BIG DATA

Scala Programming Language

K V Subramaniam
Computer Science and Engineering

BIG DATA

Hadoop Challenges

- In last lectures
  - Algorithms for relational / page rank
  - Require use of multiple files
  - Requires use of iteration
- No support from language
- Is there a language suitable for Big Data

BIG DATA

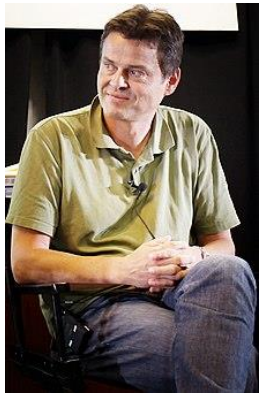
Overview of lecture

- Scala overview
- Scala Notation
- Features required for Big Data
- Functional programming
- Big Data and Scala

Scala Overview

What is Scala?

- JVM based language that can be called and call Java – SCAlable LAnguage
- A more concise, richer Java + Functional Programming
  - Blends OO + FP
- Strongly statically typed
  - But feels dynamically typed
  - Type inferencing saves typing
- Developed by Marvin Odersky at EPFL (Switzerland)
- Released in 2004



Roots in Java – Why move?

What's wrong with Java?

- Not designed for highly concurrent programs
  - The original Thread model was just *wrong* (it's been fixed)
  - Java 5+ helps by including java.util.concurrent
- Verbose
  - Too much of Thing thing = new Thing();
  - Too much "boilerplate," for example, getters and setters

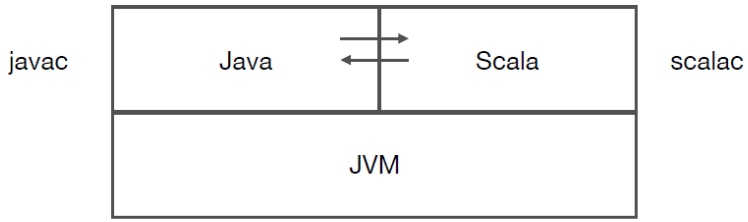
What's right with Java?

- Very popular
- Object oriented (mostly), which is important for large projects
- Strong typing (more on this later)
- The fine large library of classes
- **The JVM!** Platform independent, highly optimized

BIG DATA

Java and Scala: Spot the differences

- Java is a *good* language, and Scala is a lot like it
- For each difference, there is a *reason*--none of the changes are “just to be different”
- Scala and Java are (almost) completely interoperable
  - Call Java from Scala? No problem!
  - Call Scala from Java? Some restrictions, but mostly OK.
  - Scala compiles to .class files (a *lot* of them!), and can be run with either the scala command or the java command



Quick Tour of Scala – Differences with Java

BIG DATA

Java and Scala: Spot the differences

Declaring variables:

```
var x: Int = 7
var x = 7 // type inferred
val y = “hi” // read-only
```

Java equivalent:

```
int x = 7;

final String y = “hi”;
```

Functions:

```
def square(x: Int): Int = x*x
def square(x: Int): Int = {
  x*x
}
def announce(text: String) =
{
  println(text)
}
```

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

BIG DATA

Java and Scala: Spot the differences

- Minimal Verbosity
- Referential Transparency – type inferencing
- Concurrency
- Functional Programming

Minimal Verbosity

- Java:
  - ```
    class Person {
        private String firstName;
        private String lastName;
        private int    age;

        public Person(String firstName, String lastName, int age) {
            this.firstName = firstName;
            this.lastName  = lastName;
            this.age  = age;
        }

        public void setFirstName(String firstName) { this.firstName = firstName; }
        public void String getFirstName() { return this.firstName; }
        public void setLastName(String lastName) { this.lastName = lastName; }
        public void String getLastName() { return this.lastName; }
        public void setAge(int age) { this.age = age; }
        public void int getAge() { return this.age; }
    }
    ```
- Scala:
  - ```
    class Person(var firstName: String, var lastName: String, var age: Int)
    ```
- Source: http://blog.objectmentor.com/articles/2008/08/03/the-seductions-of-scala-part-i

Type Inferencing

- Java is statically typed--a variable has a type, and can hold only values of that type
  - You must specify the type of every variable
  - Type errors are caught by the compiler, not at runtime--this is a big win
  - However, it leads to a lot of typing (pun intended)
- Languages like Ruby and Python don’t make you declare types
  - Easier (and more fun) to write programs
  - Less fun to debug, especially if you have even slightly complicated types

Type Inferencing

- Scala is also statically typed, but it uses type inferencing--that is, it figures out the types, so you don’t have to
  - **The good news**: Less typing, more fun, type errors caught by the compiler
  - **The bad news**: More kinds of error messages to get familiar with

Consistency

- In Java, every value is an object--unless it's a primitive
  - Numbers and booleans are primitives for reasons of efficiency, so we have to treat them differently (you can't "talk" to a primitive)
  - In Scala, all values are objects. Period.
  - The compiler turns them into primitives, so no efficiency is lost (behind the scenes, there are objects like RichInt)

- Java has operators (+, <, ...) and methods, with different syntax
  - In Scala, operators are just methods, and in many cases you can use either syntax

Concurrency and Functional Programming

Concurrency

- Broadly speaking, concurrency can be either:
  - Fine-grained: Frequent interactions between threads working closely together (extremely challenging to get right)
  - Coarse-grained: Infrequent interactions between largely independent sequential processes (much easier to get right)
- Java 5 and 6 provide reasonable support for traditional fine-grained concurrency
  - Threads
- Scala has total access to the Java API
  - Hence, it can do anything Java can do
  - And it can do much more (see next slide)
- Scala also has Actors for coarse-grained concurrency
  - Sending messages (use the send ! Abstraction)

BIG DATA

Functional Programming

- The big nasty problem with concurrency is dealing with shared state--multiple threads all trying to read and maybe change the same variables
- If all data were immutable, then any thread could read it any time, no synchronization, no locks, no problem
- But if your program couldn’t ever change anything, then it couldn’t ever do anything, right?
- Wrong!
- There is an entire class of programming languages that use only immutable data, but work just fine: the functional languages

Concurrency

- The best-known functional languages are ML, OCaml, and Haskell
- Functional languages are regarded as:
  - "Ivory tower languages," used only by academics (mostly but not entirely true)
  - Difficult to learn (mostly true)
  - The solution to all concurrent programming problems everywhere (exaggerated, but not entirely wrong)
- Scala is an "impure" functional language--you can program functionally, but it isn't forced upon you

BIG DATA

Functional Programming

- Immutable – functional operations create new structures
  - Don’t modify existing structures
- Program implicitly captures data flow
- Order of operations not significant
- Functions
  - Are objects
  - can be passed as arguments to functions
  - can return functions
  - Can operate on collections

Functional Programming – differences with java

Quicksort program in Scala – java style

Observe

Focus on HOW?

Explicitly iterate

Determine when and how to swap()

```
def sort(xs: Array[Int]) {
  def swap(i: Int, j: Int) {
    val t = xs(i); xs(i) = xs(j); xs(j) = t
  }
  def sort1(l: Int, r: Int) {
    val pivot = xs((l + r) / 2)
    var i = l; var j = r
    while (i <= j) {
      while (xs(i) < pivot) i += 1
      while (xs(j) > pivot) j -= 1
      if (i <= j) {
        swap(i, j)
        i += 1
        j -= 1
      }
    }
    if (l < j) sort1(l, j)
    if (j < r) sort1(i, r)
  }
  sort1(0, xs.length - 1)
}
```

BIG DATA

Functional Programming – differences with java

Focus on solving the problem

Observe

Focus on WHAT, not HOW

```
def sort(xs: Array[Int]): Array[Int] = {
  if (xs.length <= 1) xs
  else {
    val pivot = xs(xs.length / 2)
    Array.concat(
      sort(xs filter (pivot >)),
           xs filter (pivot ==),
      sort(xs filter (pivot <)))
  }
}
```

Pick a pivot

Sort values smaller than the pivot

Sort values larger than the pivot

Concatenate the result

BIG DATA

Self Learning Exercise

- Does this sort array in ascending order or descending order?
- Consider the program with array
  - xs=3,1,2,0,7,6,4,5
- Write a program to sort in the reverse order (if ascending, sort descending)
- How can we parallelize this?

```
def sort(xs: Array[Int]): Array[Int] = {
  if (xs.length <= 1) xs
  else {
    val pivot = xs(xs.length / 2)
    Array.concat(
      sort(xs filter (pivot >)),
           xs filter (pivot ==),
      sort(xs filter (pivot <)))
  }
}
```

Functional programming and collections

Original List is left unchanged

```
val list = List(1, 2, 3)
list.foreach(x => println(x)) // prints 1, 2, 3
list.foreach(println)         // same

list.map(x => x + 2)    // returns a new List(3, 4, 5)
list.map(_ + 2)         // same

list.filter(x => x % 2 == 1)// returns a new List(1, 3)
list.filter(_ % 2 == 1)     // same

list.reduce((x, y) => x + y) // => 6
list.reduce(_ + _)           // same
```

Functional Programming and Big Data

BIG DATA

- Big data architectures leverage
  - Parallel disk, memory and CPU in clusters
- Operations consist of independently parallel operations
  - Similar to *map()* operator in a functional language
- Parallel operations have to be consolidated
  - Similar to *aggregation()* operators in functional languages
- Hence the fit

THANK YOU

K V Subramaniam, Usha Devi

Dept. of Computer Science and Engineering

subramaniamkv@pes.edu

ushadevibg@pes.edu

BIG DATA

In memory analytics with Spark : Introduction

K V Subramaniam
Computer Science and Engineering

BIG DATA

Overview of lecture – Spark Introduction

- Why Spark – the motivation?
- Moving to in memory compute
- Distribute data in memory
- Handling fault tolerance
- Programming model – Operations in Spark
- Handling key-value based operations
- Putting it all together : Word count in Spark

Motivation for Spark

BIG DATA

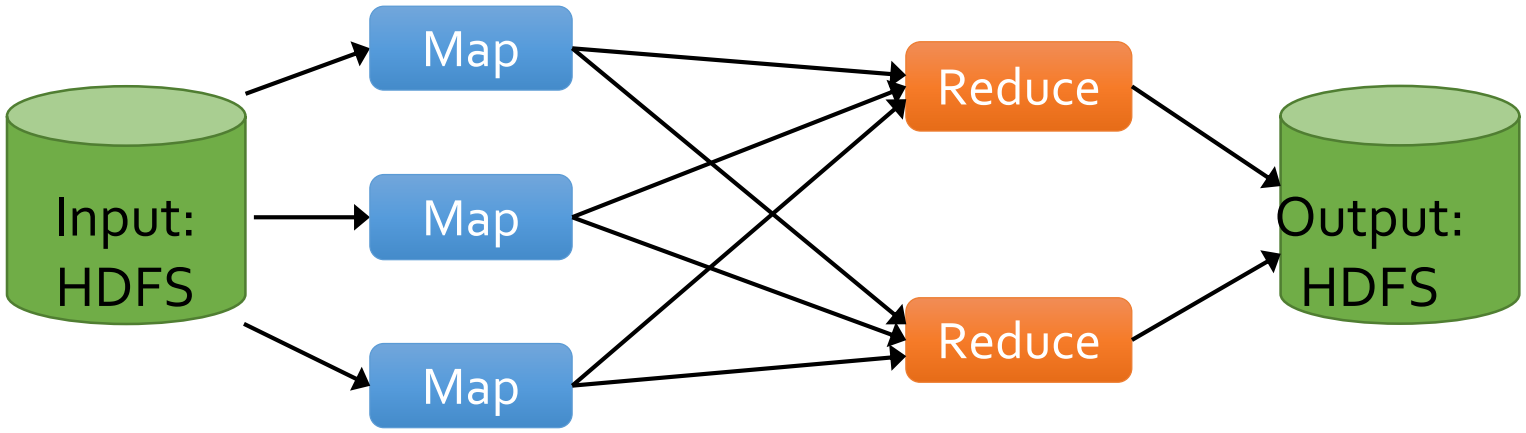
Spark: Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage

Hadoop → reads data from persistent storage in *Map* step

Writes data back to persistent store (HDFS) in reduce

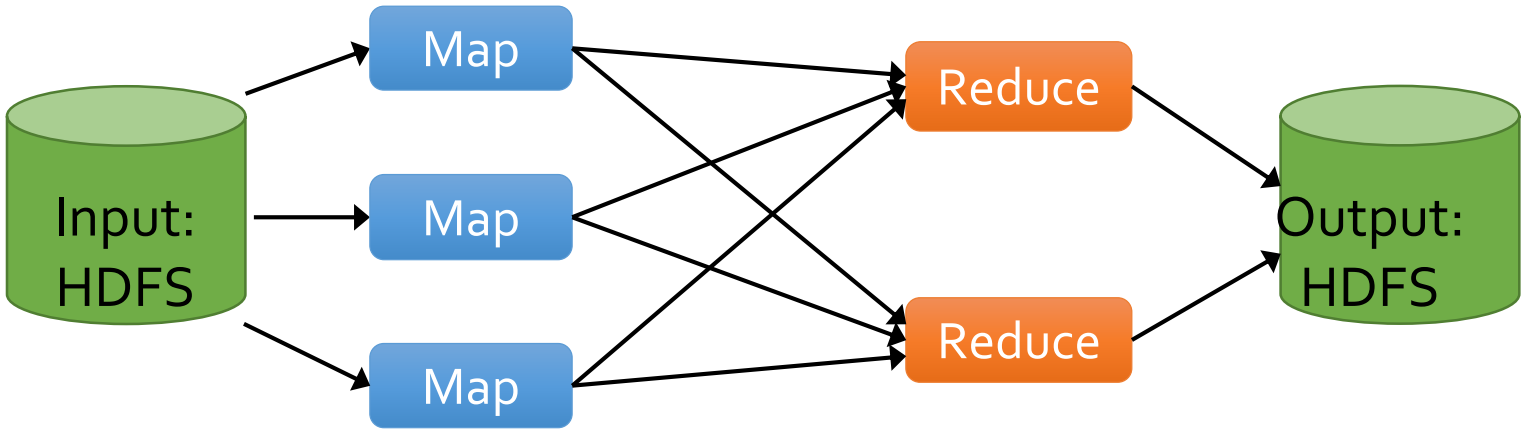
Advantage – dynamically decide machines/handle failures



Issues?

Consider the page rank exercise we did in the last unit?

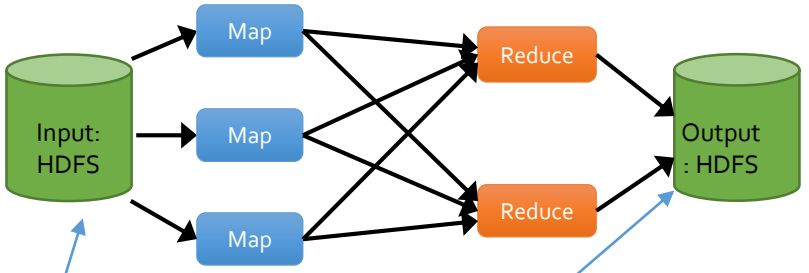
What are the issues that we see with this?



BIG DATA

Challenges

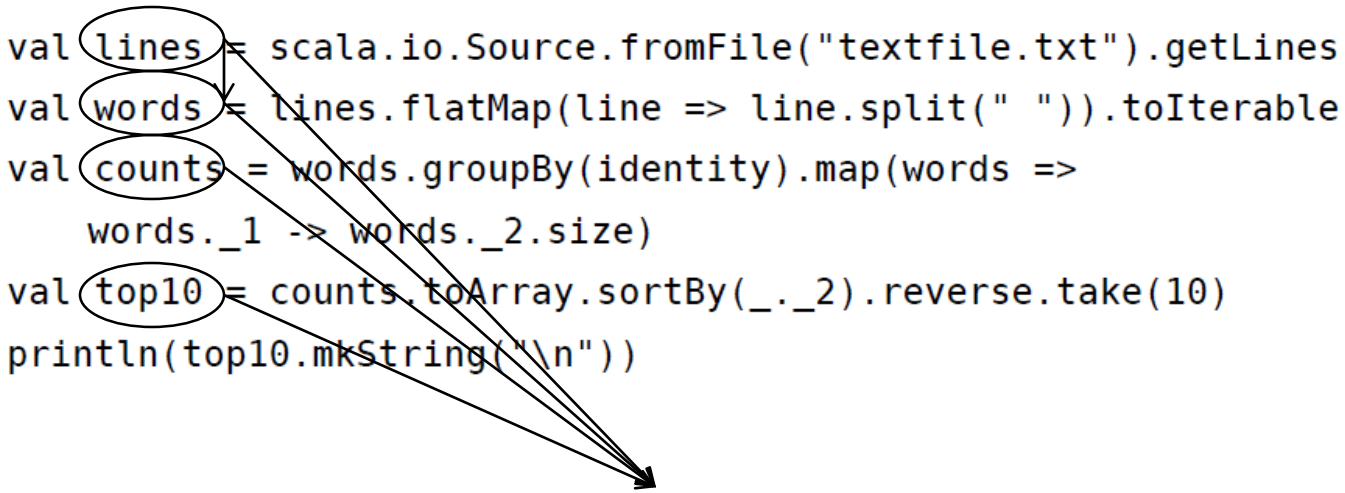
- Acyclic data flow
  - consider operating on a *data working set*
  - **Working set:** same set of data reused
  - in page rank, we keep computing importance vector and reusing in next iteration.
  - Hadoop – inefficient in such cases.
- Example of use
  - Where we need to *iterate*
    - Graph processing
    - Machine Learning
  - Where we need to do *interactive analysis*
    - Python, R



- On every iteration, storing/reloading of data from persistent storage is time consuming.

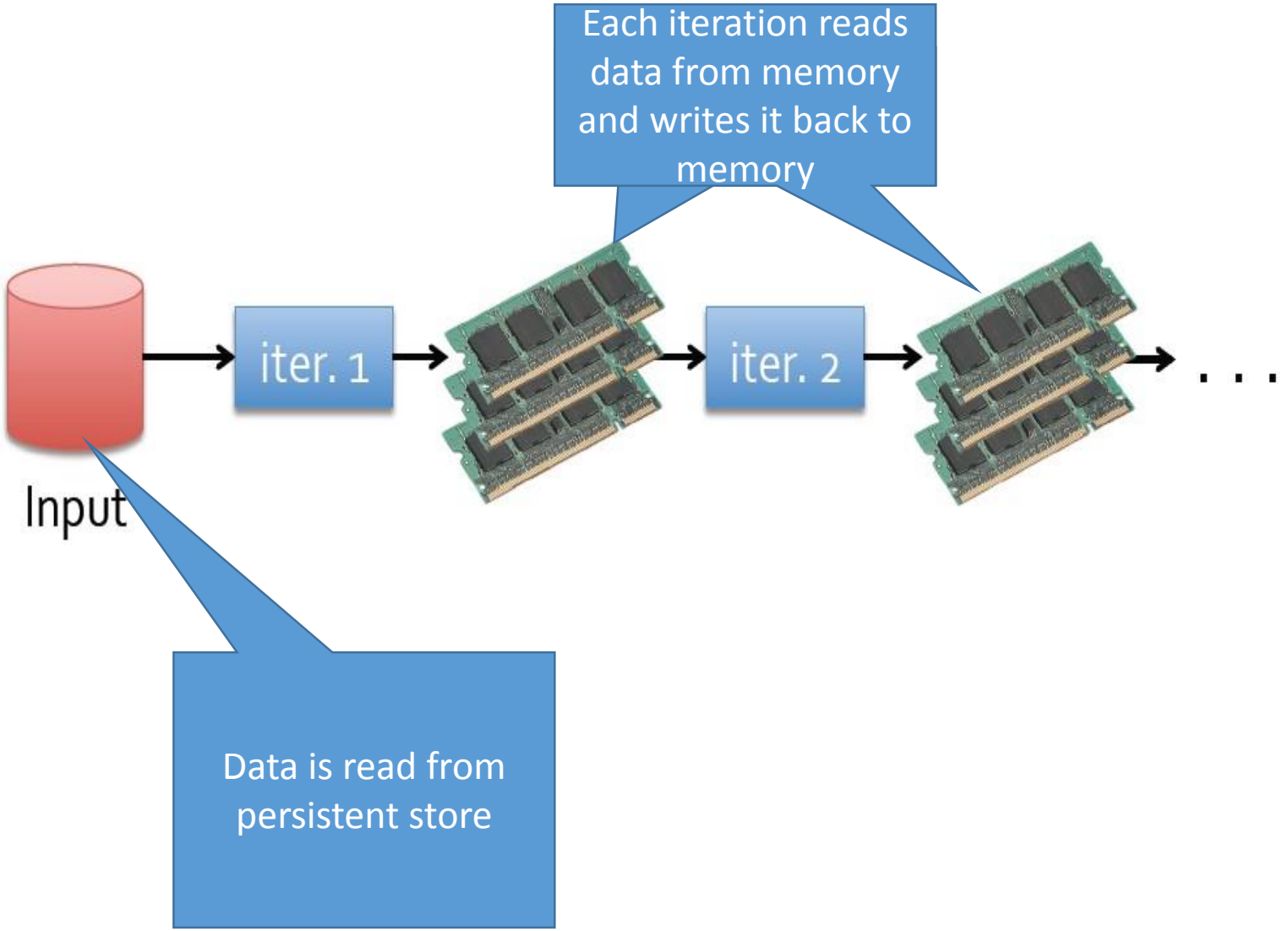
In Memory Computation

Recap: Word count in scala



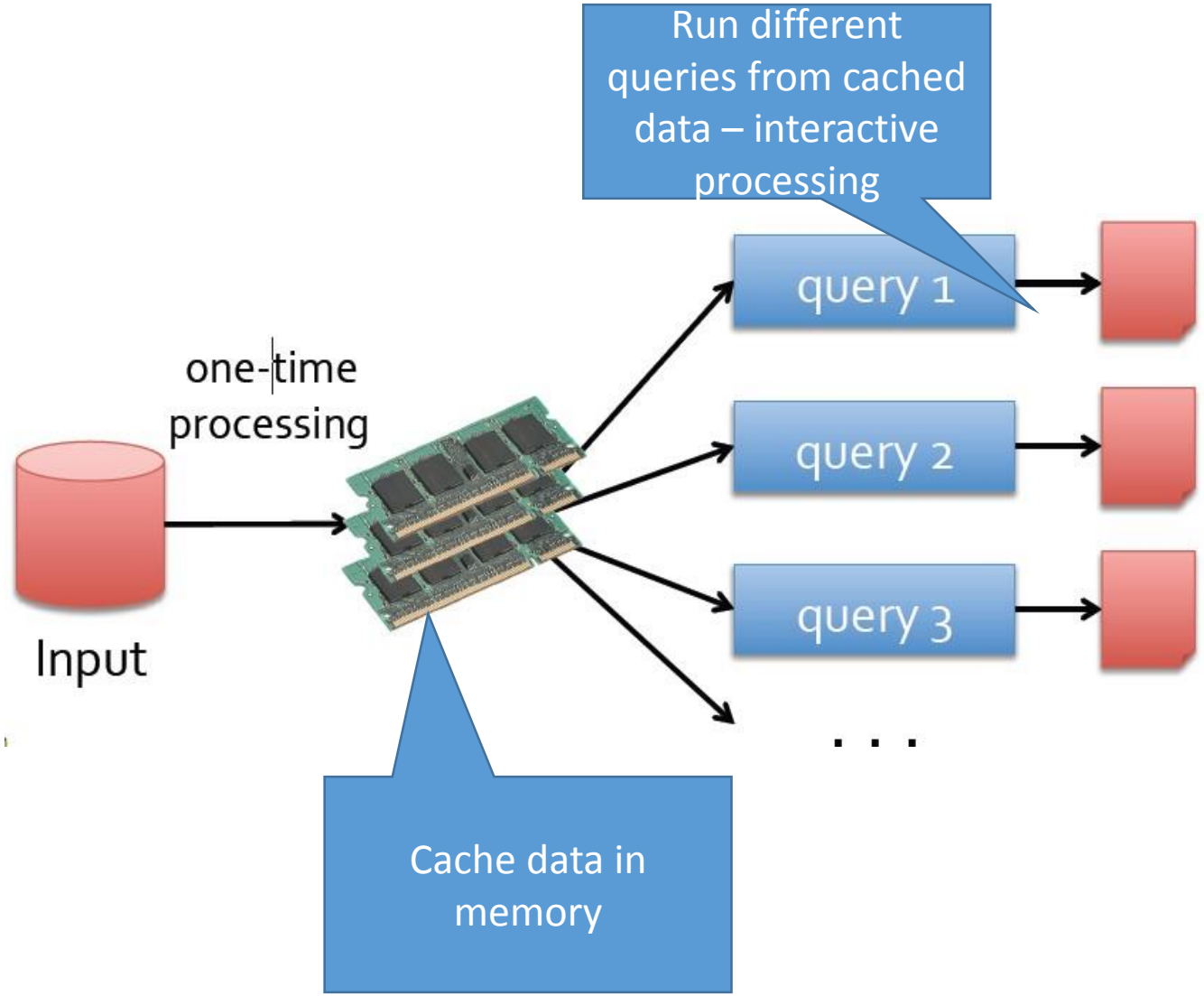
Each operation creates
A data value that can be kept in
Memory and reused.

Doing in memory processing – Iterative processing



BIG DATA

Doing in memory processing – interactive processing



Challenges of in memory processing

- How do we distribute the data among the DRAM of the cluster?
- What happens if this memory is not sufficient?
- How do we handle failures because memory is volatile?

Distributed Dataset

BIG DATA

Example: Log Processing

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(startsWithERROR())
messages = errors.map(split(“\t”),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
. . .
```

Courtesy: Zaharia et al , “Spark: Fast, Interactive, Language-Integrated Cluster Computing”, www.spark-project.org

BIG DATA

Example: Log Processing

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(startsWithERROR())
messages = errors.map(split(“\t”),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
. . .
```

Loads a file to an in memory struct called an RDD (think of it as a collection of strings)

Filter function to retain only those lines with an error. Creates another RDD

Applies a function to each element(string) in RDD and produces a new RDD

Keep it in memory as it will be reused

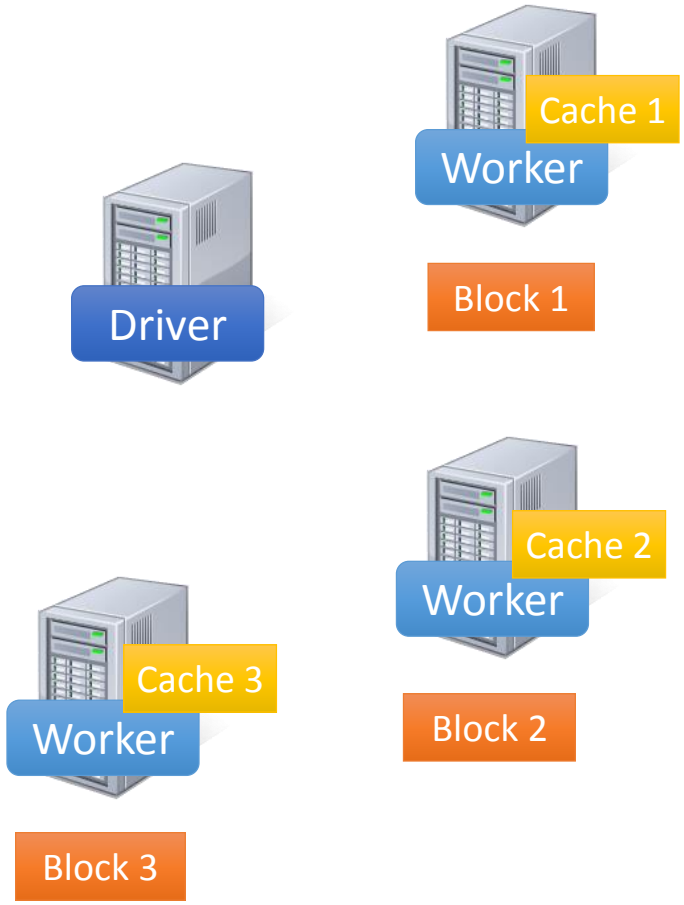
Counts #objects in the RDD

Courtesy: Zaharia et al , “Spark: Fast, Interactive, Language-Integrated Cluster Computing”, www.spark-project.org

BIG DATA

Java and Scala: Spot the differences

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(startsWithERROR())
messages = errors.map(split("\t"),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
```



Adding fault tolerance – The RDD

BIG DATA

Handling fault tolerance

Consider the following code:

```
                    Step1                    Step2
messages = textFile(...).filter(startsWithERROR())
                                .map(split("\t")(2))
                                          Step3
```

Step1: Read in the file to an in memory RDD

Step2: remove all lines that don't contain the term ERROR

Step3: split the line

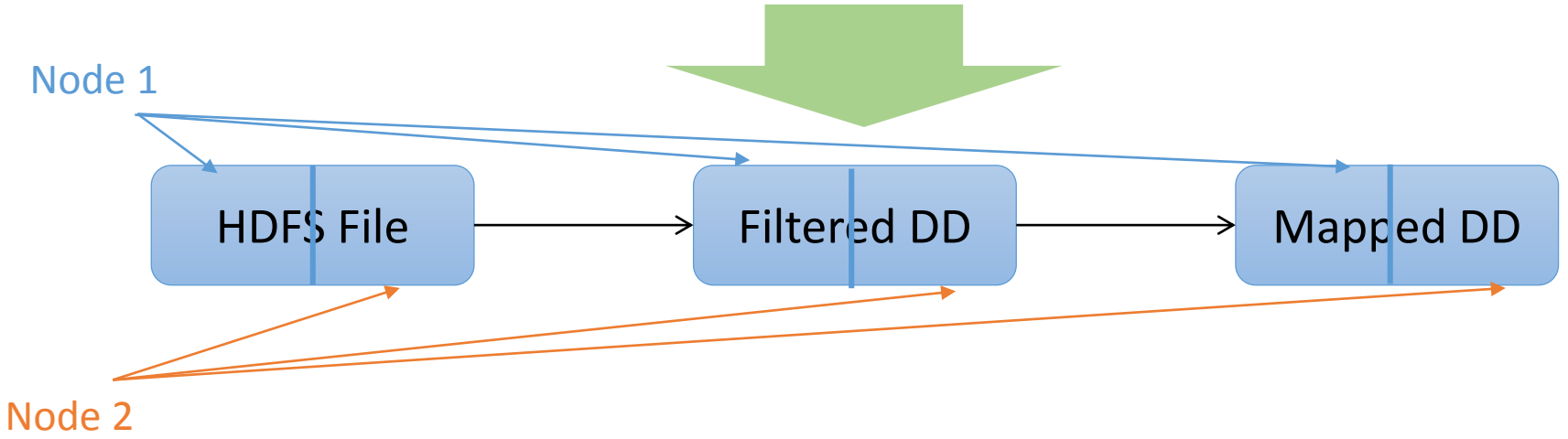
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.

BIG DATA

Handling fault tolerance

Ex:

```
messages = textFile(...).filter(startsWithERROR())
                        .map(split("\t")(2))
```



What is an RDD?

- When we add lineage information to the concept of a Distributed Dataset
  - We add ability to recreate it in case of failure
  - So, this data is now *resilient* to failures.
  - Hence called an ***RDD:*** *Resilient Distributed Dataset*

How is lineage information stored

- Lineage information is stored by keeping track of
  - Operations that are performed on
    - An RDD
    - That results in another RDD
  - What types of operations are supported?

RDD Operations : Transformations and Actions

Types of Operations

- Operations are of two types
  - ***Transformations***
  - ***Actions***

Transformation

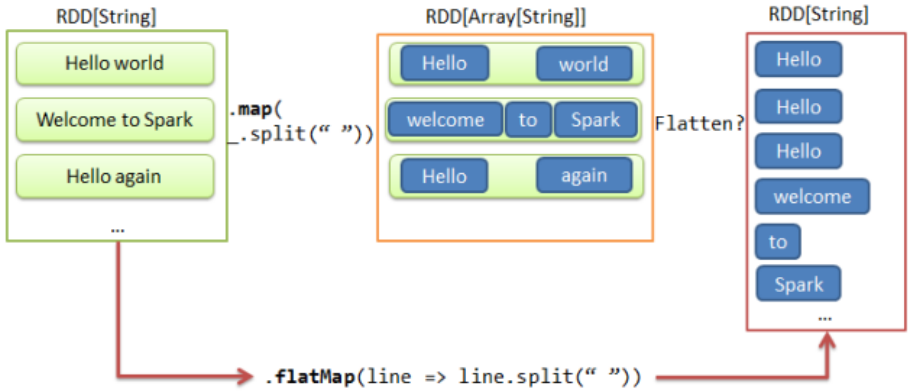
- Are operations that create a *new dataset* from an existing dataset
- For example:
  - *map()* is a transformation
    - Each line on input RDD is passed through the *map()* function
    - result of *map()* function applied on each value is stored in the output RDD.
- Note it is similar to the Map of map-reduce, but is more generic.

BIG DATA

Transformations

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic



BIG DATA

Transformations

Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training data).	`rdd.subtract(other)`	`{1, 2}`
`cartesian()`	Cartesian product with the other RDD.	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

Actions

- Are operations that return a value

- For example:
  - Reduce() is an action
  - Aggregates all elements of a RDD to produce a result.

BIG DATA

Actions

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	4
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`
`take(num)`	Return `num` elements from the RDD.	`rdd.take(2)`	`{1, 2}`
`top(num)`	Return the top `num` elements the RDD.	`rdd.top(2)`	`{3, 3}`
`takeOrdered(num)(ordering)`	Return `num` elements based on provided ordering.	`rdd.takeOrdered(2)(myOrdering)`	`{3, 3}`
`takeSample(withReplacement, num, [seed])`	Return `num` elements at random.	`rdd.takeSample(false, 1)`	Nondeterministic
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., `sum`).	`rdd.reduce((x, y) => x + y)`	9
`fold(zero)(func)`	Same as `reduce()` but with the provided zero value.	`rdd.fold(0)((x, y) => x + y)`	9

RDD Operations : Working with key-value pairs

BIG DATA

Key Value Pairs

- Consider our earlier operation of map/reduce using Spark
- Worked on datasets with only single values
- Let's consider how to represent <*key, value*> pairs
- Spark provides
  - Separate RDDs called pair RDDs for this
  - Separate operations to function on Pair RDDs

BIG DATA

Pair RDD Transformations

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey( (x, y) => x + y)`	`{(1, 2), (3, 10)}`
`groupByKey()`	Group values with the same key.	`rdd.groupByKey()`	`{(1, [2]), (3, [4, 6])}`
`combineByKey(createCombiner, mergeValue, mergeCombiners, partitioner)`	Combine values with the same key using a different result type.	See Examples 4-12 through 4-14.	

BIG DATA

Pair RDD Transformations

`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`
`flatMapValues(func)`	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	`rdd.flatMapValues(x => (x to 5)`	`{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)}`
`keys()`	Return an RDD of just the keys.	`rdd.keys()`	`{1, 3, 3}`
`values()`	Return an RDD of just the values.	`rdd.values()`	`{2, 4, 6}`

Pair RDD Actions

countByKey(k, V) → returns a HashMap of (k, Int) key value pairs with count of each key

Word Count in Spark

BIG DATA

Word Count in Spark

Create a spark context: tell Spark to create a new job

```
val sc = new SparkContext(new SparkConf().setAppName("Spark Count"))

val tokenized = sc.textFile(args(0)).flatMap(_.split(" "))

val wordCounts = tokenized.map((_, 1)).reduceByKey(_ + _)
```

Read in text file
Split it into words

Each of these is an RDD

Map each word to 1

Reduce by key. Can also use countbykey

https://docs.cloudera.com/documentation/enterprise/5-13-x/topics/spark_develop_run.html

Additional References

- What is Apache Spark? Matei Zaharia
  - https://www.youtube.com/watch?v=p8FGC49N-zM
- RDD, DataFrames and Datasets
  - https://www.youtube.com/watch?v=pZQsDloGB4w

THANK YOU

K V Subramaniam, Usha Devi

Dept. of Computer Science and Engineering

subramaniamkv@pes.edu

ushadevibg@pes.edu

BIG DATA

Spark : Architecture

K V Subramaniam
Computer Science and Engineering

BIG DATA

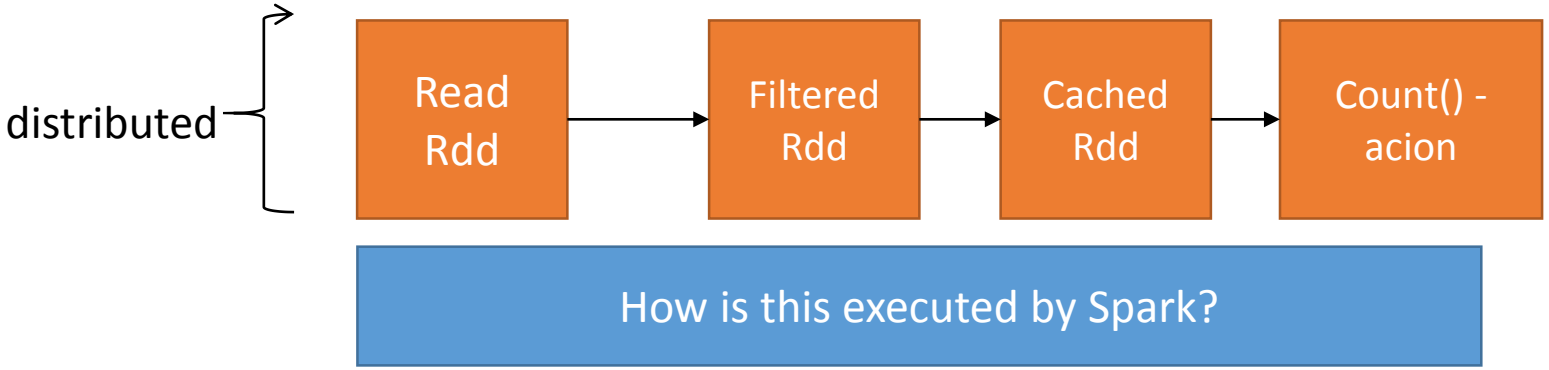
Overview of lecture

- Spark High Level Architecture
- Lifetime of a Spark Job
- Lazy Evaluation
- RDDs
- Spark Scheduling
- Dataframes

Spark High Level Architecture

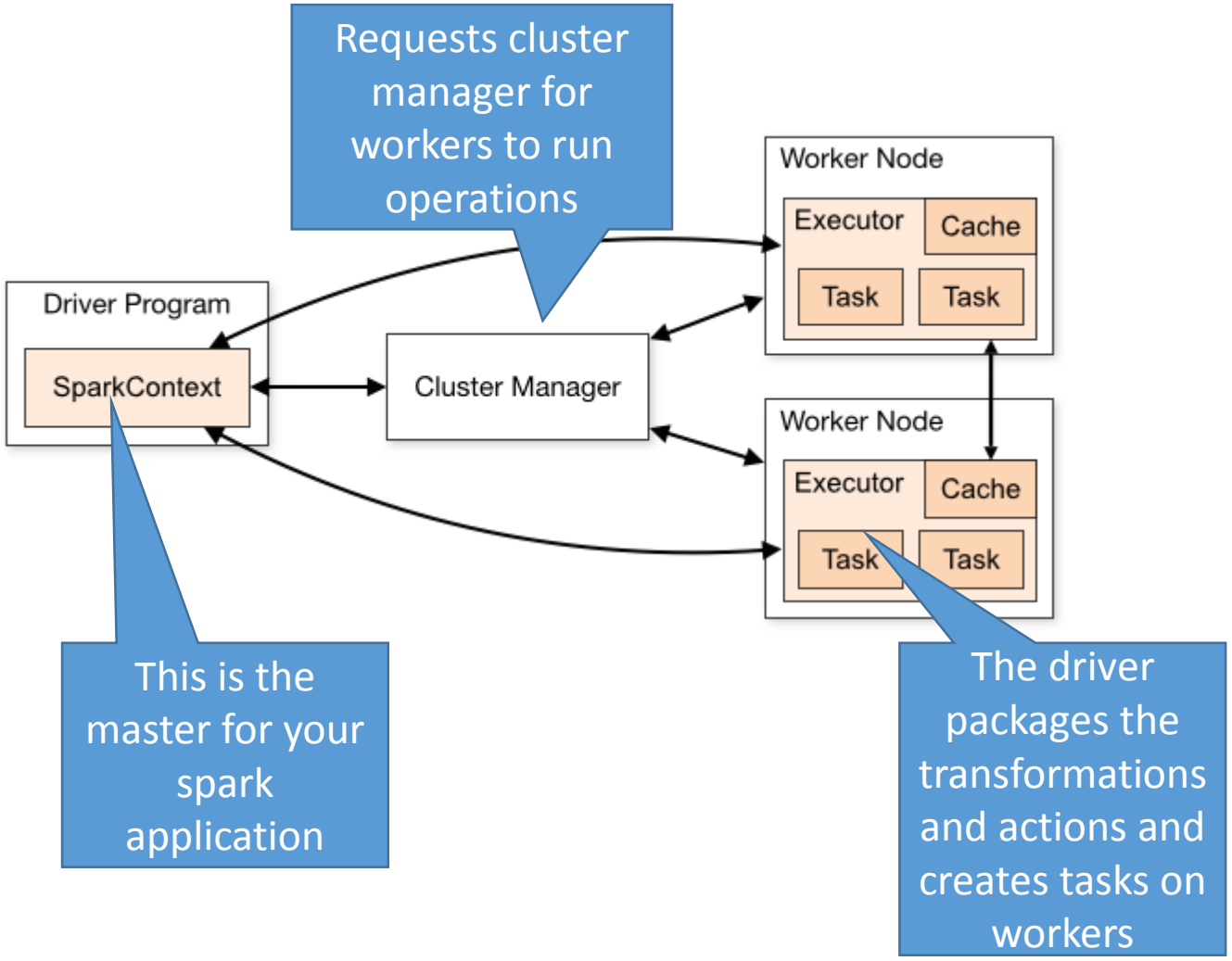
Recall: Log Mining Example

```
val sc = new SparkContext("spark://...", "MyJob", home,
    jars)

val file = sc.textFile("hdfs://...") // This is an RDD

val errors = file.filter(_.contains("ERROR")) // This is
    an RDD

errors.cache()

errors.count() // This is an action
```



BIG DATA

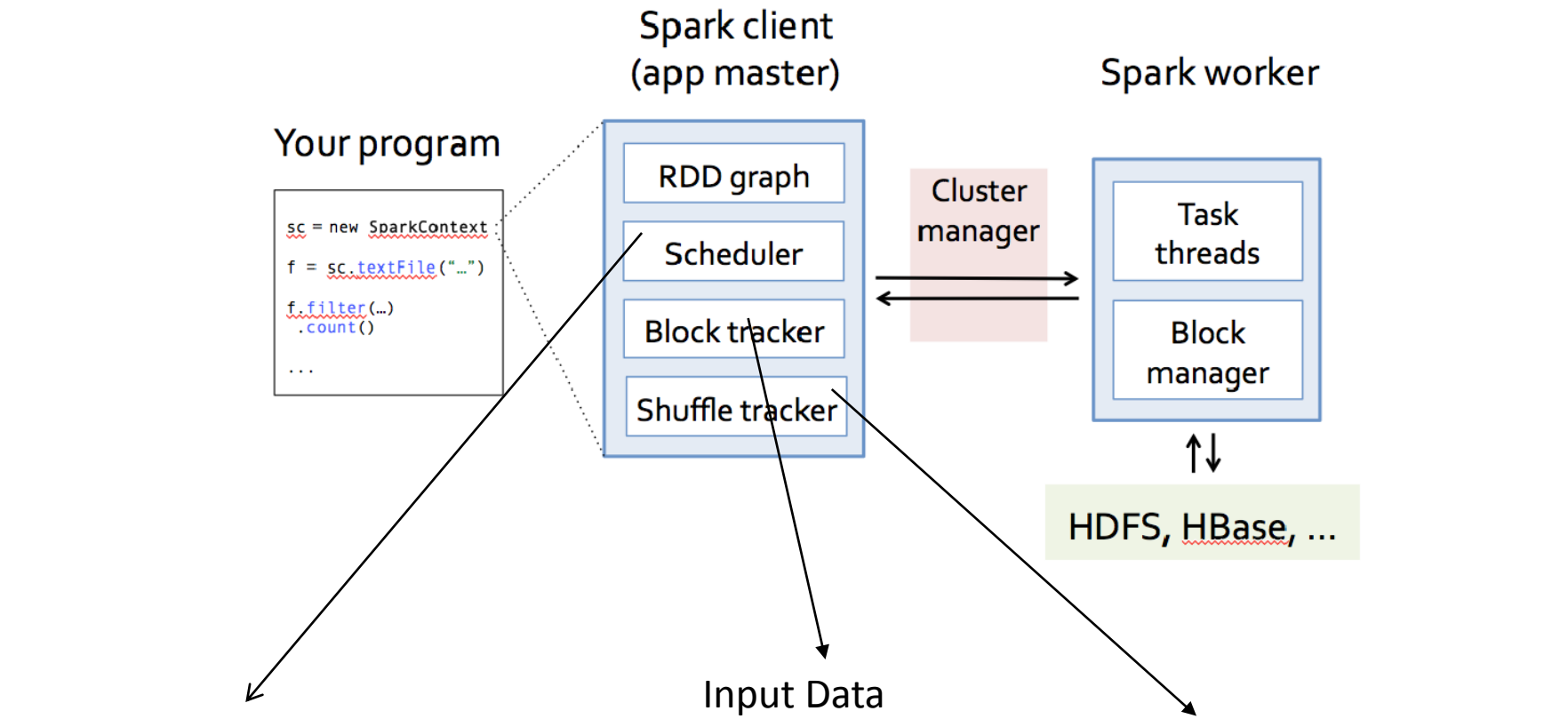
Spark Architecture



https://spark.apache.org/docs/latest/cluster-overview.html

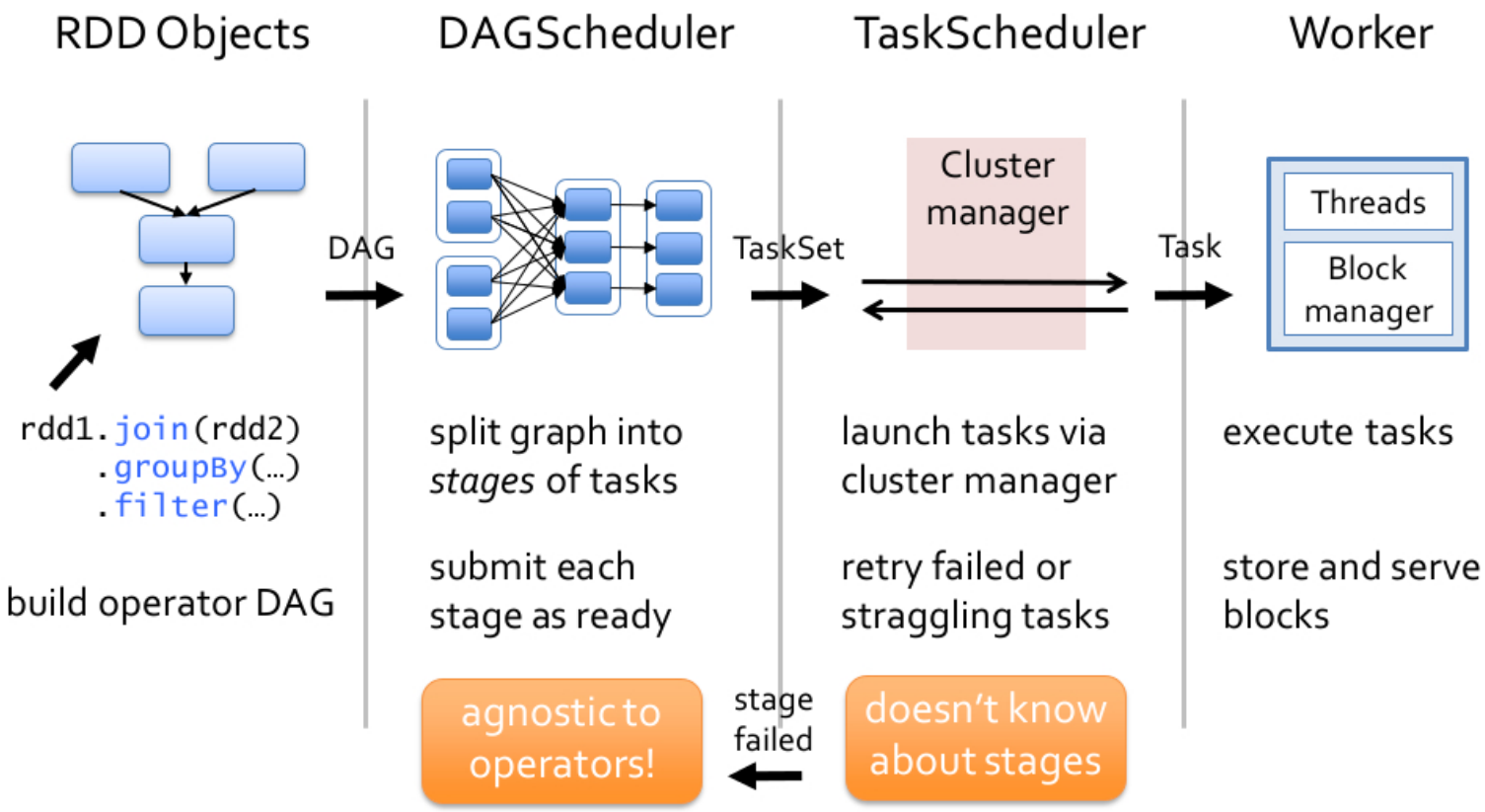
BIG DATA

Spark Architecture



BIG DATA

Spark Working details



https://hxquangnhat.com/2015/04/03/arch-spark-job-submission-breakdown/

Lazy Execution in Spark

BIG DATA

Lazy Execution

- In Hadoop, when we submit a job the master starts executing it
- In Spark, when does the master start executing the job?
    - Spark uses a technique called Lazy execution

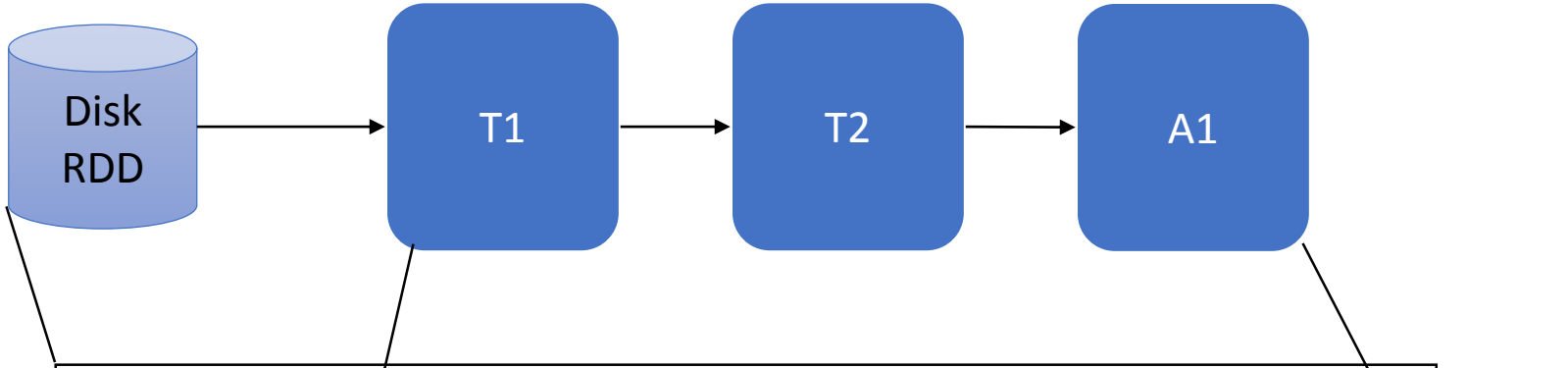
BIG DATA

Lazy Execution

- Remember that we defined Spark operations into *transformations* and *actions*
- The spark driver does not execute anything till it encounters an *action*
- *Transformations* are only noted for purpose of lineage.

BIG DATA

Lazy Execution




```
lines = spark.textFile("hdfs://...")
// Doesn't read file. Notes that RDD is to be
created
```

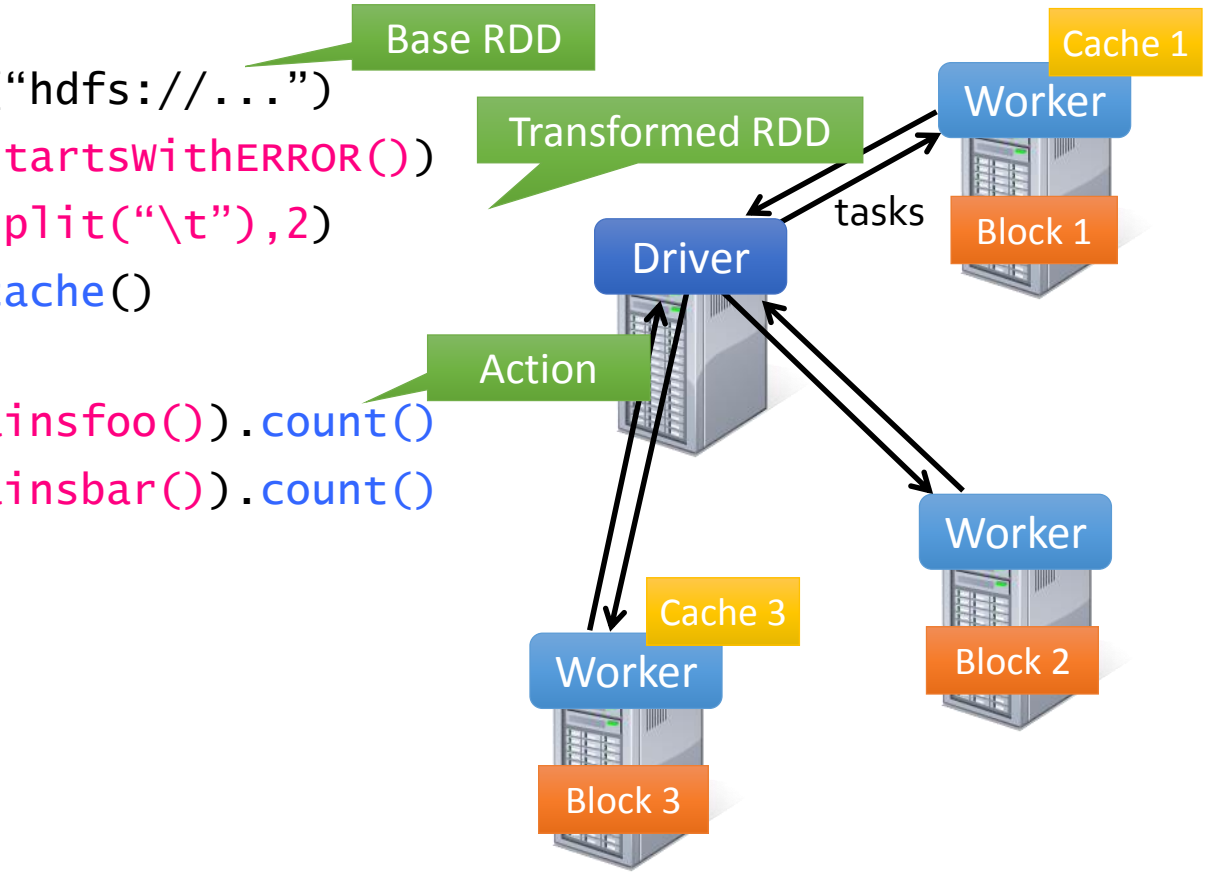
```
Transformation say map
Doesn't do transformation immediately
```

```
Action say count. Triggers execution: all
transformations done block by block on
same server followed by count
```

BIG DATA

Spark Working with Log Mining Example

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(startsWithERROR())
messages = errors.map(split("\t"),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
```



RDDs - details

What is an RDD

- RDD is partitioned, locality aware, distributed collections
  - RDDs are immutable. (Why it's necessary?)
- RDDs are data structures that either
  - Point to the source (HDFS)
  - Apply some transformations to the parent RDDs to generate new data elements
- Computations on RDDs
  - Lazily evaluated lineage DAGs composed of chained RDDs

Why the RDD abstraction?

- Support operations other than map and reduce
- Support in memory computation
- Arbitrary composition of such operators
- Simplify scheduling

How to capture dependencies generically?

Representing RDDs

- Set of partitions (“splits”)
  - Much like Hadoop. Each RDD associated with a input partitions
- List of dependencies on parent RDDs
  - Not there in Hadoop. This is new
- Function to compute a partition given parents
  - User defined code. (similar to map()/reduce() in Hadoop
- Optional preferred locations
  - For data locality
- Optional partitioning information (partitioner)
  - Advanced – for shuffle (see later)

BIG DATA

RDDs Interface

Operation	Meanning
partitions()	Return a list of Partition objects
preferredLocations(*p*)	List nodes where partition *p* can be accessed faster due to data locality
dependencies()	Return a list of dependencies
iterator(*p, parentIters*)	Compute the elements of partition *p* given iterators for its parent partitions
partitioner()	Return metadata specifying whether the RDD is hash/range partitioned

Examples of RDDs

- Hadoop RDD
  - Partitions – one per block
  - Dependencies – none
  - Compute (partition) – read corresponding block
  - Preferred locations – HDFS block location
  - Partitioner - none

- Filtered RDD (as in sample application)
  - Partitions – same as parent
  - Dependencies – 1-1 with parent
  - Compute – compute parent and filter it.
  - Preferred locations – ask parent (none)
  - Partitioner - none

Exercise

Based on the sample of the Filter RDD, can you work out what will be the partitions, compute, dependencies, preferred locations and partitioner for a joinRDD

- Filtered RDD (as in sample application)
  - Partitions – same as parent
  - Dependencies – 1-1 with parent
  - Compute – compute parent and filter it.
  - Preferred locations – ask parent (none)
  - Partitioner - none
-

Examples of RDDs

- Joined RDD
  - RDDPartitions – one per reduce task
  - Dependencies – shuffle on each parent
  - Compute (partition) – read and join shuffled data
  - Preferred locations – none
  - Partitioner – HashPartioner (num tasks)

Spark Scheduling

BIG DATA

Page Rank example in Spark

- lines = textfile (“urls.txt”))
- links = lines.map (lambda urls: urls.split()).groupByKey().cache()
- ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))
- for iteration in range(MAXITER)):
- contribs = links.join(ranks).flatMap(lambda url_neighbors_rank: computeContribs

  (url_neighbors_rank)
- ranks = contribs.reduceByKey(add).mapValues (lambda rank: rank * 0.85 + 0.15)

```
def computeContribs (url_neighbors_rank):
"""Calculates URL contributions to the rank of other
URLs.
"""
num_neighbors = len (url_neighbors_rank) – 2

rank = url_neighbors_rank [len (url_neighbors_rank) –
1]

for i in range (1, num_neighbors):
        yield (url_neighbors_rank[i], rank /
num_neighbors)
```

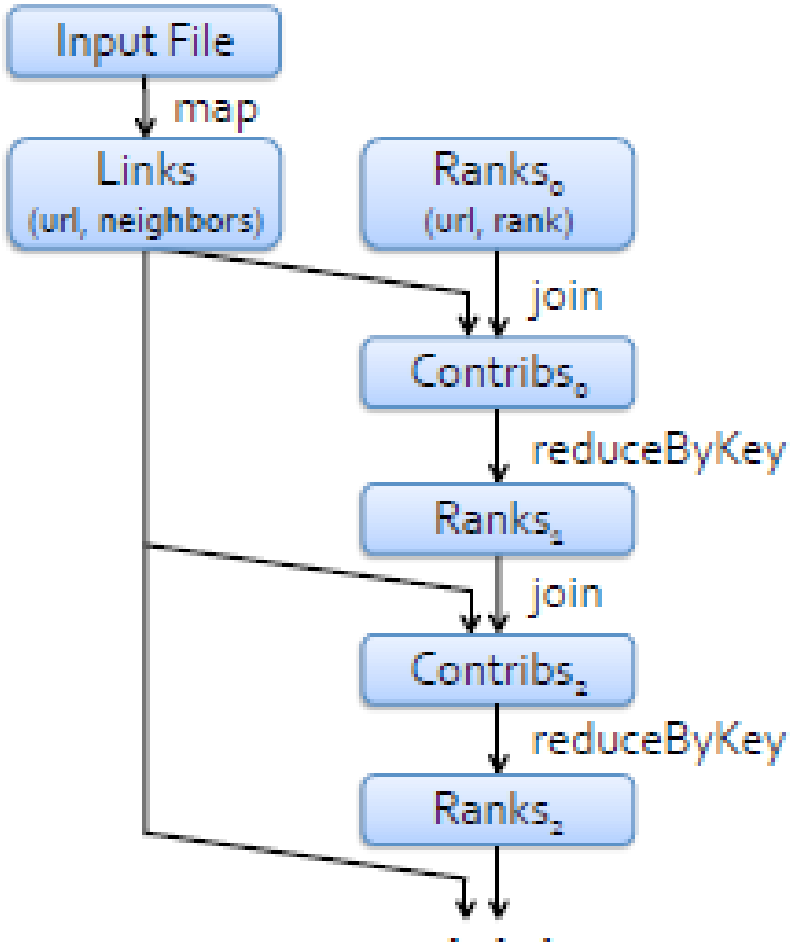
1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to

$$\Sigma_{i \in neighbors} rank_i / |neighbors_i|$$

DAG representation

- The Spark Driver will first convert this program into a DAG representation
- What does the DAG representation contain?
  - Each RDD is a node in the graph and
  - all transformations/actions on the RDD as edges

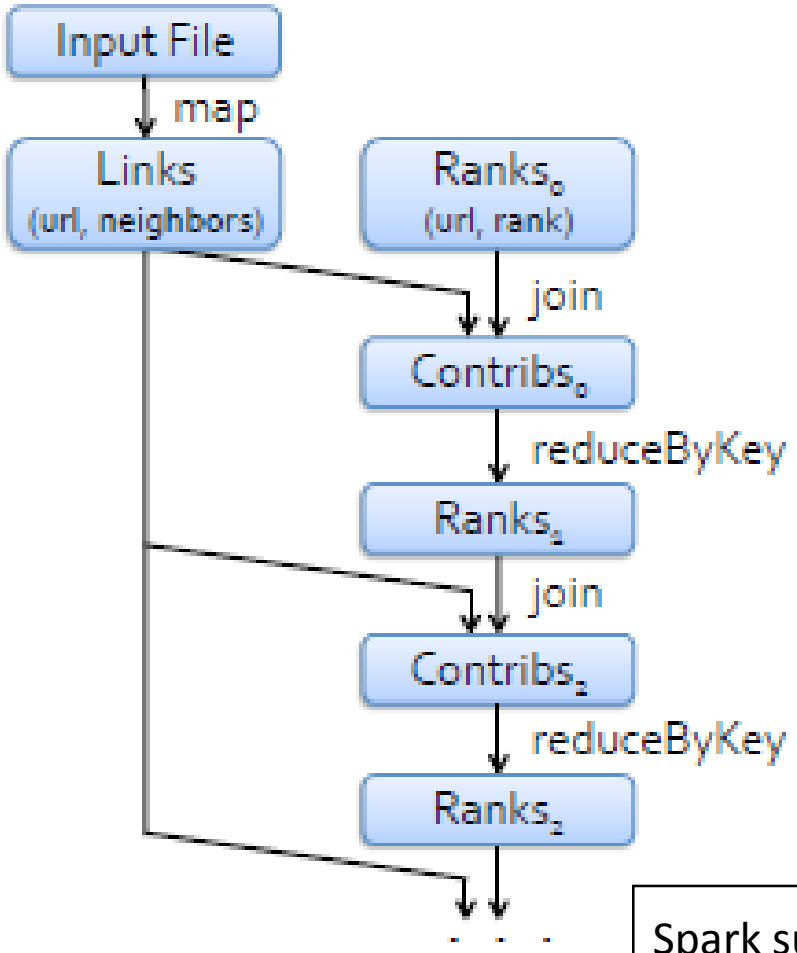
DAG representation of Page Rank




```
lines = textfile ("urls.txt"))

links = lines.map (lambda urls:
urls.split()).groupByKey().cache()

ranks = links.map(lambda url_neighbors:
(url_neighbors[0], 1.0))

 for iteration in range(MAXITER)):

        contribs = links.join(ranks).flatMap(
lambda url_neighbors_rank:
computeContribs
(url_neighbors_rank)

        ranks =
contribs.reduceByKey(add).mapValues(la
mbda rank: rank * 0.85 + 0.15)
```

Ranks and Links are spread across multiple nodes. How does Spark ensure join works properly? Hint: Think about how join works.

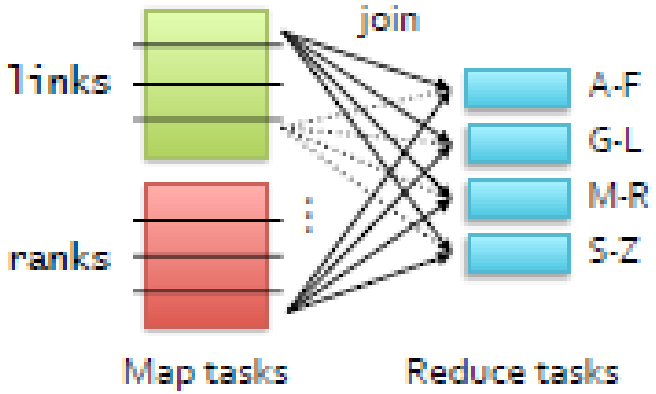
DAG representation of Page Rank



links and ranks are repeatedly joined

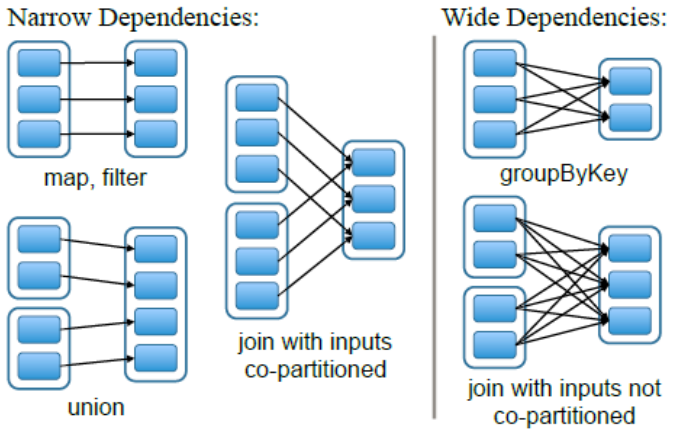
Each join requires a full shuffle over the network

» Hash both onto same nodes



Spark supports two types of partitioning – hash and range

- narrow dependencies
  - where each partition of the parent RDD is used by at most one partition of the child RDD
  - Does not need a shuffle; pipeline operations
  - Shuffle: movement of data from one node to another



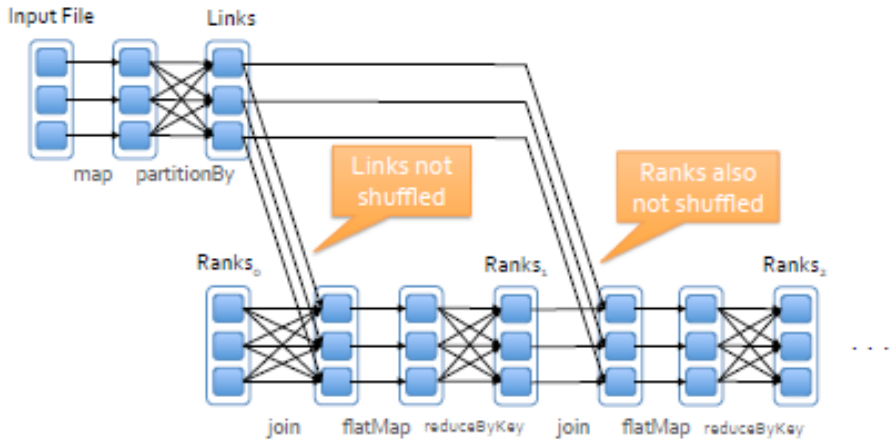
- wide dependencies
  - where multiple child partitions may depend on it.
  - May need a shuffle

- Copartition → technique to make sure that both inputs to a join are partitioned using same function

BIG DATA

Lineage and Optimizing Placement

- links & ranks repeatedly joined
- Can copartition them (e.g.hash both on URL) to avoid shuffles
- Spark supports two types of partitioning: hash and range




```
lines = textfile ("urls.txt"))

links = lines.map (lambda urls:
urls.split()).groupByKey().cache()

ranks = links.map(lambda url_neighbors:
(url_neighbors[0], 1.0))

 for iteration in range(MAXITER)):

        contribs = links.join(ranks).flatMap(
lambda url_neighbors_rank:
computeContribs
(url_neighbors_rank)

        ranks =
contribs.reduceByKey(add).mapValues(lam
bda rank: rank * 0.85 + 0.15)
```

Narrow and Wide Dependencies

Narrow Dependencies

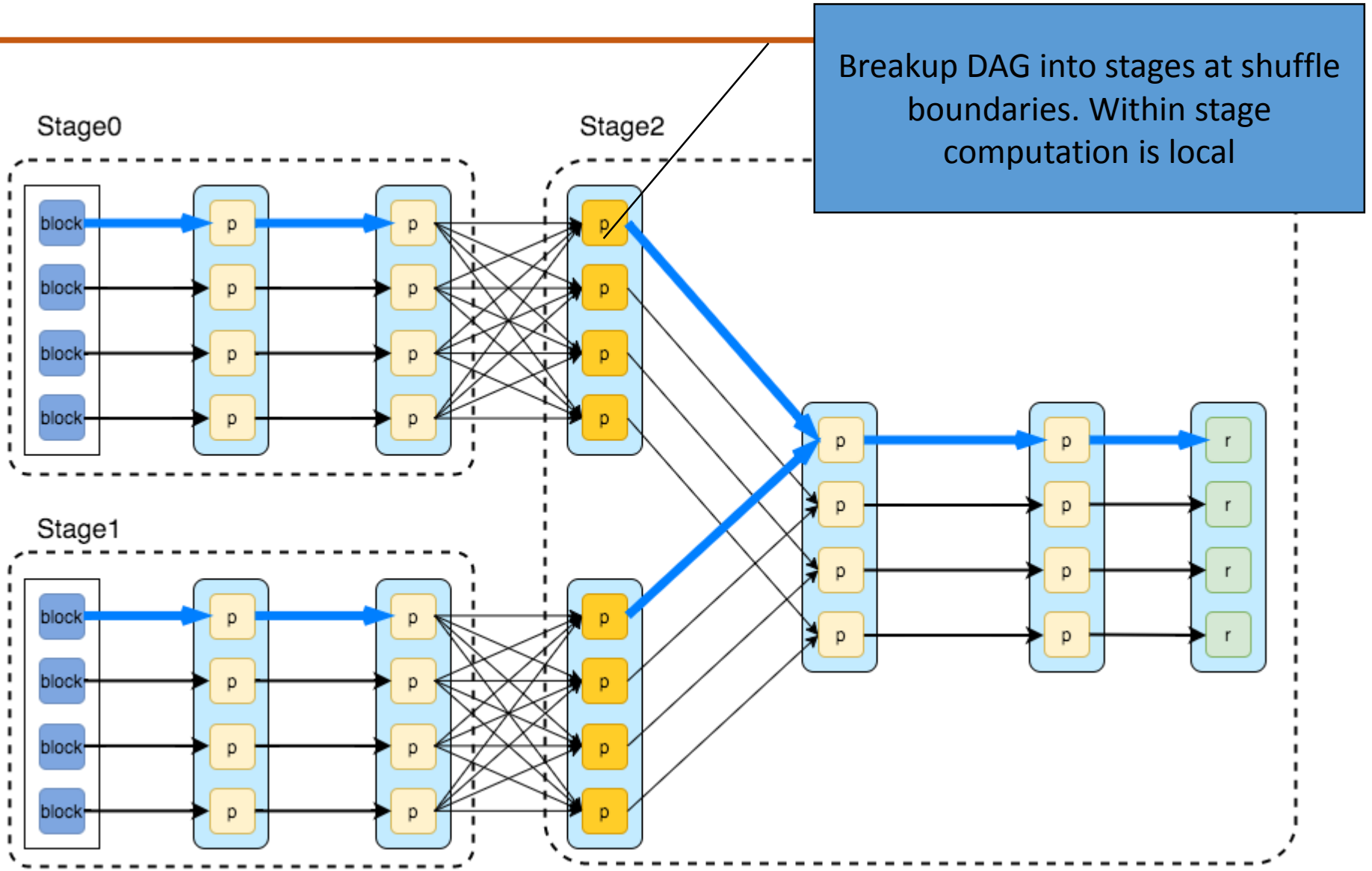
- Map
- FlatMap
- MapPartitions
- Filter
- Sample
- Union

W I d e Dependencies

- Intersection
- Distinct
- ReduceByKey
- GroupByKey
- Join
- Cartesian
- Repartition
- Coalesce

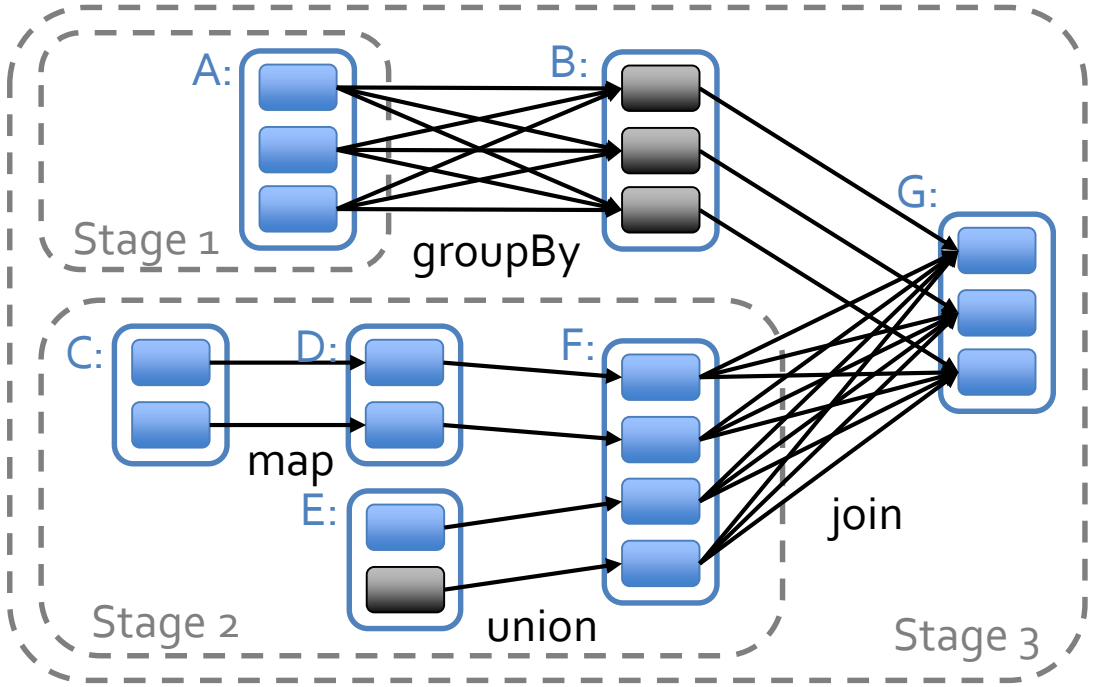
BIG DATA

Scheduling



Task Assignment

- scheduler assigns tasks to machines based on data locality using delay scheduling
  - if a task needs to process a partition that is available in
  - memory on a node, then send it to that node
  - otherwise, a task processes a partition for which the containing RDD provides preferred locations (e.g., an HDFS file), then send it to those



Simplifying tasks for programmers - DataFrames

BIG DATA

Need for dataframes

- Data RDDs are completely opaque to Spark
  - Meaning Spark cannot parse these values
- Is there some way to make Spark understand the format, so that we can do processing more easily
  - Like sql type queries
- Consider data like on the right that we need to run a query on?

USN	Name	Marks
45	Vkoli	11
10	Stendul	43
195	Abachpan	28

BIG DATA

What is a dataframe

- Introduced in 2015
- Inspired by Dataframes in R and Pandas in python
- Distributed collection of data into named columns
- An abstraction built over RDD that allows
  - Schema to be defined on a RDD
- Also has an optimizer built in for queries

USN	Name	Marks
45	Vkoli	11
10	Stendul	43
195	Abachpan	28

BIG DATA

Creating dataframes


```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.jsonFile("pes/students.json")

# let us display the contents
df.show()
## USN  name  marks
## 045  Vkoli   11
## 010  Stendul  43
## 195  Abachpan  28

Df = rdd.toDF("age", "name")
```

Dataframes can be created from existing RDDs, HIVE tables other Data sources.

This example is creating from a JSON file

Alternatively, from an existing RDD by naming the columns

BIG DATA

Using a dataframe


```
# Print the schema in a tree format
df.printSchema()
## root
## |-- usn: long (nullable = true)
## |-- name: string (nullable = true)
## |-- marks: long (nullable = true)

# Select only the "name" column
df.select("name").show()
## name
## VKoli
## STendul
## ABachpan

# Select everybody, but increment the age by 1
df.select("name", df.marks + 1).show()
## name    (marks + 1)
## Vkoli       12
## STendul     44
## ABachpan    29
```

Exercise

- Consider a case where you have data in a CSV file that consists of <pan number, date, tax_paid> and you wanted to find out the total tax paid by each individual pan holder
  - How will you do it in Spark?
  - How will you do it with Spark Data frames

Task Assignment

- Consider a case where you have data in a CSV file that consists of <pan number, date, tax_paid> and you wanted to find out the total tax paid by each individual pan holder
  - How will you do it in Spark?
  - How will you do it with Spark Data frames

Using Dataframes

```
Df = rdd.toDF("pan number", "date", "taxpaid")
Df.select("pan number", "tax paid").groupBy("pan number").sum()
```

Note that this is done using the name of the column rather than by splitting the data which we would do if used Spark.

DryadLINQ, FlumeJava

- Similar “distributed collection” API, but cannot reuse datasets efficiently *across* queries

- Relational databases
  - Lineage/provenance, logical logging, materialized views

GraphLab, Piccolo, BigTable, RAMCloud

- Fine-grained writes similar to distributed shared memory

- Iterative MapReduce (e.g. Twister, HaLoop)
  - Implicit data sharing for a fixed computation pattern

- Caching systems (e.g. Nectar)
  - Store data in files, no explicit control over what is cached

THANK YOU

K V Subramaniam, Usha Devi

Dept. of Computer Science and Engineering

subramaniamkv@pes.edu

ushadevibg@pes.edu

BIG DATA

PySpark - HandOn

K V Subramaniam
Computer Science and Engineering

BIG DATA

Overview of lecture

- What is PySpark
- Installation
- PySpark Architecture
- Word count with PySpark and Scala

What is PySpark

We looked previously at Scala as a language to program Spark

- But Spark also support a python binding

- Write code in python
  - With additional spark transformations/actions
  - Program runs as a Spark job

Interactive and Batch processing

- Supports the pyspark shell for interactive processing
- And regular batch processing jobs can be run by writing pyspark scripts

PySpark configuration

Downloading and Setting up pyspark

- Download and untar the spark tar file from the spark repository
- Pyspark is bundled along with spark
- However, it requires some configuration
  - Setup of proper paths.

BIG DATA

Pyspark configuration

- Needs environment variables to be setup
- First modify .bashrc to include

```
export SPARK_HOME = /home/hadoop/spark-2.1.0-bin-hadoop2.7
export PATH = $PATH:/home/hadoop/spark-2.1.0-bin-hadoop2.7/bin
export PYTHONPATH = $SPARK_HOME/python:$SPARK_HOME/python/lib/py4j-0.10.4-src.zip:$PYTHONPATH
export PATH = $SPARK_HOME/python:$PATH
```

Then run so that the env variables are setup

```
# source .bashrc
```

BIG DATA

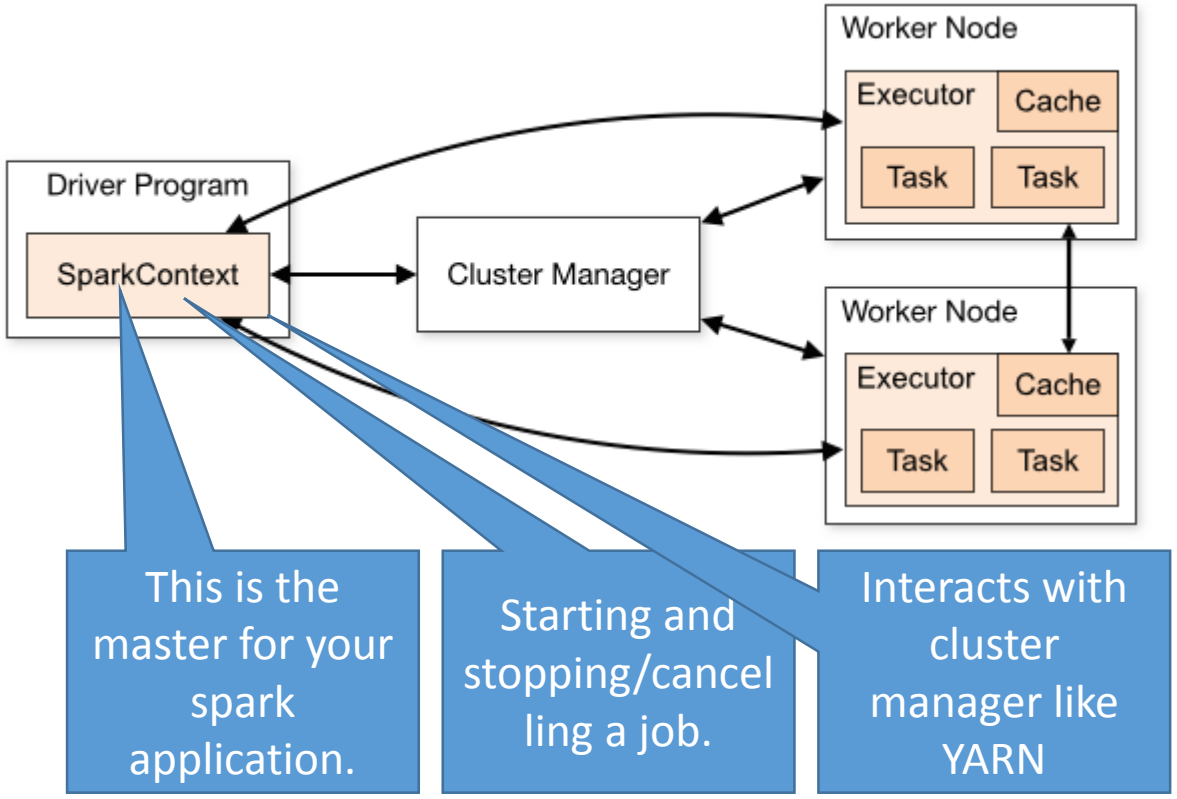
Starting pyspark

- Needs environment variables to be setup
- First modify .bashrc to include

```
./bin/pyspark
```

BIG DATA

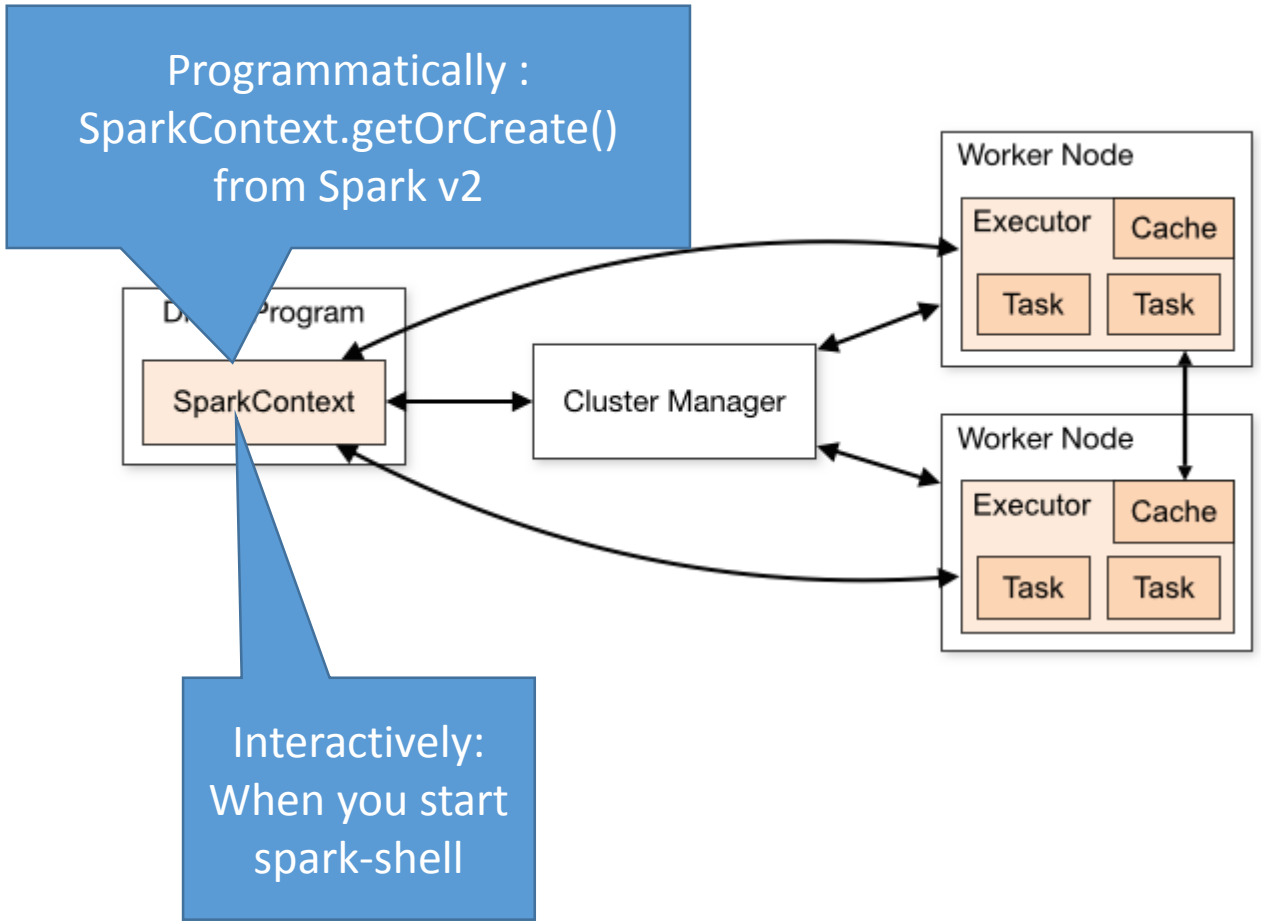
Recall: The Spark Context



https://spark.apache.org/docs/latest/cluster-overview.html

BIG DATA

When is Spark Context created?



There is one SparkContext per JVM

https://spark.apache.org/docs/latest/cluster-overview.html

Pyspark architecture

Pyspark architecture

- So if the spark context is maintained per JVM
- How does pyspark take care of the SparkContext?
  - Who creates and maintains this?

py4j

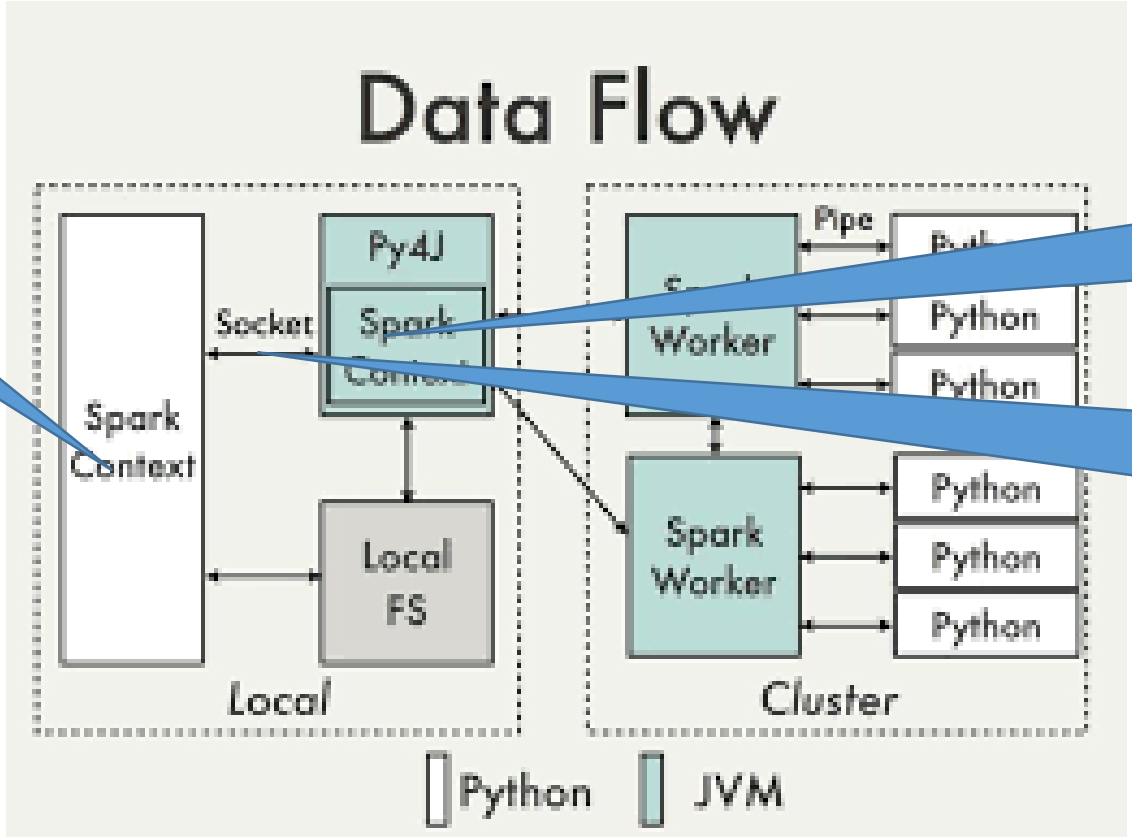
- Acts like a bridge between python and java
- Allows python interpreter to access Java objects instantiated within the JVM
- Can invoke methods on the Java objects as if they were python methods

https://www.py4j.org/

BIG DATA

Pyspark architecture

Python driver



Jave SparkContext used to drive the application

Data has to be serialized using cPickle serializer

https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals

PySpark Demo

THANK YOU

K V Subramaniam, Usha Devi

Dept. of Computer Science and Engineering

subramaniamkv@pes.edu

ushadevibg@pes.edu

BIG DATA

Hands On Session - 3

SPARK

K V Subramaniam

Usha Devi B G

Dept of Computer Science and Engineering

BIG DATA

SPARK Hands-on

SPECIFICATIONS

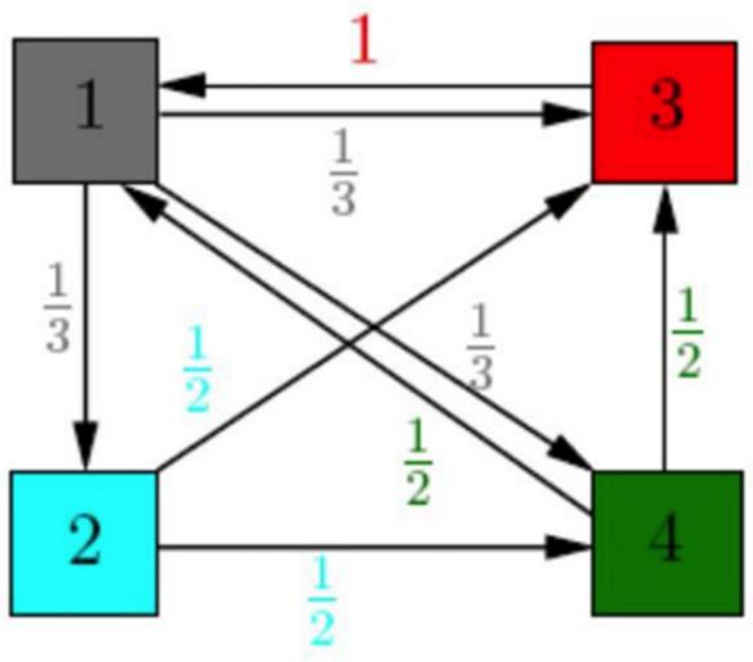
1. Hadoop: 3.2
2. Java: 1.8
3. Apache Spark 3.0
4. Dataset: Please download the dataset from the forum.

BIG DATA

Problem Statement

- Find the ranks of the 4 pages whose links have been given in the input file after 5 iterations of PageRank

Node/ Page	Edges/H yperlinks
1	3
1	2
1	4
2	3
2	4
3	1
4	3
4	1



BIG DATA

Code snippet

- lines = textfile (“urls.txt”))
- links = lines.map (lambda urls: urls.split()).groupByKey().cache()
- ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))
- for iteration in range(MAXITER)):
- contribs = links.join(ranks).flatMap(lambda url_neighbors_rank: computeContribs

  (url_neighbors_rank)
- ranks = contribs.reduceByKey(add).mapValues (lambda rank: rank * 0.85 + 0.15)

Node/ Page	Edges/H yperlinks
1	3
1	2
1	4
2	3
2	4
3	1
4	3
4	1

1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to

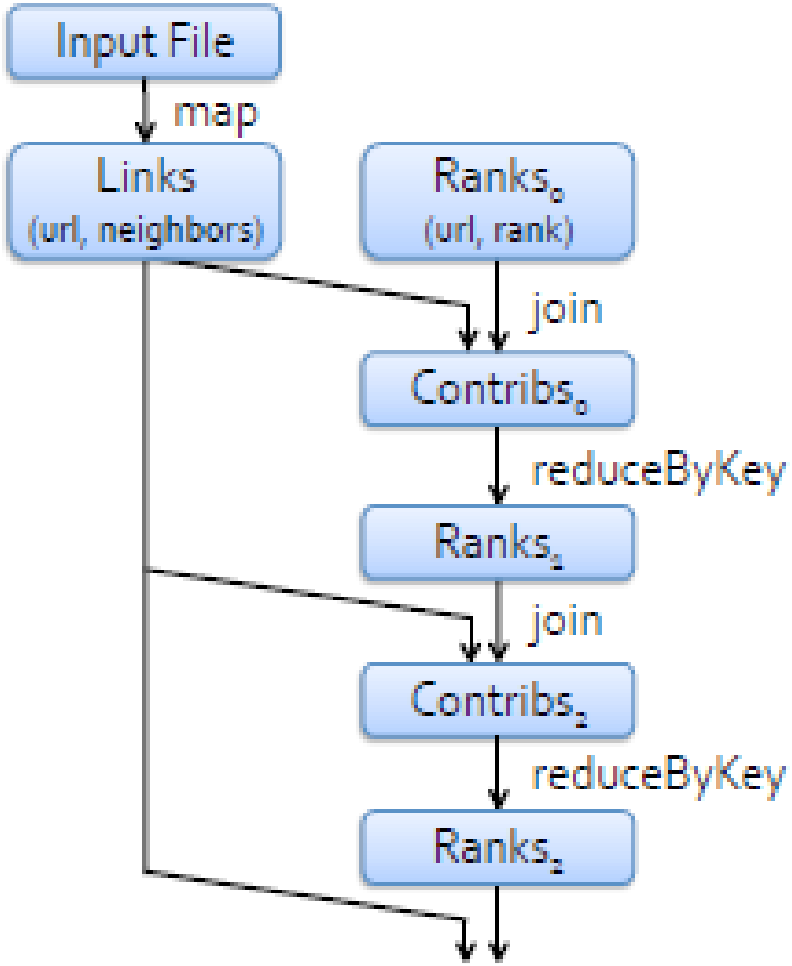
$$\Sigma_{i \in neighbors} rank_i / |neighbors_i|$$

DAG representation

- The Spark Driver will first convert this program into a DAG representation
- What does the DAG representation contain?
  - Each RDD is a node in the graph and
  - all transformations/actions on the RDD as edges

BIG DATA

DAG representation of Page Rank




```
lines = textfile ("urls.txt"))

links = lines.map (lambda urls:
urls.split()).groupByKey().cache()

ranks = links.map(lambda url_neighbors:
(url_neighbors[0], 1.0))

 for iteration in range(MAXITER)):

        contribs = links.join(ranks).flatMap(
lambda url_neighbors_rank:
computeContribs
(url_neighbors_rank)

        ranks =
contribs.reduceByKey(add).mapValues(la
mbda rank: rank * 0.85 + 0.15)
```

Steps to run PySpark Program

$ cd spark_dir

Load input data to HDFS

$ bin/spark-submit <path_to_file.py> <program_parameters>

$ bin/spark-submit pagerank.py <path/on/HDFS> 5

Problem Statement

- Update the given code to not accept number of iterations as a parameter.
- Your code should run till convergence with precision of 5 decimal digits (0.0001). Also print out the number of iterations it runs for.

THANK YOU

K V Subramaniam
Usha Devi B G

Department of Computer Science and Engineering

BIG DATA

Big Data Algorithm Complexity

K V Subramaniam
Computer Science and Engineering

BIG DATA

Overview of lecture

- Motivation – Algorithm complexity
- Communication Cost Complexity Model
- 3 Way joins with the communication cost complexity
- Key parameters
- Similarity join - analysis

Motivation – Algorithm Complexity

Why Study complexity?

- So far, we have looked at MapReduce algorithms
- However, for a particular problem, there could be many algorithms
- Which algorithm should we choose?
- This is why we study complexity of MapReduce
- We will actually study complexity of workflow systems
  - Generalization of MapReduce
  - Many important Big Data systems are workflow systems

Exercise

- Consider the following two problems
  - Matrix multiplication
  - Database query
- What would be the complexity of these algorithms when executing on a single node?
- What does complexity depend on?

Solution: Class Exercise

- Matrix multiplication
  - Expressed in terms of the bound on total #computations performed
- Database query
  - Complexity depends on disk read

Communication Cost Complexity

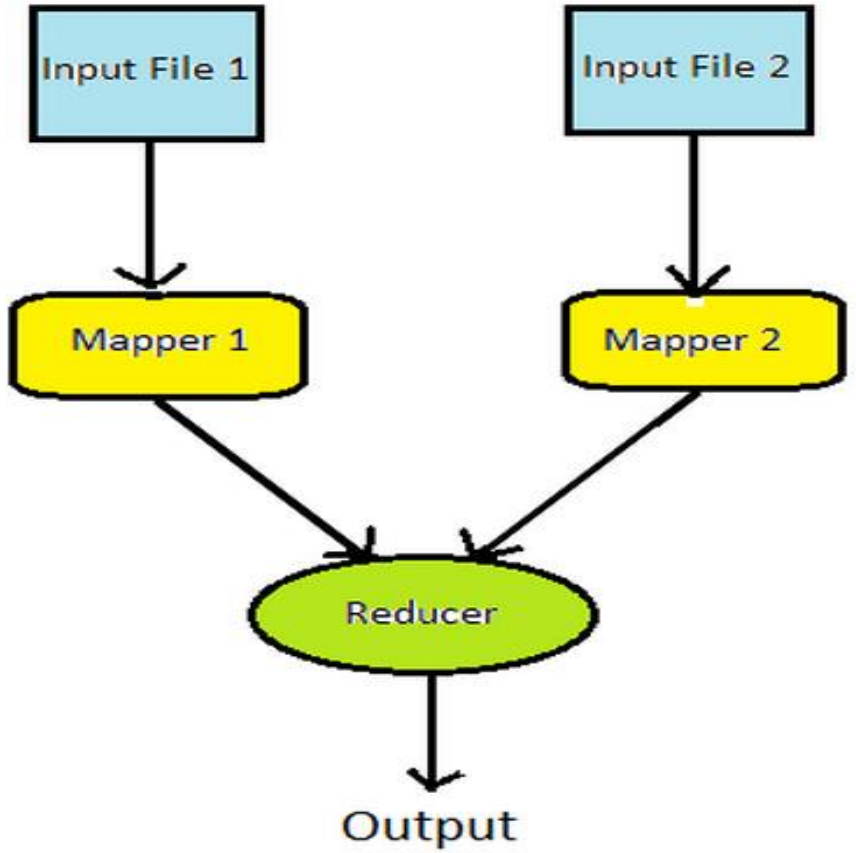
- Communication cost: size of input
- Why communication cost?
  - Algorithm tends to be linear in data
  - Network speed << CPU speed
  - Disk speed << CPU speed
  - Major time could be communication time

Communication Cost Complexity

- Why only input size?
  - Output is input to some other task
  - Final output is generally small by aggregation
    - Otherwise not human-readable

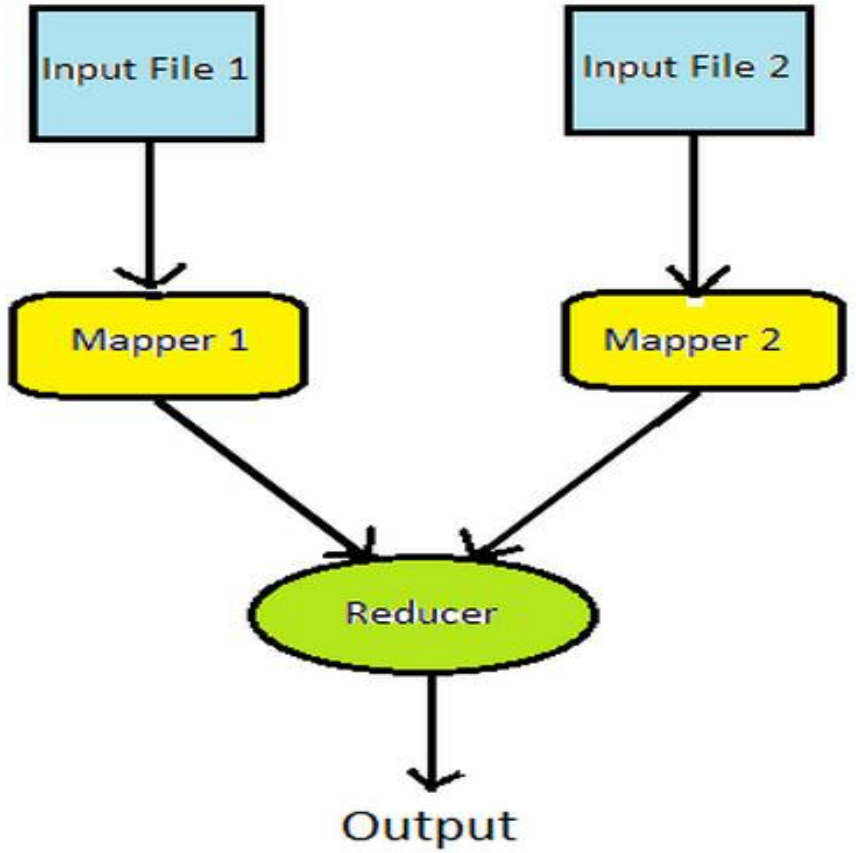
Natural Join: join R, S

- Mapper input complexity = *r+s*
  - Read data from disk
- Reducer input complexity = *r+s*
  - Network reads
- Total Complexity: *O(r+s)*



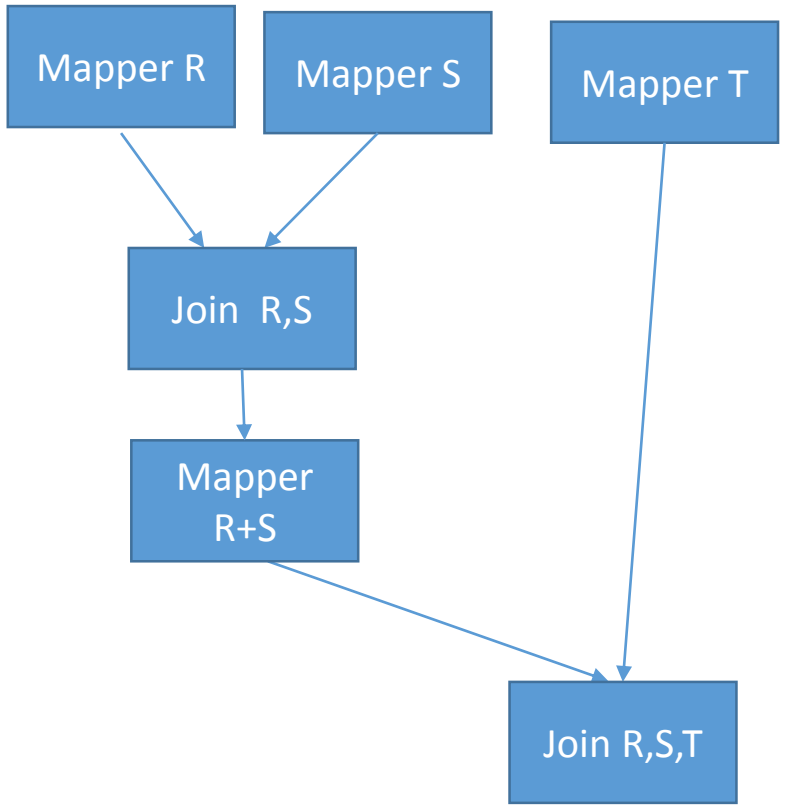
Exercise – Join complexity

- Consider three relations R, S and T
- How will you perform a join using map reduce across the three?
- Estimate the complexity



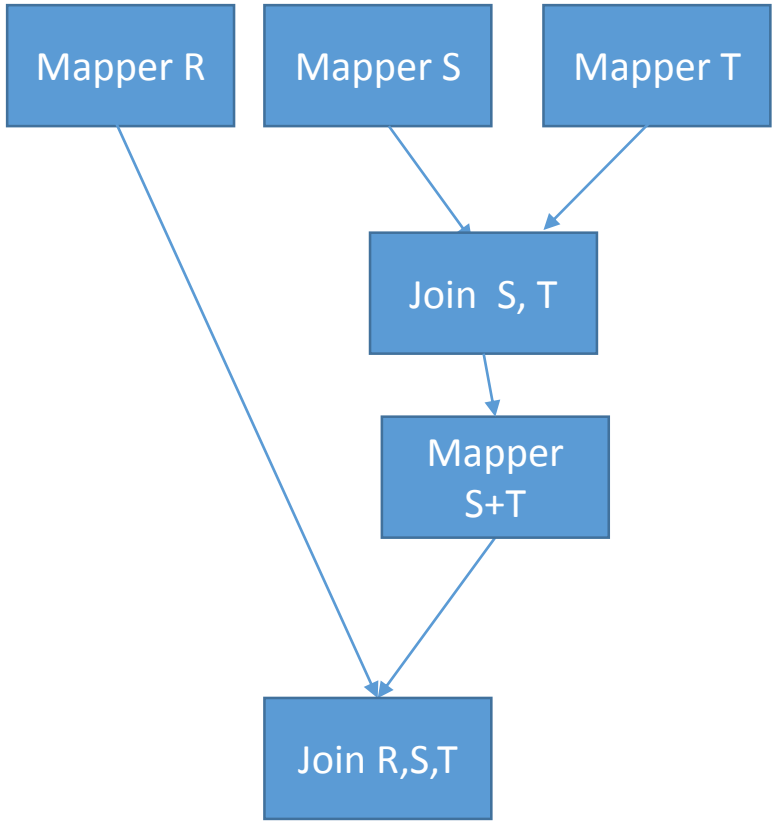
3-way Join: R, S, T

- 2 MapReduce phases
- Case 1: Join *R,S* and then join *T*
  - Input to Mapper 1: *r*
  - Input to Mapper 2: *s*
  - Input to Reducer 1: *r+s*
  - Let *p* be the probability of match between *r,s*
  - Input to Mapper 3: *prs*
  - Input to Mapper 4: *t*
  - Input to Reducer 2: *t+prs*
- Total Complexity: *O(r+s+t+prs)*



…3 Way Join R, S, T

- Case 1: Join *S,T* and then join *R*
  - Input to Mapper 1: *s*
  - Input to Mapper 2: *t*
  - Input to Reducer 1: *s+t*
  - Let *q* be the probability of match between *s,t*
  - Input to Mapper 3: *qst*
  - Input to Mapper 4: *r*
  - Input to Reducer 2: *r+qst*
- Total Complexity: *O(r+s+t+qst)*
  - Depends upon join order
  - If *p~q,* first join could be whichever of *rs, st, rt* is the smallest



Wall Clock Time

- Can make communication cost very low by executing all tasks on single CPU
- However, program may run slowly
- Need to consider *wall clock time*
  - Time taken for entire job to finish
- Dividing jobs could increase communication but reduce wall clock tme
  - Need to trade off communication time, wall clock time

Parameters: Wall Clock Time vs Communication cost

- Reducer size *q*
  - Max # of values that can have the same key
    - Not the number of reducers
  - If *q* is small, there can be more reducers
    - Suppose the number of Map outputs is *T*
    - Max number of reducers = *T/q*
    - This will reduce the wall clock time
    - But increase the communication cost
- Replication rate *r*
  - *r = (#key value pairs output by Mapper) / (# input records to Mapper)*
  - Average communication cost from Map tasks to Reduce tasks

Wall Clock Example: Similarity Join Between Images

- Assume we have a database of 1 million images
- Each image is 1 MB
- Total DB size = 1 TB
- Assume there is a function *s(x,y)* which determines how similar two images *x,y* are
  - *s(x,y) = s(y,x)*
- Problem: output all pairs *x,y* such that *s(x,y) > t*

Naïve Algorithm for example

- Assume each image P_i has an index i
- Mapper
  - Reads in (i,P_i)
  - Generates all pairs $(\{i,j\},\{P_i,P_j\})$
- Reducer
  - Reads $(\{i,j\},\{P_i,P_j\})$
  - Computes $s(P_i,Pj)$

Exercise

- What is the
  - Communication cost of the naïve algorithm?
  - Parallelism of the naïve algorithm?

Exercise - Solution

- What is the
  - Communication cost of the naïve algorithm?
  - Parallelism of the naïve algorithm?
- Algorithm doesn't work
  - Data to be transmitted = 1,000,000 x 999,999 x 1,000,000 bytes = 10^{18}
  - Communication cost is $\sim n^2$ where n is the number of images (extremely high)
  - However, potential parallelism is very high

The other extreme

- Do everything on one node
- Mapper
  - Reads in (i, P_i)
  - Generates all pairs $(\{i,j\},\{P_i, P_j\})$
- Reducer (runs on same node as mapper)
  - Reads $(\{i,j\},\{P_i, P_j\})$
  - Computes $s(P_i, Pj)$
- No communication cost
- Very low parallelism (wall clock time high)

Summary

Send one pair to each reducer

- High communication cost (bad)
- High parallelism (good)

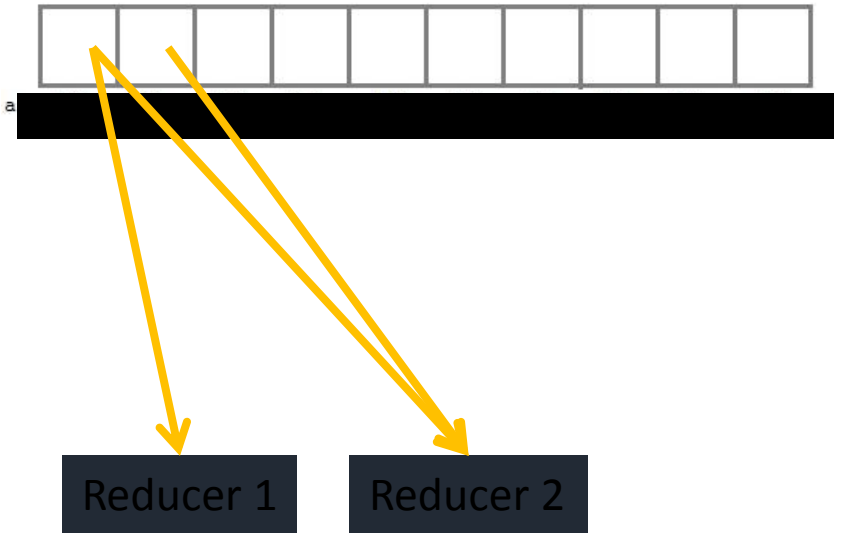
Do everything on one node

- Low parallelism (bad)
- Low communication cost (good)

Can we get something in between?

Group-based algorithm

- Overview
  - Group images
  - Read in two groups of images in a single reducer
  - Store them in memory
  - Compare all pairs from those groups
  - Output results



Example: 100 Groups

- Suppose the groups are G0, … G99
- Group G0 is sent to nodes 0, 1, ..., 98
  - Why?
- Group G0 has to be compared with 99 other groups
- Group G1 is sent to?
  - ~~0, 1, …, 98~~
- Group G1 is sent to 0, 99, 100, ...196 (0+98 other nodes)
- Group G2 is sent to?
- Group G2 is sent to 1, 99, 197, 198, ... 293 (1, 99 +97 other nodes)
- Group G3 is sent to 2, 100, 197, 294, ...389 (2, 100, 197, +96 other nodes)

In General

- If there are *g* groups
- The number of images per group *m=n/g*
- Each group is sent to *g-1* servers
- Total number of messages = *g(g-1)*
- **Total data = *mg(g-1) = n(g-1) ~ ng***
- **Parallelism = no of nodes = *(g-1) + (g-2) + (g-3) + … = g(g-1)/2 =O(g^2)***
- Another way = no of nodes = $^nC_2$ *=g(g-1)/2*

Exercise

- Suppose we have groups of 100
  - How many groups are there?
  - How many nodes is each group sent to?
- What is the
  - Communication cost of the algorithm?
  - Parallelism of the algorithm?

…Group Based Algorithm…

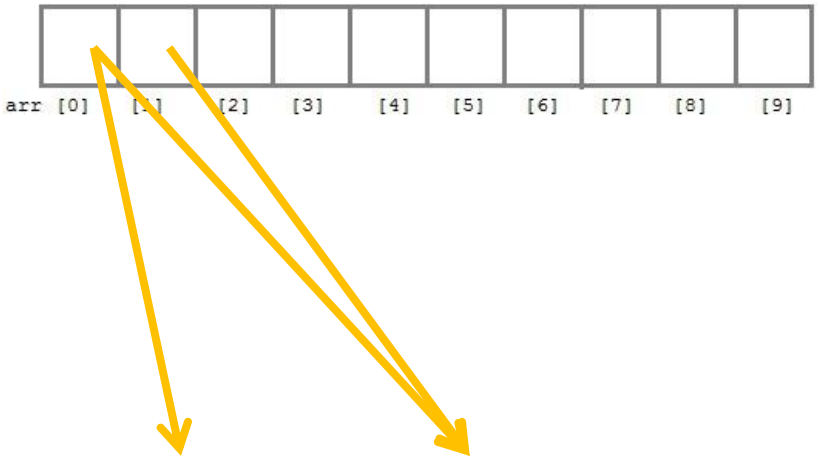
- Naïve: approximately 1,000,000x1,000,000 images = 10^{12} images, parallelism = 10^{12}
- Example: group images into groups of100
  - Communication cost: 1,000,000 images x 100 groups = 10^8 images
  - Parallelism = 10^4
- Example: group images into groups of1000
  - Communication cost = 1,000,000 x 1000 = 10^9 images
  - Parallelism = 10^6
- Trade-off: increasing group size
  - Increases communication complexity (more reads)
  - Reduces wall clock time (increases parallelism)

…Group Based Algorithm

- On next slide
- Note
  - Group-based algorithm is discussed in book as if mapper sends pictures to reducer
  - In previous slides, we have discussed as if reducer reads in pictures from disk
  - From communication cost complexity both are the same
  - Performance: probably better to read from disk

Group Based algorithm

- Overview
  - Group images
  - Read in two groups of images in a single reducer
  - Store them in memory
  - Compare all pairs from those groups
  - Output results
- Complexity
  - Communication cost ~*ng* where *g* is the number of groups
  - Can still get good parallelism



Grouping images..

- Group images into *g* groups
- Mapper
  - Input: (i, P_i)
  - Find *u* = group to which image *i* belongs
  - Output *g-1* key-value pairs $(\{u,v\}, i, P_i)$ for all *v != u*
- Reducer:
  - There is one reducer for each unique key {*u,v*}
  - Use the input to store images for groups *u,v*
  - Compare all pairs of images in groups *u,v*
  - If *v=u+1,* then compare all pairs of images in group *u*

Actual Values

- If group size is 1000
- Need ~2GB to store 2000 images in memory
- Need ~500,000 reducers
- If we have a 10,000 node cluster
  - Can do computation in 50 passes
  - Speedup of 10,000

Summary

- There could be many MapReduce algorithms to implement a particular functionality
  - Matrix multiplication
  - Multi-way joins
- There are two factors to consider when selecting an algorithm
  - Communication complexity: volume of data that is input to the different phases of the algorithm
  - Wall clock time: Total elapsed time for algorithm
    - Depends upon parallelism
  - Frequently there is a trade-off between these factors
    - Group size

References

- Mining of Massive Datasets
  - Rajaraman et. Al.
  - Chapter 2.4-2.5

THANK YOU

K V Subramaniam, Usha Devi

Dept. of Computer Science and Engineering

subramaniamkv@pes.edu

ushadevibg@pes.edu