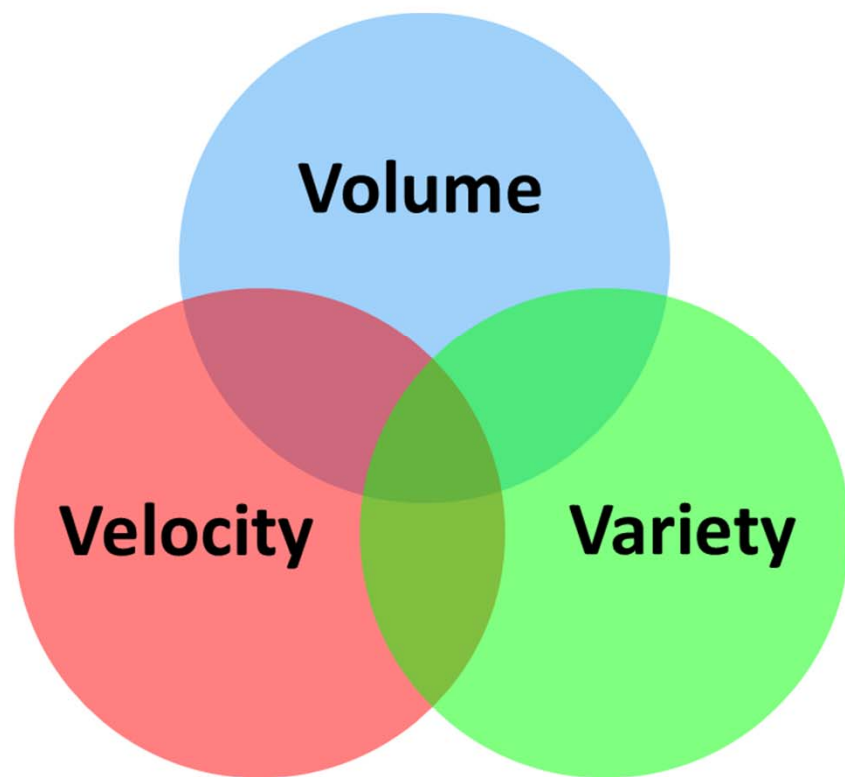


大数据系统与大规模数据分析

# 大数据运算系统 (3)



陈世敏

中科院计算所  
计算机体系结构  
国家重点实验室

©2015-2021 陈世敏

# 作业时间安排

周次	内容	作业
第4周, 3/31	大数据存储系统1: 基础, 文件系统, HDFS	作业1布置
第5周, 4/7	大数据存储系统2: 键值系统	
第6周, 4/14	大数据存储系统3: 图存储, document store	
第7周, 4/21	大数据运算系统1: MapReduce, 图计算系统	作业1提交 作业2布置
第8周, 4/28	大数据运算系统2: 图计算系统, MR+SQL	
第9周, 5/5=>5/8 (周六上周三的课)	大数据运算系统3: 内存计算系统	大作业布置 (系统, 6人/组)
第10周, 5/12	分布式哈希表, 区块链技术中的加密算法	作业2提交
第11周, 5/19	最邻近搜索和位置敏感 (LSH) 算法	作业3
第12周, 5/26	奇异值分解与数据空间的维度约化	大作业布置 (分析, 3人/组)
第13周, 6/2	推荐系统	大作业 仅选1个
第14周, 6/9	流数据采样与估计、流数据过滤与分析	
第15周, 6/16	期末考试	
第16周, 6/23	大作业验收报告 (上下午, 教1-208)	大作业验收

# Outline

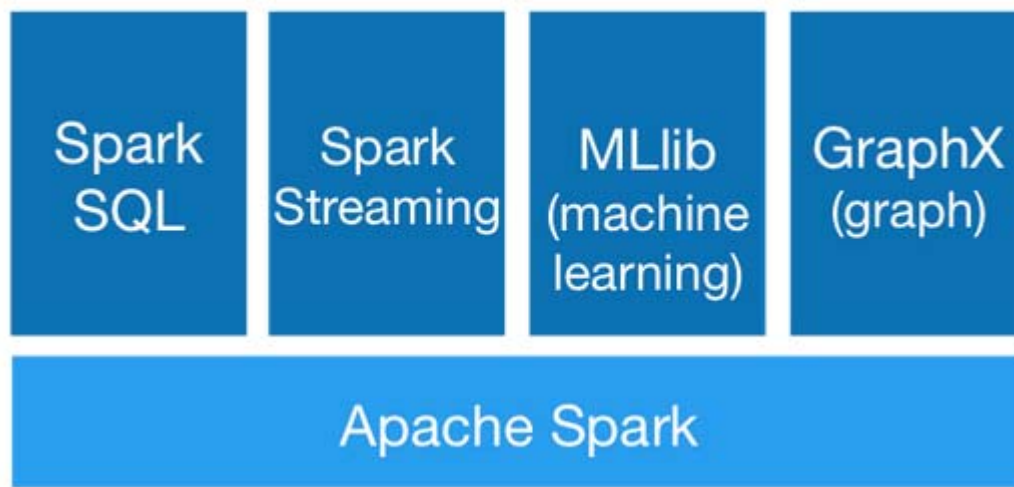
- 内存计算

- 内存通用大数据运算系统: Spark

# Spark

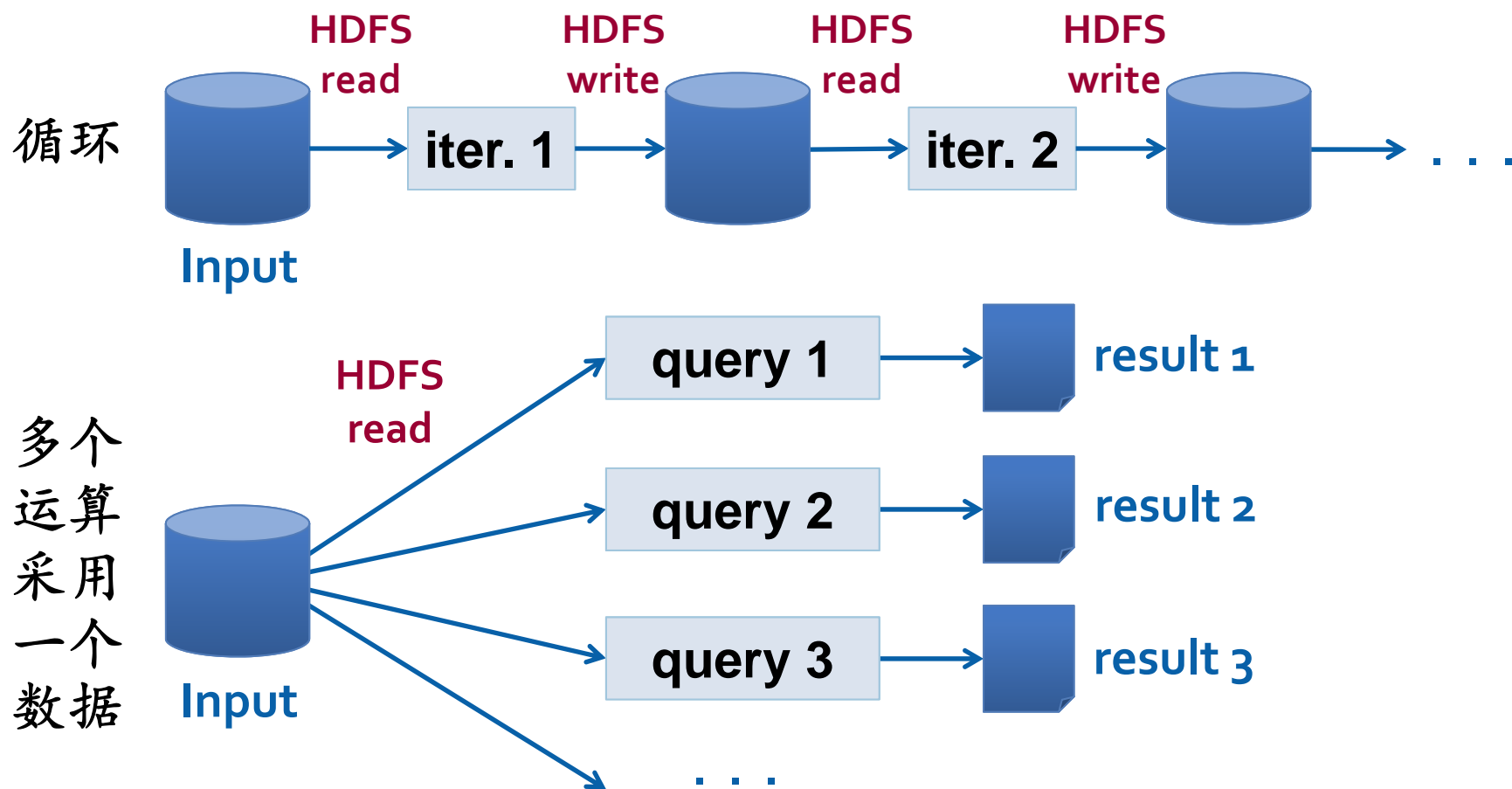
- 原理
- 编程
- 系统实现
- Spark SQL
- Spark Streaming

# Spark: 面向大数据分析的内存系统



- Berkeley AMP Lab 研发
- 可以从HDFS读数据，但是运算中数据放在内存中，不使用Hadoop，而是新实现了分布式的处理
- 目标是低延迟的分析操作
- “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”, NSDI’12

# MapReduce的问题



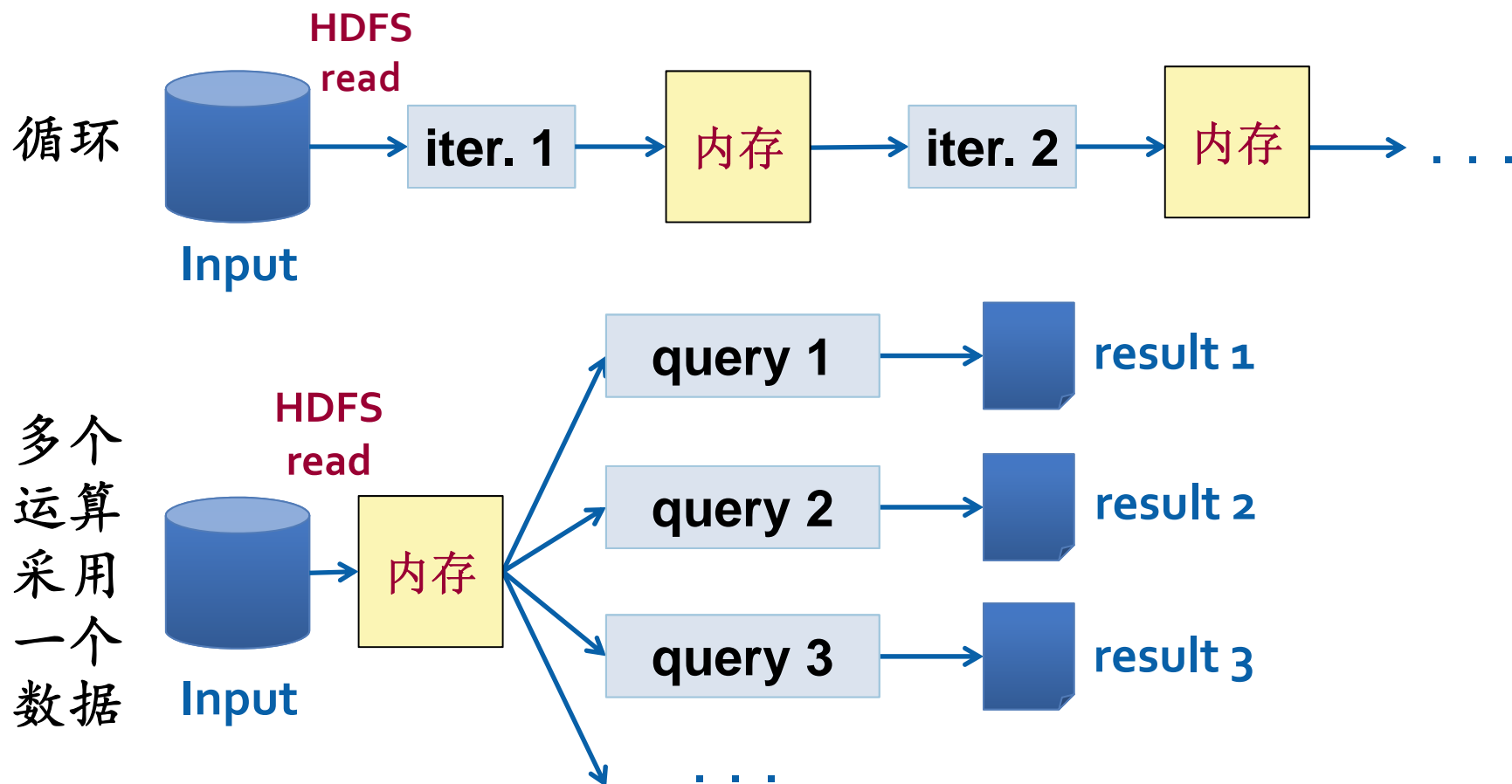
- 通过HDFS进行作业间数据共享，代价太高

图来源：NSDI'12 slides

# Spark的思路

- 内存容量越来越大
- 把数据放入多台机器的内存
- 避免HDFS开销

# Spark的思路





# 基础数据结构：RDD

- Resilient Distributed Data sets

- 一个数据集
- 只读，整个数据集创建后不能修改
- 通常进行整个数据集的运算

- 优点

- 并发控制被简化了
- 可以记录lineage（数据集上的运算序列），可以重新计算
  - 并不需要把RDD存储在stable storage上

# Spark

- 原理
- 编程
- 系统实现
- Spark SQL
- Spark Streaming

# Scala

- Spark支持的主要语言（之一）
  - 其它语言：Java, Python
- Scala是一种新的程序设计语言
  - 面向目标的（Object Oriented）
  - 函数型（Functional）
- Scala程序是在JVM上执行的
- Scala语言资料 <http://www.scala-lang.org/>
  - 本课程不进行深入讲解

# 简单的例子（Scala）

```
/* SimpleApp.scala */  
import org.apache.spark.sql.SparkSession  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()  
    val logData = spark.read.textFile(logFile).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println(s"Lines with a: $numAs, Lines with b: $numBs")  
    spark.stop()  
  }  
}
```

例子来源：Spark Manual

# 简单的例子（Scala）

```
/* SimpleApp.scala */  
import org.apache.spark.sql.Session  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()  
    val logData = spark.read.textFile(logFile).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println(s"Lines with a: $numAs, Lines with b: $numBs")  
    spark.stop()  
  }  
}
```

有些像Java import,  
或者C/C++ include,  
所需要的库说明

例子来源：Spark Manual

# 简单的例子（Scala）

```
/* SimpleApp.scala */  
import org.apache.spark.sql.SparkSession  
  
object SimpleApp {  
  def main(args: Array[String]) {  
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system  
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()  
    val logData = spark.read.textFile(logFile).cache()  
    val numAs = logData.filter(line => line.contains("a")).count()  
    val numBs = logData.filter(line => line.contains("b")).count()  
    println(s"Lines with a: $numAs, Lines with b: $numBs")  
    spark.stop()  
  }  
}
```

主程序，  
实际上是**driver**，  
发出**Spark**操作请求

例子来源：Spark Manual

# 简单的例子（Scala）

```
/* SimpleApp.scala */
import org.apache.spark.sql.SparkSession

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // should be some file on your system
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
  }
}
```

建立或获取  
SparkSession

例子来源：Spark Manual



# 简单的例子 (Scala)

```
/* SimpleApp.scala */
import org.apache.spark.sql.SparkSession

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
  }
}
```

读文本文件生  
成一个RDD

例子来源: Spark Manual



# 简单的例子（Scala）

```
/* SimpleApp.scala */
import org.apache.spark.sql.SparkSession

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
  }
}
```

**RDD操作**

- filter: 对RDD每个元素，调用给定函数，True保留, False丢弃，生成新的RDD
  - 函数line=>line.contains("a") 采用Lambda表达式
- count()是计数RDD中有多少元素

例子来源：Spark Manual

# 同一个的例子 (Java)

```
/* SimpleApp.java */
```

```
import org.apache.spark.sql.Session;  
import org.apache.spark.sql.Dataset;
```

Java import

```
public class SimpleApp {  
    public static void main(String[] args) {  
        String logFile = "YOUR_SPARK_HOME/README.md"; // should be some file on your system  
        Session spark = Session.builder().appName("Simple Application").getOrCreate();  
        Dataset<String> logData = spark.read().textFile(logFile).cache();  
  
        long numAs = logData.filter(s -> s.contains("a")).count();  
        long numBs = logData.filter(s -> s.contains("b")).count();  
  
        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);  
  
        spark.stop();  
    }  
}
```

例子来源: Spark Manual

# 同一个的例子 (Java)

```
/* SimpleApp.java */
import org.apache.spark.sql.Session;
import org.apache.spark.sql.Dataset;

public class SimpleApp {
    public static void main(String[] args) {
        String logFile = "YOUR_SPARK_HOME/README.md"; // should be some file on your system
        Session spark = Session.builder().appName("Simple Application").getOrCreate();
        Dataset<String> logData = spark.read().textFile(logFile).cache();

        long numAs = logData.filter(s -> s.contains("a")).count();
        long numBs = logData.filter(s -> s.contains("b")).count();

        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);

        spark.stop();
    }
}
```

主程序，  
实际上是driver，  
发出Spark操作请求

# 同一个的例子 (Java)

建立或获取  
SparkSession

```
/* SimpleApp.java */
import org.apache.spark.sql.Session;
import org.apache.spark.sql.Dataset;

public class SimpleApp {
    public static void main(String[] args) {
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
        SparkSession spark = SparkSession.builder().appName("Simple Application").getOrCreate();
        Dataset<String> logData = spark.read().textFile(logFile).cache();

        long numAs = logData.filter(s -> s.contains("a")).count();
        long numBs = logData.filter(s -> s.contains("b")).count();

        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);

        spark.stop();
    }
}
```



# 同一个的例子 (Java)

读文本文件  
生成一个  
**Dataset**

```
/* SimpleApp.java */  
import org.apache.spark.sql.Session;  
import org.apache.spark.sql.Dataset;  
  
public class SimpleApp {  
    public static void main(String[] args) {  
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system  
        Session spark = Session.builder().appName("Simple Application").getOrCreate();  
        Dataset<String> logData = spark.read().textFile(logFile).cache();  
  
        long numAs = logData.filter(s -> s.contains("a")).count();  
        long numBs = logData.filter(s -> s.contains("b")).count();  
  
        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);  
  
        spark.stop();  
    }  
}
```

# 同一个的例子 (Java)

```
/* SimpleApp.java */
import org.apache.spark.sql.Session;
import org.apache.spark.sql.Dataset;

public class SimpleApp {
    public static void main(String[] args) {
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
        Session spark = Session.builder().appName("Simple Application").getOrCreate();
        Dataset<String> logData = spark.read().textFile(logFile).cache();

        long numAs = logData.filter(s -> s.contains("a")).count();
        long numBs = logData.filter(s -> s.contains("b")).count();

        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);

        spark.stop();
    }
}
```

**Filter和count**

Java Lambda表达式，可以想象为一个匿名的函数，  
箭头左侧是函数的输入参数，箭头右侧是函数体，  
这里的函数体是{return s.contains("b");}的简写

# Lambda表达式==函数参数

## 使用Lambda表达式

```
JavaRDD<String> lines = sc.textFile("data.txt");  
JavaRDD<Integer> lineLengths = lines.map(s -> s.length());  
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

## 相当于定义了一个匿名函数

- 匿名函数：实现了Interface，其中有一个call() method

```
class GetLength implements Function<String, Integer> {  
    public Integer call(String s) { return s.length(); }  
}  
  
class Sum implements Function2<Integer, Integer, Integer> {  
    public Integer call(Integer a, Integer b) { return a + b; }  
}  
  
JavaRDD<String> lines = sc.textFile("data.txt");  
JavaRDD<Integer> lineLengths = lines.map(new GetLength());  
int totalLength = lineLengths.reduce(new Sum());
```

# RDD类型和运算（以Java为例）

- RDD的两种主要类型
- RDD的产生
- RDD的两类运算
  - transformation
  - action



# Java RDD的两种类型

- Class `JavaRDD<T>` 元素类型为T的RDD
- Class `JavaPairRDD<K,V>` 元素包含一个K和一个V

- 互相转换

- `JavaRDD<Tuple2<K,V>>` → `JavaPairRDD<K,V>`

- `JavaPairRDD.fromJavaRDD(rdd)`

- `JavaPairRDD<K,V>` → `JavaRDD<K>`

- `JavaPairRDD.keys()`,

- `JavaPairRDD<K,V>` → `JavaRDD<V>`

- `JavaPairRDD.values()`

# RDD产生方式1：从输入文件产生RDD

- sc是Class JavaSparkContext的一个对象

```
JavaRDD<String> distFile = sc.textFile("data.txt");
```

- 从文本文件data.txt读入数据
- 生成JavaRDD，每个元素是一行文本String

- 其它

- wholeTextFiles, sequenceFile, hadoopRDD, binaryFiles等

## RDD产生方式2：程序产生RDD

```
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);  
JavaRDD<Integer> distData = sc.parallelize(data);
```

用parallelize函数把一个list转换为JavaRDD

# RDD的两类运算

- Transformation

- 输入是RDD
- 输出是RDD

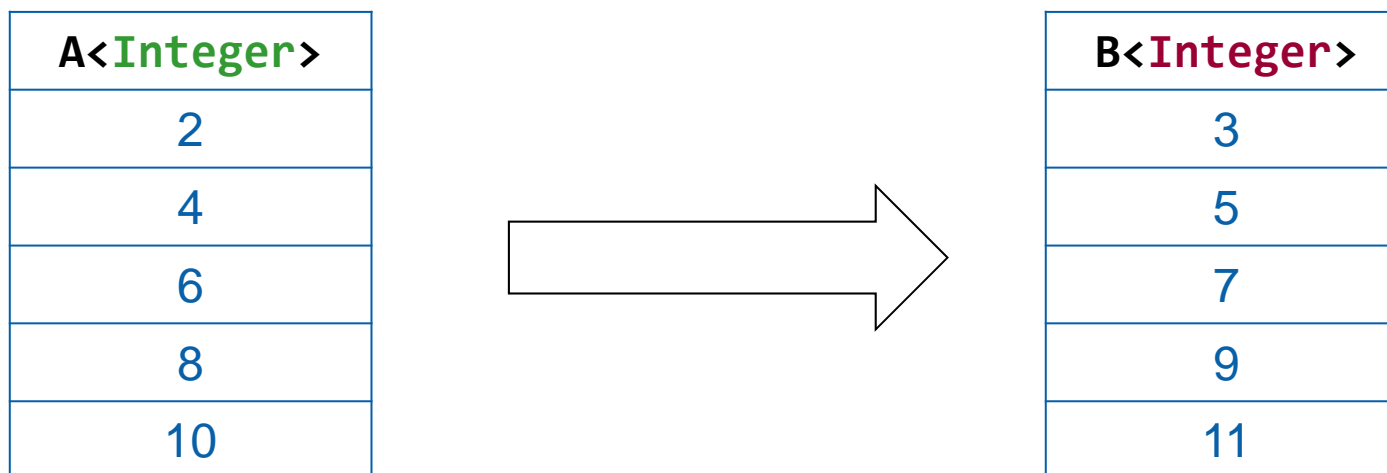
- Action

- 输入是RDD
- 输出是可以返回给driver程序的结果
- 输出不是分布式的数据集

# Transformation (1)

Transformation	涵义
<b>map</b> ( <i>func</i> )	<i>func</i> 是某种转换：一个输入元素 → 一个结果元素。 对RDD的每个元素调用 <i>func</i> 生成结果RDD。
<b>filter</b> ( <i>func</i> )	<i>func</i> 是一个过滤条件：一个输入元素 → True/False 对RDD的每个元素调用 <i>func</i> ，丢弃为False的元素， 形成结果RDD。
<b>flatMap</b> ( <i>func</i> )	<i>func</i> : 一个输入元素 → 一组元素（0个或多个） 对RDD的每个元素调用 <i>func</i> 生成结果RDD。 (类似MapReduce中的Map)
<b>union</b> ( <i>otherDataset</i> )	集合并
<b>intersection</b> ( <i>otherDataset</i> )	集合交
<b>distinct</b> ( )	去重

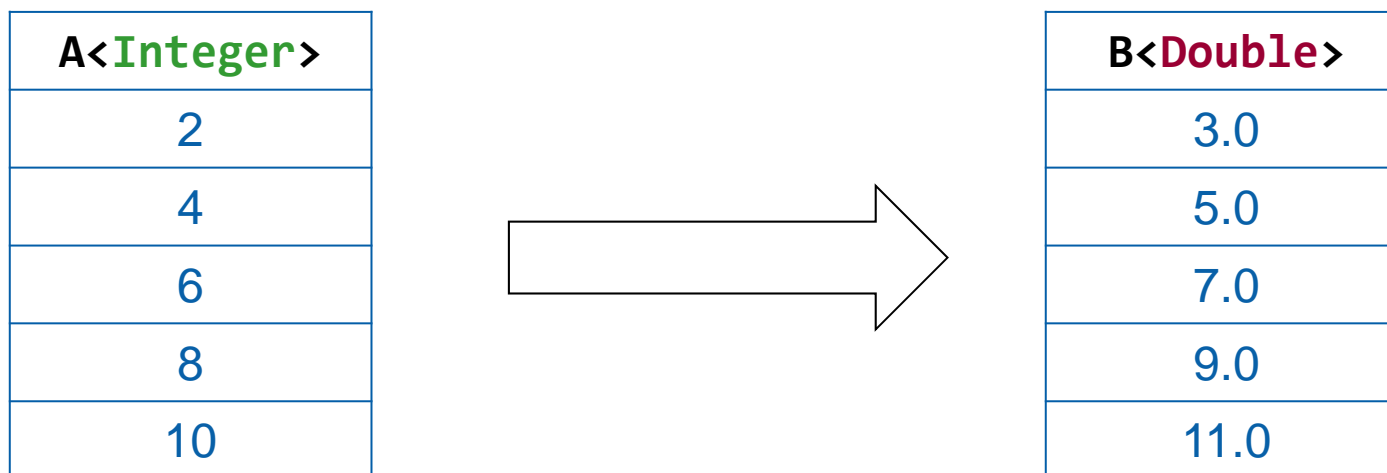
# Map举例



`B=A.map(x->x+1);`

注意：RDD的元素类型与Lambda表达式的关系

# Map举例

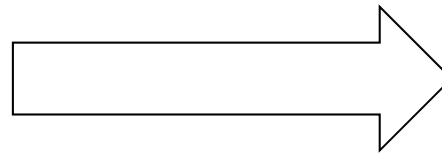


```
B=A.map(x->new Double(x+1));
```

注意：RDD的元素类型与Lambda表达式的关系

# flatMap举例

lines<String>
Hello world
Today is Wed
Hello Wed



words<String>
Hello
world
Today
is
Wed
Hello
Wed

```
words =  
  lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
```



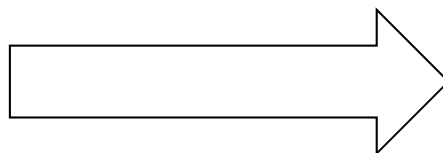
# Transformation (2)

Transformation	涵义
<b>groupByKey( )</b>	输入RDD(K, V), 返回结果RDD(K, Iterable<V>). 相同K的V放入了一个列表。
<b>reduceByKey(func)</b>	func把两个同类的值归并为一个值: (V, V)->V 输入RDD(K,V), 返回结果RDD(K,V),其中对于相同K的所有V都调用了func, 归并为一个V. (类似MapReduce中的Reduce)
<b>sortByKey([asc])</b>	对输入RDD(K,V)按照K进行排序, 可选参数asc为True则从小到大, False则从大到小排序。
<b>join(otherDataset)</b>	输入RDD1(K, V)和RDD2(K, W), 结果RDD为(K, (V, W)), 相同K进行等值连接。
<b>cogroup(otherDataset)</b>	输入RDD1(K, V)和RDD2(K, W), 结果RDD为(K, (Iterable<V>, Iterable<W>)) 把两个数据集中相同K的值放入两个表
<b>repartition(numPartitions)</b>	Reshuffle RDD数据产生numPartitions个划分。

# groupByKey 举例

ones<String, Integer>	
Hello	1
world	1
Today	1
is	1
Wed	1
Hello	1
Wed	1

ones.groupByKey()



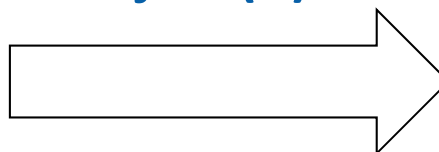
onelist<String, Iterable<Integer>>	
Hello	{1,1}
world	{1}
Today	{1}
is	{1}
Wed	{1,1}

# Join举例

A<Integer,String>	
2	"two"
4	"four"
6	"six"
8	"eight"
10	"ten"

C<Integer,Double>	
2	2.0
4	4.0
6	6.0
2	20
4	40

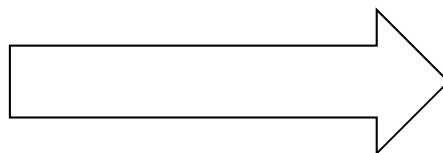
A.join(C)



B<Integer, Tuple2<String,Double>>		
2	("two", 2.0)	
2	("two", 20)	
4	("four", 4.0)	
4	("four", 40)	
6	("six", 6.0)	

# reduceByKey 举例

ones<String, Integer>
Hello,1
world,1
Today,1
is,1
Wed,1
Hello,1
Wed,1



counts<String, Integer>
Hello, 2
world, 1
Today, 1
is, 1
Wed, 2

```
counts = ones.reduceByKey((x, y) -> x+y);
```

# Action: 返回结果给main driver

Action	涵义
<b>reduce</b> ( <i>func</i> )	func把两个同类元素归并为一个元素 (T,T)->T 在输入RDD上调用func, 最后形成一个结果元素。
<b>collect</b> ()	把输入RDD转换为一个数组, 返回给main driver
<b>count</b> ()	返回RDD中元素个数
<b>first</b> ()	返回RDD中第一个元素 (基本等同于take(1)).
<b>take</b> ( <i>n</i> )	把RDD中前n个元素作为数组返回
<b>saveAsTextFile</b> ( <i>path</i> )	把RDD写成为一个文本文件
<b>saveAsSequenceFile</b> ( <i>path</i> )	把RDD写成一个Hadoop SequenceFile

# 让我们来看些例子

- Spark 2.4.1 自带的例子

- ❑ `examples/src/main/java/org/apache/spark/examples`

- Word Count

- PageRank

# WordCount

```
33 public static void main(String[] args) throws Exception {
34
35     if (args.length < 1) {
36         System.err.println("Usage: JavaWordCount <file>");
37         System.exit(1);
38     }
39
40     SparkSession spark = SparkSession
41         .builder()
42         .appName("JavaWordCount")
43         .getOrCreate();
44
45     JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
46
47     JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
48
49     JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
50
51     JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
52
53     List<Tuple2<String, Integer>> output = counts.collect();
54     for (Tuple2<?, ?> tuple : output) {
55         System.out.println(tuple._1() + ": " + tuple._2());
56     }
57     spark.stop();
58 }
```

主要部分

# WordCount

读文本文件生成一个JavaRDD,

```
45  JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
46
47  JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
48
49  JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
50
51  JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
52
53  List<Tuple2<String, Integer>> output = counts.collect();
```



# WordCount

```
45     JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
46
47     JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
48
49     JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
50
51     JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
52
53     List<Tuple2<String, Integer>> output = counts.collect();
```

**FlatMap:** 把每行文本切分为单词,  
**words**是所有单词组成的RDD

# WordCount

```
45      JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
46
47      JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
48
49      JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
50
51      JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
52
53      List<Tuple2<String, Integer>> output = counts.collect();
```

**mapToPair**类似**map**, 只不过生成K,V  
这里把每个**word**, 都变为 (word,1)

# WordCount

```
45     JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
46
47     JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
48
49     JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
50
51     JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
52
53     List<Tuple2<String, Integer>> output = counts.collect();
```

**reduceByKey**计算词频

# WordCount

```
45     JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
46
47     JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
48
49     JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
50
51     JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
52
53     List<Tuple2<String, Integer>> output = counts.collect();
```

**Collect**把计算结果作为数组返回

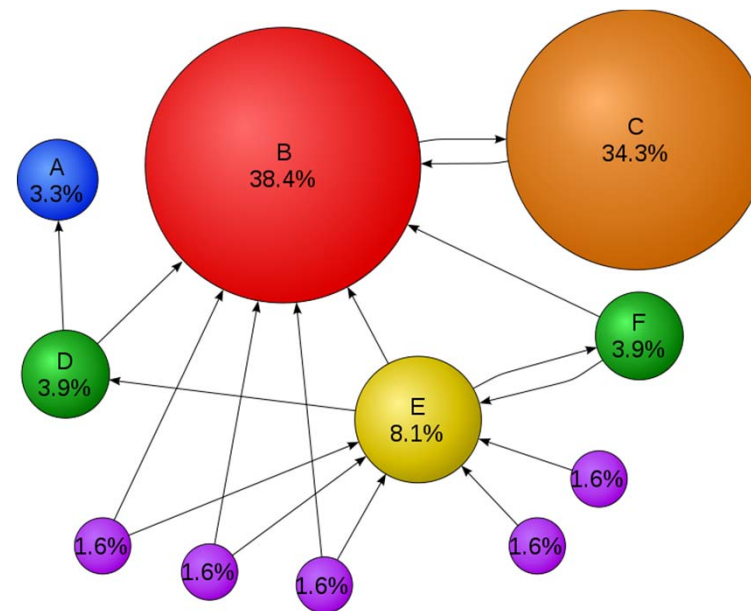
# PageRank

- $R_u = 0.15 + 0.85 \sum_{v \in B(u)} \frac{R_v}{L_v}$

- $R_v$ : 顶点v的PageRank\*N
- $L_v$ : 顶点v的出度（出边的条数）
- $B(u)$ : 顶点u的入邻居集合
- d: damping factor
- N: 总顶点个数

- 计算方法

- 初始化：所有的顶点的PageRank为1
- 迭代：用上述公式迭代直至收敛



图来源：Wikipedia

```

// Loads in input file. It should be in format of:
//      URL          neighbor URL
//      URL          neighbor URL
//      URL          neighbor URL
//      ...
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
    String[] parts = SPACES.split(s);
    return new Tuple2<>(parts[0], parts[1]);
}).distinct().groupByKey().cache();

// Loads all URLs with other URL(s) link to from input file and initialize ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);

// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(s -> {
            int urlCount = Iterables.size(s._1());
            List<Tuple2<String, Double>> results = new ArrayList<>();
            for (String n : s._1) {
                results.add(new Tuple2<>(n, s._2() / urlCount));
            }
            return results.iterator();
        });

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(new Sum()).mapValues(sum -> 0.15 + sum * 0.85);
}

```

初始化

循环

# PageRank初始化

读文本文件生成JavaRDD,  
每个元素是一行文本包括

URL起点

neighbor URL终点

```
// Loads in input file. It should be in format of:
//      URL      neighbor URL
//      URL      neighbor URL
//      URL      neighbor URL
//      ...
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
```

```
// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
    String[] parts = SPACES.split(s);
    return new Tuple2<>(parts[0], parts[1]);
}).distinct().groupByKey().cache();
```

```
// Loads all URLs with other URL(s) link to from input file and initialize ranks
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);
```

# PageRank初始化

```
// Loads in input file. It should be in format of:
```

```
//      URL          neighbor URL
```

```
//      URL          neighbor URL
```

```
//      URL          neighbor URL
```

```
//      ...
```

**mapToPair生成JavaPairRDD,**

**每个元素是边(起点, 终点)**

```
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
```

```
// Loads all URLs from input file and initialize their neighbors.
```

```
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
```

```
    String[] parts = SPACES.split(s);
```

```
    return new Tuple2<>(parts[0], parts[1]);
```

```
}).distinct().groupByKey().cache();
```

```
// Loads all URLs with other URL(s) link to from input file and initialize ranks
```

```
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);
```

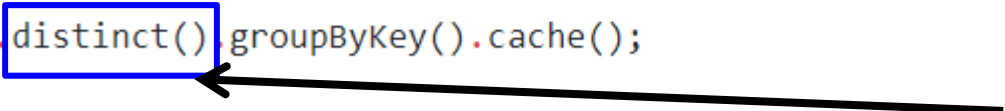


# PageRank初始化

```
// Loads in input file. It should be in format of:
//      URL          neighbor URL
//      URL          neighbor URL
//      URL          neighbor URL
//      ...
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
    String[] parts = SPACES.split(s);
    return new Tuple2<>(parts[0], parts[1]);
}).distinct().groupByKey().cache();

// Loads all URLs with other URL(s) link to from input file and initialize ranks
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);
```



去除相同的边

# PageRank初始化

```
// Loads in input file. It should be in format of:
//      URL          neighbor URL
//      URL          neighbor URL
//      URL          neighbor URL
//      ...
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
    String[] parts = SPACES.split(s);
    return new Tuple2<>(parts[0], parts[1]);
}).distinct().groupByKey().cache();
```

按照起点对边分组  
links <起点, Iterable<终点>>

```
// Loads all URLs with other URL(s) link to from input file and initialize ranks
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);
```

# PageRank初始化

```
// Loads in input file. It should be in format of:
//      URL          neighbor URL
//      URL          neighbor URL
//      URL          neighbor URL
//      ...
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
    String[] parts = SPACES.split(s);
    return new Tuple2<>(parts[0], parts[1]);
}).distinct().groupByKey().cache();

// Loads all URLs with other URL(s) link to from input file and initialize ranks
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);
```

ranks <起点, 1.0>

# PageRank循环

循环次数是程序输入参数

```
// Calculates and updates URL ranks continuously using PageRank algorithm.  
for (int current = 0; current < Integer.parseInt(args[1]); current++) {  
    // Calculates URL contributions to the rank of other URLs.  
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()  
        .flatMapToPair(s -> {  
            int urlCount = Iterables.size(s._1());  
            List<Tuple2<String, Double>> results = new ArrayList<>();  
            for (String n : s._1) {  
                results.add(new Tuple2<>(n, s._2() / urlCount));  
            }  
            return results.iterator();  
        });  
  
    // Re-calculates URL ranks based on neighbor contributions.  
    ranks = contribs.reduceByKey(a,b -> a+b).mapValues(sum -> 0.15 + sum * 0.85);  
}
```

# PageRank循环

links <起点, Iterable<终点> >

ranks<起点, rank>

join的结果是<起点, <Iterable<终点>, rank> >

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks) values()
        .flatMapToPair(s -> {
            int urlCount = Iterables.size(s._1());
            List<Tuple2<String, Double>> results = new ArrayList<>();
            for (String n : s._1) {
                results.add(new Tuple2<>(n, s._2() / urlCount));
            }
            return results.iterator();
        });

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(a,b -> a+b).mapValues(sum -> 0.15 + sum * 0.85);
}
```

# PageRank循环

links <起点, Iterable<终点> >

ranks<起点, rank>

join的结果是<起点, <Iterable<终点>, rank> >

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(s -> {
            int urlCount = Iterables.size(s._1());
            List<Tuple2<String, Double>> results = new ArrayList<>();
            for (String n : s._1) {
                results.add(new Tuple2<>(n, s._2() / urlCount));
            }
            return results.iterator();
        });

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(a,b -> a+b).mapValues(sum -> 0.15 + sum * 0.85);
}
```

value部分是<Iterable<终点>, rank>

# PageRank循环

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
```

```
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
```

```
    // Calculates URL contributions to the rank of other URLs.
```

```
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
```

```
    .flatMapToPair(s -> {
```

```
        int urlCount = Iterables.size(s._1());
```

```
        List<Tuple2<String, Double>> results = new ArrayList<>();
```

```
        for (String n : s._1) {
```

```
            results.add(new Tuple2<>(n, s._2() / urlCount));
```

```
        }
```

```
        return results.iterator();
```

```
    });
```

s是<Iterable<终点>, rank>

s.\_1()是Iterable<终点>

urlCount是出度

s.\_2()/urlCount是rank/出度

结果是contribs<终点, rank/出度>

```
// Re-calculates URL ranks based on neighbor contributions.
```

```
ranks = contribs.reduceByKey(a,b -> a+b).mapValues(sum -> 0.15 + sum * 0.85);
```

```
}
```

# PageRank循环

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(s -> {
            int urlCount = Iterables.size(s._1());
            List<Tuple2<String, Double>> results = new ArrayList<>();
            for (String n : s._1) {
                results.add(new Tuple2<>(n, s._2() / urlCount));
            }
            return results.iterator();
        });

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(a,b -> a+b).mapValues(sum -> 0.15 + sum * 0.85);
}
```

**Contribs <终点,  $\text{rank}_{\text{src}} / \text{出度}$ >**  
**reduceByKey, 得到<终点,  $\sum \frac{\text{rank}_{\text{src}}}{\text{出度}_{\text{src}}}$ >**



# PageRank循环

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
```

```
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
```

```
    // Calculates URL contributions to the rank of other URLs.
```

```
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
```

```
        .flatMapToPair(s -> {
```

```
            int urlCount = Iterables.size(s._1());
```

```
            List<Tuple2<String, Double>> results = new ArrayList<>();
```

```
            for (String n : s._1) {
```

```
                results.add(new Tuple2<>(n, s._2() / urlCount));
```

```
            }
```

```
            return results.iterator();
```

```
        });
```

**mapValues是key不变，在value上调用func**

**得到<终点,  $0.15 + \text{sum} * 0.85$ >为新的rank**

```
// Re-calculates URL ranks based on neighbor contributions.
```

```
ranks = contribs.reduceByKey(a,b -> a+b).mapValues(sum -> 0.15 + sum * 0.85);
```

```
}
```

# PageRank循环

下一次循环

links <起点, Iterable<终点>> 不变

ranks<起点, rank> 是上次循环产生的结果

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
    // Calculates URL contributions to the rank of other URLs.
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
        .flatMapToPair(s -> {
            int urlCount = Iterables.size(s._1());
            List<Tuple2<String, Double>> results = new ArrayList<>();
            for (String n : s._1) {
                results.add(new Tuple2<>(n, s._2() / urlCount));
            }
            return results.iterator();
        });

    // Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(a,b -> a+b).mapValues(sum -> 0.15 + sum * 0.85);
}
```

```
// Loads in input file. It should be in format of:
```

```
//      URL          neighbor URL
```

```
//      URL          neighbor URL
```

```
//      URL          neighbor URL
```

```
//      ...
```

```
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
```

```
// Loads all URLs from input file and initialize their neighbors.
```

```
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(s -> {
```

```
    String[] parts = SPACES.split(s);
```

```
    return new Tuple2<>(parts[0], parts[1]);
```

```
}).distinct().groupByKey().cache();
```

```
// Loads all URLs with other URL(s) link to from input file and initialize ranks of them to one.
```

```
JavaPairRDD<String, Double> ranks = links.mapValues(rs -> 1.0);
```

```
// Calculates and updates URL ranks continuously using PageRank algorithm.
```

```
for (int current = 0; current < Integer.parseInt(args[1]); current++) {
```

```
    // Calculates URL contributions to the rank of other URLs.
```

```
    JavaPairRDD<String, Double> contribs = links.join(ranks).values()
```

```
        .flatMapToPair(s -> {
```

```
            int urlCount = Iterables.size(s._1());
```

```
            List<Tuple2<String, Double>> results = new ArrayList<>();
```

```
            for (String n : s._1) {
```

```
                results.add(new Tuple2<>(n, s._2() / urlCount));
```

```
            }
```

```
            return results.iterator();
```

```
        });
```

```
// Re-calculates URL ranks based on neighbor contributions.
```

```
ranks = contribs.reduceByKey(new Sum()).mapValues(sum -> 0.15 + sum * 0.85);
```

```
}
```

总体看一下

初始化

循环

# Spark

- 原理
- 编程
- 系统实现
- Spark SQL
- Spark Streaming

# RDD vs. Distributed Shared Memory

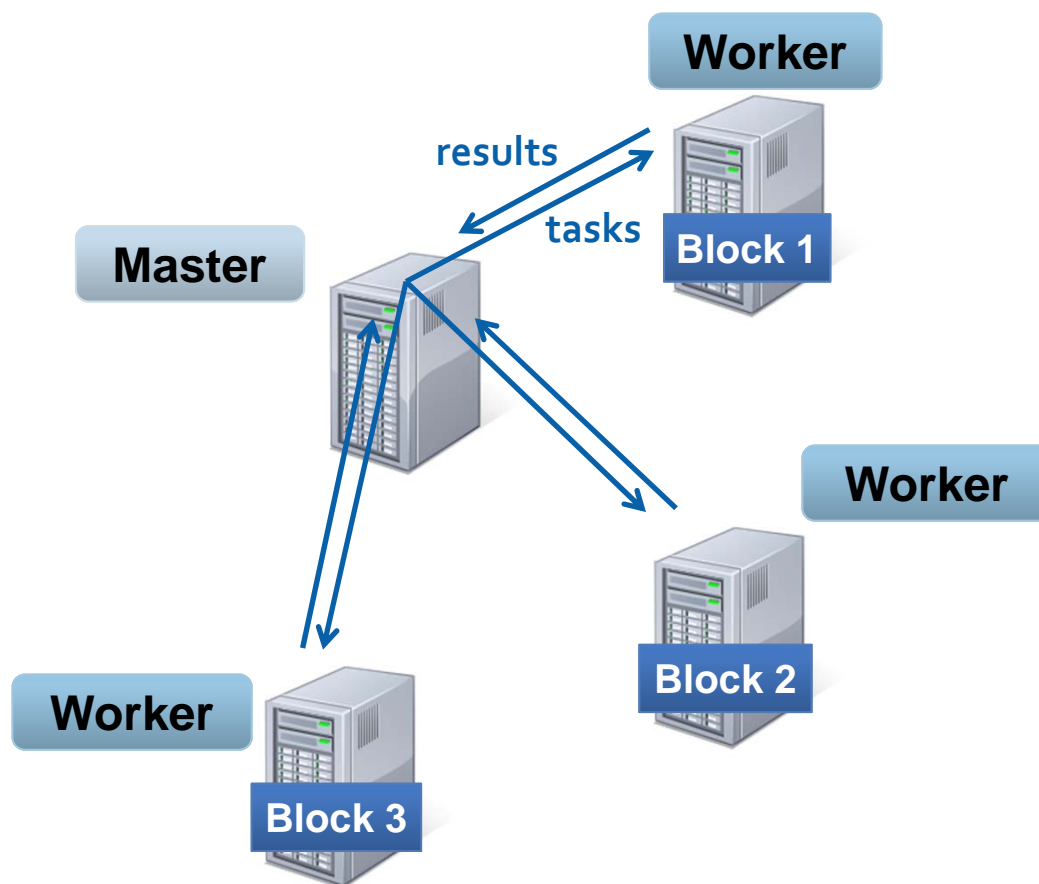
Aspect	RDDs	Distr. Shared Mem.
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks	Difficult
Work placement	Automatic based on data locality	Up to app (runtimes aim for transparency)
Behavior if not enough RAM	Similar to existing data flow systems	Poor performance (swapping?)

图来源：NSDI'12 paper

# 运算过程

读入内存一次

在内存中可以多次处理



图来源：NSDI'12 slides

# Spark运算的运行

- Transformation

- 仅记录，不运算
- Lazy execution

- Action

- 当遇到Action时，需要返回结果，才真正执行已经记录的前面的运算

- 容错/内存缓冲替换：当内存缓冲的RDD丢失时

- 可以重新执行记录的运算，重新计算这个RDD

# Spark中RDD存储方式

- 用`preserve()`函数指定当前RDD的存储方式
  - ❑ `MEMORY_ONLY`: 内存缓冲Java对象
  - ❑ `MEMORY_ONLY_SER`: 内存serialized(顺序化)
  - ❑ `MEMORY_AND_DISK`: `MEMORY_ONLY`+内存满了放入外存
  - ❑ `DISK_ONLY`: 放入外存
  - ❑ `MEMORY_ONLY_2`: `MEMORY_ONLY`在两个节点保留副本
  - ❑ `OFF_HEAP (experimental)`
- ❑ `Cache()`是`preserve(StorageLevel.MEMORY_ONLY)`的简写



# Spark

- 原理
- 编程
- 系统实现
- Spark SQL
- Spark Streaming

# Spark SQL: DataFrame

- DataFrame

- 可以看做是在RDD上定义了Relational Schema
- 列可以命名访问

- 可以从文件读取生成

```
Dataset<Row> df = spark.read().load("文件路径");
```

Spark可以支持json, parquet, jdbc, orc, csv, text等文件格式  
当然实质上文件必须是关系型数据

# DataFrame显示Schema和数据

```
df.printSchema();  
// root  
// |-- age: long (nullable = true)  
// |-- name: string (nullable = true)
```

```
df.show();  
// +-----+-----+  
// | age|    name|  
// +-----+-----+  
// |null|Michael|  
// | 30|    Andy|  
// | 19|  Justin|  
// +-----+-----+
```

关系型表有**2列**: age, name  
共有**3行**记录

# DataFrame执行SQL操作

- 投影一列

```
df.select("name").show();  
// +-----+  
// |    name|  
// +-----+  
// |Michael|  
// |   Andy|  
// |  Justin|  
// +-----+
```

```
df.show();  
// +-----+-----+  
// | age|    name|  
// +-----+-----+  
// |null|Michael|  
// |  30|   Andy|  
// |  19|  Justin|  
// +-----+-----+
```

# DataFrame执行SQL操作

- 投影2列 (age列+1)

```
df.show();  
// +-----+-----+  
// | age|    name|  
// +-----+-----+  
// |null|Michael|  
// | 30|    Andy|  
// | 19|   Justin|  
// +-----+-----+
```

```
df.select(col("name"), col("age").plus(1)).show();  
// +-----+-----+  
// |    name|(age + 1)|  
// +-----+-----+  
// |Michael|      null|  
// |   Andy|       31|  
// | Justin|       20|  
// +-----+-----+
```

# DataFrame执行SQL操作

- 选择条件

```
df.filter(col("age").gt(21)).show();  
// +---+-----+  
// |age|name|  
// +---+-----+  
// | 30|Andy|  
// +---+-----+
```

```
df.show();  
// +---+-----+  
// |age|name|  
// +---+-----+  
// |null|Michael|  
// | 30|Andy|  
// | 19|Justin|  
// +---+-----+
```

# DataFrame执行SQL操作

- Group by + Aggregation

```
df.groupBy("age").count().show();  
// +-----+-----+  
// | age|count|  
// +-----+-----+  
// | 19|    1|  
// |null|    1|  
// | 30|    1|  
// +-----+-----+
```

```
df.show();  
// +-----+-----+  
// | age|    name|  
// +-----+-----+  
// |null|Michael|  
// | 30|    Andy|  
// | 19|   Justin|  
// +-----+-----+
```

# DataFrame执行SQL操作

- SQL语句

- 先要对表命名

```
df.createOrReplaceTempView("people");
```

```
Dataset<Row> sqlDF = spark.sql("SELECT * FROM people");
```

```
sqlDF.show();
```

```
// +-----+-----+
// | age|    name|
// +-----+-----+
// |null|Michael|
// | 30|    Andy|
// | 19|   Justin|
// +-----+-----+
```

```
df.show();
// +-----+-----+
// | age|    name|
// +-----+-----+
// |null|Michael|
// | 30|    Andy|
// | 19|   Justin|
// +-----+-----+
```



# Spark

- 原理
- 编程
- 系统实现
- Spark SQL
- Spark Streaming

# Spark Streaming



- 把输入的数据流转化为一个个minibatch
- 然后在minibatch上运行计算

# 例子：JavaNetworkWordCount.java

- 对流入的数据进行Word Count
- 先创建一个JavaStreamingContext
  - 每个Minibatch代表1秒钟时间片的数据

```
JavaStreamingContext jssc = new JavaStreamingContext(conf, Durations.seconds(1));
```

- 添加数据源：网络地址+端口号

```
JavaReceiverInputDStream<String> lines = jssc.socketTextStream("localhost", 9999);
```

- 计算Word Count：这部分与普通Spark程序基本一致

```
JavaDStream<String> words = lines.flatMap(x -> Arrays.asList(x.split(" ")).iterator());  
JavaPairDStream<String, Integer> pairs = words.mapToPair(s -> new Tuple2<>(s, 1));  
JavaPairDStream<String, Integer> wordCounts = pairs.reduceByKey((i1, i2) -> i1 + i2);  
wordCounts.print();
```

# 大数据管理系统

