

An Introduction to



Apostolos N. Papadopoulos

Assistant Professor
Data Engineering Lab,
Department of Informatics
Aristotle University of Thessaloniki
GREECE

Outline

What is Spark?

Basic features

Comparison with Hadoop MR

Examples

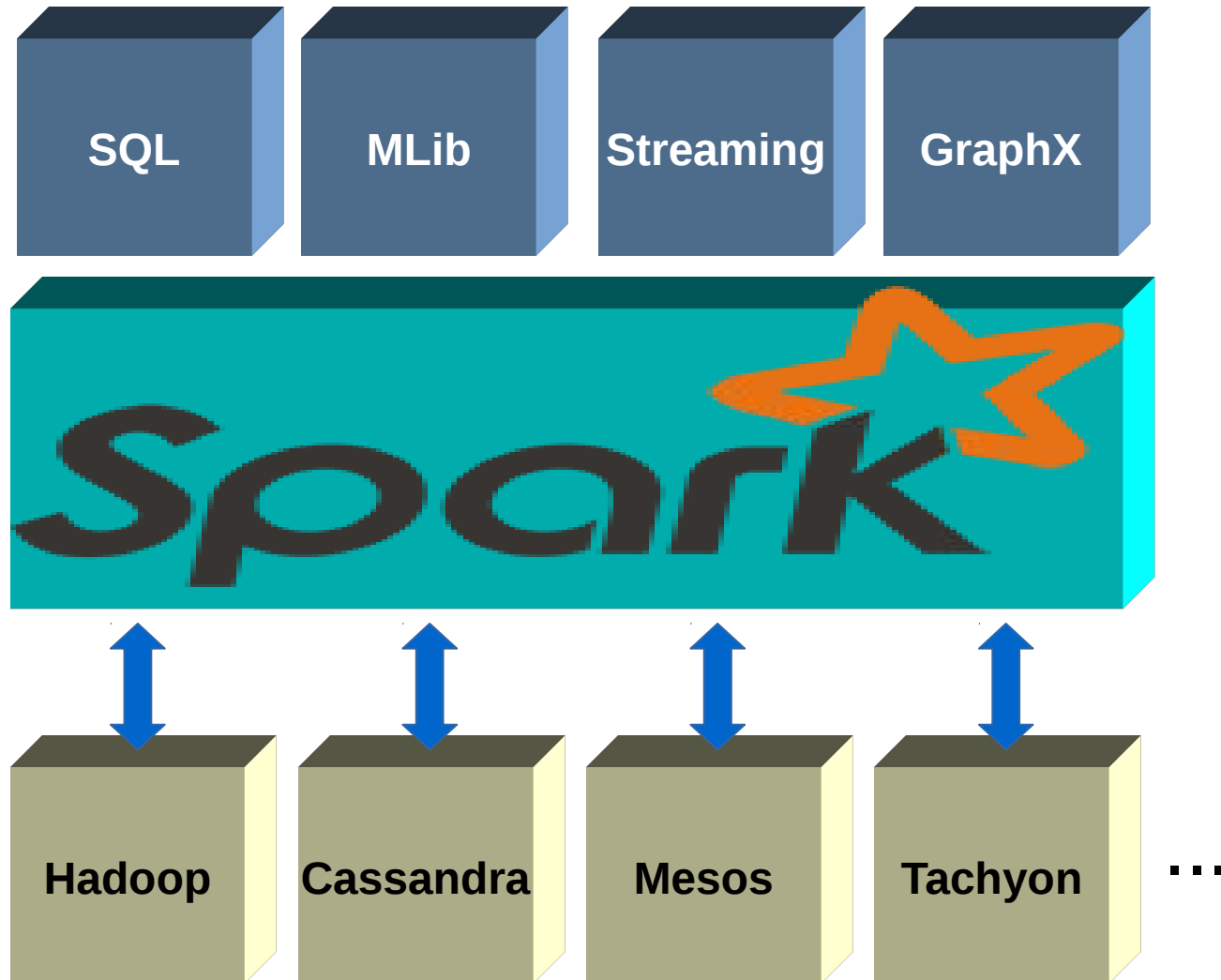
What is Spark ?

In brief, **Spark is a unified platform for cluster computing, enabling big data management and analytics**

It is an Apache Project and its current version is 1.2.1 (released in February 2015)

It is **one of the most active projects** at Apache

The Spark Ecosystem



Resilient Distributed Datasets (RDDs)

Data manipulation in Spark is heavily based on RDDs. An RDD is an interface composed of:

- a set of partitions
- a list of dependencies
- a function to compute a partition given its parents
- a partitioner (optional)
- a set of preferred locations per partition (optional)

Simply stated: **an RDD is a distributed collections of items**. In particular: an RDD is a read-only collection of items partitioned across a set of machines that can be rebuilt if a partition is destroyed.

Resilient Distributed Datasets (RDDs)

// define the spark context

```
val sc = new SparkContext(...)
```

// file is an RDD from HDFS

```
val fileRDD = sc.textFile("hdfs://...")
```

// sosRDD is another RDD based on fileRDD

```
val sosRDD = file.filter(_.contains("SOS"))
```

```
sosRDD.count() // this is an action
```

Resilient Distributed Datasets (RDDs)

There are **transformations** on RDDs that allow us to create new RDDs: `map`, `filter`, `groupByKey`, `reduceByKey`, `partitionBy`, `sortByKey`, `join`, etc

Also, there are **actions** applied in the RDDs: `reduce`, `collect`, `take`, `count`, `saveAsTextFile`, etc

Note: computation takes place only in actions and not on transformations!

RDDs and DAGs

A set of RDDs corresponds is transformed to a Directed Acyclic Graph (DAG)

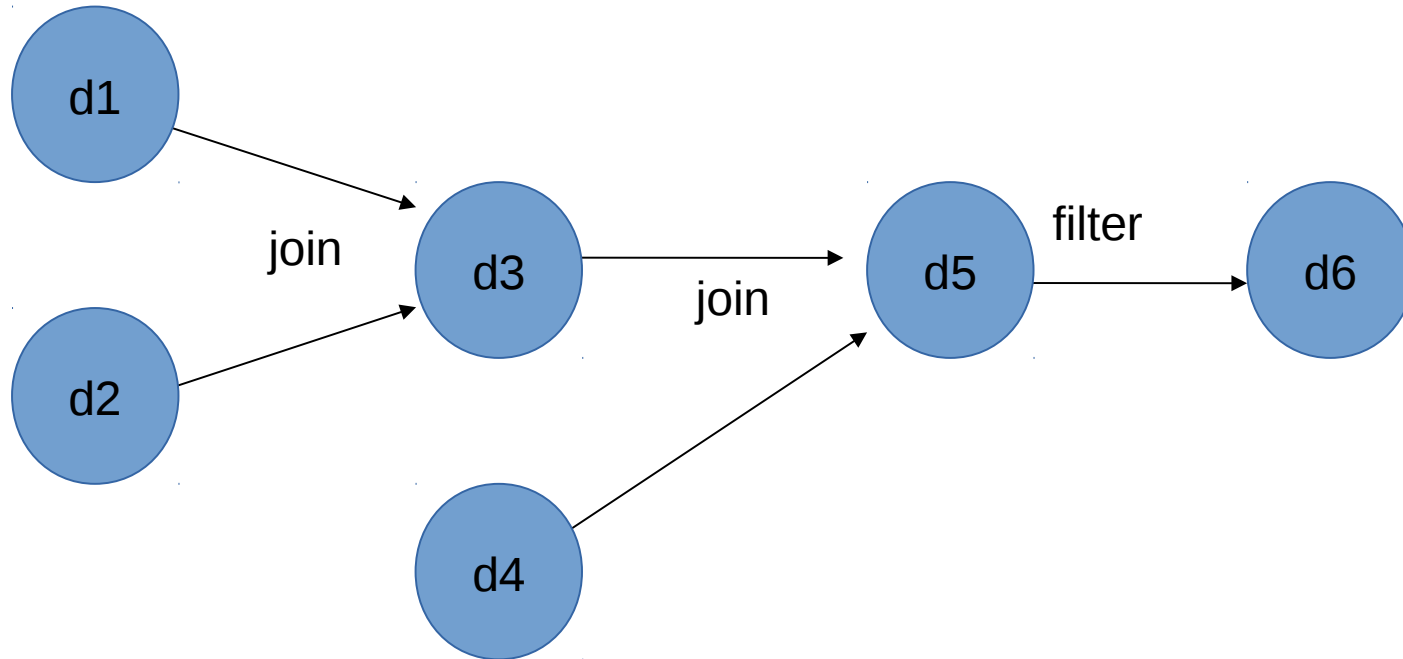
Input: RDD and partitions to compute

Output: output from actions on those partitions

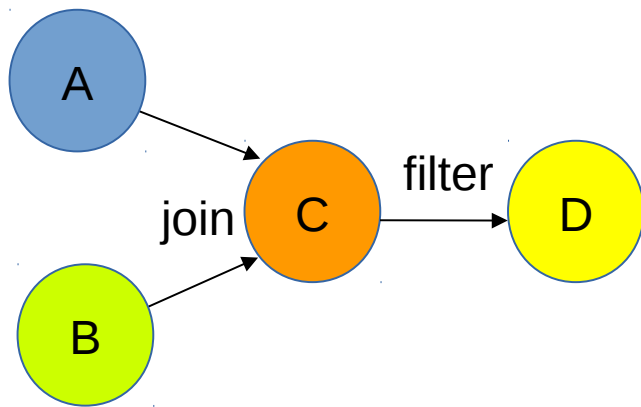
Roles:

- > Build stages of tasks
- > Submit them to lower level scheduler (e.g. YARN, Mesos, Standalone) as ready
- > Lower level scheduler will schedule data based on locality
- > Resubmit failed stages if outputs are lost

DAG Scheduling

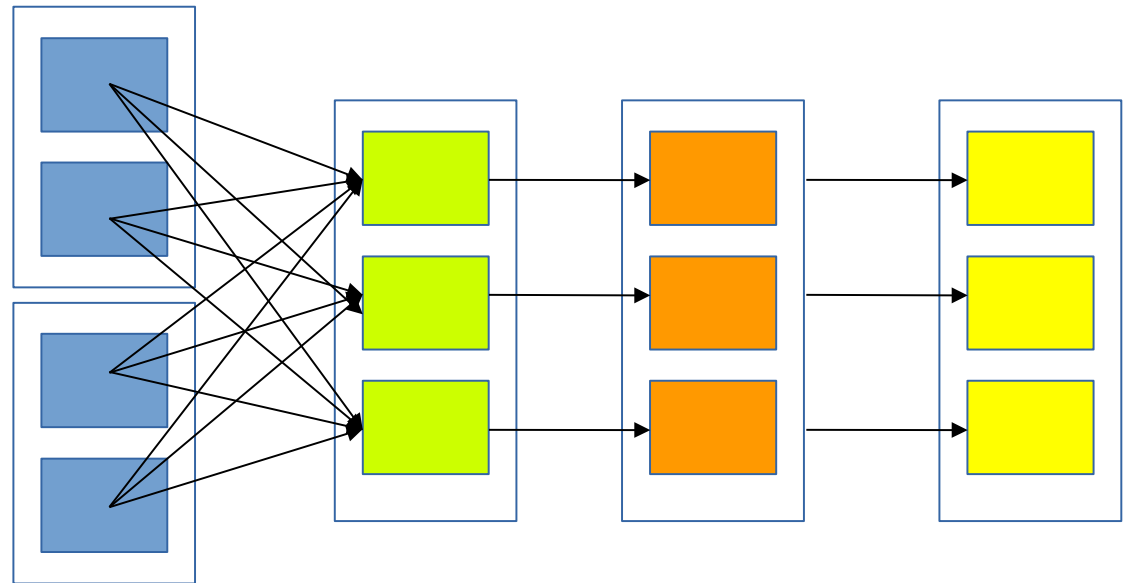


DAG Scheduling



RDD objects

```
A.join(B).filter(...).filter(...)
```



DAG scheduler

split graph into stages of tasks
submit each stage

Spark SQL

Spark SQL is a library for querying structured datasets as well as distributed datasets.

Spark SQL allows relational queries expressed in **SQL**, **HiveQL**, or **Scala** to be executed using Spark.

Example:

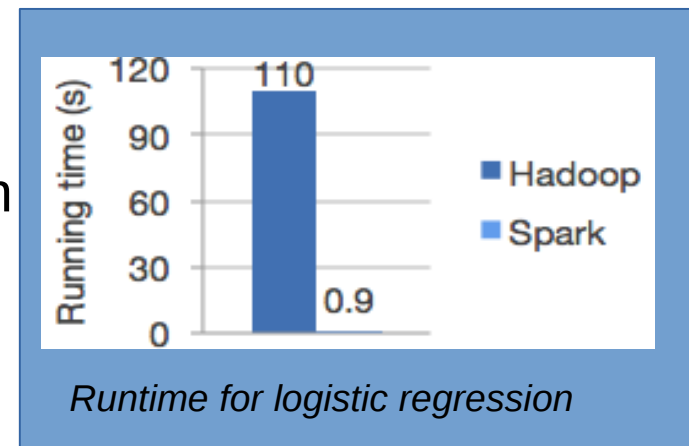
```
hc = HiveContext(sc)
rows = hc.sql("select id, name, salary from emp")
rows.filter(lambda r: r.salary > 2000).collect()
```

Spark MLlib

MLib is Spark's scalable machine learning library

Version 1.1 contains the following algorithms:

- linear SVM and logistic regression
- classification and regression tree
- k-means clustering
- recommendation via alternating least squares
- singular value decomposition
- linear regression with L1- and L2-regularization
- multinomial naive Bayes
- basic statistics
- feature transformations



Spark Streaming

Spark Streaming is a library to ease the development of complex streaming applications.

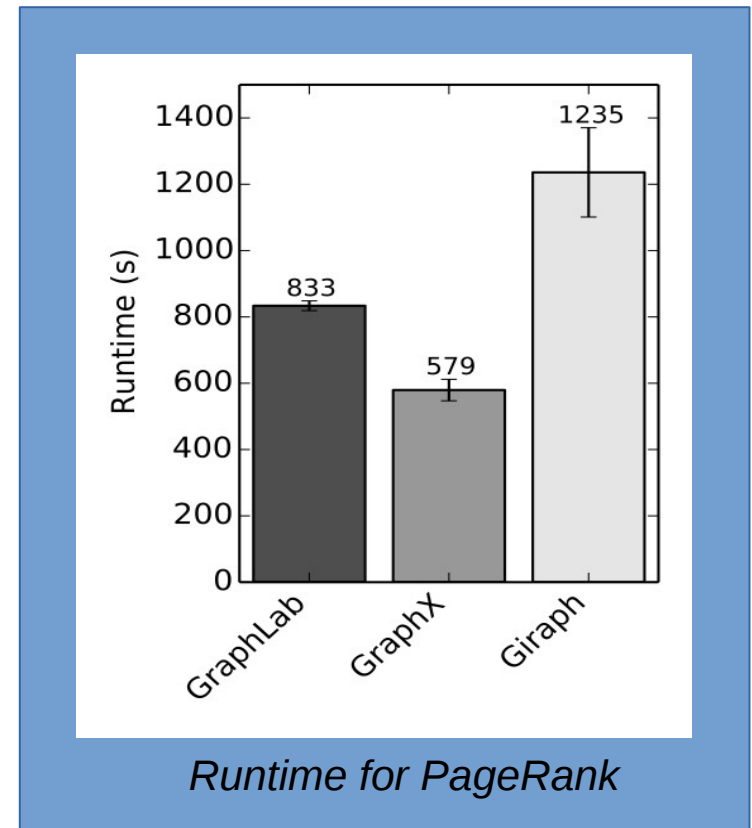
Data can be inserted into Spark from different sources like **Kafka**, **Flume**, **Twitter**, **ZeroMQ**, **Kinesis** or **TCP sockets** can be processed using complex algorithms expressed with high-level functions like **map**, **reduce**, **join** and **window**.

Spark GraphX

GraphX provides an API for graph processing and graph-parallel algorithms on-top of Spark.

The current version supports:

- **PageRank**
- **Connected components**
- Label propagation
- SVD++
- Strongly connected components
- **Triangle counting**



Spark vs Hadoop

Spark supports many different types of tasks including SQL queries, streaming applications, machine learning and graph operations.

On the other hand ...

Hadoop MR is good for heavy jobs that perform complex tasks in massive amounts of data. However, Spark can do better even in this case due to better memory utilization and optimization alternatives.

Spark vs Hadoop: sorting 1PB

	Hadoop	Spark 100TB	Spark 1PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min

Spark Examples

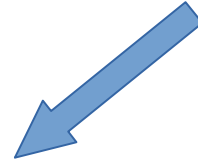
Spark supports

- ✓ Java
- ✓ Python
- ✓ **Scala**

We are going to use the Scala API in this lecture.

Hello Spark

things we
must
import



```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object HelloSpark {
    def main(args: Array[String]): Unit = {
        println("Hello, Spark!")
    }
}
```

LineCount

```
object LineCount {  
  def main(args: Array[String]) {  
    println("Hi, this is the LineCount application for Spark.")  
  
    // Create spark configuration and spark context  
    val conf = new SparkConf().setAppName("LineCount App")  
    val sc = new SparkContext(conf)  
  
    val currentDir = System.getProperty("user.dir") // get the current directory  
    val inputFile = "file://" + currentDir + "/leonardo.txt"  
    val outputDir = "file://" + currentDir + "/output"  
  
    val myData = sc.textFile(inputFile, 2).cache()  
    val num1 = myData.filter(line => line.contains("the")).count()  
    val num2 = myData.filter(line => line.contains("and")).count()  
    val totalLines = myData.map(line => 1).count  
    println("Total lines: %s, lines with \"the\": %s, lines with \"and\":  
%s".format(totalLines, num1, num2))  
  
    sc.stop()  
  }  
}
```

WordCount

```
import org.apache.spark.SparkContext._
import org.apache.spark.{SparkConf, SparkContext}

object WordCount {

  def main(args: Array[String]): Unit = {

    val sparkConf = new SparkConf().setMaster("local[2]").setAppName("WordCount") // config
    val sc = new SparkContext(sparkConf) // create spark context

    val currentDir = System.getProperty("user.dir") // get the current directory
    val inputFile = "file://" + currentDir + "/leonardo.txt"
    val outputDir = "file://" + currentDir + "/output"
    val txtFile = sc.textFile(inputFile)

    txtFile.flatMap(line => line.split(" ")) // split each line based on spaces
      .map(word => (word,1)) // map each word into a word,1 pair
      .reduceByKey(_+_) // reduce
      .saveAsTextFile(outputDir) // save the output

    sc.stop()
  }
}
```

WordCount in Hadoop

```
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.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

    public static class Map extends 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, Context context) throws
        IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text,
    IntWritable> {

        public void reduce(Text key, Iterable<IntWritable> values, Context
        context)
            throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
}
```

```
public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);

    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);

    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));

    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);

}
```

PageRank

```
object PageRank {

  def main(args: Array[String]) {
    val iters = 10 // number of iterations for pagerank computation
    val currentDir = System.getProperty("user.dir") // get the current directory
    val inputFile = "file://" + currentDir + "/webgraph.txt"
    val outputDir = "file://" + currentDir + "/output"

    val sparkConf = new SparkConf().setAppName("PageRank")
    val sc = new SparkContext(sparkConf)
    val lines = sc.textFile(inputFile, 1)

    val links = lines.map { s => val parts = s.split("\\s+")(parts(0), parts(1))}.distinct().groupByKey().cache()
    var ranks = links.mapValues(v => 1.0)
    for (i <- 1 to iters) {
      println("Iteration: " + i)
      val contribs = links.join(ranks).values.flatMap{ case (urls, rank) => val size = urls.size urls.map(url =>
        (url, rank / size)) }

      ranks = contribs.reduceByKey(_ + _).mapValues(0.15 + 0.85 * _)
    }

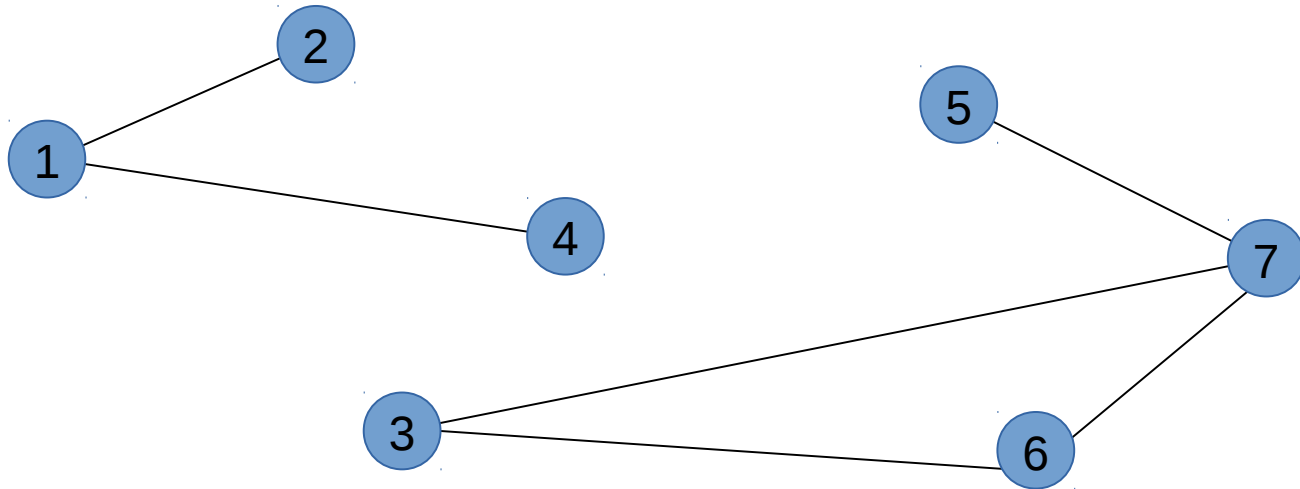
    val output = ranks.collect()
    output.foreach(tup => println(tup._1 + " has rank: " + tup._2 + "."))

    sc.stop()
  }
}
```

PageRank with GraphX

```
object PageRank {  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setAppName("PageRank App")  
    val sc = new SparkContext(conf)  
    val currentDir = System.getProperty("user.dir")  
    val edgeFile = "file://" + currentDir + "/followers.txt"  
  
    val graph = GraphLoader.edgeListFile(sc, edgeFile)  
  
    // run pagerank  
    val ranks = graph.pageRank(0.0001).vertices  
  
    println(ranks.collect().mkString("\n")) // print result  
  }  
}
```

Connected Components



This graph has two connected components:

$cc1 = \{1, 2, 4\}$

$cc2 = \{3, 5, 6, 7\}$

Output:

(1,1) (2,1) (4,1)

(3,3) (5,3) (6,3) (7,3)

Connected Components

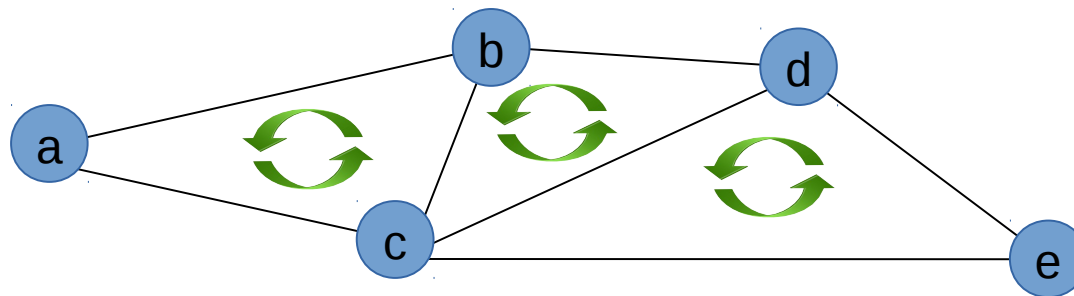
```
object ConnectedComponents {  
  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setAppName("ConnectedComponents App")  
    val sc = new SparkContext(conf)  
  
    val currentDir = System.getProperty("user.dir")  
    val edgeFile = "file://" + currentDir + "/graph.txt"  
    val graph = GraphLoader.edgeListFile(sc, edgeFile)  
  
    // find the connected components  
    val cc = graph.connectedComponents().vertices  
  
    println(cc.collect().mkString("\n")) // print the result  
  }  
}
```

Counting Triangles

Triangles are very important in Network Analysis:

- dense subgraph mining (communities, trusses)
- triangular connectivity
- network measurements (e.g. clustering coefficient)

Example



Counting Triangles

```
object TriangleCounting {  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setAppName("TriangleCounting App")  
    val sc = new SparkContext(conf)  
  
    val currentDir = System.getProperty("user.dir")  
    val edgeFile = "file://" + currentDir + "/enron.txt"  
  
    val graph = GraphLoader  
      .edgeListFile(sc, edgeFile, true)  
      .partitionBy(PartitionStrategy.RandomVertexCut)  
  
    // Find number of triangles for each vertex  
    val triCounts = graph.triangleCount().vertices  
  
    println(triCounts.collect().mkString("\n"))  
  }  
}
```

Some Spark Users



Resources

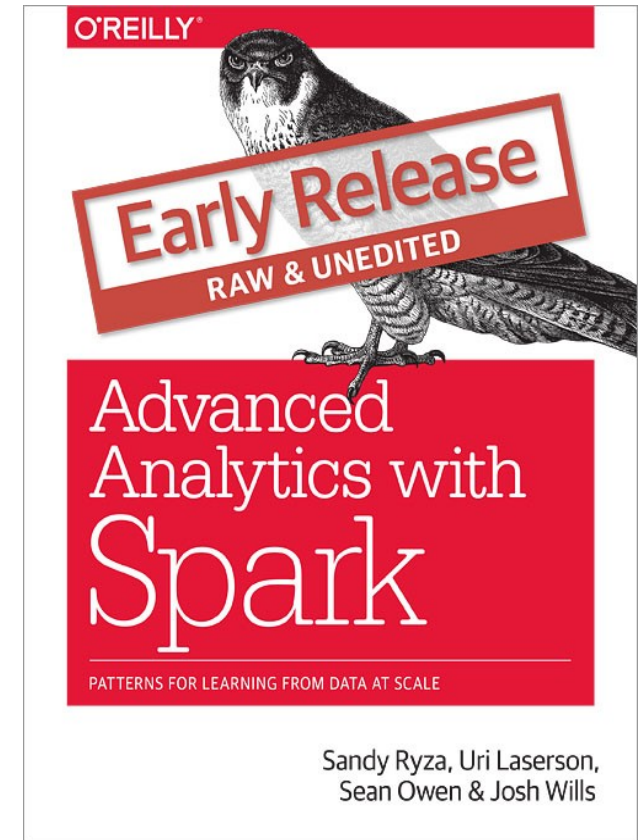
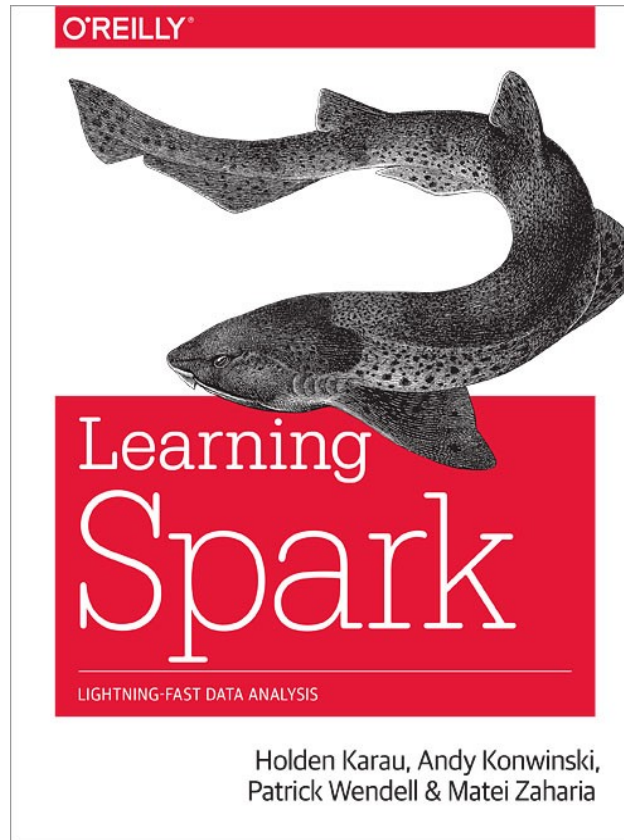
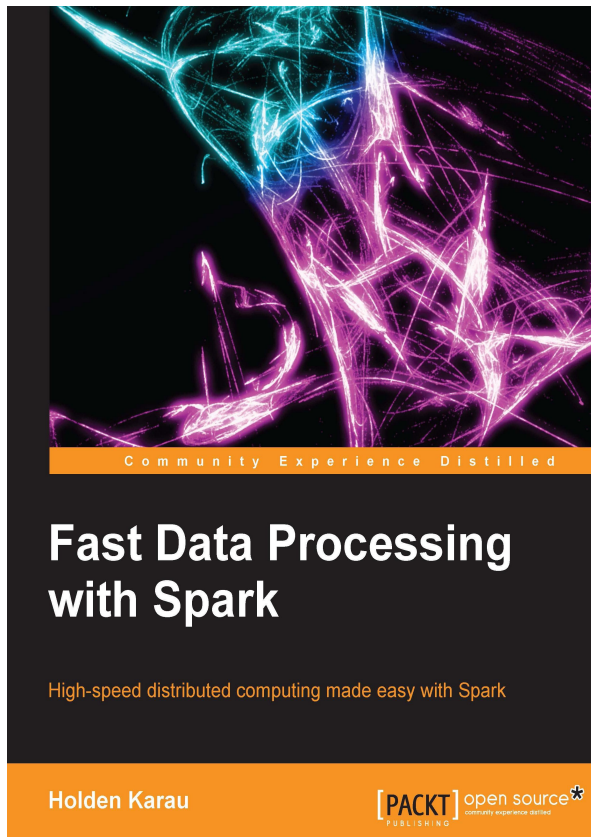
The best way to begin learning Spark is to study the material in the project's website

<https://spark.apache.org>

From this website you have access to **Spark Summit 2014** which contains useful video lectures for all Spark components

Resources

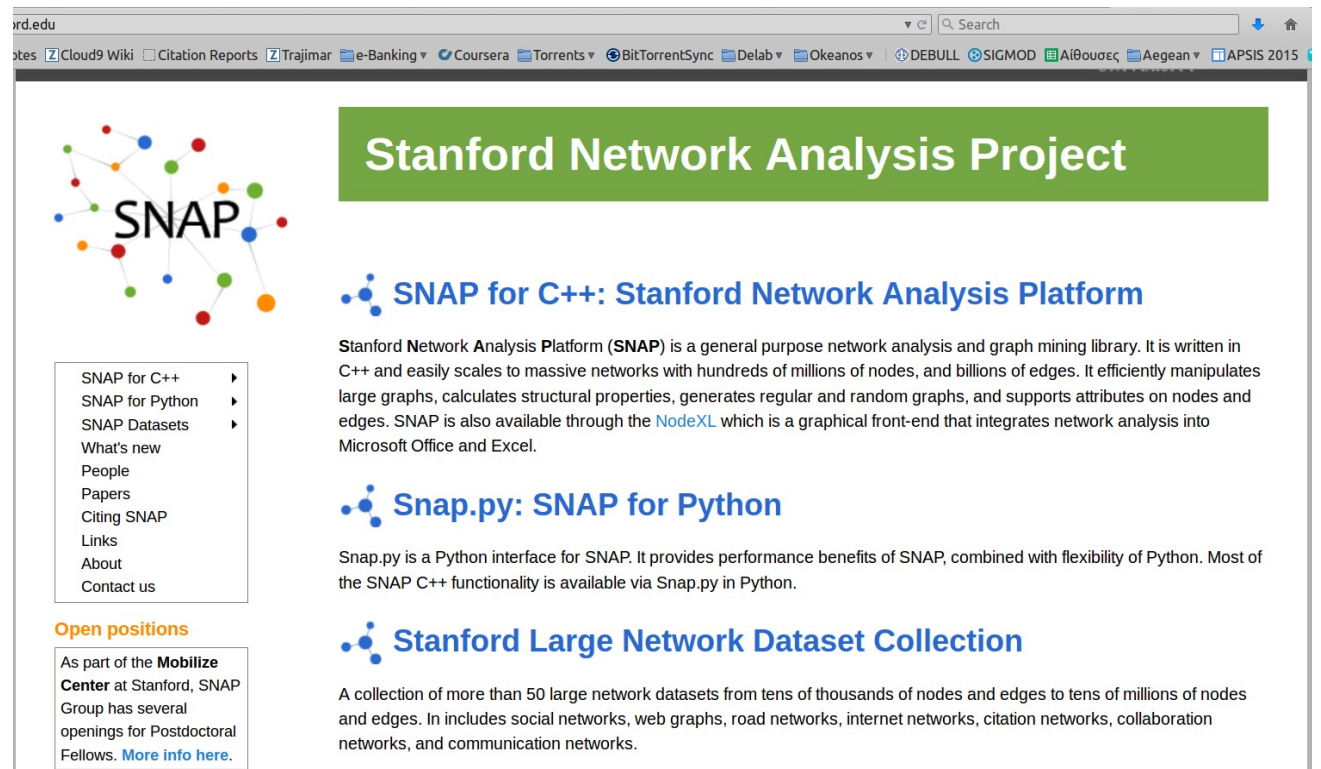
Books to learn Spark



Resources


Where to find more graph data ?

Take a look at
<http://snap.stanford.edu>



rd.edu

otes Cloud9 Wiki Citation Reports Trajimar e-Banking Coursera Torrents BitTorrentSync Delab Okeanos DEBULL SIGMOD ΑΙΘΟΥΣΑ Aegean APSIS 2015



Stanford Network Analysis Project

SNAP for C++: Stanford Network Analysis Platform

Stanford Network Analysis Platform (SNAP) is a general purpose network analysis and graph mining library. It is written in C++ and easily scales to massive networks with hundreds of millions of nodes, and billions of edges. It efficiently manipulates large graphs, calculates structural properties, generates regular and random graphs, and supports attributes on nodes and edges. SNAP is also available through the [NodeXL](#) which is a graphical front-end that integrates network analysis into Microsoft Office and Excel.

Snap.py: SNAP for Python

Snap.py is a Python interface for SNAP. It provides performance benefits of SNAP, combined with flexibility of Python. Most of the SNAP C++ functionality is available via Snap.py in Python.

Stanford Large Network Dataset Collection

A collection of more than 50 large network datasets from tens of thousands of nodes and edges to tens of millions of nodes and edges. It includes social networks, web graphs, road networks, internet networks, citation networks, collaboration networks, and communication networks.

SNAP for C++
SNAP for Python
SNAP Datasets
What's new
People
Papers
Citing SNAP
Links
About
Contact us

Open positions

As part of the **Mobilize Center** at Stanford, SNAP Group has several openings for Postdoctoral Fellows. [More info here.](#)

Thank you

Questions ?