MapReduce 工作原理

侯在钱

目 录

1. MapReduce 工作过程

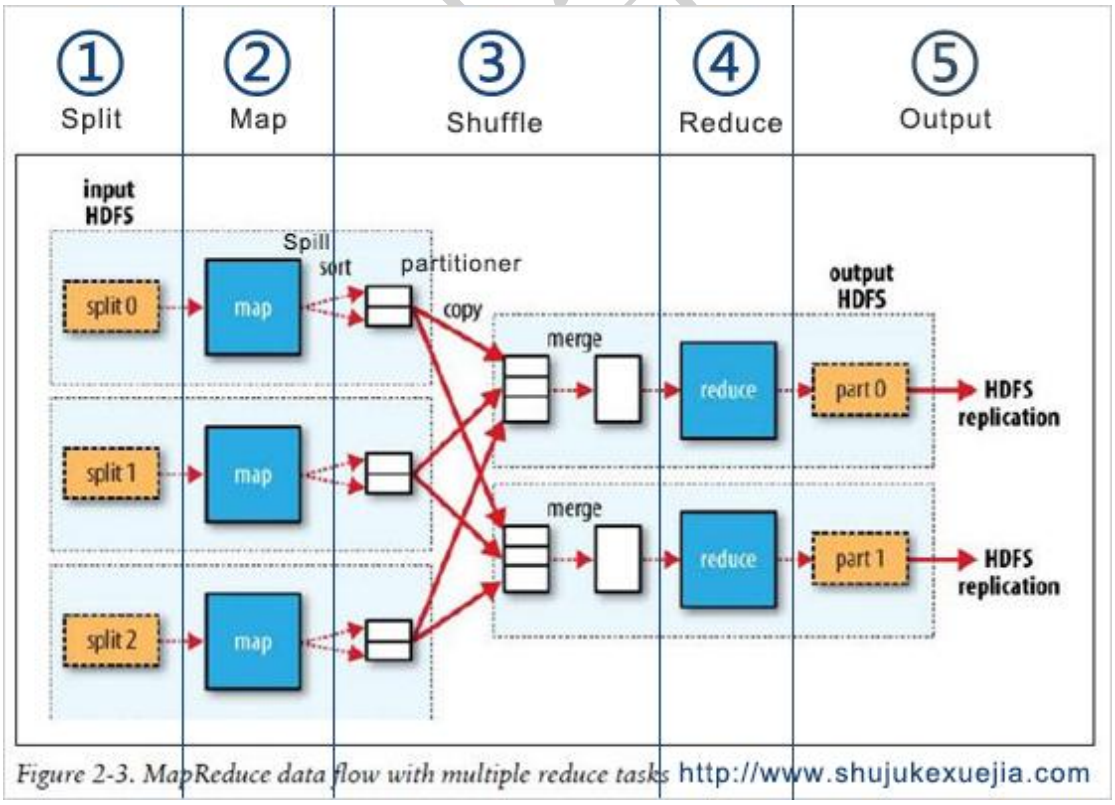


Figure 2-3. MapReduce data flow with multiple reduce tasks http://www.shujukexuejia.com

MapReduce 处理数据过程主要分成 5 个阶段：

Input（Split）-> Map -> Shuffle -> Reduce -> Output

这 5 个阶段分别的作用如下：

（1） 第 1 个阶段，Split。在执行 Map 函数前，需要对输入进行“分片”（就是将海量数据分成很多块，一个分片默认是 64M），以便于多个 Map 同时工作，每一个 Map 任务处理一个“分片”。Map 对每条记录的输出以<key,value> 值对的形式输出。

（2） 第 2 个阶段，Map。Reduce 输入数据是一个 Map 结构，输出也是一个 Map 结构。也就是 Map 是把一个 Map 数据结构变成另一种 Map 结构。

（3） 第 3 个阶段，Shuffle。Shuffle 的工作是从 Map 输出到 Reduce 输入的这段过程，是连接 Map 和 Reduce 的这个一个过程。在这个过程中需要重新组合数据，所以需要排序（Sort）、分区（Partitioner）、拷贝（Copy）、合并（Merge）等工作。

（4） 第 4 个阶段，Reduce。Reduce 的工作是削减 Map 的输出，并重新组合后的结果。Reduce 输入数据是一个 Map 结构，输出也是一个 Map 结构。也就是 Reduce 是把一个 Map 数据结构变成另一种 Map 结构，但变换的过程中需要削减 Map 中的 Value 值。

（5） 第 5 个阶段，把 Reduce 的结果写入磁盘文件。

实际上。下图中，左边的工作是放在一台机器上执行的，右边的工作是放在另一台机器上执行的。我们把执行左边工作的机器称之为 Map 端，右边的称之为 Reduce。所以有些人把 MapReduce 处理数据过程分为两大阶段，Map 阶段和 Reduce 阶段。

Map 端的工作包括：从分片读入数据，Map，Shuffle 的前半段。

Reduce 端的工作包括：Shuffle 的后半段，Reduce，输出结果写入文件。

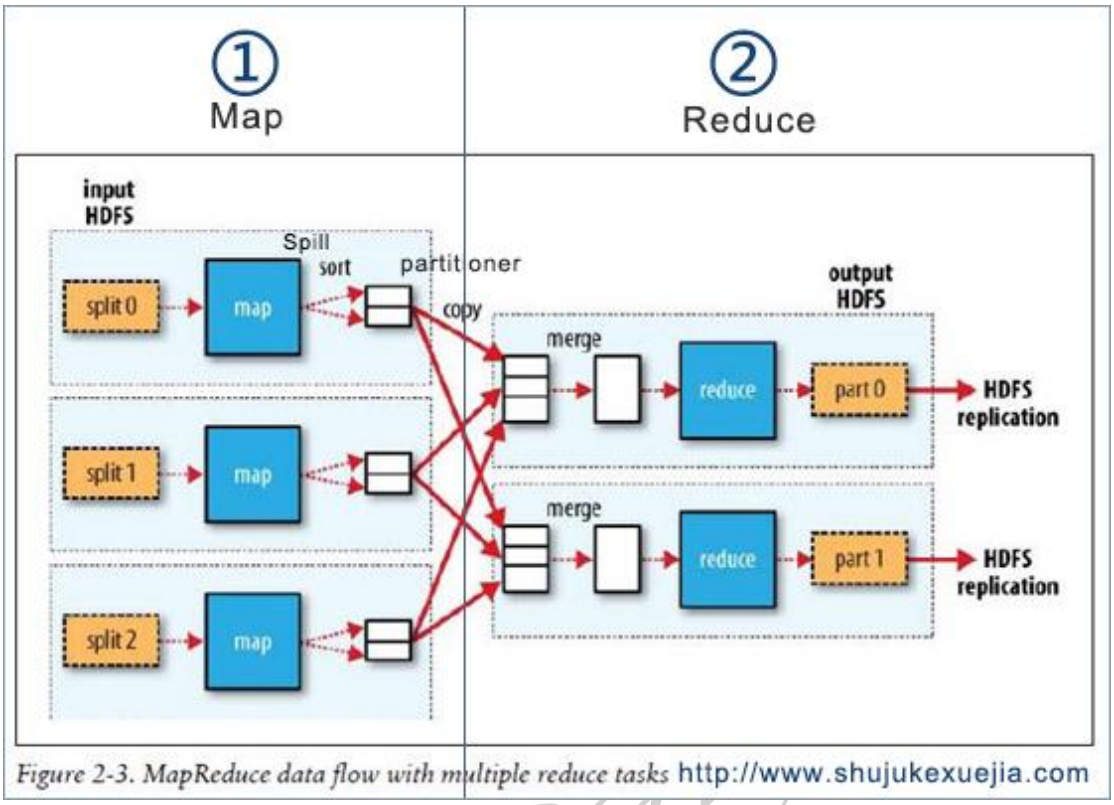


Figure 2-3. MapReduce data flow with multiple reduce tasks http://www.shujukexuejia.com

2. MapReduce 工作原理

下面以使用 MapReduce 实现对文章中单词出现个数进行统计为例介绍 MapReduce 的工作原理。

例如，要统计单词个数的文本文件如下：

want to get more

want to get get

about HTML

If you want to get more

If you want to get

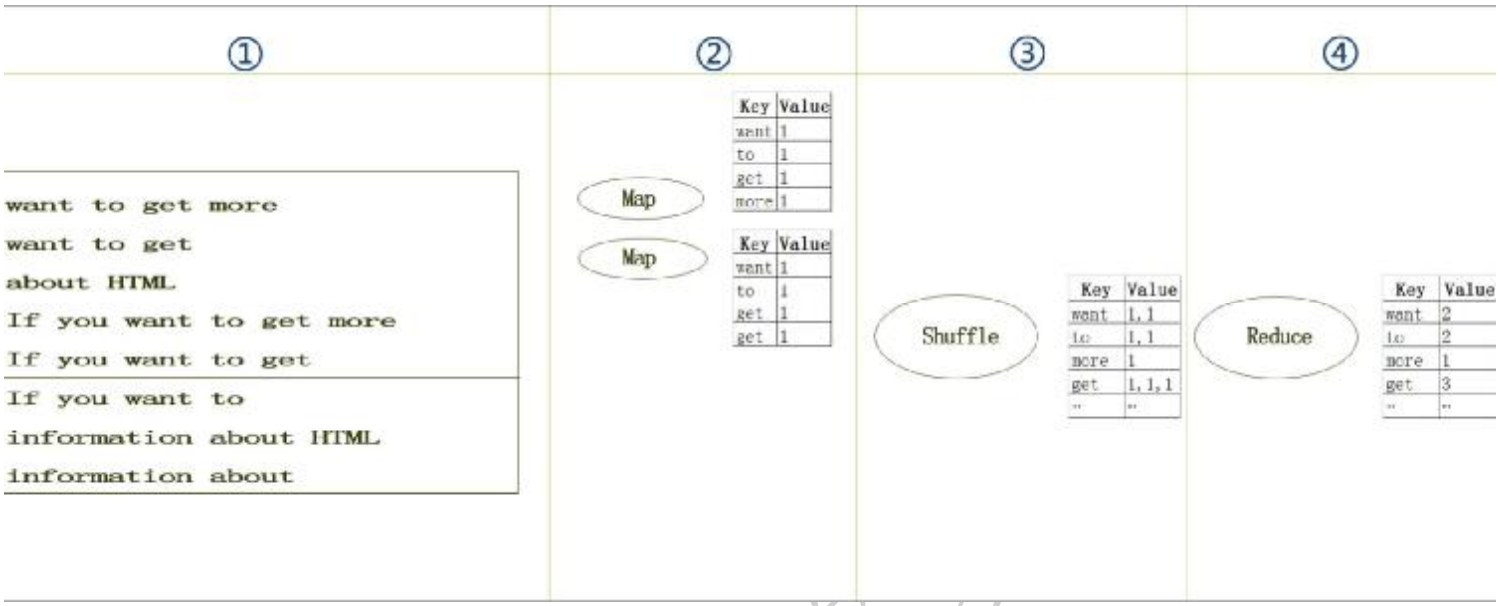
If you want to

information about HTML

information about

2.1. 统计单词个数的 MapReduce 工程流程

统计单词个数的 MapReduce 的工作过程如下：



2.2. Map 程序

```
package com.hadoop.mp;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;;

public class WordCountMapper extends  Mapper<Object, Text, Text, IntWritable> {

   public void map(LongWritable key, Text value,  Context  context)  throws
IOException,InterruptedException {
       //把value值用空格进行分割
       String s = value.toString();
       String[] words = s.split(" ");

       //把单词名作为输出Map的Key，输出Map的Value直接写上数字1。
       if (words!=null&&words.length>0) {
          for (int i=0; i<words.length; i++) {
              Text outputKey = new Text(words[i]);
              IntWritable outputValue = new IntWritable(1);
```

```
            //把结果写到写到输出Map中
            context.write(outputKey, outputValue);
        }
    }
}
```

2.3. Reduce 程序

```
package com.hadoop.mp;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer  extends Reducer<Text, IntWritable, Text, IntWritable>
{
    public  void  reduce(Text  key,  Iterable<IntWritable>  values,  Context
context)  throws IOException,InterruptedException {

        //把value值进行累加求和
        int result = 0;
        for (IntWritable val : values) {
            result = result + val.get();
        }

        //输入Map的Key作为输出Map的Key
        //value累加求和后的结果作为输出Map的Value
        context.write(key,new IntWritable(result));

    }

}
```