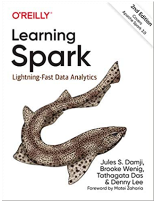
UMD DATA605 - Big Data Systems

9.1: Apache Spark: Principles

- **Instructor**: Dr. GP Saggese, gsaggese@umd.edu
- **References**:
  - Concepts in the slides
  - Academic paper
    - “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”, 2012
  - Mastery
    - “Learning Spark: Lightning-Fast Data Analytics” (2nd Edition)
    - Not my favorite, but free



O'REILLY

2nd Edition Covers Apache Spark 3.0

Learning Spark

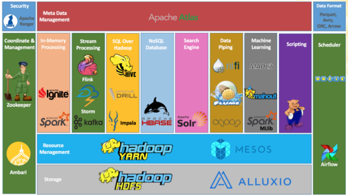
Lightning-Fast Data Analytics

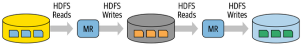
Jules S. Damji, Brooke Wenig, Tathagata Das & Denny Lee

Foreword by Matei Zaharia

Hadoop MapReduce: Shortcomings

- Hadoop is hard to administer
  - Many layers (HDFS, Yarn, Hadoop, . . .)
  - Extensive configuration
- Hadoop is hard to use
  - Verbose API
  - Limited language support (e.g., Java is native)
  - MapReduce jobs read / write data on disk





- Large but fragmented ecosystem
  - No native support for:
    - Machine learning
    - SQL
    - Streaming
    - Interactive computing
  - New systems developed on Hadoop for new workloads
    - E.g., Apache Hive, Storm, Impala, Giraph, Drill

(Apache) Spark

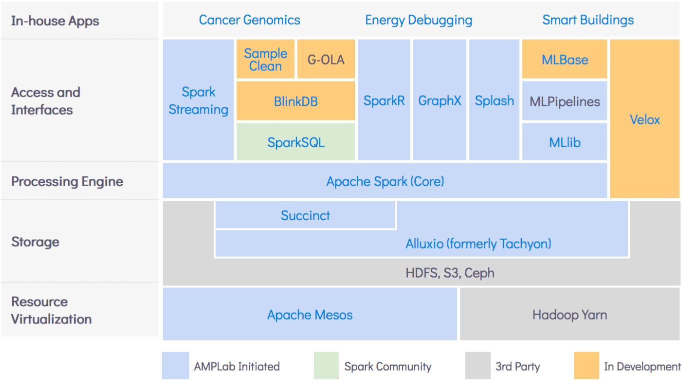
- **Open-source**
  - DataBrick monetizes it ($100B startup in 2025)
- **General processing engine**
  - Large set of operations beyond `Map()` and `Reduce()`
  - Combine operations in any order
  - Computation organized as a DAG, decomposed into parallel tasks
  - Scheduler/optimizer for parallel workers
- **Supports several languages**
  - Java, Scala (preferred), Python supported through bindings
- **Data abstraction**
  - Resilient Distributed Dataset (RDD)
  - DataFrames, Datasets built on RDDs
- **Fault tolerance through RDD lineage**
- **In-memory computation**
  - Keep intermediate results in memory, if possible

Berkeley: From Research to Companies

- Pathway from lab innovation to startups
    - Students and researchers creating companies from lab systems
    - Focus on data-intensive systems and machine learning
    - Open-source ecosystems enabling broad adoption
- **AMPLab**
    - Collaborative projects creating systems like Spark
    - Industry engagement guiding real-world impact
- **RISELab**
    - Shift to systems supporting AI, security, and automation
    - Platforms like Ray and ML-focused infrastructure

Berkeley AMPLab Data Analytics Stack

- So many tools that they have their own Big Data stack! https://amplab.cs.berkeley.edu/software/



In-house Apps: Cancer Genomics, Energy Debugging, Smart Buildings

Access and Interfaces: Spark Streaming, Sample Clean, G-OLA, BlinkDB, SparkSQL, SparkR, GraphX, Splash, MLBase, MLPipelines, MLlib, Velox

Processing Engine: Apache Spark (Core)

Storage: Succinct, Alluxio (formerly Tachyon), HDFS, S3, Ceph

Resource Virtualization: Apache Mesos, Hadoop Yarn

AMPLab Initiated

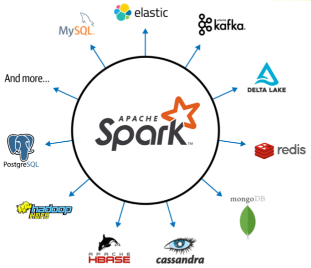
Spark Community

3rd Party

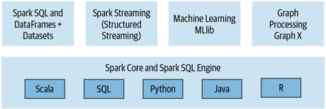
In Development

Apache Spark: Introduction

- **Unified stack**
  - Different computation models in a single framework
  - **Spark SQL**
    - ANSI SQL compliant
    - Work with structured relational data
  - **Spark MLlib**
    - Build ML pipelines
    - Support popular ML algorithms
    - Built on Spark DataFrame
  - **Spark Streaming**
    - Handle continually growing tables
    - Treat tables as static
  - **GraphX**
    - Manipulate graphs
    - Perform graph-parallel computation
- **Extensibility**
  - Read from many sources
  - Write to many backends



One computation engine



Spark SQL and DataFrames + Datasets

Spark Streaming (Structured Streaming)

Machine Learning MLlib

Graph Processing Graph X

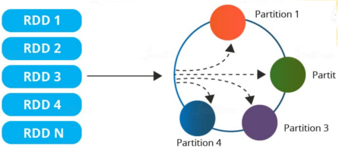
Spark Core and Spark SQL Engine

Scala | SQL | Python | Java | R

General purpose applications

Resilient Distributed Dataset (RDD)

- **Resilient Distributed Dataset (RDD)**
  - Collection of data elements
  - Partitioned across nodes
  - Operated on in parallel
  - Fault-tolerant
  - In-memory / serializable

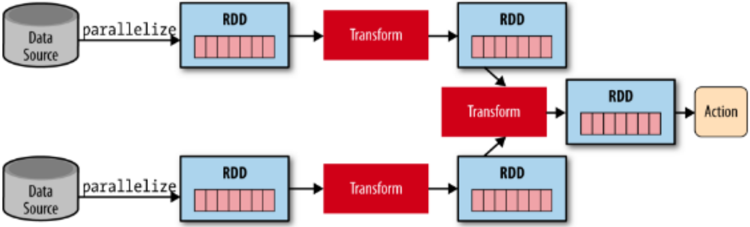


- **Applications**
  - Best for applying the same operation to all dataset elements (vectorized)
  - Less suitable for asynchronous fine-grained updates to shared state
    - E.g., updating one value in a dataframe
- **Ways to create RDDs**
  - Reference data in external storage
    - E.g., file-system, HDFS, HBase
  - Parallelize an existing collection in your driver program
  - Transform RDDs into other RDDs

SCIENCE ACADEMY

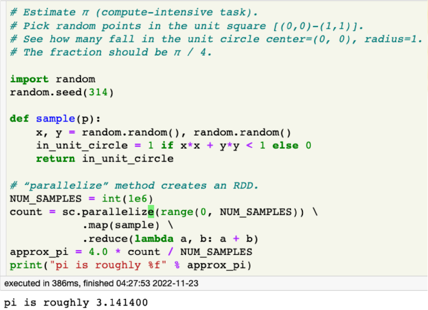
Transformations vs Actions

- Transformations
  - Lazy evaluation
  - Compute only when an Action requires it
  - Build a graph of transformations
- Actions
  - Aka “materialize”
  - Force calculations on RDDs and return values



Spark Example: Estimate Pi

- **Goal**
  - Estimate π using random sampling in the unit square
  - Fraction of points inside the unit circle approximates $\pi/4$
- `sample` generates one random point
  - Test membership in the unit circle
  - Returns 1 for inside, 0 for outside



```
# Estimate π (compute-intensive task).
# Pick random points in the unit square [(0,0)-(1,1)].
# See how many fall in the unit circle center=(0, 0), radius=1.
# The fraction should be π / 4.

import random
random.seed(314)

def sample(p):
    x, y = random.random(), random.random()
    in_unit_circle = 1 if x*x + y*y < 1 else 0
    return in_unit_circle

# "parallelize" method creates an RDD.
NUM_SAMPLES = int(1e6)
count = sc.parallelize(range(0, NUM_SAMPLES)) \
            .map(sample) \
            .reduce(lambda a, b: a + b)
approx_pi = 4.0 * count / NUM_SAMPLES
print("pi is roughly %f" % approx_pi)
```

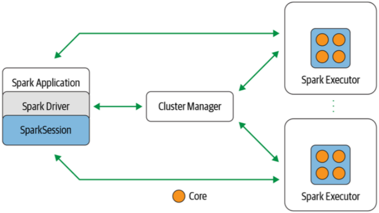
executed in 386ms, finished 04:27:53 2022-11-23

```
pi is roughly 3.141400
```

- `parallelize` distributes the sampling task
  - Each element in the RDD triggers one call to `sample`
  - "Embarrassingly parallel" computation
- `map` applies sampling across partitions
  - Each worker independently counts hits inside the circle
- `reduce` aggregates partial sums
  - Summing 0 and 1 values yields total count of hits

Spark: Architecture

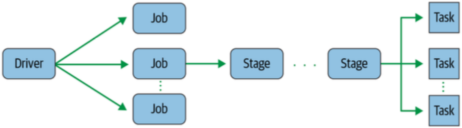
- Architecture
  - Who does what
  - I.e., responsibilities of each component
- Spark Application
  - Code describing computation
  - E.g., Python code calling Spark



- Spark Driver
  - Transform operations into DAG computations
  - Distribute task execution across *Executors*
  - Communicate with *Cluster Manager* for resources
- Spark Session
  - Interface to Spark system
- Cluster Manager
  - Manage and allocate resources
  - Support Hadoop, YARN, Mesos, Kubernetes
- Spark Executor
  - Run worker node to execute tasks
  - Typically one executor per node
  - Relies on JVM

Spark: Computation Model

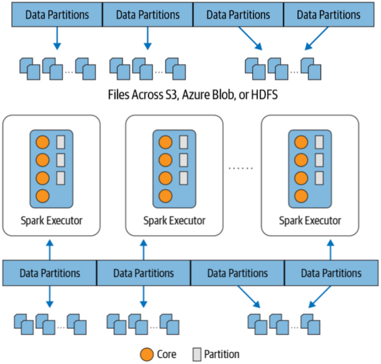
- Architecture
  - Who does what
- Computation model
  - How are things done



- Spark Driver
  - Converts *Application* into *Jobs*
  - Describes computation with *Transformations* and triggers with *Actions*
- Spark Job
  - Parallel computation in response to a *Action*
  - Each *Job* is a DAG with dependent *Stages*
- Spark Stage
  - Smaller operation within a *Job*
  - *Stages* run serially or in parallel
- Spark Task
  - Each *Stage* has multiple *Tasks*
  - Single unit of work sent to a *Executor*
  - Each *Task* maps to a single core and works on a single data partition

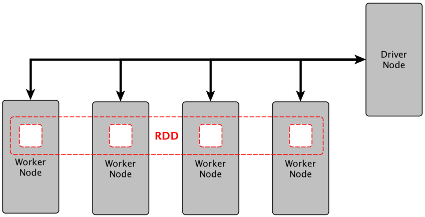
Distributed Data and Partitions

- **Data is distributed** as partitions across physical nodes
  - Store each partition in memory
  - Enable efficient parallelism
- **Spark Executors** process data “close” to them
  - Minimize network bandwidth
  - Ensure data locality
  - Similar to Hadoop



Parallelized Collections

- Parallelized collections created by calling *SparkContext* `parallelize()` on an existing collection



- Data spread across nodes
- Number of *partitions* to cut dataset into
  - Spark runs one *Task* per partition
  - Aim for 2-4 partitions per CPU
    - Spark sets partitions automatically based on your cluster
    - Set manually by passing as a second parameter to `parallelize()`

Deployment Modes

- Spark can run on several different configurations
    - Components (e.g., Driver, Cluster Manager, and Executors) split on different nodes

Deployment Mode	Where Components Run	Notes
Local	Run in a single JVM on one machine	Run Spark on a laptop
Standalone	Run in separate JVMs on different machines	Spark's built-in cluster manager
YARN / Kubernetes	Run in different pods/containers	Production clusters