

Apache Spark Through Email

Markus Dale, medale@asymmetrik.com

Nov 2018

- 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

Goals

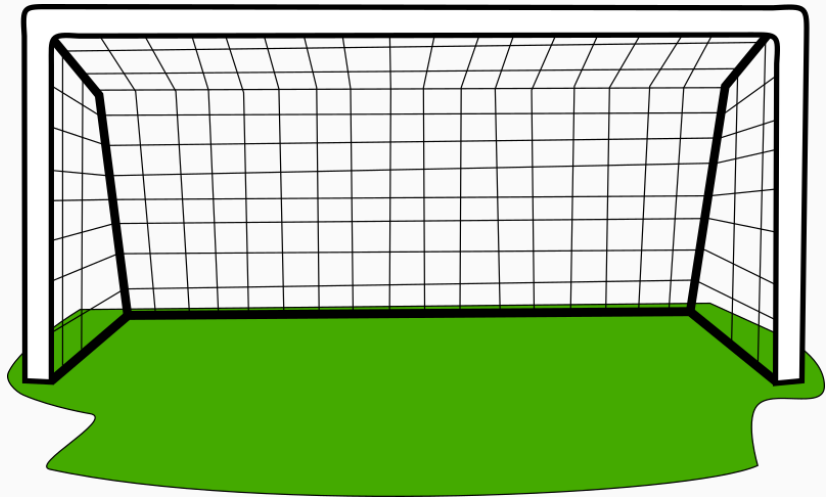


Figure 1: Intro to Apache Spark

Data Science for Small Dataset



Figure 2: Laptop



Figure 3: Standalone Server

Data Science for Larger Dataset (Vertical Scaling)

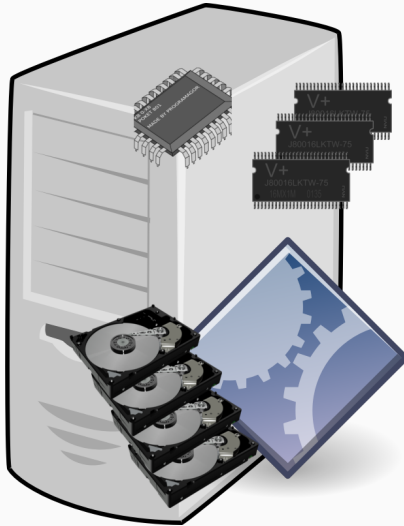


Figure 4: Beefed-up Server

Data Science for Large Datasets (Horizontal Scaling)





Figure 6: HDFS, MapReduce

Hadoop Ecosystem

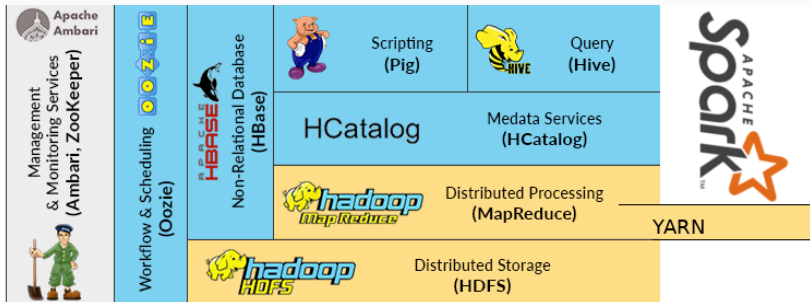


Figure 7: Some Frameworks Around Hadoop

Apache Spark Components

Structured
Streaming

Advanced
Analytics

Libraries &
Ecosystem

Structured APIs

Datasets

DataFrames

SQL

Low-level APIs

RDDs

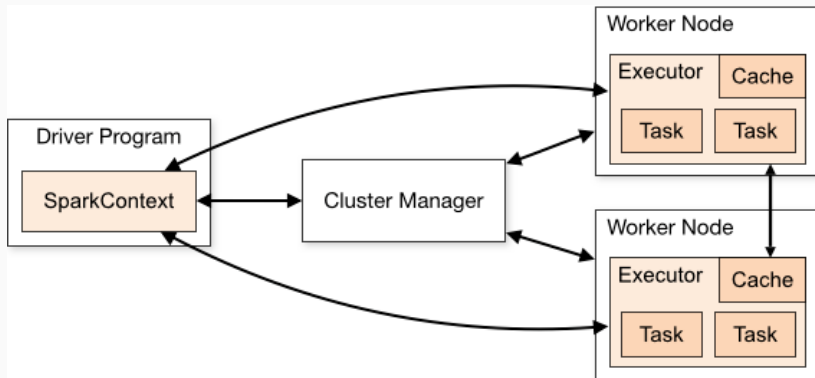
Distributed Variables

Source: Spark: The Definitive Guide

Hello, Spark Email World!

- Jupyter Notebook with Apache Toree
- See Notebook
../notebooks/html/ApacheSparkThroughEmail1.html

Cluster Manager, Driver, Executors, Tasks



Source: Apache Spark website

SparkSession: Entry to cluster

- spark: spark.sql.SparkSession

//SparkSession provided by notebook as spark

```
val records = spark.read.  
    parquet("/datasets/enron/enron-small.parquet")
```

//In regular code for spark-submit

//com.uebercomputing.spark.dataset.TopNEmailMessageSenders

```
val spark = SparkSession.builder().  
    appName("TopNEmailMessageSenders").  
    master("local[2]").getOrCreate()
```

DataFrameReader: Input for structured data

- `spark.read: spark.sql.DataFrameReader`
 - `jdbc`
 - `json`
 - `parquet`
 - `text...`
 - Also: <https://spark-packages.org> - Avro, Redshift, MongoDB...

Scaling Behind the Scenes

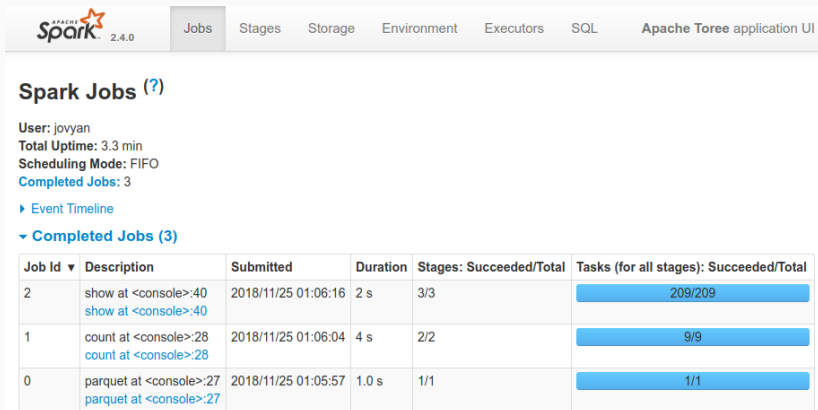


Figure 8: Jobs and Tasks

Stages: Pipeline work per stage - shuffle

▼ DAG Visualization

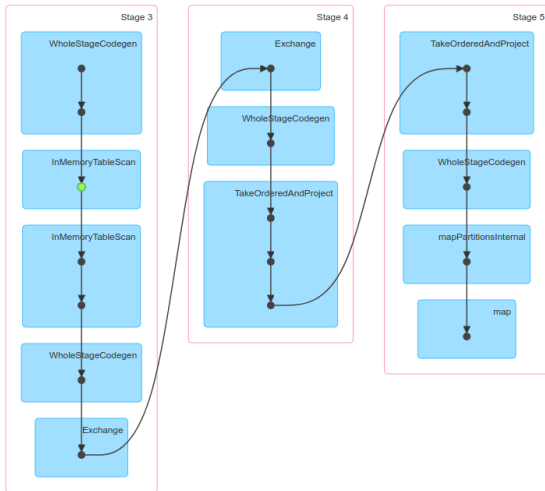


Figure 9: Stages


- See Notebook
../notebooks/html/ApacheSparkThroughEmail2.html

Spark APIs - DataFrameReader, Dataset, Column, functions

▶	<code>def parquet(paths: String*): DataFrame</code> Loads a Parquet file, returning the result as a DataFrame .
▶	<code>def parquet(path: String): DataFrame</code> Loads a Parquet file, returning the result as a DataFrame .
▶	<code>def schema(schemaString: String): DataFrameReader</code> Specifies the schema by using the Input DDL-formatted string.
▶	<code>def schema(schema: StructType): DataFrameReader</code> Specifies the input schema.
▶	<code>def table(tableName: String): DataFrame</code> Returns the specified table as a DataFrame .
▶	<code>def text(paths: String*): DataFrame</code> Loads text files and returns a DataFrame whose schema starts with a string column named "value".
▶	<code>def text(path: String): DataFrame</code> Loads text files and returns a DataFrame whose schema starts with a string column named "value".
▶	<code>def textFile(paths: String*): Dataset[String]</code> Loads text files and returns a Dataset of String .
▶	<code>def textFile(path: String): Dataset[String]</code> Loads text files and returns a Dataset of String .

Expression operators	
▶	<code>def %(other: Any): Column</code> Modulo (a.k.a.
▶	<code>def %(other: Any): Column</code> Boolean AND.
▶	<code>def *(other: Any): Column</code> Multiplication of this expression and another expression.
▶	<code>def +(other: Any): Column</code> Sum of this expression and another expression.
▶	<code>def -(other: Any): Column</code> Subtraction.
▶	<code>def /(other: Any): Column</code> Division of this expression by another expression.
▶	<code>def <(other: Any): Column</code> Less than.
▶	<code>def <=(other: Any): Column</code> Less than or equal to.
▶	<code>def <==(other: Any): Column</code> Equality test that is safe for null values.
▶	<code>def !=(other: Any): Column</code> Inequality test.

Date time functions	
▶	<code>def add_months(startDate: Column, numMonths: Int): Column</code> Returns the date that is numMonths after startDate.
▶	<code>def current_date(): Column</code> Returns the current date as a date column.
▶	<code>def current_timestamp(): Column</code> Returns the current timestamp as a timestamp column.
▶	<code>def date_add(start: Column, days: Int): Column</code> Returns the date that is days days after start.
▶	<code>def date_format(dateExpr: Column, format: String): Column</code> Converts a date/timestamping to a value of string in the format specified by the second argument.
▶	<code>def date_sub(start: Column, days: Int): Column</code> Returns the date that is days days before start.
▶	<code>def date_trunc(format: String, Timestamp: Column): Column</code> Returns timestamp truncated to the unit specified by the format.
▶	<code>def datediff(end: Column, start: Column): Column</code> Returns the number of days from start to end.
▶	<code>def dayOfMonth(e: Column): Column</code> Extracts the day of the month as an integer from a given date/timestamping string.
▶	<code>def dayOfWeek(e: Column): Column</code> Extracts the day of the week as an integer from a given date/timestamping string.

 Dataset

`class Dataset\[T\] extends Serializable`

A [Dataset](#) is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each operation available on [Datasets](#) is divided into transformations and actions. Transformations are the ones that produce new [Datasets](#), and actions filter, select, and aggregate (groupBy). Example actions count, show, or writing data out to file systems.

[Datasets](#) are "lazy", i.e. computations are only triggered when an action is invoked. Internally, a [Dataset](#) represents a logical plan that describes the optimized logical plan and generates a physical plan for efficient execution in a parallel and distributed manner. To explore the logical plan as is.

To efficiently support domain-specific objects, an [Encoder](#) is required. The encoder maps the domain specific type T to Spark's internal type system used to let Spark to generate code at runtime to serialize the [Person](#) object into a binary structure. This binary structure often has much lower size to understand the internal binary representation for data, use the [Schema](#) function.

There are typically two ways to create a [Dataset](#). The most common way is by pointing Spark to some files on storage systems, using the [read](#) function.

```
val people = spark.read.parquet(".*").as[Person] // Scala
(Dataset<Person> people = spark.read().parquet(".*").as[Encoders.bean(Person.class)]; // Java
```

[Datasets](#) can also be created through transformations available on existing [Datasets](#). For example, the following creates a new [Dataset](#) by applying:

```
val names = people.map(_.name) // In Scala; names is a Dataset[String]
Dataset<String> names = people.map((Person p) -> p.name, Encoders.STRING);
```

[Dataset](#) operations can also be triggered, through various domain-specific language (DSL) functions defined in [Dataset](#) (this class), [Column](#), and [SQL](#) in R or Python.

To select a column from the [Dataset](#), use [apply](#) method in [Scala](#) and [Col](#) in [Java](#).

```
val ageCol = people("age") // In Scala
Column ageCol = people.col("age"); // In Java
```

Note that the [Column](#) type can also be manipulated through its various functions.

```
// The following creates a new column that increases everybody's age by 18.
people("age") + 18 // In Scala
people.col("age").plus(18); // In Java
```

Parallelism and Partitioning

- Goldilocks - not too many, not too few
- Initial parallelism - number of input “blocks”
- Shuffle - `spark.sql.shuffle.partitions` configuration

- See Notebook
../notebooks/html/ApacheSparkThroughEmail3.html

And now for something completely different: Colon Cancer



- Screening saves lives!
 - Colonoscopy - talk to your doc
- Colorectal Cancer Alliance

Questions?



- medale@asymmetrik.com
- Infrequent blog/past presentations
<http://uebercomputing.com/>
- Baltimore Scala Meetup
<https://www.meetup.com/Baltimore-Scala/>
- Spark Mail repo <https://github.com/medale/spark-mail/>