Big Data Technology
Stream Processing with Storm

Eshcar Hillel
Yahoo!

Ronny Lempel
Outbrain
Roadmap

- Previous classes
  - Stream processing

- This class
  - Introduction to Storm: Topologies, Spouts and Bolts
  - Code examples
  - Architecture and fault tolerance
  - Spark streaming comparison
Apache Storm

http://hortonworks.com/hdp/
Historical Perspective

- GFS 2003
- MR 2004
- BigTable 2006
- Yahoo Hadoop 2006
- HBase 2007
- Storm 2011
- Apache Storm 2013
- Hadoop 2.0 2013
- Spark 2014
Motivation and Rational

- Real time big data analytics
  - Human scale (seconds not nanoseconds)
- Why real time?
  - Better user experience
    - Brings value to the companies
  - How frequently is data refreshed?
  - Better recommendation and prediction
  - Trend analysis
- Real time computation = processing pipeline = message processing paradigm
Introduction to Storm

- Stream processing paradigm
  - Scalable and Fault-Tolerant
- Platform to build arbitrary dataflow graphs
  - Not bound to a particular computing paradigm
- Tuple-centric execution engine
  - Simple primitives - developers need not care about how tuples are routed and tracked
  - Allows building powerful high-level abstractions
## Comparison with MR

<table>
<thead>
<tr>
<th>Storm</th>
<th>MapReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream processing</td>
<td>Batch processing</td>
</tr>
<tr>
<td>Real time analytics</td>
<td>Offline analytics</td>
</tr>
<tr>
<td>Run topologies (DAG), spout and bolts</td>
<td>Run MR jobs, map and reduce tasks</td>
</tr>
<tr>
<td>Processes messages forever</td>
<td>Eventually finishes</td>
</tr>
<tr>
<td>Handles tuples</td>
<td>Handles key-value pairs</td>
</tr>
<tr>
<td>Nimbus-supervisors arch</td>
<td>NN/RM-DN/NM arch</td>
</tr>
</tbody>
</table>
Storm Use Cases

- Stream processing
  - Processing messages and updating database (incremental indexing)
- Continuous computation
  - Continuous query on data streams (frequent items in a stream)
- Parallelizing intense queries
  - (search queries)
**Data Model**

- **Tuples**
  - Core unit of data
  - Immutable set of values

- **Stream**
  - Unbounded sequence of tuples

- **Topology**
  - Execution graph, data flow
  - DAG of spouts and bolts

- **Spout**
  - Stream source

- **Bolt**
  - Stream processor
Data Model

- Spouts and Bolts continuously emit tuples
- Parallel, Pipelined Execution
  - Multiple tuples processed concurrently
  - Each node execute in parallel as many tasks
- Stream grouping defines where to send the tuple
Word Count Topology

```java
TopologyBuilder builder =
    new TopologyBuilder();

builder.setSpout("sentences",
    new RandomSentence (), 5);

builder.setBolt("split",
    new SplitSentence(), 8)
    .shuffleGrouping("sentences");

builder.setBolt("count",
    new WordCount(), 12)
    .fieldsGrouping("split",
    new Fields("word"));
```

Source: [https://github.com/apache/storm/tree/v1.0.1/examples/storm-starter](https://github.com/apache/storm/tree/v1.0.1/examples/storm-starter)
Example: Spout Implementation

```java
public class RandomSentence extends BaseRichSpout {

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("sentence"));
    }

    public void nextTuple() {
        Utils.sleep(100);
        String[] sentences = new String[] {
            "the cow jumped over the moon",
            "an apple a day keeps the doctor away",
            ...
        };

        String sentence = sentences[_rand.nextInt(sentences.length)];
        _collector.emit(new Values(sentence));
    }
}
```
Example: Bolt Implementation

```java
public static class WordCount extends BaseBasicBolt {

    private Map<String, Integer> counts = new HashMap<String, Integer>();

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }

    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.get(word);
        if(count==null) count = 0;
        count++;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }
}
```
Storm Architecture

Nimbus

ZK

Supervisor

Task

Task

Task

Executor thread
Worker process (T1)

Worker process (T2)

ZeroMQ / Netty

Schedule

Monitor

Task

Task

Task

Executor thread
Worker process (T1)

Worker process (T3)

ZeroMQ / Netty
### Storm and YARN

<table>
<thead>
<tr>
<th>Storm</th>
<th>YARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workers spawn topology tasks</td>
<td>Topology tasks run in containers</td>
</tr>
<tr>
<td>Supervisors manage processes on node</td>
<td>Supervisors run as application master</td>
</tr>
<tr>
<td>Nimbus responsible for assignment and scheduling</td>
<td>Nimbus negotiates with RM</td>
</tr>
</tbody>
</table>
Components Responsibilities

- Nimbus calculates assignment of submitted topology and sends through ZK
- Supervisors download topology from nimbus
- Supervisors start workers (JVM processes)
- Workers heartbeat back to supervisors and nimbus via ZK
- Fault tolerance
  - on worker failure – supervisor restarts worker
  - worker fails repeatedly – nimbus reassigns
  - supervisor failure – nimbus reassigns
  - Nimbus failure – no reassignments
Guaranteed Message Processing

- No data lose - we want to process each tuple
- Each tuple generates a virtual computation tree
  - Something goes wrong at a node → Replay it
  - How can we tell something went wrong?
- Tuple **anchoring**
  - Reference a tuple on the tuple it depends on
  - Explicitly ACK the tuple’s consumption
  - Failed anchored events are replayed
Tuple Execution (Multicast) Tree

Word-count Example

[“the cow has jumped over the moon”]

[“the”] → [“the”,1]
[“cow”] → [“cow”,1]
[“has”] → [“has”,1]
[“jumped”] → [“jumped”,1]
[“over”] → [“over”,1]
[“the”] → [“the”,2]
[“moon”] → [“moon”,1]
Guaranteed Processing by Anchoring

```java
public class SplitSentence extends BaseRichBolt {

    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String sentence = tuple.getString(0);
        for(String word: sentence.split(" ")) {

            // Anchored tuple
            collector.emit(tuple, new Values(word));
            // Previous example – unanchored tuple
            // collector.emit(new Values(word));
        }
        collector.ack(tuple);
    }
}
```

- ACK implementation is highly optimized
How many users have seen this URL?
Example: Batch Bolt

```java
public static class PartialUniquer extends BaseBatchBolt {

    private BatchOutputCollector _collector;
    private Object _id;
    private Set<String> _followers = new HashSet<String>();

    public void prepare(..., BatchOutputCollector collector, Object id) {
        _collector = collector;
        _id = id;
    }

    public void execute(Tuple tuple) {
        _followers.add(tuple.getString(1));
    }

    public void finishBatch() {
        _collector.emit(new Values(_id, _followers.size()));
    }
}
```
Spark Streaming

Extension of the core Spark API divides the data into batches, processed by spark engine, generates stream of results in batches

WordCount Example

val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream(<hostname>,<port>)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word=>(word,1))
val wordCount = pairs.reduceByKey(_+_)
ssc.start()
ssc.awaitTermination()
Discretized Streams (DStreams)

High-level abstraction of a continuous stream Represented as a sequence of RDDs
Storm vs. Spark

- Storm is a stream processing framework that also does micro-batching (Trident)
  - core API, one-at-a-time, lower latency
  - micro-batch, higher throughput

- Spark is a batch processing framework that also does micro-batching (Spark Streaming)

Source: https://www.slideshare.net/ptgoetz/apache-storm-vs-spark-streaming
Summary

- **Storm**
  - Processing Model (Spouts and Bolts)
  - Arbitrary dataflow topologies
  - Data Parallelism and Pipelining
  - Grouping: control of message routing
  - Coordination mechanisms

- **Spark Streaming**
  - mini-batches
Further Reading

- **Storm Project Home**
  - http://storm-project.net/

- **Storm Tutorial**
  - https://github.com/nathanmarz/storm/wiki/Tutorial