# Apache Spark Through Email

Markus Dale

Nov 2018

# Intro, Slides And Code

- mostly Java, big data with Hadoop
- big data with Spark, Databricks, Scala
- Now Asymmetrik - Scala, Spark, Elasticsearch, Akka...
- Slides: https://github.com/medale/spark-mail/blob/master/presentation/ApacheSparkThroughEmail.pdf
- Spark Code Examples: https://github.com/medale/spark-mail/
- README.md
  - describes how to get and parse Enron email dataset
  - Spark, Docker notebook setup

# Goals

- Scalable data processing
- Spark structured processing
- Spark concepts - RDD, Dataset/DataFrame, partitions
- Notebooks, Spark UI

# Data Science for Small Dataset

- Laptop
- Explore subset, develop approaches/algorithms, find *features*

# Data Science for Larger Dataset

- Standalone server - more memory, faster CPU, more storage

# Data Science for Larger Dataset (Vertical Scaling)

- Big iron - lots of cores, memory, disk/SSDs, GPUs

# Data Science for Large Datasets (Horizontal Scaling)

- Parallelize, coordinate compute among many "commodity" machines
- Deal with *failure*

# Big Data Framework - Apache Hadoop

- Google GFS (2003), Google MapReduce (2004)
- Hadoop (Nutch - open source web crawler/Lucene) - Doug Cutting, Mike Cafarella
  - Yahoo, Cloudera, Hortonworks, MapR

# Hadoop Ecosystem

- HDFS, YARN, MapReduce (Spark replaces MR)
- HBase (Google BigTable), Cassandra, Accumulo
- Pig, Hive - MR scripting DSL/SQL

# Apache Spark Components

- Foundation: Resilient Distributed Datasets (RDD)
  - Broadcast variables, accumulators
  - Java objects, should use Kryo serialization
- *Structured APIs* (use these) - Datasets, DataFrames, SQL
  - Spark manages object layout in memory, schemas, code generation
- Streaming, MLlib (Advanced analytics)
- Scala, Java, Python, R + library ecosystems
- Submit (Batch/Stream) or Shell/Notebooks (e.g. Zeppelin, Jupyter)

# Hello, Spark Email World!

- Jupyter Notebook with Apache Toree
- Comment out spark.close, Restart & run all
- See ApacheSparkThroughEmail1

# Cluster Manager, Driver, Executors, Tasks

- *Cluster manager*: Spark Standalone, Hadoop YARN, AWS EMR, Kubernetes, Mesos
- *Driver* (start once)
  - Execute user code
  - Schedules tasks for executors
  - Serialize code (closures with data) as tasks to executors
- *Executors* on worker nodes (start once, restart)
  - Cache - distributed memory for partitions
  - Execute tasks (threads/core = parallelism of tasks)
  - Read/manage partitions (serialization - Kryo)

# SparkSession: Entry to cluster

- Notebook: `spark` provided (spark.close or JVM shutdown)
- Code: SparkSession.builder() w/ appName, master, `getOrCreate`
  - spark-submit (by hand, or Airflow, or EMR)
- Show SparkSession in Spark ScalaDocs (clickable link)

# DataFrameReader: Input for structured data

- `spark.read` - built-in: jdbc, csv, json, parquet, text
- 3rd party: https://spark-packages.org - Avro, Redshift, MongoDB...
- Also, any Hadoop InputFormat via RDD/SparkContext

# Scaling Behind the Scenes

- Executing notebook - show Spark Application UI (stay on Jobs page)
- Job - series of transformations followed by action!
  - *transformations*: select (projection), where, limit, cache
  - *actions*: count, df.write..., df.collect/take/head/first (memory!)
- Task - ~1 task per partition
  - Serialize code (no classes, non-serializable)
  - Data in closure (e.g. HashMap - use broadcast variables!)

# Stages: Pipeline work per stage - shuffle

- Click on description for job with 209 tasks
- 3 Stages: pipeline per stage
  - Wholestage Code Gen - Tungsten code gen engine
  - InMemoryTableScan
  - Exchange (shuffle data with same from value)
    - HashPartitioner (Strings etc.), RangePartitioner (values)
    - groupBy
  - TakeOrderedAndProject exchange data
    - orderBy, limit
- Behind the scenes - scaling!

# Where clause, Column methods, Built-in functions

- methods on Dataset
- methods on Columns
- sql.functions
- Ensure Notebook 1 is closed (kill on UI or spark.close)
- Restart & run all

# Spark APIs - DataFrameReader, Dataset, Column, functions

- Show Spark Scala API docs
- Show Spark documentation

# Parallelism and Partitioning

- Goldilocks - not too many, not too few
  - Too many tasks - scheduling overhead, little work
  - Too few - tasks take very long
- Initial parallelism - number of input "blocks"
- Shuffle - `spark.sql.shuffle.partitions` configuration
- repartition (shuffle)/coalesce (combine on same executor)
  - e.g. write out 1 partition

# And now for something completely different: Colon Cancer

- Screening saves lives! Colonoscopy

# Questions?

- medale@asymmetrik.com
- Ping Pong
- Baltimore Scala meetup - January 10
- Spark mail repo for getting/parsing Enron data, presentations, code, notebooks