

CS 649 Big Data: Tools and Methods  
Spring Semester, 2021  
Doc 19 Running Spark, Partitions  
Apr 6, 2021

Copyright ©, All rights reserved. 2021 SDSU & Roger Whitney,  
5500 Campanile Drive, San Diego, CA 92182-7700 USA.  
OpenContent (<http://www.opencontent.org/opl.shtml>) license  
defines the copyright on this document.

# Basic Outline

Develop & test Spark locally

Upload program file & data to S3

Configure & launch cluster

- AWS Management Console

- AWS CLI

- SDKs

Monitor cluster

Make sure you terminate cluster when done

# Starting Point

```
from random import random
from operator import add

from pyspark import SparkContext

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

def calculate_pi(partitions, output_uri):

    def calculate_hit(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 < 1 else 0

    tries = 100000 * partitions
    logger.info(
        "Calculating pi with a total of %s tries in %s partitions.", tries, partitions)
    with SparkSession.builder.appName("My PyPi").getOrCreate() as spark:
        hits = spark.sparkContext.parallelize(range(tries), partitions)\
            .map(calculate_hit)\
            .reduce(add)
        pi = 4.0 * hits / tries
        logger.info("%s tries and %s hits gives pi estimate of %s.", tries, hits, pi)
        if output_uri is not None:
            df = spark.createDataFrame(
                [(tries, hits, pi)], ['tries', 'hits', 'pi'])
```

# What is the Smallest Program that will Run

```
from pyspark.sql import SparkSession
```

```
if __name__ == "__main__":
```

```
    with SparkSession.builder.appName("Hello").getOrCreate() as spark:
```

```
        df = spark.createDataFrame([("Hello", "World")], ['A', 'B',])
```

```
        df.write.mode('overwrite').json("s3://rw-output-data/helloWorld")
```



▼  rw-696-flight	•
 flight.py	2 KB
 hello.py	279 bytes
 hellolog.py	545 bytes

---

**Add step** ✕

**Step type**

**Name**

**Deploy mode**  Run your driver on a slave node (cluster mode) or on the master node as an external client (client mode).

**Spark-submit options**  Specify other options for spark-submit.

**Application location\***  Path to a JAR with your application and dependencies (client deploy mode only supports a local path).

**Arguments**  Specify optional arguments for your application.

**Action on failure**  What happens if the step fails

[Cancel](#) [Add](#)

# Logging

```
from pyspark.sql import SparkSession
import logging
```

```
logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
```

```
if __name__ == "__main__":
    logger.info("hello world")
    logger.error("hello error")
    logger.debug("hello debug")
    logger.critical("hello critical")
    with SparkSession.builder.appName("Hello").getOrCreate() as spark:
        df = spark.createDataFrame([("Hello", "World")], ['A', 'B',])
        df.write.mode('overwrite').json("s3://rw-output-data/helloWorld")
```

```
INFO: hello world
ERROR: hello error
CRITICAL: hello crtical
```

To get log output easily  
deploy mode must be Client

To get log output deploy mode must be Client

**Add step**

**Step type** Spark application

**Name** Spark application

**Deploy mode**  Client  Cluster  Run on 1

			s-2UV660R54A8ER	Hello Print	Pending	--	controller   syslog*   stderr   stdout	
			s-LL92QURYYHYR	Client Log 1	Completed	2021-04-05 11:13 (UTC-7)	26 seconds	controller   syslog*   stderr   stdout



INFO: hello world  
ERROR: hello error  
CRITICAL: hello critical



# Printing

```
from pyspark.sql import SparkSession
```

```
if __name__ == "__main__":
```

```
    print("Hi World")
```

```
    with SparkSession.builder.appName("Hello").getOrCreate() as spark:
```

```
        df = spark.createDataFrame([("Hello", "World")], ['A', 'B',])
```

```
        df.write.mode('overwrite').json("s3://rw-output-data/helloWorld")
```

To get print output easily deploy mode must be Client

<input type="radio"/>	<input checked="" type="radio"/>	s-2UV660R54A8ER	Hello Print	Pending	--	controller   syslog*   stderr   stdout	
<input type="radio"/>	<input type="radio"/>	s-LL92QURYYHYR	Client Log 1	Completed	2021-04-05 11:13 (UTC-7)	26 seconds	controller   syslog*   stderr   stdout

Hi World

To get print output easily deploy mode must be Client

# Test Program 1 - Pi

```
from random import random
from operator import add
```

```
from pyspark import SparkContext
```

```
if __name__ == "__main__":
    sc = SparkContext(appName="PythonPi")
    partitions = 3
    n = 100000 * partitions
```

```
def f(_):
    x = random() * 2 - 1
    y = random() * 2 - 1
    return 1 if x ** 2 + y ** 2 < 1 else 0
```

```
count = sc.parallelize(range(1, n + 1), partitions).map(f).reduce(add)
print("Pi is roughly %f" % (4.0 * count / n))
```

```
sc.stop()
```

Designed to have no  
Command line dependancies  
No input or output files

# Group By Data

```
reader = spark.read  
reader.option("header",True).option("inferSchema",True)  
ordersDF = reader.csv("orders.csv")  
ordersDF.show()
```

```
+-----+-----+  
|customer|amount|  
+-----+-----+  
|         a |      2 |  
|         b |      8 |  
|         a |      3 |  
|         c |      9 |  
|         a |      4 |  
|         b |     16 |  
|         c |     11 |  
|         b |     24 |  
|         c |     30 |  
+-----+-----+
```

# groupBy

```
import pyspark.sql.functions as F

amountGrouped = ordersDF.groupBy("customer") \
    .agg(
        F.sum("amount").alias("Total"),
        F.mean("amount").alias("Average"),
        F.count("amount").alias("Number of Orders"))
amountGrouped.sort("customer").show()
```

customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

customer	Total	Average	Number of Orders
a	9	3.0	3
b	48	16.0	3
c	50	16.666666666666668	3

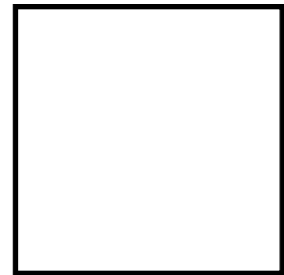
# How does this work?

```
import pyspark.sql.functions as F

amountGrouped = ordersDF.groupBy("customer") \
    .agg(F.sum("amount").alias("Total"))
amountGrouped.sort("customer").show()
```

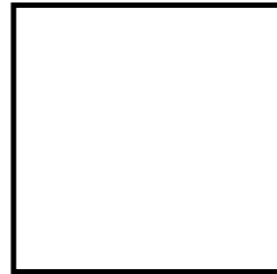
```
+-----+-----+
|customer|Total|
+-----+-----+
|      a |    9|
|      b |   48|
|      c |   50|
+-----+-----+
```

# groupBy

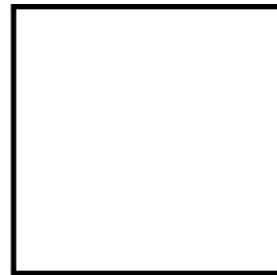


Master

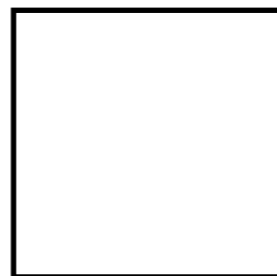
Workers



a	2
b	8
a	3



c	9
a	4
b	16



c	11
b	24
c	30

customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

# groupBy - Local Sum

Workers

a	5
b	8

c	9
a	4
b	16

c	41
b	24

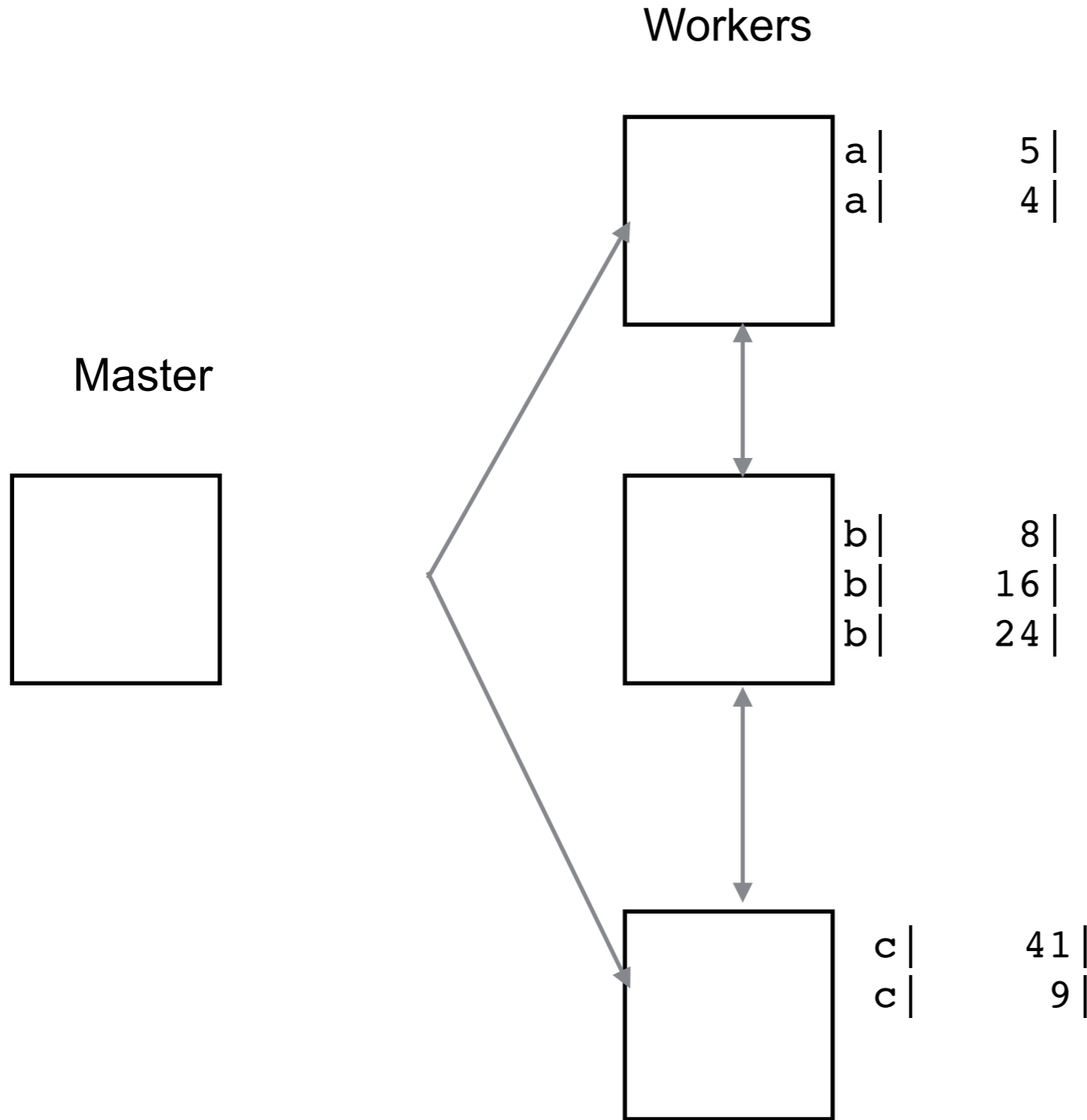
Master

--	--

customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

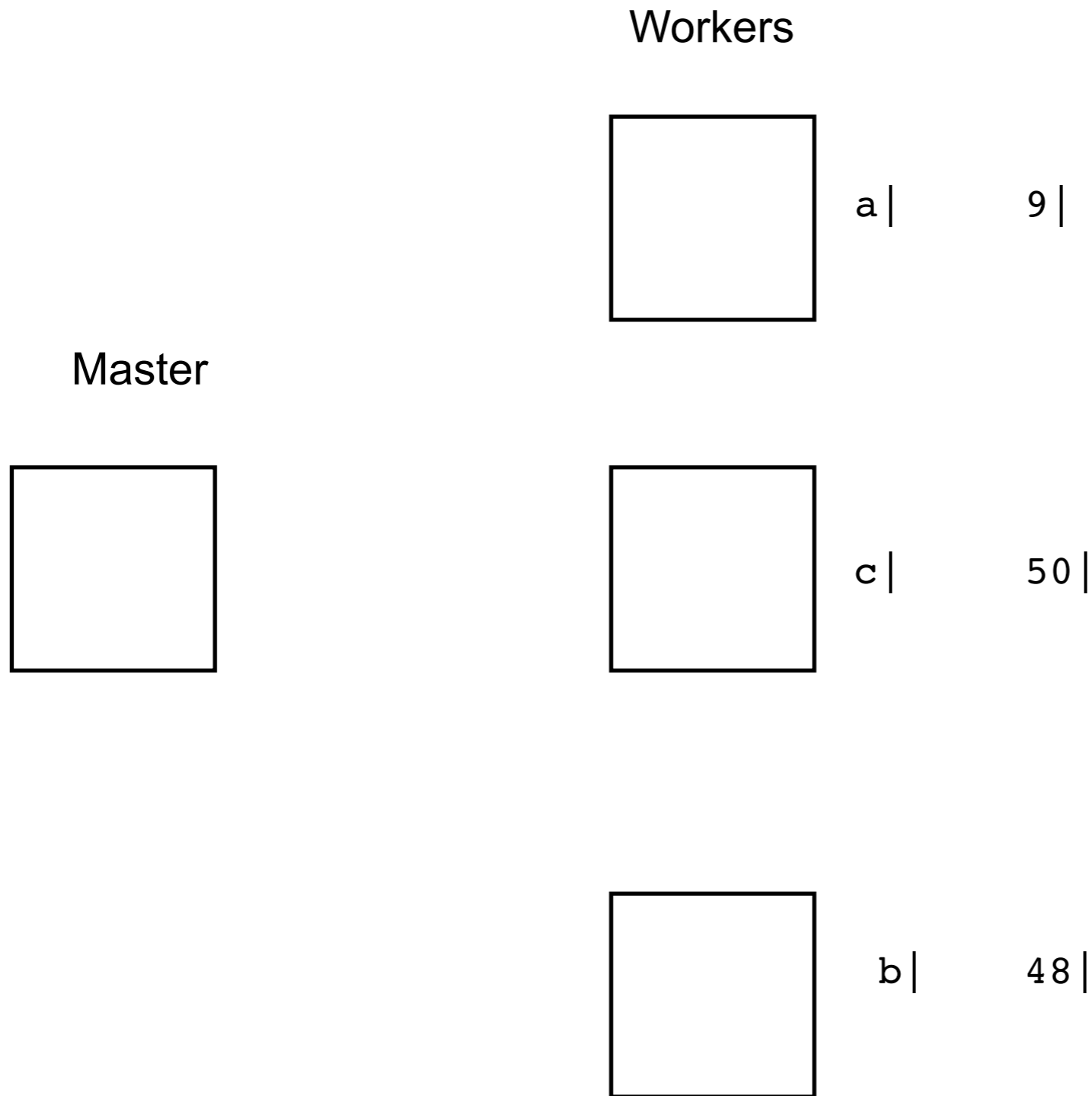


# groupBy - Shuffle



customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

# groupBy - Final Sum



customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

groupBy  
Transformation

# Partitions

Spark divides data in to separate partitions

Default is 50MB of data per partition

This allows spark to process the data in parallel

Each partition is processed in parallel

When you write a dataframe out you get a file for each partition

```
df.rdd.getNumPartitions()
```

Sample Data: 1

Full Dataset: 28

# Partitions & Worker Machines

If you have 2 worker machines & 28 partitions

Each machine gets 14 partitions

Processes the partitions independently

# Partition Size

```
spark.conf.get("spark.sql.files.maxPartitionBytes")
```

```
'134217728b'
```

```
spark.conf.set("spark.sql.files.maxPartitionBytes", 1000000)
```

```
read the data
```

```
df.rdd.getNumPartitions()
```

```
Sample Data: 18
```

# Repartitions

We can repartition a dataframe

Repartition by

- Column

- Number of partition

- Column and number of partitions

Repartitioning **reshuffles** the data

- Expensive

# Repartitioning

```
data = [("a",3),("a",1), ("b",1), ("a",2),("b",2)]  
df = spark.createDataFrame(data=data)
```

```
df.rdd.glom().collect()
```

```
[ [Row(_1='a', _2=3),  
  Row(_1='a', _2=1),  
  Row(_1='b', _2=1),  
  Row(_1='a', _2=2),  
  Row(_1='b', _2=2) ] ]
```

```
df2 = df.repartition(2,"_1")
```

```
[ [Row(_1='a', _2=3), Row(_1='a', _2=1), Row(_1='a', _2=2)],  
  [Row(_1='b', _2=1), Row(_1='b', _2=2)] ]
```

# Coalesce

Used to reduce the number of partitions

Will not increase the number of partitions

Does not shuffle the data

```
df.coalesce(2)
```



# Repartition vs Coalesce

## Repartition

Want output partitions to be equally distributed chunks

Increase the number of partitions

Perform a shuffle of the data

## Coalesce

Decrease the number of partitions

Avoid shuffle

# Changing the Number of Partitions

Can change the parallelism

```
df.coalesce(1)
```

1 partition, no parallelism

# `pyspark.sql.DataFrameWriter.partitionBy`

```
df.write.partitionBy('year').format('csv').save("howMany?.csv")
```

Does local partition of the data

Each partition is now partitioned into separate files based on 'year' column

# SSH to your Master Node

Create Amazon EC2 Key pair

Instructions

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html#having-ec2-create-your-key-pair>

Open EC2 Dashboard - Select Key Pairs

# SSH to your Master Node

In Create Cluster - Quick Options

HIVE 2.3.0 Metastore

Spark: Spark 2.2.0 on Hadoop 2.7.3 YARN with Ganglia 3.7.2 and Zeppelin 0.7.2

Use AWS Glue Data Catalog for table metadata ⓘ

## Hardware configuration

Instance type  ⌵

Number of instances  (1 master and 2 core nodes)

## Security and access

EC2 key pair  ⌵ ⓘ [Learn how to create an EC2 key pair.](#)

Permissions  Default  Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role [EMR\\_DefaultRole](#) ⓘ

EC2 instance profile [EMR\\_EC2\\_DefaultRole](#) ⓘ

Cancel

Create cluster

# SSH to your Master Node

[Clone](#) [Terminate](#) [AWS CLI export](#)

Cluster: LargeRun **Waiting** Cluster ready after last step completed.

[Summary](#) [Application history](#) [Monitoring](#) [Hardware](#) [Events](#) [Steps](#) [Configurations](#) [Bootstrap actions](#)

**Connections:** [Enable Web Connection](#) – Zeppelin, Spark History Server, Ganglia, Resource Manager ... (View All)  
**Master public DNS:** ec2-34-211-49-97.us-west-2.compute.amazonaws.com [SSH](#)  
**Tags:** -- [View All / Edit](#)

Click for Instructions

# Command-line Tools

## Flintrock

Open-source command-line tool for launching Apache Spark clusters

<https://github.com/nchammas/flintrock>

## aws cli

Amazon's command line tool

<https://aws.amazon.com/cli/>

# Generating the Command Line

[Clone](#) [Terminate](#) [AWS CLI export](#)

Cluster: CourseSlides **Starting** Configuring cluster software

[Summary](#) [Application history](#) [Monitoring](#) [Hardware](#) [Configurations](#) [Events](#) [Steps](#) [Bootstrap actions](#)

[Add step](#) [Clone step](#) [Cancel step](#)

Steps

[View all interactive jobs](#) | [View all j](#)

Filter: All steps  2 steps (all loaded) [Refresh](#)

	ID	Name	Status	Start time (UTC-7)	Elapsed time	Log files <a href="#">↗</a>	Actions
<input type="radio"/>	s-3Q0ZVOAZV3VR	Spark application	Pending		--	<a href="#">View logs</a>	<a href="#">View jobs</a>
<input type="radio"/>	s-KSWMDHULHSD8	Setup hadoop debugging	Pending		--	<a href="#">View logs</a>	<a href="#">View jobs</a>



## AWS CLI export

```
aws emr create-cluster --termination-protected --applications Name=Hadoop Name=Spark --ec2-attributes
'{"InstanceProfile":"EMR_EC2_DefaultRole","SubnetId":"subnet-0f55196b","EmrManagedSlaveSecurityGroup":"sg-
65bffa1c","EmrManagedMasterSecurityGroup":"sg-62bffa1b"}' --release-label emr-5.22.0 --log-uri 's3n://aws-
logs-834365227482-us-west-2/elasticmapreduce/' --steps '[{"Args":["spark-submit","--deploy-
mode","client","s3://rw-696-flight/pi.py"],"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE","Jar":"command-
runner.jar","Properties":"","Name":"Spark application"}]' --instance-groups
'[{ "InstanceCount":1,"InstanceGroupType":"MASTER","InstanceType":"m3.xlarge","Name":"Master - 1"},
{ "InstanceCount":2,"InstanceGroupType":"CORE","InstanceType":"m3.xlarge","Name":"Core - 2"}]' --auto-scaling-
role EMR_AutoScaling_DefaultRole --ebs-root-volume-size 10 --service-role EMR_DefaultRole --enable-
debugging --name 'CourseSlides' --scale-down-behavior TERMINATE_AT_TASK_COMPLETION --region us-west-2
```

# Hadoop Ecosystem

## **Hadoop**

HDFS

MapReduce

YARN

Tez

Pig

Hive

Hbase

Sqoop

Oozie

Falcon

Spark

ZooKeeper

Mahout

Phoenix

BigTop

+ others

---

# Apache Pig

Programming Map-Reduce can be low level

Apache Pig - high-level platform for creating programs for Hadoop

Pig Latin

```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS
                count, group AS word;

ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

# Apache Hive

SQL is common way to interact with data

Hive provides SQL like query language for HDFS, Amazon S3 data

HiveQL - converted into MapReduce

```
DROP TABLE IF EXISTS docs;  
CREATE TABLE docs (line STRING);  
LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;  
CREATE TABLE word_counts AS  
SELECT word, count(1) AS count FROM  
(SELECT explode(split(line, '\s')) AS word FROM docs) temp  
GROUP BY word  
ORDER BY word;
```

# Apache HBase

BigTable for Hadoop

Non-relational distributed database

Fault-tolerant way of storing large quantities of sparse data

# Apache Sqoop

People have data in non-hadoop databases

Sqoop

Transferring data between relational databases & Hadoop

# Apache Phoenix

But SQL is common

Phoenix

Massively parallel relational database for Hadoop

Uses HBase to store data

# Apache Spark

Hadoop has latency issues - reads data from disk

MapReduce is not conducive to solving all problems

Spark

- Uses distributed shared memory: Resilient distributed dataset (RDD)

- Iterative algorithms

- Implemented in Scala

Spark Core

Spark SQL

- Dataframes & SQL

Spark Streaming

Spark MLlib

- Machine learning



# Apache Mahout

Hadoop does not have machine learning libraries

Mahout

Environment for quickly creating scalable machine learning applications

Samsara - R-line syntax & environment

# Apache Flink, Apache Storm

Hadoop does batch jobs

Spark streaming has delays

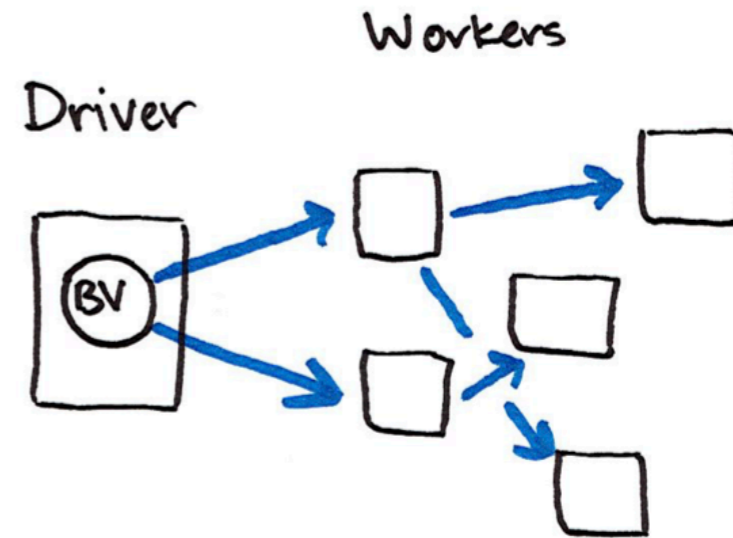
Flink & Storm

Each can have high throughput and low latency streaming

# Distributed Variables

## Broadcast

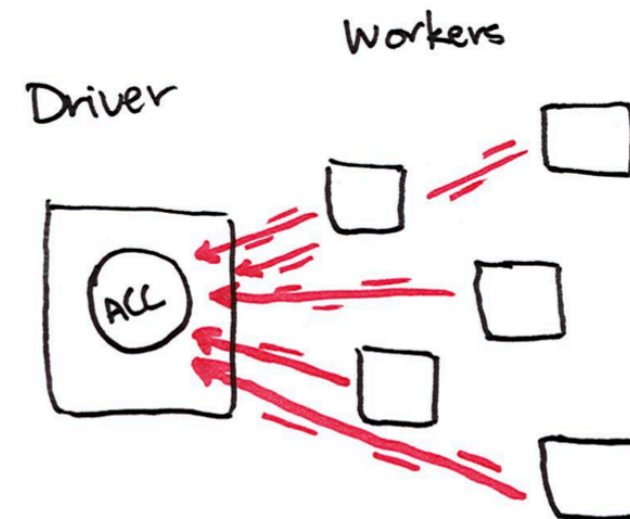
Read-only data shared among workers



## Accumulator

Write only by workers

Read only on master



# Broadcast Example

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession\  
    .builder\  
    .appName("variables")\  
    .getOrCreate()
```

```
courseSize = 45
```

```
courseSizeBroadcast = spark.sparkContext.broadcast(courseSize)
```

```
courseSizeBroadcast.value
```

```
data = spark.sparkContext.parallelize((1,2,3,4,5,6,7,8), 2)
```

```
data.map(lambda x: x + courseSizeBroadcast.value).collect()
```

# Using ComplexType

```
sampleMap = { 'a': 10, 'bat': 1 }
```

```
sampleBroadCast = spark.sparkContext.broadcast(sampleMap)
```

```
sampleBroadCast.value
```

```

import org.apache.spark.sql.SparkSession
val blockSize = "4096"
val spark = SparkSession.builder().
  appName("Broadcast Test").
  config("spark.broadcast.blockSize", blockSize).
  getOrCreate()

val sc = spark.sparkContext
val slices = 2
val num = 10000000

val arr1 = (0 until num).toArray

for (i <- 0 until 3) {
  println("Iteration " + i)
  println("=====")
  val startTime = System.nanoTime
  val barr1 = sc.broadcast(arr1)
  val observedSizes = sc.parallelize(1 to 10, slices).map(_ => barr1.value.length)
  observedSizes.collect().foreach(i => println(i))
  println("Iteration %d took %.0f milliseconds".format(i, (System.nanoTime - startTime) / 1E6))
}

```

# Accumulator Example

```
from pyspark.sql import SparkSession
```

Output

```
spark = SparkSession\
```

16

```
    .builder\
```

```
    .appName("variables")\
```

```
    .getOrCreate()
```

Accumulator

add()

value

```
counter = spark.sparkContext.accumulator(0)
```

```
def count(item):
```

```
    global counter
```

```
    print("item: ", item.id)
```

```
    counter.add(1)
```

Numbers only

Can create custom accumulators

```
df = spark.range(16)
```

```
smaller = df.coalesce(4)
```

```
smaller.foreach(count)
```

```
counter.value
```

# Machine Learning in Spark

MLlib

RDD-based

`org.apache.spark.mllib`

Maintenance mode

DataFrame based (Spark ML)

`org.apache.spark.ml`

Pipelines

Inspired by Python scikit-learn

Classification

Regression

Clustering

Collaborative Filtering

Dimension reduction

Linear Algebra

Statistics



## MLlib: Main Guide

- [Basic statistics](#)
- [Data sources](#)
- [Pipelines](#)
- [Extracting, transforming and selecting features](#)
- [Classification and Regression](#)
- [Clustering](#)
- [Collaborative filtering](#)
- [Frequent Pattern Mining](#)
- [Model selection and tuning](#)
- [Advanced topics](#)

## Machine Learning Library (MLlib) Guide

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable provides tools such as:

- **ML Algorithms:** common learning algorithms such as classification, regression, clustering, and co
- **Featurization:** feature extraction, transformation, dimensionality reduction, and selection
- **Pipelines:** tools for constructing, evaluating, and tuning ML Pipelines
- **Persistence:** saving and load algorithms, models, and Pipelines
- **Utilities:** linear algebra, statistics, data handling, etc.

## Announcement: DataFrame-based API is primary

**The MLlib RDD-based API is now in maintenance mode.**

As of Spark 2.0, the [RDD](#)-based APIs in the `spark.mllib` package have entered maintenance mode.

# Python Examples

`$SPARK_INSTALL_DIR/examples/src/main/python/ml`

aft\_survival\_regression.py  
als\_example.py  
binarizer\_example.py  
bisecting\_k\_means\_example.py  
bucketed\_random\_projection\_lsh\_example.py  
bucketizer\_example.py  
chi\_square\_test\_example.py  
chisq\_selector\_example.py  
correlation\_example.py  
count\_vectorizer\_example.py  
cross\_validator.py  
dataframe\_example.py  
dct\_example.py  
decision\_tree\_classification\_example.py  
decision\_tree\_regression\_example.py  
elementwise\_product\_example.py  
estimator\_transformer\_param\_example.py  
feature\_hasher\_example.py  
fpgrowth\_example.py  
gaussian\_mixture\_example.py  
generalized\_linear\_regression\_example.py  
gradient\_boosted\_tree\_classifier\_example.py  
gradient\_boosted\_tree\_regressor\_example.py  
imputer\_example.py  
index\_to\_string\_example.py

index\_to\_string\_example.py  
isotonic\_regression\_example.py  
kmeans\_example.py  
lda\_example.py  
linear\_regression\_with\_elastic\_net.py  
linearsvc.py  
logistic\_regression\_summary\_example.py  
logistic\_regression\_with\_elastic\_net.py  
max\_abs\_scaler\_example.py  
min\_hash\_lsh\_example.py  
min\_max\_scaler\_example.py  
multiclass\_logistic\_regression\_with\_elastic\_net.py  
multilayer\_perceptron\_classification.py  
n\_gram\_example.py  
naive\_bayes\_example.py  
normalizer\_example.py  
one\_vs\_rest\_example.py  
onehot\_encoder\_estimator\_example.py  
pca\_example.py  
pipeline\_example.py  
polynomial\_expansion\_example.py  
prefixspan\_example.py  
quantile\_discretizer\_example.py  
random\_forest\_classifier\_example.py

# Aggregations

Summarize

groupBy

roll up

cube

window

# Aggregation Functions

count

countDistinct

approx\_count\_distinct

first, last

min, max

sum

sumDistinct

avg, mean

variance, var\_samp, var\_pop

stddev, stddev\_samp, stddev\_pop

skewness, kurtosis

Covariance & Correlation

corr, covar\_samp, covar\_pop

# Example

```
flight_file = "/Users/whitney/Courses/696/Fall17/SparkBookData/flight-data/json/2015-summary.json"  
flight_df = spark.read.json(flight_file)  
flight_df.show(3)
```

```
+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|  
+-----+-----+-----+  
|    United States|             Romania|    15|  
|    United States|             Croatia|     1|  
|    United States|             Ireland|   344|  
+-----+-----+-----+
```

# Example

```
import pyspark.sql.functions as F
newDf = flight_df.select(F.sum("count").alias('Sum'),
                        F.mean("count").alias('Mean'),
                        F.max("count").alias("Max"),
                        F.stddev_samp("count").alias("Sample StdDev"),
                        F.stddev_pop("count").alias("Pop StdDev"),
                        F.count("DEST_COUNTRY_NAME").alias("Count"))
newDf.show()
```

```
+-----+-----+-----+-----+-----+-----+
|   Sum|      Mean|    Max|  Sample StdDev|  Pop StdDev|Count|
+-----+-----+-----+-----+-----+-----+
|453316|1770.765625|370002|23126.516918551915|23081.30374350104|  256|
+-----+-----+-----+-----+-----+-----+
```

# countDistinct

```
import pyspark.sql.functions as F
newDf = flight_df.select(
    F.countDistinct(F.col("DEST_COUNTRY_NAME")).alias("Distinct Dest"),
    F.countDistinct(F.col("ORIGIN_COUNTRY_NAME")).alias("Distinct Origin"))

newDf.show()
```

```
+-----+-----+
|Distinct Dest|Distinct Origin|
+-----+-----+
|          132|          125|
+-----+-----+
```



# countDistinct

```
import pyspark.sql.functions as F
newDf = flight_df.select(
    F.countDistinct(
        F.col("DEST_COUNTRY_NAME"),
        F.col("ORIGIN_COUNTRY_NAME")).alias("Distinct Pair"))

newDf.show()
```

```
+-----+
|Distinct Pair|
+-----+
|           256|
+-----+
```

# countDistinct

```
import pyspark.sql.functions as F
newDf = flight_df.select(
    F.countDistinct(
        "DEST_COUNTRY_NAME",
        "ORIGIN_COUNTRY_NAME",
        "count").alias("Distinct Rows"))

newDf.show()
```

```
+-----+
|Distinct Rows|
+-----+
|           256|
+-----+
```

# countDistinct

```
pyspark.sql.functions.countDistinct(col, *cols)
```

```
pyspark.sql.functions.count(col)
```