MapReduce交流

技术平台部-研发组-范瑞

一. MapReduce介绍

Overview

Hadoop MapReduce是一个软件框架，用于轻松编写应用程序，以可靠、容错的方式在大型集群（数千个节点）的商业硬件上并行处理大量数据（多TB数据集）。

MapReduce 作业通常将输入数据集拆分为独立的块，这些块由map任务以完全并行的方式处理。框架对Map的输出进行排序，然后输入到Reduce任务，通常作业的输入和输出都存储在文件系统中。框架负责调度任务，监视它们并重新执行失败的任务。

Overview

通常，计算节点和存储节点是相同的，即MapReduce框架和Hadoop分布式文件系统（HDFS）在同一组节点上运行。此配置允许框架有效地在已存在数据的节点上调度任务，从而在集群中产生非常高的聚合带宽。

计算向数据移动

二. MapReduce计算过程

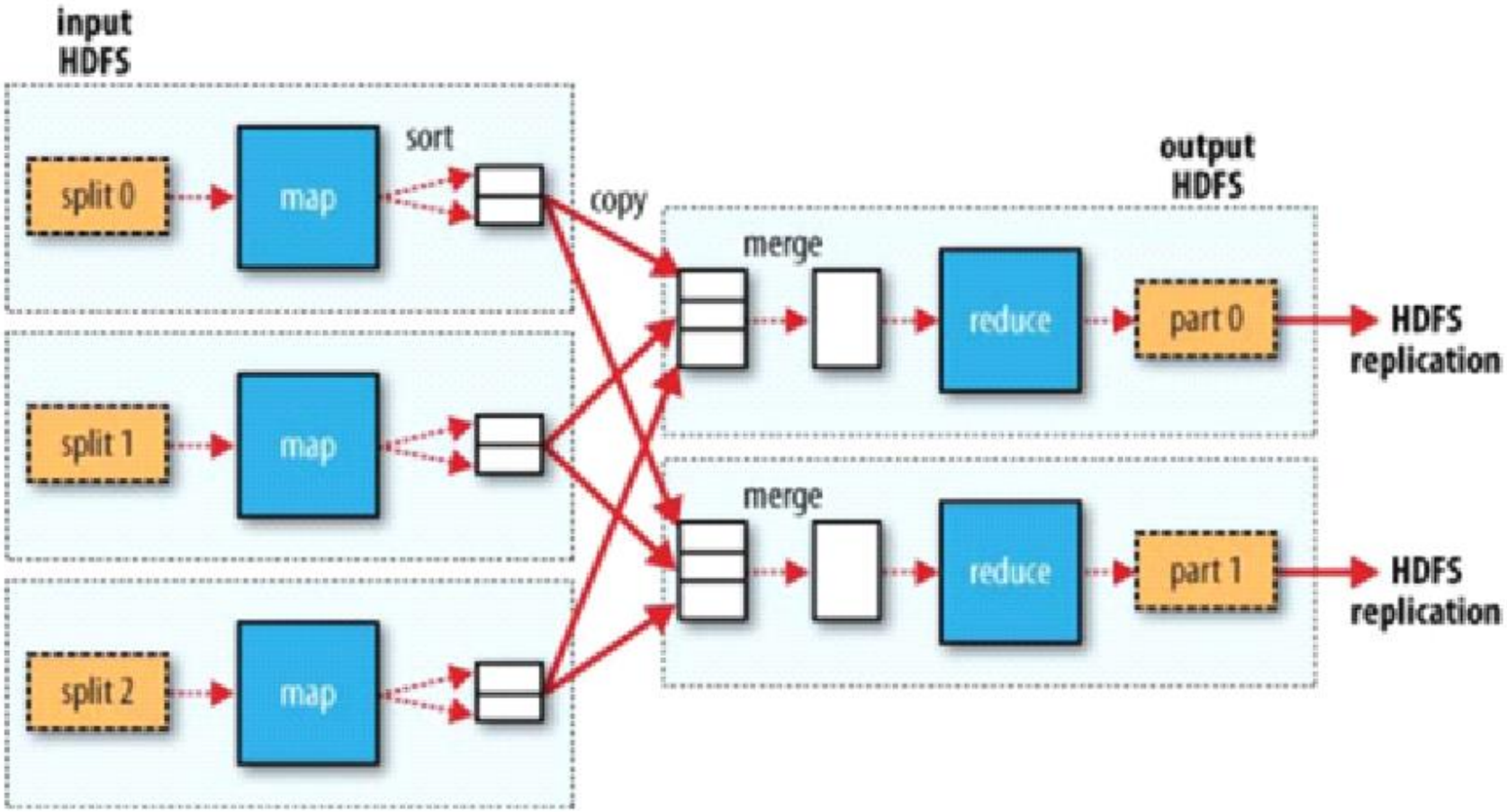
Overview

map将输入数据集映射成一个中间数据集（key，value），“相同”的key为一组，调用一次reduce方法，方法内迭代这一组数据进行计算

过程：

FileInputFormat + split + map + shuffle + reduce + FileOutputFormat

流程图


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
merge
reduce
part 0
output
HDFS
HDFS
replication
merge
reduce
part 1
HDFS
replication

竖直看，分为Map和Reduce，Map的输出作为Reduce的输入

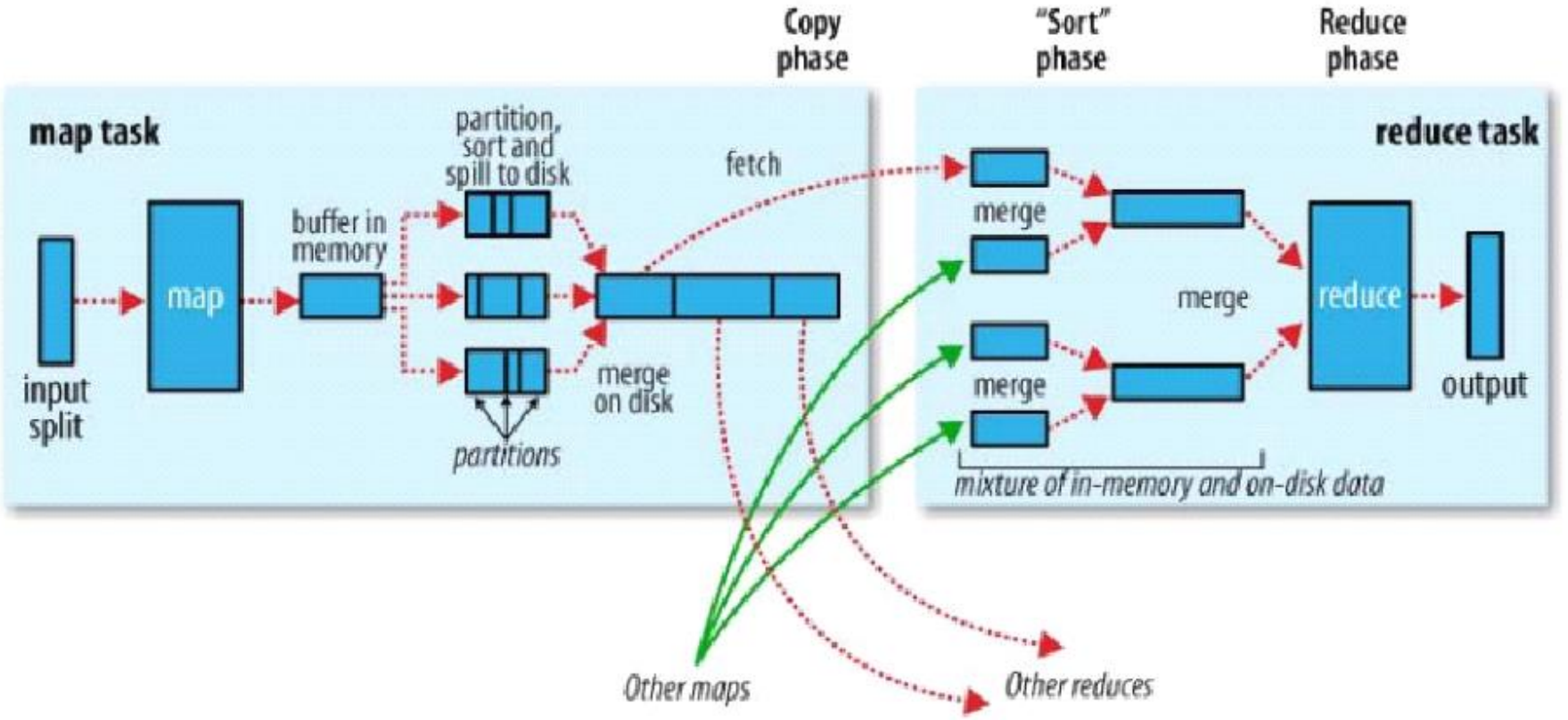
水平看，Map和Reduce并行计算，map分布在不同的节点，Reduce也分布在不同节点，数据需要网络传输（瓶颈）

并行度怎么决定？

Map的并行度由切片数量决定

Reduce的并行度由Key的分布由开发者合理设计

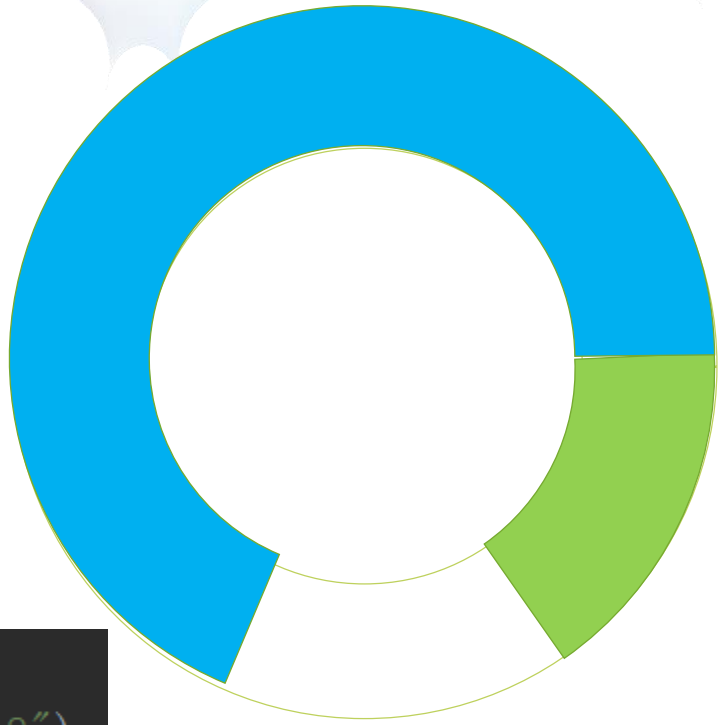
具体分析MapReduce流程



Buffer环形缓冲区

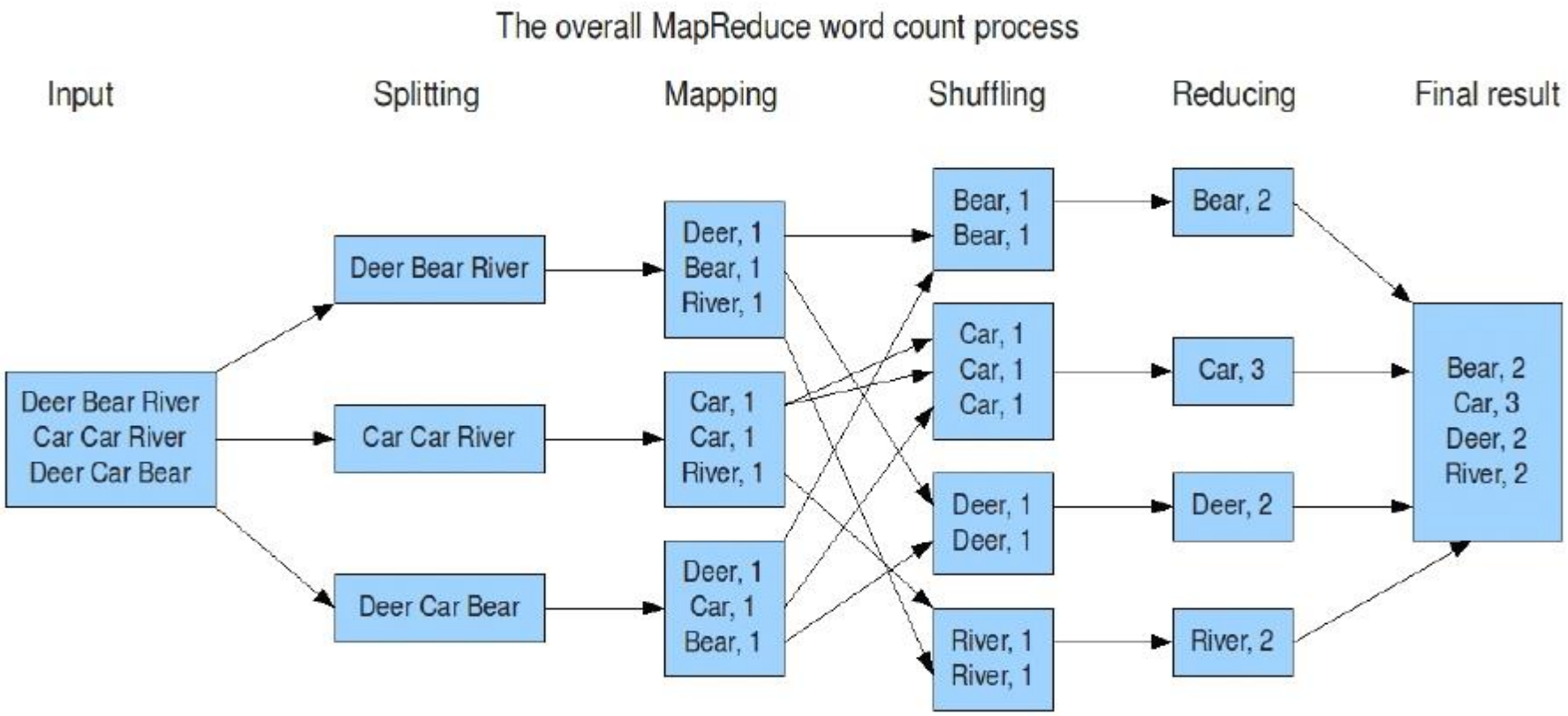
缓冲区大小默认为100M

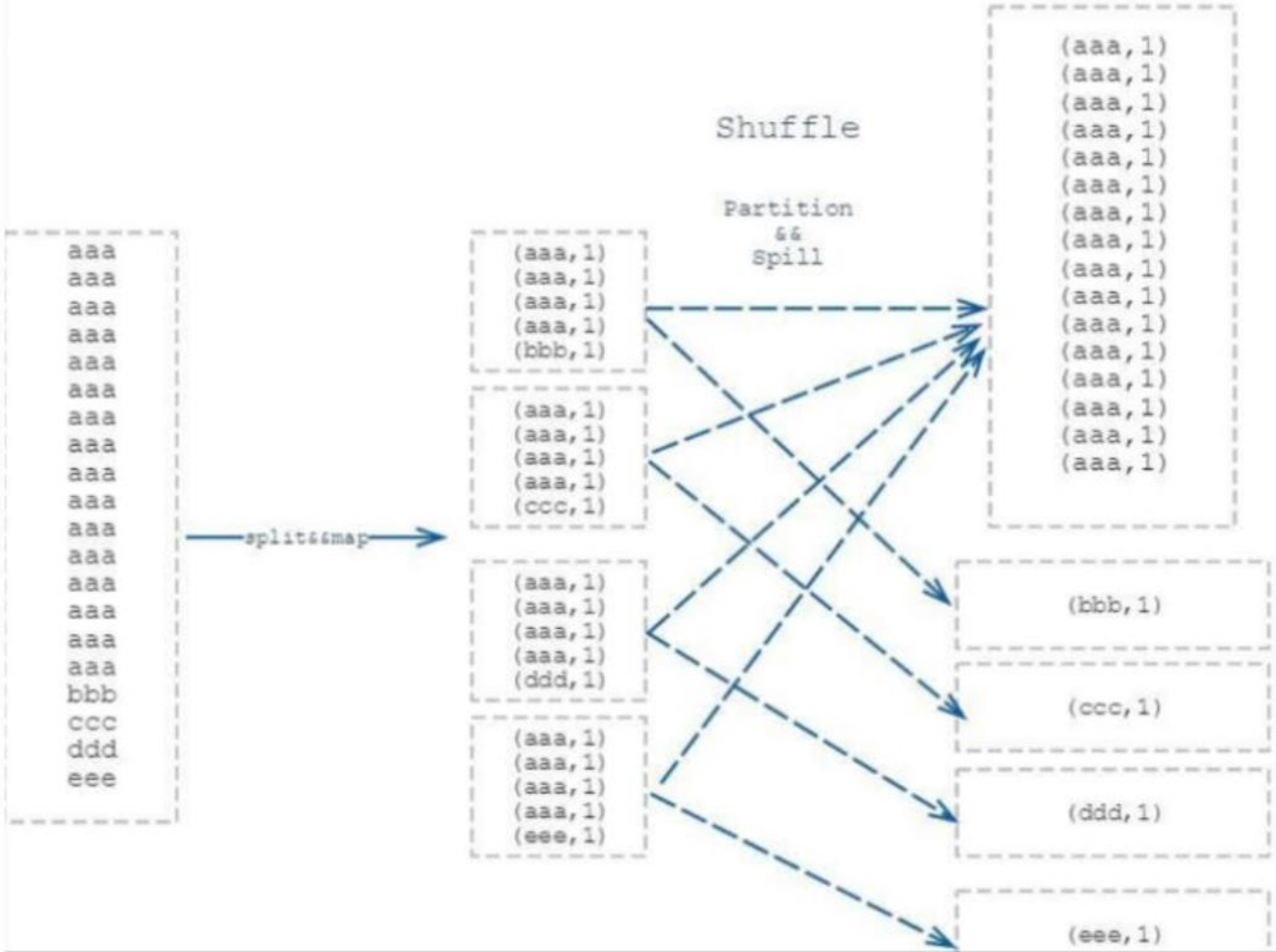
溢写阈值默认为80%



```
conf.set(MRJobConfig.IO_SORT_MB,"100");
conf.set(MRJobConfig.MAP_SORT_SPILL_PERCENT,"0.8");
```

Wordcount计算过程图




Shuffle
Partition
&&
Spill
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
aaa
bbb
ccc
ddd
eee
split&&map
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(bbb,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(ccc,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(ddd,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(eee,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(aaa,1)
(bbb,1)
(ccc,1)
(ddd,1)
(eee,1)

Overview

Map端不会倾斜，Reduce根据Key发生了倾斜，如何优化？

1、自定义分区器，根据经验值，将数据较少的key分到一个reduce端

2、在key上做手脚，把一个key变成多个key，最后再对结果进行汇总

3、设计合理的逻辑，尽量少的发生shuffle

Map

计算向数据移动

并行分布式

InputFormat生成的每个Split对应一个map任务

读懂数据，映射为KV模型

输出键值对不需要与输入键值对具有相同的类型

给定的输入键值对可以映射到零个或多个输出键值对

对Map的输出进行排序，根据Reduce进行分区

通过实现自定义分区器来控制哪些键到哪个Reduce

可以为Map指定Combiner（本地聚合，减少传输数据量）

控制Map的数量

Reduce

相同的Key汇聚到一个Reduce中
排序实现key的汇聚
相同的Key调用一次reduce方法
通过Job.setIntNumReduceTasks(int)来控制Reduce的数量，分区总数与作业的Reduce任务数相同
Reduce中可以包含不同的key
Reduce有三个阶段: shuffle、sort、reduce

K,V作为参数传递，可以使用自定义数据类型，但需要实现 Writable序列化、Comparable比较器接口

三. MR案例演示

需求分析

统计某天中华万年历Android、IOS，微鲤看看Android、IOS的PV事件数据量

数据存储在万年历集群HDFS的/user/hive/warehouse/test.db/tmp_ods_hive_pv_1d目录下

截取部分数据：

-rw-r--r--	impala	supergroup	973.76 MB	Mon Aug 06 20:44:21 +0800 2018	3	128 MB	ae4ed60f3073e5ce-2b769f4400000000_341990625_data.0.
-rw-r--r--	impala	supergroup	984.77 MB	Mon Aug 06 20:44:22 +0800 2018	3	128 MB	ae4ed60f3073e5ce-2b769f4400000001_966017313_data.0.
-rw-r--r--	impala	supergroup	1.02 GB	Mon Aug 06 20:44:24 +0800 2018	3	128 MB	ae4ed60f3073e5ce-2b769f4400000002_287103134_data.0.

File information - ae4ed60f3073e5ce-2b769f4400000000_341990625_data.0.

Download

Block information -- Block 0

Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7

Block ID: 1074614419

Block Pool ID: BP-34[illegible].119.116-1531360965913

Generation Stamp: 8[illegible]

Size: 134217728

Availability:

- dn137.hadoop.wnl.dmp.com
- dn139.hadoop.wnl.dmp.com
- dn133.hadoop.wnl.dmp.com

Close

数据格式如下：

日期,app_key,用户id

```
20180731,91988061,358fae054f2bec96d334ae6ecf92b01b
20180731,99817749,5ef7ab853740680555b62910ba3b6b31
20180731,91988061,d6b3d1168cc55aad3270879413c93edb
20180731,99817749,ca2e646f85c23656030b764e4d901158
20180731,91988061,8e90375fdbb75209c519d5ee5ec35b11
20180731,91988061,df5b6d51ec0bc02044e1bd3ce5831087
20180731,91988061,f0fa8c510abd46e0e1de99e76e151c02
20180731,91988061,db20375232ccda119ee0805cc64e6504
20180731,91988061,e7e6b9bb55ffcf3a12e541b0e679793b
```

MapReduce处理过程分析：

Map：

将app_key从数据源中解析出来，输出<app_key，1>的数据

Reduce：

拿到Map输出的一组Key，遍历value，将其累加即可

```
Job  job = Job.getInstance(conf); //job  作业
job.setJarByClass(WC.class);//参数为 当前类的 类名
job.setJobName("myjob");//给当前Job 取个名字
job.setNumReduceTasks(4);
  job.setCombinerClass( MyReducer.class );
  job.setNumReduceTasks(1);

Path input = new Path( pathString: "/user/hive/warehouse/test.db/tmp_ods_hive_pv_1d");//输入路径
FileInputFormat.addInputPath( job, input );
Path output = new Path( pathString: "/tmp/MR/pv");    //输出路径
if( output.getFileSystem(conf).exists(output) ) {        //如果路径存在,删除路径
    output.getFileSystem(conf).delete(output, recursive: true);
}
FileOutputFormat.setOutputPath( job, output );//为job设置输出路径

job.setMapperClass(MyMapper.class);     //设置map的类
job.setMapOutputKeyClass(Text.class);     //map的key输出类型
job.setMapOutputValueClass(IntWritable.class);//map的value输出类型
job.setReducerClass(MyReducer.class);     //设置reduce的类
//将以上所有的代码 提交给集群,等待 完成

job.waitForCompletion( verbose: true);
```

```java
package com.ssy.MR;

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

import java.io.IOException;

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


    public static final IntWritable one = new IntWritable( value: 1);
    private Text appKey = new Text();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        appKey.set( value.toString().split( regex: ",")[1] );
        context.write(appKey,one);
    }

}

```

```java
package com.ssy.MR;

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

import java.io.IOException;

public 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);

    }
}
```

代码一

查看yarn，看Map、Reduce执行个数、流程、时间，可以看到Reduce端从Map端拉取数据，然后进行排序，最后调用reduce方法

Job Overview

Job Name:	myjob
State:	RUNNING
Uberized:	false
Started:	Tue Aug 07 22:49:52 CST 2018
Elapsed:	2mins, 35sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Aug 07 22:49:49 CST 2018	dn132.hadoop.wnl.dmp.com:8042	logs

Task Type	Progress	Total	Pending	Running	Complete
Map		322	0	0	322
Reduce		4	0	1	3

Attempt Type	New	Running	Failed	Killed	Successful
Maps	0	0	0	0	322
Reduces	0	1	0	0	3

代码一

Task	Progress	Status	State	Start Time	Finish Time	Elapsed Time
955791_0177_m_000032		map	SUCCEEDED	Tue Aug 7 22:49:54 +0800 2018	Tue Aug 7 22:50:08 +0800 2018	13sec
955791_0177_m_000089		map	SUCCEEDED	Tue Aug 7 22:49:54 +0800 2018	Tue Aug 7 22:50:08 +0800 2018	13sec
955791_0177_m_000086		map	SUCCEEDED	Tue Aug 7 22:49:54	Tue Aug 7 22:50:08 +0800	13sec

Show 20 entries Search:

Task	Progress	Status	State	Start Time	Finish Time	Elapsed Time
task_1532128955791_0177_r_000003		reduce > copy task(attempt_1532128955791_0177_m_000246_0 succeeded at 25.49 MB/s) Aggregated copy rate(201 of 322 at 67.72 MB/s)	RUNNING	Tue Aug 7 22:50:10 +0800 2018	N/A	1mins, 30sec
task_1532128955791_0177_r_000001		reduce > reduce	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:51:08 +0800 2018	58sec
task_1532128955791_0177_r_000000		reduce > reduce	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:37 +0800 2018	27sec
task_1532128955791_0177_r_000002		reduce > reduce	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:13 +0800 2018	2sec

Showing 1 to 4 of 4 entries First Previous 1 Next Last

Show 20 entries Search:

Task	Progress	Status	State	Start Time	Finish Time	Elapsed Time
task_1532128955791_0177_r_000003		reduce > sort	RUNNING	Tue Aug 7 22:50:10 +0800 2018	N/A	2mins, 52sec
task_1532128955791_0177_r_000001		reduce > reduce	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:51:08 +0800 2018	58sec
task_1532128955791_0177_r_000000		reduce > reduce	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:37 +0800 2018	27sec
task_1532128955791_0177_r_000002		reduce > reduce	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:13 +0800 2018	2sec

Showing 1 to 4 of 4 entries First Previous 1 Next Last

Overview

Task					Successful Attempt							
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time	Finish Time	Elapsed Time Shuffle	Elapsed Time Merge	Elapsed Time Reduce	Elapsed Time
task_1532128955791_0177_r_000003	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:56:54 +0800 2018	6mins, 44sec	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:52:44 +0800 2018	Tue Aug 7 22:53:03 +0800 2018	Tue Aug 7 22:56:54 +0800 2018	2mins, 33sec	19sec	3mins, 50sec	6mins, 44sec
task_1532128955791_0177_r_000001	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:51:08 +0800 2018	58sec	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:13 +0800 2018	Tue Aug 7 22:50:41 +0800 2018	Tue Aug 7 22:51:08 +0800 2018	2sec	28sec	27sec	58sec
task_1532128955791_0177_r_000000	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:37 +0800 2018	27sec	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:12 +0800 2018	Tue Aug 7 22:50:25 +0800 2018	Tue Aug 7 22:50:37 +0800 2018	2sec	12sec	12sec	27sec
task_1532128955791_0177_r_000002	SUCCEEDED	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:13 +0800 2018	2sec	Tue Aug 7 22:50:10 +0800 2018	Tue Aug 7 22:50:12 +0800 2018	Tue Aug 7 22:50:12 +0800 2018	Tue Aug 7 22:50:13 +0800 2018	2sec	0sec	0sec	2sec

Job Name:	myjob
User Name:	root
Queue:	root.users.root
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Aug 07 22:49:47 CST 2018
Started:	Tue Aug 07 22:49:52 CST 2018
Finished:	Tue Aug 07 22:56:54 CST 2018
Elapsed:	7mins, 2sec
Diagnostics:	
Average Map Time	12sec
Average Shuffle Time	40sec
Average Merge Time	15sec
Average Reduce Time	1mins, 7sec

代码一

/tmp/MR/pv Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 22:56:54 +0800 2018	3	128 MB	_SUCCESS
-rw-r--r--	root	supergroup	18 B	Tue Aug 07 22:50:37 +0800 2018	3	128 MB	part-r-00000
-rw-r--r--	root	supergroup	18 B	Tue Aug 07 22:51:08 +0800 2018	3	128 MB	part-r-00001
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 22:50:12 +0800 2018	3	128 MB	part-r-00002
-rw-r--r--	root	supergroup	38 B	Tue Aug 07 22:56:54 +0800 2018	3	128 MB	part-r-00003

```
[root@dn131.hadoop.wnl.dmp.com ~]$hdfs dfs -cat /tmp/MR/pv/part-r-00000
91988062        35444941
[root@dn131.hadoop.wnl.dmp.com ~]$hdfs dfs -cat /tmp/MR/pv/part-r-00001
99817882        79249797
[root@dn131.hadoop.wnl.dmp.com ~]$hdfs dfs -cat /tmp/MR/pv/part-r-00002
[root@dn131.hadoop.wnl.dmp.com ~]$hdfs dfs -cat /tmp/MR/pv/part-r-00003
91988061        471126173
99817749        210773513
```

代码一

结论：

执行时间大概六七分钟，时间耗在了Reduce阶段，原因是微鲤看看和万年历安卓数据量较大，而且恰好分到了一个分区，出现了数据倾斜，由一个ReduceTask去处理大量数据

当Reduce个数大于Map输出的Key的个数时，是没有意义的，因为相同的Key为一组，调用一次reduce方法，所以相同的Key必须在同一个reduce上。目前就四种key，所以最多4个Reduce上有数据。应设计合理的分区器

代码二-增加Reduce数量

既然Key只有四个，相同的Key必须到达一个Reduce端，那么还想提高Reduce处理速度怎么办？

简单的增加ReduceTask数量没用

那怎么办？

把Key的数量变多就可以了

代码二-加前缀增加Key的数量

```
public class MyMapper extends Mapper<Object, Text, Text, IntWritable> {


    public static final IntWritable one = new IntWritable( value: 1);
    private Text appKey = new Text();
    private String appStr;

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        appStr = value.toString().split( regex: ",")[1];
        if ( "91988061".equals( appStr ) ) {
            appKey.set( appStr + (int)(MyPartitioner.WEILI_ANDROID_REDUCE_NUM*Math.random()) );
        } else if ( "99817749".equals( appStr ) ) {
            appKey.set( appStr + (int)(MyPartitioner.ZHWNL_ANDROID_REDUCE_NUM*Math.random()) );
        } else if ( "91988062".equals( appStr ) ) {
            appKey.set( appStr + (int)(MyPartitioner.WEILI_IOS_REDUCE_NUM*Math.random()) );
        } else {
            appKey.set( appStr + (int)(MyPartitioner.ZHWNL_IOS_REDUCE_NUM*Math.random()) );
        }
        context.write(appKey,one);
    }
}
```

代码二-加前缀增加Key的数量

往app_key上加前缀，使得Map输出的Key不同，那么就会分到不同的Reduce上。初步将安卓的数据分9个Reduce，ios分2个Reduce。

Show 20 entries Search:

Task					Successful Attempt							
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time	Finish Time	Elapsed Time Shuffle	Elapsed Time Merge	Elapsed Time Reduce	Elapsed Time
task_1532128955791_0178_r_000003	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:20:21 +0800 2018	1mins, 19sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:26 +0800 2018	Tue Aug 7 23:19:39 +0800 2018	Tue Aug 7 23:20:21 +0800 2018	24sec	13sec	41sec	1mins, 19sec
task_1532128955791_0178_r_000004	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:20:17 +0800 2018	1mins, 15sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:26 +0800 2018	Tue Aug 7 23:19:38 +0800 2018	Tue Aug 7 23:20:17 +0800 2018	24sec	12sec	38sec	1mins, 15sec
task_1532128955791_0178_r_000006	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:57 +0800 2018	55sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:08 +0800 2018	Tue Aug 7 23:19:27 +0800 2018	Tue Aug 7 23:19:57 +0800 2018	6sec	18sec	29sec	55sec
task_1532128955791_0178_r_000008	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:55 +0800 2018	53sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:08 +0800 2018	Tue Aug 7 23:19:27 +0800 2018	Tue Aug 7 23:19:55 +0800 2018	6sec	19sec	27sec	53sec

代码二-加前缀增加Key的数量

Job Name:	prefix
User Name:	root
Queue:	root.users.root
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Aug 07 23:18:38 CST 2018
Started:	Tue Aug 07 23:18:42 CST 2018
Finished:	Tue Aug 07 23:20:21 CST 2018
Elapsed:	1mins, 38sec
Diagnostics:	
Average Map Time	13sec
Average Shuffle Time	7sec
Average Merge Time	7sec
Average Reduce Time	13sec

代码二-加前缀增加Key的数量

task_1532128955791_0178_r_000020	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	2sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	2sec	0sec	0sec	2sec
task_1532128955791_0178_r_000021	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	2sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	Tue Aug 7 23:19:04 +0800 2018	2sec	0sec	0sec	2sec
task_1532128955791_0178_r_000015	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000018	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000014	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000019	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000016	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000017	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000013	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000012	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	5sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:06 +0800 2018	Tue Aug 7 23:19:07 +0800 2018	4sec	0sec	0sec	5sec
task_1532128955791_0178_r_000000	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:08 +0800 2018	6sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:08 +0800 2018	Tue Aug 7 23:19:08 +0800 2018	Tue Aug 7 23:19:08 +0800 2018	6sec	0sec	0sec	6sec
task_1532128955791_0178_r_000011	SUCCEEDED	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:33 +0800 2018	31sec	Tue Aug 7 23:19:01 +0800 2018	Tue Aug 7 23:19:08 +0800 2018	Tue Aug 7 23:19:19 +0800 2018	Tue Aug 7 23:19:33 +0800 2018	6sec	11sec	14sec	31sec

-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:08 +0800 2018	3	128 MB	part-r-00000
-rw-r--r--	root	supergroup	18 B	Tue Aug 07 23:19:43 +0800 2018	3	128 MB	part-r-00001
-rw-r--r--	root	supergroup	18 B	Tue Aug 07 23:19:41 +0800 2018	3	128 MB	part-r-00002
-rw-r--r--	root	supergroup	54 B	Tue Aug 07 23:20:21 +0800 2018	3	128 MB	part-r-00003
-rw-r--r--	root	supergroup	54 B	Tue Aug 07 23:20:17 +0800 2018	3	128 MB	part-r-00004
-rw-r--r--	root	supergroup	36 B	Tue Aug 07 23:19:53 +0800 2018	3	128 MB	part-r-00005
-rw-r--r--	root	supergroup	36 B	Tue Aug 07 23:19:57 +0800 2018	3	128 MB	part-r-00006
-rw-r--r--	root	supergroup	36 B	Tue Aug 07 23:19:54 +0800 2018	3	128 MB	part-r-00007
-rw-r--r--	root	supergroup	36 B	Tue Aug 07 23:19:55 +0800 2018	3	128 MB	part-r-00008
-rw-r--r--	root	supergroup	36 B	Tue Aug 07 23:19:53 +0800 2018	3	128 MB	part-r-00009
-rw-r--r--	root	supergroup	36 B	Tue Aug 07 23:19:33 +0800 2018	3	128 MB	part-r-00010
-rw-r--r--	root	supergroup	36 B	Tue Aug 07 23:19:33 +0800 2018	3	128 MB	part-r-00011
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:07 +0800 2018	3	128 MB	part-r-00012
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:07 +0800 2018	3	128 MB	part-r-00013
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:06 +0800 2018	3	128 MB	part-r-00014
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:06 +0800 2018	3	128 MB	part-r-00015
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:07 +0800 2018	3	128 MB	part-r-00016
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:07 +0800 2018	3	128 MB	part-r-00017
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:06 +0800 2018	3	128 MB	part-r-00018
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:07 +0800 2018	3	128 MB	part-r-00019
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:04 +0800 2018	3	128 MB	part-r-00020
-rw-r--r--	root	supergroup	0 B	Tue Aug 07 23:19:04 +0800 2018	3	128 MB	part-r-00021

代码二-加前缀增加Key的数量

看yarn上，多个reduce马上结束任务，看最后的结果目录，多个文件为空文件

结论：

时间明显缩短，而且还存在大量不同的key分到了相同分区的现象。默认使用的HashPartition取模，决定Map输出对应的Key分配到哪个Reduce。取模可能相等，所以可能会存在Map输出的不同的Key分配到同一个Reduce端，可以自定义RangePartition分区器。让每个Reduce端只对应一个key，防止某个Reduce端数据过多

代码三-自定义分区器

```
job.setPartitionerClass(MyPartitioner.class);    //设置分区器
```

```
public int getPartition(Text key, IntWritable value, int numPartitions) {
    //91988061 ANDROID
    //中华万年历Android 99817749
    //91988062 IOS
    //中华万年历iPhone  99817882
    offset = Integer.parseInt( key.toString().substring(8) );
    appStr = key.toString().substring(0,8);
    if( "91988061".equals( appStr ) ) { //微鲤看看安卓
        return offset;
    } else if ( "99817749".equals( appStr ) ) {//万年历安卓
        return WEILI_ANDROID_REDUCE_NUM + offset;
    } else if ( "91988062".equals( appStr ) ) {
        return WEILI_ANDROID_REDUCE_NUM + ZHWNL_ANDROID_REDUCE_NUM
                + offset;
    } else {
        return WEILI_ANDROID_REDUCE_NUM + ZHWNL_ANDROID_REDUCE_NUM
                + WEILI_IOS_REDUCE_NUM + offset;
    }
}
```

代码三-自定义分区器

看yarn上，没有reduce马上执行完成的现象，但是Reduce执行时长差距还是较大

在yarn上看哪些reduce执行慢，看分区号，比较集中

结论：

分区器设定不均匀，导致每个reduce处理的数据还是不均，问题来源主要在于某些应用对应的数据量与分配的reduce数量不成比例造成的，解决了有些分区没有数据的问题

代码四-重新调整分区

每个app每天数据量波动不是很大，所以可以参考最近的数据，将微鲤看看安卓的分区调整为25，万年历安卓11，万年历ios调整为4，微鲤看看ios仍然为2

现象：

基本每个reduce处理的时长差距不大了。

代码四-重新调整分区

结论：

所有的Reduce任务时间差距较小，总时间也缩短了。

如果微鲤看看后期数据量增长过快，而其他应用数据量增长相对较慢，可以只调节微鲤看看的Reduce数量。Reduce的数量也不是越多也好，越多占用CPU核心数较多，可能阻塞，也不见得快多少

代码五-调节Map数量大小

如何调节Map数量大小？

How Many Maps?

The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files.

The right level of parallelism for maps seems to be around 10-100 maps per-node, although it has been set up to 300 maps for very cpu-light map tasks. Task setup takes a while, so it is best if the maps take at least a minute to execute.

Thus, if you expect 10TB of input data and have a blocksize of 128MB, you'll end up with 82,000 maps, unless Configuration.set(MRJobConfig.NUM_MAPS, int) (which only provides a hint to the framework) is used to set it even higher.

看源码：

```
// Create the splits for the job
LOG.debug("Creating splits at " + jtFs.makeQualified(submitJobDir));
int maps = writeSplits(job, submitJobDir);
conf.setInt(MRJobConfig.NUM_MAPS, maps);
LOG.info("number of splits:" + maps);
```

代码五-调节map大小

源码得知：

一个split对应一个map

默认一个block对应一个split

如何控制map的数量？（控制split数量）

```
conf.setLong(FileInputFormat.SPLIT_MAXSIZE, value: 67108464);    //64M
  conf.setLong(FileInputFormat.SPLIT_MINSIZE,268433856); //256M
```

```
FileInputFormat.setMaxInputSplitSize(job, size);
FileInputFormat.setMinInputSplitSize(job, size);
```

代码五-调节map大小

将Map的数量调多一倍，注意任务的状态，看到了一些任务处于Pending，所以总的时间没有提升。

结论：

Map的数量也不是越多越好，大数据要计算向数据移动，如果集群中机器不是很多，文件也不是很多，那么Map的数据过多，就会出现大量阻塞，如果机器数量足够，map的数量也较多，但是数据集中在少量的节点上，会存在通过网络进行数据拉取，影响效率

代码六-Combiner

Users can optionally specify a combiner, via Job.setCombinerClass(Class), to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.

在Map端完成一次聚合

代码六-Combiner

```
/**
 * Set the combiner class for the job.
 * @param cls the combiner to use
 * @throws IllegalStateException if the job is submitted
 */
public void setCombinerClass(Class<? extends Reducer> cls
                             ) throws IllegalStateException {
  ensureState(JobState.DEFINE);
  conf.setClass(COMBINE_CLASS_ATTR, cls, Reducer.class);
}
```

需要一个继承了Reducer的类

```
job.setCombinerClass( MyReducer.class );
job.setNumReduceTasks(1);
```

四．扩展案例

Hive的join操作用MR实现

join操作如何实现？

1、reduce端join

2、map端join

```
/**
 * Called once at the beginning of the task.
 */
protected void setup(Context context
                     ) throws IOException, InterruptedException {
  // NOTHING
}
```

求每月温度最高的两天

“

1949-10-01 14:21:02 34c
1949-10-01 19:21:02 38c
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

求每月温度最高的两天

区分一下 分区 和 分组的概念。

分区表示一个Reduce端。

分组表示相同的key，调用一次reduce方法。

一个Reduce端可以包含多个key，一个分区内可以包含多个分组（千万别把相同key的数据分到不同的Reduce端）。

求每月温度最高的两天

区分一下分组比较器和排序比较器的概念。

通过分组比较器来区分相同的key。

通过排序比较器对同一个Reduce端要拉取得数据进行排序。

必须保障相同分组的数据必须连续。

排序比较器什么时候使用？

当Key对应的所有数据按照某种规则排序后有助于提高后续业务的效率时即可使用。


THANKS