CS4225/CS5425 Big Data Systems for Data Science

Spark I: Basics

Ai Xin
School of Computing
National University of Singapore
aixin@comp.nus.edu.sg

Intro

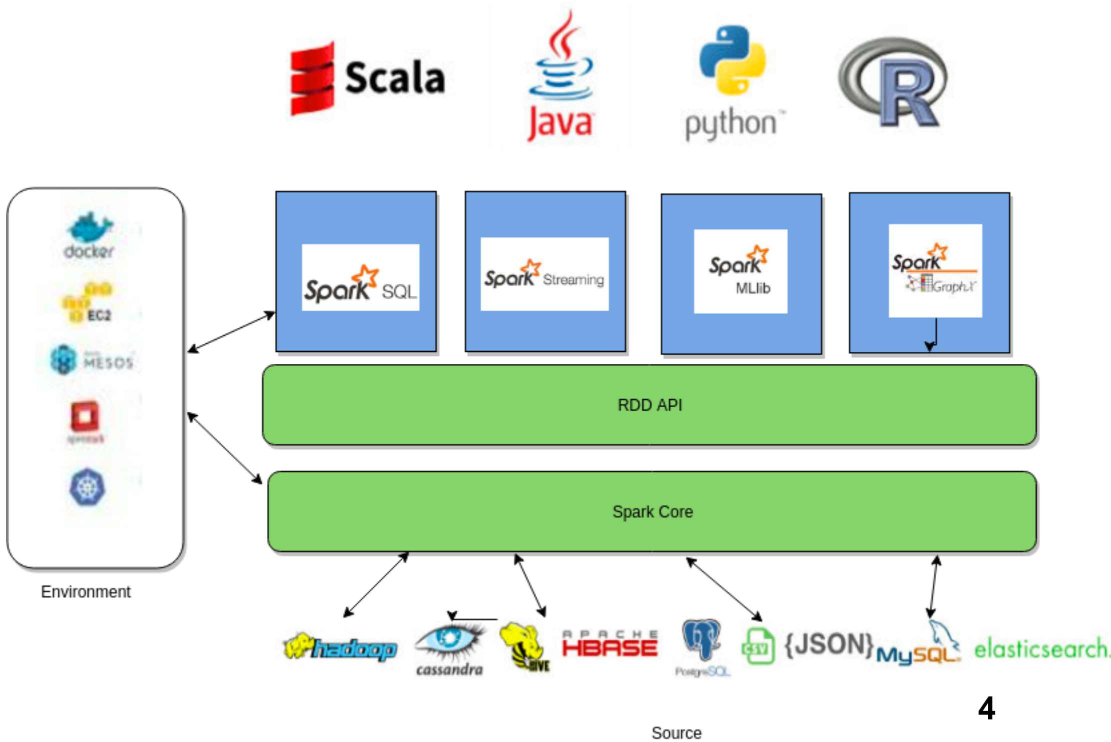
- Lecturer: Ai Xin
  - Email: aixin@comp.nus.edu.sg
  - Office Hours: 2-3pm on 20 Oct, 3, 17 and 24 Nov at COM3-B1-24
- TAs
  - Assignment 2 (Post to Canvas/Discussion or Email TAs)
    - SIDDARTH NANDANAHOSUR SURESH (Name A-G)
    - TAN TZE YEONG (Name H-L)
    - TAN YAN RONG AMELIA (Name L-R)
    - TENG YI SHIONG (Name R-W)
    - TOH WEI JIE (Name W-Z)
  - Tutorial and Lecture (Post to Canvas/Discussion or Email TAs)
    - ZHANG JIHAI (week 7 – 9)
    - GOH TECK LUN (conduct tutorials)
    - Hu Zhiyuan (week 10 – 13)

Schedule

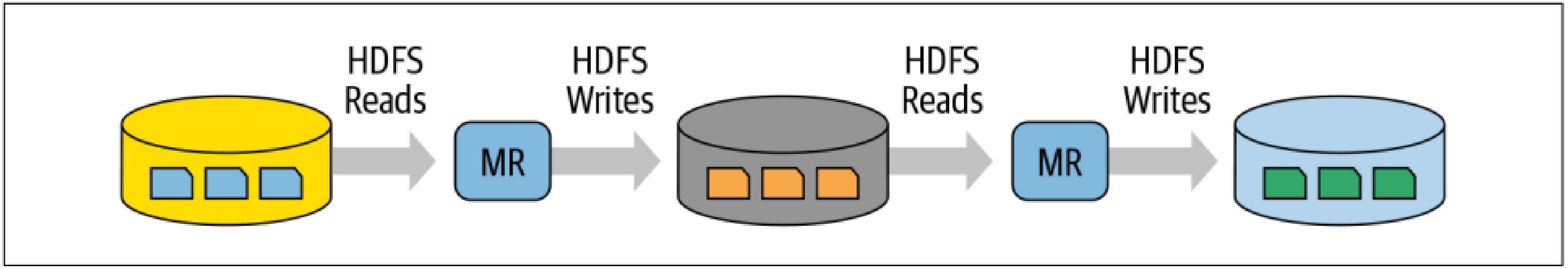
Week	Date	Topics	Tutorial	Due Dates
1	18-Aug	Overview and Introduction		
2	25-Aug	MapReduce - Introduction		
3	1-Sep	Polling Day Public Holiday		
4	8-Sep	MapReduce and Database	Tutorial: MapReduce	Assignment 1 released
5	15-Sep	NoSQL Overview 1		
6	22-Sep	NoSQL Overview 2	Tutorial: NoSQL	
Recess				
7	6-Oct	Apache Spark 1		
8	13-Oct	Apache Spark 2	Assignment 2 Briefing	Assignment 1 due (15 Oct 11:59pm) Assignment 2 released
9	20-Oct	Stream Processing	Tutorial: Spark	
10	27-Oct	Large Graph Processing 1		
11	3-Nov	Large Graph Processing 2	Tutorial: Stream Processing	
12	10-Nov	NUS Well Being		
13	17-Nov	Delta Lake + Revision	Tutorial: Graph Processing	Assignment 2 due (19 Nov 11:59pm)
	29-Nov	**Final Exam**		

Today’s Plan

- **Introduction and Basics**
- Working with RDDs
- Caching and DAGs
- DataFrames and Datasets

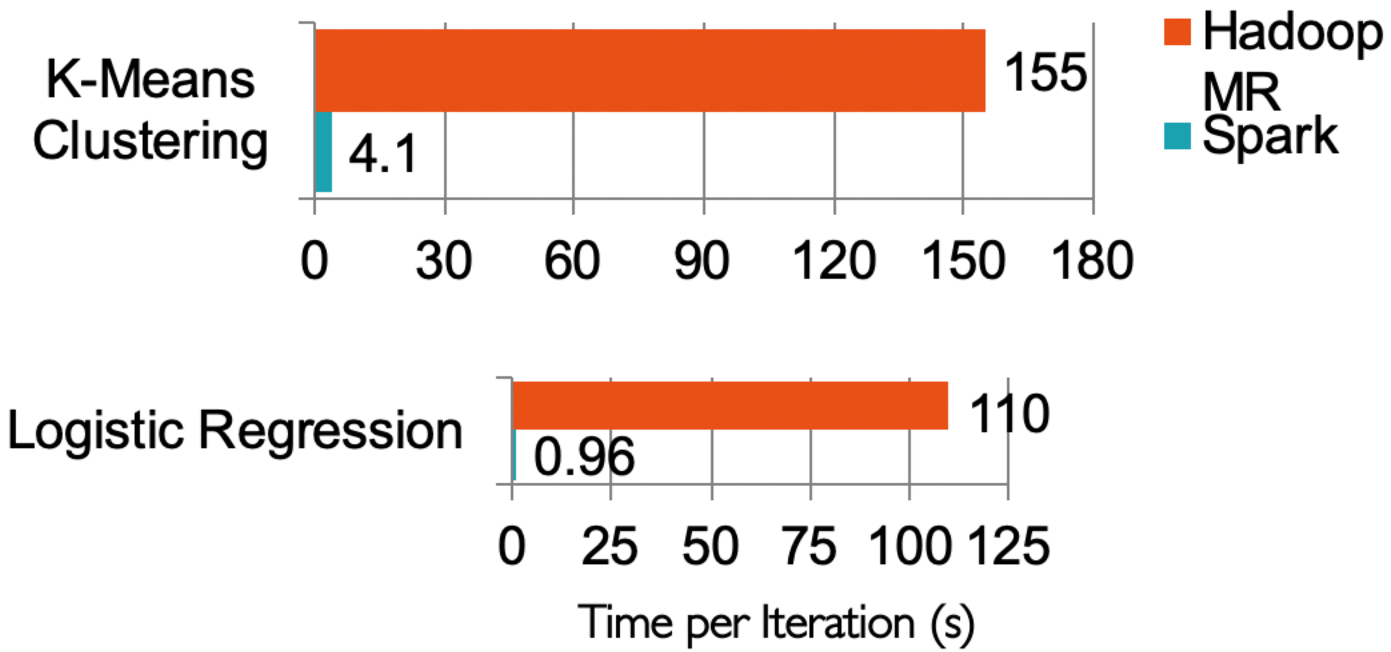


Motivation: Hadoop vs Spark



- Issues with Hadoop Mapreduce:
  - **Network and disk I/O costs**: intermediate data has to be written to local disks and shuffled across machines, which is slow
  - Not suitable for **iterative** (i.e. modifying small amounts of data repeatedly) processing, such as interactive workflows, as each individual step has to be modelled as a MapReduce job.
- Spark stores most of its intermediate results in memory, making it much faster, especially for iterative processing
  - When memory is insufficient, Spark **spills to disk** which requires disk I/O

Performance Comparison



Ease of Programmability

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

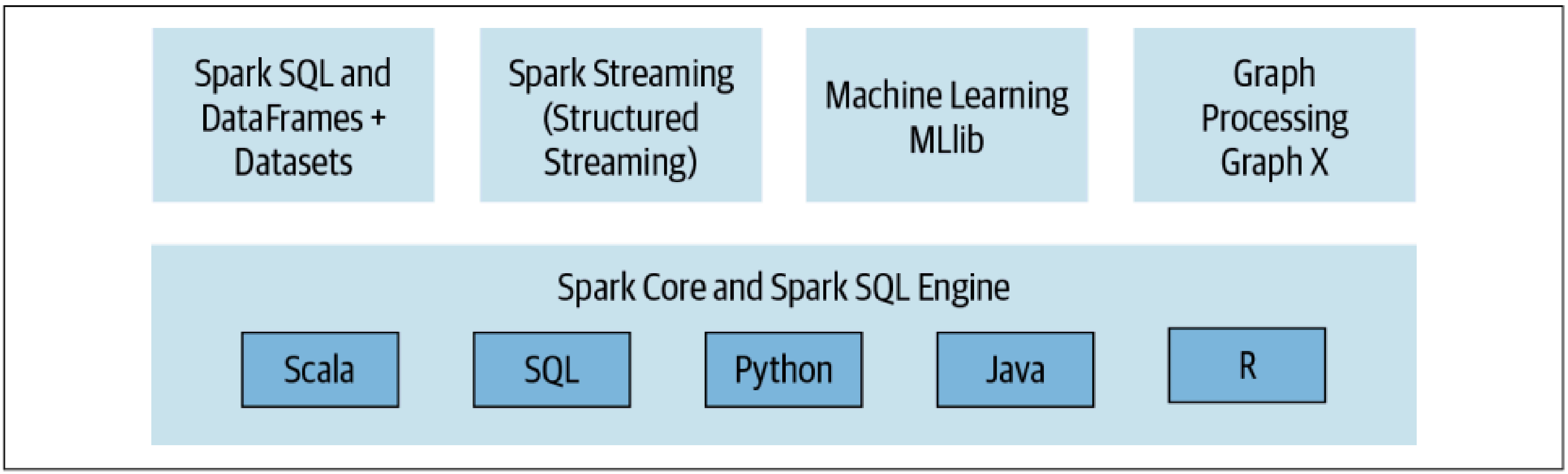
WordCount (Hadoop MapReduce)

Ease of Programmability

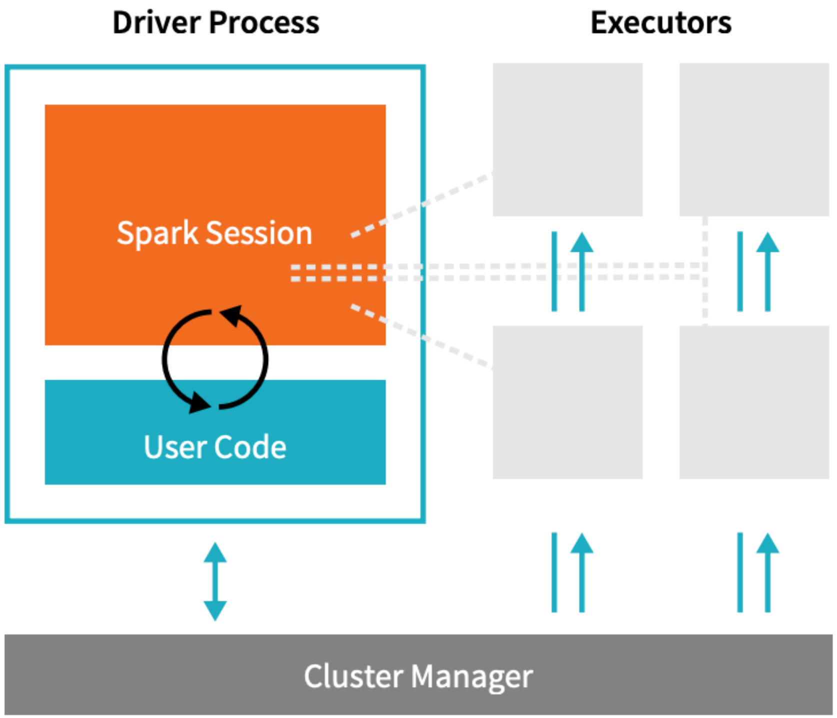
```
val file = sc.textFile("hdfs://...")

val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.save("...")
```

WordCount (Spark)

Spark Components and API Stack



Spark Architecture



- **Driver Process** responds to user input, manages the Spark application etc., and distributes work to **Executors**, which run the code assigned to them and send the results back to the driver
- **Cluster Manager** (can be Spark's standalone cluster manager, YARN, Mesos or Kubernetes) allocates resources when the application requests it
- In **local mode,** all these processes run on the same machine

Evolution of Spark APIs

Resilient Distributed Datasets (2011) → **DataFrame (2013)** → **DataSet (2013)**

Resilient Distributed Datasets (2011)

- A collection of JVM objects
- Functional operators (map, filter, etc)

DataFrame (2013)

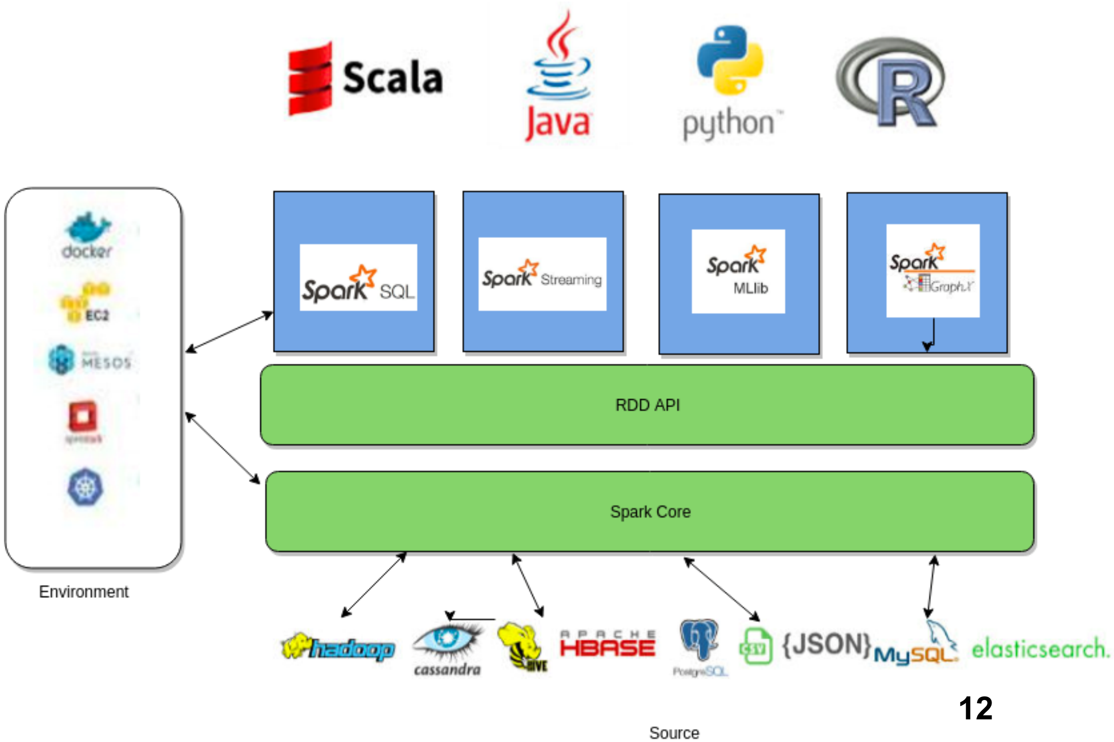
- A collection of Row objects
- Expression-based operations
- Logical plans and optimizer

DataSet (2013)

- Internally rows, externally JVM objects.
- Almost the "Best of both worlds": type safe + fast

Today's Plan

- Introduction and Basics
- **Working with RDDs**
- Caching and DAGs
- DataFrames and Datasets



Achieve fault tolerance through **lineages**

Represent a collection of objects that is **distributed over machines**

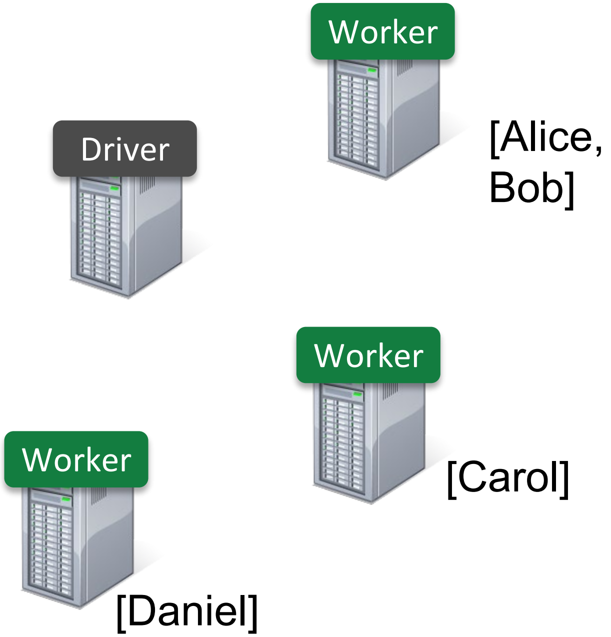
Resilient Distributed Datasets (RDDs)

RDD: Distributed Data

```
# Create an RDD of names, distributed over 3 partitions

dataRDD = sc.parallelize(["Alice", "Bob", "Carol", "Daniel"], 3)
```

Partition data
into 3 parts

- RDDs are **immutable**, i.e. they cannot be changed once created.
- This is an RDD with 4 strings. In actual hardware, it will be partitioned into the 3 workers.

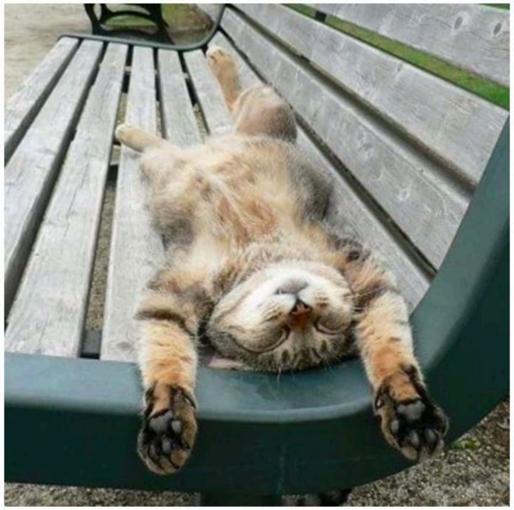


Transformations

- **Transformations** are a way of transforming RDDs into RDDs.

```
# Create an RDD: length of names

dataRDD = sc.parallelize(["Alice", "Bob", "Carol", "Daniel"], 3)

nameLen = dataRDD.map(lambda s: len(s))
```

- This represents the transformation that maps each string to its length, creating a new RDD.
- However, transformations are **lazy**. This means the transformation will not be executed yet, until an **action** is called on it
  - Q: what are the advantages of being lazy?
  - A: Spark can optimize the query plan to improve speed (e.g. removing unneeded operations)
- Examples of transformations: `map, order, groupBy, filter, join, select`



Actions

- **Actions** trigger Spark to compute a result from a series of transformations.

```
dataRDD = sc.parallelize(["Alice", "Bob", "Carol", "Daniel"], 3)
nameLen = dataRDD.map(lambda s: len(s))
nameLen.collect()
```

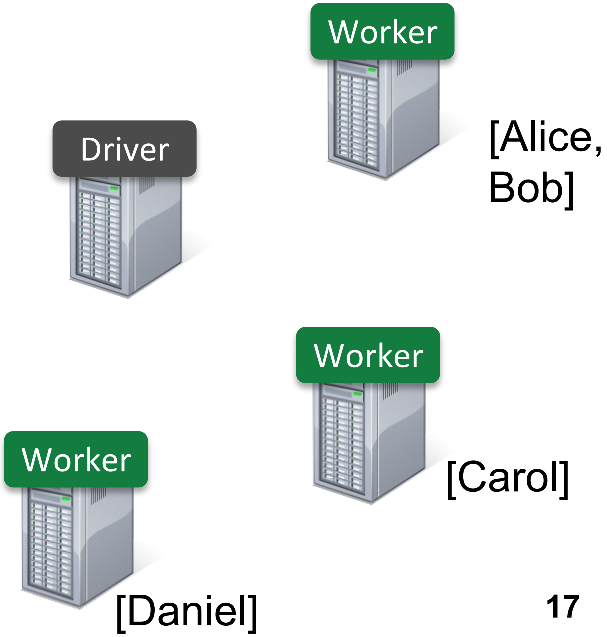
```
[5, 3, 5, 6]
```

- collect() here is an action.
  - It is the action that asks Spark to retrieve all elements of the RDD to the driver node.
- Examples of actions: `show, count, save, collect`

Distributed Processing

```python
# Create an RDD: length of names

dataRDD = sc.parallelize(["Alice", "Bob", "Carol", "Daniel"], 3)

nameLen = dataRDD.map(lambda s: len(s))

nameLen.collect()
```

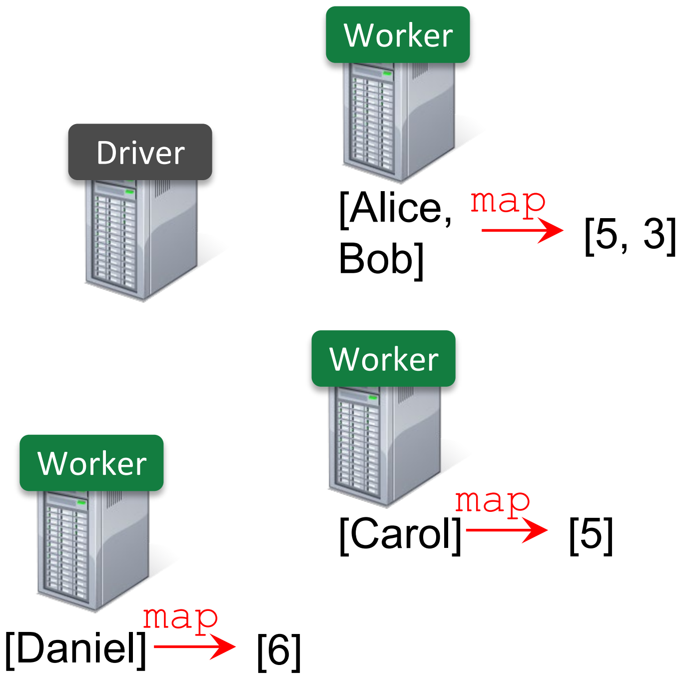
- As we previously said, RDDs are actually distributed across machines.
- Thus, the transformations and actions are executed in parallel. The results are only sent to the driver in the final step.



Distributed Processing

```
# Create an RDD: length of names

dataRDD = sc.parallelize(["Alice", "Bob", "Carol", "Daniel"], 3)

nameLen = dataRDD.map(lambda s: len(s))

nameLen.collect()
```

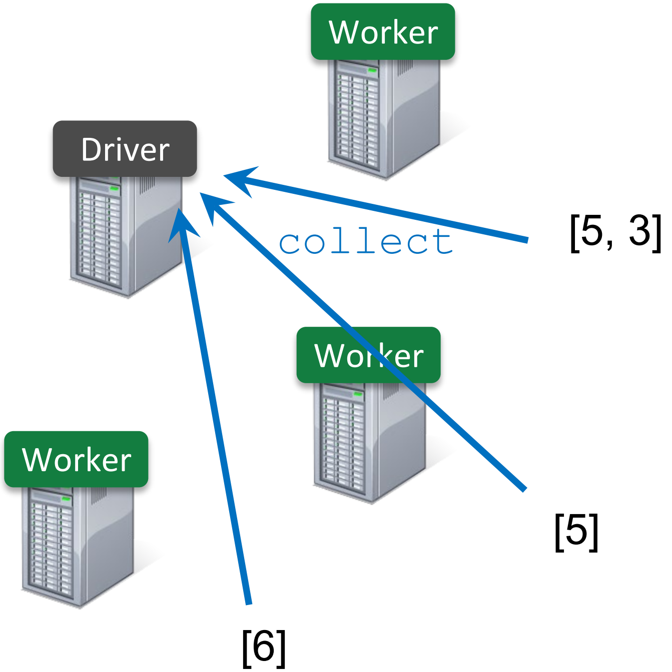
- As we previously said, RDDs are actually distributed across machines.
- Thus, the transformations and actions are executed in parallel. The results are only sent to the driver in the final step.



Distributed Processing

```
# Create an RDD: length of names
dataRDD = sc.parallelize(["Alice", "Bob", "Carol", "Daniel"], 3)
nameLen = dataRDD.map(lambda s: len(s))
nameLen.collect()
```

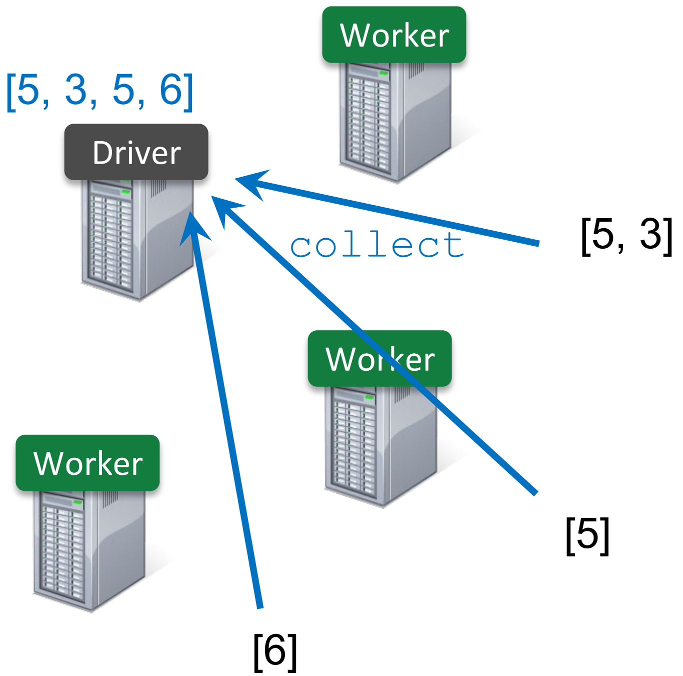
- As we previously said, RDDs are actually distributed across machines.
- Thus, the transformations and actions are executed in parallel. The results are only sent to the driver in the final step.



Distributed Processing

```
# Create an RDD: length of names
dataRDD = sc.parallelize(["Alice", "Bob", "Carol", "Daniel"], 3)
nameLen = dataRDD.map(lambda s: len(s))
nameLen.collect()
```

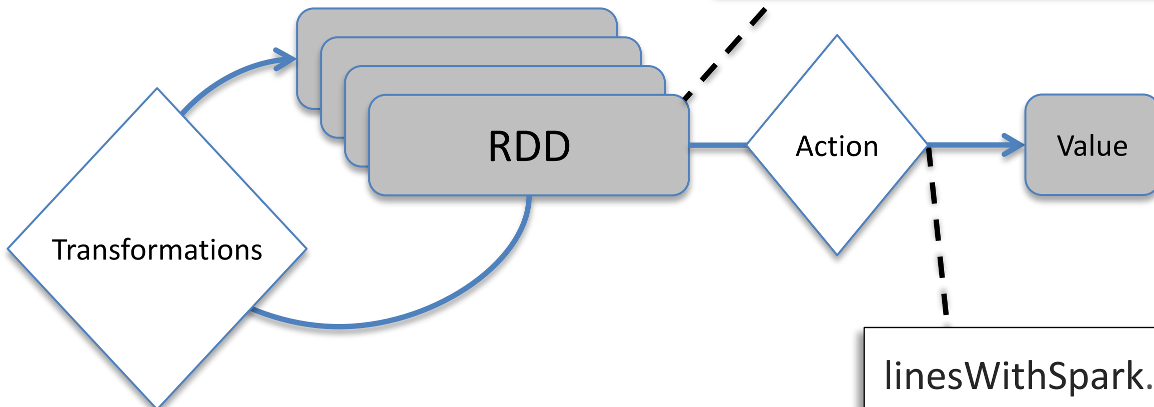
- As we previously said, RDDs are actually distributed across machines.
- Thus, the transformations and actions are executed in parallel. The results are only sent to the driver in the final step.



Working with RDDs

Note: this reads the file on each worker node in parallel, not on the driver node

```
textFile = sc.textFile("File.txt")
```

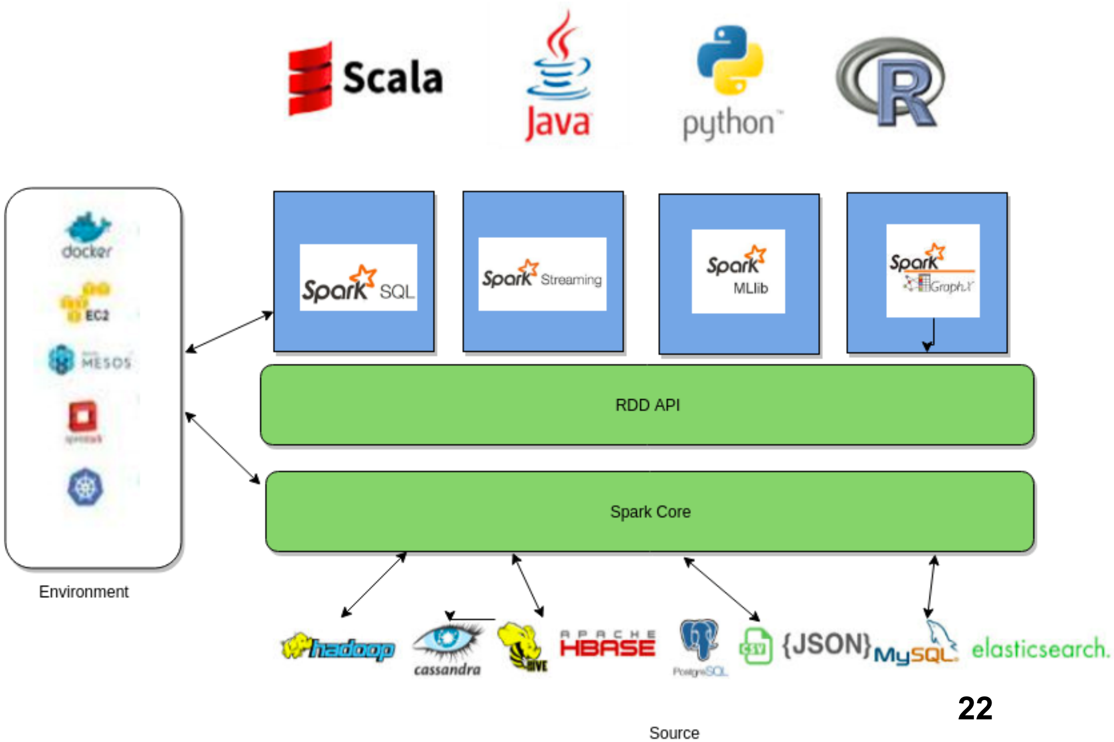



```
linesWithSpark = textFile.filter(lambda line:
"Spark" in line)
```

```
linesWithSpark.count()
74

linesWithSpark.first()
# Apache Spark
```

Today's Plan

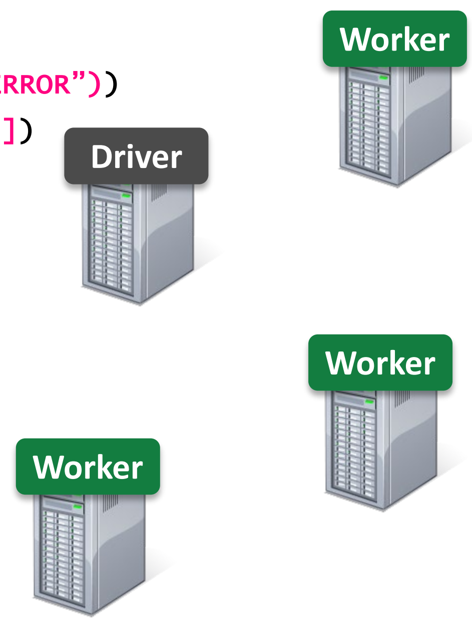
- Introduction and Basics
- Working with RDDs
- **Caching and DAGs**
- DataFrames and Datasets



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

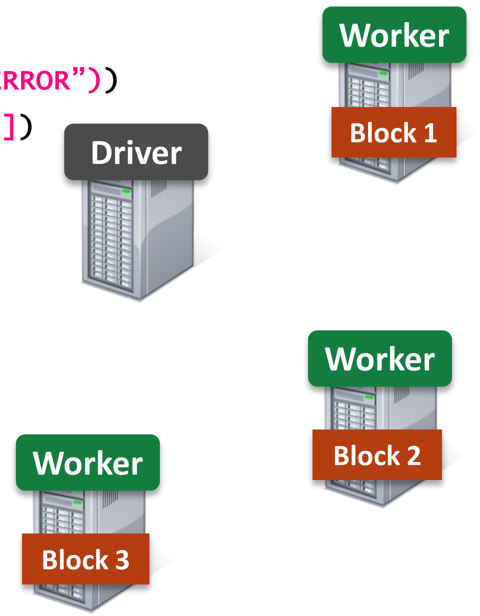
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

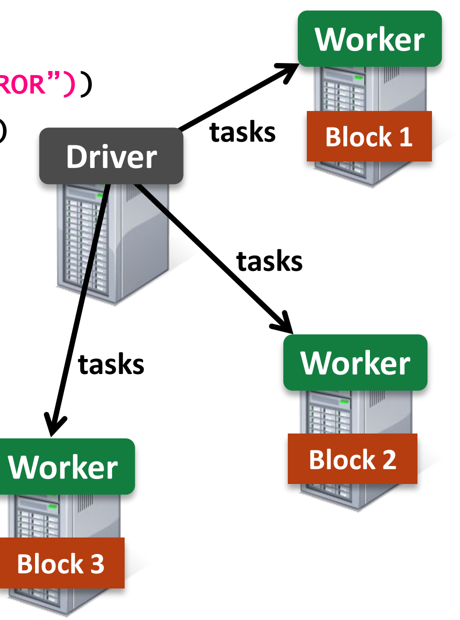
```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

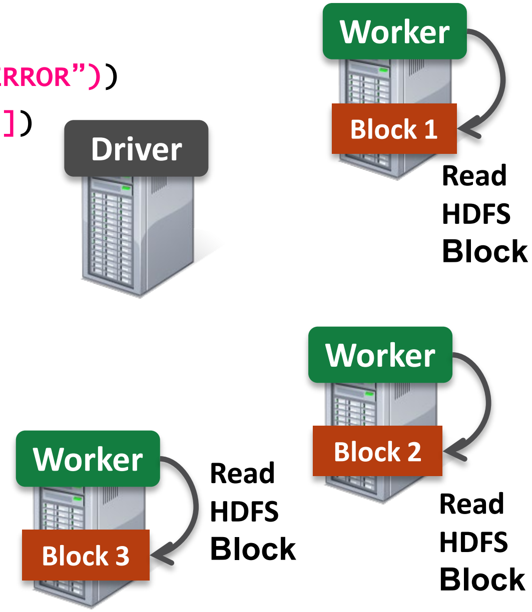
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

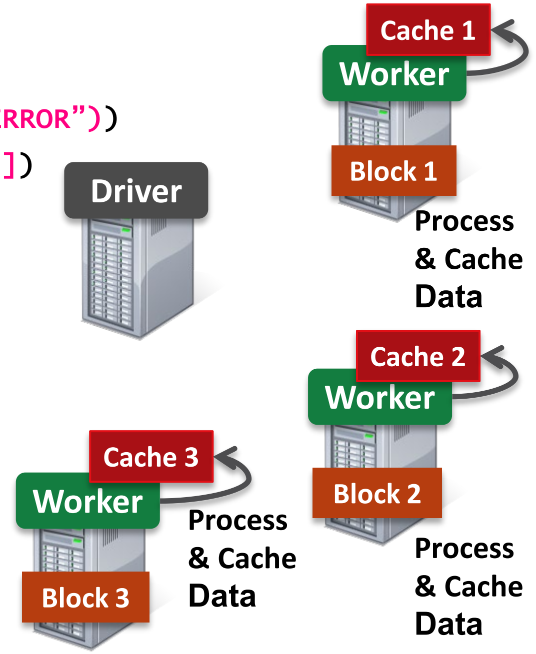
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

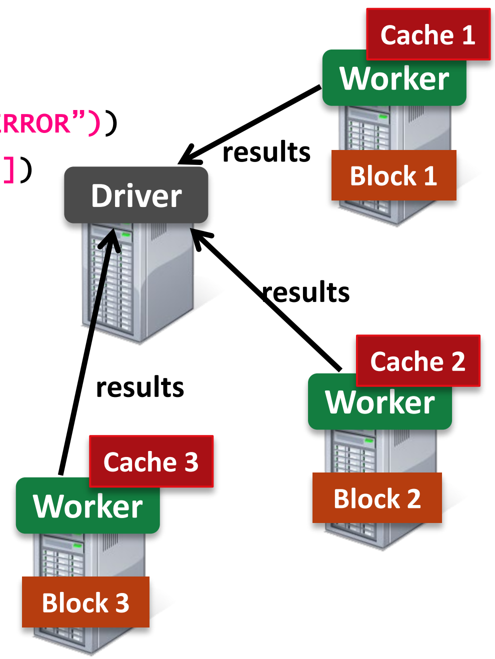
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

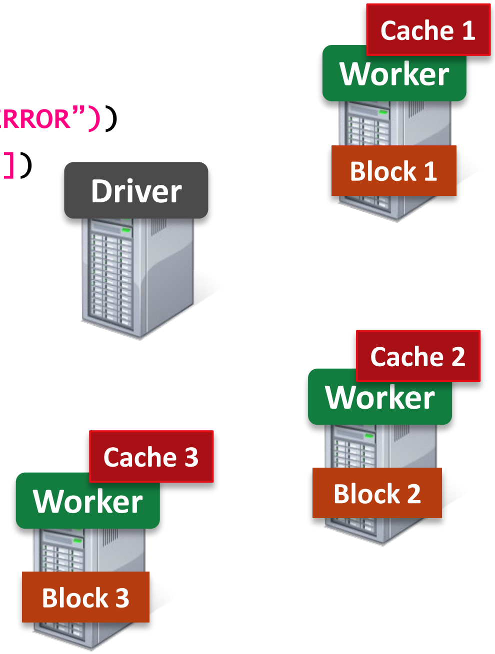
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

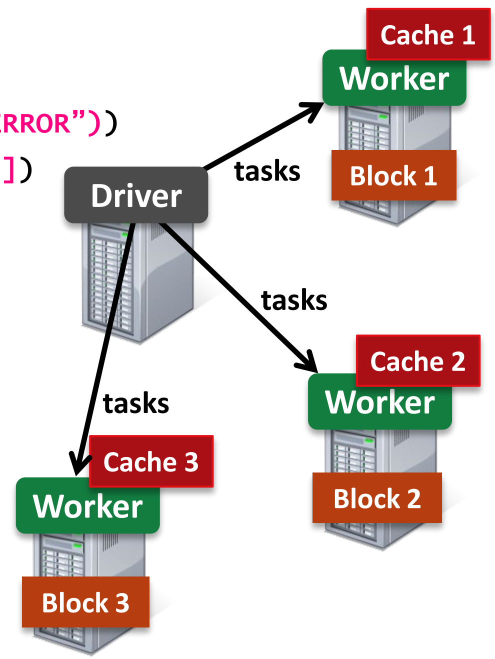
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

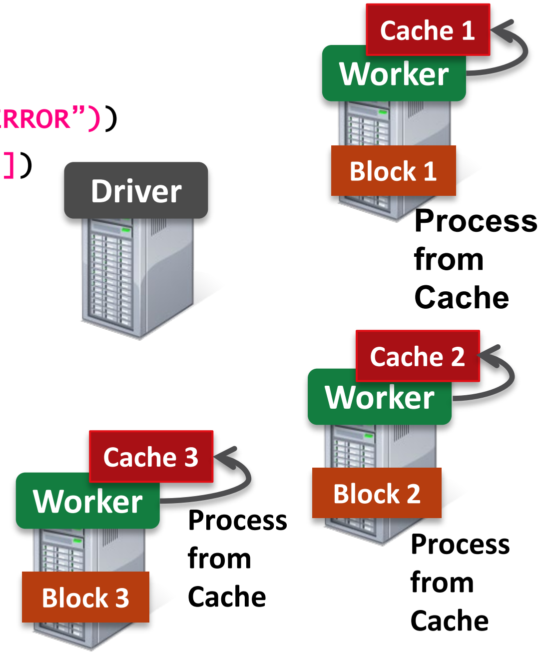
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

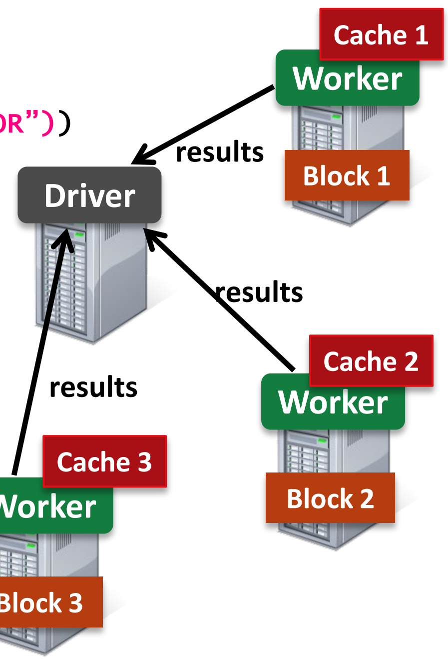
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Caching

Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Caching

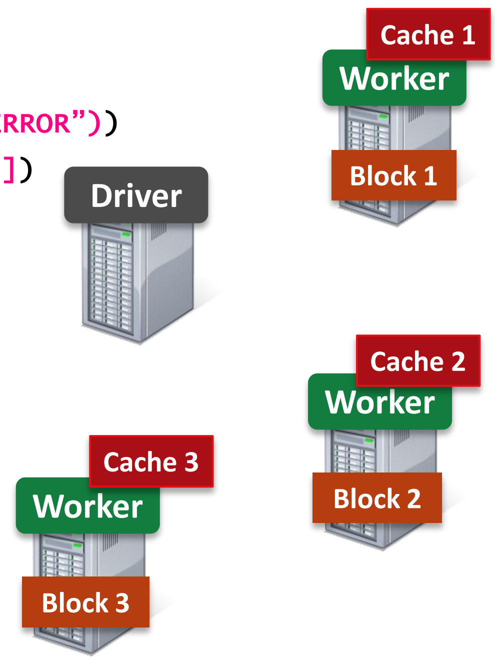
Log Mining example: Load error messages from a log into memory, then interactively search for various patterns

```
lines = sc.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()


messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Cache your data → Faster Results

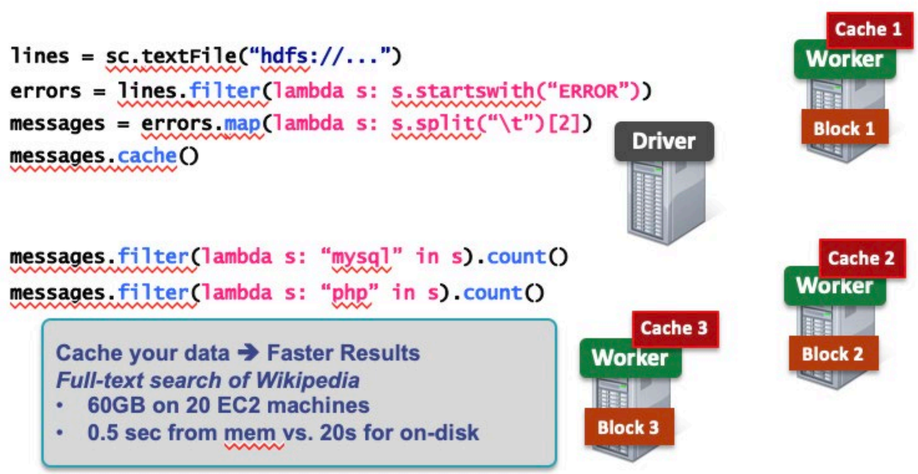
Full-text search of Wikipedia

- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk



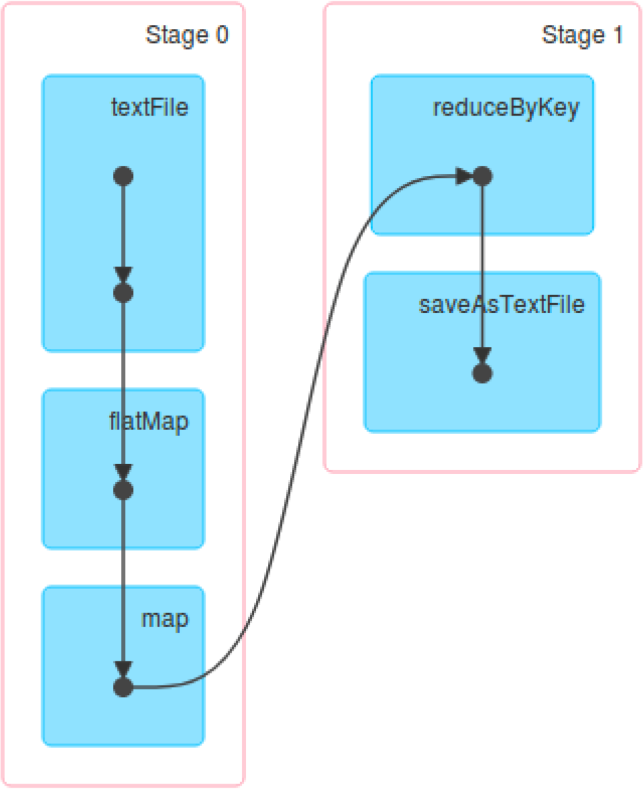
Caching

- `cache()`: saves an RDD to memory (of each worker node).
- `persist(options)`: can be used to save an RDD to memory, disk, or off-heap memory
- When should we cache or not cache an RDD?
  - When it is expensive to compute and needs to be re-used multiple times.
  - If worker nodes have not enough memory, they will evict the "least recently used" RDDs. So, be aware of memory limitations when caching.



Directed Acyclic Graph (DAG)

- Internally, Spark creates a graph ("directed acyclic graph") which represents all the RDD objects and how they will be transformed.
- Transformations construct this graph; actions trigger computations on it.

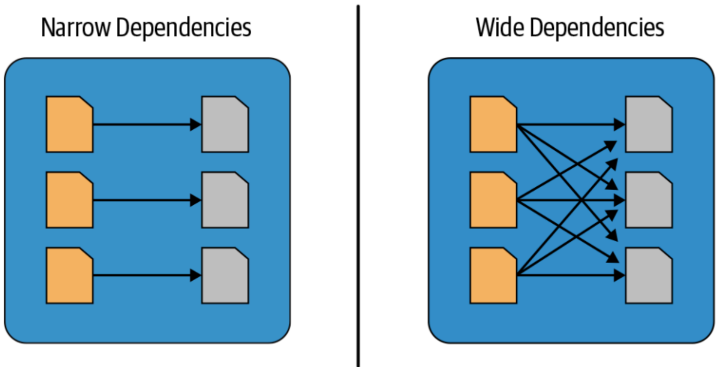


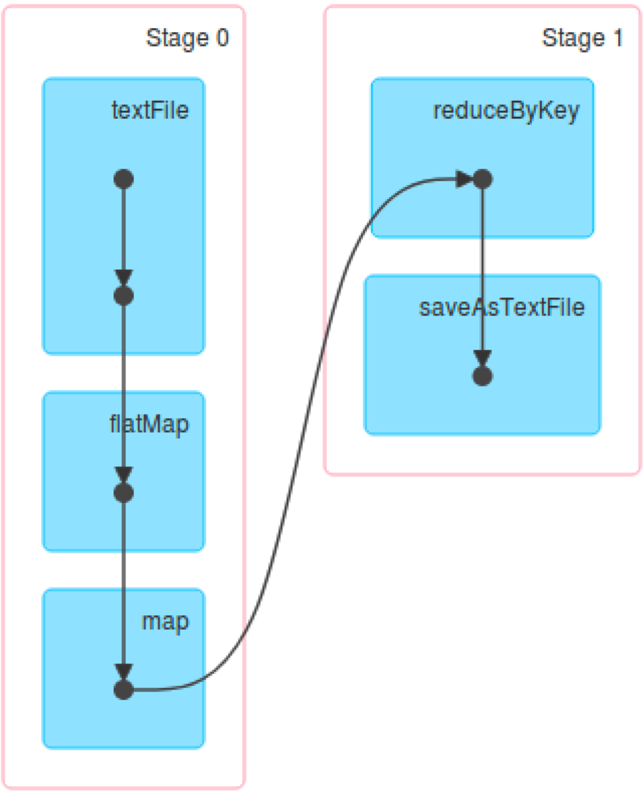

```
val file = sc.textFile("hdfs://...")

val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.save("...")
```

WordCount (Spark)

Narrow and Wide Dependencies

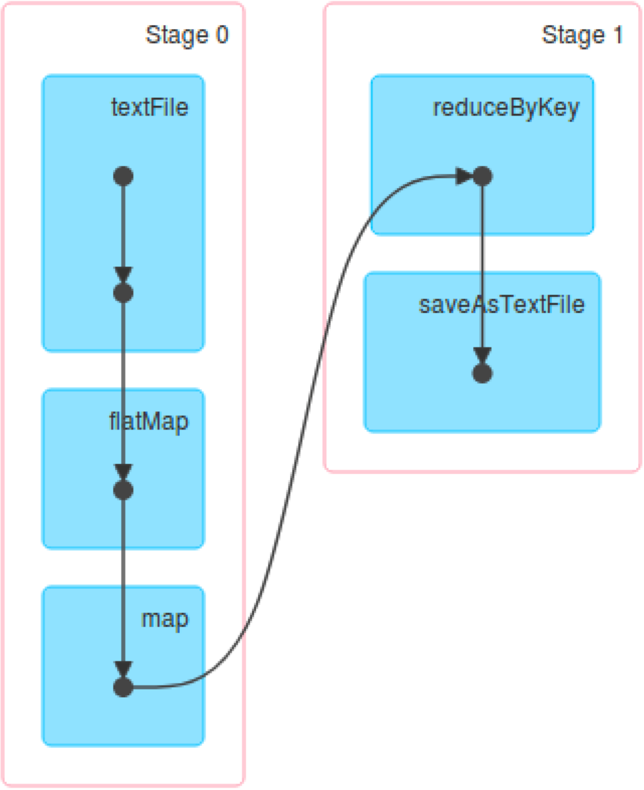
- **Narrow dependencies** are where each partition of the parent RDD is used by at most 1 partition of the child RDD
  - E.g. map, flatMap, filter, contains
- **Wide dependencies** are the opposite (each partition of parent RDD is used by multiple partitions of the child RDD)
  - E.g. reduceByKey, groupBy, orderBy
- In the DAG, consecutive narrow dependencies are grouped together as "**stages**".
- **Within stages**, Spark performs consecutive transformations on the same machines.
- **Across stages**, data needs to be **shuffled**, i.e. exchanged across partitions, in a process very similar to map-reduce, which involves writing intermediate results to disk
- Minimizing shuffling is good practice for improving performance.





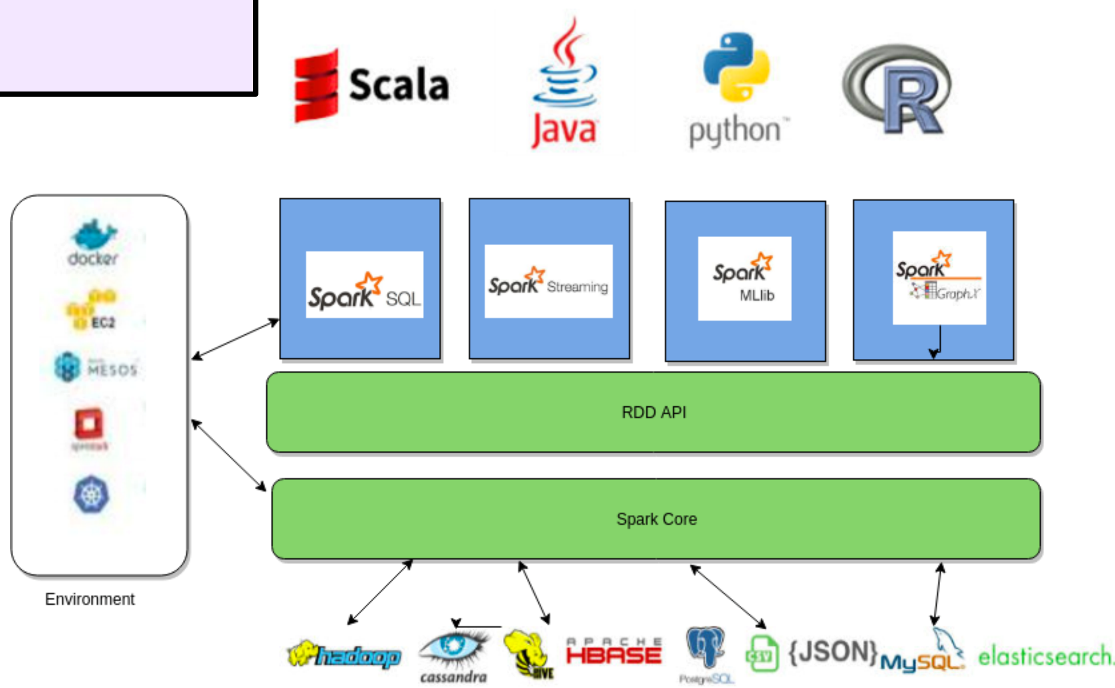
Lineage and Fault Tolerance

- Unlike Hadoop, Spark does not use replication to allow fault tolerance. Why?
  - Spark tries to store all the data in memory, not disk. Memory capacity is much more limited than disk, so simply duplicating all data is expensive.
- **Lineage approach**: if a worker node goes down, we replace it by a new worker node, and use the graph (DAG) to recompute the data in the lost partition.
  - Note that we only need to recompute the RDDs from the lost partition.



Today's Plan

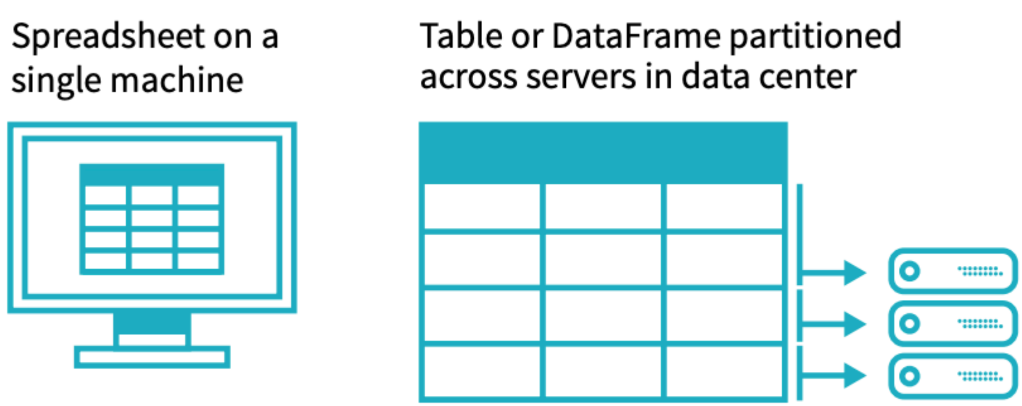
- Introduction and Basics
- Working with RDDs
- Caching and DAGs
- **DataFrames and Datasets**



Source

DataFrames

- A DataFrame represents a table of data, similar to tables in SQL, or DataFrames in pandas.
- Compared to RDDs, this is a higher level interface, e.g. it has transformations that resemble SQL operations.
  - DataFrames (and Datasets) are the recommended interface for working with Spark – they are easier to use than RDDs and almost all tasks can be done with them, while only rarely using the RDD functions.
  - However, all DataFrame operations are still ultimately compiled down to RDD operations by Spark.



DataFrames: example

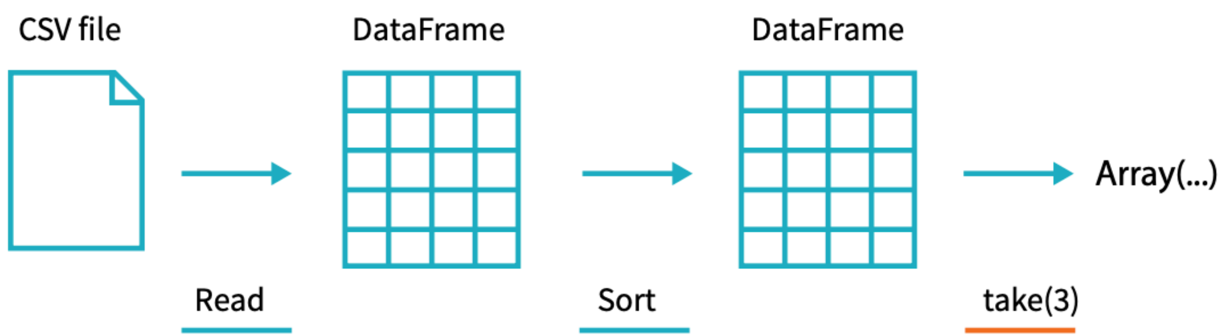
```
flightData2015 = spark\
.read\
.option("inferSchema", "true")\
.option("header", "true")\
.csv("/mnt/defg/flight-data/csv/2015-summary.csv")
```

- Reads in a DataFrame from a CSV file.

```
flightData2015.sort("count").take(3)
```

- Sorts by ‘count’ and output the first 3 rows (action)

```
Array([United States,Romania,15], [United States,Croatia...
```



DataFrames: transformations

- An easy way to transform DataFrames is to use SQL queries. This takes in a DataFrame and returns a DataFrame (the output of the query).

```
flightData2015.createOrReplaceTempView("flight_data_2015")

maxSql = spark.sql("""

SELECT DEST_COUNTRY_NAME, sum(count) as destination_total

FROM flight_data_2015

GROUP BY DEST_COUNTRY_NAME

ORDER BY sum(count) DESC

LIMIT 5

""")

maxSql.collect()
```

DataFrames: DataFrame interface

- We can also run the exact same query as follows:

```
from pyspark.sql.functions import desc

flightData2015\

.groupBy("DEST_COUNTRY_NAME")\

.sum("count")\

.withColumnRenamed("sum(count)", "destination_total")\

.sort(desc("destination_total"))\

.limit(5)\

.collect()
```

- Generally, these transformation functions (groupBy, sort, …) take in either strings or “column objects”, which represent columns.
  - For example, “desc” here returns a column object.

Datasets

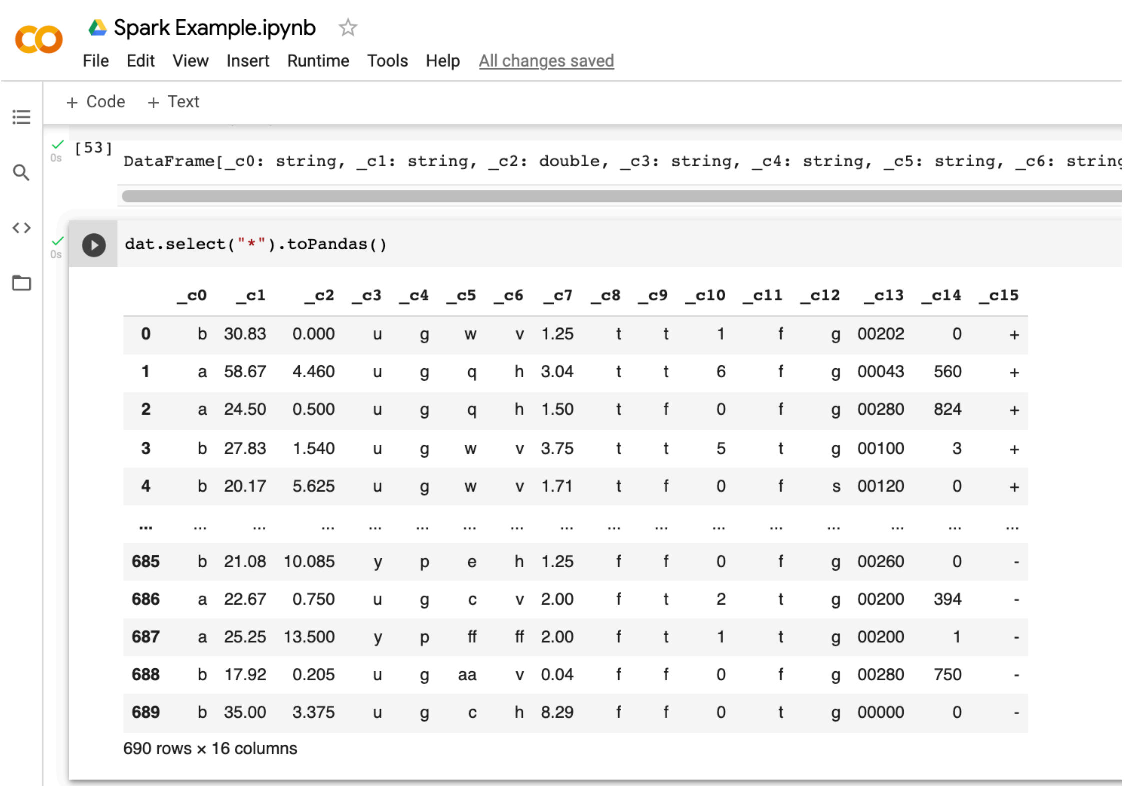
- Datasets are similar to DataFrames, but are type-safe.
  - In fact, in Spark (Scala), DataFrame is just an alias for Dataset[Row]
  - However, Datasets are not available in Python and R, since these are dynamically typed languages

```
case class Flight(DEST_COUNTRY_NAME: String, ORIGIN_COUNTRY_NAME: String, count: BigInt)

val flightsDF = spark.read.parquet("/mnt/defg/flight-data/parquet/2010-summary.parquet/")

val flights = flightsDF.as[Flight]

flights.collect()
```

- The Dataset `flights` is type safe – its type is the “Flight” class.
- Now when calling `collect()`, it will also return objects of the “Flight” class, instead of Row objects.

Example: Spark Notebook in Google Colab

- To experiment with simple Spark commands without needing to install / setup anything on your computer, you can run Spark on Google Colab
- See the simple example notebook at https://colab.research.google.com/drive/1qtNpkieNEUzyF2NnXTyqyGL3LQD1TVlI#scrollTo=pUgUMWYUKAU3



	_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9	_c10	_c11	_c12	_c13	_c14	_c15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.29	f	f	0	t	g	00000	0	-

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Python version 3.7.10 (default, Feb 26 2021 10:16:00)
Spark context Web UI available at http://192.168.79.2:4040
```

Example: Spark Notebooks in Databricks

- You need to sign up a Databricks community edition account (free)

Practical 1a: Sprark Basic I

This notebook is the end-to-end example, showing how to use DataFrame and Spark SQL for c

Source: https://github.com/databricks/LearningSparkV2

Cmd 2

Inspect location where the SF Fire Department Fire calls data set is stored in the public dataset

Cmd 3

```
%fs ls /databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv
```

Table

	path	name	size
1	dbfs:/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv	sf-fire-calls.csv	1137925

1 row | 2.04 seconds runtime

Command took 2.04 seconds -- by aixin@comp.nus.edu.sg at 2/10/2023, 2:32:32 PM on Test

Cmd 4

Define the location of the public dataset on the S3 bucket

Cmd 5

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

sf_fire_file = "/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.c
```

Command took 0.09 seconds -- by aixin@comp.nus.edu.sg at 2/10/2023, 2:32:45 PM on Test

Practical 1b: Spark Basic II

The power of Spark SQL is that it contains many DataFrame Operations (also known as Untyped Dataset operations for DataFrames and Spark SQL.

Source: https://github.com/databricks/LearningSparkV2

Cmd 2

```
from pyspark.sql.functions import expr

# Set File Paths
delays_path = "/databricks-datasets/learning-spark-v2/flights/departuredelays.csv"
airports_path = "/databricks-datasets/learning-spark-v2/flights/airport-codes-na.txt"

# Obtain airports dataset
airports = spark.read.options(header="true", inferSchema="true", sep="\t").csv(airports
airports.createOrReplaceTempView("airports_na")

# Obtain departure Delays data
delays = spark.read.options(header="true").csv(delays_path)
delays = (delays
          .withColumn("delay", expr("CAST(delay as INT) as delay"))
          .withColumn("distance", expr("CAST(distance as INT) as distance")))

delays.createOrReplaceTempView("departureDelays")

# Create temporary small table
foo = delays.filter(expr("""
            origin == 'SEA' AND
            destination == 'SFO' AND
            date like '01010%' AND
            delay > 0"""))

foo.createOrReplaceTempView("foo")
```

- Source: https://github.com/databricks/LearningSparkV2

Demo_1: Spark Web UI

```
df1 = spark.range(2, 10000000, 2)
df2 = spark.range(2, 10000000, 4)
df3 = df1.join(df2, ["id"])
df3.count()
```

▾ (4) Spark Jobs
- ▾ Job 0 View (Stages: 1/1)
  - Stage 1: 8/8
- ▾ Job 1 View (Stages: 1/1)
  - Stage 0: 8/8
- ▾ Job 2 View (Stages: 1/1, 2 skipped)
  - Stage 2: 0/8 skipped
  - Stage 3: 0/8 skipped
  - Stage 4: 8/8
- ▾ Job 3 View (Stages: 1/1, 3 skipped)
  - Stage 5: 0/8 skipped
  - Stage 6: 0/8 skipped
  - Stage 7: 0/8 skipped
  - Stage 8: 1/1

▸ df1: pyspark.sql.dataframe.DataFrame = [id: long]
▸ df2: pyspark.sql.dataframe.DataFrame = [id: long]
▸ df3: pyspark.sql.dataframe.DataFrame = [id: long]

```
Out[1]: 2500000
```

```
df1.show(10)
```

▸ (1) Spark Jobs

```
+---+
| id|
+---+
|  2|
|  4|
|  6|
|  8|
| 10|
| 12|
| 14|
| 16|
| 18|
| 20|
+---+
```

```
df2.show(10)
```

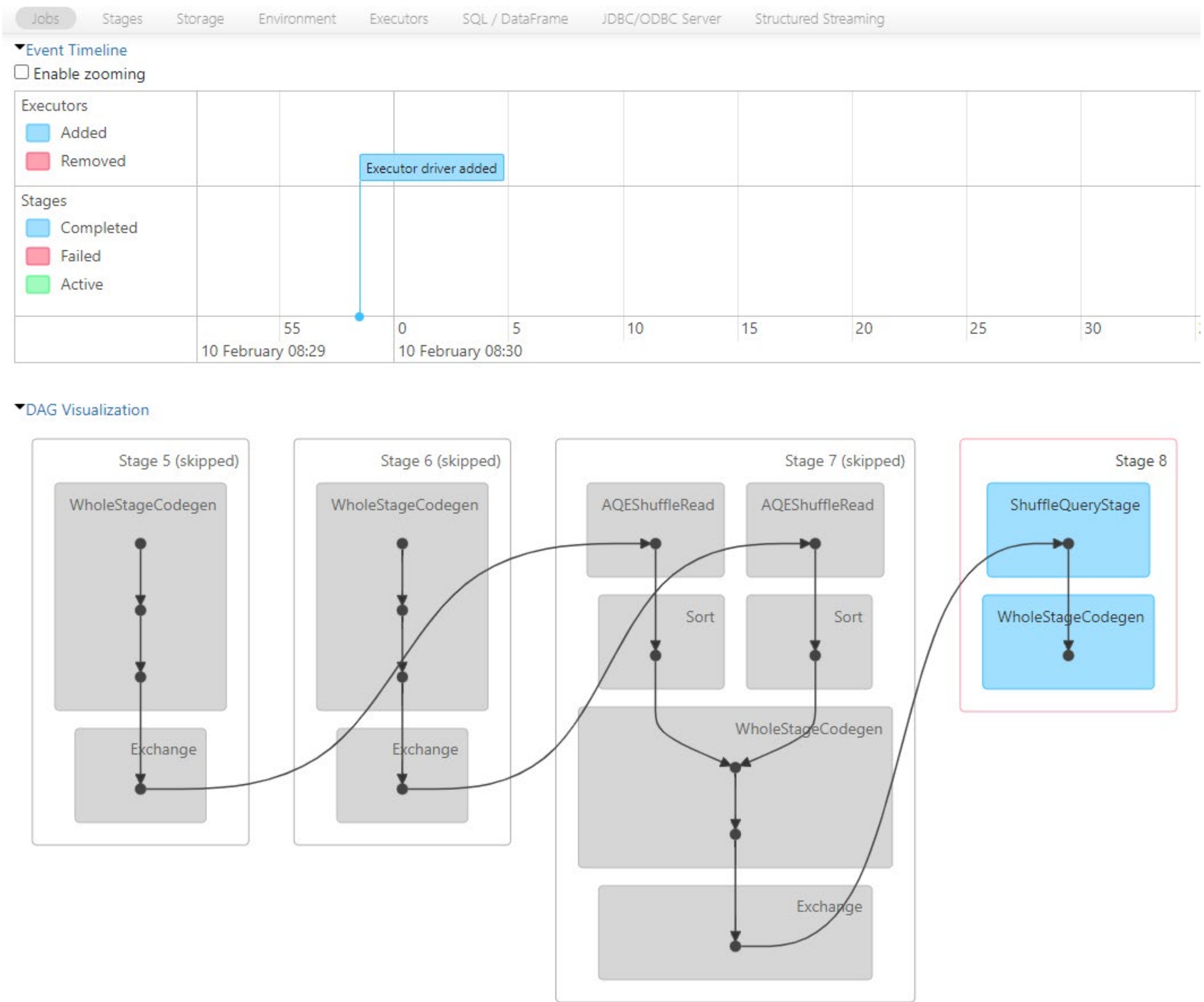
▸ (1) Spark Jobs

```
+---+
| id|
+---+
|  2|
|  6|
| 10|
| 14|
| 18|
| 22|
| 26|
| 30|
| 34|
| 38|
+---+
```

```
df3.show(10)
```

▸ (3) Spark Jobs

```
+---+
| id|
+---+
| 22|
| 26|
| 34|
| 50|
| 54|
| 94|
|110|
|126|
|130|
|190|
+---+
```


Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
JDBC/ODBC Server
Structured Streaming
Event Timeline
Enable zooming
Executors
Added
Removed
Executor driver added
Stages
Completed
Failed
Active
55
10 February 08:29
0
10 February 08:30
5
10
15
20
25
30
DAG Visualization
Stage 5 (skipped)
WholeStageCodegen
Exchange
Stage 6 (skipped)
WholeStageCodegen
Exchange
Stage 7 (skipped)
AQEShuffleRead
AQEShuffleRead
Sort
Sort
WholeStageCodegen
Exchange
Stage 8
ShuffleQueryStage
WholeStageCodegen

Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | JDBC/ODBC Server | Structured Streaming

Stages for All Jobs

Completed Stages: 4
Skipped Stages: 5

Fair Scheduler Pools (1)

Pool Name	Minimum Share	Pool Weight	Active Stages	Running Tasks	SchedulingMode
default	0	1	0	0	FIFO

Completed Stages (4)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. G

Stage Id ▾	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
8	3168022962293687376	df1 = spark.range(2, 10000000, 2) df2 = spark.r... count at NativeMethodAccessorImpl.java:0 +details	2023/02/10 08:31:15	0.3 s	1/1			472.0 B	
4	3168022962293687376	df1 = spark.range(2, 10000000, 2) df2 = spark.r... $anonfun$withThreadLocalCaptured$1 at CompletableFuture.java:1604 +details	2023/02/10 08:31:05	9 s	8/8			36.5 MiB	472.0 B
1	3168022962293687376	df1 = spark.range(2, 10000000, 2) df2 = spark.r... $anonfun$withThreadLocalCaptured$1 at CompletableFuture.java:1604 +details	2023/02/10 08:30:53	3 s	8/8				12.2 MiB
0	3168022962293687376	df1 = spark.range(2, 10000000, 2) df2 = spark.r... $anonfun$withThreadLocalCaptured$1 at CompletableFuture.java:1604 +details	2023/02/10 08:30:52	8 s	8/8				24.3 MiB

Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | JDBC/ODBC Server | Structured Streaming

Executors

Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(1)	0	0.0 B / 3.9 GiB	0.0 B	8	0	0	25	25	5.6 min (4 s)	0.0 B	36.5 MiB	36.5 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	0.0 B / 3.9 GiB	0.0 B	8	0	0	25	25	5.6 min (4 s)	0.0 B	36.5 MiB	36.5 MiB	0

Executors

Show 20 entries

Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump	Heap Histogram	Exec Loss Reason
driver	10.172.213.39:43725	Active	0	0.0 B / 3.9 GiB	0.0 B	8	0	0	25	25	5.6 min (4 s)	0.0 B	36.5 MiB	36.5 MiB	Thread Dump	Heap Histogram	

Showing 1 to 1 of 1 entries

Previous Next

Jobs Stages Storage Environment Executors SQL / DataFrame JDBC/ODBC Server Structured Streaming

SQL / DataFrame

Completed Queries: 5

Completed Queries (5)

Page: 1

1 Pages. Jump to 1 . Show 100 item

ID ▾	Description	Submitted	Duration	Job IDs	Sub Execution IDs
4	show tables in `default` +details	2023/02/10 08:31:23	31 ms		
3	show tables in `default` +details	2023/02/10 08:31:22	0.1 s		
2	show databases +details	2023/02/10 08:31:21	52 ms		
1	df1 = spark.range(2, 10000000, 2) df2 = spark.r... +details	2023/02/10 08:30:48	27 s	[0][1][2][3]	
0	show databases +details	2023/02/10 08:30:39	41 s		

Jobs Stages Storage Environment Executors SQL / DataFrame JDBC/ODBC Server Structured Streaming

Storage

Parquet IO Cache

Data Read from External Filesystem (All Formats)	Data Read from IO Cache (Cache Hits, Compressed)	Data Written to IO Cache (Compressed)	Cache Misses (Compressed)	True Cache Misses	Partial Cache Misses	Rescheduling Cache Misses	Cache Hit Ratio	Number of Local Scan Tasks	Number of Rescheduled Scan Tasks	Cache Metadata Manager Peak Disk Usage
0.0 B	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B	0 %	0	0	0.0 B

Jobs Stages Storage Environment Executors SQL / DataFrame JDBC/ODBC Server Structured Streaming

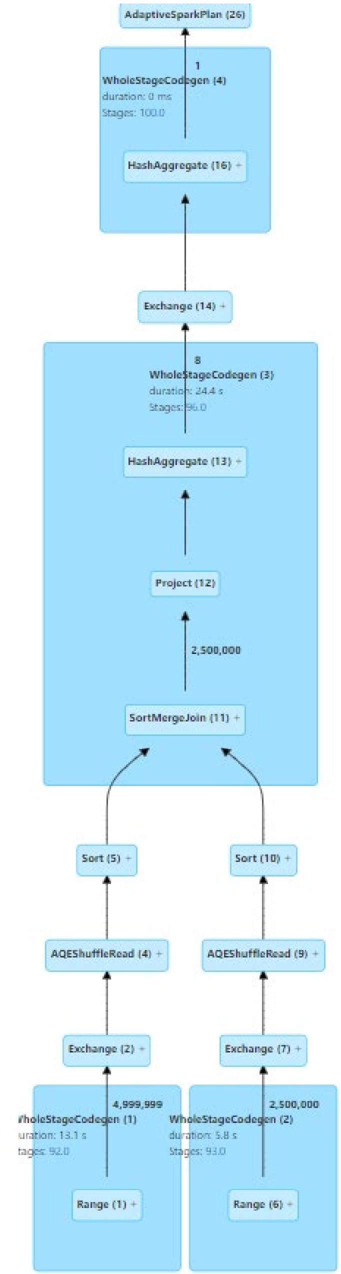
Environment

Runtime Information

Name	Value
Java Home	/usr/lib/jvm/zulu8-ca-amd64/jre
Java Version	1.8.0_345 (Azul Systems, Inc.)
Scala Version	version 2.12.14

Spark Properties

Name	Value
eventLog.rolloverIntervalSeconds	900
libraryDownload.sleepIntervalSeconds	5
libraryDownload.timeoutSeconds	180
spark.akka.frameSize	256
spark.app.id	local-1676017796375
spark.app.name	Databricks Shell
spark.app.startTime	1676017791391
spark.cleaner.referenceTracking.blocking	false
spark.databricks.acl.client	com.databricks.spark.sql.acl.client.SparkSqlAclClient
spark.databricks.acl.provider	com.databricks.sql.acl.ReflectionBackedAclProvider
spark.databricks.acl.scim.client	com.databricks.spark.sql.acl.client.DriverToWebappScimClient
spark.databricks.automl.serviceEnabled	true
spark.databricks.cloudProvider	AWS
spark.databricks.cloudfetch.hasRegionSupport	true
spark.databricks.cloudfetch.requesterClassName	*********(redacted)
spark.databricks.clusterSource	UI



Demo_2: Caching Data

```
from pyspark.sql.functions import col

df = spark.range(1 * 10000000).toDF("id").withColumn("square", col("id") * col("id"))
df.cache().count()
```

▸ (3) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [id: long, square: long]

Out[1]: 10000000

Command took 10.90 seconds -- by aixin@comp.nus.edu.sg at 8/24/2023, 3:40:21 PM on MyCluster

Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | JDBC/ODBC Server | Structured Streaming

Storage

Parquet IO Cache

Data Read from External Filesystem (All Formats)	Data Read from IO Cache (Cache Hits, Compressed)	Data Written to IO Cache (Compressed)	Cache Misses (Compressed)	True Cache Misses	Rescheduling Cache Misses	Cache Hit Ratio	Number of Local Scan Tasks	Number of Rescheduled Scan Tasks	Cache Metadata Manager Peak Disk Usage
172.3 MiB	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B	0 %	0	0	0.0 B

RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
190	*(1) Project [id#814L, (id#814L * id#814L) AS square#818L] +- *(1) Range (0, 10000000, step=1, splits=8)	Disk Memory Deserialized 1x Replicated	8	100%	86.2 MiB	0.0 B

```
df.count()
```

▸ (2) Spark Jobs

Out[2]: 10000000

Command took 1.26 seconds -- by aixin@comp.nus.edu.sg at 8/24/2023, 3:40:46 PM on MyCluster

Acknowledgements

- CS4225 slides by He Bingsheng and Bryan Hooi
- Jules S. Damji, Brooke Wenig, Tathagata Das & Denny Lee, “Learning Spark: Lightning-Fast Data Analytics”
- Databricks, “The Data Engineer’s Guide to Spark”
- https://www.pinterest.com/pin/739364463807740043/
- https://colab.research.google.com/github/jmbanda/BigDataProgramming_2019/blob/master/Chapter_5_Loading_and_Saving_Data_in_Spark.ipynb
- https://untitled-life.github.io/blog/2018/12/27/wide-vs-narrow-dependencies/