Spark and Dask

CSCI 8360: Data Science Practicum
Lecture 2

Quick Reference

- Need help with learning git?
  https://try.github.io/
- Specifically, need help with git branching?
  https://learngitbranching.js.org/

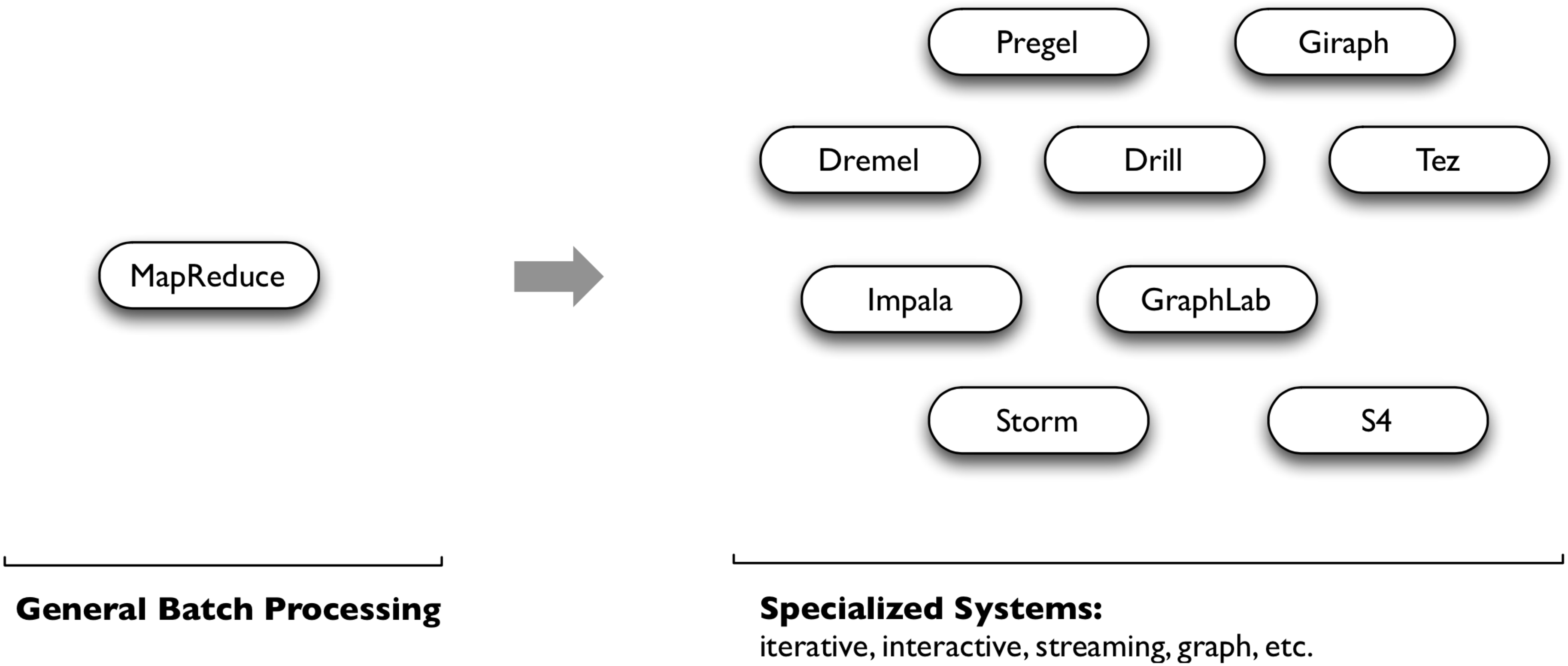
- Web-based, interactive, highly-visual walkthroughs
- **Highly recommend, even if you've used git before**

Apache Spark

Apache Spark

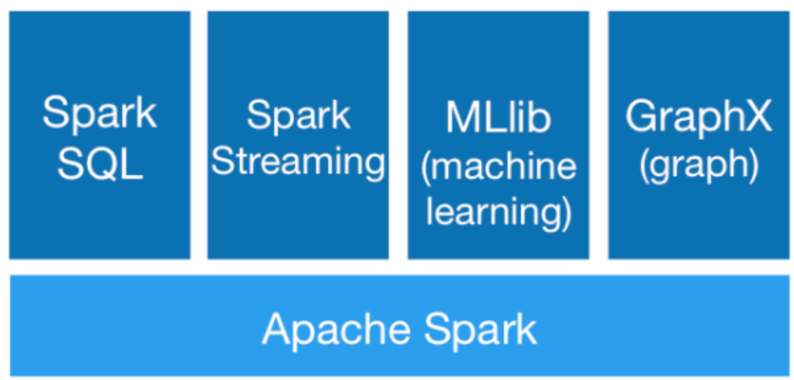
- Out of the UC Berkeley AMPLab in 2014
- Born out of frustration with the only open source distributed programming paradigm / implementation at the time: Hadoop MapReduce
  - Too much Hadoop boilerplate
  - Too many serialization / deserialization operations
  - Map-reduce paradigm is inflexible (graph analytics? real-time processing? iterative algorithms?)
  - Focused on bringing data to code
  - Assumed the absolute worst in terms of hardware reliability

Initial Workaround: Specialization



Along Came Spark

- Spark's goal was to *generalize* MapReduce to support new applications within the same engine
- Two additions:
  - Fast data sharing
  - General DAGs (directed acyclic graphs)
- Best of both worlds: easy to program & more efficient engine in general



More on Spark

- More general
  - Supports map/reduce paradigm
  - Supports vertex-based paradigm
  - Supports streaming algorithms
  - General compute engine (DAG)
- More API hooks
  - Scala, Java, Python, R
- More interfaces

Spark APIs

- Two main APIs: **DataSets** and **DataFrames**
- Both DataSets and DataFrames are high-level abstractions on RDDs, or **Resilient Distributed Datasets**
  - You can directly operate on RDDs if you want
  - (in fact, this was the default behavior until Spark 2.x)
- DataSets
  - Benefits of RDDs (next slide) + benefits of SparkSQL's execution engine
  - **Not available in Python or R** (wtf m8)
- DataFrames
  - Just a DataSet, but with named columns
  - Conceptually equivalent to a table in a database or dataframe in R/Python

Resilient Distributed Datasets (RDDs)

- **R**esilient **D**istributed **D**atasets (RDDs) are primary data abstraction in Spark
  - Fault-tolerant
  - Strongly-typed (within the JVM)
  - Immutable
  - Can be operated on in parallel
    1. Parallelized Collections
    2. Hadoop datasets
- Two types of RDD operations
    1. Transformations (lazy)
    2. Actions (immediate)

Resilient Distributed Datasets (RDDs)

- Can create RDDs from any file stored in HDFS
  - Local filesystem
  - Amazon S3
  - HBase
- Text files, SequenceFiles, or any other Hadoop `InputFormat`
- Any directory or glob
  - /data/201414*

Resilient Distributed Datasets (RDDs)

- Transformations
  - Create a new RDD from an existing one
  - *Lazily* evaluated: results are not immediately computed
    - Pipeline of subsequent transformations can be optimized
    - Lost data partitions can be recovered

Resilient Distributed Datasets (RDDs)

- Actions
  - Create a new RDD from an existing one
  - *Eagerly* evaluated: results are immediately computed
    - Applies previous transformations
    - (cache results?)

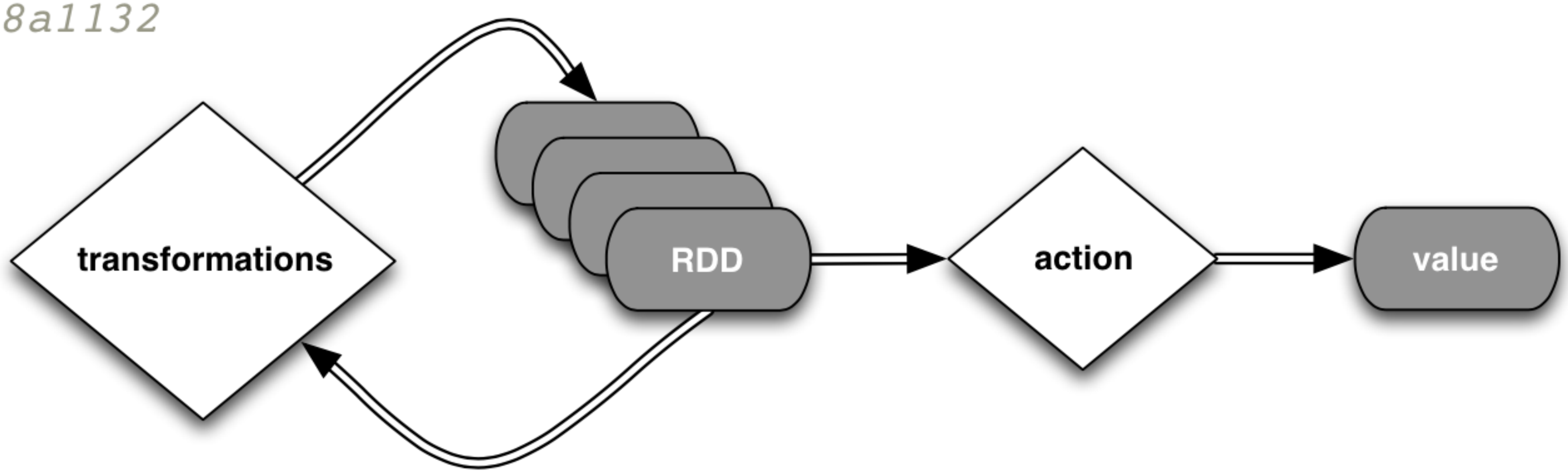
Resilient Distributed Datasets (RDDs)

- Spark can persist / cache an RDD in memory across operations
- Each slice is persisted in memory and reused in subsequent actions involving that RDD
- Cache provides fault-tolerance: if partition is lost, it will be recomputed using transformations that created it

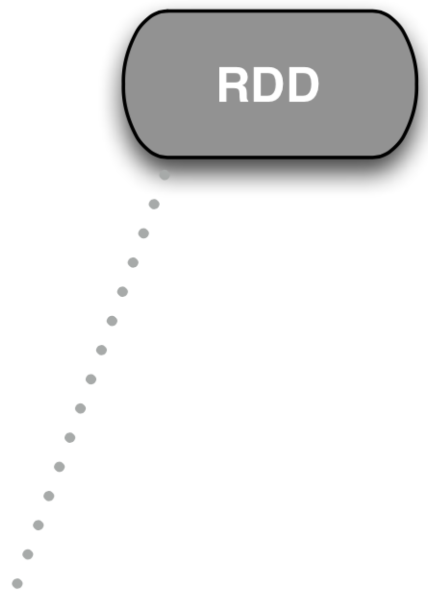
Introduction / Demo

Spark Operations

c08a1132

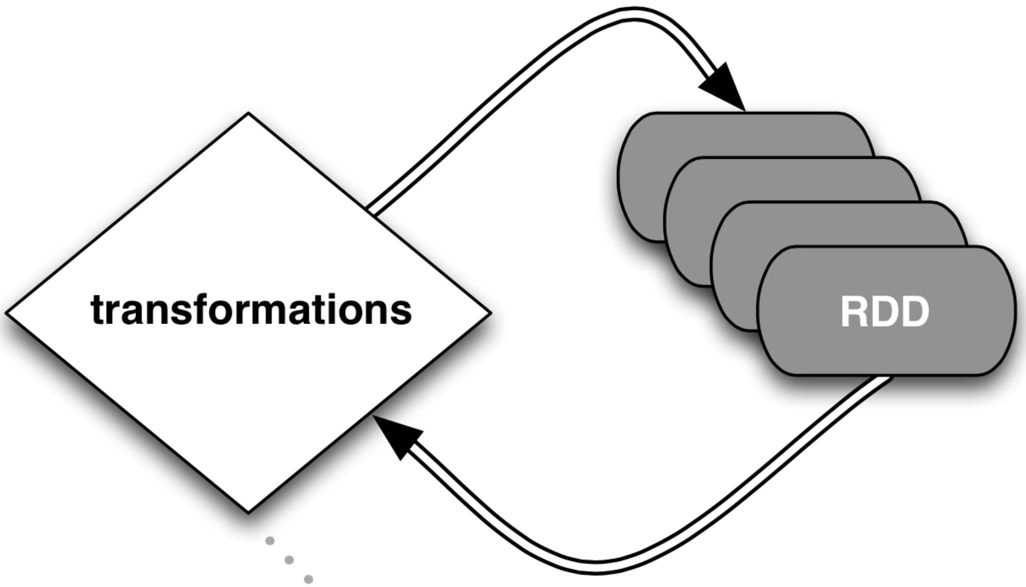


Step by step



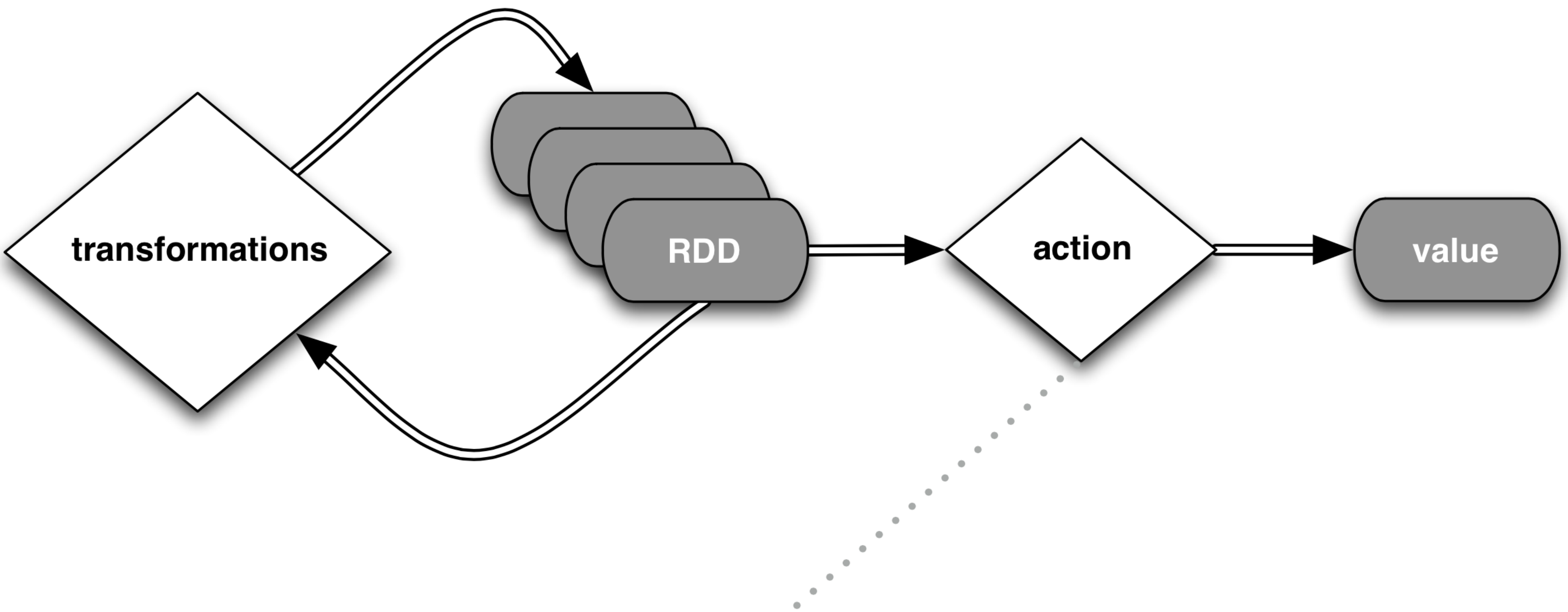

```
// base RDD
textfile = sc.textFile("enrollment.txt")
```

Step by step




```
// transformed RDDs
linesWithZ = textfile.filter(lambda x: x.lower().find("z") > -1)
lineLengths = linesWithZ.map(lambda x: len(x))
totalLength = linesWithZ.reduce(lambda x, y: x + y)
```

Step by step




```
// action 1
linesWithZ.count()
```

API Hooks

- Scala / Java
  - All Java libraries
  - *.jar
  - http://www.scala-lang.org
- Python
  - Anaconda: https://www.anaconda.com/download/
- ...R?
  - If you really want to
  - http://spark.apache.org/docs/latest/sparkr.html

Example: WordCount

Source Code

WordCount.java

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

  public static class Map extends MapReduceBase implements 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, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
      }
    }
  }

  public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

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

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }
}

```

Example: WordCount

Scala:

```
val f = sc.textFile("README.md")
val wc = f.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
wc.saveAsTextFile("wc_out.txt")
```

Python:

```
from operator import add
f = sc.textFile("README.md")
wc = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).reduceByKey(add)
wc.saveAsTextFile("wc_out.txt")
```

Interactive Shells

- Spark creates a SparkSession object (cluster information)
- For either shell: spark
- External programs use a static constructor to instantiate the context
- Pull the SparkContext out spark.SparkContext

```
./bin/spark-shell
```

```
./bin/pyspark
```

Scala:

```
scala> sc
res: spark.SparkContext = spark.SparkContext@470d1f30
```

Python:

```
>>> sc
<pyspark.context.SparkContext object at 0x7f7570783350>
```

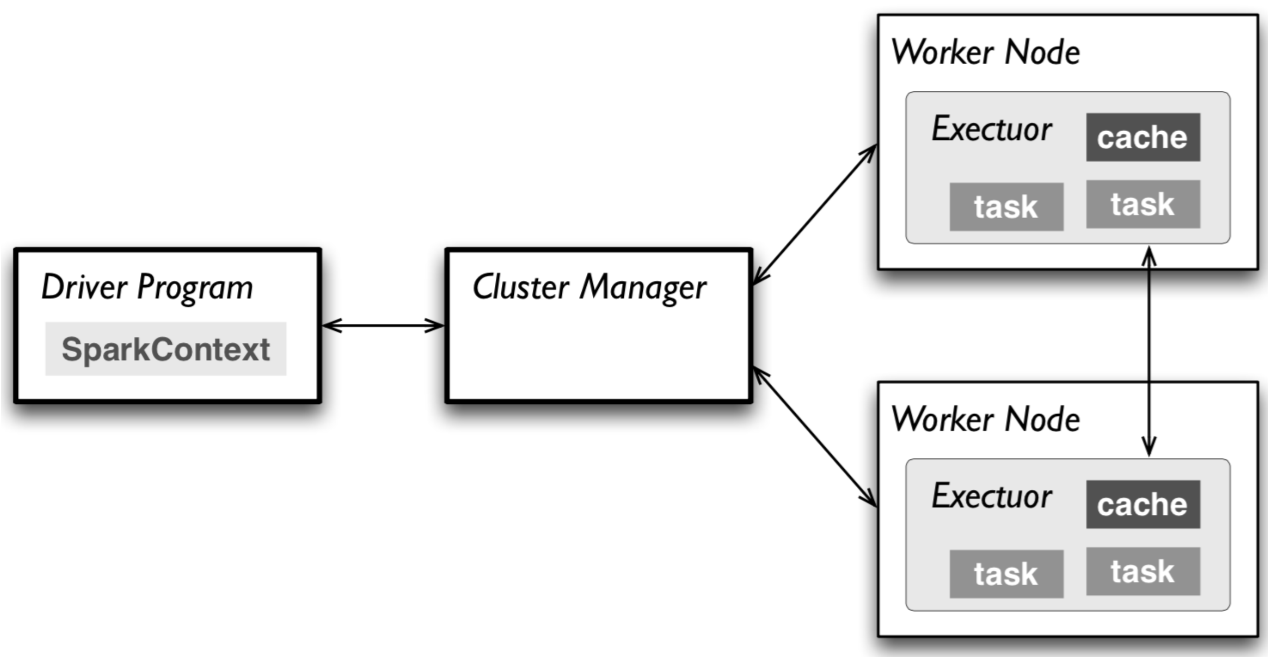
Interactive Shells

- spark-shell --*master*

master	*description*
local	run Spark locally with one worker thread (no parallelism)
local[K]	run Spark locally with K worker threads (ideally set to # cores)
spark://HOST:PORT	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
mesos://HOST:PORT	connect to a Mesos cluster; PORT depends on config (5050 by default)

Interactive Shells

- Master connects to the cluster manager, which allocates resources across applications
- Acquires executors on cluster nodes: worker processes to run computations and store data
- Sends app code to executors
- Sends tasks for executors to run



Resilient Distributed Datasets (RDDs)

Scala:

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val distData = sc.parallelize(data)
distData: spark.RDD[Int] = spark.ParallelCollection@10d13e3e
```

Python:

```
>>> data = [1, 2, 3, 4, 5]
>>> data
[1, 2, 3, 4, 5]

>>> distData = sc.parallelize(data)
>>> distData
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:229
```

Resilient Distributed Datasets (RDDs)

transformation	*description*
map(*func***)**	return a new distributed dataset formed by passing each element of the source through a function *func*
filter(*func***)**	return a new dataset formed by selecting those elements of the source on which *func* returns true
flatMap(*func***)**	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)
sample(*withReplacement, fraction, seed***)**	sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*
union(*otherDataset***)**	return a new dataset that contains the union of the elements in the source dataset and the argument
distinct([*numTasks***]))**	return a new dataset that contains the distinct elements of the source dataset

Resilient Distributed Datasets (RDDs)

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

distFile is a collection of lines

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

Resilient Distributed Datasets (RDDs)

Scala:

```scala
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

closures

Python:

```python
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

Resilient Distributed Datasets (RDDs)

action	*description*
reduce(*func***)**	aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
collect()	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
count()	return the number of elements in the dataset
first()	return the first element of the dataset – similar to *take(1)*
take(*n***)**	return an array with the first *n* elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
takeSample(*withReplacement, fraction, seed***)**	return an array with a random sample of *num* elements of the dataset, with or without replacement, using the given random number generator seed

Resilient Distributed Datasets (RDDs)

Scala:

```
val f = sc.textFile("README.md")
val words = f.flatMap(l => l.split(" ")).map(word => (word, 1))
words.reduceByKey(_ + _).collect.foreach(println)
```

Python:

```
from operator import add
f = sc.textFile("README.md")
words = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1))
words.reduceByKey(add).collect()
```

Broadcast Variables

- Spark's version of Hadoop's `DistributedCache`
- Read-only variable cached on each node
- Spark [internally] distributed broadcast variables in such a way to minimize communication cost

Broadcast Variables

Scala:

```scala
val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar.value
```

Python:

```python
broadcastVar = sc.broadcast(list(range(1, 4)))
broadcastVar.value
```

Accumulators

- Spark's version of Hadoop's `Counter`
- Variables that can only be added through an associative operation
- Native support of numeric accumulator types and standard mutable collections
  - Users can extend to new types
- Only driver program can *read* accumulator value

Accumulators

Scala:

```
val accum = sc.accumulator(0)
sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)

accum.value
```

driver-side

Python:

```
accum = sc.accumulator(0)
rdd = sc.parallelize([1, 2, 3, 4])
def f(x):
   global accum
   accum += x

rdd.foreach(f)

accum.value
```

Key/Value Pairs

Scala:

```scala
val pair = (a, b)

    pair._1 // => a
    pair._2 // => b
```

Python:

```python
pair = (a, b)

    pair[0] # => a
    pair[1] # => b
```

Java:

```java
Tuple2 pair = new Tuple2(a, b);

    pair._1 // => a
    pair._2 // => b
```

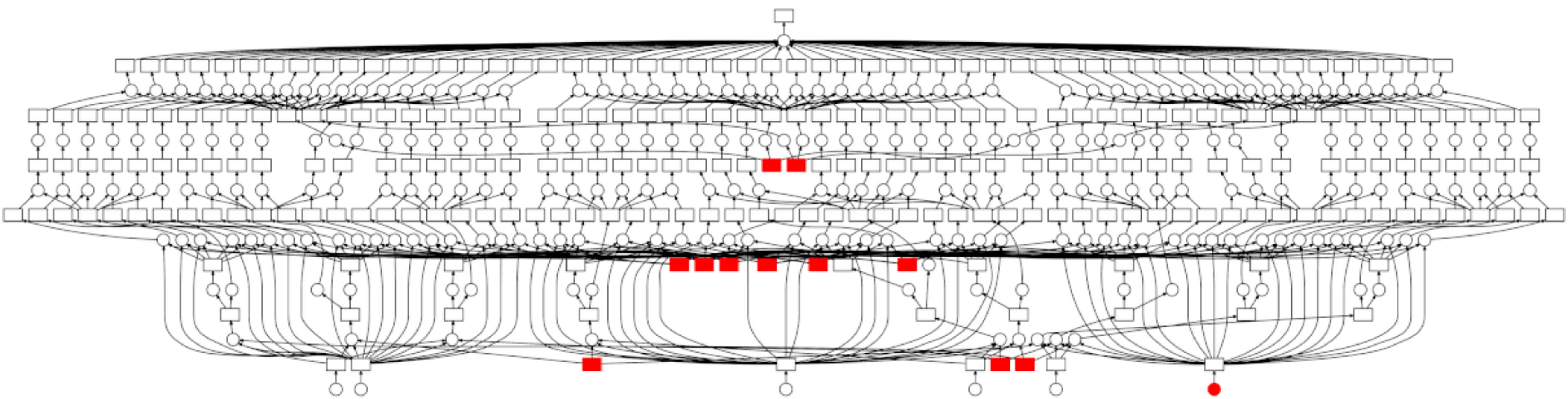

Dask

Dask

- **Exclusive to the Python ecosystem** (sorry JVM / R folks)
- First released in 2018
- Tight integration with the SciPy ecosystem
  - NumPy
  - pandas
  - scikit-learn
  - matplotlib / bokeh
  - RAPIDS (more recently)

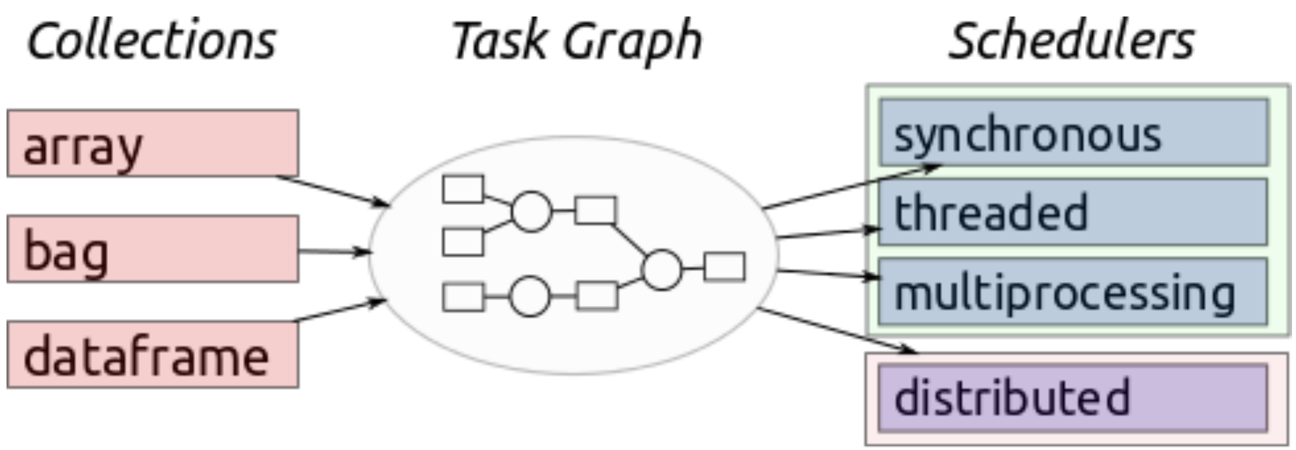
Dask

- Philosophy: parallel computing with minimal fanfare
  - Distributed computing is almost an accidental byproduct
- Uses a sophisticated but lightweight task scheduler
  - Builds a dependency graph of tasks (kind of like a compiler)



Task Scheduler

- Dask has three primary data structures:
  - Array (modeled after NumPy)
  - DataFrame (modeled after pandas)
  - Bag (modeled after lists)
- Uses delayed and futures to perform lazy evaluation while building a dependency graph of tasks
- The scheduler then executes the task graph—in sequence, multithreaded, multiprocessed, or distributed.



Dask APIs

- Major, major effort to make APIs as seamless as possible
  - Array follows NumPy
  - DataFrame follows pandas
  - Bag follows map/filter/groupby/reduce common in Spark and Python lists
  - Dask-ML follows scikit-learn
  - Delayed wraps generic Python code
  - Futures follow concurrent.futures from standard library

```python
# Arrays implement the Numpy API
import dask.array as da
x = da.random.random(size=(10000, 10000),
                     chunks=(1000, 1000))
x + x.T - x.mean(axis=0)
```

```python
# Dataframes implement the Pandas API
import dask.dataframe as dd
df = dd.read_csv('s3://.../2018-*-*.csv')
df.groupby(df.account_id).balance.sum()
```

```python
# Dask-ML implements the Scikit-Learn API
from dask_ml.linear_model \
  import LogisticRegression
lr = LogisticRegression()
lr.fit(train, test)
```

Comparisons to Spark

- Entire write-up on the Dask website
  - https://docs.dask.org/en/latest/spark.html

Want to learn more?

- The documentation on the dask website is **second to none**
- https://docs.dask.org/

Fun Fact

- Both Spark and Dask have pre-built VMs available on Google Cloud

Project 0

- Out later today!
- Due **Tuesday, January 26 at 11:59pm**
- Can’t use nltk, breeze, or other NLP-specific packages
  - Really, you won’t need them
- Spark / Dask, & “NLP”
  - Count words in documents (term frequencies)
  - Incorporate stopword filtering (will **need** broadcast variables for this)
  - Truncate out punctuation
  - Implement TF-IDF for improved word counting
  - **CANNOT STORE VOCABULARY LOCALLY.** Need to distribute / parallelize!

Project 0

- **Pay attention to the requirements of the deliverables**
  - Incorrectly-named or formatted JSON files will cause autograder to fail
  - Name GitHub repo correctly
  - Include README and CONTRIBUTORS files
  - Practice using git (commit, push, branch, merge) and GitHub functionality (issues, milestones, pull requests)

Questions?