Spark and Cassandra

Solving classical data analytic task by using modern distributed databases

Artem Aliev
DataStax
Let us know what you think

Click ‘engage’ to rate sessions and ask questions

Join the conversation #gotocph
Spark and Cassandra

Solving classical data analytic task by using modern distributed databases

Artem Aliev
Software Developer
Agenda: Lambda Architecture

http://lambda-architecture.net
Apache Cassandra

Apache Cassandra™ is a massively scalable NoSQL OLTP database.

• Masterless architecture with read/write anywhere design.
• Continuous availability with no single point of failure.
• Multi-data center and cloud availability zone support.
• Linear scale performance with online capacity expansion.
• CQL – SQL-like language.
Cassandra was designed with the understanding that system/hardware failures can and do occur
- Peer-to-peer, distributed system
- All nodes the same
- Multi data centre support out of the box
- Configurable replication factor
- Configurable data consistency per request
- Active-Active replication architecture
Cassandra Query Language

CREATE TABLE sporty_league (  
  team_name    varchar,  
  player_name  varchar,  
  jersey       int,  
  PRIMARY KEY (team_name, player_name)  
);

SELECT * FROM sporty_league WHERE team_name = 'Mighty Mutts'  
and player_name = 'Lucky';

INSERT INTO sporty_league (team_name, player_name, jersey)  
VALUES ('Mighty Mutts','Felix',90);
Apache Spark

- Distributed computing framework
- Created by UC AMP Lab since 2009
- Apache Project since 2010
- Solves problems Hadoop is bad at
  - Iterative Algorithms
  - Interactive Machine Learning
  - More general purpose than MapReduce
- Streaming!
- In memory!

Spark
Lightning-Fast Cluster Computing
Fast

* Logistic Regression Performance

![Bar chart showing logistic regression performance with Hadoop and Spark.](chart)

- **Hadoop**: 110 sec / iteration
- **Spark**: first iteration 80 sec, further iterations 1 sec
1. file = spark.textFile("hdfs://...")
2. counts = file.flatMap(lambda line: line.split(" "))
   .map(lambda word: (word, 1))
   .reduceByKey(lambda a, b: a + b)
3. counts.saveAsTextFile("hdfs://...")
A Quick Comparison to Hadoop

Hadoop

- HDFS
- map()
- reduce()
- map()
- reduce()

Spark

- Data Source 1
- Data Source 2
- map()
- join()
- cache()
- transform()
- transform()
<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>reduce</td>
<td>sample</td>
</tr>
<tr>
<td>filter</td>
<td>count</td>
<td>take</td>
</tr>
<tr>
<td>groupBy</td>
<td>fold</td>
<td>first</td>
</tr>
<tr>
<td>sort</td>
<td>reduceByKey</td>
<td>partitionBy</td>
</tr>
<tr>
<td>union</td>
<td>cogroup</td>
<td>mapWith</td>
</tr>
<tr>
<td>join</td>
<td>cross</td>
<td>pipe</td>
</tr>
<tr>
<td>leftOuterJoin</td>
<td>zip</td>
<td>save</td>
</tr>
<tr>
<td>rightOuterJoin</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Resilient Distributed Datasets (RDD)
- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure

Operations
- Transformations (e.g. map, filter, groupBy)
- Actions (e.g. count, collect, save)
Operator Graph: Optimization and Fault Tolerance

Stage 1:
- A: join
- B: groupBy

Stage 2:
- C: map
- D: filter

Stage 3:
- E: join

Symbols:
- = RDD
- = Cached partition
Why Spark on Cassandra?

* Data model independent queries
* Cross-table operations (JOIN, UNION, etc.)
* Complex analytics (e.g. machine learning)
* Data transformation, aggregation, etc.
* Stream processing
How to Spark on Cassandra?

* DataStax Cassandra Spark driver
  * Open source: https://github.com/datastax/cassandra-driver-spark

* Compatible with
  * Spark 0.9+
  * Cassandra 2.0+
  * DataStax Enterprise 4.5+
Cassandra Spark Driver

* Cassandra tables exposed as Spark RDDs
* Read from and write to Cassandra
* Mapping of C* tables and rows to Scala objects
* All Cassandra types supported and converted to Scala types
* Server side data selection
* Spark Streaming support
* Scala and Java support
// Import Cassandra-specific functions on SparkContext and RDD objects
import com.datastax.driver.spark._

// Spark connection options
val conf = new SparkConf(true)
  .setMaster("spark://192.168.123.10:7077")
  .setAppName("cassandra-demo")
  .set("cassandra.connection.host", "192.168.123.10") // initial contact
  .set("cassandra.username", "cassandra")
  .set("cassandra.password", "cassandra")

val sc = new SparkContext(conf)
Accessing Data

```
CREATE TABLE test.words (word text PRIMARY KEY, count int);

INSERT INTO test.words (word, count) VALUES ('bar', 30);
INSERT INTO test.words (word, count) VALUES ('foo', 20);

// Use table as RDD
val rdd = sc.cassandraTable("test", "words")
// rdd: CassandraRDD[CassandraRow] = CassandraRDD[0]

rdd.toArray.foreach(println)
// CassandraRow[word: bar, count: 30]
// CassandraRow[word: foo, count: 20]

rdd.columnNames  // Stream(word, count)
rdd.size         // 2

val firstRow = rdd.first  // firstRow: CassandraRow = CassandraRow[word: bar, count: 30]
firstRow.getInt("count")  // Int = 30
```
Saving Data

```scala
val newRdd = sc.parallelize(Seq(('cat', 40), ('fox', 50)))
newRdd.saveToCassandra("test", "words", Seq(('word', 'count')))
```

*RDD above saved to Cassandra:

```sql
SELECT * FROM test.words;
```

```
word | count
-----|------
bar  | 30   
foo  | 20   
cat  | 40   
fox  | 50   
(4 rows)
```
<table>
<thead>
<tr>
<th>CQL Type</th>
<th>Scala Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii</td>
<td>String</td>
</tr>
<tr>
<td>bigint</td>
<td>Long</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>counter</td>
<td>Long</td>
</tr>
<tr>
<td>decimal</td>
<td>BigDecimal, java.math.BigDecimal</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>inet</td>
<td>java.net.InetAddress</td>
</tr>
<tr>
<td>int</td>
<td>Int</td>
</tr>
<tr>
<td>list</td>
<td>Vector, List, Iterable, Seq, IndexedSeq, java.util.List</td>
</tr>
<tr>
<td>map</td>
<td>Map, TreeMap, java.util.HashMap</td>
</tr>
<tr>
<td>set</td>
<td>Set, TreeSet, java.util.HashSet</td>
</tr>
<tr>
<td>text, varchar</td>
<td>String</td>
</tr>
<tr>
<td>timestamp</td>
<td>Long, java.util.Date, java.sql.Date, org.joda.time.DateTime</td>
</tr>
<tr>
<td>timeuuid</td>
<td>java.util.UUID</td>
</tr>
<tr>
<td>uuid</td>
<td>java.util.UUID</td>
</tr>
<tr>
<td>varint</td>
<td>BigInt, java.math.BigInteger</td>
</tr>
<tr>
<td>*nullable values</td>
<td>Option</td>
</tr>
</tbody>
</table>
Mapping Rows to Objects

* Mapping rows to Scala Case Classes
* CQL underscore case column mapped to Scala camel case property
* Custom mapping functions (see docs)

```sql
CREATE TABLE test.cars (
  id text PRIMARY KEY,
  model text,
  fuel_type text,
  year int
);
```

```scala
case class Vehicle(  
  id: String,  
  model: String,  
  fuelType: String,  
  year: Int
)

sc.cassandraTable[Vehicle]("test", "cars").toArray
// Array(Vehicle(KF334L, Ford Mondeo, Petrol, 2009),
//       Vehicle(MT8787, Hyundai x35, Diesel, 2011)
```
Server Side Data Selection

* Reduce the amount of data transferred

* Selecting columns

```scala
sc.cassandraTable("test", "users").select("username").toArray.foreach(println)
// CassandraRow{username: john}
// CassandraRow{username: tom}
```

* Selecting rows (by clustering columns and/or secondary indexes)

```scala
sc.cassandraTable("test", "cars").select("model").where("color = ?", "black").toArray.foreach(println)
// CassandraRow{model: Ford Mondeo}
// CassandraRow{model: Hyundai x35}
```
Spark SQL

Compatible

Spark SQL  Streaming  ML  Graph

Spark (General execution engine)

Cassandra/HDFS
Spark SQL

* SQL query engine on top of Spark
* Hive compatible (JDBC, UDFs, types, metadata, etc.)
* Support for in-memory processing
* Pushdown of predicates to Cassandra when possible
Spark SQL Example

```scala
import com.datastax.spark.connector._

// Connect to the Spark cluster
val conf = new SparkConf(true)...
val sc = new SparkContext(conf)

// Create Cassandra SQL context
val cc = new CassandraSQLContext(sc)

// Execute SQL query
val df = cc.sql("SELECT * FROM keyspace.table WHERE ...")
```
Spark Streaming

Spark SQL  Streaming  ML  Graph

Spark (General execution engine)

Cassandra/HDFS
Spark Streaming

* Micro batching
* Each batch represented as RDD
* Fault tolerant
* Exactly-once processing
* Unified stream and batch processing framework
import com.datastax.spark.connector.streaming._

// Spark connection options
val conf = new SparkConf(true)...

// streaming with 1 second batch window
val ssc = new StreamingContext(conf, Seconds(1))

// stream input
val lines = ssc.socketTextStream(serverIP, serverPort)

// count words
val wordCounts = lines.flatMap(_.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)

// stream output
wordCounts.saveToCassandra("test", "words")

// start processing
ssc.start()
ssc.awaitTermination()
Spark MLlib

Spark (General execution engine)

Cassandra/HDFS
Spark MLlib

* Classification (SVM, LogisticRegression, NaiveBayes, RandomForest...)
* Clustering (Kmeans, LDA, PIC)
* Linear Regression
* Collaborative Filtering
* Dimensionality reduction (SVD, PCA)
* Frequent pattern mining
* Word2Vec
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.classification.NaiveBayes

// Spark connection options
val conf = new SparkConf(true)... 

// read data
val data = sc.cassandraTable[Iris]("test", "iris")

// convert data to LabeledPoint
val parsedData = data.map { i => LabeledPoint(class2id(i.species),
Array(i.petal_l,i.petal_w,i.sepal_l,i.sepal_w)) }

// train the model
val model = NaiveBayes.train(parsedData)

//predict
model.predict(Array(5, 1.5, 6.4, 3.2))

APIs

- Scala
  - Native
- Python
  - Popular
- Java
  - Ugly
- R
  - No closures
Questions?
Please Remember to rate this session

Thank you!