Northeastern

EECE5645
Parallel Processing for Data Analytics

Lecture 2: Introduction to Spark

What is Spark?

- A general engine for **large-scale data processing**
- **Fast**, **expressive cluster computing** system compatible with Apache Hadoop
- Improves **efficiency** through:
  - In-memory computing primitives
  - General computation graphs

→ Up to 100× faster

- Improves **usability** through:
  - Rich APIs in Java, Scala, Python
  - Interactive shell

→ Often 2-10× less code

Ecosystem

- Growing community with more than 200 companies contributing
- Details, tutorials, videos: http://spark.apache.org/

Why are we using it in EECE5645?

- Usability: easy to go from Python to Spark
- Makes parallelism easy!
- Good tool to know!

Usability

Integrated with

- Java
- Scala
- **Python**

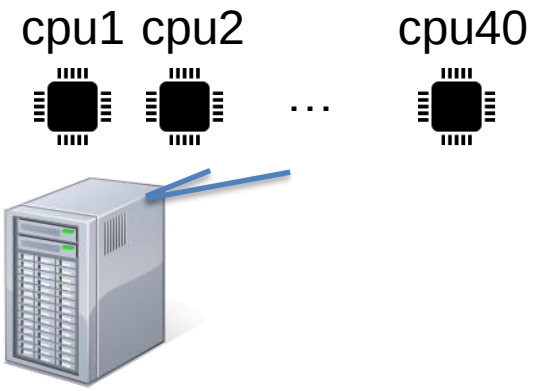
Programmer's point of view:

- "Normal" python program, with "special" parallel data structures
- Easy to write short, clear map/reduce operations
- Full expressibility of java/scala/python "included"

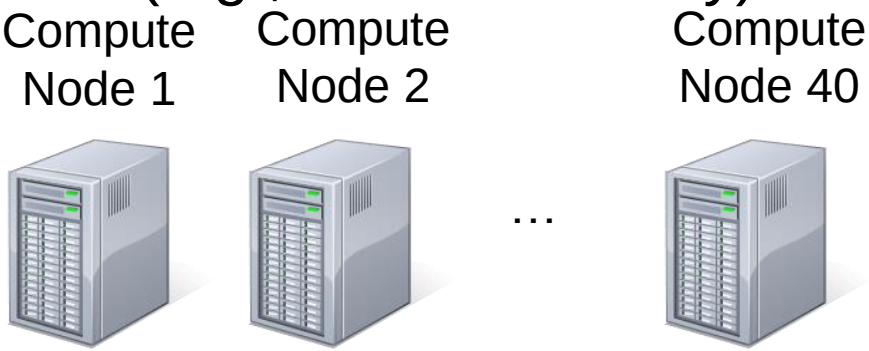
Portability

Same code runs on different environments

- Multi-threading on single machine



- Execution over distributed cluster (e.g., NEU Discovery)



- Amazon AWS, GCP, Azure
- …

Efficiency

- **Persistence.** Results stay **in memory**, vs. read and write from hard disk (e.g., Hadoop).

- **Built-In** fault tolerance: if a machine crashes and intermediate computations lost, Spark automatically recomputes it

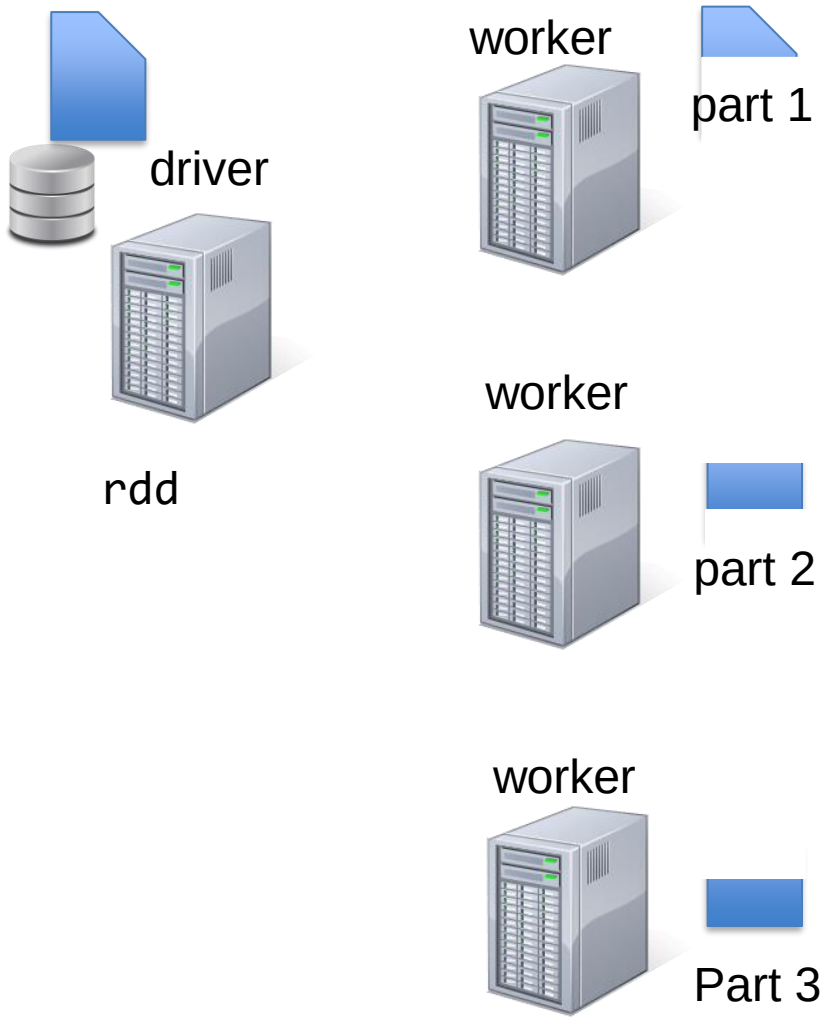
Outline

- ❑ Overview
- ❑ Tour of Operations
- ❑ Job execution
- ❑ Standalone Programs

Key Idea

- Distributed data is just another collection type (like, e.g. python lists or dictionaries)

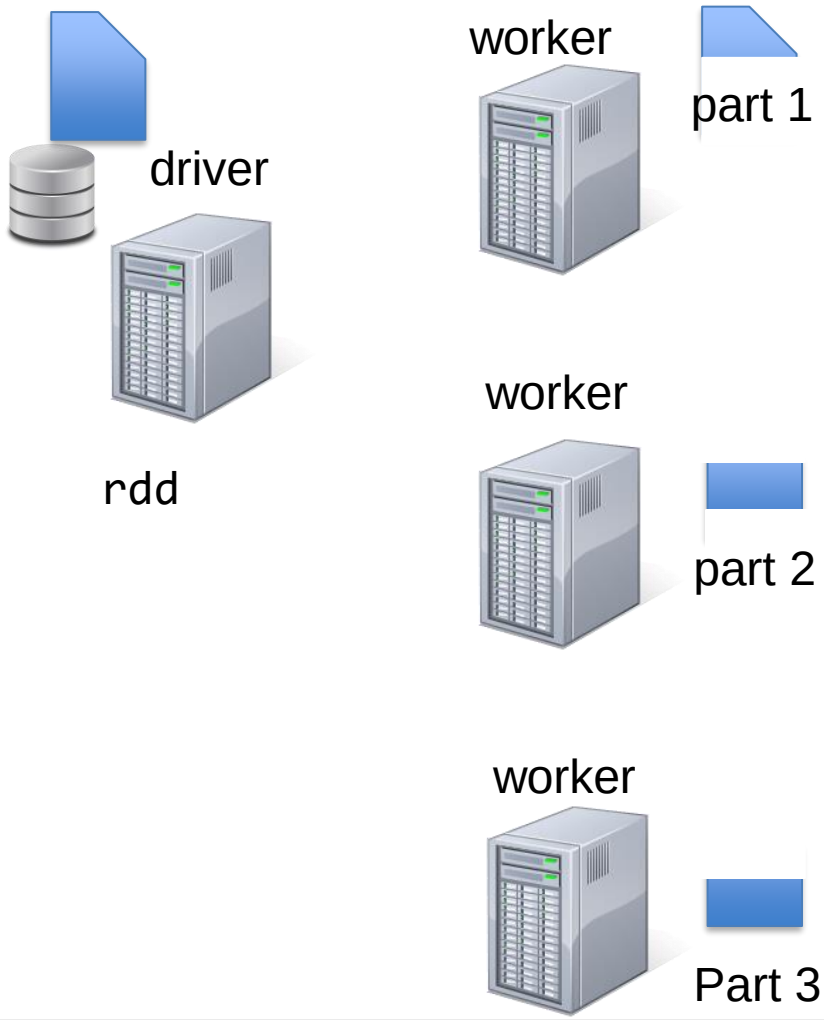
- **Resilient Distributed Datasets (RDDs):** Appear as variables in your program, but are actually stored over **multiple machines.**




```
…
rdd=sc.textFile(“WarAndPeace.txt”)
…
```

Resilient Distributed Datasets (RDDs)

- **Immutable** collections of objects spread across a cluster
- Built through **parallel transformations** (map, filter, etc)
- Automatically **rebuilt on failure**
- Controllable **persistence** (e.g. caching in RAM)



Types of Operations

- **Transformations** (e.g. map, filter, groupBy, join)
  - operations to build RDDs from other RDDs

- **Actions** (e.g. count, collect, save)
  - Return a result to driver program
  - Write a result to hard disk

Outline

- ❑ Overview
- ❑ (Very Quick) Tour of Operations
- ❑ Job execution
- ❑ Standalone Programs

Learning Spark

- Easiest Way: Spark interpreter (`pyspark`)
- Runs in local mode with 1 thread by default, but can be controlled through `master` command line option

```
pyspark --master "local[20]"
```

(more on this soon)

SparkContext

- Starting point for spark functionalities

- In pyspark interpreter, automatically loaded variable:

```
sc
```

- In standalone programs, you create your own

Creating an RDD

```
# Turn a local collection into an RDD
sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS
sc.textFile("file.txt")
sc.textFile("directory/*.txt")
sc.textFile("hdfs://namenode:9000/path/file")
```

More on textfile

```
myrdd=sc.textFile("file.txt")
```

- Think: `myrdd` is a list (sort of)
- Elements in the list are **lines of the file**
- The list is **distributed over multiple machines**
- The list is **immutable**
  - You cannot change it,
  - Cannot see individual elements (e.g., 5th element)
  - Can only interact with it through specific ops

Basic Transformations

```
nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
squares = nums.map(lambda x: x*x)   # => {1, 4, 9}

# Keep elements passing a predicate
even = squares.filter(lambda x: x % 2 == 0) # => {4}

# Map each element to zero or more others
nums.flatMap(lambda x: range(0, x))  # => {0, 0, 1, 0, 1, 2}
```

Range function (list of numbers 0, 1, …, x-1)

Basic Actions

```
nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
nums.collect() # => [1, 2, 3]

# Return first K elements
nums.take(2)   # => [1, 2]

# Count number of elements
nums.count()   # => 3

# Merge elements with an associative function
nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

More Details

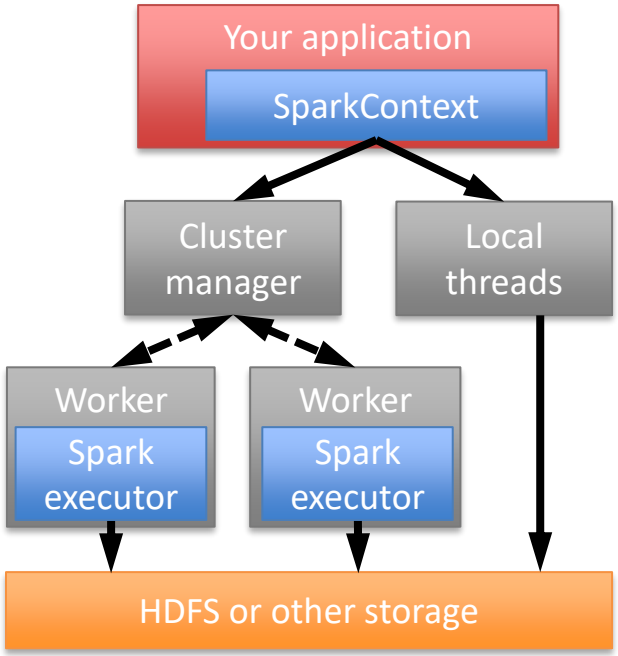
- Spark supports lots of transforms and actions
- Full programming guide: spark.apache.org/documentation
- We will dive into map, reduce, and many more transforms and actions in upcoming lectures

Outline

- ❑ Overview
- ❑ Tour of Operations
- ❑ Job execution
- ❑ Standalone Programs

Software Components

- Spark shows up as a library in your program
- Runs tasks **locally** or on a **cluster**
  - **Standalone cluster**
  - Mesos Cluster
  - YARN Cluster



Outline

- ❑ Overview
- ❑ Tour of Operations
- ❑ Job execution
- ❑ Standalone Programs

Create a SparkContext within Standalone Program

```
from pyspark import SparkContext

sc = SparkContext('masterUrl', 'name')
```

Cluster URL, or local / local[N]

App name

Complete App: Python

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( 'local[10]', 'WordCount')
    lines = sc.textFile(sys.argv[1])

    lines.flatMap(lambda s: s.split()) \
         .map(lambda word: (word, 1)) \
         .reduceByKey(lambda x, y: x + y) \
        .sortBy(lambda pair:pair[1], ascending=False) \
         .saveAsTextFile(sys.argv[2])
```

More to Come in the Next Few Weeks

- Dive into map, reduce, joins and other operations
- Partitioning & Caching
- How to launch Spark on Discovery