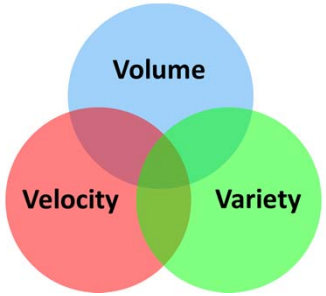
大数据系统与大规模数据分析

大数据运算系统（5）



陈世敏

中科院计算所
计算机体系结构
国家重点实验室

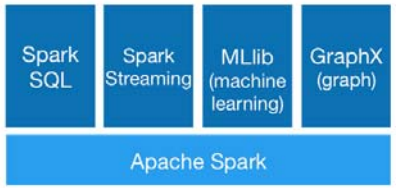
Outline

- 内存计算
  - 内存数据库
  - 内存键值系统
  - 内存MapReduce
    - Spark
    - Cloudera Impala

Spark

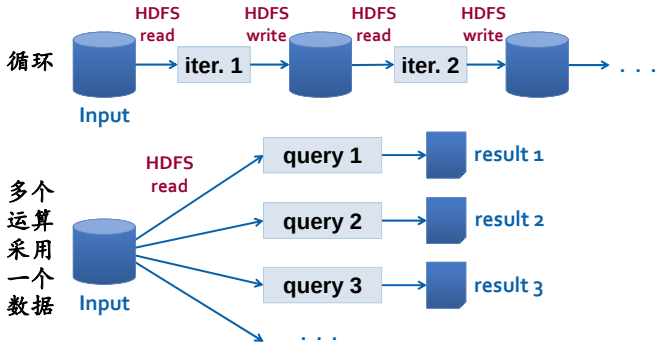
- 原理
- 编程
- 系统实现

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



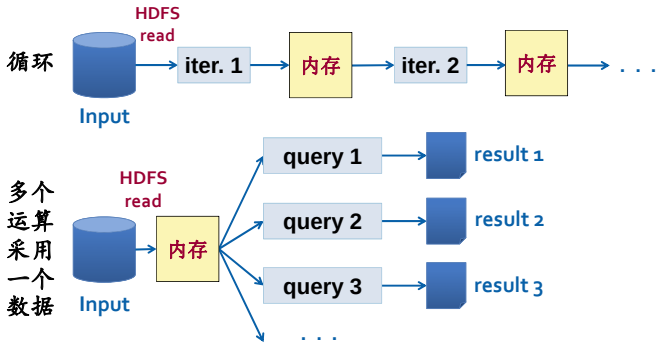
- 通过HDFS进行作业间数据共享，代价太高

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

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

两类RDD运算

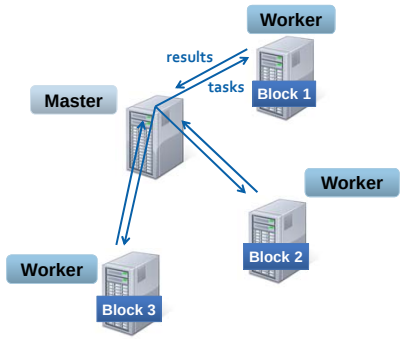
- Transformation
  - 输入是RDD(数据集)
  - 输出也是RDD(数据集)
  - RDD → RDD
- Action
  - 输入是RDD(数据集)
  - 输出是某种计算结果（例如，一个数值或者一列数值）
    - 注意：RDD可能非常大，但是计算结果总是比较小的
  - RDD → 计算结果

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

- 原理
- 编程
- 系统实现

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

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

例子来源：Spark Manual

简单的例子（Scala）

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

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

简单的例子（Scala）

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

主程序，实际上是driver，发出Spark操作请求

简单的例子（Scala）

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

建立SparkContext

简单的例子（Scala）

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

读文本文件生成一个RDD

简单的例子（Scala）

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

RDD操作

简单的例子（Scala）

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

RDD filter对每个元素，调用给定函数，如果True保留，False丢弃

简单的例子（Scala）

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

RDD count 有多少元素

简单的例子（Scala）

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

RDD操作

同一个的例子（Java）

```java
import org.apache.spark.api.java.*;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.Function;

public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
    SparkConf conf = new SparkConf().setAppName("Simple Application");
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

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

Java import

例子来源：Spark Manual

同一个的例子（Java）

```java
public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
    SparkConf conf = new SparkConf().setAppName("Simple Application");
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

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

主程序，
实际上是driver，
发出Spark操作请求

同一个的例子（Java）

```
public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
    SparkConf conf = new SparkConf().setAppName("Simple Application");
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

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

建立SparkContext

同一个的例子（Java）

```
public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
    SparkConf conf = new SparkConf().setAppName("Simple Application");
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

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

读文本文件生成一个JavaRDD

同一个的例子（Java）

```
public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
    SparkConf conf = new SparkConf().setAppName("Simple Application");
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

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

RDD操作

看一下filter的参数：

- 一个类的对象：这里是匿名的类，实现了spark.api.java.function
- 包含call函数
- call 的参数是logData这个JavaRDD的元素类型，即String
- call 的返回值是Boolean
- 将对logData的每个元素调用call一次，true 保留，false 丢弃

同一个的例子（Java）

```
public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
    SparkConf conf = new SparkConf().setAppName("Simple Application");
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

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

另一个filter, count

以Java为例讲一下

- RDD输入
- RDD transformation
- RDD action

Java RDD的类型

- 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
- 转换
  - JavaRDD ➔JavaPairRDD
    - JavaPairRDD.fromJavaRDD(rdd)
    - 要求rdd的每个元素是Tuple2<K,V>类型
  - JavaPairRDD ➔ JavaRDD
    - JavaPairRDD.keys(), JavaPairRDD.values()
    - 可以把K和V部分分别形成一个JavaRDD

从输入文件产生RDD

- Class JavaSparkContext中函数
- JavaSparkContext. textFile(path)
  - 从文本文件读入，每一行是一个元素，元素类型为String
  - 返回JavaRDD<String>
- JavaSparkContext.wholeTextFiles(path)
  - 读一个目录，每个文件成为一个元素
  - key是文件路径，value是文件内容
  - 返回JavaPairRDD<String,String>
- 其它
  - sequenceFile, hadoopRDD, binearyFiles, 等

程序产生RDD

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

Class JavaSparkContext中parallelize函数

可以根据Java程序中的List产生JavaRDD

RDD运算

map(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]	**Transformation**
filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]	
flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]	
sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)	
groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]	
reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]	
union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]	
join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]	
cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]	
crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]	
mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)	
sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]	
partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]	
count()	:	RDD[T] ⇒ Long	**Action**
collect()	:	RDD[T] ⇒ Seq[T]	
reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T	
lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)	
save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS	

图来源：NSDI'12 paper

RDD Transformation

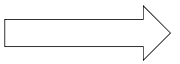
- 运算的结果是新的RDD
  - 有2类
  - 通用的：在JavaRDD和JavaPairRDD上都可以运算
  - 仅在JavaPairRDD上的运算：需要Key

Transformation

- JavaRDD<T> A;
- Map：一对一映射
  - B=A.map(f)
    - 对A的每个元素调用f.call，其返回值成为B的一个元素
    - 这样形成的RDD就是B
  - JavaRDD<R> **map**(Function<T,R> f)
  - Function interface中仅定义了一个函数
    - R call(T v)

Map举例

A<Integer>
2
4
6
8
10



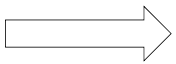
B<Integer>
3
5
7
9
11

```
B=A.map(new Function<Integer, Integer>() {
   @override
   public Integer call (Integer x) {
      return x+1;
   }
});
```

注意：RDD的元素类型

Map举例

A<Integer>
2
4
6
8
10



B<Double>
3.0
5.0
7.0
9.0
11.0

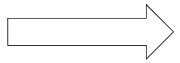
```
B=A.map(new Function<Integer, Double>() {
   @override
   public Double call (Integer x) {
      return new Double(x+1);
   }
});
```

注意：RDD的元素类型

Transformation

- MapToPair
  - JavaPairRDD<K2,V2>**mapToPair**(PairFunction<T,K2,V2> f)
  - B= A.mapToPair(f)

A<Integer>
2
4
6
8
10



B<Integer, Double>
3, 3.0
5, 5.0
7, 7.0
9, 9.0
11, 11.0

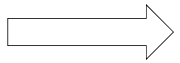
```
B= A.mapToPair(new PairFunction<Integer, Integer, Double>() {
   @override
   public Tuple2<Integer,Double> call (Integer x) {
      return new Tuple2<Integer,Double>(x+1, x+1);
   }
});
```


Transformation

- Filter：过滤
  - JavaRDD<T> **filter**(Function<T,Boolean> f)
  - B=A.filter(f)
  - 对输入RDD的每个元素调用f.call
    - 如果返回true，那么此元素成为输出RDD的一个元素
    - 如果返回false，那么就把这个元素丢弃

Filter 举例

A<Integer>
2
4
6
8
10



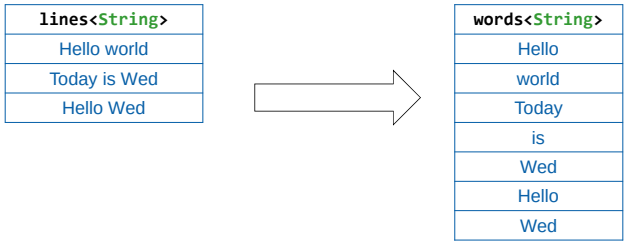
B<Integer>
8
10

```
B=A.filter(new Function<Integer,Boolean>() {
   @override
   public Boolean call (Integer x) {
      return (x>6);
   }
});
```

Transformation

- FlatMap：一对多映射
  - JavaRDD<R> **flatMap**(FlatMapFunction<T,R> f)
  - B=A.flatMap(f)
  - f.call有这样的定义：Iterable<R> call(T v)
  - 对输入RDD的每个元素调用f.call，其返回值的每个R都成为输出RDD的一个元素
  - 类似于MapReduce中的Map

flatMap举例




```
JavaRDD<String> words = lines.flatMap(
  new FlatMapFunction<String, String>() {
  @Override
  public Iterable<String> call(String s) {
    return Arrays.asList(SPACE.split(s));
  }
});
```


Transformation

- Sample：采样
  - JavaRDD<T> sample(boolean withReplacement, double fraction)
  - B=A.sample(true, 0.1)
  - 对输入RDD进行采样，采样的结果放入输出RDD
- Union：并集
- Intersection：交集
- Distinct: 去掉重复元素

JavaPairRDD Transformation

- groupByKey
  - JavaPairRDD<K,Iterable<V>> **groupByKey**()
  - B=A.groupByKey()
  - 把输入JavaPairRDD中，相同的Key的元素group by，所有的同组的Value放入Iterable，形成输出JavaPairRDD

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}

JavaPairRDD Transformation

- Join
  - JavaPairRDD<K, Tuple2<V,W>> **join**(JavaPairRDD<K,W> C)
  - B=A.join(C)
  - 把Key相同的A和C的元素Join在一起，把A的value和C的value形成一个Tuple2结构，成为输出JavaPairRDD的value

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)

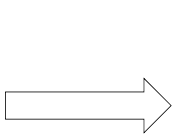
B<Integer, Tuple2<String,Double>>
2, (“two”, 2.0)
2, (“two”, 20)
4, (“four”, 4.0)
4, (“four”, 40)
6, (“six”, 6.0)

JavaPairRDD Transformation

- reduceByKey
  - JavaPairRDD<K,V> **reduceByKey**(Function2<V,V,V> f)
  - B=A.reduceByKey(f)
  - f.call有这样的定义：T3 call(T1 v1, T2 v2)
  - 把输入JavaPairRDD中，相同的Key的元素，group by，所有的同组的Value放入调用f.call，生成一个元素，形成输出JavaPairRDD
  - 与MapReduce中的Reduce相似

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

```
JavaPairRDD<String, Integer> counts = ones.reduceByKey(
  new Function2<Integer, Integer, Integer>() {
  @Override
  public Integer call(Integer i1, Integer i2) {
    return i1 + i2;
  }
});
```


JavaPairRDD Transformation

- 其它
  - cartesian
  - aggregateByKey
  - sortByKey

Action

- 在RDD上的运算结果，而不是RDD
- Reduce
  - T reduce(Function2<T,T,T> f)
  - f.call有这样的定义：T3 call(T1 v1, T2 v2)
  - 输入RDD的所有元素都调用f.call，生成一个值返回
- Collect：返回List
  - List<T> **collect**()
  - 与parallelize功能相反，把RDD转换为List返回
- Count: 元素数
- 其它
  - saveAsTextFile，saveAsSequenceFile等
  - countByKey, lookup等

让我们来看些例子

- Spark自带的例子
  - examples/src/main/java/org/apache/spark/examples
- Word Count
- PageRank

```
public final class JavaWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");

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

    if (args.length < 1) {
      System.err.println("Usage: JavaWordCount <file>");
      System.exit(1);
    }

    SparkConf sparkConf = new SparkConf().setAppName("JavaWordCount");
    JavaSparkContext ctx = new JavaSparkContext(sparkConf);
    JavaRDD<String> lines = ctx.textFile(args[0], 1);

    JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String, String>() {
      @Override
      public Iterable<String> call(String s) {
        return Arrays.asList(SPACE.split(s));
      }
    });

    JavaPairRDD<String, Integer> ones = words.mapToPair(new PairFunction<String, String, Integer>() {
      @Override
      public Tuple2<String, Integer> call(String s) {
        return new Tuple2<String, Integer>(s, 1);
      }
    });

    JavaPairRDD<String, Integer> counts = ones.reduceByKey(new Function2<Integer, Integer, Integer>() {
      @Override
      public Integer call(Integer i1, Integer i2) {
        return i1 + i2;
      }
    });

    List<Tuple2<String, Integer>> output = counts.collect();
    for (Tuple2<?,?> tuple : output) {
      System.out.println(tuple._1() + ": " + tuple._2());
    }
    ctx.stop();
  }
}
```

主要部分

例子来源：Spark Code

Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

JavaRDD<String> words = lines.flatMap(
  new FlatMapFunction<String, String>() {
  @Override
  public Iterable<String> call(String s) {
    return Arrays.asList(SPACE.split(s));
  }
});

JavaPairRDD<String, Integer> ones = words.mapToPair(
  new PairFunction<String, String, Integer>() {
  @Override
  public Tuple2<String, Integer> call(String s) {
    return new Tuple2<String, Integer>(s, 1);
  }
});

JavaPairRDD<String, Integer> counts = ones.reduceByKey(
  new Function2<Integer, Integer, Integer>() {
  @Override
  public Integer call(Integer i1, Integer i2) {
    return i1 + i2;
  }
});
```

读文本文件
生成一个
JavaRDD,
1是partition个数

Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

JavaRDD<String> words = lines.flatMap(
  new FlatMapFunction<String, String>() {
  @Override
  public Iterable<String> call(String s) {
    return Arrays.asList(SPACE.split(s));
  }
});

JavaPairRDD<String, Integer> ones = words.mapToPair(
  new PairFunction<String, String, Integer>() {
  @Override
  public Tuple2<String, Integer> call(String s) {
    return new Tuple2<String, Integer>(s, 1);
  }
});

JavaPairRDD<String, Integer> counts = ones.reduceByKey(
  new Function2<Integer, Integer, Integer>() {
  @Override
  public Integer call(Integer i1, Integer i2) {
    return i1 + i2;
  }
});
```

FlatMap
words是所有单
词组成的RDD

Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

JavaRDD<String> words = lines.flatMap(
  new FlatMapFunction<String, String>() {
  @Override
  public Iterable<String> call(String s) {
    return Arrays.asList(SPACE.split(s));
  }
});

JavaPairRDD<String, Integer> ones = words.mapToPair(
  new PairFunction<String, String, Integer>() {
  @Override
  public Tuple2<String, Integer> call(String s) {
    return new Tuple2<String, Integer>(s, 1);
  }
});

JavaPairRDD<String, Integer> counts = ones.reduceByKey(
  new Function2<Integer, Integer, Integer>() {
  @Override
  public Integer call(Integer i1, Integer i2) {
    return i1 + i2;
  }
});
```

mapToPair
类似map, 生成K,V

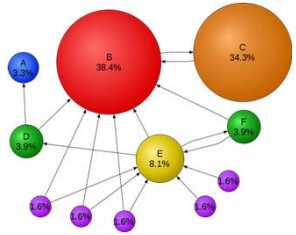
Word Count

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

JavaRDD<String> words = lines.flatMap(
  new FlatMapFunction<String, String>() {
  @Override
  public Iterable<String> call(String s) {
    return Arrays.asList(SPACE.split(s));
  }
});

JavaPairRDD<String, Integer> ones = words.mapToPair(
  new PairFunction<String, String, Integer>() {
  @Override
  public Tuple2<String, Integer> call(String s) {
    return new Tuple2<String, Integer>(s, 1);
  }
});

JavaPairRDD<String, Integer> counts = ones.reduceByKey(
  new Function2<Integer, Integer, Integer>() {
  @Override
  public Integer call(Integer i1, Integer i2) {
    return i1 + i2;
  }
});
```

reduceByKey

PageRank

- $R_u = 1 - d + d\sum_{v \in B(u)} \frac{R_v}{L_v}$
  - 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



图来源：Wikipedia

- 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

Page Rank

```
JavaRDD<String> lines = ctx.textFile(args[0], 1);

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

// Loads all URLs with other URL(s) link to from input file
// and initializes ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
  new Function<Iterable<String>, Double>() {
  @Override
  public Double call(Iterable<String> rs) {
    return 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(new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
      @Override
      public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
        int urlCount = Iterables.size(s._1);
        List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
        for (String d : s._1) {
          results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
        }
        return results;
      }
    });

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```

初始化

循环

例子来源：Spark Code

Page Rank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 = ctx.textFile(args[0], 1);
```

读文本文件生成一个JavaRDD，每行一个元素

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

// Loads all URLs with other URL(s) link to from input file
// and initializes ranks of them to one.
JavaPairRDD<String, Double> ranks = links.mapValues(
  new Function<Iterable<String>, Double>() {
  @Override
  public Double call(Iterable<String> rs) {
    return 1.0;
  }
});
```

Page Rank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 = ctx.textFile(args[0], 1);

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
  new PairFunction<String, String, String>() {
  @Override
  public Tuple2<String, String> call(String s) {
    String[] parts = SPACES.split(s);
    return new Tuple2<String, String>(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(
  new Function<Iterable<String>, Double>() {
  @Override
  public Double call(Iterable<String> rs) {
    return 1.0;
  }
});
```

mapToPair提取(URL, neighbor URL)为一个JavaPairRDD

Page Rank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 = ctx.textFile(args[0], 1);

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
  new PairFunction<String, String, String>() {
  @Override
  public Tuple2<String, String> call(String s) {
    String[] parts = SPACES.split(s);
    return new Tuple2<String, String>(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(
  new Function<Iterable<String>, Double>() {
  @Override
  public Double call(Iterable<String> rs) {
    return 1.0;
  }
});
```

去除相同的边

Page Rank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 = ctx.textFile(args[0], 1);

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
  new PairFunction<String, String, String>() {
  @Override
  public Tuple2<String, String> call(String s) {
    String[] parts = SPACES.split(s);
    return new Tuple2<String, String>(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(
  new Function<Iterable<String>, Double>() {
  @Override
  public Double call(Iterable<String> rs) {
    return 1.0;
  }
});
```

按照起点src对边分组
links <src, Iterable<dest>>

Page Rank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 = ctx.textFile(args[0], 1);

// Loads all URLs from input file and initialize their neighbors.
JavaPairRDD<String, Iterable<String>> links = lines.mapToPair(
  new PairFunction<String, String, String>() {
  @Override
  public Tuple2<String, String> call(String s) {
    String[] parts = SPACES.split(s);
    return new Tuple2<String, String>(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(
  new Function<Iterable<String>, Double>() {
  @Override
  public Double call(Iterable<String> rs) {
    return 1.0;
  }
});
```

按照起点src对边分组
links <src, Iterable<dest>>

产生初始化的Rank：把links的value部分代替成为1.0，
ranks<src, rank=1.0>

Page Rank循环

循环次数是输入确定的

循环是driver程序控制的

```
// 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(
     new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
      @Override
      public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
        int urlCount = Iterables.size(s._1);
        List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
        for (String d : s._1) {
          results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
        }
        return results;
      }
  });

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```


Page Rank循环

links <src, Iterable<dest>>, ranks<src, rank>
join的结果的value部分是<Iterable<dest>, rank>

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

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```


Page Rank循环

links <src, Iterable<dest>>, ranks<src, rank>;
输入<Iterable<dest>, rank>

输出contribs=<dest, rank/n>

```
// Calculates and updates URL ranks continuously using P
for (int current = 0; current < Integer.parseInt(args[1]);
  // Calculates URL contributions to the rank of other U
  JavaPairRDD<String, Double> contribs = links.join(ranks).values()
    .flatMapToPair(
     new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
      @Override
      public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
        int urlCount = Iterables.size(s._1);
        List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
        for (String d : s._1) {
          results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
        }
        return results;
      }
  });

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```


Page Rank循环

links <src, Iterable<dest>>, ranks<src, rank>;
输入<Iterable<dest>, rank>

输出contribs=<dest, rank/n>

```
// Calculates and updates URL ranks continuously using P
for (int current = 0; current < Integer.parseInt(args[1]);
  // Calculates URL contributions to the rank of other U
  JavaPairRDD<String, Double> contribs = links.join(ranks).values()
    .flatMapToPair(
     new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
      @Override
      public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
        int urlCount = Iterables.size(s._1);
        List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
        for (String d : s._1) {
          results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
        }
        return results;
      }
  });

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```

Page Rank循环

links <src, Iterable<dest>>, ranks<src, rank>;
输入<Iterable<dest>, rank>
输出contribs=<dest, rank/n>

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

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```


Page Rank循环

links <src, Iterable<dest>>, ranks<src, rank>;
输入<Iterable<dest>, rank>
输出contribs=<dest, rank/n>

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

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```

S._1: Iterable<dest>
S._2: rank

Page Rank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(
      new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
       @Override
       public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
         int urlCount = Iterables.size(s._1);
         List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
         for (String d : s._1) {
           results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
         }
         return results;
       }
    });

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```

Contribs <dest, $rank_{src}$/n>
reduceByKey，得到<dest, $\sum \frac{rank_{src}}{n_{src}}$>

Page Rank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(
      new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
       @Override
       public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
         int urlCount = Iterables.size(s._1);
         List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
         for (String d : s._1) {
           results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
         }
         return results;
       }
    });

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```

得到<dest, 0.15+sum*0.85>为新的ranks

Page Rank循环

每次循环都生成新的contribs和ranks

```
// 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(
      new PairFlatMapFunction<Tuple2<Iterable<String>, Double>, String, Double>() {
      @Override
      public Iterable<Tuple2<String, Double>> call(Tuple2<Iterable<String>, Double> s) {
        int urlCount = Iterables.size(s._1);
        List<Tuple2<String, Double>> results = new ArrayList<Tuple2<String, Double>>();
        for (String d : s._1) {
          results.add(new Tuple2<String, Double>(d, s._2() / urlCount));
        }
        return results;
      }
    });

  // Re-calculates URL ranks based on neighbor contributions.
  ranks = contribs.reduceByKey(new Sum()).mapValues(new Function<Double, Double>() {
    @Override
    public Double call(Double sum) {
      return 0.15 + sum * 0.85;
    }
  });
}
```

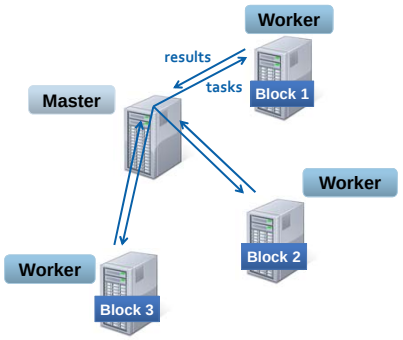

Spark

- 原理
- 编程
- 系统实现

运算过程

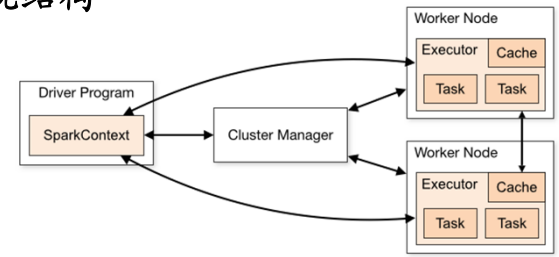
读入内存一次

在内存中可以多次处理



图来源：NSDI'12 slides

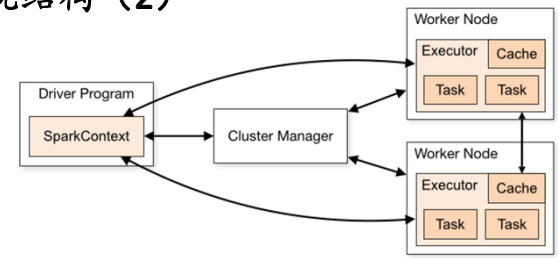
系统结构



- 每个应用程序
  - 一个自己的SparkContext，多个Executor
  - SparkContext从外部的某种资源管理系统获取资源
    - 例如：standalone, hadoop YARN, apache Mesos
  - 每个executor运行在一个不同的worker node上
  - SparkContext协调多个worker运行

图来源：Spark Manual

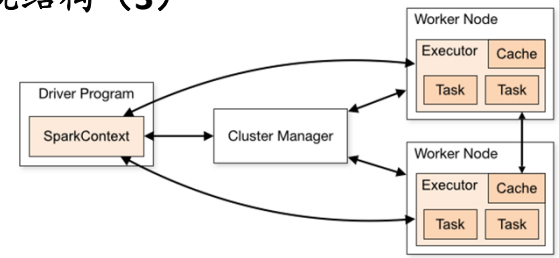
系统结构（2）



- 应用程序
  - 有一个driver主程序，创建SparkContext，发出各种RDD操作要求
- Executor：执行并行的运算，存储数据

图来源：Spark Manual

系统结构（3）



- 多个应用程序
  - 各自有自己的SparkContext
    - 互相隔离，但是也无法共享数据
  - 必须通过外部的文件系统进行数据共享

图来源：Spark Manual

Spark中RDD存储方式

- 有多种方式，可以通过preserve()函数指定每个RDD的存储方式
  - 内存缓冲
  - 内存serialized(顺序化)
  - 多个内存副本
  - 硬盘保存
  - 多种组合

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

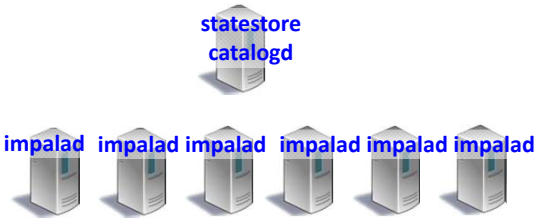
Outline

- 内存计算
  - 内存数据库
  - 内存键值系统
  - 内存MapReduce
    - Spark
    - Cloudera Impala

Cloudera Impala

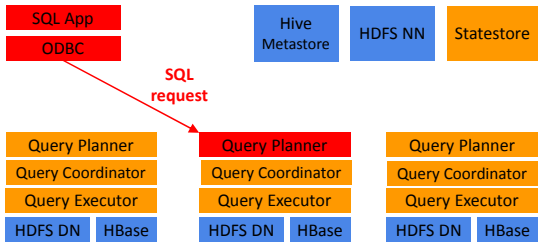
- Cloudera公司
  - 2009成立
  - 致力于开源大数据平台和服务
  - CDH (Cloudera Distribution Including Apache Hadoop)
    - Cloudera发布的软件，包括Hadoop, Hbase, Hive, Spark, Impala, etc.
- Cloudera Impala
  - 2012/10第一次发布
  - 开源的SQL query engine，支持HDFS, HBase等
  - 支持Hive相同的SQL界面

Cloudera Impala系统结构



- Impalad：并行运算
- Statestore：监控impalad的状态
- Catalogd：SQL的表格的schema的增删改操作

Impala内部模块



图来源：Cloudera Impala: A Modern SQL Engine for Apache Hadoop

Impala内部

- 前端（Java）：支持HiveQL
  - SQL parsing
  - Query planning
- 后端（C++）
  - Impalad, statestore, catalogd
  - 其中impalad
    - Query coordinator
    - Query execution engine
    - Code generation using LLVM