

# Apache Spark Through Email

---

Markus Dale

July 2024

- mostly Java, big data with Hadoop
- big data with Spark, Databricks, Scala
- Now Asymmetrik/BlueHalo - Scala, 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

- Scalable data processing (batch processing)
- Spark structured processing
- Spark concepts - RDD, Dataset/DataFrame, partitions

- Laptop
- Explore subset, develop approaches/algorithms, find *features*
- Outliers, distribution
- Data cleaning - missing values

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

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

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

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



- Local
  - Download from <https://spark.apache.org>, untar, add to PATH
  - SDKMAN - `curl -s "https://get.sdkman.io" | bash`
    - `sdk install spark`
  - `spark-shell` or `pyspark`
- Standalone cluster, Hadoop YARN
  - Need shared file system or common datastore (e.g. AWS S3)
- Cloud-based managed:
  - AWS EMR
  - GCP Dataproc
  - Databricks on Azure, GCP or AWS

- 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, (Jobs -> 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)

- 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)

- `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

- 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!



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

- 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 - Adaptive Query Execution (dynamic partitioning)
- repartition (shuffle)/coalesce (combine on same executor)
  - e.g. write out 1 partition

- UDF

- RDD - Matei Zaharia
- SQL - Michael Armbrust

## And now for something completely different: Colon Cancer

- Screening saves lives! Colonoscopy

## Questions?

- [markus.dale@bluehalo.com](mailto:markus.dale@bluehalo.com)
- Spark mail repo for getting/parsing Enron data, presentations, code, notebooks