

## 三种方法实现Hadoop(MapReduce)全局排序(1)

我们可能会有些需求要求MapReduce的输出全局有序，这里说的有序是指Key全局有序。但是我们知道，MapReduce默认只是保证同一个分区内的Key是有序的，但是不保证全局有序。基于此，本文提供三种方法来对MapReduce的输出进行全局排序。

### 生成测试数据

在介绍如何实现之前，我们先来生成一些测试数据，实现如下：

```
#!/bin/sh

for i in {1..100000};do
    echo $RANDOM
done;
```

将上面的代码保存到 iteblog.sh 的文件里面，然后运行：

```
$ sh iteblog.sh > data1
$ sh iteblog.sh > data2
$ hadoop fs -put data1 /user/iteblog/input
$ hadoop fs -put data2 /user/iteblog/input
```

\$RANDOM 变量是Shell内置的，使用它能够生成五位内的随机正整数。上面我们一共运行了两次，这样我们就有两份随机数文件data1和data2；最后我们把生成的随机数文件上传到HDFS上。现在我们可以来写程序对这两个文件里面的数据进行排序了。

### 使用一个Reduce进行排序

前面我们说了，MapReduce默认只是保证同一个分区内的Key是有序的，但是不保证全局有序。如果我们将所有的数据全部发送到一个Reduce，那么不就可以实现结果全局有序吗？这种方法实现很简单，如下：

```
package com.iteblog.mapreduce.sort;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import java.io.IOException;

public class TotalSortV1 extends Configured implements Tool {
    static class SimpleMapper extends
        Mapper<LongWritable, Text, IntWritable, IntWritable> {
        @Override
        protected void map(LongWritable key, Text value,
            Context context) throws IOException, InterruptedException {
            IntWritable intWritable = new IntWritable(Integer.parseInt(value.toString()));
            context.write(intWritable, intWritable);
        }
    }

    static class SimpleReducer extends
        Reducer<IntWritable, IntWritable, IntWritable, NullWritable> {
        @Override
        protected void reduce(IntWritable key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {
            for (IntWritable value : values)
                context.write(value, NullWritable.get());
        }
    }

    @Override
    public int run(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("<input> <output>");
            System.exit(127);
        }

        Job job = Job.getInstance(getConf());
        job.setJarByClass(TotalSortV1.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
    }
}
```

```

    job.setMapperClass(SimpleMapper.class);
    job.setReducerClass(SimpleReducer.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(NullWritable.class);
    job.setNumReduceTasks(1);
    job.setJobName("TotalSort");
    return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new TotalSort(), args);
    System.exit(exitCode);
}
}

```

上面程序的实现很简单，我们直接使用 `TextInputFormat` 类来读取上面生成的随机数文件（`data1` 和 `data2`）。因为文件里面的数据是正整数，所以我们在 `SimpleMapper` 类里面直接将 `value` 转换成 `int` 类型，然后赋值给 `IntWritable`。等数据到 `SimpleReducer` 的时候，同一个 `Reduce` 里面的 `Key` 已经全部有序；因为我们设置了一个 `Reduce` 作业，这样的话，我们就实现了数据全局有序。运行如下：

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop jar total-sort-0.1.jar com.iteblog.mapreduce.sort.TotalSortV1 /user/iteblog/input /user/iteblog/output
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -ls /user/iteblog/output
Found 2 items
-rw-r--r--  3 iteblog supergroup      0 2017-05-09 11:41 /user/iteblog/output/_SUCCESS
-rw-r--r--  3 iteblog supergroup 1131757 2017-05-09 11:41 /user/iteblog/output/part-r-00000
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output/part-r-00000 | head -n 10
0
0
0
0
1
1
1
1
```

```
1
1

[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output/part-
r-00000 | tail -n 10
32766
32766
32766
32766
32767
32767
32767
32767
32767
32767
```

从上面的测试结果也可以看出，我们只生成了一个数据文件，而且这个文件里面的数据已经全局有序了。

## 自定义分区函数实现全局有序

上面实现数据全局有序有个很大的局限性：所有的数据都发送到一个Reduce进行排序，这样不能充分利用集群的计算资源，而且在数据量很大的情况下，很有可能会出现OOM问题。我们分析一下，MapReduce默认的分区函数是HashPartitioner，其实现的原理是计算map输出key的hashCode，然后对Reduce个数求模，这样只要求模结果一样的Key都会发送到同一个Reduce。如果我们能够实现一个分区函数，使得

- 所有 Key
- 所有 10000
- 其余的Key都发送到Reduce 2；

这就实现了Reduce 0的数据一定全部小于Reduce 1，且Reduce 1的数据全部小于Reduce 2，再加上同一个Reduce里面的数据局部有序，这样就实现了数据的全局有序。实现如下：

```
package com.iteblog.mapreduce.sort;

import com.iteblog.mapreduce.secondSort.IntPair;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import java.io.IOException;

public class TotalSortV2 extends Configured implements Tool {
    static class SimpleMapper extends Mapper<LongWritable, Text, IntWritable, IntWritable> {
        @Override
        protected void map(LongWritable key, Text value,
                           Context context) throws IOException, InterruptedException {
            IntWritable intWritable = new IntWritable(Integer.parseInt(value.toString()));
            context.write(intWritable, intWritable);
        }
    }

    static class SimpleReducer extends Reducer<IntWritable, IntWritable, IntWritable, NullWritable> {
        @Override
        protected void reduce(IntWritable key, Iterable<IntWritable> values,
                              Context context) throws IOException, InterruptedException {
            for (IntWritable value : values)
                context.write(value, NullWritable.get());
        }
    }

    public static class IteblogPartitioner extends Partitioner<IntWritable, IntWritable> {
        @Override
        public int getPartition(IntWritable key, IntWritable value, int numPartitions) {
            int keyInt = Integer.parseInt(key.toString());
            if (keyInt < 10000) {
                return 0;
            } else if (keyInt < 20000) {
                return 1;
            } else {
                return 2;
            }
        }
    }
}
```

```
@Override
public int run(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("<input> <output>");
        System.exit(127);
    }

    Job job = Job.getInstance(getConf());
    job.setJarByClass(TotalSortV2.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(SimpleMapper.class);
    job.setReducerClass(SimpleReducer.class);
    job.setPartitionerClass(IteblogPartitioner.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(NullWritable.class);
    job.setNumReduceTasks(3);
    job.setJobName("dw_subject");
    return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new TotalSortV2(), args);
    System.exit(exitCode);
}
}
```

第二版的排序实现除了自定义的  
IteblogPartitioner，其余的和第一种实现一样。现在我们来运行一下：

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop jar total-
sort-0.1.jar com.iteblog.mapreduce.sort.TotalSortV2 /user/iteblog/input /user/iteblog/output1
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -ls /user/iteblog/output1
Found 4 items
-rw-r--r--  3 iteblog supergroup      0 2017-05-09 13:53 /user/iteblog/output1/_SUCCESS
-rw-r--r--  3 iteblog supergroup 299845 2017-05-09 13:53 /user/iteblog/output1/part-
r-00000
-rw-r--r--  3 iteblog supergroup 365190 2017-05-09 13:53 /user/iteblog/output1/part-
r-00001
```

```
-rw-r--r--  3 iteblog supergroup  466722 2017-05-09 13:53 /user/iteblog/output1/part-  
r-00002
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output1/part-  
r-00000 | head -n 10
```

```
0  
0  
0  
0  
1  
1  
1  
1  
1  
1  
1
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output1/part-  
r-00000 | tail -n 10
```

```
9998  
9998  
9998  
9999  
9999  
9999  
9999  
9999  
9999  
9999  
9999
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output1/part-  
r-00001 | head -n 10
```

```
10000  
10000  
10000  
10000  
10000  
10000  
10001  
10001  
10001  
10001
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output1/part-  
r-00001 | tail -n 10
```

```
19997  
19997
```

```
19998
19998
19998
19998
19999
19999
19999
19999
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output1/part-
r-00002 | head -n 10
```

```
20000
20000
20000
20000
20000
20000
20001
20001
20001
20001
```

```
[iteblog@www.iteblog.com /home/iteblog]$ hadoop fs -cat /user/iteblog/output1/part-
r-00002 | tail -n 10
```

```
32766
32766
32766
32766
32767
32767
32767
32767
32767
32767
```

我们已经看到了这个程序生成了三个文件（因为我们设置了Reduce个数为3），而且每个文件都是局部有序；所有小于10000的数据都在part-r-00000里面，所有小于20000的数据都在part-r-00001里面，所有大于20000的数据都在part-r-00002里面。part-r-00000、part-r-00001和part-r-00002三个文件实现了全局有序。

这个方法也实现了数据的全局有序，但是也有一些问题，明天我再写一篇文章介绍第三种数据全局排序的方法。





优秀人才不缺工作机会，只缺适合自己的好机会。但是他们往往没有精力从海量机会中找到最适合的那个。

100offer 会对平台上的人才和企业进行严格筛选，让「最好的人才」和「最好的公司」相遇。

注册 100offer，谈谈你对下一份工作的期待。一周内，收到 5-10 个满足你要求的好机会！

**本博客文章除特别声明，全部都是原创！**

**禁止个人和公司转载本文、谢谢理解：过往记忆 ( <https://www.iteblog.com/> )**

**本文链接: 【】 ( )**