mapr-sandbox)
Comparison
<table>
<thead>
<tr>
<th></th>
<th>Samza</th>
<th>Storm</th>
<th>Spark Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stream Source</strong></td>
<td>Consumers</td>
<td>Spouts</td>
<td>Receivers</td>
</tr>
<tr>
<td><strong>Stream Primitive</strong></td>
<td>Message</td>
<td>Tuple</td>
<td>DStream</td>
</tr>
<tr>
<td><strong>Stream Computation</strong></td>
<td>Tasks</td>
<td>Bolts</td>
<td>Transformations</td>
</tr>
</tbody>
</table>

Samza vs Storm vs Spark
## Samza vs Storm vs Spark

<table>
<thead>
<tr>
<th></th>
<th>Samza</th>
<th>Storm</th>
<th>Spark Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>processing model</strong></td>
<td>one record at a time</td>
<td>one record at a time</td>
<td>micro-batch</td>
</tr>
<tr>
<td><strong>latency</strong></td>
<td>milliseconds</td>
<td>milliseconds</td>
<td>seconds</td>
</tr>
<tr>
<td><strong>throughput</strong></td>
<td>100k+ records per node per second</td>
<td>10k+ records per node per second</td>
<td>100k+ records per node per second</td>
</tr>
<tr>
<td><strong>processing guarantees</strong></td>
<td>at-least-once delivery; support for exactly-once planned</td>
<td>at-least-once / exactly once (with Trident)</td>
<td>exactly once</td>
</tr>
<tr>
<td><strong>stateful operations</strong></td>
<td>yes</td>
<td>no / yes (with Trident)</td>
<td>yes</td>
</tr>
<tr>
<td><strong>language support</strong></td>
<td>++</td>
<td>+++</td>
<td>++</td>
</tr>
<tr>
<td><strong>community size/ committer &amp; user base</strong></td>
<td>+</td>
<td>+++</td>
<td>++</td>
</tr>
<tr>
<td><strong>special</strong></td>
<td>agile, state management, Kappa-native</td>
<td>distributed RPC</td>
<td>unified processing (batch, SQL, etc.)</td>
</tr>
</tbody>
</table>

---

© 2015 MapR Technologies
## When to use what?

<table>
<thead>
<tr>
<th>use case</th>
<th>Samza</th>
<th>Storm</th>
<th>Spark Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>filtering</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>counting (incl. aggregations)</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>joins</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>distributed RPC</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>re-processing (aka Kappa architecture)</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>materialized view maintenance (cache invalidation)</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Spark vs Storm: Throughput

- Spark Streaming: **670k** records/sec/node
- Storm: **115k** records/sec/node
- Commercial systems: **100-500k** records/sec/node
Comparison Resources

- http://www.slideshare.net/ChicagoHUG/yahoo-compares-storm-and-spark
- http://www.slideshare.net/ptgoetz/apache-storm-vs-spark-streaming
- http://samza.apache.org/learn/documentation/0.8/comparisons/storm.html
Free on-demand Hadoop training leading to certification

Start becoming an expert now
mapr.com/training
Q&A
Engage with us!

@MapR  maprtech

MapR Technologies  +MaprTechnologies

jscott@mapr.com  maprtech