我们只写三级和四级标题

知识点是三级标题（HDFS）

题目是四级标题

## 4Hadoop

### 4.1HDFS

##### 4.1.1x

##### 4.1.2x

##### 4.1.2x

### 4.2MapReduce

#### 介绍 MadpReduce 整个过程，比如把 WordCount 的例子的细节将清楚（重点讲解 Shuffle）？

public class WordCount {

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

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

@Override

protected 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);

}

}

}

public static class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

private IntWritable result = new IntWritable();

@Override

protected 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);

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(MyMapper.class);

job.setCombinerClass(MyReducer.class);

job.setReducerClass(MyReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true)?0:1);

}

}

#### 什么是mapreduce

//TODO

#### mapreduce中常见的组件有哪些？

//TODO

#### mr启动过程

#### 怎样决定一个job的map和reduce的数量

#### Shffule过程中排序用的什么算法

如果Reduce个数和分区数不一致时，会发生什么

能否说一下MR的join？

方案1：Reduce端join算法实现

实现思路：

①将关联的条件作为map输出的key

②将两表满足join条件的数据并携带数据所来源的文件信息，发往同一个Reduce Task

③在Reduce中进行数据的串联

缺点：

①Join的操作是在Reduce阶段完成

②Reduce阶段处理压力太大

③Map节点的运算负载很低，资源利用率不高

④在Reduce阶段极易产生数据倾斜

方案2：Map端join算法实现

实现思路：

①将小表分发到所有Map。

②Map节点就可以在本地对自己读到的大表的数据进行join。

③输出最终的结果。

适用场景：适用于关联表中有小表的情形。

优点：可以大大提高join操作的并发度，加快处理速度。

说一下Combiner有什么作用？

MapReduce中的Combiner：

①是为了避免MapTask和ReduceTask之间的数据传输而设置的。

②Hadoop允许针对Map Task的输出指定一个合并函数。

③Combiner为了减少传输到Reduce阶段的数据量

减少了Mapper阶段的输出，从而减少网络带宽和Reduce阶段的负载

MapReduce运行原理

离线计算框架，过程分为split map shuffle reduce四个过程

架构节点有：Jobtracker TaskTracker

Split将文件分割，传输到mapper，mapper接收KV形式的数据，经过处理，再传到shuffle过程。

Shuffle先进行HashPartition或者自定义的partition，会有数据倾斜和reduce的负载均衡问题；再进行排序，默认按字典排序；为减少mapper输出数据，再根据key进行合并，相同key的数据value会被合并；最后分组形成（key,value{}）形式的数据，输出到下一阶段

Reduce输入的数据就变成了，key+迭代器形式的数据，再进行处理

用mr设计一个分组排重计数算法

输入文件格式:二级域名,一级频道,二级频道,访问ip地址,访问者id

需求:按照二级域名,一级频道,二级频道分组,计算pageview数,计算独立ip数和独立访问者id数

#### HADOOP的一个job提交到resourcemanager之后，会生成什么容器来放这个job

#### Hadoop用的什么版本？他们公司用的是商业版

#### 两个单词文件求交集MR代码

public int[] intersect(int[] nums1, int[] nums2) {

int[] news = new int[0];

if (nums1 == null || nums2 == null) {

return null;

}

if (nums1.length == 0 || nums2.length == 0) {

return news;

}

news = new int[nums1.length];

int[] count = new int[nums1.length];

int index = 0;

for (int i = 0; i < nums2.length; i++) {

for (int j = 0; j < nums1.length; j++) {

if (nums1[j] == nums2[i] && count[j] == 0) {

news[index] = nums1[j];

index++;

count[j] = 1;

break;

}

}

}

int[] ints = new int[index];

for (int i = 0; i < ints.length; i++) {

ints[i] = news[i];

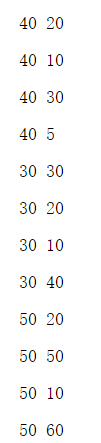
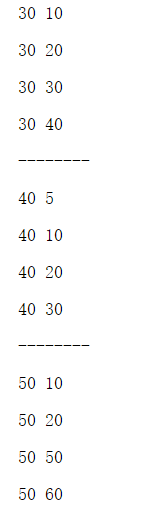
}

return ints;

}

#### 二次排序

sort.txt内容: 输出:

public class SecondarySort {

public static class Map extends Mapper<LongWritable, Text, IntPair, IntWritable> {

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

int left = 0;

int right = 0;

if (tokenizer.hasMoreTokens()) {

left = Integer.parseInt(tokenizer.nextToken());

if (tokenizer.hasMoreTokens())

right = Integer.parseInt(tokenizer.nextToken());

context.write(new IntPair(left, right), new IntWritable(right));

}

}

}

/\*

\* 自定义分区函数类FirstPartitioner，根据 IntPair中的first实现分区

\*/

public static class FirstPartitioner extends Partitioner<IntPair, IntWritable>{

@Override

public int getPartition(IntPair key, IntWritable value,int numPartitions){

return Math.abs(key.getFirst() \* 127) % numPartitions;

}

}

/\*

\* 自定义GroupingComparator类，实现分区内的数据分组

\*/

@SuppressWarnings("rawtypes")

public static class GroupingComparator extends WritableComparator{

protected GroupingComparator(){

super(IntPair.class, true);

}

@Override

public int compare(WritableComparable w1, WritableComparable w2){

IntPair ip1 = (IntPair) w1;

IntPair ip2 = (IntPair) w2;

int l = ip1.getFirst();

int r = ip2.getFirst();

return l == r ? 0 : (l < r ? -1 : 1);

}

}

public static class Reduce extends Reducer<IntPair, IntWritable, Text, IntWritable> {

public void reduce(IntPair key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

for (IntWritable val : values) {

context.write(new Text(Integer.toString(key.getFirst())), val);

}

}

}

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

// 读取配置文件

Configuration conf = new Configuration();

// 判断路径是否存在，如果存在，则删除

Path mypath = new Path(args[1]);

FileSystem hdfs = mypath.getFileSystem(conf);

if (hdfs.isDirectory(mypath)) {

hdfs.delete(mypath, true);

}

Job job = new Job(conf, "secondarysort");

// 设置主类

job.setJarByClass(SecondarySort.class);

// 输入路径

FileInputFormat.setInputPaths(job, new Path(args[0]));

// 输出路径

FileOutputFormat.setOutputPath(job, new Path(args[1]));

// Mapper

job.setMapperClass(Map.class);

// Reducer

job.setReducerClass(Reduce.class);

// 分区函数

job.setPartitionerClass(FirstPartitioner.class);

// 本示例并没有自定义SortComparator，而是使用IntPair中compareTo方法进行排序 job.setSortComparatorClass();

// 分组函数

job.setGroupingComparatorClass(GroupingComparator.class);

// map输出key类型

job.setMapOutputKeyClass(IntPair.class);

// map输出value类型

job.setMapOutputValueClass(IntWritable.class);

// reduce输出key类型

job.setOutputKeyClass(Text.class);

// reduce输出value类型

job.setOutputValueClass(IntWritable.class);

// 输入格式

job.setInputFormatClass(TextInputFormat.class);

// 输出格式

job.setOutputFormatClass(TextOutputFormat.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

#### 给定两张表，手写MR进行join操作

表一：NAME，SEX 表二：NAME，AGE

public class MapJoin {

public static class MyMapper extends Mapper<LongWritable, Text, Text, Text>{

Map<String,String> tableA = new ConcurrentHashMap<>();

Map<String,String> tableB = new ConcurrentHashMap<>();

@Override

protected void setup(Context context)

throws IOException, InterruptedException {

URI[] uris = context.getCacheFiles();

for(URI u : uris) {

String uriPath = u.toString();

if(uriPath.endsWith("tableA")) {

BufferedReader bs = new BufferedReader(new FileReader(new File(u.toString())));

String str = null;

while((str = bs.readLine())!=null) {

String[] strs = str.split("\t");

sexMap.put(strs[0], strs[1]);

}

}else if(uriPath.endsWith("tableB")){

BufferedReader bs = new BufferedReader(new FileReader(new File(u.toString())));

String str = null;

while((str = bs.readLine())!=null) {

String[] strs = str.split("\t");

userMap.put(strs[0], strs[1]);

}

}

}

}

@Override

protected void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

if(tableA.containsKey(name) && tableB.containsKey(name)) {

//输出

context.write(new Text(sex+"\t"+tableA.get(name)+"\t"+"age"+tableB.get(name)+"\t"), new Text(""));

}

}

}

}

#### 有三个map,一个reduce来做top10.哪种方法最优。数据量特别大。

#### 1000个maptask任务,同时在运行,reduce所拉取的文件是如何找到相应的map端

#### 用MapReduce来实现下面需求？

现在有10个文件夹,每个文件夹都有1000000个url.现在让你找出top1000000url。

解答：topk

#### MapReduce中Combiner和Partition的作用

Combiner是Reduce的实现，在Map端运行计算任务，减少Map端的输出数据。

作用就是优化。

但是Combiner的使用场景是MapReduce的Map输出结果必须和Reduce输入输出一样。

Partition的默认实现是HashPartition，是Map端将数据按照Reduce个数取余，进行分区，不同的Reduce来Copy自己的数据。

Partiton的作用是将数据分到不同的Reduce进行计算，加快计算效果

#### 我们开发job时，是否可以去掉reduce阶段。

可以。设置reduce数为0 即可。

#### combiner出现在那个过程

出现在map阶段的map方法后等。

#### 描述一下Hadoop中，有哪些地方使用了缓存机制，作用分别是什么？

在MapReduce提交job的获取id之后，会将所有文件存储到分布式缓存上，这样文件可以被所有的MapReduce共享

#### MapReduce 的 map 数量 和 reduce 数量 怎么确定 ,怎么配置

map的数量有数据块决定，reduce数量随便配置

#### 如果要存储海量的小文件（大小都是几百K~几M），请简述自己的设计方案

1. 将小文件打成har文件存储
2. 将小文件序列化到HDFS中

#### 有两个文本**文件，文件中数据按行存放，请编写MapReduce程序，找到两个文件中彼此不相同的行（写出思路即可）**

写个MapReduce链 用依赖关系，一共三个MapReduce，第一个处理第一个文件，第二个处理第二个文件，第三个处理前两个的输出结果，第一个MapReduce将文件去重，第二个MapReduce也将文件去重，第三个做wordcount，wordcount为1的结果就是不同的

#### 找共同朋友：



思路：例如A，他的朋友是B\C\D\E\F\，那么BC的共同朋友就是A。所以将BC作为key，将A作为value，在map端输出即可！其他的朋友循环处理。

|  |
| --- |
| import java.io.IOException;  2.  import java.util.Set;  3.  import java.util.StringTokenizer;  4.  import java.util.TreeSet;  5.    6.  import org.apache.Hadoop.conf.Configuration;  7.  import org.apache.Hadoop.fs.Path;  8.  import org.apache.Hadoop.io.Text;  9.  import org.apache.Hadoop.MapReduce.Job;  10. import org.apache.Hadoop.MapReduce.Mapper;  11. import org.apache.Hadoop.MapReduce.Reducer;  12. import org.apache.Hadoop.MapReduce.Mapper.Context;  13. import org.apache.Hadoop.MapReduce.lib.input.FileInputFormat;  14. import org.apache.Hadoop.MapReduce.lib.output.FileOutputFormat;  15. import org.apache.Hadoop.util.GenericOptionsParser;  16.  17. public class FindFriend {  www.aboutyun.com  18.  19. public static class ChangeMapper extends Mapper<Object, Text, Text,  Text>{  20. @Override  21. public void map(Object key, Text value, Context context) throws  IOException, InterruptedException {  22. StringTokenizer itr = new StringTokenizer(value.toString());  23. Text owner = new Text();  24. Set<String> set = new TreeSet<String>();  25. owner.set(itr.nextToken());  26. while (itr.hasMoreTokens()) {  27. set.add(itr.nextToken());  28. }  29. String[] friends = new String[set.size()];  30. friends = set.toArray(friends);  31.  32. for(int i=0;i<friends.length;i++){  33. for(int j=i+1;j<friends.length;j++){  34. String outputkey = friends[i]+friends[j];  35. context.write(new Text(outputkey),owner);  36. }  37. }  38. }  39. }  40.  41. public static class FindReducer extends Reducer<Text,Text,Text,Text>  {  42. public void reduce(Text key, Iterable<Text> values,  43. Context context) throws IOException,  InterruptedException {  44. String commonfriends ="";  www.aboutyun.com  45. for (Text val : values) {  46. if(commonfriends == ""){  47. commonfriends = val.toString();  48. }else{  49. commonfriends =  commonfriends+":"+val.toString();  50. }  51. }  52. context.write(key, new  Text(commonfriends));  53. }  54. }  55.  56.  57. public static void main(String[] args) throws IOException,  58. InterruptedException, ClassNotFoundException {  59.  60. Configuration conf = new Configuration();  61. String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  62. if (otherArgs.length < 2) {  63. System.err.println("args error");  64. System.exit(2);  65. }  66. Job job = new Job(conf, "word count");  67. job.setJarByClass(FindFriend.class);  68. job.setMapperClass(ChangeMapper.class);  69. job.setCombinerClass(FindReducer.class);  70. job.setReducerClass(FindReducer.class);  71. job.setOutputKeyClass(Text.class);  72. job.setOutputValueClass(Text.class);  73. for (int i = 0; i < otherArgs.length - 1; ++i) {  www.aboutyun.com  74. FileInputFormat.addInputPath(job, new Path(otherArgs[i]));  75. }  76. FileOutputFormat.setOutputPath(job,  77. new Path(otherArgs[otherArgs.length - 1]));  78. System.exit(job.waitForCompletion(truE、 ? 0 : 1);  79.  80. }  81.  82. } |

结果：

|  |
| --- |
| 1. AB E:C:D  2. AC E:B  3. AD B:E  4. AE C:B:D  5. BC A:E  6. BD A:E  7. BE C:D:A  8. BF A  9. CD E:A:B  10. CE A:B  11. CF A  12. DE B:A  13. DF A  14. EF A |

#### 你的项目提交到 job 的时候数据量有多大？(面试了三家，都问这个问题)

答：1）回答出数据是什么格式，有没有采用什么压缩，采用了压缩的话，压缩比大概是多少；2）文件大概多大：大概起了多少个map，起了多少个reduce，map阶段读取了多少数据，reduce阶段读取了多少数据，程序大约执行了多久，3）集群什么规模，集群有多少节点，多少内存，多少CPU核数等。把这些点回答进去，而不是给个数字了事。

#### reduce 后输出的数据量有多大？

并不是想知道确切的数据量有多大这个，而是想问你，MR的执行机制，开发完程序，有没有认真评估程序运行效率

1）用于处理redcue任务的资源情况，如果是MRV1的话，分了多少资源给map，多少个reduce

如果是MRV2的话，可以提一下，集群有分了多少内存、CPU给yarn做计算 。

2）结合实际应用场景回答，输入数据有多大，大约多少条记录，做了哪些逻辑操作，输出的时候有多少条记录，执行了多久，reduce执行时候的数据有没有倾斜等

3）再提一下，针对mapReduce做了哪几点优化，速度提升了多久，列举1,2个优化点就可以

#### job 的运行流程(提交一个 job 的流程)？

1. 任务提交：在main方法中调用job.waitForCompletion(true)[这里的参数true是确定打印进度信息给用户]，waitForCompletion中又调用了job.submit(),这个时候提交的过程就开始了。
2. 提交作业到集群：
   1. 当提交开始后，作业就进入到了定义状态（DEFINE）。之后会默认地使用新的api；
   2. 获取ClientProtocol代理对象：job提交到yarn上实际上就是客户端与RM进行RPC通信，这里使用的是ClientProtocol的通讯协议，所以就需要在客户端获取到ClientProtocol的代理对象。这个过程就是由job.connect()完成的。它返回的是一个集群信息对象（cluster）。Cluster内部构建了这个代理对象。
   3. 获取提交器：获取到集群信息对象后就会根据该对象封装的信息创建一个job的提交器对象（jobSubmitter）。【getJobSubmitter(cluster.getFileSystem(), cluster.getClient())】.

通讯

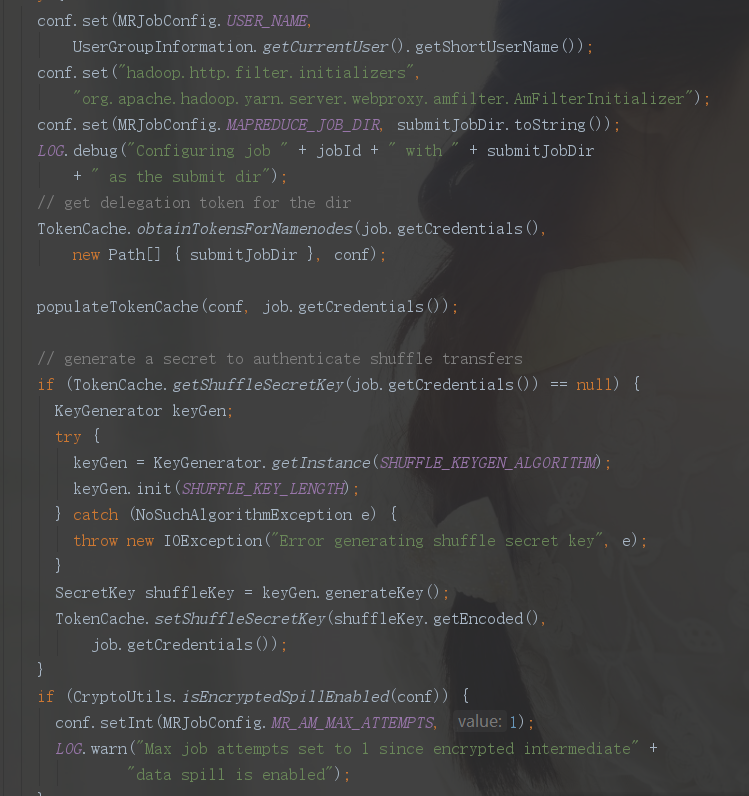
* 1. 客户端与RM的通信操作：客户端拿到了通信的代理对象后就会与RM进行通信，并完成一系列的操作：
     1. 检查作业输出路径是否存在，若存在就报异常。【checkSpecs(job)】；
     2. 然后将job的配置对象放入到RM的框架中(JobSubmitter.addMRFrameworkToDistributedCache())；
     3. 获取登录目录路径【JobSubmissionFiles.getStagingDir(cluster, conf)】：获取这个路径后会与后面形成的jobID拼接形成一个submitJobDir（job资源提交路径：hdfs://……./. Staging/jobid ）的目录路径。这个目录用于存放客户端提交的作业文件。

submitJobDir

* + 1. 配置提交任务的主机ID与主机名；
    2. 创建一个新的jobID并将这个ID设置到job上；

创建id

* + 1. 配置相关权限；

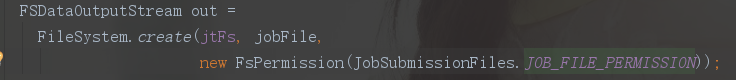


* + 1. CopyAndConfigureFiles(job, submitJobDir)：将job所需的全部文件上传到对应的目录下；（-libjars, -files, -archives）
    2. 计算作业的输入分片（JobSubmitter.writeSplits()）：
       1. 这里面会获取到一个job的配置对象，如果配置对象里采用的是新的mapper就会根据这个新的作业计算新的分分片数，如果不是新的mapper就会沿用以前的分片数。（自己理解的）新计算分片数的话：会根据FileInputFormat类的规则（文件总剩余大小大于块大小的110%时分片大小就等于块大小，否则就切割成一片）进行分片。
       2. 分片完成后会将inputSpilt放入到一个List集合中然后会将这个集合转化成一个泛型数组，再将数组里的元素按照文件大小进行排序，排好序后会取出数组的长度（即分片个数）；

泛型数组

* + - 1. 之后会将分片个数配置到集群中的job配置里。conf.setInt(MRJobConfig.NUM\_MAPS, maps);

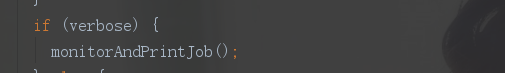
Job.writeConf()这个现将job的配置转化成一个job.xml文件，再将这个文件上传到文件系统中（JOB\_FILE\_PERMISSION：rw-r--r--）；



JobSubmitter.submitClient.submitJob()：到这一步就是真正的job的提交，注意的是这里的提交者是cluster对象中构建的ClientProtocol的代理对象，返回一个状态信息。

真正提交

1. 提交完成后持续监控程序并输出监控信息；



1. 返回当前MR应用程序最终成功与否的状态。（return isSuccessful()）

#### 假如：Flume 收集到的数据很多个小文件,我需要写 MR 处理时将这些文件合并(是在 MR 中进行优化,不让一个小文件一个 MapReduce)

（1） 默认情况下，TextInputFormat对任务的切片机制是按照文件规划切片，不管有多少个小文件，都会是单独的切片，都会交给一个maptask，这样，如果有大量的小文件

就会产生大量的maptask，处理效率极端底下

（2）优化策略

最好的方法：在数据处理的最前端（预处理、采集），就将小文件合并成大文件，在上传到HDFS做后续的分析

补救措施：如果已经是大量的小文件在HDFS中了，可以使用另一种inputformat来做切片（CombineFileInputformat），它的切片逻辑跟FIleinputformat不同：

    它可以将多个小文件从逻辑上规划到一个切片中，这样，多个小文件就可以交给一个maptask了

//如果不设置InputFormat，它默认的用的是TextInputFormat.class

/\*CombineTextInputFormat为系统自带的组件类

\* setMinInputSplitSize 中的2048是表示n个小文件之和不能大于2048

\* setMaxInputSplitSize 中的4096是

当满足setMinInputSplitSize中的2048情况下 在满足n+1个小文件之和不能大于4096

\*/

job.setInputFormatClass(CombineTextInputFormat.class);

CombineTextInputFormat.setMinInputSplitSize(job, 2048);

CombineTextInputFormat.setMaxInputSplitSize(job, 4096);

补充：当n+1大于最大值时候，大文件会被切开，放到n个文件去

#### “MapReduce 2.0”与“YARN”是否等同，尝试解释说明

MapReduce 2.0是一个计算框架，在运行时环境上引入全新的资源管理框架：**YARN**，将JobTracker中的**资源管理**和**作业控制**功能**分开**，分别由两个不同进程**ResourceManager**和**ApplicationMaster**实现，ResourceManager负责所有应用程序的资源分配，Application负责管理一个应用程序。

YARN：是资源管理框架

#### MapReduce 2.0 中，MRAppMaster 主要作用是什么，MRAppMaster 如何实现任务容错的？

作用：

 MRAppMaster是MapReduce的ApplicationMaster实现，它使得MapReduce应用程序可以直接运行于YARN之上。在YARN中，MRAppMaster负责管理MapReduce作业的生命周期，包括作业管理、资源申请与再分配、Container启动与释放、作业恢复等。

容错：

MRAppMaster采用了任务级别的恢复机制，即以任务为基本单位进行恢复，这种机制是基于事务型日志完成作业恢复的，它只关注两种任务：运行完成的任务和未完成的任务。作业执行过程中，MRAppMaster会以日志的形式将作业以及状态记录下来，一旦MRAppMaster重启，则可从日志中恢复作业的运行状态。

      当前MRAppMaster的作业恢复机制仅能做到恢复上一次已经运行完成的任务，对于正在运行的任务，则在前一次MRAppMaster运行实例退出时由ResourceManager强制将其杀死并回收资源。

#### 怎么提升多个 JOB 同时执行带来的压力,如何优化,说说思路

#### MapReduce 计算模型、 MapReduce 基础知识点（MapReduce 新旧 API 的使用、在 linux 命令行运行 MapReduce 程序、自定义 Hadoop 数据类型）？

#### MapReduce 执行流程：“天龙八步”，计数器、自定义分区、自定义排序、自定义分组、如何对 value 进行排序：次排序+自定义分组、归约？

#### MapReduce 的 shuffle 工作原理、 MapReduce 工作原理（MapReduce 源码、InputStream 源码、waitForCompletion()源码）、jobtracker 如何创建 map 任务和 reduce任务是面试的重点。

#### MapReduce 进阶知识：Hadoop 的几种文件格式、常见输入输出格式化类、多输入多输出机制、 MapReduce 的常见算法（各种 join 原理和优缺点、次排序和总排序）？

#### MapReduce 性能优化（shuffle 调优、压缩算法、更换调度器、设置 InputSplit 大小减少 map 任务数量、 map 和 reduce 的 slot 如何设置、数据倾斜原理和如何解决）？

#### hive 的两张表关联，使用 mapreduce 是怎么写的？

#### map 方法是如何调用 reduce 方法的

#### 每个 map 槽就是一个线程。（ ）

#### Mapreduce 的 input split 就是一个 block。（ ）

#### 例举 hadoop 中定义的最常用的 InputFormats，哪个是默认的？

提 示 ： DBInputFormat 、 FileInputFormat （ KeyValueTextInputFormat 、NlineInputFormat、 TextInputFormat）

默认：TextInputFormat

#### TextInputFormat 和 KeyValueInputFormat 类之间的不同之处在于哪里？

提示：TextInputFormat 中的 key 表示行的偏移量，value 是行文本内容

KeyValueInputFormat 的 key value 是通过第一个制表符进行划分的

#### hadoop 中的 InputSplit 是什么？

#### hadoop 中 RecordReader 的目的是什么？

#### 如果 hadoop 中没有定义定制分区，那么如何在输出到 reducer 前执行数据分区？

­-------------------------------------------------------------------------------------------------------------------------------------------

#### hadoop 中 job 和 task 之间是什么关系？

#### 假设 hadoop 一个 job 产生 100 个 task，并且其中的一个 task 失败了，hadoop会怎样处理？

#### MapReduce 中排序发生在哪几个阶段？这些排序是否可以避免？为什么？

#### 编写 MapReduce 作业时，如何做到在 Reduce 阶段，先对 Key 排序，再对 Value排序？

#### mapreduce 全排序原理。

#### 你认为用Java,Sreaming, pipe 方式开发 map/reduce, 各有哪些优缺点

#### 请简述hadoop 怎样实现二级排序

#### 如何使用 MapReduce 实现两个表 join，可以考虑一下几种情况

1. 一个表大，一个表小（可放到内存中）；（2）两个表都是大表？

#### MapReduce 怎么实现 Top10？

如：求音乐数据的点击率top10

Mapper端：完成数据抽取后，自定义一个比较器，可以按照点击率降序排序

reducer端输出前10组数据

#### 简述 hadoop 实现 join 的几种方法

#### flush 的过程？

#### 写代码实现 1G 大小的文本文件，行分隔符为\x01\x02,统计一下该文件中的总行数, 要求注意边界情况的处理

#### 用 mapreduce 实现 sql 语句 select count(x) from a group by b？

#### 知道 combiner, partition 作用，设置 compression

#### hadoop mapreduce 创建类 DataWritable 的作用是什么？

#### 为什么创建类 DataWritable？

#### 举一个简略的比方阐明 mapreduce 是怎么来运转的

#### 面试的人给你出一些疑问,让你用 mapreduce 来完成？

比方: 如今有 10 个文件夹, 每个文件夹都有 1000000 个 url. 如今让你找出top1000000url。

#### 用 Hadoop 分析海量日志文件，每行日志记录了如下数据：

TableName(表名)，Time(时间)，User(用户)，TimeSpan(时间开销)。

要求：

编写 MapReduce 程序算出高峰时间段（如上午 10 点）哪张表被访问的最频繁，以及这段时间访问这张表最多的用户，以及这个用户的总时间开销。

### 4.3yarn

### 4.4Hadoop优化

### 4.5Hadoop