CS 649 Big Data: Tools and Methods
Spring Semester, 2022
Doc 24 Running Spark, Partitions
Apr 12, 2022

Basic Outline

Develop & test Spark locally

Upload program file & data to S3

Configure & launch cluster
- AWS Management Console
- AWS CLI
- SDKs

Monitor cluster

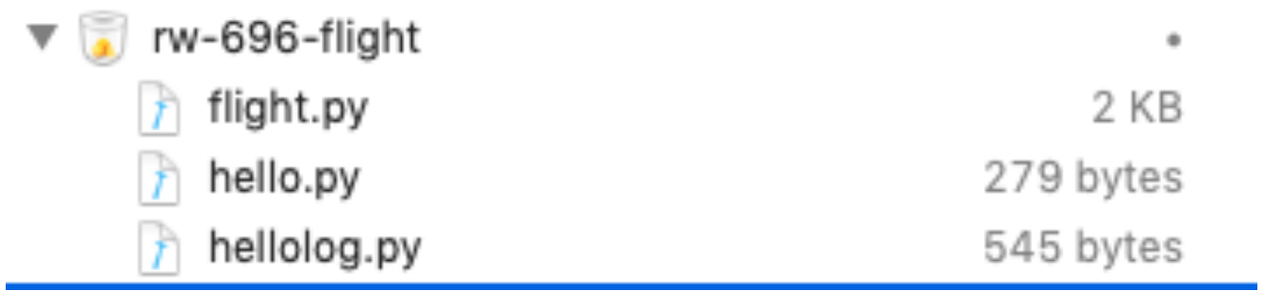
Make sure you terminate cluster when done

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: %(mes

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, partiti
with SparkSession.builder.appName("My PyPi").getOrCreate() as spa
    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
    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")
```



Add step

Step type: Spark application

Name: Spark application

Deploy mode: Cluster

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*: s3://rw-696-flight/hello.py

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: Continue

What happens if the step fails

Cancel Add

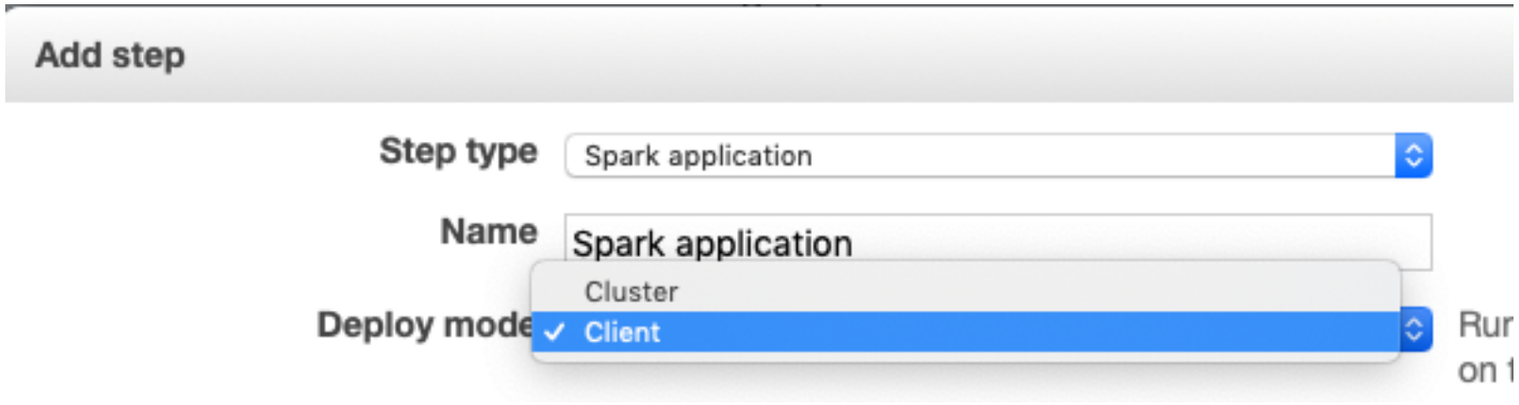
Logging

```
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



▸	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

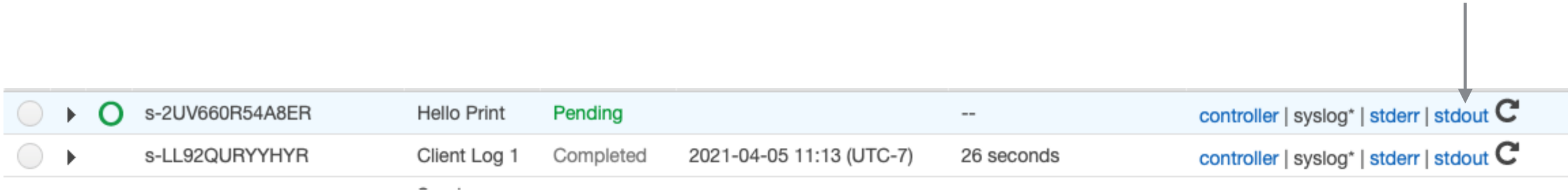
ERROR: hello error

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")
```

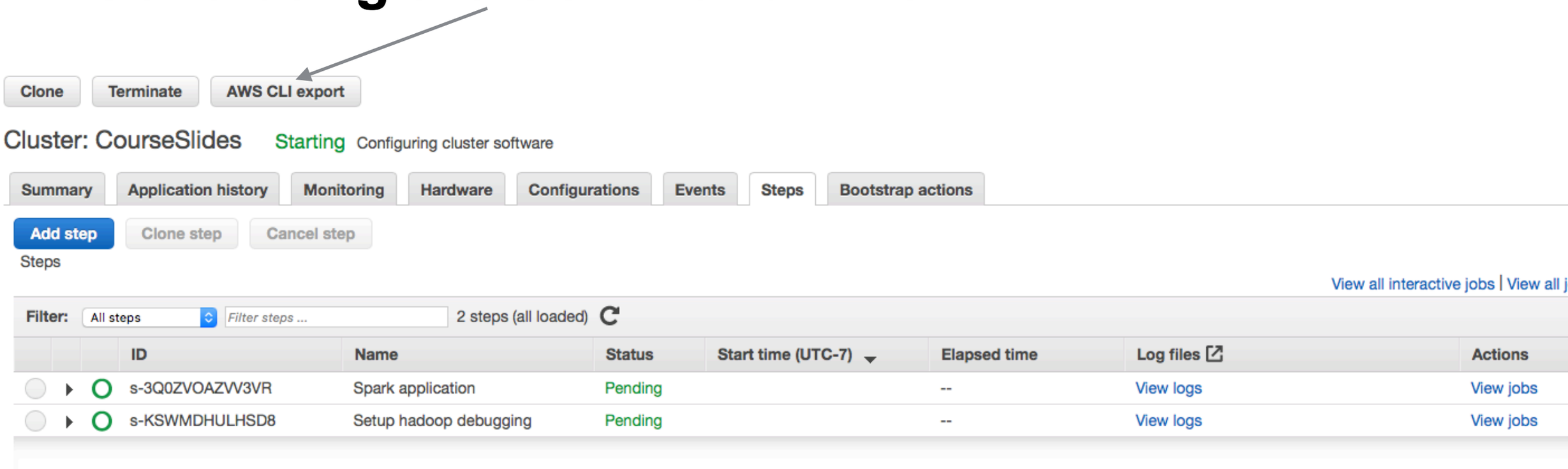
To get print output easily deploy mode must be Client



Hi World

To get print output easily deploy mode must be Client

Generating the Command Line



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
```

Test Program 1 - Pi

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

```
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()
```

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|
+--------+-----+------------------+----------------+
```

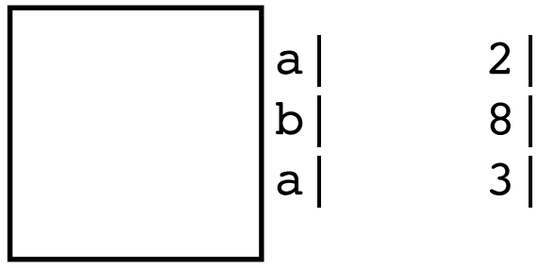
How does this work?

```
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|
+--------+-----+
```

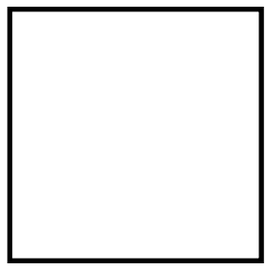
groupBy

Workers

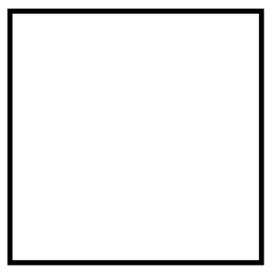


a	2
b	8
a	3

Master



c	9
a	4
b	16

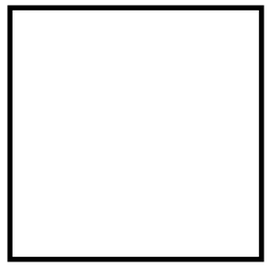


c	11
b	24
c	30

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

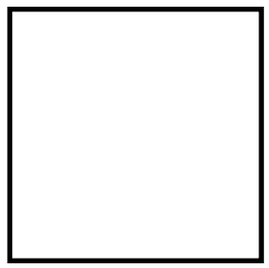
groupBy - Local Sum

Workers

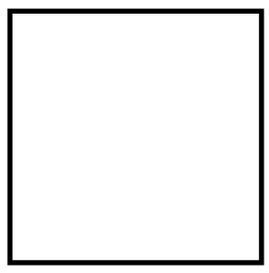


customer	amount
a	5
b	8

Master



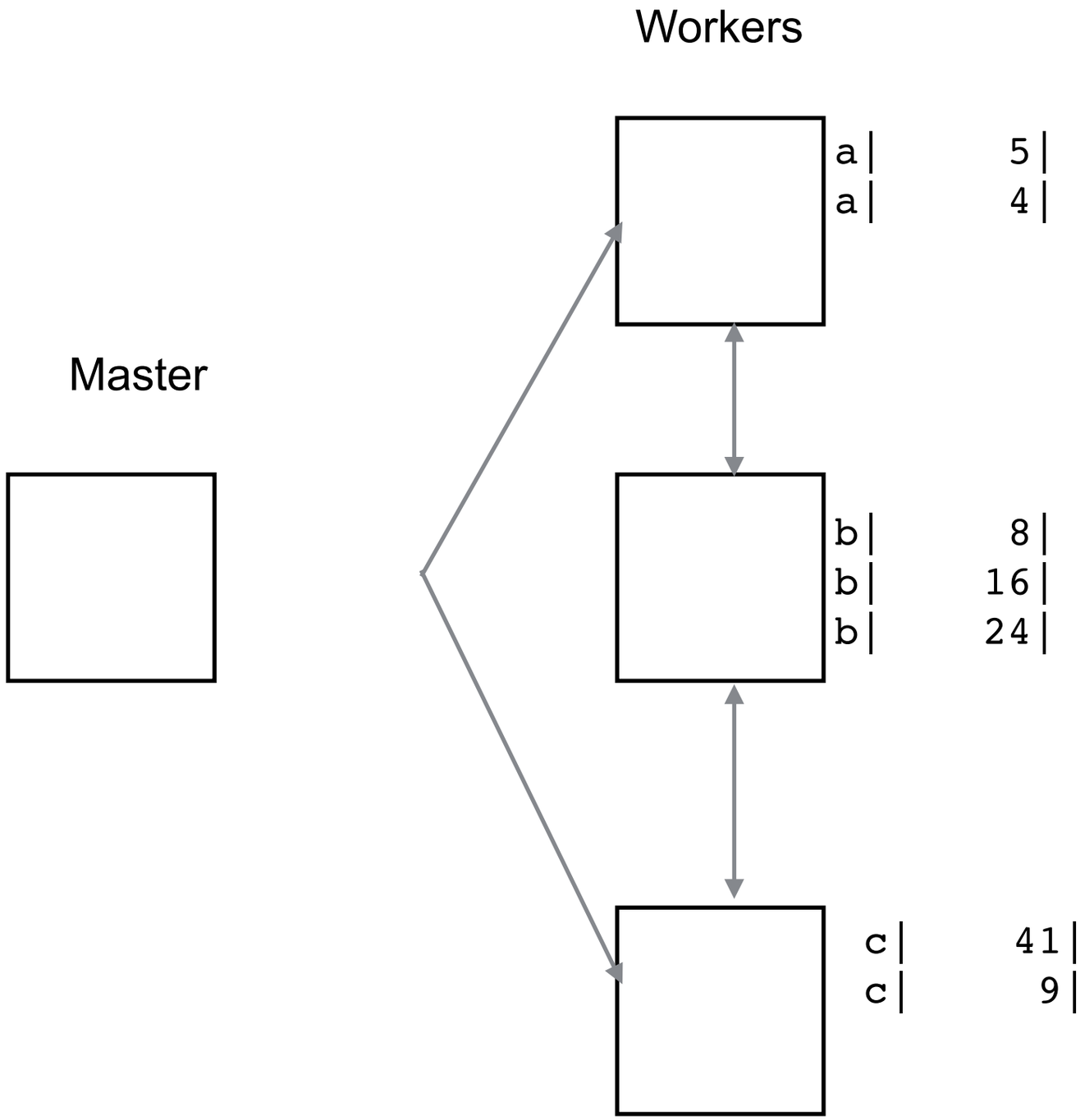
customer	amount
c	9
a	4
b	16



customer	amount
c	41
b	24

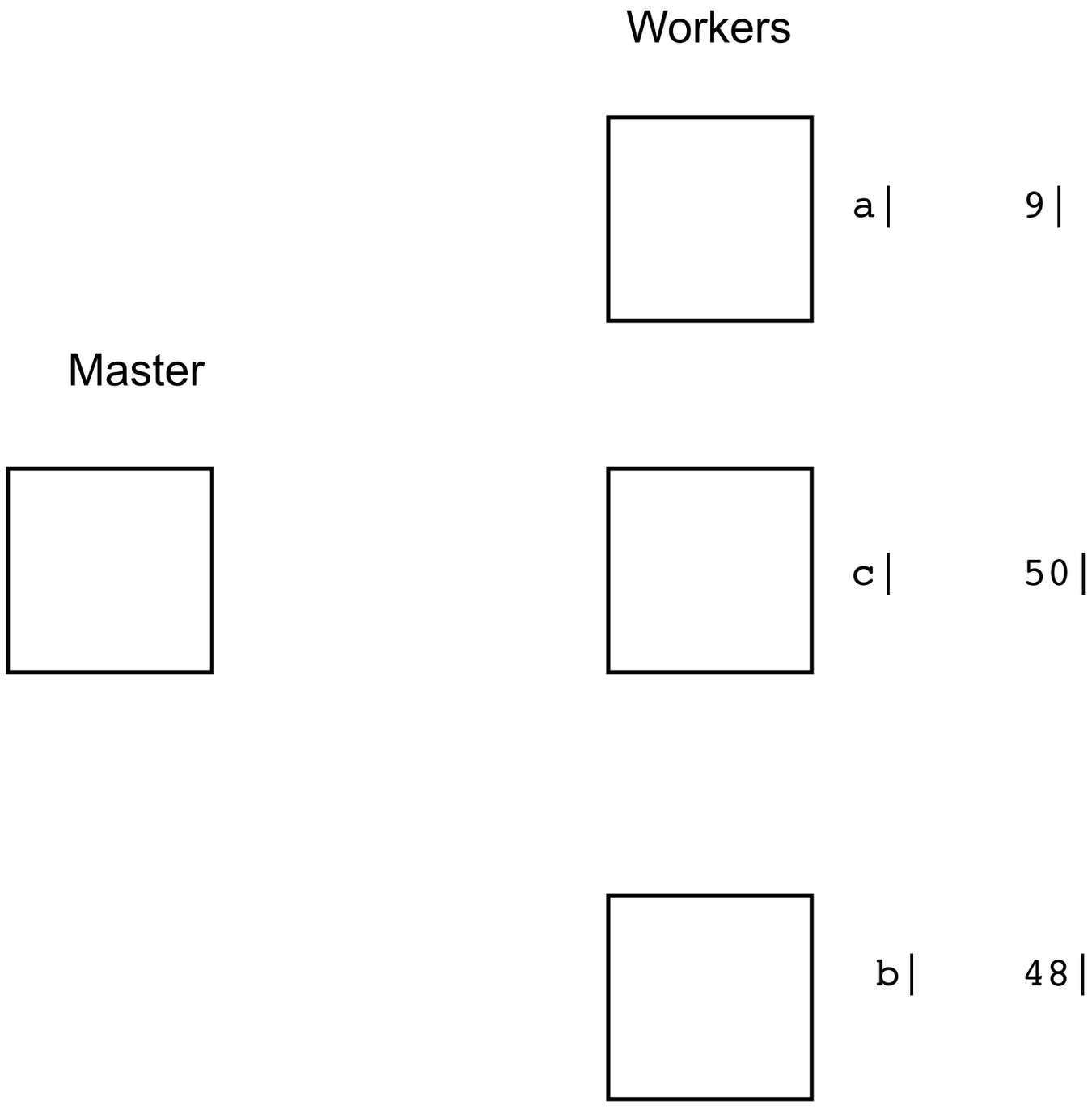
customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

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

Transformations & Actions

Transformations
- Done on worker machines
- Lazy

Actions
- Bring results to master machine
- Triggers transformations

Aggregations

Summarize

groupBy

roll up

cube

window

Aggregation Functions

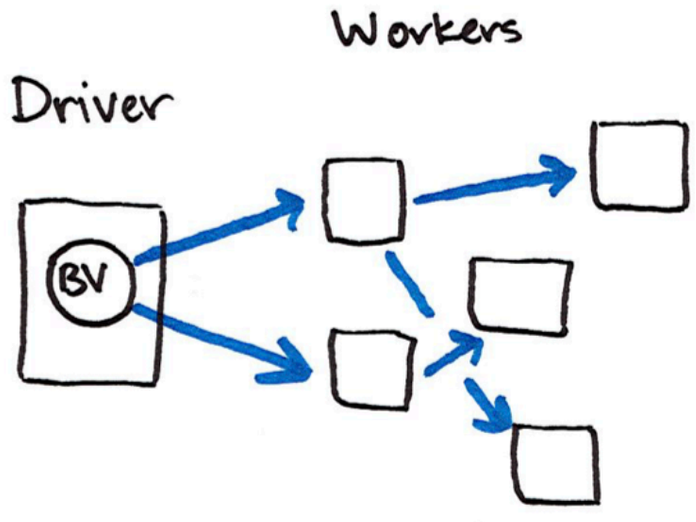
count
countDistinct
approx_count_distinct
first, last
min, max
sum
sumDistinct
avg, mean
variance, var_samp, var_pop
stddev, stddev_samp, stddev_pop
skewness, kurtosis

Covariance & Correlation
    corr, covar_samp, covar_pop

Distributed Variables

Broadcast

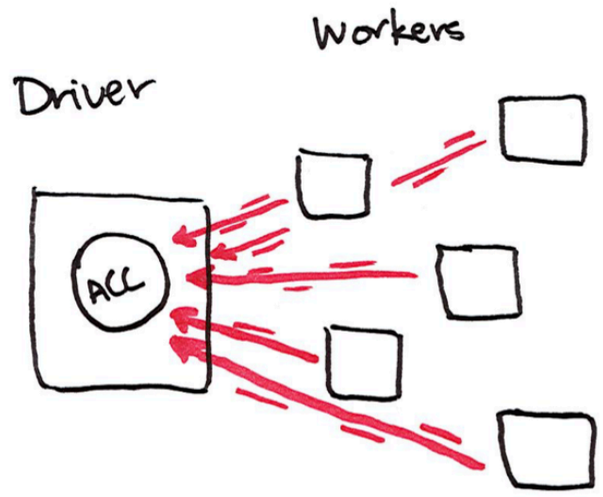
Read-only data shared among workers



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

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

counter = spark.sparkContext.accumulator(0)
def count(item):
    global counter
    print("item: ", item.id)
    counter.add(1)

df = spark.range(16)
smaller = df.coalesce(4)
smaller.foreach(count)
counter.value
```

Output

16

Accumulator

add()

value

Numbers only

Can create custom accumulators

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)

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 quantites 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

Fling & Storm

Each calin to have high throughput and low latency streaming