Kindling

Getting Started With Spark & Cassandra

Erich Ess
CTO at SimpleRelevance
erich@simplerelevance.com

Data Source

- *I used the movielens data set for my examples:*
- *Details about the data are at: http:// www.grouplens.org/datasets/movielens/*

- *I used the 1M review dataset*
- *A direct link to the1M dataset files: http://files.grouplens.org/datasets/movielens/ml-1m.zip*

- *I also made up some of my own data*

Code Examples

- *Here is a Gist with the Spark Shell code I executed during the presentation:*

- *https://gist.github.com/erichgess/292dd29513e3393bf969*

Overview

- *What is Spark*
- *Spark Introduction*
- *Spark + Cassandra*
- *Demonstrations*

Goal

- *Build out the basic foundations for using Spark with Cassandra*
- *Simple Introduction*
- *Give a couple examples showing how to use Spark with Cassandra*
- *SparkSQL – Demonstrate one of the frameworks that augments Spark's power*

What is Spark

- *Distributed Compute Platform*
- *In Memory (FAST)*
- *Batch and Stream Processing*
- *Multi-language Support*
  - *Java, Python, and Scala out of the box*
- *Shell – You can do interactive distributed computing and analytics in Scala or Python*

The Basics

- *Spark Context*
  - *The connection to the cluster*
- *The Resilient Distributed Dataset*
  - *Abstracts the distributed data*
  - *The core of Spark*
- *Functional First Approach*
  - *Less code, more obvious intentions*

Spark Context

- *This is your connection to the Spark cluster*
- *Create RDDs*
- *When you open the Spark Shell, it automatically creates a context to the cluster*
- *When writing a standalone application to run on Spark you create a SparkContext and configure it to connect to the cluster*

Spark Context (cont.)

- *Configuring the Spark cluster for your application*
- *If you are using a database then the SparkContext is how you will set up the connection*
- *For Example, Cassandra:*
  - val conf = new SparkConf(true).set("spark.cassandra.connection.host", “localhost")

The Resilient Distributed Dataset

- *Use this to interact with data that has been distributed across the cluster*
- *The RDD is the starting point for doing parallel computations on Spark*
- *External Datasets*
  - *HDFS, S3, SQL, Cassandra, and so on*

The Resilient Distributed Dataset

- *Functional Transformations*
  - *These transform the individual elements of the RDD in one form or another*
  - *Map, Reduce, Filter, etc.*
  - *Lazy Evaluation: until you do something which requires a result, nothing is evaluated*
  - *These will be familiar if you work with any functional language (Haskell/F#/Clojure) or a language with functional elements (e.g. Scala/C#/Java8)*

The RDD (2)

- *Cache into Memory*
  - *Lets you put an RDD into memory*
  - *Dramatically speeds up processing on large datasets*
  - *The RDD will not be put in memory until an action forces the RDD to be computed (this is the lazy evaluation again)*

Transformations

- *Transformations are chainable*
  - *They take an RDD and return an RDD*
  - *The type the RDD wraps is irrelevant*
  - *Can be chained together*
  - *Map, filter, etc.*
- *Simply Put: Transformations return another RDD, Actions do not*

Transformations

- *myData.filter(x => x %2 == 1)*
- *myData.filter(x => x%2 == 1).map(y => 2*y)*
- *myData.map(x=> x/4).groupBy(x=>x > 10)*

Actions

- *Actions*
  - *These are functions which “unwrap” the RDD*
  - *They return a value of a non RDD type*
  - *Because of this they force the RDD and transformation chain to be evaluated*
  - *Reduce, fold, count, first, etc.*

Actions

- *myData.first*
- *myData.filter(x => x > 10).reduce(_+_)*
- *myData.take(5)*
- *myData.top(3)*

Fault Tolerance

- *For batch processing the chain of transformations IS fault tolerance*
- *Spark keeps a family tree for every RDD, from which it can recreate the exact chain of transformations and actions used to create the RDD*
- *If something fails, the Spark just replays the source data through the transformation chain to recreate the RDD*

The Shell

- *Spark provides a Scala and a Python shell*
- *Do interactive distributed computing*
- *Let's you build complex programs while testing them on the cluster*
- *Connects to the full Spark cluster*

Spark + Data

- *Out of the Box*
  - *Spark supports standard HDFS, S3, etc.*
- *Other Data Sources*
  - *Third Party drivers allow connecting to other data stores*
  - *SQL Databases*
  - *Cassandra*
- *Data gets put into an RDD*

Spark + Cassandra

- *DataStax provides an easy to use driver to connect Spark to Cassandra*
- *Configure everything with DataStax Enterprise and the DataStax Analytics stack*
- *Read and write data from Spark*
- *Interact with Cassandra through the Spark Shell*

Spark + DSE

- *Each node has both a Spark worker and Cassandra*
- *Data Locality Awareness*
  - *The Spark workers are aware of the locality of data and will pull the data on their local Cassandra nodes*

Pulling Some Data from Cassandra

- *Use the SparkContext to get to the data*
  - *sc.cassandraTable(keyspace,table)*
  - This returns an RDD (which has not actually been evaluated because it's lazy)
  - The RDD represents all the data in that table
- The RDD is of Row type
  - The Row type is a type which can represent any single row from a Cassandra table

Quick Sample

Pulling Data a Little Cleaner

- *The Row type is a little messy to deal with*
- *Let's use a case class* to load a table directly into a type which represents what's in the table

Saving back into Cassandra

- *The RDD has a function called saveToCassandra*
- *MyData.saveToCassandra(keyspace,table)*

Sample Code

- *case class Example(A: Int, B: Int)*
- *val data = Seq(Example(1,1), Example(2,2))*
- *val pdata = sc.parallelize(data)*
- *pdata.saveToCassandra(“demo”, “first_example”)*

Beyond

- *Streams*
- *Machine Learning*
- *Graph Processing*