Big Data Overview

Section Overview

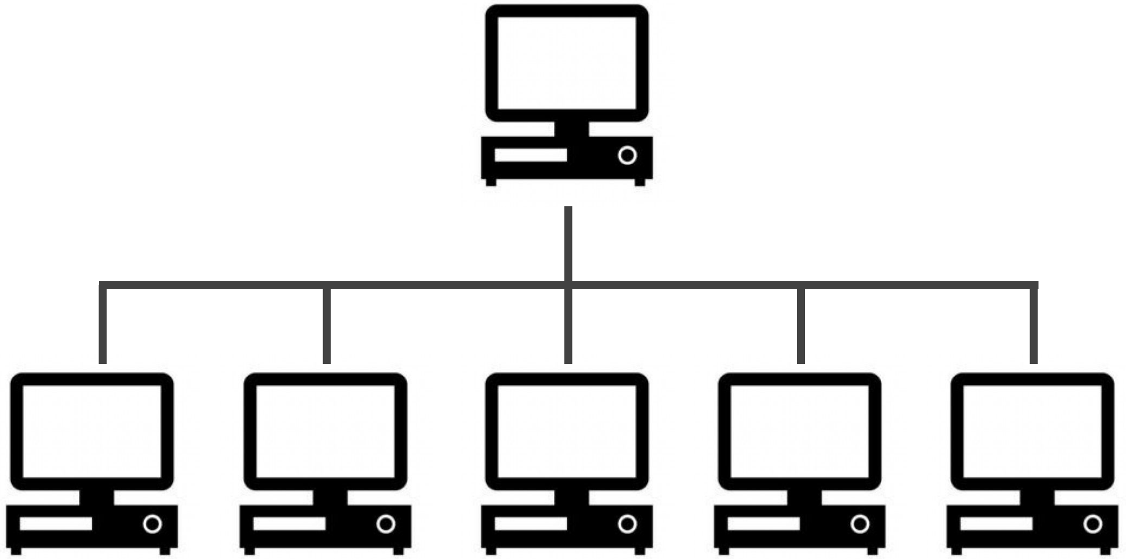
- Explanation of Hadoop, MapReduce, Spark, and PySpark
- Local versus Distributed Systems
- Overview of Hadoop Ecosystem
- Detailed overview of Spark
- Set-up on Amazon Web Services
- Resources on other Spark Options
- Jupyter Notebook hands-on code with PySpark and RDDs

Big Data

- We've worked with Data that can fit on a local computer, in the scale of 0-8 GB.
- But what can we do if we have a larger set of data?
    - Try using a SQL database to move storage onto hard drive instead of RAM
    - Or use a distributed system, that distributes the data to multiple machines/computer.

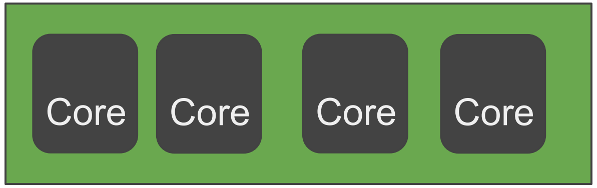
Local versus Distributed

Local

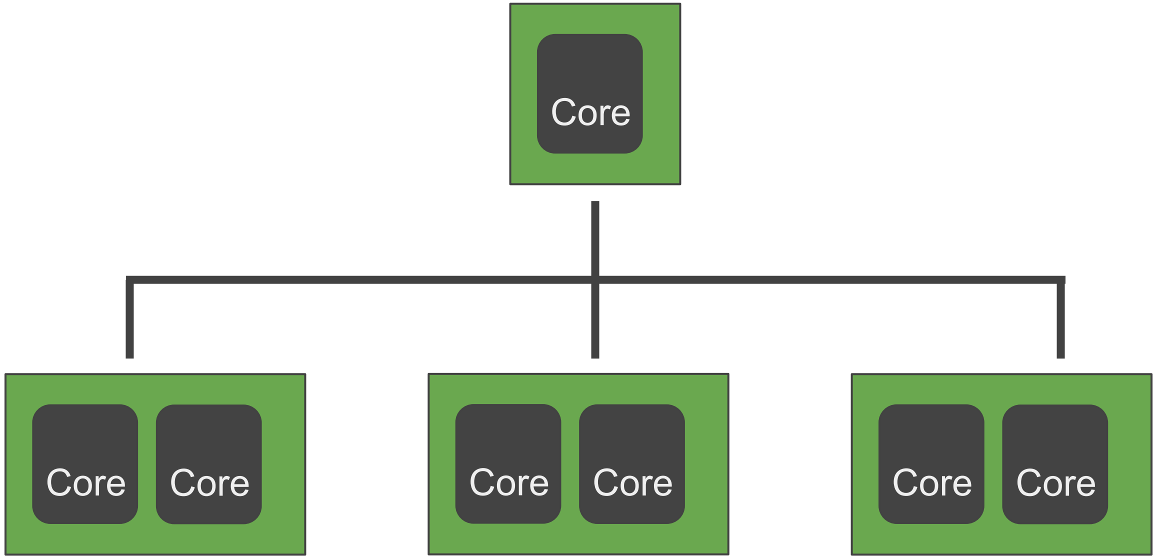


Distributed

Local versus Distributed



Local



Distributed

Big Data

- A local process will use the computation resources of a single machine
- A distributed process has access to the computational resources across a number of machines connected through a network
- After a certain point, it is easier to scale out to many lower CPU machines, than to try to scale up to a single machine with high a CPU

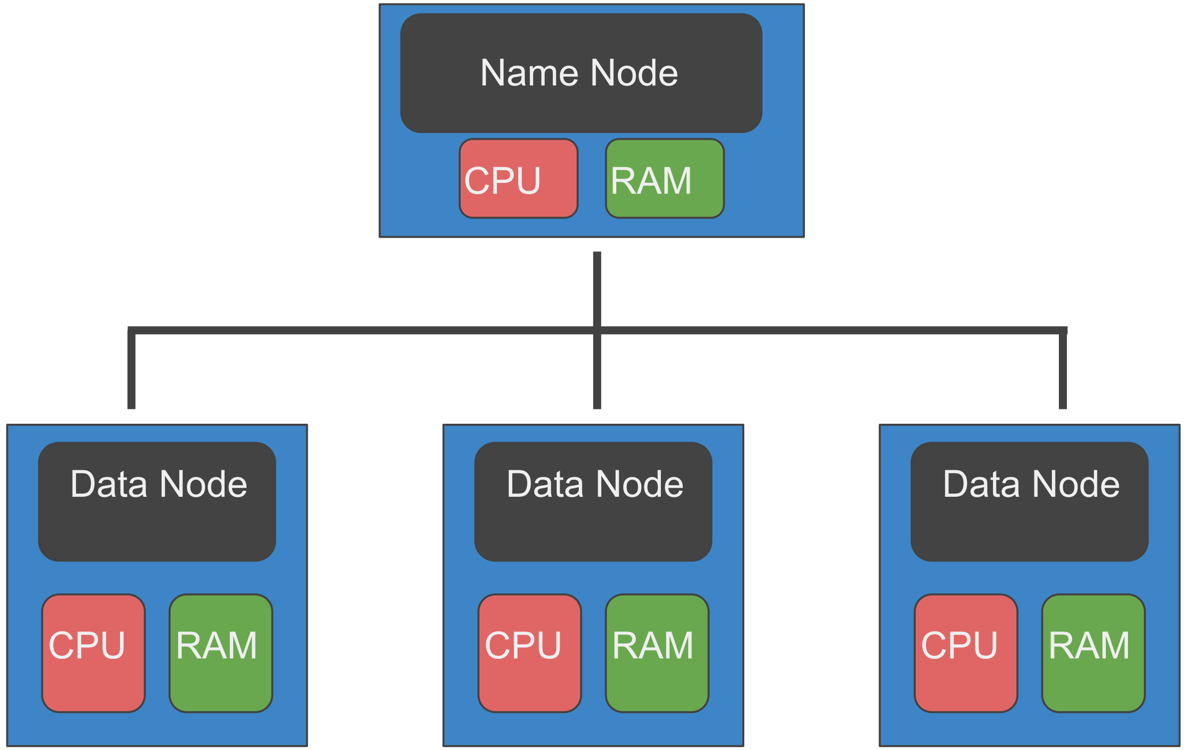
Big Data

- Distributed machines also have the advantage of easily scaling, you can just add more machines
- They also include fault tolerance, if one machine fails, the whole network can still go on.
- Let’s discuss the typical format of a distributed architecture that uses Hadoop

Hadoop

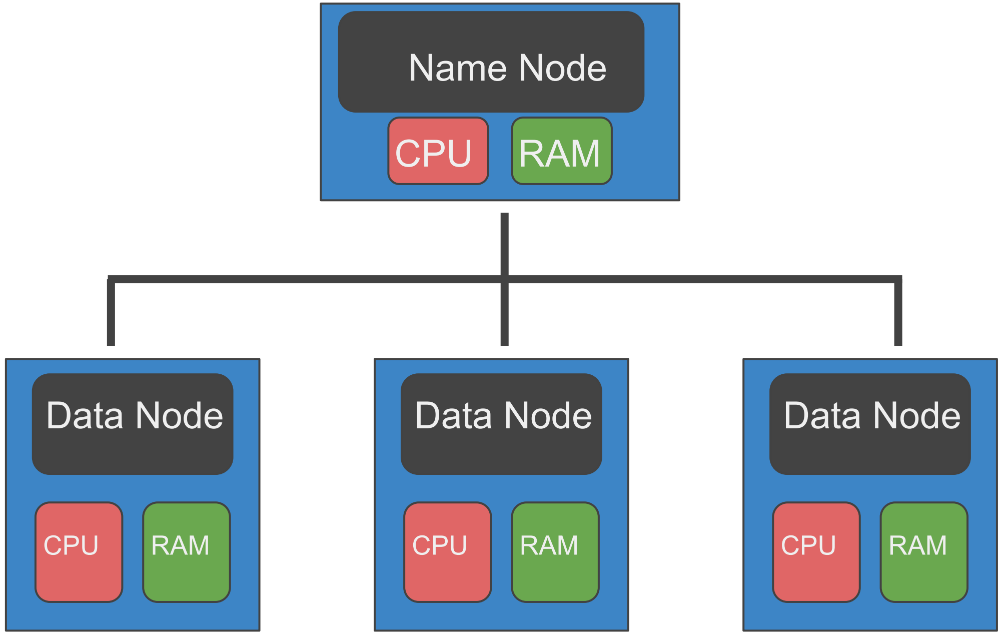
- Hadoop is a way to distribute very large files across multiple machines.
- It uses the Hadoop Distributed File System (HDFS)
- HDFS allows a user to work with large data sets
- HDFS also duplicates blocks of data for fault tolerance
- It also then uses MapReduce
- MapReduce allows computations on that data

Distributed Storage - HDFS



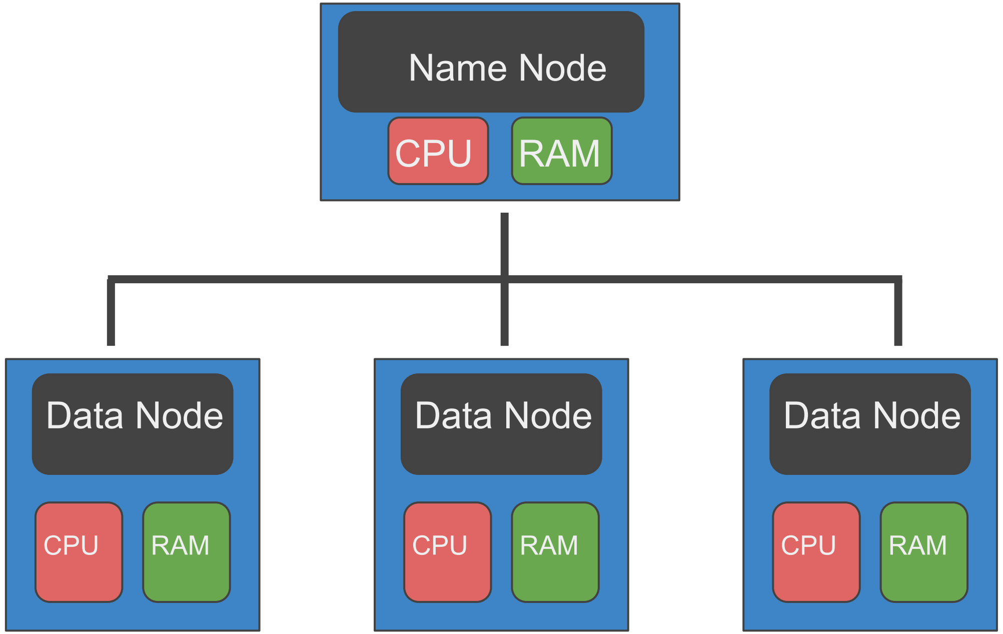
Distributed Storage - HDFS

- HDFS will use blocks of data, with a size of 128 MB by default
- Each of these blocks is replicated 3 times
- The blocks are distributed in a way to support fault tolerance



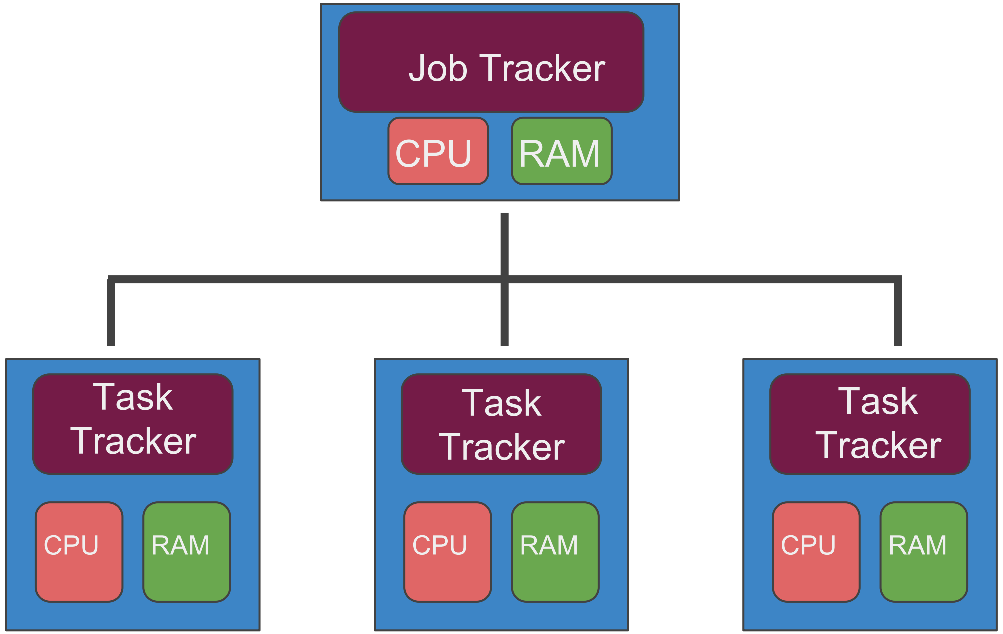
Distributed Storage - HDFS

- Smaller blocks provide more parallelization during processing
- Multiple copies of a block prevent loss of data due to a failure of a node



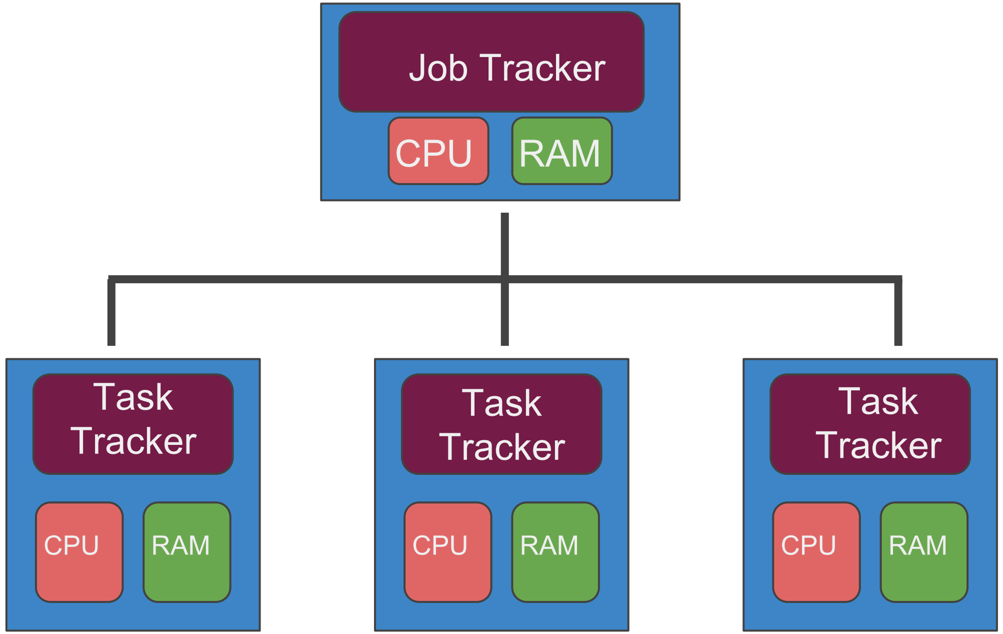
MapReduce

- MapReduce is a way of splitting a computation task to a distributed set of files (such as HDFS)
- It consists of a Job Tracker and multiple Task Trackers



MapReduce

- The Job Tracker sends code to run on the Task Trackers
- The Task trackers allocate CPU and memory for the tasks and monitor the tasks on the worker nodes



Big Data

- What we covered can be thought of in two distinct parts:
  - Using HDFS to distribute large data sets
  - Using MapReduce to distribute a computational task to a distributed data set
- Next we will learn about the latest technology in this space known as Spark.
- Spark improves on the concepts of using distribution

Up next we’ll learn about Spark!

Spark Overview

Spark

- This lecture will be an abstract overview, we will discuss:
  - Spark
  - Spark vs MapReduce
  - Spark RDDs
  - RDD Operations
- Don’t worry about having to understand all the operations, we will review and cover this again when we actually program with Spark and Python

Spark

- Spark is one of the latest technologies being used to quickly and easily handle Big Data
- It is an open source project on Apache
- It was first released in February 2013 and has exploded in popularity due to it's ease of use and speed
- It was created at the AMPLab at UC Berkeley

Spark

- You can think of Spark as a flexible alternative to MapReduce
- Spark can use data stored in a variety of formats
  - Cassandra
  - AWS S3
  - HDFS
  - And more

Spark vs MapReduce

- MapReduce requires files to be stored in HDFS, Spark does not!
- Spark also can perform operations up to 100x faster than MapReduce
- So how does it achieve this speed?

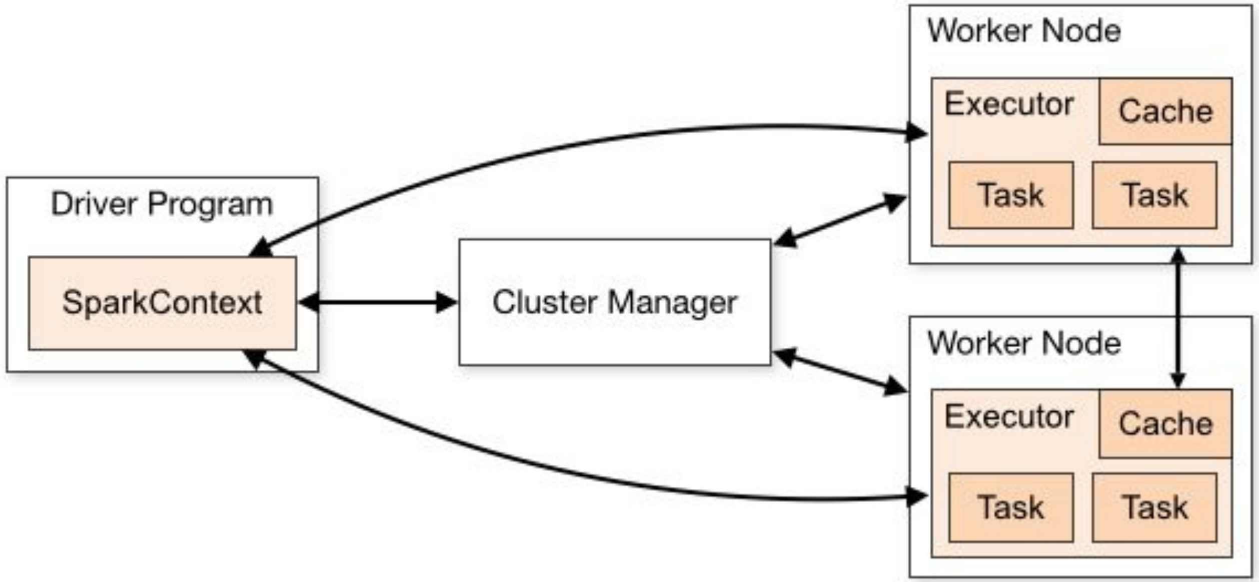
Spark vs MapReduce

- MapReduce writes most data to disk after each map and reduce operation
- Spark keeps most of the data in memory after each transformation
- Spark can spill over to disk if the memory is filled

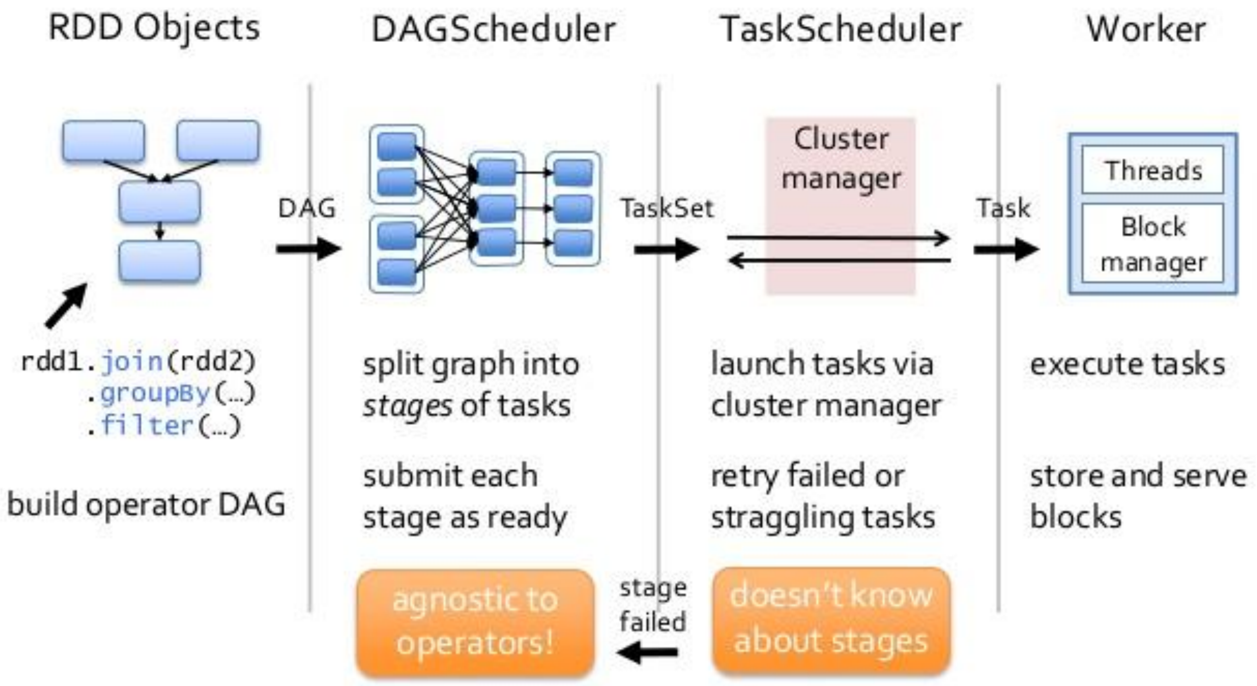
Spark RDDs

- At the core of Spark is the idea of a Resilient Distributed Dataset (RDD)
- Resilient Distributed Dataset (RDD) has 4 main features:
    - Distributed Collection of Data
    - Fault-tolerant
    - Parallel operation - partioned
    - Ability to use many data sources

Spark RDDs


Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task
Worker Node
Executor
Cache
Task
Task

Spark RDDs


RDD Objects
DAGScheduler
TaskScheduler
Worker
DAG
TaskSet
Cluster manager
Task
Threads
Block manager
rdd1.join(rdd2)
.groupBy(…)
.filter(…)
build operator DAG
split graph into *stages* of tasks
submit each stage as ready
launch tasks via cluster manager
retry failed or straggling tasks
execute tasks
store and serve blocks
agnostic to operators!
stage failed
doesn't know about stages

Spark RDDs

- RDDs are immutable, lazily evaluated, and cacheable
- There are two types of RDD operations:
  - Transformations
  - Actions

Spark RDDs

- Basic Actions
  - First
  - Collect
  - Count
  - Take

Spark RDDs

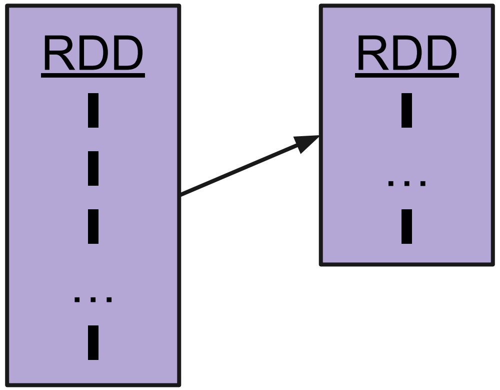
- Collect - Return all the elements of the RDD as an array at the driver program.
- Count - Return the number of elements in the RDD
- First - Return the first element in the RDD
- Take - Return an array with the first n elements of the RDD

Spark RDDs

- Basic Transformations
  - Filter
  - Map
  - FlatMap

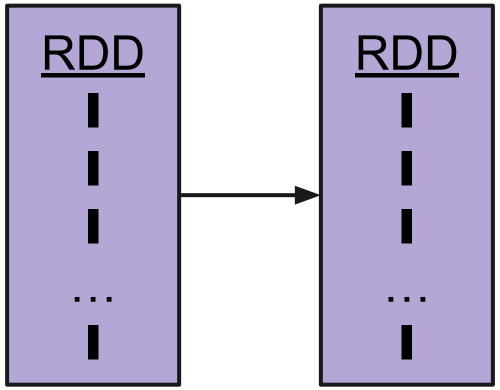
Spark RDDs

- RDD.filter()
  - Applies a function to each element and returns elements that evaluate to true



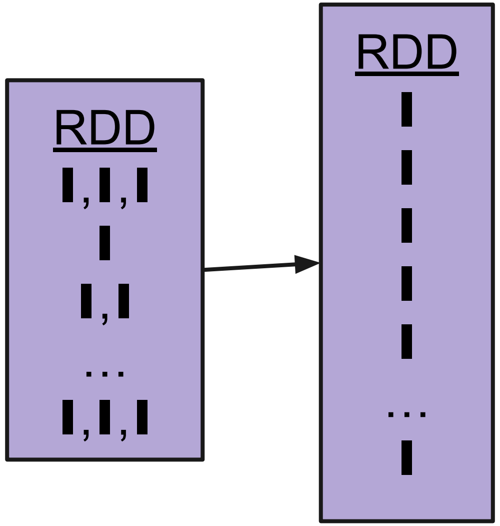
Spark RDDs

- RDD.map()
  - Transforms each element and preserves # of elements, very similar idea to pandas .apply()



Spark RDDs

- RDD.flatMap()
  - Transforms each element into 0-N elements and changes # of elements



Map vs FlatMap

- Map()
  - Grabbing first letter of a list of names
- FlatMap()
  - Transforming a corpus of text into a list of words
- We will show many more examples when programing with PySpark

Pair RDDs

- Often RDDs will be holding their values in tuples
  - (key,value)
- This offers better partitioning of data and leads to functionality based on reduction

Reduce and ReduceByKey

- Reduce()
  - An action that will aggregate RDD elements using a function that returns a single element
- ReduceByKey()
  - An action that will aggregate Pair RDD elements using a function that returns a Pair RDD
- These ideas are similar to a Group By operation

Big Data

- Spark is being continually developed and new releases come out often!
- The Spark Ecosystem now includes:
    - Spark SQL
    - Spark DataFrames
    - MLlib
    - GraphX
    - Spark Streaming

Big Data

- Now we’ve learned enough to get started!
- We’re now going to show you how to set up an Amazon Web Services account to get Spark up and running!
- We’ll also have text article lecture for some other options in case you don’t want to use AWS.

AWS Account Set-Up

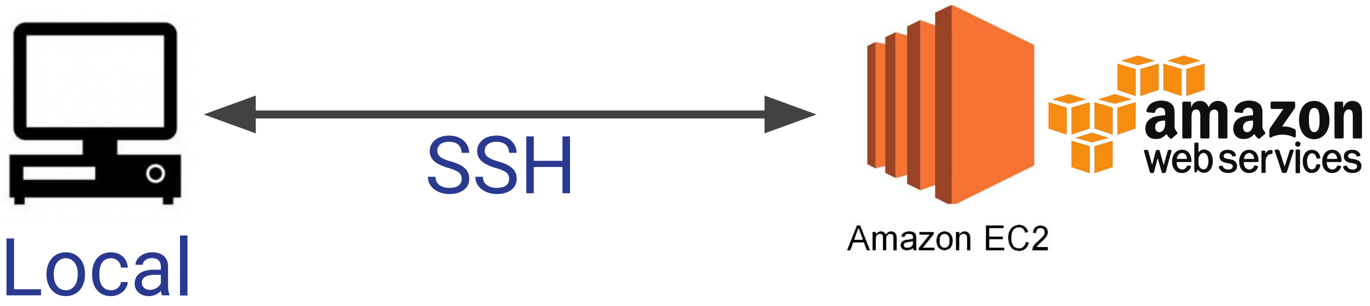
- Go to:
    - https://aws.amazon.com/free
- Then click on Create Free Account
- Sign up with an email address
- Then fill out the profile information

- The profile information:
  - Contact Information
  - Billing Information
  - ID Verification
  - Choose free support plan
- Next lecture we will explore AWS and create an EC2 instance

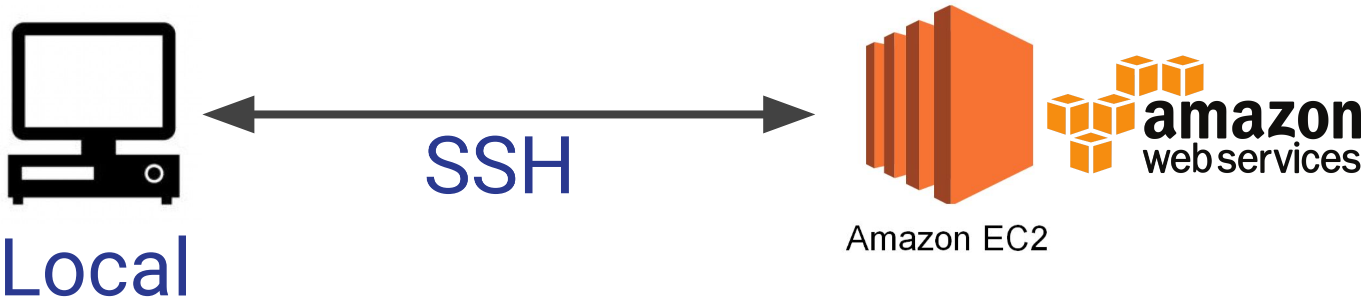
EC2 Instance Set-up

- Now that we have our AWS account we will create an EC2 Instance.
- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud.
- We can basically think of it as a virtual computer we can access through the internet.

- So here is our plan:
  - Create EC2 Instance on AWS
  - Use SSH to connect to EC2 over internet
  - SSH is different for Windows vs Mac/Linux
  - Set-up Spark and Juptyer on EC2 Instance



- SSH (Secure Shell Connection)
    - Watch this lecture all the way through for Windows
    - Skip to next lecture after EC2 Set-up for Mac/Linux
- Our goal is to remotely connect to the command line of our virtual machine on Amazon EC2



PIERIAN DATA

Login to your AWS console at:
aws.amazon.com

PySpark Set-up

SSH with Mac and Linux

- Skip this lecture if you are on Windows, you should have connected to your instance already from the previous lecture
- We’ve created our EC2 instance using AWS console
- You should have also downloaded the .pem file
- Now we are going to connect to our instance through our terminal using SSH

- Make sure you can locate your .pem file
    - Recommend you relocate it to your Desktop
- Make sure you have the DNS address of your EC2 instance
- Check the resource link for the step by step instructions from Amazon
- Let’s get started by opening our terminal