分布式计算框架MapReduce

Grissom | 2025年10月

TRANSWARP
星 环 科 技

目录》 CONTENTS

1 MapReduce 简介

2 MapReduce 原理

3 MapReduce 作业管理

4 MapReduce 开发案例

1

chapter

MapReduce 简介

MapReduce 简介

起源

- 2004年10月Google发表了MapReduce论文
- 设计初衷是解决搜索引擎中大规模网页数据的并行处理
- Hadoop MapReduce是Google MapReduce的开源实现
- MapReduce是Apache Hadoop的核心子项目

概念

- 面向离线批处理的分布式计算框架
- 分布式编程模型：MapReduce程序被分为Map（映射）阶段和Reduce（化简）阶段

核心思想

- 分而治之，并行计算
- 移动计算，非移动数据

MapReduce 简介

特点

- 计算跟着数据走
- 良好的扩展性：计算能力随着节点数增加，近似线性增长
- 高容错
- 状态监控
- 适合海量数据的离线批处理
- 降低了分布式编程的门槛

MapReduce 简介

适用场景

- 数据统计，如网站的PV、UV统计
- 搜索引擎构建索引
- 海量数据查询
- 复杂数据分析算法实现

不适用场景

- OLAP：要求毫秒或秒级返回结果
- 流计算：输入数据集是动态的，而MapReduce是静态的
- DAG计算
  - 多个任务之间存在依赖关系，后一个的输入是前一个的输出，构成DAG有向无环图
  - MapReduce很难避免Suffle，造成大量磁盘IO，导致性能较为低下

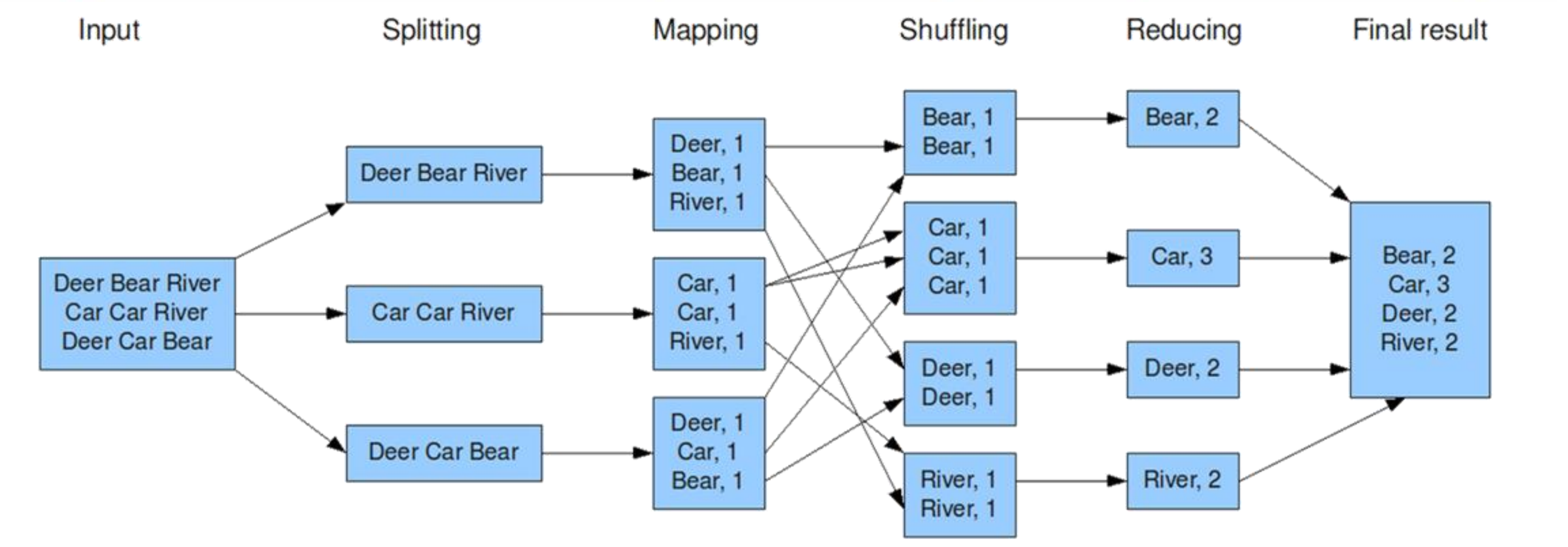
2

chapter

MapReduce 原理

案例：WordCount

➢ 示例：WordCount



MR编程模型

➢ Job & Task（作业与任务）

- 作业是客户端请求执行的一个工作单元
  - 包括输入数据、MapReduce程序、配置信息
- 任务是将作业分解后得到的细分工作单元
  - 分为Map任务和Reduce任务

➢ Split（切片）

- 输入数据被划分成等长的小数据块，称为输入切片（Input Split），简称切片
- Split是逻辑概念，仅包含元数据信息，如数据的起始位置、长度、所在节点等
- 每个Split交给一个Map任务处理，Split的数量决定Map任务的数量
- Split大小
  - 默认等于HDFS Block大小
  - Split的划分方式由程序设定，Split与HDFS Block没有严格的对应关系
  - Split越小，Map任务越多，并发度越高，但开销也越大；Split越大，任务越少，并发度降低

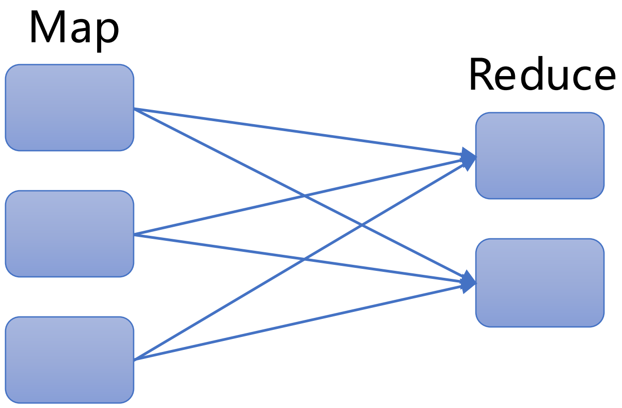
MR编程模型

Map阶段（映射）

- 由若干Map任务组成，任务数量由Split数量决定
- 输入：Split切片（key-value）
- 输出：中间计算结果（key-value）

Reduce阶段（化简）

- 由若干Reduce任务组成，任务数量由程序指定
- 输入：Map阶段输出的中间结果（key-value）
- 输出：最终结果（key-value）



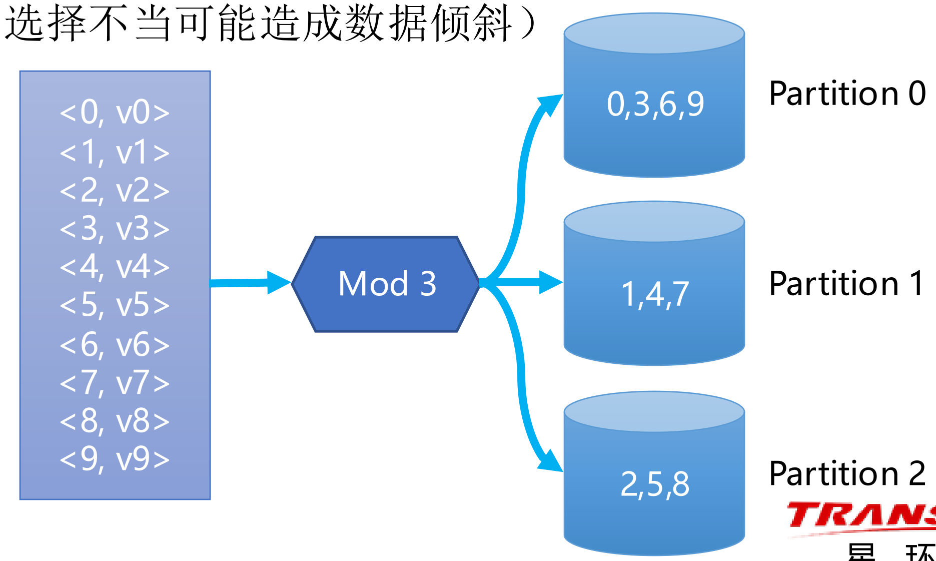
Shuffle Dependency

Shuffle阶段（混洗）

- Shuffle是Map和Reduce之间的强依赖关系（Shuffle依赖）导致的，即每个Reduce的输入依赖于所有Map的输出
- Map和Reduce阶段的中间环节（虚拟阶段），分为Map端Shuffle和Reduce端Shuffle

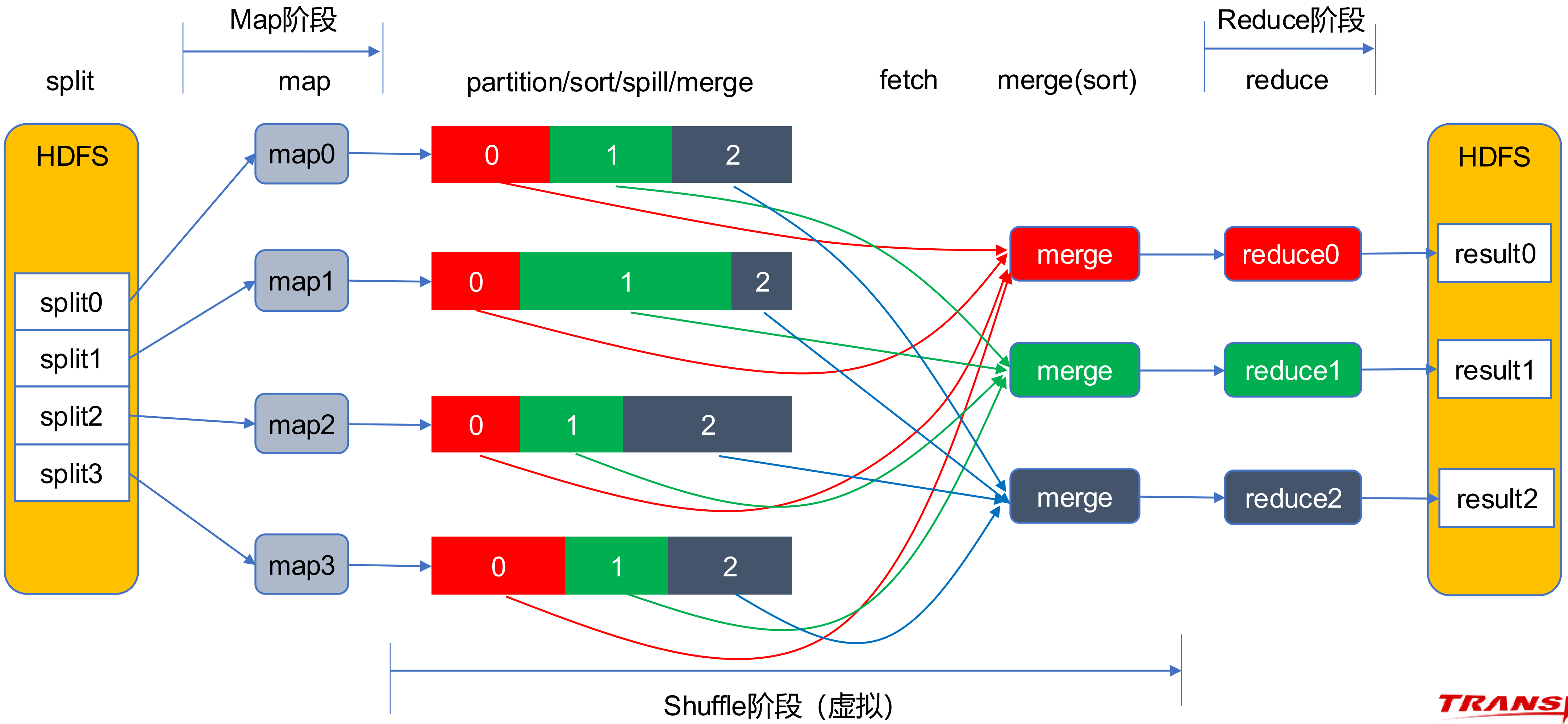
➢ Shuffle阶段（混洗）

- 包括Partition（分区）、Sort（排序）、Spill（溢写）、Merge（合并）和Fetch（抓取）等工作
- Partition（分区）
  - Reduce任务数量决定了Partition数量，Partition编号 = Reduce任务编号
  - 利用“哈希取模”对Map输出数据分区，即Partition编号 = key hashcode % reduce task num（%为取模）
  - Partition为具有相同编号的Reduce任务供数
  - 哈希取模的作用
    - ✓ 数据划分：将一个数据集随机分成若干个子集（Hash函数选择不当可能造成数据倾斜）
    - ✓ 数据聚合：将Key相同的数据聚合在一起
- 避免和减少Shuffle是MapReduce程序调优的关键



MR工作机制

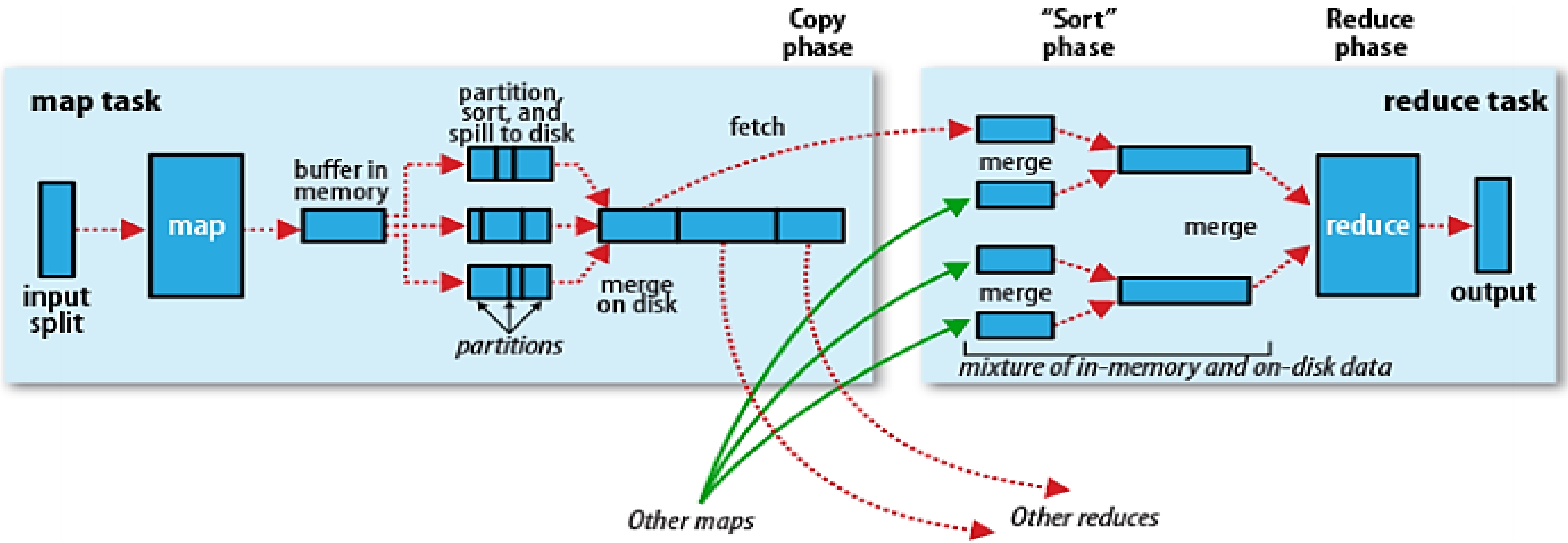
➢ MR执行过程



TRANSWARP
星 环 科 技

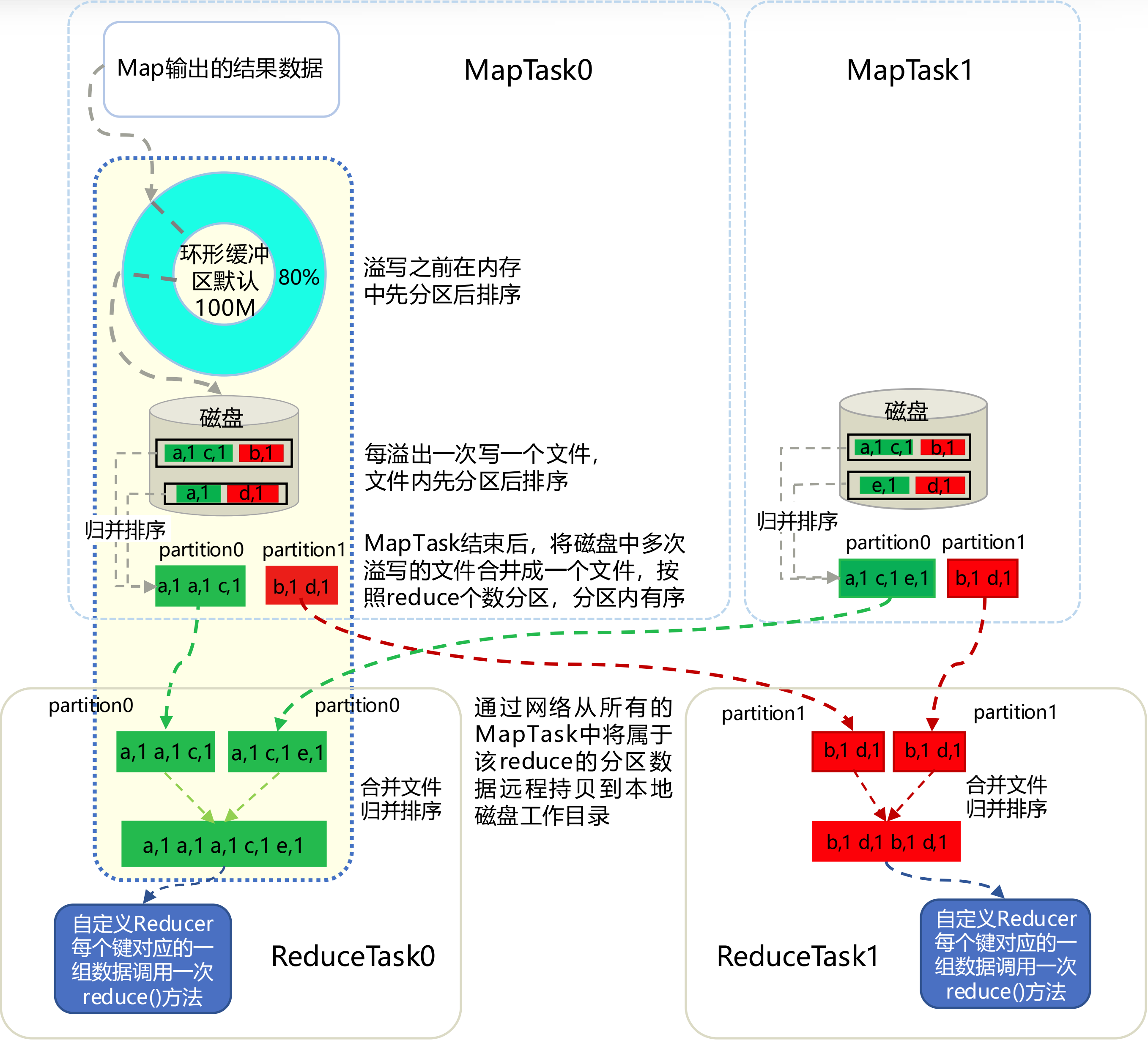
MR工作机制：Shuffle

- Shuffle详解



MR工作机制：Shuffle

- Shuffle详解



溢写之前在内存中先分区后排序

每溢出一次写一个文件，文件内先分区后排序

MapTask结束后，将磁盘中多次溢写的文件合并成一个文件，按照reduce个数分区，分区内有序

通过网络从所有的MapTask中将属于该reduce的分区数据远程持贝到本地磁盘工作目录

MR工作机制：Shuffle

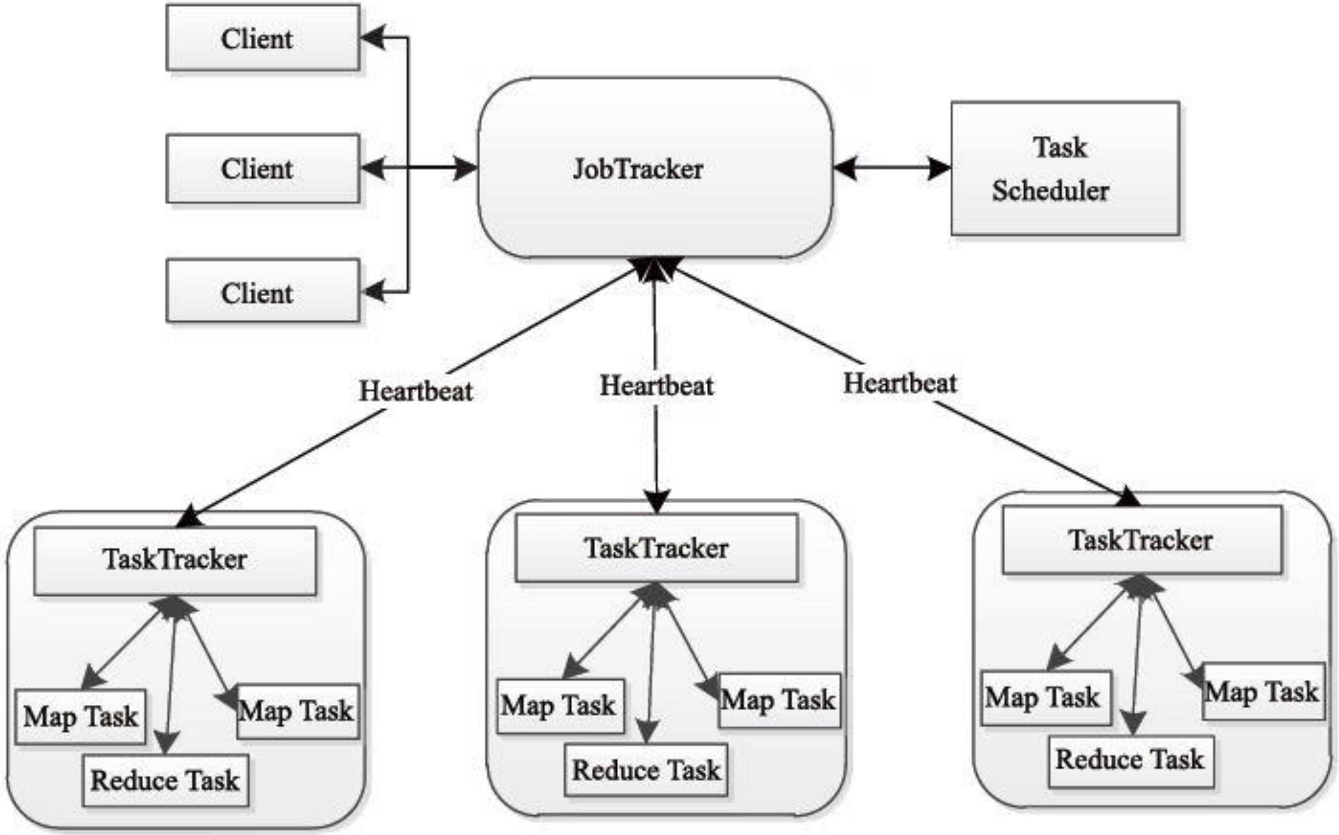
➢ Shuffle详解

- Map端
  - Map任务将中间结果写入环形内存缓冲区Buffer（默认100M），同时进行分区（Partition）和排序（Sort）
    - ✓ 先按“key hashcode % reduce task num”对数据进行分区，分区内再按key排序
  - 当Buffer的数据量达到阈值（默认80%）时，将数据溢写（Spill）到磁盘的一个临时文件中，文件内数据先分区后排序
  - Map任务结束前，将多个临时文件合并（Merge）为一个Map输出文件，文件内数据先分区后排序
- Reduce端
  - Reduce任务从多个Map输出文件中抓取（Fetch）属于自己的分区数据（Partition编号=Reduce任务编号）
  - 对抓取到的分区数据做归并排序，生成一个Reduce输入文件（文件内数据按key排序）
    - ✓ 如果内存缓冲区够大，就直接在内存中完成归并排序，然后落盘
    - ✓ 如果内存缓冲区不够，先将分区数据写到相应的文件中，再通过归并排序合并为一个大文件

MR作业运行模式

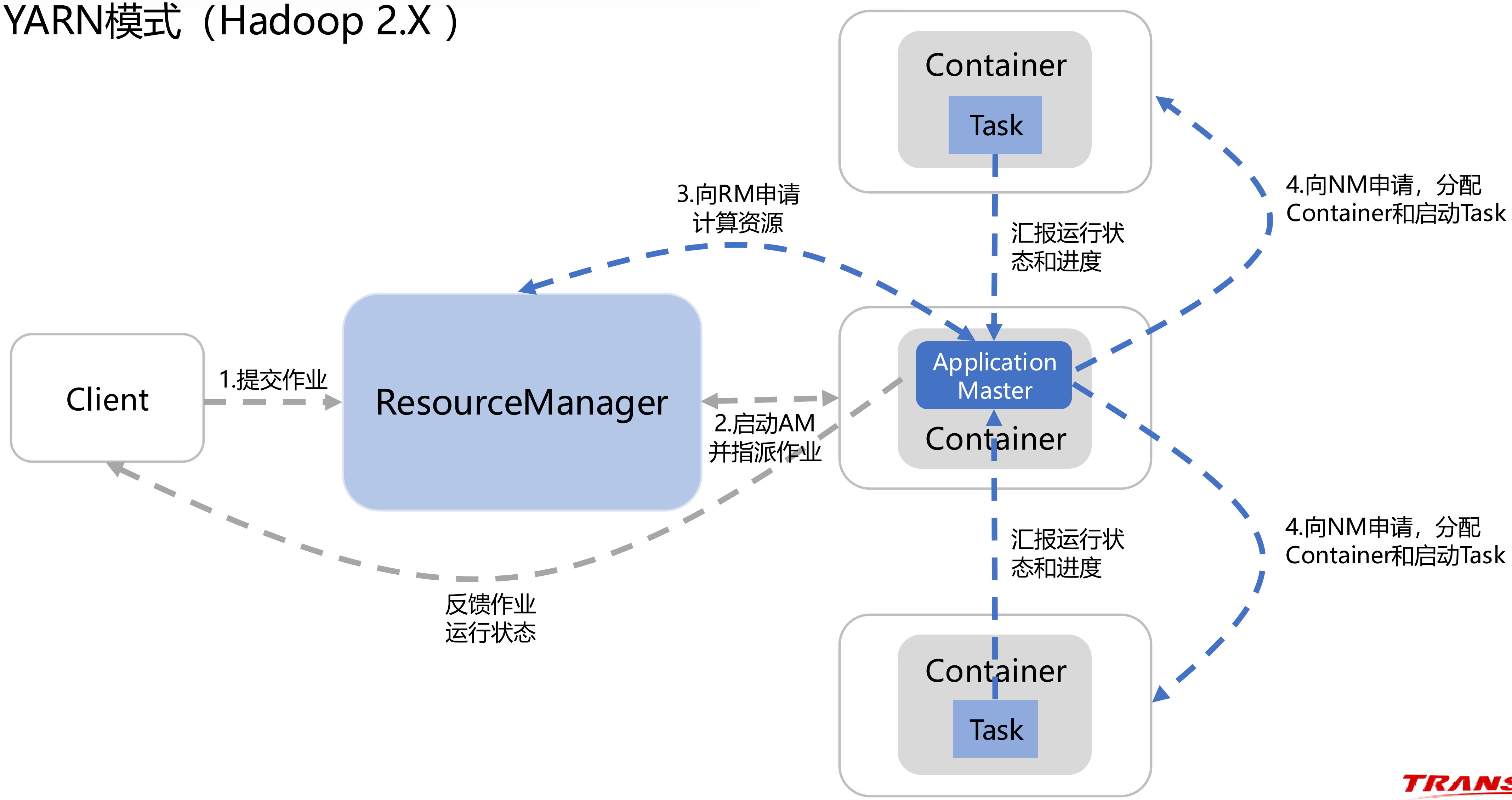
➢ JobTracker/TaskTracker模式（Hadoop 1.X）

- JobTracker节点（Master）
  - 调度任务在TaskTracker上运行
  - 若任务失败，指定新TaskTracker重新运行
- TaskTracker节点（Slave）
  - 执行任务，发送进度报告
- 存在的问题
  - JobTracker存在单点故障
  - JobTracker负载太重（上限4k节点）
  - JobTracker缺少对资源的全面管理
  - TaskTracker对资源的描述过于简单
  - 源码难于理解



MR作业运行模式

➢ YARN模式（Hadoop 2.X ）



3

chapter

MapReduce 作业管理

1.3 作业管理

➢ 提交作业

```
# hadoop  jar  {jarFile}  [mainClass]  args
  -jarFIle: MapReduce运行程序的jar包
  -mainClass: jar包中main函数所在类的类名
  -args: 程序调用需要的参数，如输入输出路径
```

➢ 查看作业

```
# sudo –u yarn application -list
```

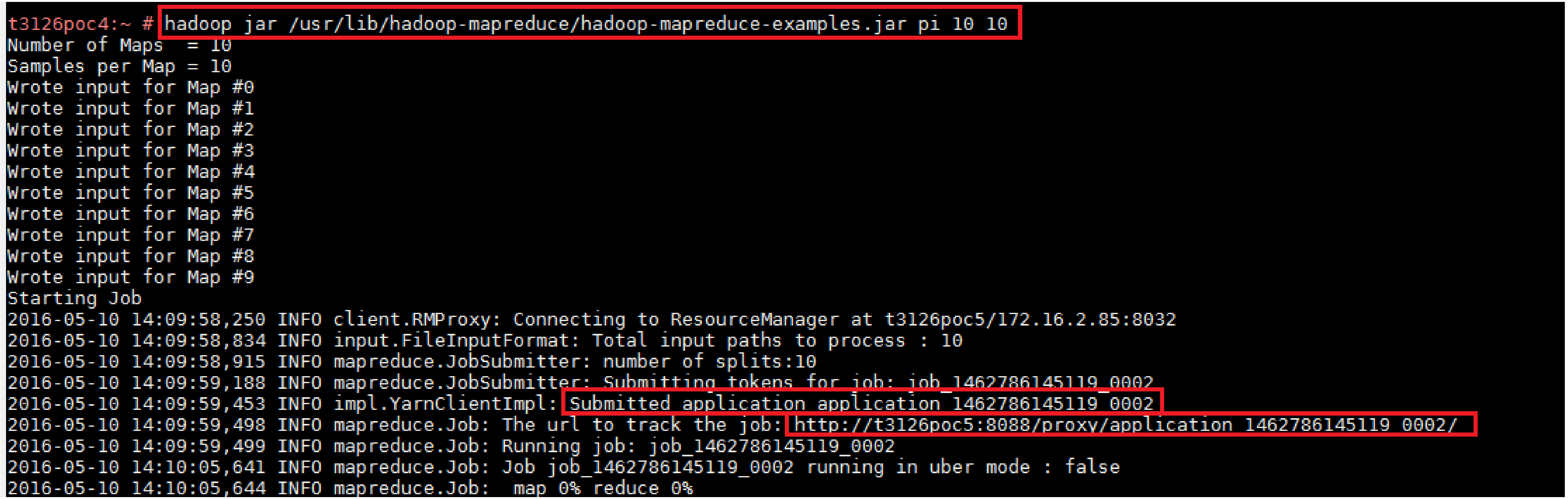
➢ 终止作业

```
# sudo –u yarn application -kill {application_id}
```

1.3 作业管理

➢ 示例：提交作业

```
# hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-example.jar pi 10 10
```

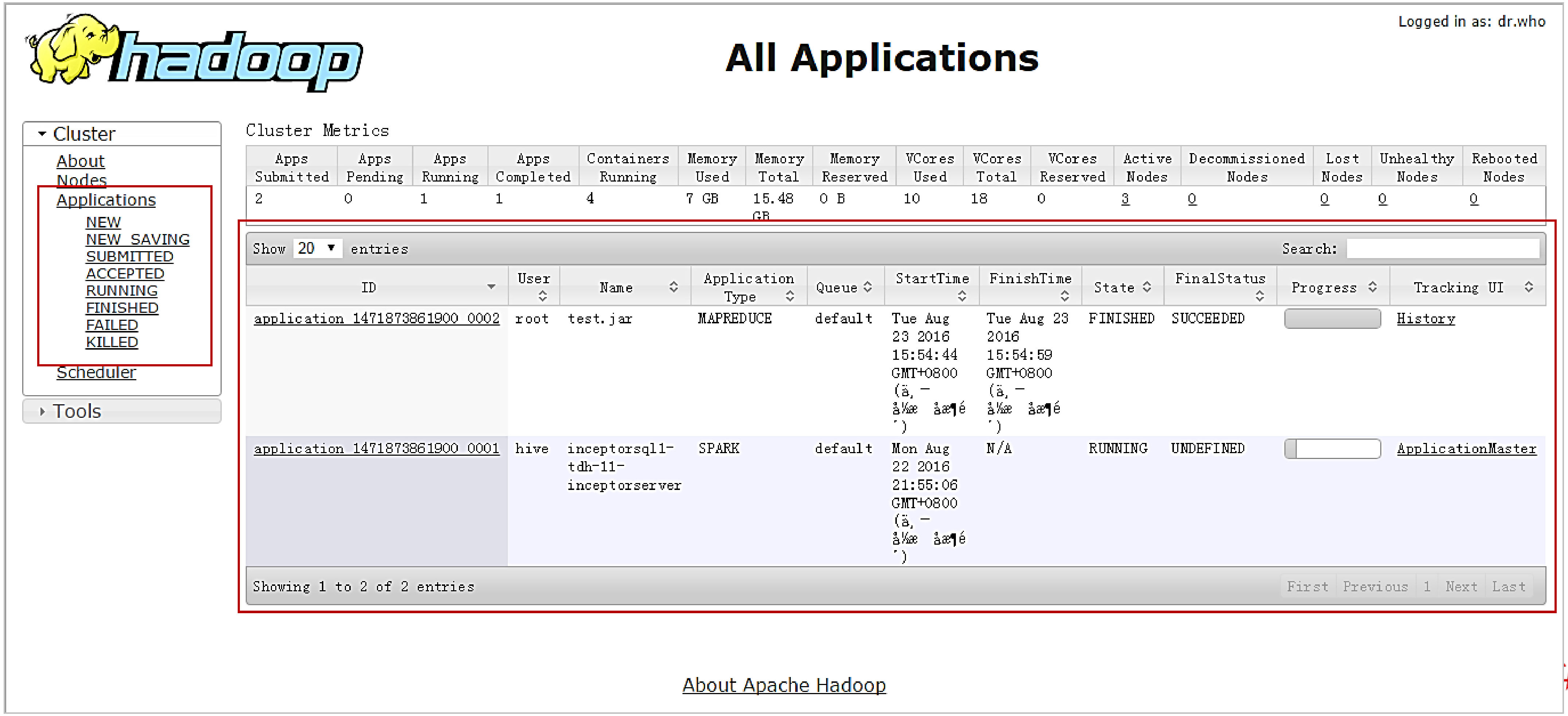


```
t3126poc4:~ # hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar pi 10 10
Number of Maps  = 10
Samples per Map = 10
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Wrote input for Map #8
Wrote input for Map #9
Starting Job
2016-05-10 14:09:58,250 INFO client.RMProxy: Connecting to ResourceManager at t3126poc5/172.16.2.85:8032
2016-05-10 14:09:58,834 INFO input.FileInputFormat: Total input paths to process : 10
2016-05-10 14:09:58,915 INFO mapreduce.JobSubmitter: number of splits:10
2016-05-10 14:09:59,188 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1462786145119_0002
2016-05-10 14:09:59,453 INFO impl.YarnClientImpl: Submitted application application_1462786145119_0002
2016-05-10 14:09:59,498 INFO mapreduce.Job: The url to track the job: http://t3126poc5:8088/proxy/application_1462786145119_0002/
2016-05-10 14:09:59,499 INFO mapreduce.Job: Running job: job_1462786145119_0002
2016-05-10 14:10:05,641 INFO mapreduce.Job: Job job_1462786145119_0002 running in uber mode : false
2016-05-10 14:10:05,644 INFO mapreduce.Job:  map 0% reduce 0%
```

1.3 作业管理

➢ 作业监控

- Web监控：http://{AM_IP}:8088/proxy/{application_id}/



Logged in as: dr.who

All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	1	1	4	7 GB	15.48 GB	0 B	10	18	0	3	0	0	0	0

Show 20 entries Search:

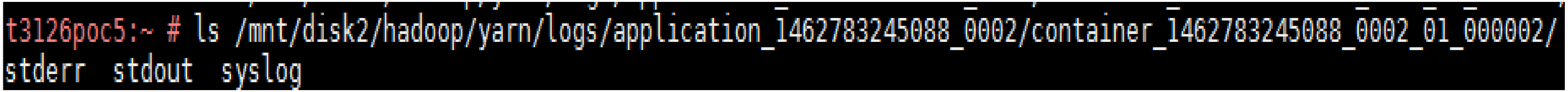
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1471873861900_0002	root	test.jar	MAPREDUCE	default	Tue Aug 23 2016 15:54:44 GMT+0800 (ä¸ – å¼æ å‡¶é ´)	Tue Aug 23 2016 15:54:59 GMT+0800 (ä¸ – å¼æ å‡¶é ´)	FINISHED	SUCCEEDED		History
application_1471873861900_0001	hive	inceptorsql1-tdh-11-inceptorserver	SPARK	default	Mon Aug 22 2016 21:55:06 GMT+0800 (ä¸ – å¼æ å‡¶é ´)	N/A	RUNNING	UNDEFINED		ApplicationMaster

Showing 1 to 2 of 2 entries

About Apache Hadoop

➢ 作业诊断

- 配置参数：yarn.nodemanager.log-dirs
  - MapReduce运行日志目录
  - 默认值为/mnt/disk*/hadoop/yarn/
- 根据运行出错信息，到指定节点下分析日志



```
t3126poc5:~ # ls /mnt/disk2/hadoop/yarn/logs/application_1462783245088_0002/container_1462783245088_0002_01_000002/
stderr  stdout  syslog
```

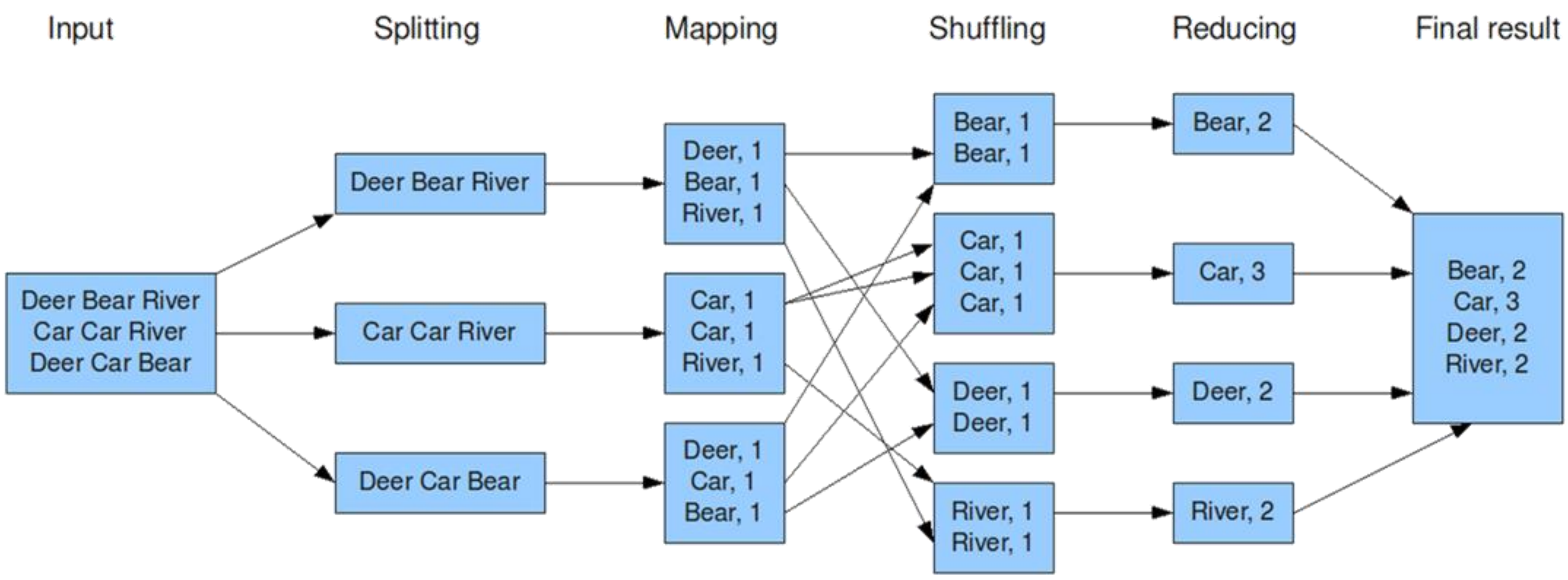
4

chapter

MapReduce 开发案例

MR开发案例：WordCount 词频统计

- WordCount：



MR开发案例：WordCount 词频统计

- 代码：

WordCountMapper

```
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] words = StringUtils.split(value.toString(), ' ');

        for (String word : words) {
            context.write(new Text(word), new IntWritable(1));
        }
    }
}
```

MR开发案例：WordCount 词频统计

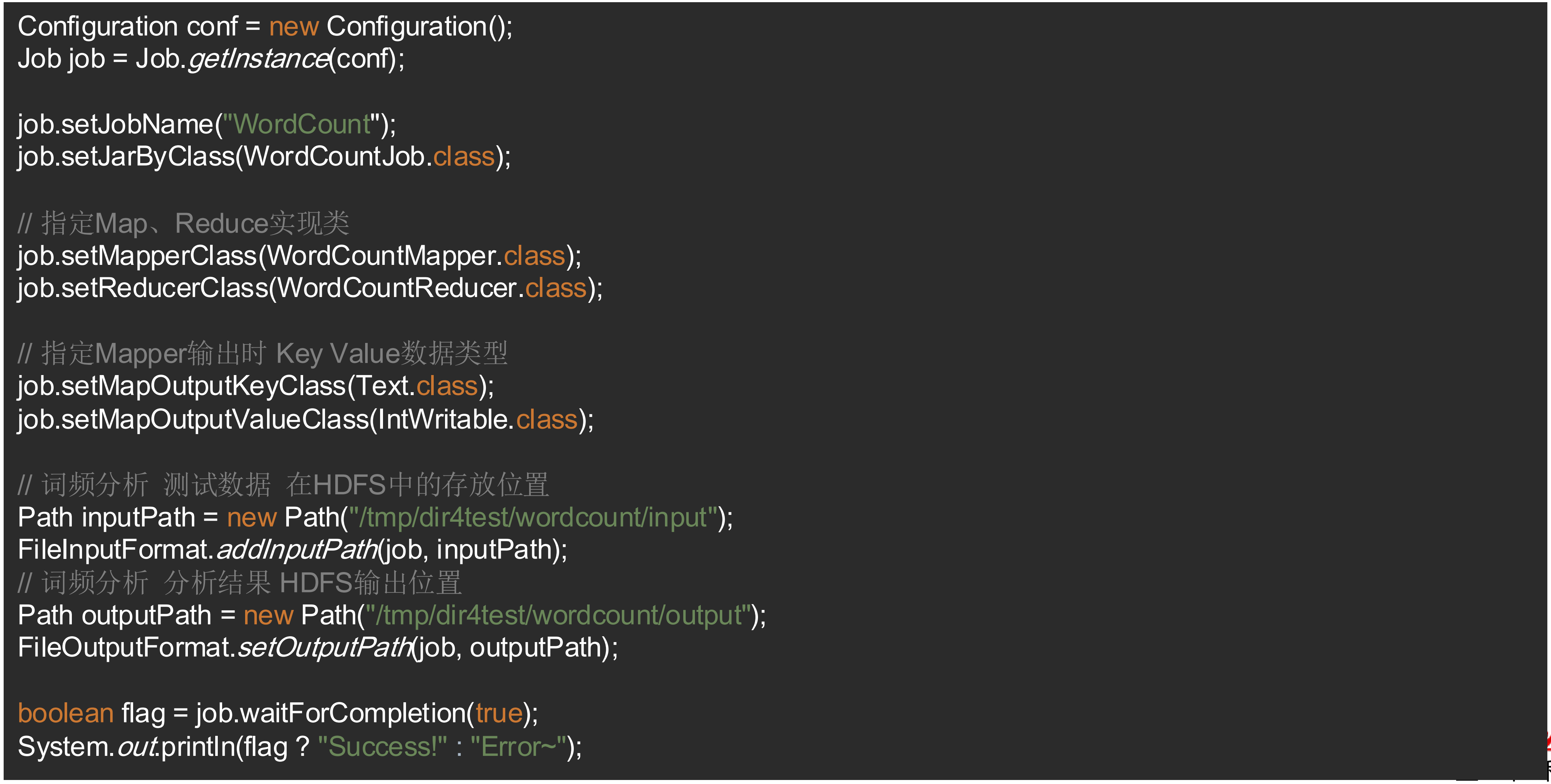
- 代码：

WordCountReducer

```
public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

        int sum = 0;

        for (IntWritable i : values) {
            sum = sum + i.get();
        }

        context.write(key, new IntWritable(sum));
    }
}
```

MR开发案例：WordCount 词频统计

➢ 代码：WordCountJob



```
Configuration conf = new Configuration();
Job job = Job.getInstance(conf);

job.setJobName("WordCount");
job.setJarByClass(WordCountJob.class);

// 指定Map、Reduce实现类
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);

// 指定Mapper输出时 Key Value数据类型
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

// 词频分析 测试数据 在HDFS中的存放位置
Path inputPath = new Path("/tmp/dir4test/wordcount/input");
FileInputFormat.addInputPath(job, inputPath);
// 词频分析 分析结果 HDFS输出位置
Path outputPath = new Path("/tmp/dir4test/wordcount/output");
FileOutputFormat.setOutputPath(job, outputPath);

boolean flag = job.waitForCompletion(true);
System.out.println(flag ? "Success!" : "Error~");
```

MR开发案例：WordCount 词频统计

➢ 代码：WordCountJob

```
Configuration conf = new Configuration();
Job job = Job.getInstance(conf);

job.setJobName("WordCount");
job.setJarByClass(WordCountJob.class);

// 指定Map、Reduce实现类
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);

// 指定Mapper输出时 Key Value数据类型
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

// 词频分析  测试数据  在HDFS中的存放位置
Path inputPath = new Path("/tmp/dir4test/wordcount/input");
FileInputFormat.addInputPath(job, inputPath);
// 词频分析  分析结果 HDFS输出位置
Path outputPath = new Path("/tmp/dir4test/wordcount/output");
FileOutputFormat.setOutputPath(job, outputPath);

boolean flag = job.waitForCompletion(true);
System.out.println(flag ? "Success!" : "Error~");
```

```
// Combiner
job.setCombinerClass(WordCountMapper.class);
```

MR开发案例：二次排序

➢ 二次排序：

- 需求：现有海量的历史天气数据，统计出不同月份中温度最高的日期。
- 数据：

```
1949-10-01 14:21:02        34c
1949-10-02 14:01:02        36c
1950-01-01 11:21:02        32c
1950-10-01 12:21:02        37c
1951-12-01 12:21:02        23c
1950-10-02 12:21:02        41c
1950-10-03 12:21:02        27c
1951-07-01 12:21:02        45c
1951-07-02 12:21:02        46c
1951-07-03 12:21:03        47c
... ...
```

MR开发案例：二次排序

- 代码：自定义组合Key

```
public class Weather implements WritableComparable<Weather> {

    private int year;
    private int month;
    private int day;
    private int wd;      // 温度

    public int getYear() {
        return year; }

    public void setYear(int year) {
        this.year = year; }

    // 部分set、get代码略...

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeInt(year);
        out.writeInt(month);
        out.writeInt(day);
        out.writeInt(wd);
    }
```

```
@Override
    public void readFields(DataInput in) throws IOException {
        this.year = in.readInt();
        this.month = in.readInt();
        this.day = in.readInt();
        this.wd = in.readInt();
    }

    @Override
    public int compareTo(Weather w) {
        int c1 = Integer.compare(this.year, w.getYear());
        if( c1 == 0 ) {
            int c2 = Integer.compare(this.month, w.getMonth());

            if( c2 == 0 ) {
                return Integer.compare(this.wd, w.getWd());
            }

            return c2;
        }

        return c1;
    }
```

MR开发案例：二次排序

- 代码：自定义Sort

```java
public class TQSort extends WritableComparator {

    public TQSort() {
        super(Weather.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        Weather w1 = (Weather)a;
        Weather w2 = (Weather)b;

        int c1 = Integer.compare(w1.getYear(), w2.getYear());

        if( c1 == 0) {
            int c2 = Integer.compare(w1.getMonth(), w2.getMonth());

            if( c2 == 0 ) {
                return -Integer.compare(w1.getWd(), w2.getWd());
            }
            return c2;
        }
        return c1;
    }
}
```

MR开发案例：二次排序

- 代码：自定义Group

```
public class TQGroup extends WritableComparator {

    public TQGroup() {
        super(Weather.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        Weather w1 = (Weather)a;
        Weather w2 = (Weather)b;

        int c1 = Integer.compare(w1.getYear(), w2.getYear());

        if( c1 == 0) {
            int c2 = Integer.compare(w1.getMonth(), w2.getMonth());
            return c2;
        }
        return c1;
    }
}
```

MR开发案例：二次排序

- 代码：自定义Partition

```
public class TQPartition extends HashPartitioner<Weather, IntWritable> {

    @Override
    public int getPartition(Weather key, IntWritable value, int numReduceTasks) {

        return (key.getYear() - 1949 ) % numReduceTasks;

        // return super.getPartition(key, value, numReduceTasks);
    }
}
```

MR开发案例：二次排序

➢ 代码：Mapper

```
public class TQMapper extends Mapper<LongWritable, Text, Weather, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context)  throws IOException, InterruptedException {

        String[] strs = StringUtils.split(value.toString(), '\t');

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Calendar cal = Calendar.getInstance();
        try {
            cal.setTime(sdf.parse(strs[0]));

            Weather w = new Weather();
            w.setYear(cal.get(Calendar.YEAR));
            w.setMonth(cal.get(Calendar.MONTH) + 1);
            w.setDay(cal.get(Calendar.DAY_OF_MONTH));

            int wd = Integer.parseInt(strs[1].substring(0, strs[1].lastIndexOf("c")));
            w.setWd(wd);

            context.write(w, new IntWritable(wd));

        } catch (ParseException e) {
            e.printStackTrace();    }    }    }
```

MR开发案例：二次排序

➢ 代码：Reducer

```
public class TQReducer extends Reducer<Weather, IntWritable, Text, NullWritable> {

    @Override
    protected void reduce(Weather weather, Iterable<IntWritable> iterable, Context context)
            throws IOException, InterruptedException {

        int flag = 0;
        for (IntWritable i : iterable) {
            flag++;
            if(flag > 2) {
                break;
            }
            String msg = weather.getYear() + "-" + weather.getMonth() + "-" + weather.getDay() +
                    "-" + i.get();

            context.write(new Text(msg), NullWritable.get());
        }
    }
}
```

MR开发案例：二次排序

- 代码：Job

```
Configuration conf = new Configuration();
Job job = Job.getInstance(conf);

job.setJobName("TQ sort");
job.setJarByClass(TQJob.class);

// 指定Map、Reduce实现类
job.setMapperClass(TQMapper.class);
job.setReducerClass(TQReducer.class);

// 指定Mapper输出时 Key Value数据类型
job.setMapOutputKeyClass(Weather.class);
job.setMapOutputValueClass(IntWritable.class);

// 指定自定义Partition、Sort、Group
job.setPartitionerClass(TQPartition.class);
job.setSortComparatorClass(TQSort.class);
job.setGroupingComparatorClass(TQGroup.class);

// 指定Reduce Task 数量
job.setNumReduceTasks(3);
```

```
// 二次排序 测试数据 在HDFS中的存放位置
Path inputPath = new Path("/tmp/dir4test/weather/input");
FileInputFormat.addInputPath(job, inputPath);
// 二次排序 分析结果 HDFS输出位置
Path outputPath = new Path("/tmp/dir4test/weather/output");
FileOutputFormat.setOutputPath(job, outputPath);

boolean flag = job.waitForCompletion(true);
System.out.println(flag ? "Success!" : "Error~");
```

Q&A

TRANSWARP
星 环 科 技

温故知新

- 简述MR Split与HDFS Block的关系。
- 为什么MapReduce要求输入输出必须是key-value键值对?
- 简述Shuffle的工作原理。
- 从编程模型的视角，MapReduce有哪些优缺点?
- 简述“哈希取模”在MapReduce中的作用。