Speaker Notes: Data Engineering with Apache Spark

Markus Dale, medale@asymmetrik.com

May 2019

Setup

- Open Spark API:
  https://spark.apache.org/docs/latest/api/scala/index.html

Intro, Slides And Code

- Bio:
    - mostly Java, big data with Hadoop
    - big data with Spark, Databricks, Scala
    - Now Asymmetrik - Scala, Spark, Elasticsearch, Akka...
    - Data Engineer
- Slides: https://github.com/medale/prez-spark-dataengineering/blob/master/presentation/SparkDataEngineering.pdf
- Scala Spark Code Examples: https://github.com/medale/prez-spark-dataengineering
- Also https://github.com/medale/spark-mail

Data Science Mission - ID malicious GitHub Pull Requests?

- https://www.gharchive.org/

Data Engineering Mission

- https://www.gharchive.org/
- Old API/Events API (we won’t deal with old API)
- Events API - PullRequestEvent

Data Engineering

- dataquest.io: “transform data into a useful format for analysis”

Apache Spark - Big data tooling

- Shell for exploration at scale
- Dataset batch API - many supported input sources/formats
    - builds on Hadoop and other 3rd party libraries
- Streaming API
- ML library
- Graph library

Apache Spark: Data engineering on small dataset

- Take subset of data
- Figure out structure, approaches

Apache Spark: Data engineering for larger dataset (Vertical Scaling)

- Server-grade machine - more cores
- More memory, more data

Apache Spark: Data engineering for large datasets (Horizontal Scaling)

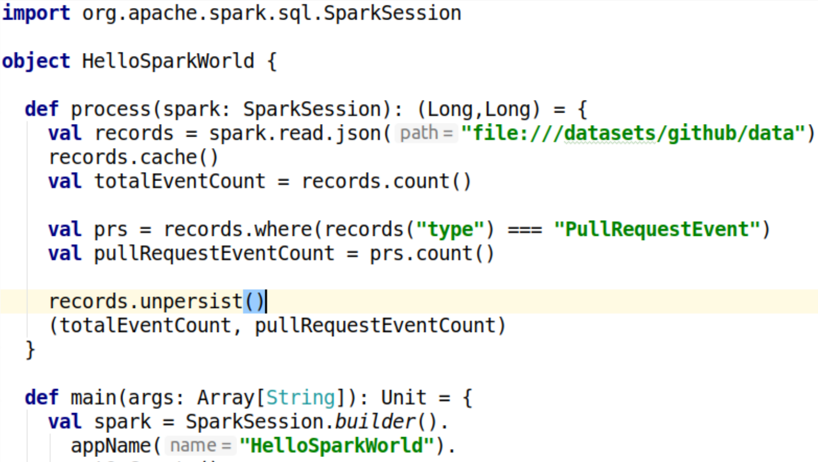
- Cluster manager manages resources
- Spark manages Spark application (driver, executors)
    - Sunny day
    - Error handling (machine dies, slows, network...)

Cluster Manager - Manage cores, memory, special capabilities

- Spark local mode (not a cluster manager)
- Spark Standalone
- Kubernetes, Mesos
- Spark on Hadoop YARN
- In cloud: Spark on AWS EMR, Google, Azure, Databricks
- Schedule resources

Anatomy of a Spark Application

- One cluster manager - multiple Spark applications
- Per Spark application
  - 1 driver
  - n executors (cache memory, task slots)



```
import org.apache.spark.sql.SparkSession

object HelloSparkWorld {

  def process(spark: SparkSession): (Long,Long) = {
    val records = spark.read.json( path = "file:///datasets/github/data")
    records.cache()
    val totalEventCount = records.count()

    val prs = records.where(records("type") === "PullRequestEvent")
    val pullRequestEventCount = prs.count()

    records.unpersist()
    (totalEventCount, pullRequestEventCount)
  }

  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder().
      appName( name = "HelloSparkWorld").
```

Hello, Spark World!

- spark session
- spark.read (DataFrameReader) - json (.gz, 1 json per line)
- lazy transformation - read to get schema
- count - action - execute a job
- Datasets, DataFrame and RDD are immutable
- contain lineage (how did we get to this dataset?)
- where - transformation

SparkSession - Gateway to the Cluster

- builder static method - Builder
- appName
- config
- master
- getOrCreate()

API - SparkSession Object

- spark.apache.org - Documentation - API Docs
- Object (“static” methods) vs. class

API - SparkSession Class

- read - DataFrameReader input
- createDataFrame/createDataset
- emptyDataFrame/emptyDataset
- spark.implicits._ ($ and Scala object encoder)
- udf
- stop

API - DataFrameReader

- csv
- json
- parquet
- text - DataFrame - column “value”
- textFile - Dataset[String]
- schema - specify read schema

DataFrame = Dataset[Row]

- sql package object - `type DataFrame = Dataset[Row]`
- DataFrame has a schema

DataFrame Schema

- printSchema
- .schema ()

GitHub Data

- subset to local drive
- for production - need distributed storage system
    - S3
    - Hadoop HDFS

Preliminary Exploration

- What’s in the gz?
- How many lines?

Editor: one JSON per line

- visual exploration - seems like one JSON per line

Pretty Print One Record?

- split
- python json.tool

Open pretty.json in Atom - PullRequestEvent

- see JSON, look at fields for one record

Starting Spark Standalone Cluster Manager

- start master to explicit host (default port 7077)
- one or more workers to spark://...:

Running spark-shell in cluster

- –master - what cluster manager to ask for resources
- –deploy-mode (default client or cluster)
- driver: coordinates this Spark application
- executors - cores - how many tasks in parallel
- jars (built via assembly)

Spark Shell Startup

- web ui (for this Spark application)
    - 4040, 4041 etc.
- Special vars: spark: SparkSession, sc: SparkContext
- Exit: `:quit`

Spark Standalone Cluster Manager - 1 running application

- 4 cores (total)
- 2 executors with 2 cores each
- 2GB/executor
- Link to Spark shell - Spark application UI

spark-shell auto-imports

- SparkContext - old
- spark.implicits._ - $ function, encoders for Scala primitives and case classes
- spark.sql package - DataFrame (Dataset[Row])
- functions: math, string, date for columns

Data Exploration - schema and counting

- urls - file, hdfs, s3a
- schema - superset of all JSONs
- just execute job (list files, read to find schema)

Spark Application UI - Jobs, stages, tasks

- job0 - read 3 unsplittable files, determine JSON schema
    - 1 stage - everything in parallel
    - 2 executors with two task slots each
    - 3 tasks - read unsplittable files
- job1 - count
    - 2 stages - count local, shuffle, add up total
    - 4 tasks - 3 local counts, 1 shuffle add total

Spark Application UI - Stages

- 3 stages - last stage 1 task

Spark Application UI - Stage details

- input, output

Job 1 - Stages 1 and 2 DAG

- See two stages - shuffle

One Job = n lazy transformations, 1 action

- lazy transformations
- Dataset api - select (projection)
- distinct
- show - action, count (2 job)
- cache/unpersist

Spark Application UI - Storage (caching)

- cache (persist level)
- unpersist
- in memory, spill to local disk

Default file system/file system URLs

- no URL prefix needed for defaultFS
- file, hdfs, s3a

Input partitions - splittable file?

- Splittable: bzip2, parquet, avro
- Non-splittable: gzip (1 task per file)
- small file problem

Datasets/DataFrames compiled to RDDs

- Catalyst query optimizer for built-in functions
- Project Tungsten - memory management
    - Row storage (Apache Arrow)
    - Encoders for Dataset objects (spark.implicits._)

Data Exploration - event type distribution

- where clause
- groupBy - RelationalGroupedDataset
    - count
    - avg, sum, agg (agg functions mean, std dev...)
- where(String), where(Column)
- getting column - apply on Dataset, $, col function

Narrow vs. wide transformations

- https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-transformations.html
- narrow, wide

Shuffle Partitions default

- 200 (“spark.sql.shuffle.partitions”)

Setting Shuffle Partitions

- “spark.sql.shuffle.partitions”

Shuffle Partitions Optimized

- Less time - 0.1 vs. 0.8

API - Some Dataset Transformations

- select
- where
- distinct
- limit
- orderBy
- join

API - Some Dataset Actions

- collect
- count
- take(n)
- head
- write - DataFrameWriter

DataFrameWriter

- DataSourceRegister, implicits

Just the PullRequestEvents and their schema

- text - one column “value”

Writing partitions - Output directory

- _ hidden
- also hidden .crc files
- part-00000-UUID...

API - Column

- Some dataset - use string, some columns
- +, -, *, %
- ===, =!=, >, <, …
- asc, desc
- startsWith, contains, endsWith, like, rlike
- isNull, isNaN, isIn

API - functions

- array functions, explode
- date/time functions
- math, string
- use built-in to allow Catalyst optimization
- also user-defined-functions, user-defined-agg-functions

Date Exploration

- created_at is a string but in ISO 8601

Adding year, month, day, hour columns

- immutable, 4 transformations
- use time functions (require column)
- spark.implicits._ $

Saving to Parquet with partitioning columns

- year=2019 etc.

Catalyst optimizer, tungsten memory management

Memory pressure - partitions, executors, shuffle partitions

Serialization code

Airflow - Spark workflows

S3

sc.hadoopConfiguration.set("fs.s3a.secret.key",

sc.hadoopConfiguration.set("fs.s3a.access.key",

https://www.gharchive.org/ wget

http://data.gharchive.org/2019-04-28-0.json.gz wget

http://data.gharchive.org/2019-04-28-1.json.gz wget

http://data.gharchive.org/2019-04-28-13.json.gz

store under data directory run spark-shell from parent of data directory (gz of .json file with one json per line)

```
val records = spark.read.json("data")
//slow - needs to figure out JSON schema
records.cache
records.count
//235728
```