Practical Big Data Processing
An Overview of Apache Flink

Tilmann Rabl
Berlin Big Data Center
www.dima.tu-berlin.de | bbdc.berlin | rabl@tu-berlin.de

With slides from Volker Markl and data artisans
What is Apache Flink?

Apache Flink is an open source platform for scalable batch and stream data processing.

- The core of Flink is a distributed streaming dataflow engine.
  - Executing dataflows in parallel on clusters
  - Providing a reliable foundation for various workloads
- **Data**Set and **DataStream** programming abstractions are the foundation for user programs and higher layers

http://flink.apache.org
What can I do with it?

A big data processing system that can **natively** support all these workloads.

- Batch processing
- Stream processing
- Machine Learning at scale
- Graph Analysis

Flink
Flink in the Analytics Ecosystem

Applications & Languages
- Hive
- Cascading
- Giraph
- Mahout
- Pig
- Crunch

Data processing engines
- MapReduce
- Spark
- Storm
- Tez
- Flink

App and resource management
- Yarn
- Mesos

Storage, streams
- HDFS
- HBase
- Kafka
- ...

...
Rich set of operators

Map, Reduce, Join, CoGroup, Union, Iterate, Delta Iterate, Filter, FlatMap, GroupReduce, Project, Aggregate, Distinct, Vertex-Update, Accumulators, ...

All you will ever need for any big data processing.
Sneak peak: Two of Flink’s APIs

```scala
case class Word (word: String, frequency: Int)

### DataSet API (batch):

```scala
def env = ExecutionEnvironment.getExecutionEnvironment()
val lines = env.readTextFile(...)
lines.flatMap {line => line.split(" ")
  .map(word => Word(word,1))
  .groupBy("word").sum("frequency")
  .print()```

### DataStream API (streaming):

```scala
def env = StreamExecutionEnvironment.getExecutionEnvironment()
val lines = env.fromSocketStream(...)
lines.flatMap {line => line.split(" ")
  .map(word => Word(word,1))
  .keyBy("word")
  .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))
  .sum("frequency")
  .print()
```
Flink Execution Model

- Flink program = DAG* of operators and intermediate streams
- Operator = computation + state
- Intermediate streams = logical stream of records
Architecture

• Hybrid MapReduce and MPP database runtime

• Pipelined/Streaming engine
  – Complete DAG deployed
Built-in vs. driver-based looping

Loop outside the system, in driver program

Iterative program looks like many independent jobs

Dataflows with feedback edges

System is iteration-aware, can optimize the job
Managed Memory

- Language APIs automatically converts objects to tuples
  - Tuples mapped to pages/buffers of bytes
  - Operators can work on pages/buffers
- Full control over memory, out-of-core enabled
- Operators (e.g., Hybrid Hash Join) address individual fields (not deserialize object): robust
Mini-Batching vs Native Streaming

Discretized Streams (D-Streams)

```
while (true) {
    // get next few records
    // issue batch computation
}
```

Native streaming

```
while (true) {
    // process next record
}
```

Long-standing operators
Problems of Mini-Batch

• Latency
  – Each mini-batch schedules a new job, loads user libraries, establishes DB connections, etc

• Programming model
  – Does not separate business logic from recovery – changing the mini-batch size changes query results

• Power
  – Keeping and updating state across mini-batches only possible by immutable computations
Flink’s Windowing

- Windows can be any combination of (multiple) triggers & evictions
  - Arbitrary tumbling, sliding, session, etc. windows can be constructed.

- Common triggers/evictions part of the API
  - Time (processing vs. event time), Count

- Even more flexibility: define your own UDF trigger/eviction

- Examples:
  ```
  dataStream.windowAll(TumblingEventTimeWindows.of(Time.seconds(5)));
  dataStream.keyBy(0).window(TumblingEventTimeWindows.of(Time.seconds(5)));
  ```
Yahoo! Benchmark Results

Performed by Yahoo! Engineering, Dec 16, 2015

[..]Storm 0.10.0, 0.11.0-SNAPSHOT and Flink 0.10.1 show sub-second latencies at relatively high throughputs[..]. Spark streaming 1.5.1 supports high throughputs, but at a relatively higher latency.

Flink achieves highest throughput with competitive low latency!
Our benchmarks

### Streaming

<table>
<thead>
<tr>
<th></th>
<th>2-node</th>
<th>4-node</th>
<th>8-node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storm</td>
<td>408K</td>
<td>696K</td>
<td>992K</td>
</tr>
<tr>
<td>Spark</td>
<td>379K</td>
<td>642K</td>
<td>912K</td>
</tr>
<tr>
<td>Flink</td>
<td>1230K</td>
<td>1260K</td>
<td>1260K</td>
</tr>
</tbody>
</table>

### Batch

<table>
<thead>
<tr>
<th></th>
<th>2-node</th>
<th>4-node</th>
<th>8-node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>365K</td>
<td>632K</td>
<td>947K</td>
</tr>
<tr>
<td>Flink</td>
<td>851K</td>
<td>1128K</td>
<td>1190K</td>
</tr>
</tbody>
</table>

Windowed Aggregations / Joins

Flink consistently outperforms other streaming engines in throughput and latency

Iterative Algorithms

---

**Figure 10:** k-means Strong Scaling experiments for Spark and Flink in 200 GB of generated data with 100 dimensions and $k=10$ clusters

**Figure 11:** k-means Production Scaling experiments for Spark and Flink on 30 nodes with $k=30$
Flink in the BBDC Stack

See details in the demos!
Thank You

Contact:
Tilmann Rabl
rabl@tu-berlin.de