第五讲 MapReduce编程

徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

大纲

2

- 编程方式
  - Java
  - Hadoop Streaming
- 编程实例
  - 单元运算：WordCount
  - 二元运算：Join
  - 迭代运算：K-Means
  - 广播变量：Join优化

MapReduce编程范型

3

```
Class X {
    map() {              //map函数的实现
      …
    }

    reduce() {           //reduce函数的实现

    }

    main(){
      Job job = …        //定义分布式作业
      job.config =       //作业参数设置
    }
}
```

数据类型

4

- Map函数
  - 输入：LongWritable, Text
  - 输出：Text, IntWritable
- Reduce函数
  - 输入：Text, IntWritable
  - 输出：Text, IntWritable

Java primitive	Writable implementation	Serialized size(bytes)
boolean	BooleanWritable	1
byte	ByteWritable	1
short	ShtortWritable	2
int	IntWritable	4
	VIntWritable	1-5
float	FloatWritable	4
long	LongWritable	8
	VLongWritable	1-9
double	DoubleWritable	8
String	Text	?

Writable接口

5

- Writable接口定义了两个方法：
  - 将状态写到DataOutput二进制流
  - 从DataInput二进制流读取状态
  - MapReduce的key和value都必须实现Writable接口

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public interface Writable {
        void wirte(DataOutput out) throws IOException;
        void read(DataInput in) throws IOException;
}
```

Hadoop序列化

6

- 序列化（Serialization）是指把结构化对象转化为字节流以便在网络上传输或在磁盘上永久存储的过程
- 反序列化（Deserialization）是指将将字节流转化为结构化对象的逆过程
- 序列化格式特点：
  - 紧凑：高效使用存储空间。
  - 快速：读写数据的额外开销小
  - 可扩展：可透明地读取老格式的数据
  - 互操作：支持多语言的交互

Hadoop的序列化格式：Writable

大纲

7

- 编程方式
  - Java
  - Hadoop Streaming
- 编程实例
  - 单元运算：WordCount
  - 二元运算：Join
  - 迭代运算：K-Means
  - 广播变量：Join优化

Hadoop Streaming

8

- Hadoop基于Java开发，但MapReduce编程不仅限于Java语言
- Hadoop提供的一个编程工具，它允许用户使用任何可执行文件或者脚本文件作为Mapper和Reducer
  - C
  - C++
  - Shell脚本
  - Python
  - …

Hadoop Streaming用法

9

- $HADOOP_HOME/contrib/streaming/hadoop-*-streaming.jar [options]
- options：

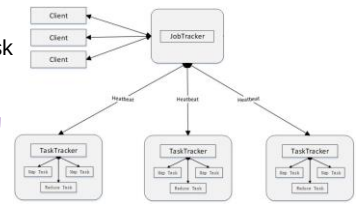
（1）-input：输入文件路径
（2）-output：输出文件路径
（3）-mapper：用户自己写的mapper程序，可以是可执行文件或者脚本
（4）-reducer：用户自己写的reducer程序，可以是可执行文件或者脚本
（5）-file：打包文件到提交的作业中，可以是mapper或者reducer要用的输入文件，如配置文件，字典等。
（6）-partitioner：用户自定义的partitioner程序
（7）-combiner：用户自定义的combiner程序（必须用java实现）
（8）-D：作业的一些属性（以前用的是-jonconf），具体有：

Hadoop Streaming原理

10

- **执行任务**：Map和Reduce函数如何执行?
  - Map Task
  - Reduce Task

利用反射和代理机制动态加载代码



- Java进程(Task)可以动态加载其它语言编写的代码

大纲

11

- 编程方式
  - Java
  - Hadoop Streaming
- 编程实例
  - 单元运算：WordCount
  - 二元运算：Join
  - 迭代运算：K-Means
  - 广播变量：Join优化

实例：WordCount程序任务

12

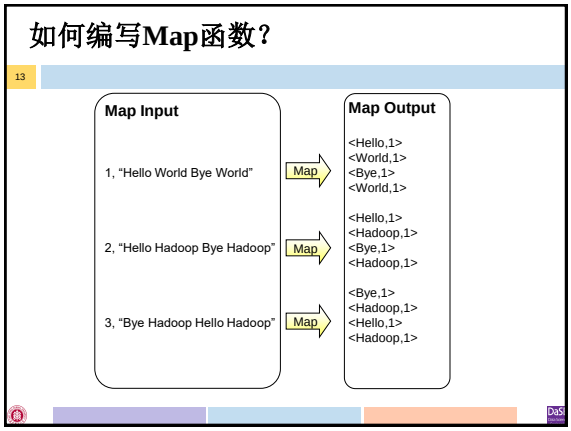
表7-2 WordCount程序任务

程序	WordCount
输入	一个包含大量单词的文本文件
输出	文件中每个单词及其出现次数（频数），并按照单词字母顺序排序，每个单词和其频数占一行，单词和频数之间有间隔

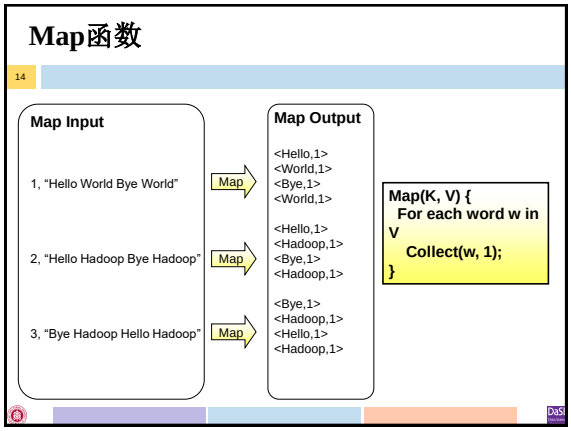
表7-3 一个WordCount的输入和输出实例

输入	输出
Hello World Hello Hadoop Hello MapReduce	Hadoop 1 Hello 3 MapReduce 1 World 1

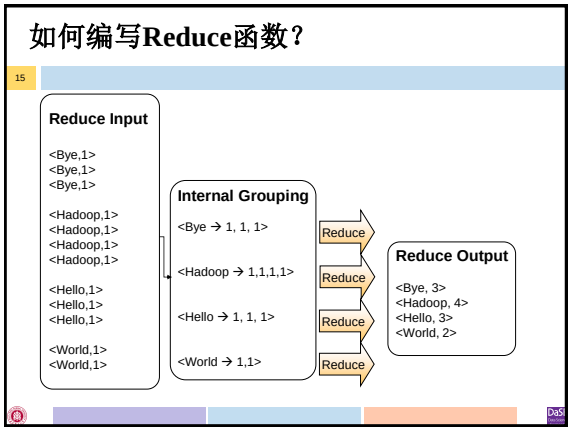
如何编写Map函数？



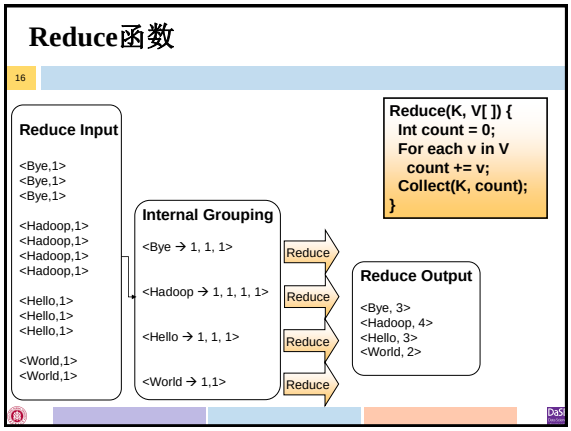
Map函数



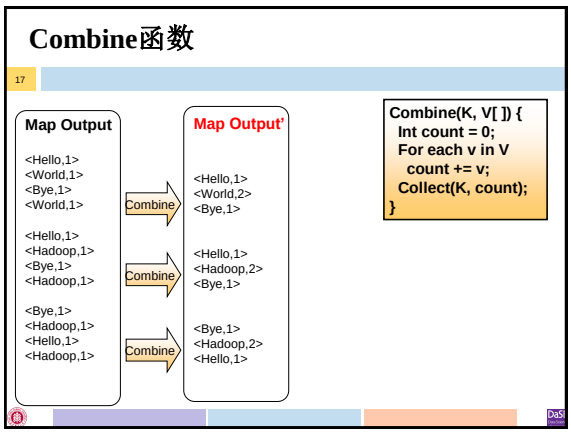
如何编写Reduce函数？



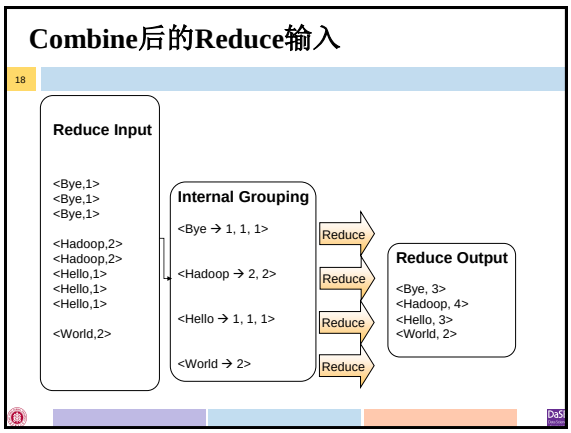
Reduce函数



Combine函数



Combine后的Reduce输入



编写Map处理逻辑

19

- Map输入类型为<key,value>
- 期望的Map输出类型为<单词，出现次数>
- Map输入类型最终确定为<Object,Text>
- Map输出类型最终确定为<Text,IntWritable>

```
public static class MyMapper extends Mapper<Object,Text,Text,IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context) throws
IOException,InterruptedException{
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens())
            {
                word.set(itr.nextToken());
                context.write(word,one);
            }
        }
    }
```

编写Reduce处理逻辑

20

- 在Reduce处理数据之前，Map的结果首先通过Shuffle阶段进行整理
- Reduce阶段的任务：对输入数字序列进行求和
- Reduce的输入数据为<key,Iterable容器>

Reduce任务的输入数据：
<"I",<1,1>>
<"is",1>
……
<"from",1>
<"China",<1,1,1>>

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

编写main方法

21

```
public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();  //程序运行时参数
    String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
    if (otherArgs.length != 2)
    {    System.err.println("Usage: wordcount <in> <out>");
        System.exit(2);
    }
    Job job = new Job(conf,"word count");  //设置环境参数
    job.setJarByClass(WordCount.class);  //设置整个程序的类名
    job.setMapperClass(MyMapper.class);  //添加MyMapper类
    job.setReducerClass(MyReducer.class);  //添加MyReducer类
    job.setOutputKeyClass(Text.class);  //设置输出类型
    job.setOutputValueClass(IntWritable.class);  //设置输出类型
    FileInputFormat.addInputPath(job,new Path(otherArgs[0]));  //设置输入文件
    FileOutputFormat.setOutputPath(job,new Path(otherArgs[1]));  //设置输出文件
    System.exit(job.waitForCompletion(true)?0:1);
}
```

大纲

22

- 编程方式
  - Java
  - Hadoop Streaming
- 编程实例
  - 单元运算：WordCount
  - 二元运算：Join
  - 迭代运算：K-Means
  - 广播变量：Join优化

关系的自然连接

23

- 关系R(A，B)和S(B,C)，公共属性为B

雇员

Name	EmpId	DeptName
Harry	3415	财务
Sally	2241	销售
George	3401	财务
Harriet	2202	销售

部门

DeptName	Manager
财务	George
销售	Harriet
生产	Charles

雇员 ⋈ 部门

Name	EmpId	DeptName	Manager
Harry	3415	财务	George
Sally	2241	销售	Harriet
George	3401	财务	George
Harriet	2202	销售	Harriet

- 问题：对于Map/Reduce函数来说，如何确定一条记录来自那个关系表?

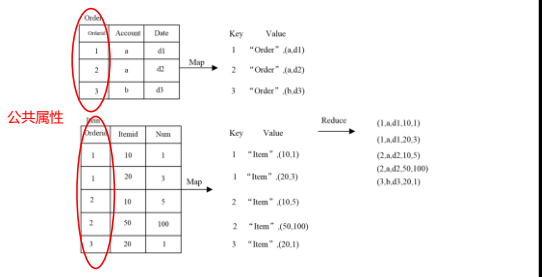
思路

24

- Map过程：标记来自哪个关系表
  - 把来自R的每个元组<a,b>转换成一个键值对<b,<R,a>>，其中的键就是属性B的值
  - 把来自S的每个元组<b,c>，转换成一个键值对<b,<S,c>>
- Reduce过程：
  - 具有相同B值的元组被发送到同一个Reduce中
  - 来自关系R和S的、具有相同属性B值的元组进行合并
  - 输出则是连接后的元组<a,b,c>，通常写到一个单独的输出文件中

自然连接过程示意

25



大纲

26

- 编程方式
  - Java
  - Hadoop Streaming
- 编程实例
  - 单元运算：WordCount
  - 二元运算：Join
  - 迭代运算：K-Means
  - 广播变量：Join优化

迭代式任务

27

- 迭代式任务的特征
  - 整个任务一系列子任务的循环构成
  - 子任务的执行的操作是完全相同的
  - 一个子任务的输出是下一个子任务的输入
  - 一个子任务是一个MapReduce Job
- 举例
  - K-Means
  - PageRank

迭代程序的写法

28

- runIteration()实现一个MapReduce Job

```
//main function
while (!converged && iteration <= maxIterations) {
      log.info("K-Means Iteration {}", iteration);
      // point the output to a new directory per iteration
      Path clustersOut = new Path(output, AbstractCluster.CLUSTERS_DIR + iteration);
      converged = runIteration(conf, input, clustersIn, clustersOut,
measure.getClass().getName(), delta);
      // now point the input to the old output directory
      clustersIn = clustersOut;
      iteration++;
}
```

K-Means一次更新clusters

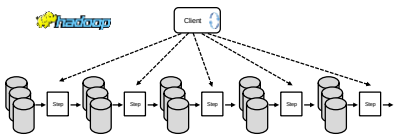
29

```
private static boolean runIteration(Configuration conf,
                                    Path input,
                                    Path clustersIn,
                                    Path clustersOut,
                                    String measureClass,
                                    String convergenceDelta)
  throws IOException, InterruptedException, ClassNotFoundException {

  conf.set(KMeansConfigKeys.CLUSTER_PATH_KEY, clustersIn.toString());
  conf.set(KMeansConfigKeys.DISTANCE_MEASURE_KEY, measureClass);
  conf.set(KMeansConfigKeys.CLUSTER_CONVERGENCE_KEY, convergenceDelta);

  Job job = new Job(conf, "KMeans Driver running runIteration over clustersIn: " + clustersIn);
  job.setMapOutputKeyClass(Text.class);
  job.setMapOutputValueClass(ClusterObservations.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Cluster.class);

  job.setInputFormatClass(SequenceFileInputFormat.class);
  job.setOutputFormatClass(SequenceFileOutputFormat.class);
  job.setMapperClass(KMeansMapper.class);
  job.setCombinerClass(KMeansCombiner.class);
  job.setReducerClass(KMeansReducer.class);

  FileInputFormat.addInputPath(job, input);
  FileOutputFormat.setOutputPath(job, clustersOut);

  job.setJarByClass(KMeansDriver.class);
  HadoopUtil.delete(conf, clustersOut);
  if (!job.waitForCompletion(true)) {
    throw new InterruptedException("K-Means Iteration failed processing " + clustersIn);
  }
  FileSystem fs = FileSystem.get(clustersOut.toUri(), conf);

  return isConverged(clustersOut, conf, fs);
}
```

迭代MapReduce示意

30

- 每一迭代步(Step)结束时将结果写入HDFS，下一步将该结果再次从HDFS读出



大纲

31

- 编程方式
  - Java
  - Hadoop Streaming
- 编程实例
  - 单元运算：WordCount
  - 二元运算：Join
  - 迭代运算：K-Means
  - 广播变量：Join优化

大表⋈小表

32

- 假如“部门”表比较小，“雇员”表非常大

雇员

Name	EmpId	DeptName
Harry	3415	财务
Sally	2241	销售
George	3401	财务
Harriet	2202	销售

部门

DeptName	Manager
财务	George
销售	Harriet
生产	Charles

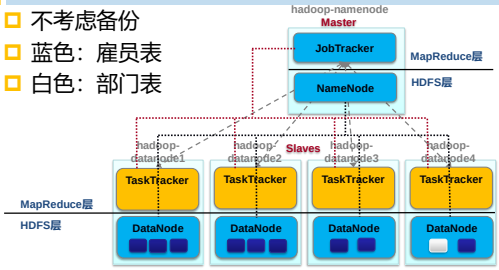
雇员 ⋈ 部门

Name	EmpId	DeptName	Manager
Harry	3415	财务	George
Sally	2241	销售	Harriet
George	3401	财务	George
Harriet	2202	销售	Harriet

数据分布示列

33

- 不考虑备份
- 蓝色：雇员表
- 白色：部门表

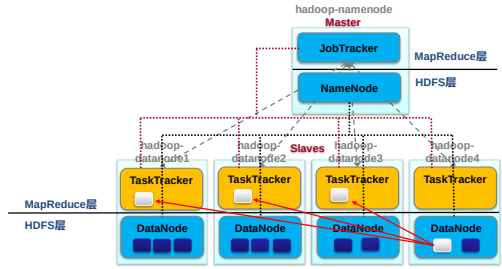


- 若数据本身无序，连接将有大量的数据移动

优化方案

34

- 编程时将“小表”广播出去



Distributed Cache

35

- 声明：

```
Job job = new Job();
...
job.addCacheFile(new Path(filename).toUri());
```

- 使用：

```
Path[] localPaths = context.getLocalCacheFiles();
```

K-Means

36

- 可以使用Distributed Cache吗?

部署与编程

37

- 部署
  - 单机集中式：MapReduce所有角色位于同一进程，此时相当于一个普通的Java工具包
  - 单机伪分布式： MR所有进程在同一物理机器
    - 实验时，请在个人用户中独立完成
  - 分布式部署：MR各进程在不同的物理机器
    - 实验时，请多位同学创建相同的公共用户（例如，ecnu）协作完成
- 编程
  - 实验时，请在个人用户中独立完成

本讲小结

38

- 编程方式
  - Java
  - Hadoop Streaming
- 编程实例
  - 单元运算
  - 二元运算
  - 迭代运算
  - 广播变量

谢谢！Q&A