Part 9: Real-Time Data Analytics (1/2)
March 27, 2018

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2018w/
This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See http://creativecommons.org/licenses/by-nc-sa/3.0/us/ for details
My data is a day old…

Meh.
Twitter’s data warehousing architecture
Struggling with complex data? Head to Data Science 2/20 to retii... 

*Clinton Paquin* @clintonpaquin

Simply stated, "The only pro muscle memory" @TheChang

*Brad Anderson* @boorad

Followed by *Florian Leibert*...

*Sheila Morrissey* @sheilaMorr

Popular accounts · Find friends

*The Hill* @thehill · 1h

Republicans take debt ceiling... 

*Popehat* @Popehat · 10h

In a world in which few things... 

*CNN Breaking News* @cnnbrk · 1h

Ukrainian Pres. says he has begun work on key...
Case Study: Steve Jobs passes away
Initial Implementation

Algorithm: Co-occurrences within query sessions
Implementation: Pig scripts over query logs on HDFS

Problem: Query suggestions were several hours old!

Why?
  Log collection lag
  Hadoop scheduling lag
  Hadoop job latencies

We need real-time processing!
Solution?

Can we do better than one-off custom systems?
Stream Processing Frameworks

Source: Wikipedia (River)
real-time vs. online vs. streaming
What is a data stream?

Sequence of items:

Structured (e.g., tuples)
Ordered (implicitly or timestamped)
Arriving continuously at high volumes
Sometimes not possible to store entirely
Sometimes not possible to even examine all items
Applications

Network traffic monitoring
Datacenter telemetry monitoring
Sensor networks monitoring
Credit card fraud detection
Stock market analysis
Online mining of click streams
Monitoring social media streams
What exactly do you do?

“Standard” relational operations:

- Select
- Project
- Transform (i.e., apply custom UDF)
- Group by
- Join
- Aggregations

What else do you need to make this “work”?
Issues of Semantics

Group by… aggregate
When do you stop grouping and start aggregating?

Joining a stream and a static source
Simple lookup

Joining two streams
How long do you wait for the join key in the other stream?

Joining two streams, group by and aggregation
When do you stop joining?

What’s the solution?
Windows

Windows restrict processing scope:
- Windows based on ordering attributes (e.g., time)
- Windows based on item (record) counts
- Windows based on explicit markers (e.g., punctuations)
Windows on Ordering Attributes

Assumes the existence of an attribute that defines the order of stream elements (e.g., time)

Let $T$ be the window size in units of the ordering attribute
Windows on Counts

Window of size N elements (sliding, tumbling) over the stream
Windows from “Punctuations”

Application-inserted “end-of-processing”
Example: stream of actions… “end of user session”

Properties

Advantage: application-controlled semantics
Disadvantage: unpredictable window size (too large or too small)
Streams Processing Challenges

Inherent challenges

Latency requirements
Space bounds

System challenges

Bursty behavior and load balancing
Out-of-order message delivery and non-determinism
Consistency semantics (at most once, exactly once, at least once)
Stream Processing Frameworks

Source: Wikipedia (River)
How do consumers get data from producers?
Producer/Consumers

Producer pushes e.g., callback
Producer/Consumers

Producer

Consumer pulls
e.g., poll, tail

Consumer
Producer/Consumers

Producer

Consumer

Consumer

Consumer

Consumer
Producer/Consumers

Producer

Producer

Kafka

Broker

Consumer

Consumer

Consumer

Consumer
Stream Processing Frameworks

Storm/Heron

Source: Wikipedia (River)
Storm/Heron

Storm: real-time distributed stream processing system
   Started at BackType
   BackType acquired by Twitter in 2011
   Now an Apache project

Heron: API compatible re-implementation of Storm
   Introduced by Twitter in 2015
   Open-sourced in 2016
Want real-time stream processing? I got your back.

I’ve got the most intuitive implementation: a computation graph!
Topologies

Storm topologies = “job”
Once started, runs continuously until killed

A topology is a computation graph
Graph contains vertices and edges
Vertices hold processing logic
Directed edges indicate communication between vertices

Processing semantics
At most once: without acknowledgments
At least once: with acknowledgements
Spouts and Bolts: Logical Plan

Components

- Tuples: data that flow through the topology
- Spouts: responsible for emitting tuples
- Bolts: responsible for processing tuples
Spouts and Bolts: Physical Plan

Physical plan specifies execution details
Parallelism: how many instances of bolts and spouts to run
Placement of bolts/spouts on machines
...
Stream Groupings

Bolts are executed by multiple instances in parallel
User-specified as part of the topology

When a bolt emits a tuple, where should it go?
Answer: Grouping strategy
Shuffle grouping: randomly to different instances
Field grouping: based on a field in the tuple
Global grouping: to only a single instance
All grouping: to every instance
Heron Architecture

Source: https://blog.twitter.com/2015/flying-faster-with-twitter-heron
Heron Architecture

Stream Manager
Manages routing tuples between spouts and bolts
Responsible for applying backpressure
Some me some code!

```java
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("word", new WordSpout(), parallelism);
builder.setBolt("consumer", new ConsumerBolt(), parallelism)
    .fieldsGrouping("word", new Fields("word"));

Config conf = new Config();
// Set config here
// ...

StormSubmitter.submitTopology("my topology", conf,
    builder.createTopology());
```
public static class WordSpout extends BaseRichSpout {
    @Override
    public void declareOutputFields(
        OutputFieldsDeclarer outputFieldsDeclarer) {
        outputFieldsDeclarer.declare(new Fields("word"));
    }

    @Override
    public void nextTuple() {
        // ...
        collector.emit(word);
    }
}
public static class ConsumerBolt extends BaseRichBolt {
    private OutputCollector collector;
    private Map<String, Integer> countMap;

    public void prepare(Map map, TopologyContext topologyContext, OutputCollector outputCollector) {
        collector = outputCollector;
        countMap = new HashMap<String, Integer>();
    }

    @Override
    public void execute(Tuple tuple) {
        String key = tuple.getString(0);
        if (countMap.get(key) == null) {
            countMap.put(key, 1);
        } else {
            Integer val = countMap.get(key);
            countMap.put(key, ++val);
        }
    }
}
Spark Streaming

Stream Processing Frameworks

Source: Wikipedia (River)
Want real-time stream processing? I got your back.

I’ve got the most intuitive implementation: a computation graph!

Hmm, I gotta get in on this streaming thing…

But I got all this batch processing framework that I gotta lug around.

I know: we’ll just chop the stream into little pieces, pretend each is an RDD, and we’re on our merry way!
Spark Streaming: Discretized Streams

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

Chop up the stream into batches of $X$ seconds

Process as RDDs!

Return results in batches

Source: All following Spark Streaming slides by Tathagata Das
Example: Get hashtags from Twitter

```scala
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
```

DStream: a sequence of RDD representing a stream of data

Twitter Streaming API

DStream

tweets DStream

stored in memory as an RDD (immutable, distributed)
Example: Get hashtags from Twitter

val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
Example: Get hashtags from Twitter

```scala
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

output operation: to push data to external storage

tweets DStream

hashTags DStream

every batch saved to HDFS
Fault Tolerance

Bottom line: they’re just RDDs!
Fault Tolerance

Bottom line: they’re just RDDs!

tweets RDD

hashTags RDD

flatMap

input data replicated in memory

lost partitions recomputed on other workers
Key Concepts

DStream – sequence of RDDs representing a stream of data
  Twitter, HDFS, Kafka, Flume, ZeroMQ, Akka Actor, TCP sockets

Transformations – modify data from on DStream to another
  Standard RDD operations – map, countByValue, reduce, join, …
  Stateful operations – window, countByValueAndWindow, …

Output Operations – send data to external entity
  saveAsHadoopFiles – saves to HDFS
  foreach – do anything with each batch of results
Example: Count the hashtags

```scala
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.countByValue()
```

![Diagram showing the flow of data through flatMap, map, reduceByKey, and batch operations to count hashtags.](image-url)
Example: Count the hashtags over last 10 mins

```scala
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```
Example: Count the hashtags over last 10 mins

val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
val tagCounts = hashtags.countByValueAndWindow(Minutes(10), Seconds(1))
**Smart window-based reduce**

Incremental counting generalizes to many reduce operations
Need a function to “inverse reduce” (“subtract” for counting)

```scala
val tagCounts = hashtags
  .countByValueAndWindow(Minutes(10), Seconds(1))

val tagCounts = hashtags
  .reduceByKeyAndWindow(_ + _, _ - _, Minutes(10), Seconds(1))
```
What’s the problem?

event time vs. processing time
Apache Beam

Stream Processing Frameworks

Source: Wikipedia (River)
Apache Beam

2013: Google publishes paper about MillWheel

2015: Google releases Cloud Dataflow

2016: Google donates API and SDK to Apache to become Apache Beam
Programming Model

Core Concepts

- **Pipeline**: a data processing task
- **PCollection**: a distributed dataset that a pipeline operates on
- **Transform**: a data processing operation
  - **Source**: for reading data
  - **Sink**: for writing data

Processing semantics: exactly once
Looks a lot like Spark!

Pipeline p = Pipeline.create(options);

p.apply(TextIO.Read.from("gs://your/input/"))

  .apply(FlatMapElements.via((String word) ->
      Arrays.asList(word.split("[^a-zA-Z]+")))))
  .apply(Filter.by((String word) -> !word.isEmpty()))
  .apply(Count.perElement())
  .apply(MapElements.via((KV<String, Long> wordCount) ->
      wordCount.getKey() + " : " + wordCount.getValue())
  .apply(TextIO.Write.to("gs://your/output/"));
The Beam Model

**What** results are computed?

**Where** in event time are the results computed?

**When** in processing time are the results materialized?

**How** do refinements of results relate?
Event Time vs. Processing Time

What’s the distinction?

Watermark: System’s notion when all data in a window is expected to arrive

Where in event time are the results computed?

When in processing time are the results materialized?

How do refinements of results relate?

Trigger: a mechanism for declaring when output of a window should be materialized

Default trigger “fires” at watermark

Late and early firings: multiple “panes” per window
Event Time vs. Processing Time
What’s the distinction?

Watermark: System’s notion when all data in a window is expected to arrive

Where in event time are the results computed?

When in processing time are the results materialized?

How do refinements of results relate?

How do multiple “firings” of a window (i.e., multiple “panes”) relate?

Options: Discarding, Accumulating, Accumulating & retracting
Pipeline p = Pipeline.create(options);

p.apply(TextIO.Read.from("gs://your/input/"))

.apply(FlatMapElements.via((String word) ->
   Arrays.asList(word.split("[^a-zA-Z]+"))))
 .apply(Filter.by((String word) -> !word.isEmpty()))
 .apply(Count.perElement())
 .apply(MapElements.via((KV<String, Long> wordCount) ->
   wordCount.getKey() + " : " + wordCount.getValue()))
 .apply(TextIO.Write.to("gs://your/output/"));
Word Count
With windowing...

Pipeline p = Pipeline.create(options);

p.apply(KafkaIO.read("tweets")
  .withTimestampFn(new TweetTimestampFunction())
  .withWatermarkFn(kv ->
      Instant.now().minus(Duration.standardMinutes(2))))
  .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2)))
    .triggering(AtWatermark()
      .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
      .withLateFirings(AtCount(1)))
    .accumulatingAndRetractingFiredPanes())
  .apply(FlatMapElements.via((String word) ->
      Arrays.asList(word.split("[^a-zA-Z']"))))
  .apply(Filter.by((String word) -> !word.isEmpty()))
  .apply(Count.perElement())
  .apply(KafkaIO.write("counts"))

Where in event time?
When in processing time?
How do refines relate?
Questions?