CS 649 Big Data: Tools and Methods
Spring Semester, 2022
Doc 10 Spark Intro
Feb 10, 2022

Spark

Created at UC Berkeley's AMPLab

2009 Project started
2014 May - 1.0
2016 July - 2.0.2
2017 July - 2.2.0
2020 June - 3.0
2021 March - 3.1.1

Programming interface for
- Java, Python, Scala, R

Interactive shell for
- Python, Scala, R (experimental)

Runs on
- Linux, Mac, Windows

Cluster manager
- Native Spark cluster
- Hadoop YARN
- Apache Mesos

File System
- HDFS
- MapR File System
- Cassandra
- OpenStack Swift
- S3

Pseudo-Distributed Mode
- Single machine
- Uses local file system

Time Line

1991 - Java project started

1995 - Java 1.0 released, Design Patterns book published

2000 - Java 3

2001 - Scala project started

2002 - Nutch started

2004 - Google MapReduce paper

Scala version 1 released

2005 - F# released

2006 - Hadoop split from Nutch

Scala version 2 released

2007 - Clojure released

2009 - Spark project started

2012 - Hadoop 1.0

2014 - Spark 1.0

Spark Word Count - Python

```
from __future__ import print_function
import sys
from pyspark.sql import SparkSession

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: wordcount <file>", file=sys.stderr)
        sys.exit(-1)

spark = SparkSession.builder.appName("PythonWordCount").getOrCreate()

lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
counts = lines.flatMap(lambda x: x.split(' ')) \
                  .map(lambda x: (x, 1)) \
                  .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")

spark.stop()
```

Scala

```
object SparkWordCount {
  def main(args: Array[String]) {
    val spark = SparkSession.builder.appName("Spark Pi").getOrCreate()
    val textFile = sc.textFile("hdfs://...")
    val counts = textFile.flatMap(line => line.split(" "))
                         .map(word => (word, 1))
                        .reduceByKey(_ + _)
    counts.saveAsTextFile("hdfs://...")
    spark.stop()
  }
}
```

Spark Word Count - Java

```
public final class JavaWordCount {
    private static final Pattern SPACE = Pattern.compile(" ");
    public static void main(String[] args) throws Exception {

    if (args.length < 1) {
      System.err.println("Usage: JavaWordCount <file>");
      System.exit(1);
    }

    SparkSession spark = SparkSession.builder().appName("JavaWordCount").getOrCreate();

    JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
    JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
    JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
    JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);

    counts.saveAsTextFile("hdfs://...");
    spark.stop();
  }
}
```

Hadoop Word Count - Map

```
public class WordCount {

    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

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

Hadoop Word Count - Reduce

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

Hadoop Word Count - Main

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

Python vs Scala on Spark

Scala is faster that Python

- But that is not so important here
- Most of the computation on Spark is done in Spark

Using Python with Spark

- Python data has to be
  - Converted between Python format and Scala/Java format
  - Sent between Python process and JVM

Installing & Running Spark

Two different ways to install PySpark

- pip
- download from Spark site

Two different ways to run spark

- Command line
- Jupyter notebook

Installing PySpark using Anaconda

pip install pyspark

make sure using Anaconda pip

Need Java 8 installed

Sample Program

```
import pyspark
import random
sc = pyspark.SparkContext(appName="Pi")
num_samples = 100000
def inside(p):
  x, y = random.random(), random.random()
  return x*x + y*y < 1
count = sc.parallelize(range(0, num_samples)).filter(inside).count()
pi = 4 * count / num_samples
print(pi)
sc.stop()
```

Installing Spark - Java/Scala/Python

http://spark.apache.org/downloads.html

Download Apache Spark™

1. Choose a Spark release: 3.2.1 (Jan 26 2022)
2. Choose a package type: Pre-built for Apache Hadoop 3.3 and later
3. Download Spark: spark-3.2.1-bin-hadoop3.2.tgz
4. Verify this release using the 3.2.1 signatures, checksums and project release KEYS.

Read the Readme.md file

Helps to set your path

https://spark.apache.org/docs/latest/

Running Spark in Jupyter Notebook

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .master("local") \
    .appName("Print") \
    .getOrCreate()
print(spark.range(5000).selectExpr("sum(id)").collect())
```

In [1]:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .master("local") \
    .appName("Print") \
    .getOrCreate()
print(spark.range(5000).selectExpr("sum(id)").collect())
```

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLev
el).
22/02/10 14:08:23 WARN NativeCodeLoader: Unable to load native-hadoop library for your p
latform... using builtin-java classes where applicable
[Stage 0:>                                                          (0 + 1) / 1]
```

```
[Row(sum(id)=12497500)]
```

Running From Command Line

File: print.py

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .master("local") \
    .appName("Print") \
    .getOrCreate()
print(spark.range(5000).selectExpr("sum(id)").collect())
```

```
➜ spark-submit print.py
```

Running From Command Line - Output

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
22/02/10 14:14:26 INFO SparkContext: Running Spark version 3.2.1
22/02/10 14:14:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
22/02/10 14:14:26 INFO ResourceUtils: ==============================================================
22/02/10 14:14:26 INFO ResourceUtils: No custom resources configured for spark.driver.
22/02/10 14:14:26 INFO ResourceUtils: ==============================================================
22/02/10 14:14:26 INFO SparkContext: Submitted application: PySparkShell
22/02/10 14:14:26 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
22/02/10 14:14:26 INFO ResourceProfile: Limiting resource is cpu
22/02/10 14:14:26 INFO ResourceProfileManager: Added ResourceProfile id: 0
22/02/10 14:14:26 INFO SecurityManager: Changing view acls to: rwhitney
22/02/10 14:14:26 INFO SecurityManager: Changing modify acls to: rwhitney
22/02/10 14:14:26 INFO SecurityManager: Changing view acls groups to:
22/02/10 14:14:26 INFO SecurityManager: Changing modify acls groups to:
22/02/10 14:14:26 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(rwhitney); groups with view permissions: Set(); users  with modify permissions: Set(rwhitney); groups with modify permissions: Set()
22/02/10 14:14:27 INFO Utils: Successfully started service 'sparkDriver' on port 53950.
22/02/10 14:14:27 INFO SparkEnv: Registering MapOutputTracker
22/02/10 14:14:27 INFO SparkEnv: Registering BlockManagerMaster
22/02/10 14:14:27 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
22/02/10 14:14:27 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
22/02/10 14:14:27 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
22/02/10 14:14:27 INFO DiskBlockManager: Created local directory at /private/var/folders/cq/kvtrv716ss649swm0wj_7w40000gs/T/blockmgr-b9a9f1a2-47b7-4ae3-bf6a-0754649072a3
22/02/10 14:14:27 INFO MemoryStore: MemoryStore started with capacity 366.3 MiB
22/02/10 14:14:27 INFO SparkEnv: Registering OutputCommitCoordinator
22/02/10 14:14:27 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
22/02/10 14:14:27 INFO Utils: Successfully started service 'SparkUI' on port 4041.
22/02/10 14:14:27 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://10.130.118.90:4041
22/02/10 14:14:28 INFO Executor: Starting executor ID driver on host 10.130.118.90
22/02/10 14:14:28 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 53953.
22/02/10 14:14:28 INFO NettyBlockTransferService: Server created on 10.130.118.90:53953
22/02/10 14:14:28 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
22/02/10 14:14:28 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 10.130.118.90, 53953, None)
22/02/10 14:14:28 INFO BlockManagerMasterEndpoint: Registering block manager 10.130.118.90:53953 with 366.3 MiB RAM, BlockManagerId(driver, 10.130.118.90, 53953, None)
22/02/10 14:14:28 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 10.130.118.90, 53953, None)
22/02/10 14:14:28 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 10.130.118.90, 53953, None)
22/02/10 14:14:28 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir.
22/02/10 14:14:28 INFO SharedState: Warehouse path is 'file:/Users/rwhitney/Courses/649/Spring22/examples/spark-warehouse'.
22/02/10 14:14:33 INFO CodeGenerator: Code generated in 345.295334 ms
22/02/10 14:14:33 INFO SparkContext: Starting job: collect at /Users/rwhitney/Courses/649/Spring22/examples/print.py:6
22/02/10 14:14:33 INFO DAGScheduler: Got job 0 (collect at /Users/rwhitney/Courses/649/Spring22/examples/print.py:6) with 1 output partitions
22/02/10 14:14:33 INFO DAGScheduler: Final stage: ResultStage 0 (collect at /Users/rwhitney/Courses/649/Spring22/examples/print.py:6)
22/02/10 14:14:33 INFO DAGScheduler: Parents of final stage: List()
22/02/10 14:14:33 INFO DAGScheduler: Missing parents: List()
22/02/10 14:14:33 INFO DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[3] at collect at /Users/rwhitney/Courses/649/Spring22/examples/print.py:6), which has no missing parents
22/02/10 14:14:33 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 13.9 KiB, free 366.3 MiB)
22/02/10 14:14:34 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 6.0 KiB, free 366.3 MiB)
22/02/10 14:14:34 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 10.130.118.90:53953 (size: 6.0 KiB, free: 366.3 MiB)
22/02/10 14:14:34 INFO SparkContext: Created broadcast 0 from broadcast at DAGScheduler.scala:1478
22/02/10 14:14:34 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 0 (MapPartitionsRDD[3] at collect at /Users/rwhitney/Courses/649/Spring22/examples/print.py:6) (first 15 tasks are for partitions Vector(0))
22/02/10 14:14:34 INFO TaskSchedulerImpl: Adding task set 0.0 with 1 tasks resource profile 0
22/02/10 14:14:34 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0) (10.130.118.90, executor driver, partition 0, PROCESS_LOCAL, 4578 bytes) taskResourceAssignments Map()
22/02/10 14:14:34 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
22/02/10 14:14:34 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 1742 bytes result sent to driver
22/02/10 14:14:34 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 723 ms on 10.130.118.90 (executor driver) (1/1)
22/02/10 14:14:34 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
22/02/10 14:14:34 INFO DAGScheduler: ResultStage 0 (collect at /Users/rwhitney/Courses/649/Spring22/examples/print.py:6) finished in 1.113 s
22/02/10 14:14:34 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
22/02/10 14:14:34 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
22/02/10 14:14:34 INFO DAGScheduler: Job 0 finished: collect at /Users/rwhitney/Courses/649/Spring22/examples/print.py:6, took 1.198735 s
[Row(sum(id)=12497500)]
22/02/10 14:14:34 INFO SparkContext: Invoking stop() from shutdown hook
22/02/10 14:14:34 INFO SparkUI: Stopped Spark web UI at http://10.130.118.90:4041
22/02/10 14:14:35 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
22/02/10 14:14:35 INFO MemoryStore: MemoryStore cleared
22/02/10 14:14:35 INFO BlockManager: BlockManager stopped
22/02/10 14:14:35 INFO BlockManagerMaster: BlockManagerMaster stopped
22/02/10 14:14:35 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
22/02/10 14:14:35 INFO SparkContext: Successfully stopped SparkContext
22/02/10 14:14:35 INFO ShutdownHookManager: Shutdown hook called
22/02/10 14:14:35 INFO ShutdownHookManager: Deleting directory /private/var/folders/cq/kvtrv716ss649swm0wj_7w40000gs/T/spark-9151943b-e3c6-4b09-a1d7-ea9c22271d8c/pyspark-38f4a808-5007-4e16-9c2a-ed94c3aa1fa5
22/02/10 14:14:35 INFO ShutdownHookManager: Deleting directory /private/var/folders/cq/kvtrv716ss649swm0wj_7w40000gs/T/spark-8a433a7b-c520-4ac6-b130-7f453df05116
22/02/10 14:14:35 INFO ShutdownHookManager: Deleting directory /private/var/folders/cq/kvtrv716ss649swm0wj_7w40000gs/T/spark-9151943b-e3c6-4b09-a1d7-ea9c22271d8c
```

Interactive PySpark from Command Line

```
➜  pyspark
Python 3.9.7 (default, Sep 16 2021, 08:50:36)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/02/10 14:20:49 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
22/02/10 14:20:50 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.2.1
      /_/

Using Python version 3.9.7 (default, Sep 16 2021 08:50:36)
Spark context Web UI available at http://10.130.118.90:4041
Spark context available as 'sc' (master = local[*], app id = local-1644531650255).
SparkSession available as 'spark'.
>>>
```

Interactive PySpark from Command Line

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder \
...         .master("local") \
...         .appName("Print") \
...         .getOrCreate()
>>> print(spark.range(5000).selectExpr("sum(id)").collect())
[Row(sum(id)=12497500)]
```

Standard Warning

Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

Major Parts of Spark

Spark Core
  Resilient Distributed Dataset (RDD)

Original (Old) Spark

Spark SQL
  SQL, csv, json
  Dataframe

Newer Version Spark

Spark Streaming
  Near real-time response

MLib Machine Learning Library
  Statistics, regression, clustering, dimension reduction, feature extraction
  Optimization

GraphX

Spark

Ecosystem of packages, libraries and systems on top of Spark Core

Unstructured API

Resilient Distributed Datasets (RDD)
Accumulators
Broadcast variables

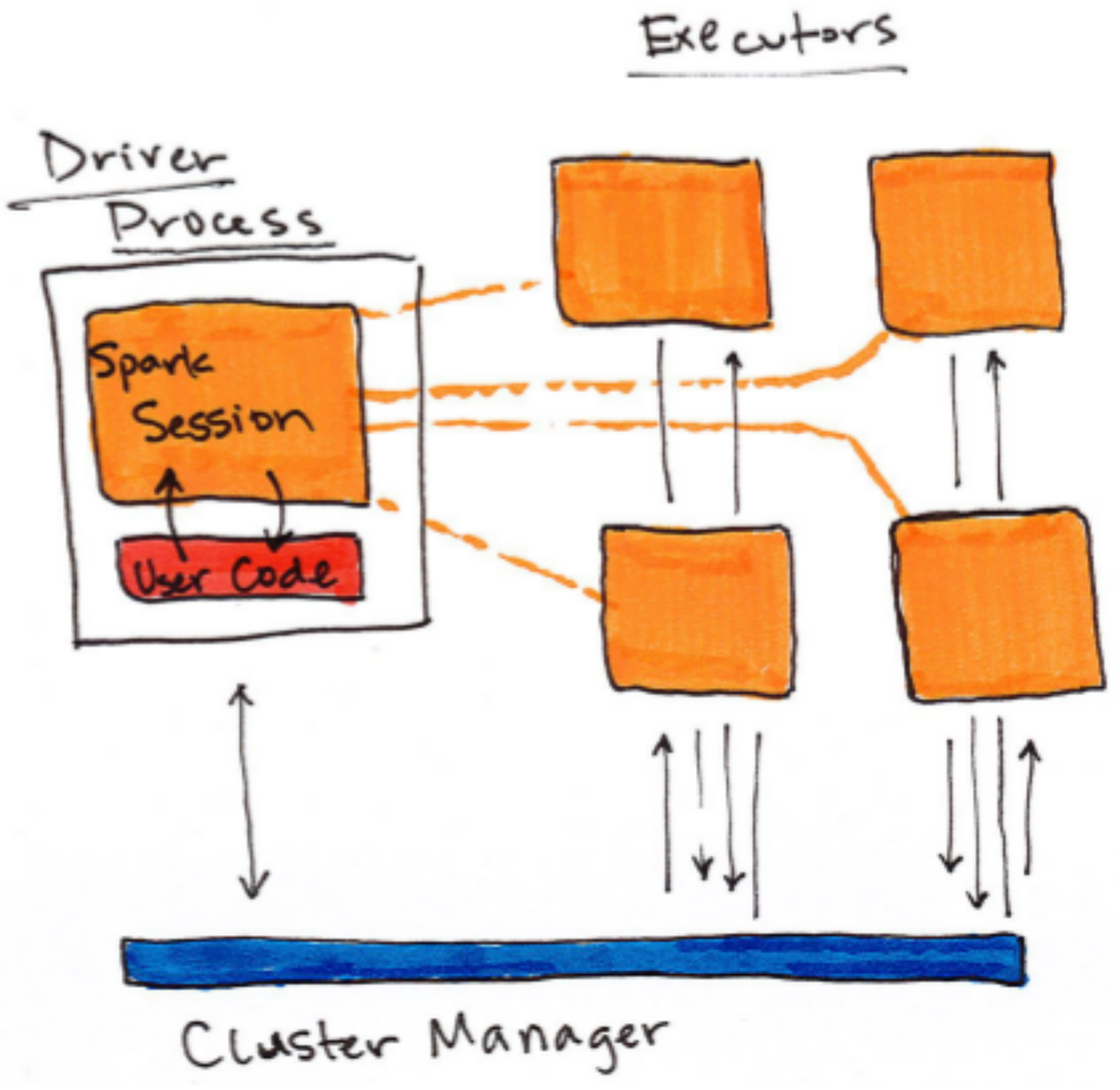
Old Spark

Structured API

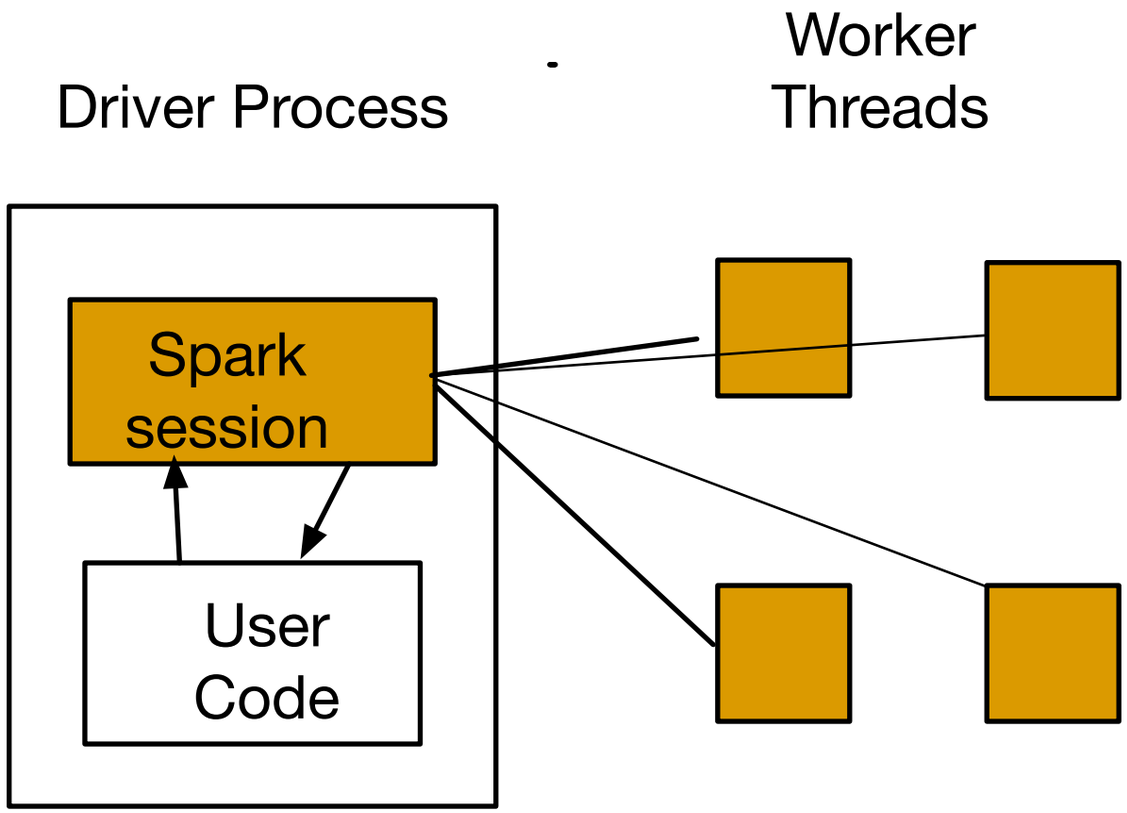
DataFrames
Datasets
Spark SQL

Newer, faster, higher level
Preferred over Unstructured

Basic Architecture



Local Mode



We will start using local mode

Use local mode to

Develop Spark code

SparkContext

Entry point for Unstructured API (Old Spark)

- Connection to Spark cluster
- Runs on master node
- Used to create RDDs, accumulators, broadcast variables
- Only one SparkContext per JVM
- stop() the current SparkContext before starting another

SparkContext org.apache.spark.SparkContext
  Scala version

JavaSparkContext org.apache.spark.api.java.JavaSparkContext
  Java version

pyspark.SparkContext
  Python version

SparkSession

org.apache.spark.sql.SparkSession
pyspark.sql.SparkSession

Contains a SparkContext

Entry point to use Dataset & DataFrame

Connection to Spark cluster
Runs on master node

Major Data Structures

Resilient Distributed Datasets (RDDs)

- Fault-tolerant collection of elements that can be operated on in parallel

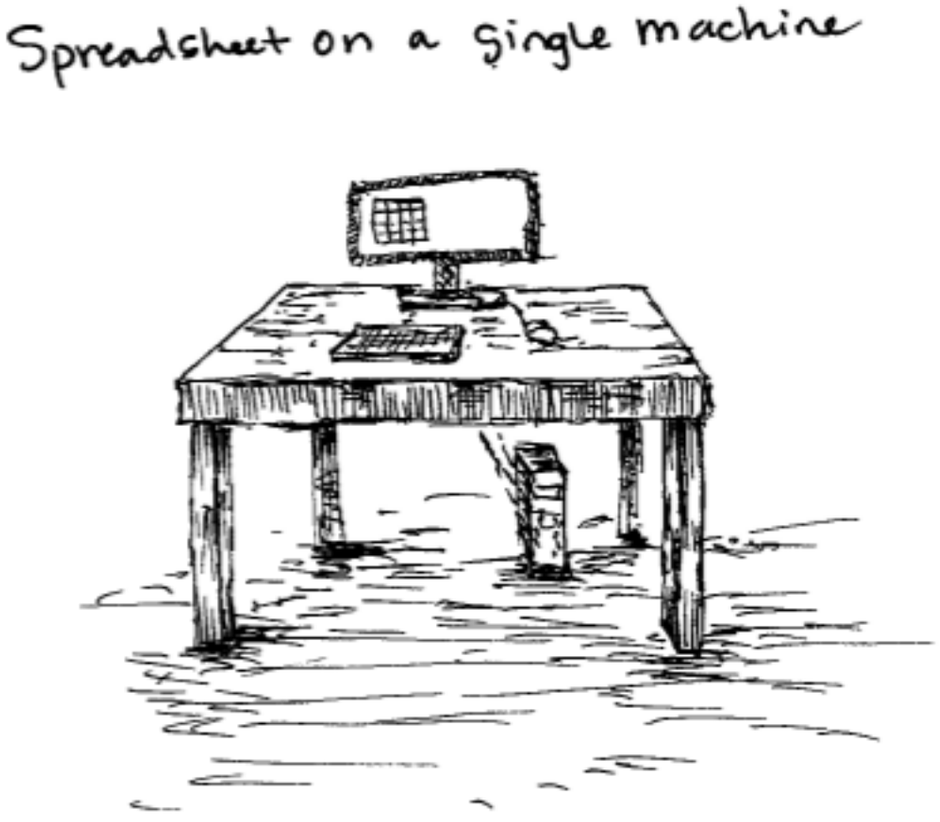
Dataset & Dataframes

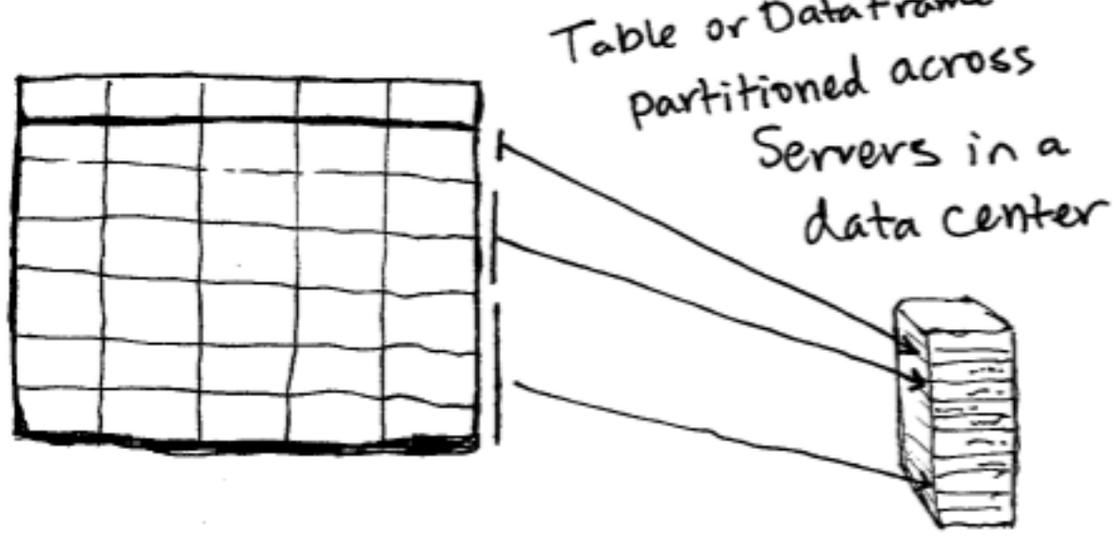
- Fault-tolerant collection of elements that can be operated on in parallel
- Rows & Columns
- JSON, csv, SQL tables
- Part of SparkSQL
- Use RDDs as underlying data structure

Partitions

RDD & Dataset

- Divided into partitions
- Each partition is on different machine





Resilient & Distributed

Distributed

- Partitions on different machines

Resilient

- Each partition can be replicated on multiple machines
- Data structure knows how to reproduce operations

Basic Operations

RDDs, Dataframes, Datasets
- Immutable

Transformations
- Create new dataset (RDD) from existing one
- Lazy
- Only done when needed by an action
- Examples
  - map, filter, sample, union, distinct, groupByKey, repartition

Actions
- Return results to driver program
- Examples
  - reduce, collect, count, first, take

Actions & Transformations on DataSet

View the Spark Scala API

org.apache.spark.sql.Dataset

Actions

```
def collect(): Array[T]
```
Returns an array that contains all rows in this Dataset.

```
def collectAsList(): List[T]
```
Returns a Java list that contains all rows in this Dataset.

```
def count(): Long
```
Returns the number of rows in the Dataset.

```
def describe(cols: String*): DataFrame
```
Computes statistics for numeric and string columns, including count, mean, stddev, min, and max.

```
def first(): T
```
Returns the first row.

Typed transformations

```
def alias(alias: Symbol): Dataset[T]
```
(Scala-specific) Returns a new Dataset with an alias set.

```
def alias(alias: String): Dataset[T]
```
Returns a new Dataset with an alias set.

```
def as(alias: Symbol): Dataset[T]
```
(Scala-specific) Returns a new Dataset with an alias set.

DataFrame, DataSet & RDD

What are they

What is the difference

When do use which one

Which languages can use them

DataFrame

Table with rows and Columns

Row

org.apache.spark.sql.Row

Schema

Column labels

Column types

```
+-------+----+
|   name| age|
+-------+----+
|   Andy|  30|
| Justin|  19|
|Michael|null|
+-------+----+
```

Partitioner

Distributes DataFrame among cluster

Plan

Series of transformations to perform on DataFrame

Langauges

Scala, Java, JVM languages, Python, R

Optimized

Spark Catalyst Optimizer

Python DataFrame & Spark DataFrame

They are different

Need Apache Arrow to convert between them

DataSet

Same as DataFrame except for Rows

Programmer defines Row class
- Scala Cas Class
- Java Bean

Difference from DataFrame
- Compiler knows column names and column types in DataSet
- Compile time error checking
- Better data layout

Languages
- Scala, Java, JVM languages

RDD

Table

No information about types

```
+-------+----+
|   Andy|  30|
| Justin|  19|
|Michael|null|
+-------+----+
```

No compile time or runtime type checking

Shares same basic operations as DataFrames & DataSets

Far fewer optimizations

No Catalyst Optimizer

No space optimization

Example - Same data

RDD	33.3 MB
DataFrame	7.3 MB

Languages

Java, Scala

Python, R - not recommended

Spark Types

Java Types are not space efficient

“abcd” - 48 bytes

Spark has its own types

Special memory representation of each type

Space efficient

Cache aware

Spark	Scala	Python	Python API
ByteType	Byte	int or long	ByteType()
ShortType	Short	int or long	ShortType()
IntegerType	Int	int or long	IntergerType()
LongType	Long	int or long	LongType()

Structured verses Unstructured

Structured = DataSet, DataFrame

Unstructured = RDD

Typed verses Untyped

Typed = DataSet

Untyped = DataFrame

DataFrame

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

from datetime import datetime, date
import pandas as pd
from pyspark.sql import Row

df = spark.createDataFrame([
    Row(a=1, b=2., c='string1', d=date(2000, 1, 1), e=datetime(2000, 1, 1, 12, 0)),
    Row(a=2, b=3., c='string2', d=date(2000, 2, 1), e=datetime(2000, 1, 2, 12, 0)),
    Row(a=4, b=5., c='string3', d=date(2000, 3, 1), e=datetime(2000, 1, 3, 12, 0))
])
df
```

```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

About the Output - Logging

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLev
el).
22/02/10 15:46:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your p
latform... using builtin-java classes where applicable
```

Out[2]: DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

sc = spark.sparkContext
sc.setLogLevel("OFF")

from datetime import datetime, date
import pandas as pd
from pyspark.sql import Row

df = spark.createDataFrame([
    Row(a=1, b=2., c='string1', d=date(2000, 1, 1), e=datetime(2000, 1, 1, 12, 0)),
    Row(a=2, b=3., c='string2', d=date(2000, 2, 1), e=datetime(2000, 1, 2, 12, 0)),
    Row(a=4, b=5., c='string3', d=date(2000, 3, 1), e=datetime(2000, 1, 3, 12, 0))
])
df
```

Debug levels

ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE, WARN

Specifying Types

```
df = spark.createDataFrame([
    (1, 2., 'string1', date(2000, 1, 1), datetime(2000, 1, 1, 12, 0)),
    (2, 3., 'string2', date(2000, 2, 1), datetime(2000, 1, 2, 12, 0)),
    (3, 4., 'string3', date(2000, 3, 1), datetime(2000, 1, 3, 12, 0))
], schema='a long, b double, c string, d date, e timestamp')
df
```

```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

From Panda Dataframe

```
pandas_df = pd.DataFrame({
    'a': [1, 2, 3],
    'b': [2., 3., 4.],
    'c': ['string1', 'string2', 'string3'],
    'd': [date(2000, 1, 1), date(2000, 2, 1), date(2000, 3, 1)],
    'e': [datetime(2000, 1, 1, 12, 0), datetime(2000, 1, 2, 12, 0), datetime(2000, 1, 3, 12, 0)]
})
df = spark.createDataFrame(pandas_df)
df
```

Viewing Data

df.show()

```
+---+---+-------+----------+--------------------+
|  a|  b|      c|         d|                   e|
+---+---+-------+----------+--------------------+
|  1|2.0|string1|2000-01-01|2000-01-01 12:00:00|
|  2|3.0|string2|2000-02-01|2000-01-02 12:00:00|
|  3|4.0|string3|2000-03-01|2000-01-03 12:00:00|
+---+---+-------+----------+--------------------+
```

df.printSchema()

```
root
 |-- a: long (nullable = true)
 |-- b: double (nullable = true)
 |-- c: string (nullable = true)
 |-- d: date (nullable = true)
 |-- e: timestamp (nullable = true)
```

Viewing Data

df.select("a", "b", "c").describe().show()

```
+-------+---+---+-------+
|summary|  a|  b|      c|
+-------+---+---+-------+
|  count|  3|  3|      3|
|   mean|2.0|3.0|   null|
| stddev|1.0|1.0|   null|
|    min|  1|2.0|string1|
|    max|  3|4.0|string3|
+-------+---+---+-------+
```

Explain This

```
In [ ]: df.select("a", "b", "c").describe().show()

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```

Running on 10 Cores

df.rdd.getNumPartitions()

```
Out[22]: 10
```

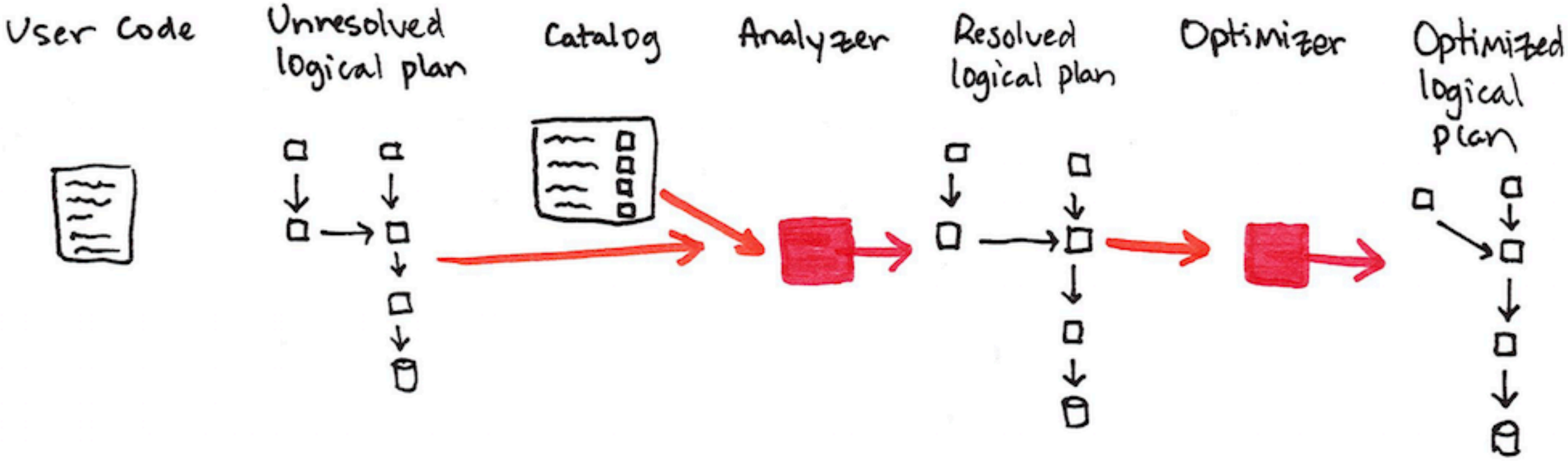
Lazy Evaluation

```
df = spark.createDataFrame([
    Row(a=1, b=2., c='string1', d=date(2000, 1, 1), e=datetime(2000, 1, 1, 12, 0)),
    Row(a=2, b=3., c='string2', d=date(2000, 2, 1), e=datetime(2000, 1, 2, 12, 0)),
    Row(a=4, b=5., c='string3', d=date(2000, 3, 1), e=datetime(2000, 1, 3, 12, 0))
])

x = df.select("a", "b", "c")
y = x.filter(x.a > 1)
y.explain()
```

```
== Physical Plan ==
*(1) Project [a#0L, b#1, c#2]
+- *(1) Filter (isnotnull(a#0L) AND (a#0L > 1))
   +- *(1) Scan ExistingRDD[a#0L,b#1,c#2,d#3,e#4]
```

Planning


User Code
Unresolved logical plan
Catalog
Analyzer
Resolved logical plan
Optimizer
Optimized logical plan