How to build leakproof stream processing pipelines with Apache Kafka and Apache Spark
Introduction

- Guru Medasani
  - Data Science Architect at Domino Data Lab
  - Previously senior solutions architect at Cloudera

- Jordan Hambleton - Consulting Manager in San Francisco
  - Nearly 4 years as Resident Senior Architect at large technology firm
  - Previously software engineer building operational data systems on CDH
Agenda

- Intro
- Overview of Spark Streaming from Kafka
  - Workflow of the DStream and RDD
  - Spark Streaming Kafka consumer types
- Offset management
  - Motivation
  - Storing offsets in external data stores
- Q & A
Overview

Kafka Cluster

Topic A

Hadoop / YARN Cluster

server

partition

1

142

143

144

... 

server

partition

2

121

122

123

... 

server

partition

3

129

130

131

... 

server

partition

n

137

138

139

... 

executor

1

executor

2

executor

3

... 

executor

n

more parallelism
Overview Spark Streaming from Kafka

- **DStream** - sequence of RDDs
- **Two approaches in KafkaUtils**
  - Receiver based
  - Direct approach (recommended & the method we talk about)
- **Spark streaming embeds a kafka client**
  - Spark 1.6 uses the 0.9.0-kafka-2.0.0 client (SimpleConsumer)
  - Spark 2.x kafka 0-8-0 uses the 0.9.0-kafka-2.0.2 client (SimpleConsumer)
  - Spark 2.x kafka 0-10-0 uses the 0.10.0-kafka-2.1.0 client (KafkaConsumer)
DStream and RDD Workflow

- Spark Streaming
  - batchIntervalInSeconds
  - stopGracefullyOnShutdown

- Kafka
  - bootstrap.servers
  - auto.offset.reset
  - group.id
  - key.deserializer
  - value.deserializer

New Direct Kafka integration w/o Receivers and WALs

1. Query latest offsets and decide offset ranges for batch
2. Launch jobs using offset ranges
3. Read data using offset ranges in jobs
Spark Streaming Kafka Consumer # 1

- spark-streaming-kafka-0-8 / 0.9.0-kafka-2.0.2
- DStream
  - Gets range of each topic/partition - throttle maxRatePerPartition
  - auto.offset.reset (smallest|largest)
  - refresh.leader.backoff.ms - lost leader
- KafkaRDD for set of topic, partition, offsets
  - User can now get offset ranges from RDD
    - topic, partition, fromOffset (inclusive), untilOffset (exclusive)
- KafkaRDDPartition iterator
  - SimpleConsumer initialized and batches of events fetched
  - refresh.leader.backoff.ms - lost leader
Spark Streaming Kafka Consumer # 2

- **Supported** - spark-streaming-kafka-0-10 / 0.10.0-kafka-2.1.0
- Internal Kafka client uses new Java KafkaConsumer
- ConsumerStrategies
  - subscribe, assign, subscribe pattern
- LocationStrategies
  - executor distribution strategy (consistent, fixed, brokers)
- DStream
  - Gets range of each topic/partition - throttle maxRatePerPartition
  - auto.offset.reset (earliest|latest)
  - Be careful - enable.auto.commit (default true)
  - heartbeat & session timeouts
Spark Streaming Kafka Consumer # 2

- **DStream**
  - Consumer poll for group coordination & discovery
  - Identify new partitions, from offsets
  - Pause consumer
  - `seekToEnd` to get `untilOffsets`

- **KafkaRDD**
  - Fixed `[enable.auto.commit = false, auto.offset.reset = none, spark-executor-${group.id}]`
  - Attempts to assign offset range consistently for optimal consumer caching

- **KafkaRDDPartition iterator**
  - Initialize/lookup `CachedKafkaConsumer` with executor group
    - consumer assigned per single topic, partition with internal buffer
    - on cache miss, seek and poll
Keeping Track
Motivation for Tracking Offsets

- Planned Maintenance
  - Upgrades
  - Bug-fixes
- Unplanned Maintenance
  - Failures
- Application Processing Errors
  - Wrong calculations
  - Updated algorithm over known streaming data
- More control over messages
  - Just earliest and latest are insufficient
Obtaining Offsets

- Cast RDD to HasOffsetRanges
- DStream’s first transformation
Offset management Workflow

- Limited options prior to spark-streaming-kafka-0-10
- Store offsets in external datastore
  - Checkpoints (Not recommended)
  - ZooKeeper
  - Kafka
  - HBase
- Do not have to manage offsets
Offset Management in ZooKeeper

- ZooKeeper
  - znode - /consumers/[groupId]/offsets/[topic]/[partitionId] -> long (offset)
  - Only retains latest committed offsets
  - Can easily be managed by external tools
  - Leverage existing monitoring for Lag, no historical insight
Offset Management in Kafka

- Kafka
  - CanCommitOffsets provides async commit to internal kafka topic
  - More difficult to manage internal kafka topic manually
  - Leverage existing monitoring for Lag, no historical insight
Offset Management in HBase

- HBase
  - Unique entry per consumer group, batch

- Fine-grained monitoring over time
- HBase shell for easy management
- Get latest entry -
  - scan 'prod_stream',
  - STARTROW =>'device_alerts:csi_group',
  - REVERSED =>TRUE,
  - LIMIT =>1

| schema: | <TOPIC_NAME>:<GROUP_ID>:<EPOCH_BATCHTIME_MS> |
| row:    | <TOPIC_NAME>:<GROUP_ID>:<EPOCH_BATCHTIME_MS> |
| column family: | offsets |
| qualifier: | <PARTITION_ID> |
| value:   | <OFFSET_ID> |

```
Offset Management in HBase

- HBase
  - Unique entry per consumer group, batch

- Fine-grained monitoring over time
- HBase shell for easy management
- Get latest entry -
  - scan 'prod_stream',
  - STARTROW =>'device_alerts:csi_group',
  - REVERSED =>TRUE,
  - LIMIT =>1

| schema: | <TOPIC_NAME>:<GROUP_ID>:<EPOCH_BATCHTIME_MS> |
| row:    | <TOPIC_NAME>:<GROUP_ID>:<EPOCH_BATCHTIME_MS> |
| column family: | offsets |
| qualifier: | <PARTITION_ID> |
| value:   | <OFFSET_ID> |

```
Starting Streaming Jobs with Known Offsets

- Spark Streaming job started for the first time
- No changes in Kafka partitions
- Increase in number of Kafka partitions

Questions?

Thank you
Jordan Hambleton
Guru Medasani