Realtime Processing with Storm
Who are we?

Dario Simonassi
@ldsimonassi

Gabriel Eisbruch
@geisbruch

Jonathan Leibiusky
@xetorthio
Agenda

● The problem
● Storm - Basic concepts
● Trident
● Deploying
● Some other features
● Contributions
Tons of information arriving every second
Heavy workload

And you think you're overworked...
CHALLENGE ACCEPTED
What is Storm?

- Is a highly distributed realtime computation system.
- Provides general primitives to do realtime computation.
- Can be used with any programming language.
- It's scalable and fault-tolerant.
Example - Main Concepts

https://github.com/geisbruch/strata2012
"Count tweets related to strataconf and show hashtag totals."
Main Concepts - Spout

- First thing we need is to read all tweets related to strataconf from twitter.
- A Spout is a special component that is a source of messages to be processed.
public class ApiStreamingSpout extends BaseRichSpout {
    SpoutOutputCollector collector;
    TwitterReader reader;

    public void nextTuple() {
        Tweet tweet = reader.getNextTweet();
        if(tweet != null)
            collector.emit(new Values(tweet));
    }

    public void open(Map conf, TopologyContext context, 
                     SpoutOutputCollector collector) {
        reader = new TwitterReader(conf.get("track"), conf.get("user"), conf.get("pass"));
        this.collector = collector;
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("tweet"));
    }
}
Main Concepts - Bolt

- For every tweet, we find hashtags
- A Bolt is a component that does the processing.
public class HashtagsSplitter extends BaseBasicBolt {
    public void execute(Tuple input, BasicOutputCollector collector) {
        Tweet tweet = (Tweet) input.getValueByField("tweet");
        for (String hashtag : tweet.getHashTags()) {
            collector.emit(new Values(hashtag));
        }
    }
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("hashtag"));
    }
}
public class HashtagsCounterBolt extends BaseBasicBolt {
    private Jedis jedis;

    public void execute(Tuple input, BasicOutputCollector collector) {
        String hashtag = input.getStringByField("hashtag");
        jedis.hincrBy("hashs", hashtag, 1);
    }

    public void prepare(Map conf, TopologyContext context) {
        jedis = new Jedis("localhost");
    }
}
```java
public static Topology createTopology() {
    TopologyBuilder builder = new TopologyBuilder();

    builder.setSpout("tweets-collector", new ApiStreamingSpout(), 1);
    builder.setBolt("hashtags-splitter", new HashTagsSplitter(), 2)
        .shuffleGrouping("tweets-collector");
    builder.setBolt("hashtags-counter", new HashtagsCounterBolt(), 2)
        .fieldsGrouping("hashtags-splitter", new Fields("hashtags"));

    return builder.createTopology();
}
```
Example Overview - Read Tweet

This tweet has #hello #world hashtags!!!
Example Overview - Split Tweet

This tweet has #hello #world hashtags!!!

Spout -> Splitter

Splitter: #hello -> Counter
#world -> Redis

Redis

Diagram showing the flow from Spout to Splitter, then to a counter and Redis.
Example Overview - Split Tweet

This tweet has #hello #world hashtags!!

#hello
incr(#hello)

#world
incr(#world)

#hello = 1
#world = 1
Example Overview - Split Tweet

This tweet has #bye #world hashtags

#hello = 1
#world = 1
This tweet has #bye #world hashtags
Example Overview - Split Tweet

This tweet has #bye #world hashtags

#hello = 1
#world = 2
#bye = 1
Main Concepts - In summary

- Topology
- Spout
- Stream
- Bolt
Guaranteeing Message Processing
Guaranteeing Message Processing

- Spout
- HashtagsSplitter
- HashtagsCounterBolt
- Redis
Guaranteeing Message Processing

[tweetstream]

Spout

HashtagsSplitter

HashtagsSplitter

HashtagsCounterBolt

HashtagsCounterBolt

 collector.ack(tuple);
Guaranteeing Message Processing
Guaranteeing Message Processing

- Spout
- HashtagsSplitter
- HashtagsCounterBolt
- Redis
void fail(Object msgId);
Trident - What is Trident?

- High-level abstraction on top of Storm.
- Will make it easier to build topologies.
- Similar to Hadoop, Pig
Trident - Topology

TridentTopology topology = new TridentTopology();
Stream tweets = topology
    .newStream("tweets", new ApiStreamingSpout());
Trident

Stream hashtagsStream =
tweetsStrem.each(new Fields("tweet"), new BaseFunction() {
    public void execute(TridentTuple tuple, TridentCollector collector) {
        Tweet tweet = (Tweet)tuple.getValueByField("tweet");
        for (String hashtag : tweet.getHashtags())
            collector.emit(new Values(hashtag));
    }
}, new Fields("hashtag")
);
Trident - States

TridentState hashtagCounts = hashtagsStream.groupBy(new Fields("hashtag"))
.persistentAggregate(new MemoryMapState.Factory(),
    new Count(),
    new Fields("count"));
TridentState hashtagCounts =
hashtagsStream.groupBy(new Fields("hashtag"))
.persistentAggregate(new MemoryMapState.Factory(),
new Count(),
new Fields("count"));
Trident - States

TridentState hashtagCounts = hashtagsStream.groupBy(new Fields("hashtag"))
.persistentAggregate(new MemoryMapState.Factory(),
    new Count(),
    new Fields("count"));
Trident - States

TridentState hashtagCounts = hashtagsStream.groupBy(new Fields("hashtag"))
  .persistentAggregate(new MemoryMapState.Factory(),
    new Count(),
    new Fields("count"));
TridentState hashtags = topology
   .newStream("tweets", new ApiStreamingSpout())
   .each(new Fields("tweet"), new HashTagSplitter(), new Fields("hashtag"))
   .groupBy(new Fields("hashtag"))
   .persistentAggregate(new MemoryMapState.Factory(),
                      new Count(),
                      new Fields("count"))
Transactions

By Frits Ahlefeldt
Transactions

- Exactly once message processing semantic
Transactions

- The intermediate processing may be executed in a parallel way
Transactions

- But persistence should be strictly sequential
Transactions

- Spouts and persistence should be designed to be transactional
We want to know **on-line** the count of tweets with hashtag *hadoop* and the amount of tweets with *strataconf* hashtag.
Trident - Query

DRPCClient drpc = new DRPCClient("MyDRPCServer", "MyDRPCPort");

drpc.execute("counts", "strataconf hadoop");
Stream drpcStream =
    topology.newDRPCStream("counts", drpc)
Trident - Query

Stream singleHashtag =
  drpcStream.each(new Fields("args"), new Split(), new Fields("hashtag"))
Trident - Query

Stream hashtagsCountQuery =
    singleHashtag.stateQuery(hashtagCountMemoryState, new Fields("hashtag"),
    new MapGet(), new Fields("count"));
Trident - Query

Stream drpcCountAggregate =
  hashtagsCountQuery.groupBy(new Fields("hashtag"))
  .aggregate(new Fields("count"), new Sum(), new Fields("sum"));
topology.newDRPCStream("counts", drpc)
  .each(new Fields("args"), new Split(), new Fields("hashtag"))
  .stateQuery(hashtags, new Fields("hashtag"),
              new MapGet(), new Fields("count"))
  .each(new Fields("count"), new FilterNull())
  .groupBy(new Fields("hashtag"))
  .aggregate(new Fields("count"), new Sum(), new Fields("sum"));
Trident - Query - Complete Example
Trident - Query - Complete Example
Trident - Query - Complete Example
Trident - Query - Complete Example

Request continue held

DRPC Server

Hashtag Splitter

Hashtag Count Query

Aggregate Count

DRPC stream

On finish return to DRPC server
Trident - Query - Complete Example

DRPC stream

Hashtag Splitter

Hashtag Count Query

Aggregate Count

Return response to the client
Running Topologies
Running Topologies - Local Cluster

```java
Config conf = new Config();
conf.put("some_property","some_value")
LocalCluster cluster = new LocalCluster();
cluster.submitTopology("twitter-hashtag-summarizer", conf, builder.createTopology());
```

- The whole topology will run in a single machine.
- Similar to run in a production cluster
- Very Useful for test and development
- Configurable parallelism
- Similar methods than StormSubmitter
package twitter.streaming;
public class Topology {
    public static void main(String[] args) {
        StormSubmitter.submitTopology("twitter-hashtag-summarizer", conf, builder.createTopology());
    }
}

mvn package

storm jar Strata2012-0.0.1.jar twitter.streaming.Topology \
"$REDIS_HOST" $REDIS_PORT "strata,strataconf" $TWITTER_USER $TWITTER_PASS
Running Topologies - Cluster Layout

- Single Master process
- No SPOF
- Automatic Task distribution
- Amount of task configurable by process
- Automatic tasks re-distribution on supervisors fails
- Work continue on nimbus fails

Zookeeper Cluster
(Only for coordination and cluster state)

Storm Cluster

Nimbus
(The master Process)

Supervisor
(where the processing is done)

Supervisor
(where the processing is done)

Supervisor
(where the processing is done)
Running Topologies - Cluster Architecture - No SPOF

Storm Cluster

Nimbus
(The master Process)

Supervisor
(where the processing is done)

Supervisor
(where the processing is done)

Supervisor
(where the processing is done)
Running Topologies - Cluster Architecture - No SPOF

Storm Cluster

Supervisor
(where the processing is done)

Supervisor
(where the processing is done)

Supervisor
(where the processing is done)
Running Topologies - Cluster Architecture - No SPOF

**Storm Cluster**

- **Nimbus** (The master Process)
- **Supervisor** (where the processing is done)
Other features
Other features - Non JVM languages

- Use Non-JVM Languages
- Simple protocol
- Use stdin & stdout for communication
- Many implementations and abstraction layers done by the community

```json
{
  "conf": {
    "topology.message.timeout.secs": 3,
    // etc
  },
  "context": {
    "task->component": {
      "1": "example-spout",
      "2": "__acker",
      "3": "example-bolt"
    },
    "taskid": 3
  },
  "pidDir": "..."
}
```
**Contribution**

- A collection of community developed spouts, bolts, serializers, and other goodies.
- You can find it in: [https://github.com/nathanmarz/storm-contrib](https://github.com/nathanmarz/storm-contrib)

- Some of those goodies are:
  - **cassandra**: Integrates Storm and Cassandra by writing tuples to a Cassandra Column Family.
  - **mongodb**: Provides a spout to read documents from mongo, and a bolt to write documents to mongo.
  - **SQS**: Provides a spout to consume messages from Amazon SQS.
  - **hbase**: Provides a spout and a bolt to read and write from HBase.
  - **kafka**: Provides a spout to read messages from kafka.
  - **rdbms**: Provides a bolt to write tuples to a table.
  - **redis-pubsub**: Provides a spout to read messages from redis pubsub.
  - And more...
Questions?