尚硅谷大数据技术之 Hadoop 阶段考试题及答案

(作者：WHAlex)

官网：www.atguigu.com

版本：V1.0

Linux

1. Linux 常用命令

```
参考答案：find、df、tar、ps、top、netstat 等。（尽量说一些高级命令）
```

2. Linux 查看内存、磁盘存储、io 读写、端口占用、进程等命令

```
答案：
1、查看内存：top
2、查看磁盘存储情况：df -h
3、查看磁盘 IO 读写情况：iotop（需要安装一下：yum install iotop）、iotop
-o（直接查看输出比较高的磁盘读写程序）
4、查看端口占用情况：netstat -tunlp | grep 端口号
5、查看进程：ps aux
```

Shell

3. 使用 Linux 命令查询 file1 中空行所在的行号

```
答案：
[atguigu@hadoop102 datas]$ awk '/^$/{print NR}' file1.txt
5
```

4. 有文件 chengji.txt 内容如下：

张三 40
李四 50
王五 60

使用 Linux 命令计算第二列的和并输出

```
[atguigu@hadoop102 datas]$ cat chengji.txt | awk -F " "
'{sum+=$2} END{print sum}'
150
```

5. Shell 脚本里如何检查一个文件是否存在？如果不存在该如何处理？

```
#!/bin/bash

if [ -f file.txt ]; then
   echo "文件存在!"
else
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
   echo "文件不存在!"
fi
```

6. 用 shell 写一个脚本，对文本中无序的一列数字排序

```
[root@CentOS6-2 ~]# cat test.txt
9
8
7
6
5
4
3
2
10
1
[root@CentOS6-2 ~]# sort -n test.txt|awk '{a+=$0;print
$0}END{print "SUM="a}'
1
2
3
4
5
6
7
8
9
10
SUM=55
```

7. 请用 shell 脚本写出查找当前文件夹（/home）下所有的文本文件内容中包含有字符"shen"的文件名称

```
[atguigu@hadoop102 datas]$ grep -r "shen" /home | cut -d ":"
-f 1
/home/atguigu/datas/sed.txt
/home/atguigu/datas/cut.txt
```

入门

1. 简要描述如何安装配置 apache 的一个开源 Hadoop，只描述即可，无需列出具体步骤，列出具体步骤更好。

1) 准备三台客户机（配置 IP，配置主机名...）
2) 安装 jdk，安装 hadoop
3) 配置 JAVA_HOME 和 HADOOP_HOME
4) 使每个节点上的环境变量生效（source /etc/profile）
5) 准备分发脚本 xsync
   a) **在/user/atguigu/bin 下创建脚本：xsync
6) 明确集群的配置

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

7) 修改配置文件
   a) **core-site.xml
   b) **hadoop-env.sh
   c) **hdfs-site.xml
   d) **yarn-env.sh
   e) **yarn-site.xml
   f) **mapred-env.sh
   g) **mapred-site.xml
   h) **配置 slaves
8) 分发配置文件
   a) **xsync /etc/hadoop
9) 删掉 data 和 logs 文件夹
10) 配置 ssh（hadoop102，hadoop103）
11) 分发配置文件
12) 格式化 hdfs（hdfs namenode -format）
13) 群启 hdfs
14) 群启 yarn

2. Hadoop 中需要哪些配置文件，其作用是什么？

1）core-site.xml：

(1)fs.defaultFS:hdfs://cluster1(域名)，这里的值指的是默认的 HDFS 路径 。

(2)hadoop.tmp.dir:/export/data/hadoop_tmp,这里的路径默认是 NameNode、DataNode、secondaryNamenode 等存放数据的公共目录。用户也可以自己单独指定这三类节点的目录。

(3)ha.zookeeper.quorum:hadoop101:2181,hadoop102:2181,hadoop103:2181,这里是 ZooKeeper 集群的地址和端口。注意，数量一定是奇数，且不少于三个节点 。

2）hadoop-env.sh: 只需设置 jdk 的安装路径，如：export JAVA_HOME=/usr/local/jdk。

3）hdfs-site.xml：

(1) dfs.replication:他决定着系统里面的文件块的数据备份个数，默认为 3 个。

(2) dfs.data.dir:datanode 节点存储在文件系统的目录 。

(3) dfs.name.dir:是 namenode 节点存储 hadoop 文件系统信息的本地系统路径 。

4）mapred-site.xml：

mapreduce.framework.name: yarn 指定 mr 运行在 yarn 上。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

3. 请列出正常工作的 Hadoop 集群中 Hadoop 都分别需要启动哪些进程，它们的作用分别是什么?

1）NameNode 它是 hadoop 中的主服务器，管理文件系统名称空间和对集群中存储的文件的访问，保存有 metadate。

2）SecondaryNameNode 它不是 namenode 的冗余守护进程，而是提供周期检查点和清理任务。帮助 NN 合并 editslog，减少 NN 启动时间。

3）DataNode 它负责管理连接到节点的存储（一个集群中可以有多个节点）。每个存储数据的节点运行一个 datanode 守护进程。

4）ResourceManager（JobTracker）JobTracker 负责调度 DataNode 上的工作。每个 DataNode 有一个 TaskTracker，它们执行实际工作。

5）NodeManager（TaskTracker）执行任务。

6）DFSZKFailoverController 高可用时它负责监控 NN 的状态，并及时的把状态信息写入 ZK。它通过一个独立线程周期性的调用 NN 上的一个特定接口来获取 NN 的健康状态。FC 也有选择谁作为 Active NN 的权利，因为最多只有两个节点，目前选择策略还比较简单（先到先得，轮换）。

7）JournalNode 高可用情况下存放 namenode 的 editlog 文件。

4. 简述 Hadoop 的几个默认端口及其含义。

1）dfs.namenode.http-address:50070

2）SecondaryNameNode 辅助名称节点端口号：50090

3）dfs.datanode.address:50010

4）fs.defaultFS:8020 或者 9000

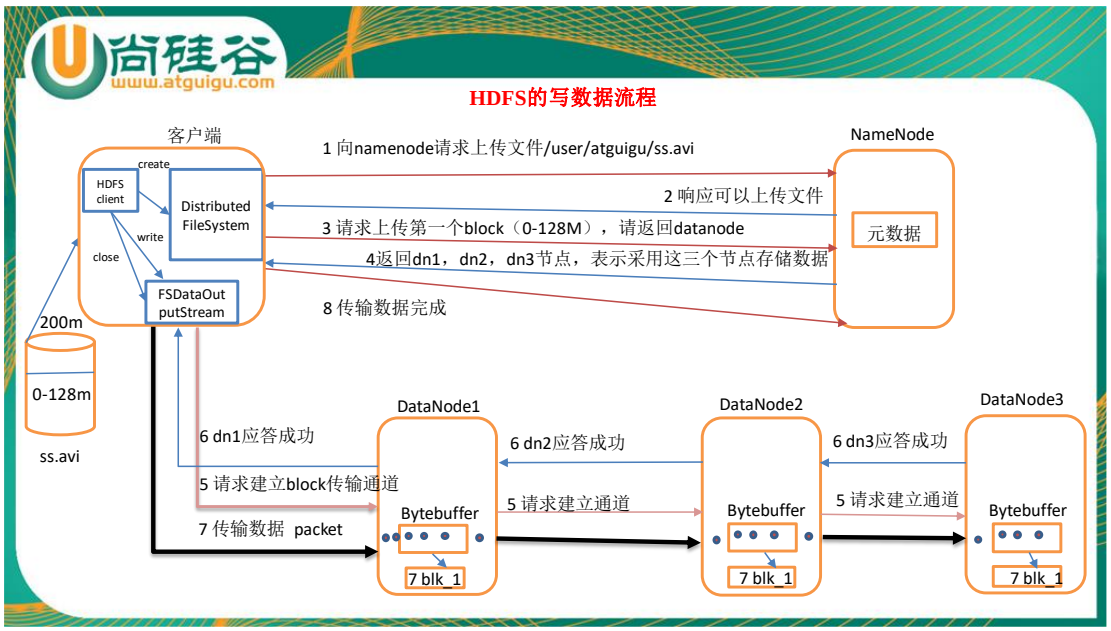
5）yarn.resourcemanager.webapp.address:8088

HDFS

1. HDFS 的存储机制（读写流程）。

HDFS 存储机制，包括 HDFS 的写入过程和读取过程两个部分

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】



1）客户端向 namenode 请求上传文件，namenode 检查目标文件是否已存在，父目录是否存在。

2）namenode 返回是否可以上传。

3）客户端请求第一个 block 上传到哪几个 datanode 服务器上。

4）namenode 返回 3 个 datanode 节点，分别为 dn1、dn2、dn3。

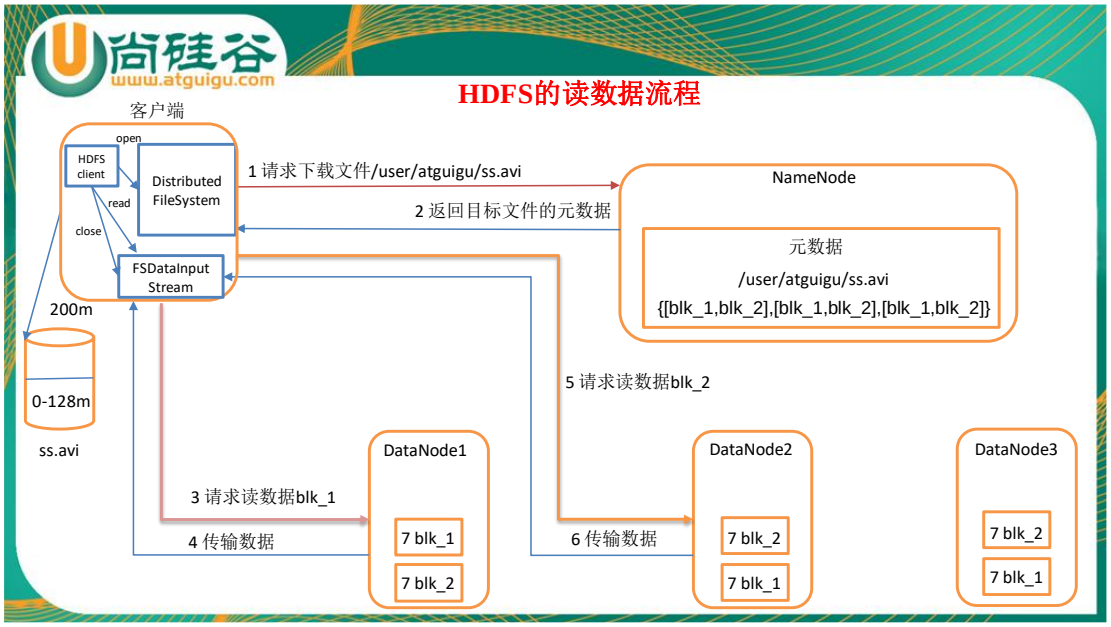
5）客户端请求 dn1 上传数据，dn1 收到请求会继续调用 dn2，然后 dn2 调用 dn3，将这个通信管道建立完成。

6）dn1、dn2、dn3 逐级应答客户端

7）客户端开始往 dn1 上传第一个 block（先从磁盘读取数据放到一个本地内存缓存），以 packet 为单位，dn1 收到一个 packet 就会传给 dn2，dn2 传给 dn3；dn1 每传一个 packet 会放入一个应答队列等待应答

8）当一个 block 传输完成之后，客户端再次请求 namenode 上传第二个 block 的服务器。（重复执行 3-7 步）

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】



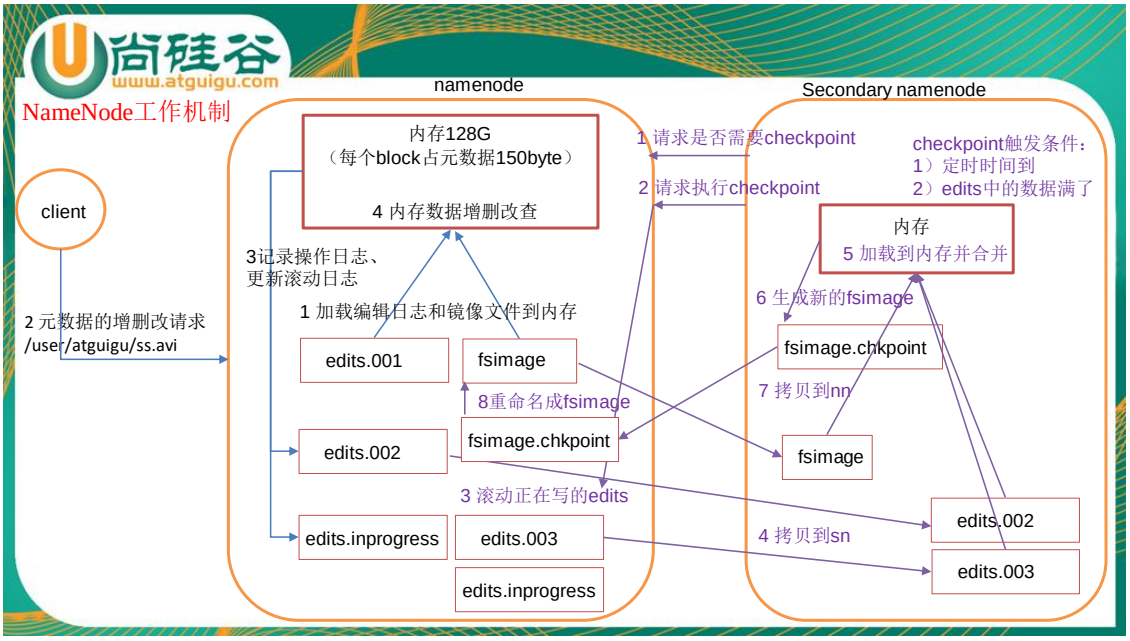
1）客户端向 namenode 请求下载文件，namenode 通过查询元数据，找到文件块所在的 datanode 地址。

2）挑选一台 datanode（就近原则，然后随机）服务器，请求读取数据。

3）datanode 开始传输数据给客户端（从磁盘里面读取数据放入流，以 packet 为单位来做校验）。

4）客户端以 packet 为单位接收，先在本地缓存，然后写入目标文件。

2. SecondaryNameNode 工作机制。



1）第一阶段：namenode 启动

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

（1）第一次启动 namenode 格式化后，创建 fsimage 和 edits 文件。如果不是第一次启动，直接加载编辑日志和镜像文件到内存。

（2）客户端对元数据进行增删改的请求

（3）namenode 记录操作日志，更新滚动日志。

（4）namenode 在内存中对数据进行增删改查

2）第二阶段：Secondary NameNode 工作

（1）Secondary NameNode 询问 namenode 是否需要 checkpoint。直接带回 namenode 是否检查结果。

（2）Secondary NameNode 请求执行 checkpoint。

（3）namenode 滚动正在写的 edits 日志

（4）将滚动前的编辑日志和镜像文件拷贝到 Secondary NameNode

（5）Secondary NameNode 加载编辑日志和镜像文件到内存，并合并。

（6）生成新的镜像文件 fsimage.chkpoint

（7）拷贝 fsimage.chkpoint 到 namenode

（8）namenode 将 fsimage.chkpoint 重新命名成 fsimage

3. NameNode 与 SecondaryNameNode 的区别与联系？

1）机制流程同上；

2）区别

（1）NameNode 负责管理整个文件系统的元数据，以及每一个路径（文件）所对应的数据块信息。

（2）SecondaryNameNode 主要用于定期合并命名空间镜像和命名空间镜像的编辑日志。

3）联系：

（1）SecondaryNameNode 中保存了一份和 namenode 一致的镜像文件（fsimage）和编辑日志（edits）。

（2）在主 namenode 发生故障时（假设没有及时备份数据），可以从 SecondaryNameNode 恢复数据。

4. 服役新数据节点和退役旧节点步骤

1）节点上线操作：

当要新上线数据节点的时候，需要把数据节点的名字追加在 dfs.hosts 文件中

（1）关闭新增节点的防火墙

（2）在 NameNode 节点的 hosts 文件中加入新增数据节点的 hostname

（3）在每个新增数据节点的 hosts 文件中加入 NameNode 的 hostname

（4）在 NameNode 节点上增加新增节点的 SSH 免密码登录的操作

（5）在 NameNode 节点上的 dfs.hosts 中追加上新增节点的 hostname,

（6）在其他节点上执行刷新操作：hdfs dfsadmin -refreshNodes

（7）在 NameNode 节点上，更改 slaves 文件，将要上线的数据节点 hostname 追加到 slaves 文件中

（8）启动 DataNode 节点

（9）查看 NameNode 的监控页面看是否有新增加的节点

2）节点下线操作：

（1）修改/conf/hdfs-site.xml 文件

（2）确定需要下线的机器，dfs.osts.exclude 文件中配置好需要下架的机器，这个是阻止下架的机器去连接 NameNode。

（3）配置完成之后进行配置的刷新操作./bin/hadoop dfsadmin -refreshNodes,这个操作的作用是在后台进行 block 块的移动。

（4）当执行三的命令完成之后，需要下架的机器就可以关闭了，可以查看现在集群上连接的节点，正在执行 Decommission，会显示：Decommission Status : Decommission in progress 执行完毕后，会显示：Decommission Status : Decommissioned

（5）机器下线完毕，将他们从 excludes 文件中移除。

5. Namenode 挂了怎么办？

方法一：将 SecondaryNameNode 中数据拷贝到 namenode 存储数据的目录；

方法二：使用-importCheckpoint 选项启动 namenode 守护进程，从而将 SecondaryNameNode 中数据拷贝到 namenode 目录中。

MapReduce

1. 谈谈 Hadoop 序列化和反序列化及自定义 bean 对象实现序列化？

1）序列化和反序列化

序列化就是把内存中的对象，转换成字节序列（或其他数据传输协议）以便于存储（持久化）和网络传输。

反序列化就是将收到字节序列（或其他数据传输协议）或者是硬盘的持久化数据，转换成内存中的对象。

Java 的序列化是一个重量级序列化框架（Serializable），一个对象被序列化后，会附带很多额外的信息（各种校验信息，header，继承体系等），不便于在网络中高效传输。所以，hadoop 自己开发了一套序列化机制（Writable），精简、高效。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

2）自定义 bean 对象要想序列化传输步骤及注意事项：。

（1）必须实现 Writable 接口

（2）反序列化时，需要反射调用空参构造函数，所以必须有空参构造

（3）重写序列化方法

（4）重写反序列化方法

（5）注意反序列化的顺序和序列化的顺序完全一致

（6）要想把结果显示在文件中，需要重写 toString()，且用”\t”分开，方便后续用

（7）如果需要将自定义的 bean 放在 key 中传输，则还需要实现 comparable 接口，因为 mapreduce 框中的 shuffle 过程一定会对 key 进行排序

2. FileInputFormat 切片机制

（1）简单地按照文件的内容长度进行切片

（2）切片大小，默认等于 block 大小

（3）切片时不考虑数据集整体，而是逐个针对每一个文件单独切片

3. 自定义 InputFormat 流程

（1）自定义一个类继承 FileInputFormat

（2）改写 RecordReader，实现一次读取一个完整文件封装为 KV

4. 如何决定一个 job 的 map 和 reduce 的数量?

1）map 数量

splitSize=max{minSize,min{maxSize,blockSize}}

map 数量由处理的数据分成的 block 数量决定 default_num = total_size / split_size;

2）reduce 数量

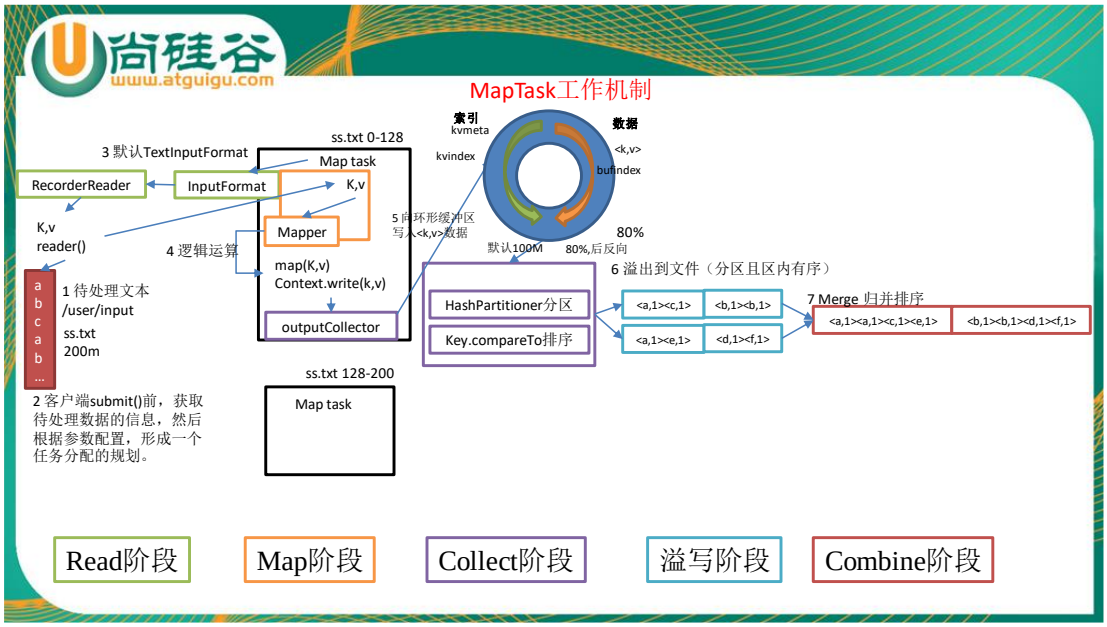
reduce 的数量 job.setNumReduceTasks(x);x 为 reduce 的数量。不设置的话默认为 1。

5. Maptask 的个数由什么决定？

一个 job 的 map 阶段 MapTask 并行度（个数），由客户端提交 job 时的切片个数决定。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

6. MapTask 工作机制



（1）Read 阶段：Map Task 通过用户编写的 RecordReader，从输入 InputSplit 中解析出一个个 key/value。

（2）Map 阶段：该节点主要是将解析出的 key/value 交给用户编写 map()函数处理，并产生一系列新的 key/value。

（3）Collect 收集阶段：在用户编写 map()函数中，当数据处理完成后，一般会调用 OutputCollector.collect()输出结果。在该函数内部，它会将生成的 key/value 分区（调用 Partitioner），并写入一个环形内存缓冲区中。

（4）Spill 阶段：即“溢写”，当环形缓冲区满后，MapReduce 会将数据写到本地磁盘上，生成一个临时文件。需要注意的是，将数据写入本地磁盘之前，先要对数据进行一次本地排序，并在必要时对数据进行合并、压缩等操作。

溢写阶段详情：

步骤 1：利用快速排序算法对缓存区内的数据进行排序，排序方式是，先按照分区编号 partition 进行排序，然后按照 key 进行排序。这样，经过排序后，数据以分区为单位聚集在一起，且同一分区内所有数据按照 key 有序。

步骤 2：按照分区编号由小到大依次将每个分区中的数据写入任务工作目录下的临时文件 output/spillN.out（N 表示当前溢写次数）中。如果用户设置了 Combiner，则写入文件之

前，对每个分区中的数据进行一次聚集操作。

步骤 3：将分区数据的元信息写到内存索引数据结构 SpillRecord 中，其中每个分区的元信息包括在临时文件中的偏移量、压缩前数据大小和压缩后数据大小。如果当前内存索引大小超过 1MB，则将内存索引写到文件 output/spillN.out.index 中。

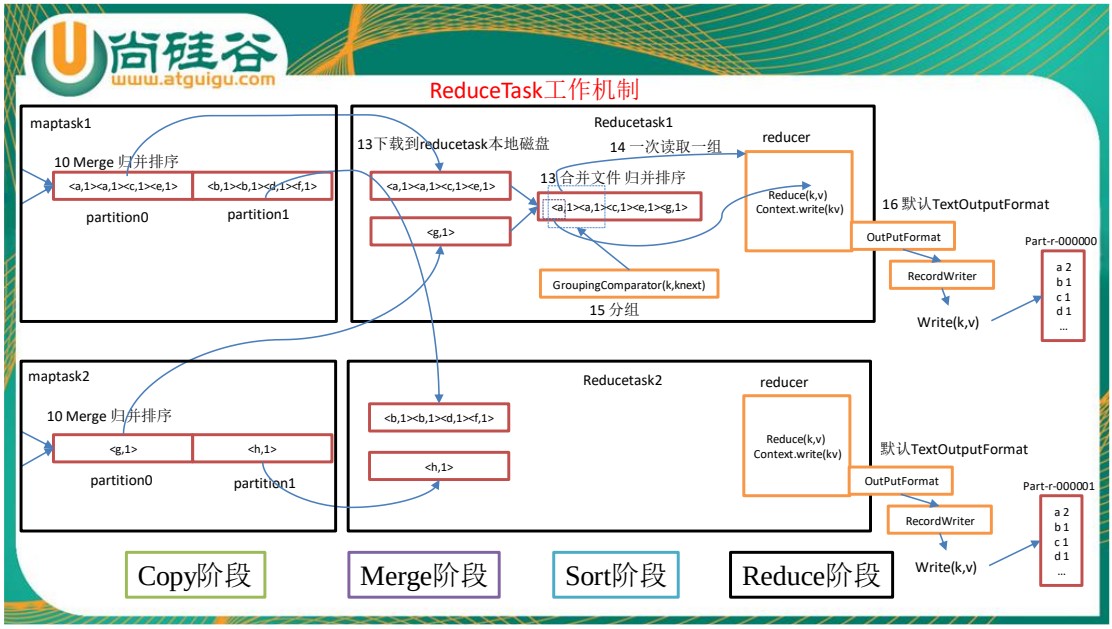
（5）Combine 阶段：当所有数据处理完成后，MapTask 对所有临时文件进行一次合并，以确保最终只会生成一个数据文件。

当所有数据处理完后，MapTask 会将所有临时文件合并成一个大文件，并保存到文件 output/file.out 中，同时生成相应的索引文件 output/file.out.index。

在进行文件合并过程中，MapTask 以分区为单位进行合并。对于某个分区，它将采用多轮递归合并的方式。每轮合并 io.sort.factor（默认 100）个文件，并将产生的文件重新加入待合并列表中，对文件排序后，重复以上过程，直到最终得到一个大文件。

让每个 MapTask 最终只生成一个数据文件，可避免同时打开大量文件和同时读取大量小文件产生的随机读取带来的开销。

7. ReduceTask 工作机制。



（1）Copy 阶段：ReduceTask 从各个 MapTask 上远程拷贝一片数据，并针对某一片数据，如果其大小超过一定阈值，则写到磁盘上，否则直接放到内存中。

（2）Merge 阶段：在远程拷贝数据的同时，ReduceTask 启动了两个后台线程对内存和

磁盘上的文件进行合并，以防止内存使用过多或磁盘上文件过多。

（3）Sort 阶段：按照 MapReduce 语义，用户编写 reduce()函数输入数据是按 key 进行聚集的一组数据。为了将 key 相同的数据聚在一起，Hadoop 采用了基于排序的策略。由于各个 MapTask 已经实现对自己的处理结果进行了局部排序，因此，ReduceTask 只需对所有数据进行一次归并排序即可。

（4）Reduce 阶段：reduce()函数将计算结果写到 HDFS 上。

8. 请描述 mapReduce 有几种排序及排序发生的阶段。

1）排序的分类：

（1）部分排序：

MapReduce 根据输入记录的键对数据集排序。保证输出的每个文件内部排序。

（2）全排序：

如何用 Hadoop 产生一个全局排序的文件？最简单的方法是使用一个分区。但该方法在处理大型文件时效率极低，因为一台机器必须处理所有输出文件，从而完全丧失了 MapReduce 所提供的并行架构。

替代方案：首先创建一系列排好序的文件；其次，串联这些文件；最后，生成一个全局排序的文件。主要思路是使用一个分区来描述输出的全局排序。例如：可以为待分析文件创建 3 个分区，在第一分区中，记录的单词首字母 a-g，第二分区记录单词首字母 h-n，第三分区记录单词首字母 o-z。

（3）辅助排序：（GroupingComparator 分组）

Mapreduce 框架在记录到达 reducer 之前按键对记录排序，但键所对应的值并没有被排序。甚至在不同的执行轮次中，这些值的排序也不固定，因为它们来自不同的 map 任务且这些 map 任务在不同轮次中完成时间各不相同。一般来说，大多数 MapReduce 程序会避免让 reduce 函数依赖于值的排序。但是，有时也需要通过特定的方法对键进行排序和分组等以实现对值的排序。

（4）二次排序：

在自定义排序过程中，如果 compareTo 中的判断条件为两个即为二次排序。

2）自定义排序 WritableComparable

bean 对象实现 WritableComparable 接口重写 compareTo 方法，就可以实现排序

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
@Override
public int compareTo(FlowBean o) {
    // 倒序排列，从大到小
    return this.sumFlow > o.getSumFlow() ? -1 : 1;
}
```

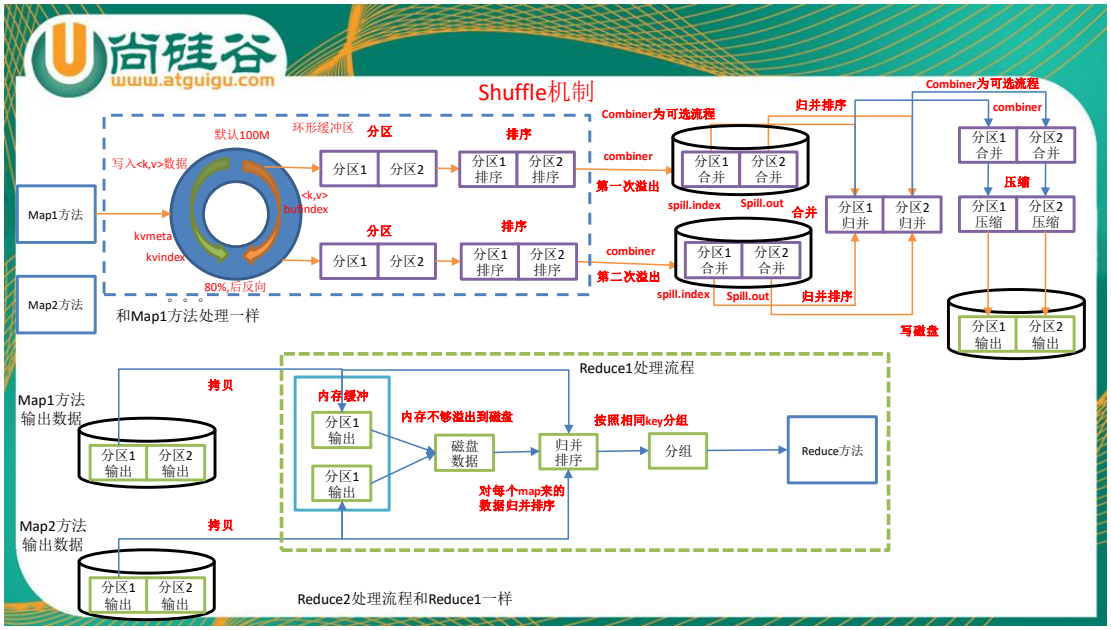
3）排序发生的阶段：

（1）一个是在 map side 发生在 spill 后 partition 前。

（2）一个是在 reduce side 发生在 copy 后 reduce 前。

9. 请描述 mapReduce 中 shuffle 阶段的工作流程，如何优化 shuffle 阶段？

分区，排序，溢写，拷贝到对应 reduce 机器上，增加 combiner，压缩溢写的文件。



10. 请描述 mapReduce 中 combiner 的作用是什么，一般使用情景，哪些情况不需要，及和 reduce 的区别？

1）Combiner 的意义就是对每一个 maptask 的输出进行局部汇总，以减小网络传输量。

2）Combiner 能够应用的前提是不能影响最终的业务逻辑，而且，Combiner 的输出 kv 应该跟 reducer 的输入 kv 类型要对应起来。

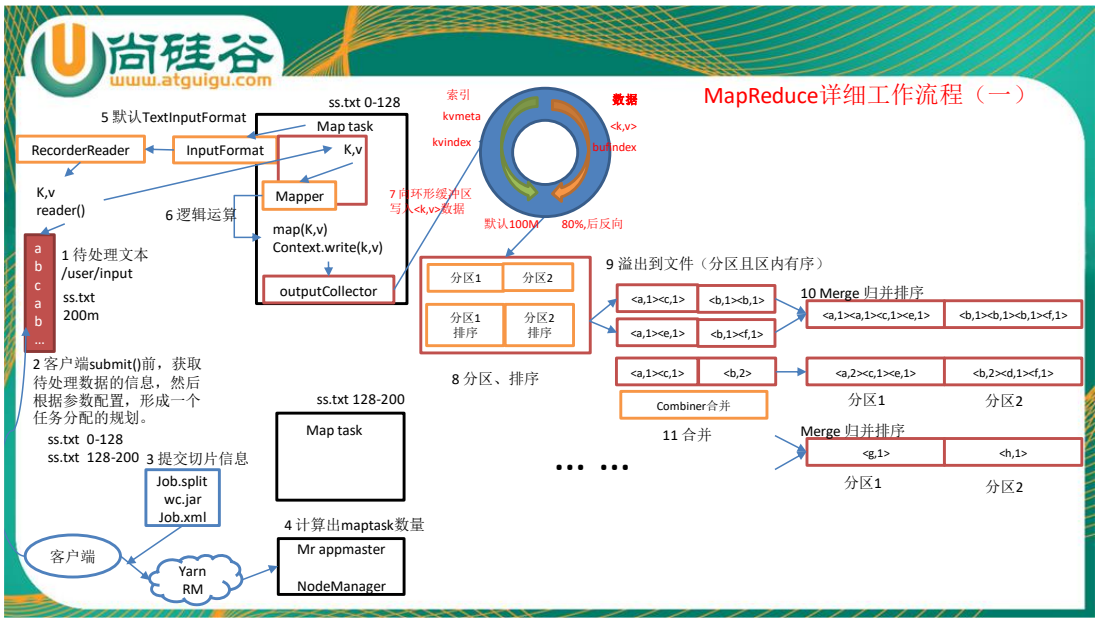
3）Combiner 和 reducer 的区别在于运行的位置。

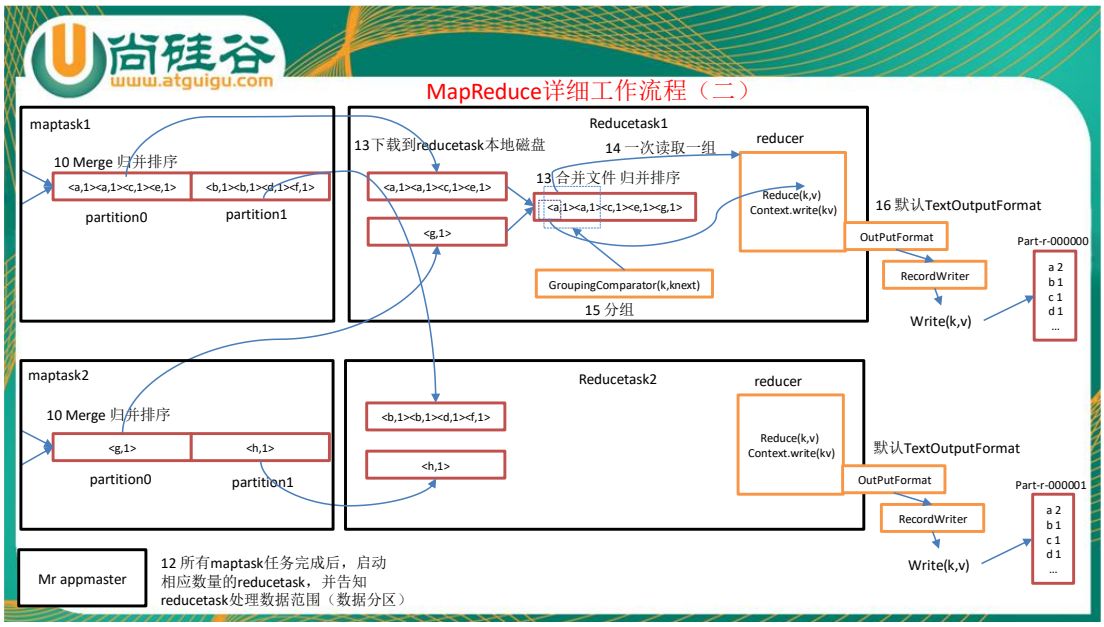
Combiner 是在每一个 maptask 所在的节点运行；

Reducer 是接收全局所有 Mapper 的输出结果。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

11.Mapreduce 的工作原理，请举例子说明 mapreduce 是怎么运行的？





12. 如果没有定义 partitioner，那数据在被送达 reducer 前是如何被分区的？

如果没有自定义的 partitioning，则默认的 partition 算法，即根据每一条数据的 key 的 hashcode 值摸运算（%）reduce 的数量，得到的数字就是“分区号”。

13. MapReduce 怎么实现 TopN？

可以自定义 groupingcomparator，或者在 map 端对数据进行排序，然后再 reduce 输出时，控制只输出前 n 个数。就达到了 topn 输出的目的。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

14. 有可能使 Hadoop 任务输出到多个目录中么？如果可以，怎么做？

1）可以输出到多个目录中，采用自定义 OutputFormat。

2）实现步骤：

（1）自定义 outputformat，

（2）改写 recordwriter，具体改写输出数据的方法 write()

15. 简述 hadoop 实现 join 的几种方法及每种方法的实现。

1）reduce side join

Map 端的主要工作：为来自不同表(文件)的 key/value 对打标签以区别不同来源的记录。然后用连接字段作为 key，其余部分和新加的标志作为 value，最后进行输出。

Reduce 端的主要工作：在 reduce 端以连接字段作为 key 的分组已经完成，我们只需要在每一个分组当中将那些来源于不同文件的记录(在 map 阶段已经打标志)分开，最后进行合并就 ok 了。

2）map join

在 map 端缓存多张表，提前处理业务逻辑，这样增加 map 端业务，减少 reduce 端数据的压力，尽可能的减少数据倾斜。

具体办法：采用 distributedcache

（1）在 mapper 的 setup 阶段，将文件读取到缓存集合中。

（2）在驱动函数中加载缓存。

job.addCacheFile(new URI("file:/e:/mapjoincache/pd.txt"));// 缓存普通文件到 task 运行节点

16. 请简述 hadoop 怎样实现二级排序。

对 map 端输出的 key 进行排序，实现的 compareTo 方法。 在 compareTo 方法中排序的条件有二个。

17. 参考下面的 MR 系统的场景:

--hdfs 块的大小为 128MB

--输入类型为 FileInputFormat

--有三个文件的大小分别是:64KB 130MB 260MB

Hadoop 框架会把这些文件拆分为多少块？

4 块：64K，130M，128M，132M

18. Hadoop 中 RecordReader 的作用是什么？

（1）以怎样的方式从分片中读取一条记录，每读取一条记录都会调用 RecordReader 类；

（2）系统默认的 RecordReader 是 LineRecordReader

（3）LineRecordReader 是用每行的偏移量作为 map 的 key，每行的内容作为 map 的 value；

（4）应用场景：自定义读取每一条记录的方式；自定义读入 key 的类型，如希望读取的 key 是文件的路径或名字而不是该行在文件中的偏移量。

19. 给你一个 1G 的数据文件。分别有 id,name,mark,source 四个字段，按照 mark 分组，id 排序，手写一个 MapReduce?其中有几个 Mapper？

在 map 端对 mark 排序，在 reduce 端对 id 分组。

```
    @Override
    public int compareTo(GroupBean o) {
        int result = this.mark.compareTo(o.mark);
        if (result == 0)
            return Integer.compare(this.id,o.id);
        else
            return result;
    }
```

```
@Override
    public int compare(WritableComparable a, WritableComparable b) {

        GroupBean aBean = (GroupBean) a;
        GroupBean bBean = (GroupBean) b;

        int result;
        if (aBean.getMark() > bBean. getMark()) {
            result = 1;
        } else if (aBean. getMark() < bBean. getMark()) {
            result = -1;
        } else {
            result = 0;
        }
```

```
        return result;
    }
```

2）几个 mapper

（1）1024m/128m=8 块

Yarn

1. 简述 Hadoop1 与 Hadoop2 的架构异同。

加入了 yarn 解决了资源调度的问题。

加入了对 zookeeper 的支持实现比较可靠的高可用。

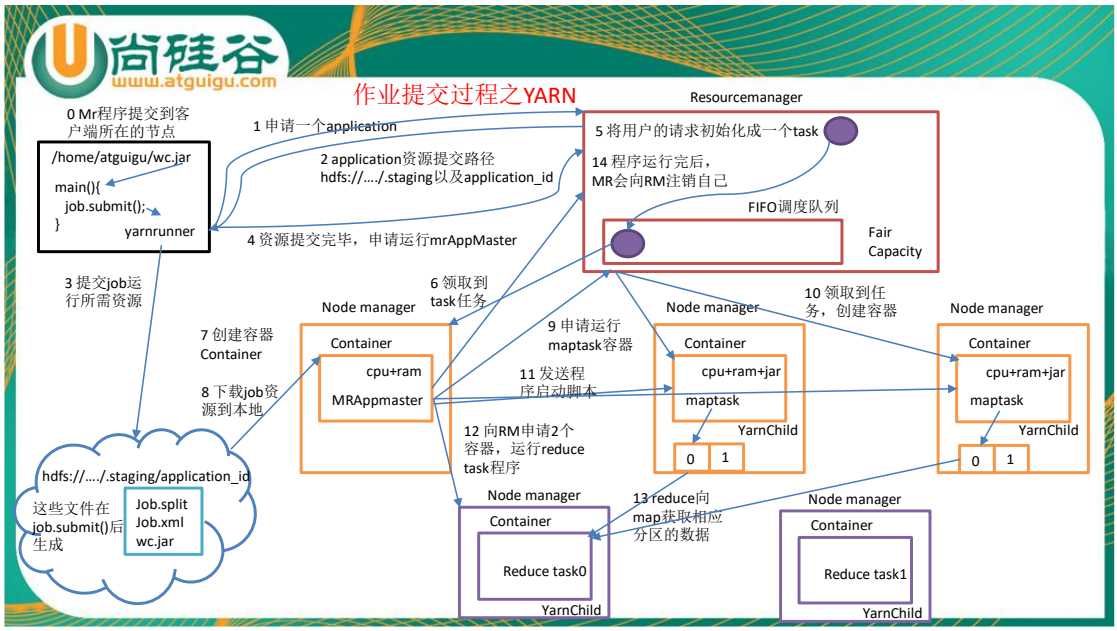
2. 为什么会产生 yarn,它解决了什么问题，有什么优势？

Yarn 最主要的功能就是解决运行的用户程序与 yarn 框架完全解耦。

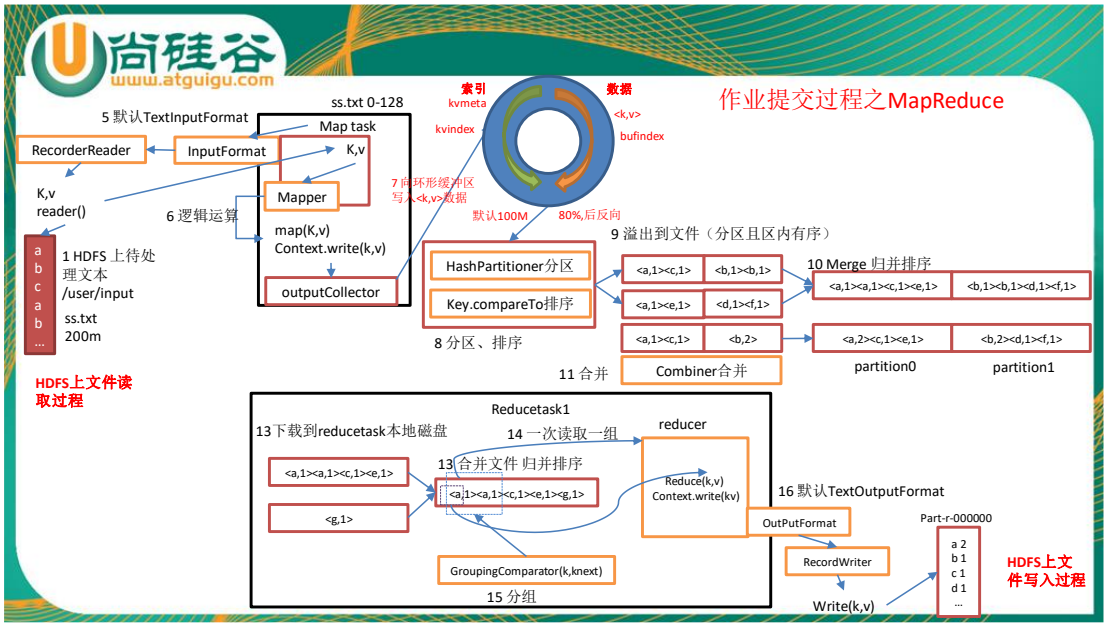
Yarn 上可以运行各种类型的分布式运算程序（mapreduce 只是其中的一种），比如 mapreduce、storm 程序，spark 程序……

3. MR 作业提交全过程。

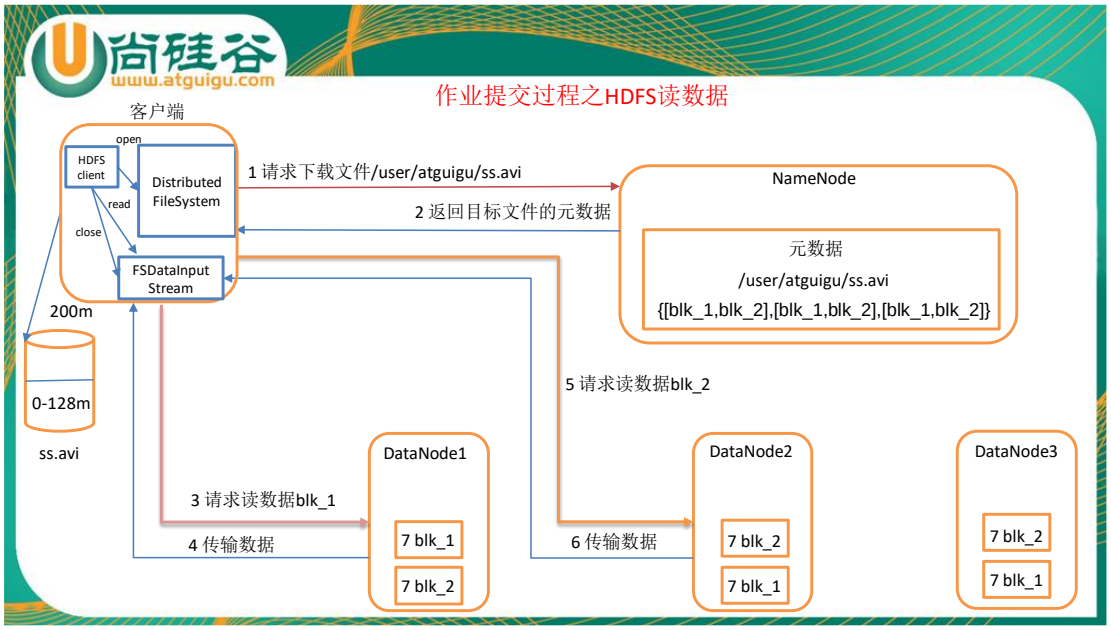
1）作业提交过程之 YARN



2）作业提交过程之 MapReduce

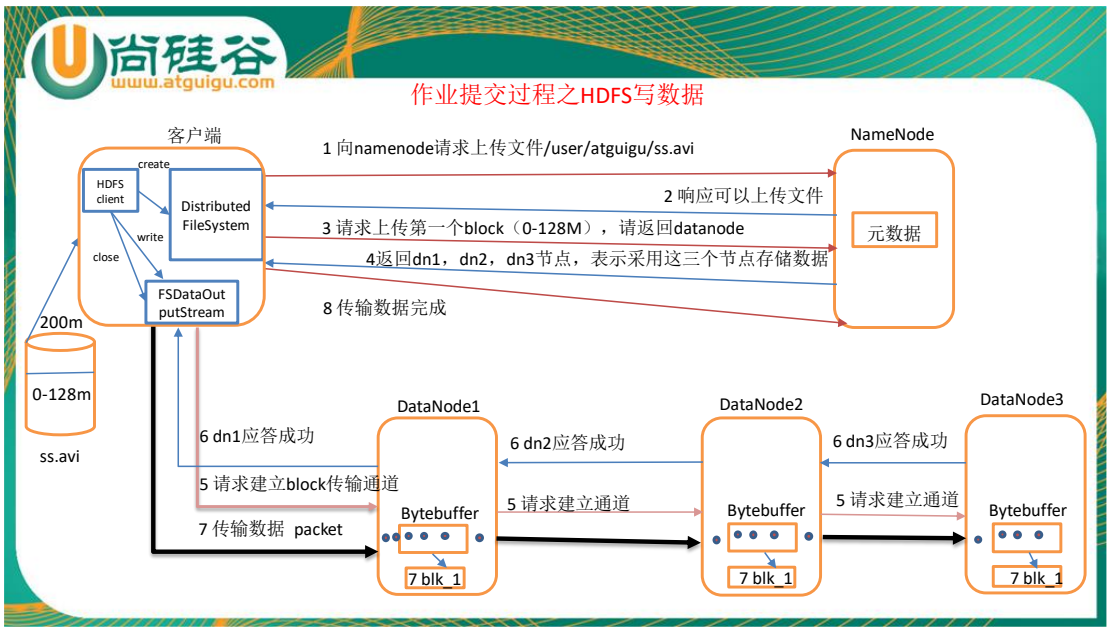


3）作业提交过程之读数据



4）作业提交过程之写数据

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】



4. HDFS 的数据压缩算法？及每种算法的应用场景？

1）gzip 压缩

优点：压缩率比较高，而且压缩/解压速度也比较快；hadoop 本身支持，在应用中处理 gzip 格式的文件就和直接处理文本一样；大部分 linux 系统都自带 gzip 命令，使用方便。

缺点：不支持 split。

应用场景：当每个文件压缩之后在 130M 以内的（1 个块大小内），都可以考虑用 gzip 压缩格式。例如说一天或者一个小时的日志压缩成一个 gzip 文件，运行 mapreduce 程序的时候通过多个 gzip 文件达到并发。hive 程序，streaming 程序，和 java 写的 mapreduce 程序完全和文本处理一样，压缩之后原来的程序不需要做任何修改。

2）Bzip2 压缩

优点：支持 split；具有很高的压缩率，比 gzip 压缩率都高；hadoop 本身支持，但不支持 native；在 linux 系统下自带 bzip2 命令，使用方便。

缺点：压缩/解压速度慢；不支持 native。

应用场景：适合对速度要求不高，但需要较高的压缩率的时候，可以作为 mapreduce 作业的输出格式；或者输出之后的数据比较大，处理之后的数据需要压缩存档减少磁盘空间并且以后数据用得比较少的情况；或者对单个很大的文本文件想压缩减少存储空间，同时又需要支持 split，而且兼容之前的应用程序（即应用程序不需要修改）的情况。

3）Lzo 压缩

优点：压缩/解压速度也比较快，合理的压缩率；支持 split，是 hadoop 中最流行的压缩

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

格式；可以在 linux 系统下安装 lzop 命令，使用方便。

缺点：压缩率比 gzip 要低一些；hadoop 本身不支持，需要安装；在应用中对 lzo 格式的文件需要做一些特殊处理（为了支持 split 需要建索引，还需要指定 inputformat 为 lzo 格式）。

应用场景：一个很大的文本文件，压缩之后还大于 200M 以上的可以考虑，而且单个文件越大，lzo 优点越越明显。

4）Snappy 压缩

优点：高速压缩速度和合理的压缩率。

缺点：不支持 split；压缩率比 gzip 要低；hadoop 本身不支持，需要安装；

应用场景：当 Mapreduce 作业的 Map 输出的数据比较大的时候，作为 Map 到 Reduce 的中间数据的压缩格式；或者作为一个 Mapreduce 作业的输出和另外一个 Mapreduce 作业的输入。

5. Hadoop 的调度器总结。

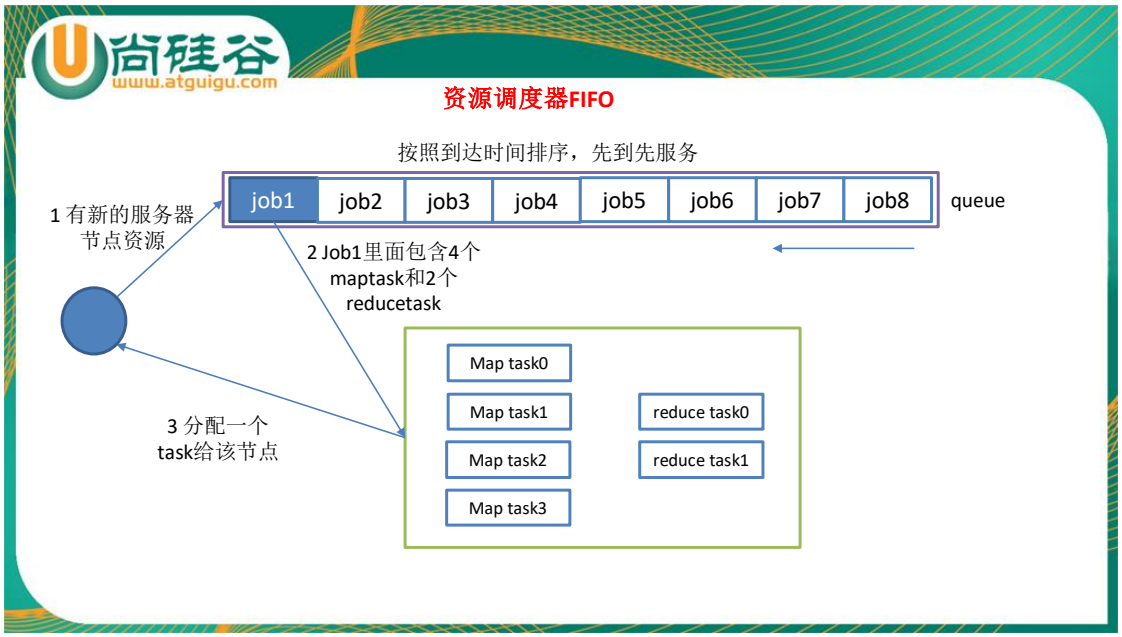
目前，Hadoop 作业调度器主要有三种：FIFO、Capacity Scheduler 和 Fair Scheduler。Hadoop2.7.2 默认的资源调度器是 Capacity Scheduler。

具体设置详见：yarn-default.xml 文件

```
<property>
    <description>The class to use as the resource scheduler.</description>
    <name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</
value>
</property>
```

1）先进先出调度器（FIFO）

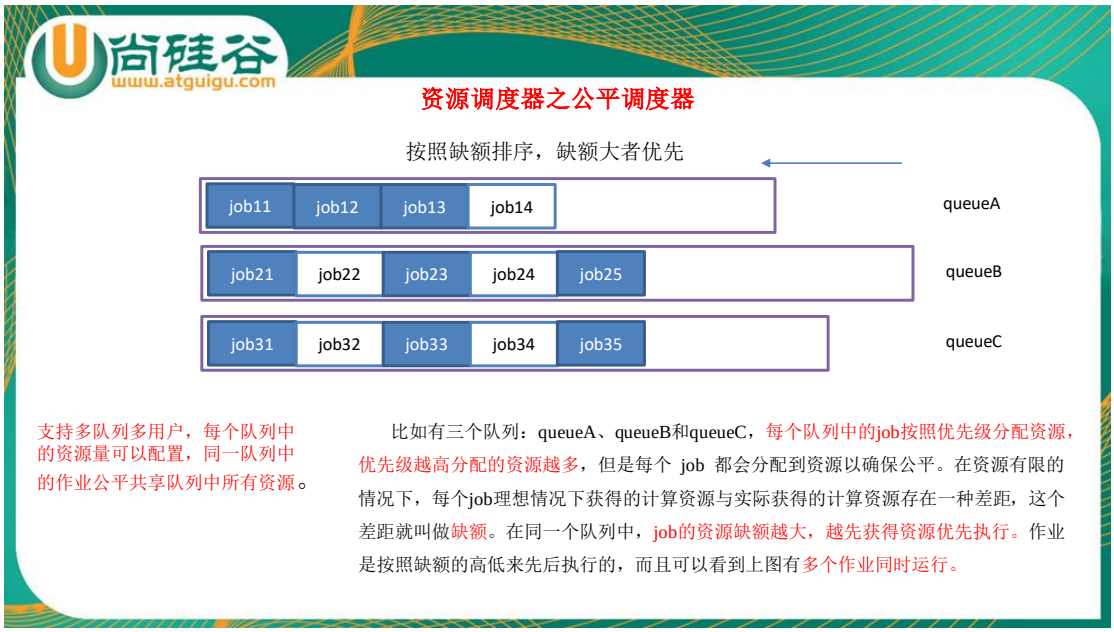
【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】



2）容量调度器（Capacity Scheduler）

3）公平调度器（Fair Scheduler）

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】



6. mapreduce 推测执行算法及原理。

1）作业完成时间取决于最慢的任务完成时间

一个作业由若干个 Map 任务和 Reduce 任务构成。因硬件老化、软件 Bug 等，某些任务可能运行非常慢。

典型案例：系统中有 99%的 Map 任务都完成了，只有少数几个 Map 老是进度很慢，完不成，怎么办？

2）推测执行机制：

发现拖后腿的任务，比如某个任务运行速度远慢于任务平均速度。为拖后腿任务启动一个备份任务，同时运行。谁先运行完，则采用谁的结果。

3）执行推测任务的前提条件

（1）每个 task 只能有一个备份任务；

（2）当前 job 已完成的 task 必须不小于 0.05（5%）

（3）开启推测执行参数设置。Hadoop2.7.2 mapred-site.xml 文件中默认是打开的。

```
<property>
    <name>mapreduce.map.speculative</name>
    <value>true</value>
    <description>If true, then multiple instances of some map tasks
                    may be executed in parallel.</description>
</property>

<property>
    <name>mapreduce.reduce.speculative</name>
    <value>true</value>
```

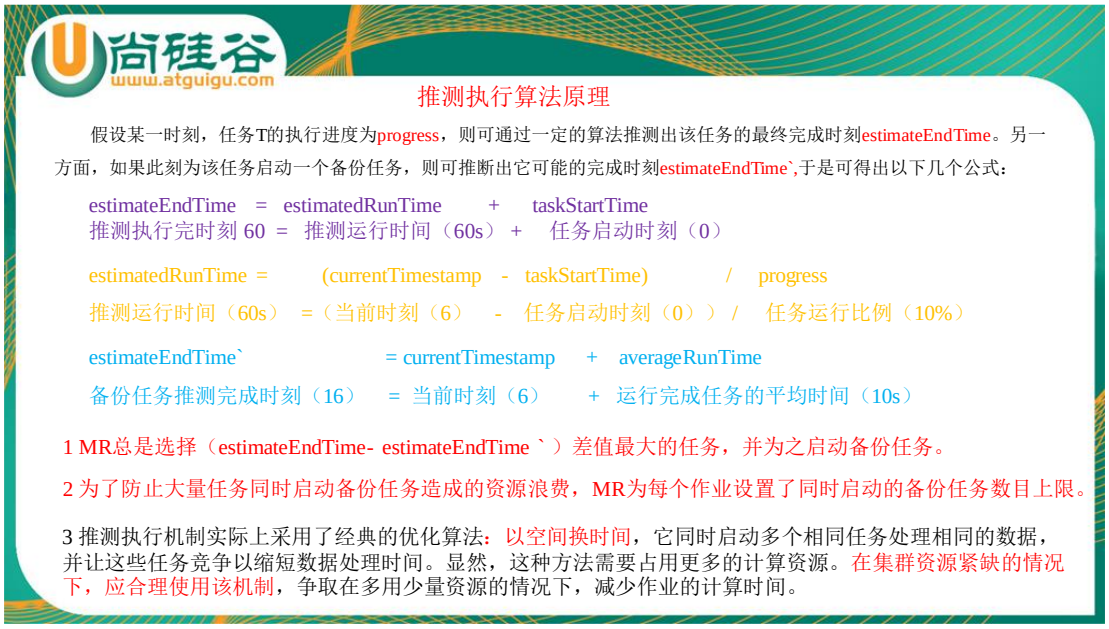
```
    <description>If true, then multiple instances of some reduce tasks
                    may be executed in parallel.</description>
</property>
```

4）不能启用推测执行机制情况

（1）任务间存在严重的负载倾斜；

（2）特殊任务，比如任务向数据库中写数据。

5）算法原理：



优化

1. mapreduce 跑的慢的原因？

Mapreduce 程序效率的瓶颈在于两点：

1）计算机性能

CPU、内存、磁盘健康、网络

2）I/O 操作优化

（1）数据倾斜

（2）map 和 reduce 数设置不合理

（3）reduce 等待过久

（4）小文件过多

（5）大量的不可分块的超大文件

（6）spill 次数过多

（7）merge 次数过多等。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

2. mapreduce 优化方法。

1）数据输入：

（1）合并小文件：在执行 mr 任务前将小文件进行合并，大量的小文件会产生大量的 map 任务，增大 map 任务装载次数，而任务的装载比较耗时，从而导致 mr 运行较慢。

（2）采用 ConbinFileInputFormat 来作为输入，解决输入端大量小文件场景。

2）map 阶段

（1）减少 spill 次数：通过调整 io.sort.mb 及 sort.spill.percent 参数值，增大触发 spill 的内存上限，减少 spill 次数，从而减少磁盘 IO。

（2）减少 merge 次数：通过调整 io.sort.factor 参数，增大 merge 的文件数目，减少 merge 的次数，从而缩短 mr 处理时间。

（3）在 map 之后先进行 combine 处理，减少 I/O。

3）reduce 阶段

（1）合理设置 map 和 reduce 数：两个都不能设置太少，也不能设置太多。太少，会导致 task 等待，延长处理时间；太多，会导致 map、reduce 任务间竞争资源，造成处理超时等错误。

（2）设置 map、reduce 共存：调整 slowstart.completedmaps 参数，使 map 运行到一定程度后，reduce 也开始运行，减少 reduce 的等待时间。

（3）规避使用 reduce，因为 Reduce 在用于连接数据集的时候将会产生大量的网络消耗。

（4）合理设置 reduc 端的 buffer，默认情况下，数据达到一个阈值的时候，buffer 中的数据就会写入磁盘，然后 reduce 会从磁盘中获得所有的数据。也就是说，buffer 和 reduce 是没有直接关联的，中间多个一个写磁盘->读磁盘的过程，既然有这个弊端，那么就可以通过参数来配置，使得 buffer 中的一部分数据可以直接输送到 reduce，从而减少 IO 开销：mapred.job.reduce.input.buffer.percent，默认为 0.0。当值大于 0 的时候，会保留指定比例的内存读 buffer 中的数据直接拿给 reduce 使用。这样一来，设置 buffer 需要内存，读取数据需要内存，reduce 计算也要内存，所以要根据作业的运行情况进行调整。

4）IO 传输

（1）采用数据压缩的方式，减少网络 IO 的的时间。安装 Snappy 和 LZOP 压缩编码器。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

（2）使用 SequenceFile 二进制文件

5）数据倾斜问题

（1）数据倾斜现象

数据频率倾斜——某一个区域的数据量要远远大于其他区域。

数据大小倾斜——部分记录的大小远远大于平均值。

（2）如何收集倾斜数据

在 reduce 方法中加入记录 map 输出键的详细情况的功能。

```
public static final String MAX_VALUES = "skew.maxvalues";
private int maxValueThreshold;

@Override
public void configure(JobConf job) {
    maxValueThreshold = job.getInt(MAX_VALUES, 100);
}
@Override
public void reduce(Text key, Iterator<Text> values,
                   OutputCollector<Text, Text> output,
                   Reporter reporter) throws IOException {
    int i = 0;
    while (values.hasNext()) {
      values.next();
      i++;
    }

    if (++i > maxValueThreshold) {
      log.info("Received " + i + " values for key " + key);
    }
}
```

（3）减少数据倾斜的方法

方法 1：抽样和范围分区

可以通过对原始数据进行抽样得到的结果集来预设分区边界值。

方法 2：自定义分区

另一个抽样和范围分区的替代方案是基于输出键的背景知识进行自定义分区。例如，如果 map 输出键的单词来源于一本书。其中大部分必然是省略词（stopword）。那么就可

以将自定义分区将这部分省略词发送给固定的一部分 reduce 实例。而将其他的都发送给剩余的 reduce 实例。

方法 3：Combine

使用 Combine 可以大量地减小数据频率倾斜和数据大小倾斜。在可能的情况下，combine 的目的就是聚合并精简数据。

6）常用的调优参数

（1）资源相关参数

（a）以下参数是在用户自己的 mr 应用程序中配置就可以生效（mapred-default.xml）

配置参数	参数说明
mapreduce.map.memory.mb	一个 Map Task 可使用的资源上限（单位:MB），默认为 1024。如果 Map Task 实际使用的资源量超过该值，则会被强制杀死。
mapreduce.reduce.memory.mb	一个 Reduce Task 可使用的资源上限（单位:MB），默认为 1024。如果 Reduce Task 实际使用的资源量超过该值，则会被强制杀死。
mapreduce.map.cpu.vcores	每个 Map task 可使用的最多 cpu core 数目，默认值: 1
mapreduce.reduce.cpu.vcores	每个 Reduce task 可使用的最多 cpu core 数目，默认值: 1
mapreduce.reduce.shuffle.parallelcopies	每个 reduce 去 map 中拿数据的并行数。默认值是 5
mapreduce.reduce.shuffle.merge.percent	buffer 中的数据达到多少比例开始写入磁盘。默认值 0.66
mapreduce.reduce.shuffle.input.buffer.percent	buffer 大小占 reduce 可用内存的比例。默认值 0.7
mapreduce.reduce.input.buffer.percent	指定多少比例的内存用来存放 buffer 中的数据，默认值是 0.0

（b）应该在 yarn 启动之前就配置在服务器的配置文件中才能生效（yarn-default.xml）

配置参数	参数说明
yarn.scheduler.minimum-allocation-mb 1024	给应用程序 container 分配的最小内存

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

yarn.scheduler.maximum-allocation-mb 8192	给应用程序 container 分配的最大内存
yarn.scheduler.minimum-allocation-vcores 1	每个 container 申请的最小 CPU 核数
yarn.scheduler.maximum-allocation-vcores 32	每个 container 申请的最大 CPU 核数
yarn.nodemanager.resource.memory-mb 8192	给 containers 分配的最大物理内存

（c）shuffle 性能优化的关键参数，应在 yarn 启动之前就配置好（mapred-default.xml）

配置参数	参数说明
mapreduce.task.io.sort.mb 100	shuffle 的环形缓冲区大小，默认 100m
mapreduce.map.sort.spill.percent 0.8	环形缓冲区溢出的阈值，默认 80%

（2）容错相关参数(mapreduce 性能优化)

配置参数	参数说明
mapreduce.map.maxattempts	每个 Map Task 最大重试次数，一旦重试参数超过该值，则认为 Map Task 运行失败，默认值：4。
mapreduce.reduce.maxattempts	每个 Reduce Task 最大重试次数，一旦重试参数超过该值，则认为 Map Task 运行失败，默认值：4。
mapreduce.task.timeout	Task 超时时间，经常需要设置的一个参数，该参数表达的意思为：如果一个 task 在一定时间内没有任何进入，即不会读取新的数据，也没有输出数据，则认为该 task 处于 block 状态，可能是卡住了，也许永远会卡主，为了防止因为用户程序永远 block 住不退出，则强制设置了一个该超时时间（单位毫秒），默认是 600000。如果你的程序对每条输入数据的处理时间过长（比如会访问数据库，通过网络拉取数据等），建议将该参数调大，该参数过小常出现的错误提示是“AttemptID:attempt_14267829456721_123456_m_000224_0 Timed out after 300 secsContainer killed by the ApplicationMaster.”。

3. HDFS 小文件优化方法。

1）HDFS 小文件弊端

HDFS 上每个文件都要在 namenode 上建立一个索引，这个索引的大小约为 150byte，这样当小文件比较多的时候，就会产生很多的索引文件，一方面会大量占用 namenode 的内存空间，另一方面就是索引文件过大是的索引速度变慢。

2）解决方案

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

1）Hadoop Archive:

是一个高效地将小文件放入 HDFS 块中的文件存档工具，它能够将多个小文件打包成一个 HAR 文件，这样在减少 namenode 内存使用的同时。

2）Sequence file：

sequence file 由一系列的二进制 key/value 组成，如果 key 为文件名，value 为文件内容，则可以将大批小文件合并成一个大文件。

3）CombineFileInputFormat：

CombineFileInputFormat 是一种新的 inputformat，用于将多个文件合并成一个单独的 split，另外，它会考虑数据的存储位置。

4）开启 JVM 重用

对于大量小文件 Job，可以开启 JVM 重用会减少 45%运行时间。

JVM 重用理解：一个 map 运行一个 jvm，重用的话，在一个 map 在 jvm 上运行完毕后，jvm 继续运行其他 jvm

具体设置：mapreduce.job.jvm.numtasks 值在 10-20 之间。

4. MapReduce 怎么解决数据均衡问题，如何确定分区号？

数据均衡问题指的就是某个节点或者某几个节点的任务运行的比较慢，拖慢了整个 Job 的进度。实际上数据均衡问题就是数据倾斜问题，解决方案同解决数据倾斜的方案。MapReduce 中分区默认是按 hashcode 来分的，用户可以自定义分区类，需要继承系统的 Partitioner 类，重写 getPartition()方法即可。

5. Hadoop 中 job 和 Tasks 之间的区别是什么？

编写好的一个程序，我们称为 Mapreduce 程序，一个 Mapreduce 程序就是一个 Job，而一个 Job 里面可以有一个或多个 Task，Task 又可以区分为 Map Task 和 Reduce Task.

Zookeeper

1. 请简述 ZooKeeper 的选举机制

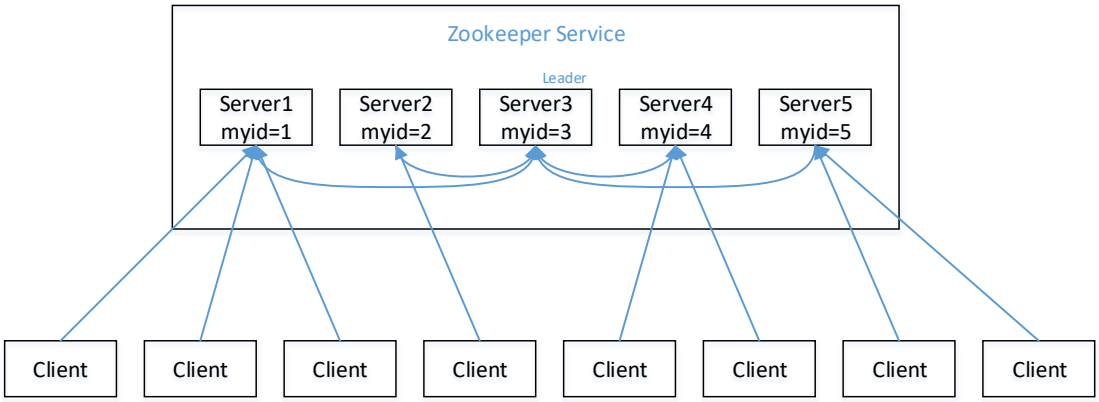
1）半数机制（Paxos 协议）：集群中半数以上机器存活，集群可用。所以 zookeeper 适合装在奇数台机器上。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

2）Zookeeper 虽然在配置文件中并没有指定 master 和 slave。但是，zookeeper 工作时，是有一个节点为 leader，其他则为 follower，Leader 是通过内部的选举机制临时产生的

3）以一个简单的例子来说明整个选举的过程。

假设有五台服务器组成的 zookeeper 集群，它们的 id 从 1-5，同时它们都是最新启动的，也就是没有历史数据，在存放数据量这一点上，都是一样的。假设这些服务器依序启动，来看看会发生什么。



（1）服务器 1 启动，此时只有它一台服务器启动了，它发出去的报没有任何响应，所以它的选举状态一直是 LOOKING 状态。

（2）服务器 2 启动，它与最开始启动的服务器 1 进行通信，互相交换自己的选举结果，由于两者都没有历史数据，所以 id 值较大的服务器 2 胜出，但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是 3)，所以服务器 1、2 还是继续保持 LOOKING 状态。

（3）服务器 3 启动，根据前面的理论分析，服务器 3 成为服务器 1、2、3 中的老大，而与上面不同的是，此时有三台服务器选举了它，所以它成为了这次选举的 leader。

（4）服务器 4 启动，根据前面的分析，理论上服务器 4 应该是服务器 1、2、3、4 中最大的，但是由于前面已经有半数以上的服务器选举了服务器 3，所以它只能接收当小弟的命了。

（5）服务器 5 启动，同 4 一样当小弟。

2. ZooKeeper 的监听原理是什么？

1）首先要有一个 main()线程

2）在 main 线程中创建 Zookeeper 客户端，这时就会创建两个线程，一个负责网络连接通信（connet），一个负责监听（listener）。

3）通过 connect 线程将注册的监听事件发送给 Zookeeper。

4）在 Zookeeper 的注册监听器列表中将注册的监听事件添加到列表中。

5）Zookeeper 监听到有数据或路径变化，就会将这个消息发送给 listener 线程。

6）listener 线程内部调用了 proces()方法。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

3. ZooKeeper 的部署方式有哪几种？集群中的角色有哪些？集群最少需要几台机器？

1. 单机模式、伪集群模式和集群模式
2. Leader 和 Follower
3. 3

4. ZooKeeper 的常用命令

ls create get delete set…

Hive

1. Hive 表关联查询，如何解决数据倾斜的问题？

1） 过滤掉脏数据：如果大 key 是无意义的脏数据，直接过滤掉。本场景中大 key 无实际意义，为非常脏数据，直接过滤掉。

2）数据预处理：数据做一下预处理，尽量保证 join 的时候，同一个 key 对应的记录不要有太多。

3） 增加 reduce 个数：如果数据中出现了多个大 key，增加 reduce 个数，可以让这些大 key 落到同一个 reduce 的概率小很多。

4） 转换为 mapjoin：如果两个表 join 的时候，一个表为小表，可以用 mapjoin 做。

5） 大 key 单独处理：将大 key 和其他 key 分开处理

6）hive.optimize.skewjoin：会将一个 join sql 分为两个 job。另外可以同时设置下 hive.skewjoin.key，默认为 10000。参考：

https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties

参数对 full outer join 无效。

7）调整内存设置：适用于那些由于内存超限内务被 kill 掉的场景。通过加大内存起码能让任务跑起来，不至于被杀掉。该参数不一定会明显降低任务执行时间。如：

setmapreduce.reduce.memory.mb=5120 ;

setmapreduce.reduce.java.opts=-Xmx5000M -XX:MaxPermSize=128m ;

2. 请谈一下 Hive 的特点，Hive 和 RDBMS 有什么异同？

特点：

1）Hive 处理的数据存储在 HDFS

2）Hive 分析数据底层的实现是 MapReduce

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

3）执行程序运行在 Yarn 上

异同：

1）查询语言：由于 SQL 被广泛的应用在数据仓库中，因此，专门针对 Hive 的特性设计了类 SQL 的查询语言 HQL。熟悉 SQL 开发的开发者可以很方便的使用 Hive 进行开发。

2） 数据存储位置：Hive 是建立在 Hadoop 之上的，所有 Hive 的数据都是存储在 HDFS 中的。而数据库则可以将数据保存在块设备或者本地文件系统中。

3）数据更新：由于 Hive 是针对数据仓库应用设计的，而数据仓库的内容是读多写少的。因此，Hive 中不建议对数据的改写，所有的数据都是在加载的时候确定好的。而数据库中的数据通常是需要经常进行修改的，因此可以使用 INSERT INTO … VALUES 添加数据，使用 UPDATE … SET 修改数据。

4）索引：Hive 在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些 Key 建立索引。Hive 要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延迟较高。由于 MapReduce 的引入， Hive 可以并行访问数据，因此即使没有索引，对于大数据量的访问，Hive 仍然可以体现出优势。数据库中，通常会针对一个或者几个列建立索引，因此对于少量的特定条件的数据的访问，数据库可以有很高的效率，较低的延迟。由于数据的访问延迟较高，决定了 Hive 不适合在线数据查询。

5） 执行：Hive 中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的。而数据库通常有自己的执行引擎。

6） 执行延迟：Hive 在查询数据的时候，由于没有索引，需要扫描整个表，因此延迟较高。另外一个导致 Hive 执行延迟高的因素是 MapReduce 框架。由于 MapReduce 本身具有较高的延迟，因此在利用 MapReduce 执行 Hive 查询时，也会有较高的延迟。相对的，数据库的执行延迟较低。当然，这个低是有条件的，即数据规模较小，当数据规模大到超过数据库的处理能力的时候，Hive 的并行计算显然能体现出优势。

7） 可扩展性：由于 Hive 是建立在 Hadoop 之上的，因此 Hive 的可扩展性是和 Hadoop 的可扩展性是一致的（世界上最大的 Hadoop 集群在 Yahoo!，2009 年的规模在 4000 台节点左右）。而数据库由于 ACID 语义的严格限制，扩展行非常有限。目前最先进的并行数据库 Oracle 在理论上的扩展能力也只有 100 台左右。

8）数据规模：由于 Hive 建立在集群上并可以利用 MapReduce 进行并行计算，因此可以支持很大规模的数据；对应的，数据库可以支持的数据规模较小。

3. 请说明 Hive 中 Sort By，Order By，Cluster By，Distrbute By 各代表什么意思？

Sort By：每个 Reducer 内部有序；

Order By：全局排序，只有一个 Reducer；

Cluster By：当 distribute by 和 sorts by 字段相同时，可以使用 cluster by 方式。cluster by 除了具有 distribute by 的功能外还兼具 sort by 的功能。但是排序只能是倒序排序，不能指定排序规则为 ASC 或者 DESC。

Distrbute By：类似 MR 中 partition，进行分区，结合 sort by 使用，Hive 要求 DISTRIBUTE BY 语句要写在 SORT BY 语句之前。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

4. Hive 有哪些方式保存元数据，各有哪些特点？

1）Single User Mode：默认安装 hive，hive 是使用 derby 内存数据库保存 hive 的元数据，这样是不可以并发调用 hive 的。

2）User Mode：通过网络连接到一个数据库中，是最经常使用到的模式。假设使用本机 mysql 服务器存储元数据。这种存储方式需要在本地运行一个 mysql 服务器，可并发调用

3）Remote Server Mode：在服务器端启动一个 MetaStoreServer，客户端利用 Thrift 协议通过 MetaStoreServer 访问元数据库。

5. Hive 内部表外部表的区别？

1）默认创建的表都是管理表，有时也被称为内部表。因为这种表，Hive 会（或多或少地）控制着数据的生命周期。Hive 默认情况下会将这些表的数据存储在由配置项 hive.metastore.warehouse.dir(例如，/user/hive/warehouse)所定义的目录的子目录下。 当我们删除一个管理表时，Hive 也会删除这个表中数据。管理表不适合和其他工具共享数据。

2）Hive 并非认为其完全拥有这份数据。删除该表并不会删除掉这份数据，不过描述表的元数据信息会被删除掉。

6. 写出将 text.txt 文件放入 Hive 中 test 表‘2016-10-10’分区的语句，test 的分区字段是 l_date。

load data local inpath ‘/text.txt’ into table test partition(l_date=’ 2016-10-10’);

7. Hive 自定义 UDF 函数的流程?

1）自定义一个 Java 类
2）继承 UDF 类
3）重写 evaluate 方法
4）打成 jar 包
6）在 hive 执行 add jar 方法
7）在 hive 执行创建模板函数
8）hql 中使用

8. 对于 Hive，你写过哪些 udf 函数，作用是什么？

时间日期类 UDF 函数，传入时间，返回年，月，日…

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

9. Hive 中的压缩格式 TextFile、SequenceFile、RCfile 、ORCfile 各有什么区别？

TextFile: 默认格式，数据不做压缩，磁盘开销大，数据解析开销大。可结合 Gzip、Bzip2 使用，但使用 Gzip 这种方式，hive 不会对数据进行切分，从而无法对数据进行并行操作。

SequenceFile: SequenceFile 是 Hadoop API 提供的一种二进制文件，它将数据以<key,value>的形式序列化到文件中。这种二进制文件内部使用 Hadoop 的标准的 Writable 接口实现序列化和反序列化。它与 Hadoop API 中的 MapFile 是互相兼容的。Hive 中的 SequenceFile 继承自 Hadoop API 的 SequenceFile，不过它的 key 为空，使用 value 存放实际的值， 这样是为了避免 MR 在运行 map 阶段的排序过程。

RCFile: RCFile 是 Hive 推出的一种专门面向列的数据格式。 它遵循“先按列划分，再垂直划分”的设计理念。当查询过程中，针对它并不关心的列时，它会在 IO 上跳过这些列。需要说明的是，RCFile 在 map 阶段从 远端拷贝仍然是拷贝整个数据块，并且拷贝到本地目录后 RCFile 并不是真正直接跳过不需要的列，并跳到需要读取的列， 而是通过扫描每一个 row group 的头部定义来实现的，但是在整个 HDFS Block 级别的头部并没有定义每个列从哪个 row group 起始到哪个 row group 结束。所以在读取所有列的情况下，RCFile 的性能反而没有 SequenceFile 高。

ORCfile: ORC 是列式存储，有多种文件压缩方式，并且有着很高的压缩比。文件是可切分（Split）的。因此，在 Hive 中使用 ORC 作为表的文件存储格式，不仅节省 HDFS 存储资源，查询任务的输入数据量减少，使用的 MapTask 也就减少了。提供了多种索引，row group index、bloom filter index。ORC 可以支持复杂的数据结构（比如 Map 等）

10. Hive join 过程中大表小表的放置顺序？

小表在前，大表在后。Hive 会将小表进行缓存。在 0.7 版本后。也能够用配置来自己主动优化：set hive.auto.convert.join=true;

11. Hive 的两张表关联，使用 MapReduce 怎么实现？

可以将小表缓存在 map 端实现 map 端 join 防止数据倾斜。并将 join 字段作为 key，在 Reducer 阶段实际执行相应的业务处理。

12. 所有的 Hive 任务都会有 MapReduce 的执行吗？

不是。Hive 中对某些情况的查询可以不必使用 MapReduce 计算。例如：SELECT * FROM employees;在这种情况下，Hive 可以简单地读取 employee 对应的存储目录下的文件，然后输出查询结果到控制台。

在 hive-default.xml.template 文件中 hive.fetch.task.conversion 默认是 more，老版本 hive

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

默认是 minimal，该属性修改为 more 以后，在全局查找、字段查找、limit 查找等都不走 mapreduce。

13.Hive 的函数：UDF、UDAF、UDTF 的区别？

（1）UDF（User-Defined-Function）
一进一出
（2）UDAF（User-Defined Aggregation Function）
聚集函数，多进一出
类似于：count/max/min
（3）UDTF（User-Defined Table-Generating Functions）
一进多出
如 lateral view explore()

14.说说对 Hive 桶表的理解？

分区针对的是数据的存储路径，分桶针对的是数据文件。分区提供一个隔离数据和优化查询的便利方式。不过，并非所有的数据集都可形成合理的分区，特别是之前所提到过的要确定合适的划分大小这个疑虑。

分桶是将数据集分解成更容易管理的若干部分的另一个技术。

15.Hive 可以像关系型数据库那样建立多个库吗？

可以

16.Hive 实现统计的查询语句是什么？

聚合函数：count(),sum(),avg(),max(),min()

17.Hive 优化措施

17.1 Fetch 抓取

Fetch 抓取是指，Hive 中对某些情况的查询可以不必使用 MapReduce 计算。例如：SELECT * FROM employees;在这种情况下，Hive 可以简单地读取 employee 对应的存储目录下的文件，然后输出查询结果到控制台。

在 hive-default.xml.template 文件中 hive.fetch.task.conversion 默认是 more，老版本 hive

默认是 minimal，该属性修改为 more 以后，在全局查找、字段查找、limit 查找等都不走 mapreduce。

```
      <property>
         <name>hive.fetch.task.conversion</name>
         <value>more</value>
         <description>
           Expects one of [none, minimal, more].
           Some select queries can be converted to single FETCH task minimizing
latency.
           Currently the query should be single sourced not having any subquery and
should not have
           any aggregations or distincts (which incurs RS), lateral views and joins.
           0. none : disable hive.fetch.task.conversion
           1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only
           2. more  : SELECT, FILTER, LIMIT only (support TABLESAMPLE and virtual
columns)
         </description>
       </property>
```

案例实操：

1）把 hive.fetch.task.conversion 设置成 none，然后执行查询语句，都会执行 mapreduce 程序。

```
hive (default)> set hive.fetch.task.conversion=none;
hive (default)> select * from emp;
hive (default)> select ename from emp;
hive (default)> select ename from emp limit 3;
```

2）把 hive.fetch.task.conversion 设置成 more，然后执行查询语句，如下查询方式都不会执行 mapreduce 程序。

```
hive (default)> set hive.fetch.task.conversion=more;
hive (default)> select * from emp;
hive (default)> select ename from emp;
hive (default)> select ename from emp limit 3;
```

17.2 本地模式

大多数的 Hadoop Job 是需要 Hadoop 提供的完整的可扩展性来处理大数据集的。不过，有时 Hive 的输入数据量是非常小的。在这种情况下，为查询触发执行任务消耗的时间可能会比实际 job 的执行时间要多的多。对于大多数这种情况，Hive 可以通过本地模式在单台机器上处理所有的任务。对于小数据集，执行时间可以明显被缩短。

用户可以通过设置 hive.exec.mode.local.auto 的值为 true，来让 Hive 在适当的时候自动启动这个优化。

```
set hive.exec.mode.local.auto=true;  //开启本地 mr
//设置 local mr 的最大输入数据量，当输入数据量小于这个值时采用 local  mr 的方式，默认为 134217728，即 128M
set hive.exec.mode.local.auto.inputbytes.max=50000000;
//设置 local mr 的最大输入文件个数，当输入文件个数小于这个值时采用 local mr 的方式，默认为 4
set hive.exec.mode.local.auto.input.files.max=10;
```

案例实操：

1）开启本地模式，并执行查询语句

```
hive (default)> set hive.exec.mode.local.auto=true;
hive (default)> select * from emp cluster by deptno;
Time taken: 1.328 seconds, Fetched: 14 row(s)
```

2）关闭本地模式，并执行查询语句

```
hive (default)> set hive.exec.mode.local.auto=false;
hive (default)> select * from emp cluster by deptno;
Time taken: 20.09 seconds, Fetched: 14 row(s)
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

17.3 表的优化

17.3.1 小表、大表 Join

将 key 相对分散，并且数据量小的表放在 join 的左边，这样可以有效减少内存溢出错误发生的几率；再进一步，可以使用 map join 让小的维度表（1000 条以下的记录条数）先进内存。在 map 端完成 reduce。

实际测试发现：新版的 hive 已经对小表 JOIN 大表和大表 JOIN 小表进行了优化。小表放在左边和右边已经没有明显区别。

案例实操

1. 需求

测试大表 JOIN 小表和小表 JOIN 大表的效率

2. 建大表、小表和 JOIN 后表的语句

```
// 创建大表
create table bigtable(id bigint, time bigint, uid string, keyword string, url rank int, click num int, click url string) row format delimited fields terminated by '\t';
// 创建小表
create table smalltable(id bigint, time bigint, uid string, keyword string, url_rank int, click_num int, click_url string) row format delimited fields terminated by '\t';
// 创建 join 后表的语句
create table jointable(id bigint, time bigint, uid string, keyword string, url_rank int, click_num int, click_url string) row format delimited fields terminated by '\t';
```

3. 分别向大表和小表中导入数据

```
hive (default)> load data local inpath '/opt/module/datas/bigtable' into table bigtable;
hive (default)>load data local inpath '/opt/module/datas/smalltable' into table smalltable;
```

4. 关闭 mapjoin 功能（默认是打开的）

```
set hive.auto.convert.join = false;
```

5. 执行小表 JOIN 大表语句

```
insert overwrite table jointable
select b.id, b.time, b.uid, b.keyword, b.url rank, b.click num, b.click url
from smalltable s
left join bigtable b
on b.id = s.id;
```

Time taken: 35.921 seconds

6. 执行大表 JOIN 小表语句

```
insert overwrite table jointable
select b.id, b.time, b.uid, b.keyword, b.url_rank, b.click_num, b.click_url
from bigtable b
left join smalltable s
on s.id = b.id;
```

Time taken: 34.196 seconds

17.3.2 大表 Join 大表

1. 空 KEY 过滤

有时 join 超时是因为某些 key 对应的数据太多，而相同 key 对应的数据都会发送到相同的 reducer 上，从而导致内存不够。此时我们应该仔细分析这些异常的 key，很多情况

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

下，这些 key 对应的数据是异常数据，我们需要在 SQL 语句中进行过滤。例如 key 对应的字段为空，操作如下：

案例实操

（1）配置历史服务器

配置 mapred-site.xml

```
<property>
<name>mapreduce.jobhistory.address</name>
<value>hadoop102:10020</value>
</property>
<property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>hadoop102:19888</value>
</property>
```

启动历史服务器

```
sbin/mr-jobhistory-daemon.sh start historyserver
```

查看 jobhistory

http://192.168.1.102:19888/jobhistory

（2）创建原始数据表、空 id 表、合并后数据表

```
// 创建原始表
create table ori(id bigint, time bigint, uid string, keyword string, url rank int, click_num int, click_url string) row format delimited fields terminated by '\t';
// 创建空 id 表
create table nullidtable(id bigint, time bigint, uid string, keyword string, url_rank int, click_num int, click_url string) row format delimited fields terminated by '\t';
// 创建 join 后表的语句
create table jointable(id bigint, time bigint, uid string, keyword string, url rank int, click num int, click url string) row format delimited fields terminated by '\t';
```

（3）分别加载原始数据和空 id 数据到对应表中

```
hive (default)> load data local inpath '/opt/module/datas/ori' into table ori;
hive (default)> load data local inpath '/opt/module/datas/nullid' into table nullidtable;
```

（4）测试不过滤空 id

```
hive (default)> insert overwrite table jointable
select n.* from nullidtable n left join ori o on n.id = o.id;
```

Time taken: 42.038 seconds

Time taken: 37.284 seconds

（5）测试过滤空 id

```
hive (default)> insert overwrite table jointable
select n.* from (select * from nullidtable where id is not null ) n  left join ori o on n.id = o.id;
```

Time taken: 31.725 seconds

Time taken: 28.876 seconds

2. 空 key 转换

有时虽然某个 key 为空对应的数据很多，但是相应的数据不是异常数据，必须要包含在 join 的结果中，此时我们可以表 a 中 key 为空的字段赋一个随机的值，使得数据随机均匀地分不到不同的 reducer 上。例如：

案例实操：

不随机分布空 null 值：

（1）设置 5 个 reduce 个数

set mapreduce.job.reduces = 5;

（2）JOIN 两张表

```
insert overwrite table jointable
select n.* from nullidtable n left join ori b on n.id = b.id;
```

结果：如图 6-13 所示，可以看出来，出现了数据倾斜，某些 reducer 的资源消耗远大于其他 reducer。

Show 20 entries

Task					Succes		
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time
task 1506334829052 0015 r 000000	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:24 +0800 2017	18sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	Mon Sep 25 19:18:18 +0800 2017
task 1506334829052 0015 r 000001	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	12sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:13 +0800 2017	Mon Sep 25 19:18:13 +0800 2017
task 1506334829052 0015 r 000004	SUCCEEDED	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	11sec	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:14 +0800 2017	Mon Sep 25 19:18:14 +0800 2017
task 1506334829052 0015 r 000003	SUCCEEDED	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:17 +0800 2017	10sec	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:14 +0800 2017	Mon Sep 25 19:18:14 +0800 2017
task 1506334829052 0015 r 000002	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:16 +0800 2017	10sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:13 +0800 2017	Mon Sep 25 19:18:13 +0800 2017

图 6-13 空 key 转换

随机分布空 null 值

（1）设置 5 个 reduce 个数

set mapreduce.job.reduces = 5;

（2）JOIN 两张表

```
insert overwrite table jointable
select n.* from nullidtable n full join ori o on
case when n.id is null then concat('hive', rand()) else n.id end = o.id;
```

结果：如图 6-14 所示，可以看出来，消除了数据倾斜，负载均衡 reducer 的资源消耗

Task					Successful			
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time	F
task 1506334829052 0016 r 000000	SUCCEEDED	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:21:04 +0800 2017	20sec	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:56 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000001	SUCCEEDED	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:21:00 +0800 2017	16sec	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000003	SUCCEEDED	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:21:00 +0800 2017	15sec	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000002	SUCCEEDED	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:59 +0800 2017	15sec	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000004	SUCCEEDED	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:20:59 +0800 2017	14sec	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:20:54 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
ID	State	Start Time	Finish Time	Elapsed	Start Time	Shuffle Time	Merge Time	Fin

图 6-14 随机分布空值

17.3.3 MapJoin

如果不指定 MapJoin 或者不符合 MapJoin 的条件，那么 Hive 解析器会将 Join 操作转换成 Common Join，即：在 Reduce 阶段完成 join。容易发生数据倾斜。可以用 MapJoin 把小表全部加载到内存在 map 端进行 join，避免 reducer 处理。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

1. 开启 MapJoin 参数设置

（1）设置自动选择 Mapjoin

```
set hive.auto.convert.join = true; 默认为 true
```

（2）大表小表的阈值设置（默认 25M 一下认为是小表）：

```
set hive.mapjoin.smalltable.filesize=25000000;
```

2. MapJoin 工作机制，如图 6-15 所示

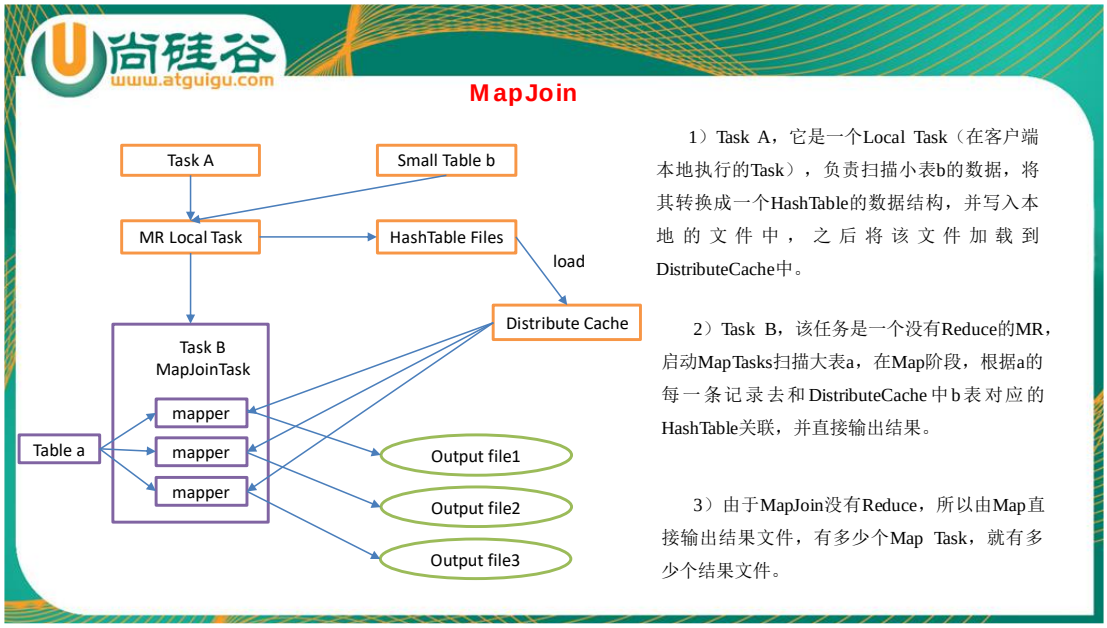


图 6-15 MapJoin 工作机制

案例实操：

（1）开启 Mapjoin 功能

```
set hive.auto.convert.join = true; 默认为 true
```

（2）执行小表 JOIN 大表语句

```
insert overwrite table jointable
select b.id, b.time, b.uid, b.keyword, b.url rank, b.click num, b.click url
from smalltable s
join bigtable  b
on s.id = b.id;
```

Time taken: 24.594 seconds

（3）执行大表 JOIN 小表语句

```
insert overwrite table jointable
select b.id, b.time, b.uid, b.keyword, b.url rank, b.click num, b.click url
from bigtable  b
join smalltable  s
on s.id = b.id;
```

Time taken: 24.315 seconds

17.3.4 Group By

默认情况下，Map 阶段同一 Key 数据分发给一个 reduce，当一个 key 数据过大时就倾斜了。

并不是所有的聚合操作都需要在 Reduce 端完成，很多聚合操作都可以先在 Map 端进行部分聚合，最后在 Reduce 端得出最终结果。

1. 开启 Map 端聚合参数设置

（1）是否在 Map 端进行聚合，默认为 True

```
hive.map.aggr = true
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

（2）在 Map 端进行聚合操作的条目数目

```
hive.groupby.mapaggr.checkinterval = 100000
```

（3）有数据倾斜的时候进行负载均衡（默认是 false）

```
hive.groupby.skewindata = true
```

当选项设定为 true，生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

17.3.5 Count(Distinct) 去重统计

数据量小的时候无所谓，数据量大的情况下，由于 COUNT DISTINCT 操作需要用一个 Reduce Task 来完成，这一个 Reduce 需要处理的数据量太大，就会导致整个 Job 很难完成，一般 COUNT DISTINCT 使用先 GROUP BY 再 COUNT 的方式替换：

案例实操

1. 创建一张大表

```
hive (default)> create table bigtable(id bigint, time bigint, uid string, keyword
string, url rank int, click num int, click url string) row format delimited
fields terminated by '\t';
```

2. 加载数据

```
hive (default)> load data local inpath '/opt/module/datas/bigtable' into table
 bigtable;
```

3. 设置 5 个 reduce 个数

```
set mapreduce.job.reduces = 5;
```

4. 执行去重 id 查询

```
hive (default)> select count(distinct id) from bigtable;
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.12 sec   HDFS Read: 120741990
HDFS Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 120 msec
OK
c0
100001
Time taken: 23.607 seconds, Fetched: 1 row(s)
```

5. 采用 GROUP by 去重 id

```
hive (default)> select count(id) from (select id from bigtable group by id) a;
Stage-Stage-1: Map: 1  Reduce: 5   Cumulative CPU: 17.53 sec   HDFS Read: 120752703
HDFS Write: 580 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.29 sec   HDFS Read: 9409 HDFS
Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 21 seconds 820 msec
OK
 c0
100001
Time taken: 50.795 seconds, Fetched: 1 row(s)
```

虽然会多用一个 Job 来完成，但在数据量大的情况下，这个绝对是值得的。

17.3.6 笛卡尔积

尽量避免笛卡尔积，join 的时候不加 on 条件，或者无效的 on 条件，Hive 只能使用 1 个 reducer 来完成笛卡尔积。

17.3.7 行列过滤

列处理：在 SELECT 中，只拿需要的列，如果有，尽量使用分区过滤，少用 SELECT *。

行处理：在分区剪裁中，当使用外关联时，如果将副表的过滤条件写在 Where 后面，那么就会先全表关联，之后再过滤，比如：

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

案例实操：

1．测试先关联两张表，再用 where 条件过滤

```
hive (default)> select o.id from bigtable b
join ori o on o.id = b.id
where o.id <= 10;
```

Time taken: 34.406 seconds, Fetched: 100 row(s)

2．通过子查询后，再关联表

```
hive (default)> select b.id from bigtable b
join (select id from ori where id <= 10 ) o on b.id = o.id;
```

Time taken: 30.058 seconds, Fetched: 100 row(s)

17.3.8 动态分区调整

关系型数据库中，对分区表 Insert 数据时候，数据库自动会根据分区字段的值，将数据插入到相应的分区中，Hive 中也提供了类似的机制，即动态分区(Dynamic Partition)，只不过，使用 Hive 的动态分区，需要进行相应的配置。

1．开启动态分区参数设置

（1）开启动态分区功能（默认 true，开启）

hive.exec.dynamic.partition=true

（2）设置为非严格模式（动态分区的模式，默认 strict，表示必须指定至少一个分区为静态分区，nonstrict 模式表示允许所有的分区字段都可以使用动态分区。）

```
hive.exec.dynamic.partition.mode=nonstrict
```

（3）在所有执行 MR 的节点上，最大一共可以创建多少个动态分区。

```
hive.exec.max.dynamic.partitions=1000
```

（4）在每个执行 MR 的节点上，最大可以创建多少个动态分区。该参数需要根据实际的数据来设定。比如：源数据中包含了一年的数据，即 day 字段有 365 个值，那么该参数就需要设置成大于 365，如果使用默认值 100，则会报错。

```
hive.exec.max.dynamic.partitions.pernode=100
```

（5）整个 MR Job 中，最大可以创建多少个 HDFS 文件。

```
hive.exec.max.created.files=100000
```

（6）当有空分区生成时，是否抛出异常。一般不需要设置。

```
hive.error.on.empty.partition=false
```

2．案例实操

需求：将 ori 中的数据按照时间(如：20111230000008)，插入到目标表 ori_partitioned_target 的相应分区中。

（1）创建分区表

```
create table ori_partitioned(id bigint, time bigint, uid string, keyword string,
 url_rank int, click_num int, click_url string)
partitioned by (p_time bigint)
row format delimited fields terminated by '\t';
```

（2）加载数据到分区表中

```
hive (default)> load data local inpath '/home/atguigu/ds1' into table
 ori_partitioned partition(p_time='20111230000010') ;
hive (default)> load data local inpath '/home/atguigu/ds2' into table
ori_partitioned partition(p_time='20111230000011') ;
```

（3）创建目标分区表

```
create table ori_partitioned_target(id bigint, time bigint, uid string,
 keyword string, url_rank int, click_num int, click_url string)
PARTITIONED BY (p_time STRING) row format delimited fields terminated by '\t';
```

（4）设置动态分区

```
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode = nonstrict;
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
        set hive.exec.max.dynamic.partitions = 1000;
        set hive.exec.max.dynamic.partitions.pernode = 100;
        set hive.exec.max.created.files = 100000;
        set hive.error.on.empty.partition = false;

        hive (default)> insert overwrite table ori partitioned target partition
(p time)
        select id, time, uid, keyword, url rank, click num, click url, p time
from ori_partitioned;
```

（5）查看目标分区表的分区情况

```
        hive (default)> show partitions ori_partitioned_target;
```

17.3.9 分桶

1）分桶表数据存储

分区针对的是数据的存储路径；分桶针对的是数据文件。

分区提供一个隔离数据和优化查询的便利方式。不过，并非所有的数据集都可形成合理的分区，特别是之前所提到过的要确定合适的划分大小这个疑虑。

分桶是将数据集分解成更容易管理的若干部分的另一个技术。

1．先创建分桶表，通过直接导入数据文件的方式

（1）数据准备

student.txt

（2）创建分桶表

```
create table stu buck(id int, name string)
clustered by(id)
into 4 buckets
row format delimited fields terminated by '\t';
```

（3）查看表结构

```
hive (default)> desc formatted stu_buck;
Num Buckets:            4
```

（4）导入数据到分桶表中

```
hive (default)> load data local inpath '/opt/module/datas/student.txt' into table
 stu_buck;
```

（5）查看创建的分桶表中是否分成 4 个桶，如图 6-7 所示

Browse Directory

/user/hive/warehouse/stu_buck Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	152 B	2017/9/3 下午4:41:49	3	128 MB	student.txt

图 6-7 未分桶

发现并没有分成 4 个桶。是什么原因呢？

2．创建分桶表时，数据通过子查询的方式导入

（1）先建一个普通的 stu 表

```
create table stu(id int, name string)
row format delimited fields terminated by '\t';
```

（2）向普通的 stu 表中导入数据

```
 load data local inpath '/opt/module/datas/student.txt' into table stu;
```

（3）清空 stu_buck 表中数据

```
truncate table stu_buck;
select * from stu_buck;
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

（4）导入数据到分桶表，通过子查询的方式

```
insert into table stu_buck
select id, name from stu;
```

（5）发现还是只有一个分桶，如图 6-8 所示

/user/hive/warehouse/stu_buck Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午4:55:11	3	128 MB	000000_0

图 6-8 未分桶

（6）需要设置一个属性

```
hive (default)> set hive.enforce.bucketing=true;
hive (default)> set mapreduce.job.reduces=-1;
hive (default)> insert into table stu_buck
select id, name from stu;
```

Browse Directory

/user/hive/warehouse/stu_buck Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:53	3	128 MB	000000_0
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:52	3	128 MB	000001_0
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:53	3	128 MB	000002_0
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:54	3	128 MB	000003_0

图 6-9 分桶

（7）查询分桶的数据

```
hive (default)> select * from stu_buck;
OK
stu_buck.id     stu_buck.name
1004    ss4
1008    ss8
1012    ss12
1016    ss16
1001    ss1
1005    ss5
1009    ss9
1013    ss13
1002    ss2
1006    ss6
1010    ss10
1014    ss14
1003    ss3
1007    ss7
1011    ss11
1015    ss15
```

2）分桶抽样查询

对于非常大的数据集，有时用户需要使用的是一个具有代表性的查询结果而不是全部结果。Hive 可以通过对表进行抽样来满足这个需求。

查询表 stu_buck 中的数据。

```
hive (default)> select * from stu_buck tablesample(bucket 1 out of 4 on id);
```

注：tablesample 是抽样语句，语法：TABLESAMPLE(BUCKET x OUT OF y) 。

y 必须是 table 总 bucket 数的倍数或者因子。hive 根据 y 的大小，决定抽样的比例。例如，table 总共分了 4 份，当 y=2 时，抽取(4/2=)2 个 bucket 的数据，当 y=8 时，抽取(4/8=)1/2 个 bucket 的数据。

x 表示从哪个 bucket 开始抽取，如果需要取多个分区，以后的分区号为当前分区号加上

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

y。例如，table 总 bucket 数为 4，tablesample(bucket 1 out of 2)，表示总共抽取（4/2=）2 个 bucket 的数据，抽取第 1(x)个和第 4(x+y)个 bucket 的数据。

注意：x 的值必须小于等于 y 的值，否则

FAILED: SemanticException [Error 10061]: Numerator should not be bigger than denominator in sample clause for table stu_buck

17.3.10 分区

分区表实际上就是对应一个 HDFS 文件系统上的独立的文件夹，该文件夹下是该分区所有的数据文件。Hive 中的分区就是分目录，把一个大的数据集根据业务需要分割成小的数据集。在查询时通过 WHERE 子句中的表达式选择查询所需要的指定的分区，这样的查询效率会提高很多。

1）分区表基本操作

1. 引入分区表（需要根据日期对日志进行管理）

```
/user/hive/warehouse/log partition/20170702/20170702.log
/user/hive/warehouse/log partition/20170703/20170703.log
/user/hive/warehouse/log_partition/20170704/20170704.log
```

2. 创建分区表语法

```
hive (default)> create table dept_partition(
            deptno int, dname string, loc string
            )
            partitioned by (month string)
            row format delimited fields terminated by '\t';
```

3. 加载数据到分区表中

```
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table
default.dept partition partition(month='201709');
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table
default.dept_partition partition(month='201708');
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table
default.dept_partition partition(month='201707');
```

/user/hive/warehouse/dept_partition/month=201709 Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	82 B	2017/8/30 下午4:56:54	3	128 MB	dept.txt

图 6-5 加载数据到分区表

/user/hive/warehouse/dept_partition/ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxrwxr-x	atguigu	supergroup	0 B	2017/8/30 下午5:01:52	0	0 B	month=201707
drwxrwxr-x	atguigu	supergroup	0 B	2017/8/30 下午5:01:18	0	0 B	month=201708
drwxrwxr-x	atguigu	supergroup	0 B	2017/8/30 下午4:56:54	0	0 B	month=201709

图 6-6 分区表

4. 查询分区表中数据

单分区查询

```
hive (default)> select * from dept_partition where month='201709';
```

多分区联合查询

```
hive (default)> select * from dept partition where month='201709'
            union
            select * from dept_partition where month='201708'
            union
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
            select * from dept_partition where month='201707';

 u3.deptno      u3.dname       u3.loc  u3.month
10      ACCOUNTING      NEW YORK       201707
10      ACCOUNTING      NEW YORK       201708
10      ACCOUNTING      NEW YORK       201709
20      RESEARCH       DALLAS  201707
20      RESEARCH       DALLAS  201708
20      RESEARCH       DALLAS  201709
30      SALES   CHICAGO 201707
30      SALES   CHICAGO 201708
30      SALES   CHICAGO 201709
40      OPERATIONS      BOSTON  201707
40      OPERATIONS      BOSTON  201708
40      OPERATIONS      BOSTON  201709
```

5. 增加分区

创建单个分区

```
hive (default)> alter table dept_partition add partition(month='201706') ;
```

同时创建多个分区

```
hive (default)> alter table dept_partition add partition(month='201705')
partition(month='201704');
```

6. 删除分区

删除单个分区

```
hive (default)> alter table dept_partition drop partition (month='201704');
```

同时删除多个分区

```
hive (default)> alter table dept partition drop partition (month='201705'),
partition (month='201706');
```

7. 查看分区表有多少分区

```
hive> show partitions dept_partition;
```

8. 查看分区表结构

```
  hive> desc formatted dept_partition;

# Partition Information
# col_name            data_type             comment
month                 string
```

2）分区表注意事项

1. 创建二级分区表

```
hive (default)> create table dept_partition2(
             deptno int, dname string, loc string
             )
             partitioned by (month string, day string)
             row format delimited fields terminated by '\t';
```

2. 正常的加载数据

（1）加载数据到二级分区表中

```
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table
 default.dept_partition2 partition(month='201709', day='13');
```

（2）查询分区数据

```
hive (default)> select * from dept_partition2 where month='201709' and day='13';
```

3. 把数据直接上传到分区目录上，让分区表和数据产生关联的三种方式

（1）方式一：上传数据后修复

上传数据

```
hive (default)> dfs -mkdir -p
 /user/hive/warehouse/dept_partition2/month=201709/day=12;
hive (default)> dfs -put /opt/module/datas/dept.txt
/user/hive/warehouse/dept_partition2/month=201709/day=12;
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

查询数据（查询不到刚上传的数据）

```
hive (default)> select * from dept_partition2 where month='201709' and day='12';
```

执行修复命令

```
hive> msck repair table dept_partition2;
```

再次查询数据

```
hive (default)> select * from dept_partition2 where month='201709' and day='12';
```

（2）方式二：上传数据后添加分区

上传数据

```
hive (default)> dfs -mkdir -p
 /user/hive/warehouse/dept_partition2/month=201709/day=11;
hive (default)> dfs -put /opt/module/datas/dept.txt
/user/hive/warehouse/dept_partition2/month=201709/day=11;
```

执行添加分区

```
hive (default)> alter table dept partition2 add partition(month='201709',
 day='11');
```

查询数据

```
hive (default)> select * from dept_partition2 where month='201709' and day='11';
```

（3）方式三：上传数据后 load 数据到分区

创建目录

```
hive (default)> dfs -mkdir -p
 /user/hive/warehouse/dept_partition2/month=201709/day=10;
```

上传数据

```
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table
 dept_partition2 partition(month='201709',day='10');
```

查询数据

```
hive (default)> select * from dept_partition2 where month='201709' and day='10';
```

17.4 数据倾斜

17.4.1 合理设置 Map 数

1）通常情况下，作业会通过 input 的目录产生一个或者多个 map 任务。

主要的决定因素有：input 的文件总个数，input 的文件大小，集群设置的文件块大小。

2）是不是 map 数越多越好？

答案是否定的。如果一个任务有很多小文件（远远小于块大小 128m），则每个小文件也会被当做一个块，用一个 map 任务来完成，而一个 map 任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的 map 数是受限的。

3）是不是保证每个 map 处理接近 128m 的文件块，就高枕无忧了？

答案也是不一定。比如有一个 127m 的文件，正常会用一个 map 去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，如果 map 处理的逻辑比较复杂，用一个 map 任务去做，肯定也比较耗时。

针对上面的问题 2 和 3，我们需要采取两种方式来解决：即减少 map 数和增加 map 数；

17.4.2 小文件进行合并

在 map 执行前合并小文件，减少 map 数：CombineHiveInputFormat 具有对小文件进行合并的功能（系统默认的格式）。HiveInputFormat 没有对小文件合并功能。

```
set hive.input.format= org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

17.4.3 复杂文件增加 Map 数

当 input 的文件都很大，任务逻辑复杂，map 执行非常慢的时候，可以考虑增加 Map 数，来使得每个 map 处理的数据量减少，从而提高任务的执行效率。

增加 map 的方法为：根据 computeSliteSize(Math.max(minSize,Math.min(maxSize,blocksize)))=blocksize=128M 公式，调整 maxSize 最大值。让 maxSize 最大值低于 blocksize 就可以增加 map 的个数。

案例实操：

1. 执行查询

```
hive (default)> select count(*) from emp;
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
```

2. 设置最大切片值为 100 个字节

```
hive (default)> set mapreduce.input.fileinputformat.split.maxsize=100;
hive (default)> select count(*) from emp;
Hadoop job information for Stage-1: number of mappers: 6; number of reducers: 1
```

17.4.4 合理设置 Reduce 数

1. 调整 reduce 个数方法一

（1）每个 Reduce 处理的数据量默认是 256MB

```
hive.exec.reducers.bytes.per.reducer=256000000
```

（2）每个任务最大的 reduce 数，默认为 1009

```
hive.exec.reducers.max=1009
```

（3）计算 reducer 数的公式

```
N=min(参数 2，总输入数据量/参数 1)
```

2. 调整 reduce 个数方法二

在 hadoop 的 mapred-default.xml 文件中修改

设置每个 job 的 Reduce 个数

```
set mapreduce.job.reduces = 15;
```

3. reduce 个数并不是越多越好

1）过多的启动和初始化 reduce 也会消耗时间和资源；

2）另外，有多少个 reduce，就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；

在设置 reduce 个数的时候也需要考虑这两个原则：处理大数据量利用合适的 reduce 数；使单个 reduce 任务处理数据量大小要合适；

17.5 并行执行

Hive 会将一个查询转化成一个或者多个阶段。这样的阶段可以是 MapReduce 阶段、抽样阶段、合并阶段、limit 阶段。或者 Hive 执行过程中可能需要的其他阶段。默认情况下，Hive 一次只会执行一个阶段。不过，某个特定的 job 可能包含众多的阶段，而这些阶段可能并非完全互相依赖的，也就是说有些阶段是可以并行执行的，这样可能使得整个 job 的执行时间缩短。不过，如果有更多的阶段可以并行执行，那么 job 可能就越快完成。

通过设置参数 hive.exec.parallel 值为 true，就可以开启并发执行。不过，在共享集群中，需要注意下，如果 job 中并行阶段增多，那么集群利用率就会增加。

```
set hive.exec.parallel=true;              //打开任务并行执行
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
set hive.exec.parallel.thread.number=16;  //同一个 sql 允许最大并行度，默认为 8。
```

当然，得是在系统资源比较空闲的时候才有优势，否则，没资源，并行也起不来。

17.6 严格模式

Hive 提供了一个严格模式，可以防止用户执行那些可能意向不到的不好的影响的查询。

通过设置属性 hive.mapred.mode 值为默认是非严格模式 nonstrict 。开启严格模式需要修改 hive.mapred.mode 值为 strict，开启严格模式可以禁止 3 种类型的查询。

```
<property>
   <name>hive.mapred.mode</name>
   <value>strict</value>
   <description>
     The mode in which the Hive operations are being performed.
     In strict mode, some risky queries are not allowed to run. They include:
       Cartesian Product.
       No partition being picked up for a query.
       Comparing bigints and strings.
       Comparing bigints and doubles.
       Orderby without limit.
   </description>
   </property>
```

1) 对于分区表，除非 where 语句中含有分区字段过滤条件来限制范围，否则不允许执行。换句话说，就是用户不允许扫描所有分区。进行这个限制的原因是，通常分区表都拥有非常大的数据集，而且数据增加迅速。没有进行分区限制的查询可能会消耗令人不可接受的巨大资源来处理这个表。
2) 对于使用了 order by 语句的查询，要求必须使用 limit 语句。因为 order by 为了执行排序过程会将所有的结果数据分发到同一个 Reducer 中进行处理，强制要求用户增加这个 LIMIT 语句可以防止 Reducer 额外执行很长一段时间。
3) 限制笛卡尔积的查询。对关系型数据库非常了解的用户可能期望在执行 JOIN 查询的时候不使用 ON 语句而是使用 where 语句，这样关系数据库的执行优化器就可以高效地将 WHERE 语句转化成那个 ON 语句。不幸的是，Hive 并不会执行这种优化，因此，如果表足够大，那么这个查询就会出现不可控的情况。

17.7 JVM 重用

JVM 重用是 Hadoop 调优参数的内容，其对 Hive 的性能具有非常大的影响，特别是对于很难避免小文件的场景或 task 特别多的场景，这类场景大多数执行时间都很短。

Hadoop 的默认配置通常是使用派生 JVM 来执行 map 和 Reduce 任务的。这时 JVM 的启动过程可能会造成相当大的开销，尤其是执行的 job 包含有成百上千 task 任务的情况。JVM 重用可以使得 JVM 实例在同一个 job 中重新使用 N 次。N 的值可以在 Hadoop 的 mapred-site.xml 文件中进行配置。通常在 10-20 之间，具体多少需要根据具体业务场景测试得出。

```
<property>
  <name>mapreduce.job.jvm.numtasks</name>
  <value>10</value>
  <description>How many tasks to run per jvm. If set to -1, there is
  no limit.
  </description>
</property>
```

这个功能的缺点是，开启 JVM 重用将一直占用使用到的 task 插槽，以便进行重用，直

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

到任务完成后才能释放。如果某个“不平衡的”job 中有某几个 reduce task 执行的时间要比其他 Reduce task 消耗的时间多的多的话，那么保留的插槽就会一直空闲着却无法被其他的 job 使用，直到所有的 task 都结束了才会释放。

17.8 推测执行

在分布式集群环境下，因为程序 Bug（包括 Hadoop 本身的 bug），负载不均衡或者资源分布不均等原因，会造成同一个作业的多个任务之间运行速度不一致，有些任务的运行速度可能明显慢于其他任务（比如一个作业的某个任务进度只有 50%，而其他所有任务已经运行完毕），则这些任务会拖慢作业的整体执行进度。为了避免这种情况发生，Hadoop 采用了推测执行（Speculative Execution）机制，它根据一定的法则推测出“拖后腿”的任务，并为这样的任务启动一个备份任务，让该任务与原始任务同时处理同一份数据，并最终选用最先成功运行完成任务的计算结果作为最终结果。

设置开启推测执行参数：Hadoop 的 mapred-site.xml 文件中进行配置

```
<property>
  <name>mapreduce.map.speculative</name>
  <value>true</value>
  <description>If true, then multiple instances of some map tasks
             may be executed in parallel.</description>
</property>

<property>
  <name>mapreduce.reduce.speculative</name>
  <value>true</value>
  <description>If true, then multiple instances of some reduce tasks
             may be executed in parallel.</description>
</property>
```

不过 hive 本身也提供了配置项来控制 reduce-side 的推测执行：

```
  <property>
    <name>hive.mapred.reduce.tasks.speculative.execution</name>
    <value>true</value>
    <description>Whether speculative execution for reducers should be turned on.
</description>
  </property>
```

关于调优这些推测执行变量，还很难给一个具体的建议。如果用户对于运行时的偏差非常敏感的话，那么可以将这些功能关闭掉。如果用户因为输入数据量很大而需要执行长时间的 map 或者 Reduce task 的话，那么启动推测执行造成的浪费是非常巨大大。

17.9 压缩

17.9.1 Hadoop 源码编译支持 Snappy 压缩

1）资源准备

1. CentOS 联网

配置 CentOS 能连接外网。Linux 虚拟机 ping www.baidu.com 是畅通的

注意：采用 root 角色编译，减少文件夹权限出现问题

2. jar 包准备(hadoop 源码、JDK8 、maven、protobuf)

（1）hadoop-2.7.2-src.tar.gz

（2）jdk-8u144-linux-x64.tar.gz

（3）snappy-1.1.3.tar.gz

（4）apache-maven-3.0.5-bin.tar.gz

（5）protobuf-2.5.0.tar.gz

17.9.2 jar 包安装

注意：所有操作必须在 root 用户下完成

1. JDK 解压、配置环境变量 JAVA_HOME 和 PATH，验证 java-version(如下都需要验证是否配置成功)

```
[root@hadoop101 software] # tar -zxf jdk-8u144-linux-x64.tar.gz -C /opt/module/
[root@hadoop101 software]# vi /etc/profile
#JAVA_HOME
export JAVA_HOME=/opt/module/jdk1.8.0_144
export PATH=$PATH:$JAVA_HOME/bin
[root@hadoop101 software]#source /etc/profile
```

验证命令：java -version

2. Maven 解压、配置 MAVEN_HOME 和 PATH

```
[root@hadoop101 software]# tar -zxvf apache-maven-3.0.5-bin.tar.gz -C /opt/module/
[root@hadoop101 apache-maven-3.0.5]# vi /etc/profile
#MAVEN_HOME
export MAVEN_HOME=/opt/module/apache-maven-3.0.5
export PATH=$PATH:$MAVEN_HOME/bin
[root@hadoop101 software]#source /etc/profile
```

验证命令：mvn -version

17.9.3 编译源码

1. 准备编译环境

```
[root@hadoop101 software]# yum install svn
[root@hadoop101 software]# yum install autoconf automake libtool cmake
[root@hadoop101 software]# yum install ncurses-devel
[root@hadoop101 software]# yum install openssl-devel
[root@hadoop101 software]# yum install gcc*
```

2. 编译安装 snappy

```
[root@hadoop101 software]# tar -zxvf snappy-1.1.3.tar.gz -C /opt/module/
[root@hadoop101 module]# cd snappy-1.1.3/
[root@hadoop101 snappy-1.1.3]# ./configure
[root@hadoop101 snappy-1.1.3]# make
[root@hadoop101 snappy-1.1.3]# make install
# 查看 snappy 库文件
[root@hadoop101 snappy-1.1.3]# ls -lh /usr/local/lib |grep snappy
```

3. 编译安装 protobuf

```
[root@hadoop101 software]# tar -zxvf protobuf-2.5.0.tar.gz -C /opt/module/
[root@hadoop101 module]# cd protobuf-2.5.0/
[root@hadoop101 protobuf-2.5.0]# ./configure
[root@hadoop101 protobuf-2.5.0]#  make
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
[root@hadoop101 protobuf-2.5.0]#  make install
# 查看 protobuf 版本以测试是否安装成功
[root@hadoop101 protobuf-2.5.0]# protoc --version
```

4. 编译 hadoop native

```
[root@hadoop101 software]# tar -zxvf hadoop-2.7.2-src.tar.gz
[root@hadoop101 software]# cd hadoop-2.7.2-src/
[root@hadoop101 software]# mvn clean package -DskipTests -Pdist,native -Dtar -
Dsnappy.lib=/usr/local/lib -Dbundle.snappy
```

执行成功后，/opt/software/hadoop-2.7.2-src/hadoop-dist/target/hadoop-2.7.2.tar.gz 即为新生成的支持 snappy 压缩的二进制安装包。

17.9.4 Hadoop 压缩配置

1）MR 支持的压缩编码

表 6-8

压缩格式	工具	算法	文件扩展名	是否可切分
DEFAULT	无	DEFAULT	.deflate	否
Gzip	gzip	DEFAULT	.gz	否
bzip2	bzip2	bzip2	.bz2	是
LZO	lzop	LZO	.lzo	是
Snappy	无	Snappy	.snappy	否

为了支持多种压缩/解压缩算法，Hadoop 引入了编码/解码器，如下表所示：

表 6-9

压缩格式	对应的编码/解码器
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
gzip	org.apache.hadoop.io.compress.GzipCodec
bzip2	org.apache.hadoop.io.compress.BZip2Codec
LZO	com.hadoop.compression.lzo.LzopCodec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

压缩性能的比较：

表 6-10

压缩算法	原始文件大小	压缩文件大小	压缩速度	解压速度
gzip	8.3GB	1.8GB	17.5MB/s	58MB/s
bzip2	8.3GB	1.1GB	2.4MB/s	9.5MB/s
LZO	8.3GB	2.9GB	49.3MB/s	74.6MB/s

http://google.github.io/snappy/

On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more.

2） 压缩参数配置

要在 Hadoop 中启用压缩，可以配置如下参数（mapred-site.xml 文件中）：

表 6-11

参数	默认值	阶段	建议
io.compression.codecs	org.apache.hadoop.io.compress.DefaultCodec,	输入压缩	Hadoop 使

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

（在 core-site.xml 中配置）	org.apache.hadoop.io.compress.GzipCodec, org.apache.hadoop.io.compress.BZip2Codec, org.apache.hadoop.io.compress.Lz4Codec		用文件扩展名判断是否支持某种编解码器
mapreduce.map.output.compress	false	mapper 输出	这个参数设为 true 启用压缩
mapreduce.map.output.compress.codec	org.apache.hadoop.io.compress.DefaultCodec	mapper 输出	使用 LZO、LZ4 或 snappy 编解码器在此阶段压缩数据
mapreduce.output.fileoutputformat.compress	false	reducer 输出	这个参数设为 true 启用压缩
mapreduce.output.fileoutputformat.compress.codec	org.apache.hadoop.io.compress. DefaultCodec	reducer 输出	使用标准工具或者编解码器，如 gzip 和 bzip2
mapreduce.output.fileoutputformat.compress.type	RECORD	reducer 输出	SequenceFile 输出使用的压缩类型：NONE 和 BLOCK

17.9.5 开启 Map 输出阶段压缩

开启 map 输出阶段压缩可以减少 job 中 map 和 Reduce task 间数据传输量。具体配置如下：

案例实操：

1. 开启 hive 中间传输数据压缩功能

```
hive (default)>set hive.exec.compress.intermediate=true;
```

2. 开启 mapreduce 中 map 输出压缩功能

```
hive (default)>set mapreduce.map.output.compress=true;
```

3. 设置 mapreduce 中 map 输出数据的压缩方式

```
hive (default)>set mapreduce.map.output.compress.codec=
 org.apache.hadoop.io.compress.SnappyCodec;
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

4. 执行查询语句

```
hive (default)> select count(ename) name from emp;
```

17.9.6 开启 Reduce 输出阶段压缩

当 Hive 将输出写入到表中时，输出内容同样可以进行压缩。属性 hive.exec.compress.output 控制着这个功能。用户可能需要保持默认设置文件中的默认值 false，这样默认的输出就是非压缩的纯文本文件了。用户可以通过在查询语句或执行脚本中设置这个值为 true，来开启输出结果压缩功能。

案例实操：

1. 开启 hive 最终输出数据压缩功能

```
hive (default)>set hive.exec.compress.output=true;
```

2. 开启 mapreduce 最终输出数据压缩

```
hive (default)>set mapreduce.output.fileoutputformat.compress=true;
```

3. 设置 mapreduce 最终数据输出压缩方式

```
hive (default)> set mapreduce.output.fileoutputformat.compress.codec =
 org.apache.hadoop.io.compress.SnappyCodec;
```

4. 设置 mapreduce 最终数据输出压缩为块压缩

```
hive (default)> set mapreduce.output.fileoutputformat.compress.type=BLOCK;
```

5. 测试一下输出结果是否是压缩文件

```
hive (default)> insert overwrite local directory
 '/opt/module/datas/distribute-result' select * from emp distribute by deptno sort
by empno desc;
```

17.10 执行计划（Explain）

1. 基本语法

EXPLAIN [EXTENDED | DEPENDENCY | AUTHORIZATION] query

2. 案例实操

（1）查看下面这条语句的执行计划

```
hive (default)> explain select * from emp;
hive (default)> explain select deptno, avg(sal) avg_sal from emp group by deptno;
```

（2）查看详细执行计划

```
hive (default)> explain extended select * from emp;
hive (default)> explain extended select deptno, avg(sal) avg_sal from emp group
by deptno;
```

18.Hive 数据分析面试题

场景举例.北京市学生成绩分析.

成绩的数据格式:时间,学校,年级,姓名,科目,成绩

样例数据如下:

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

2013,北大,1,裘容絮,语文,97
2013,北大,1,庆眠拔,语文,52
2013,北大,1,乌洒筹,语文,85
2012,清华,0,钦尧,英语,61
2015,北理工,3,冼殿,物理,81
2016,北科,4,况飘索,化学,92
2014,北航,2,孔须,数学,70
2012,清华,0,王脊,英语,59
2014,北航,2,方部盾,数学,49
2014,北航,2,东门雹,数学,77
问题:

18.1 情景题：分组 TOPN

```
# 1.分组 TOPN 选出今年每个学校,每个年级,每个科目分数前三.
hive -e "
select t.*
from
(
select
        school,
        class,
        subjects,
        score,
        row_number() over (partition by school,class,subjects order by score desc) rank_code
from spark_test_wx
where partition_id = "2017"
) t
where t.rank_code <= 3;
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

北大	0	英语	95	1
北大	0	英语	77	2
北大	0	英语	50	3
北大	2	数学	80	1
北大	2	数学	62	2
北大	2	数学	56	3
北大	3	物理	82	1
北大	3	物理	61	2
北大	3	物理	57	3
北大	4	化学	72	1
北大	4	化学	56	2
北大	4	化学	52	3
北大	5	生物	84	1
北大	5	生物	62	2
北大	5	生物	57	3
北理工	0	英语	93	1
北理工	0	英语	71	2
北理工	0	英语	59	3
北理工	1	语文	79	1
北理工	1	语文	62	2
北理工	1	语文	49	3
北理工	2	数学	93	1
北理工	2	数学	85	2
北理工	2	数学	73	3
北理工	3	物理	78	1
北理工	3	物理	49	2
北理工	4	化学	97	1
北理工	4	化学	89	2
北理工	4	化学	78	3
北理工	5	生物	86	1
北理工	5	生物	77	2
北理工	5	生物	57	3

18.2 情景题：where 与 having

```
-- 今年 清华 1 年级 总成绩大于 200 分的学生 以及学生数
SELECT
    school,
    class,
    NAME,
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
        sum(score) AS total_score,
        count(1) over () nct
FROM
        spark_test_wx
WHERE
        partition_id = "2017"
AND school = "清华"
AND class = 1
GROUP BY
        school,
        class,
        NAME
HAVING
        total_score > 200;
```

having 是分组（group by）后的筛选条件，分组后的数据组内再筛选，也就是说 HAVING 子句可以让我们筛选成组后的各组数据。

where则是在分组,聚合前先筛选记录。也就是说作用在GROUP BY子句和HAVING子句前。

```
OK
清华    1    枚欧涧    231.0    4
清华    1    禹钥蜗    276.0    4
清华    1    苏俘      219.0    4
清华    1    顾藻娘    285.0    4
```

18.3 情景题：数据倾斜

今年加入进来了 10 个学校,学校数据差异很大计算每个学校的平均分。简述需要注意的点。

该题主要是考察数据倾斜的处理方式。

Group by 方式很容易产生数据倾斜。需要注意一下几点

1）Map 端部分聚合

hive.map.aggr=true（用于设定是否在 map 端进行聚合，默认值为真，相当于 combine）

hive.groupby.mapaggr.checkinterval=100000（用于设定 map 端进行聚合操作的条数）

2）有数据倾斜时进行负载均衡

设定 hive.groupby.skewindata，当选项设定为 true 是，生成的查询计划有两个 MapReduce 任务。

在第一个 MapReduce 中，map 的输出结果集合会随机分布到 reduce 中， 每个 reduce 做部分聚合操作，并输出结果。这样处理的结果是，相同的 Group By Key 有可能分发到不同的 reduce 中，从而达到负载均衡的目的；

第二个 MapReduce 任务再根据预处理的数据结果按照 Group By Key 分布到 reduce 中（这个过程可以保证相同的 Group By Key 分布到同一个 reduce 中），最后完成最终的聚合

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

操作。

18.4 情景题：分区表

假设我创建了一张表，其中包含了 2016 年客户完成的所有交易的详细信息：CREATE TABLE transaction_details (cust_id INT, amount FLOAT, month STRING, country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,’ ;

现在我插入了 100 万条数据，我想知道每个月的总收入。

问：如何高效的统计出结果。写出步骤即可。

建立按月份的分区表，每个月的总收入分别查询

19.Hive 中，建的表为压缩表，但是输入文件为非压缩格式，会产生怎样的现象或者结果？

使用 load 加载数据的时候，不会是压缩文件。

使用查询插入数据的时候，存储的数据是压缩的。

20.已知表 a 是一张内部表，如何将它转换成外部表？请写出相应的 hive 语句。

alter table a set tblproperties('EXTERNAL'='TRUE');

21.Hive 中 mapjoin 的原理和实际应用？

原理就是在 Map 阶段将小表读入内存，顺序扫描大表完成 Join。

通过 MapReduce Local Task，将小表读入内存，生成 HashTableFiles 上传至 Distributed Cache 中，这里会对 HashTableFiles 进行压缩。

MapReduce Job 在 Map 阶段，每个 Mapper 从 Distributed Cache 读取 HashTableFiles 到内存中，顺序扫描大表，在 Map 阶段直接进行 Join，将数据传递给下一个 MapReduce 任务。

作用：优化 hive 语句，避免出现在 Reducer 端的数据倾斜。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

22.订单详情表 ord_det(order_id 订单号，sku_id 商品编号，sale_qtty 销售数量，dt 日期分区)任务计算 2016 年 1 月 1 日商品销量的 Top100，并按销量降级排序

```
SELECT
    sku_id,
    sum(sale_qtty) sale_sum
FROM
    ord_det PARTITION (dt = '20160101')
ORDER BY
    sale_sum DESC
LIMIT 100;
```

23.某日志的格式如下：

pin|-|request_tm|-url|-|sku_id|-|amount
分隔符为‘|-|’，
数据样例为:
张三|-|q2013-11-23 11:59:30|-|www.jd.com|-|100023|-|110.15
假设本地数据文件为 sample.txt,先将其导入到 hive 的 test 库的表 t_sample 中，并计算每个用户的总消费金额，写出详细过程包括表结构。

1.https://www.cnblogs.com/cxchanpin/p/6911286.html
org.apache.hadoop.hive.serde2.RegexSerDe

```
2. create table t_sample1(pin String,
request_tm String,
url String,
sku_id bigint,
amount double)
ROW FORMAT    SERDE 'org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe'
WITH    SERDEPROPERTIES ("field.delim"="|-|")
stored as    textfile ;
```

```
hive-site.xml
<property>
   <name>hive.aux.jars.path</name>
   <value>file:////opt/module/hive-1.2.1/lib/hive-contrib-1.2.1.jar</value>
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

```
    <description>Added by tiger.zeng on 20120202.These JAR file are available to all users for all jobs</description>
  </property>
```

3.先清洗数据，改变其分割符变为‘\t’，存入本地文件 jd.txt 中(注意 split()方法也是需要转义的)

```
CREATE external TABLE t_sample (
    pin string,
    request_tm string,
    url string,
    sku_id INT,
    amount DOUBLE
) ROW format delimited FIELDS TERMINATED BY "\t";

LOAD DATA LOCAL inpath '/opt/module/datas/jd.txt' INTO TABLE t_sample;

SELECT
    *
FROM
    t_sample;

SELECT
    pin,
    sum(amount)
FROM
    t_sample
GROUP BY
    pin;
```

24.有一张很大的表：TRLOG，该表大概有 2T 左右

```
CREATE TABLE TRLOG
(   PLATFORM string,
    USER_ID int,
    CLICK_TIME string,
    CLICK_URL string)
    row format delimited fields terminated by ‘\t’;
```

数据:

PLATFORM	USER_ID	CLICK_TIME	CLICK_URL
WEB	12332321	2013-03-21 13:48:31.324	/home/
WEB	12332321	2013-03-21 13:48:32.954	/selectcat/er/

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

WEB 12332321 2013-03-21 13:48:46.365 /er/viewad/12.html
WEB 12332321 2013-03-21 13:48:53.651 /er/viewad/13.html
…… …… …… ……

把上述数据处理为如下结构的表 ALLOG:

```
CREATE TABLE ALLOG
    (PLATFORM string,
    USER_ID int,
    SEQ int,
    FROM_URL string,
    TO_URL string)
    row format delimited fields terminated by '\t';
```

整理后的数据结构：

PLATFORM	USER_ID	SEQ	FROM_URL	TO_URL
WEB	12332321	1	NULL	/home/
WEB	12332321	2	/home/	/selectcat/er/
WEB	12332321	3	/selectcat/er/	/er/viewad/12.html
WEB	12332321	4	/er/viewad/12.html	/er/viewad/13.html
WEB	12332321	1	NULL	/m/home/
WEB	12332321	2	/m/home/	/m/selectcat/fang/

PLATFORM 和 USER_ID 还是代表平台和用户 ID:SEQ 字段代表用户按时间排序后的访问顺序，FROM_URL 和 TO_URL 分别代表用户从哪一页跳转到哪一页。某个用户的第一条访问记录的 FROM_URL 是 NULL（空值）。实现基于纯 Hive SQL 的 ETL 过程，从 TRLOG 表生成 ALLOG 表:（结果是一套 SQL）

```
INSERT INTO TABLE allog SELECT
    platform,
    user_id,
    row_number () over (
        PARTITION BY user_id
        ORDER BY
            click_time
    ) seq,
    lag (click_url, 1) over (
        PARTITION BY user_id
        ORDER BY
            click_time
    ) AS from_url,
    click_url AS to_url
FROM
    trlog;
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

25. 已知一个表 STG.ORDER，有如下字段:Date，Order_id，User_id，amount。请给出 sql 进行统计:数据样例:2017-01-01,10029028,1000003251,33.57。

1) 给出 2017 年每个月的订单数、用户数、总成交金额。

```
SELECT
    count(Order_id) order_count,
    count(DISTINCT(User_id)) user_count,
    sum(amount) amount_sum,
    substring(Date, 1, 7) MONTH
FROM
    STG. ORDER
WHERE
    substring(Date, 1, 4) = ’2017’
GROUP BY
    MONTH;
```

2) 给出 2017 年 11 月的新客数(指在 11 月才有第一笔订单)。

```
SELECT
    count(1)
FROM
    (
        SELECT
            Order_id,
            DATE,
            LAG (DATE, 1) over(partition by User_id order by Date) firstOrder
        FROM
            STG. ORDER
    ) t1
WHERE
    firstOrder IS NULL
AND SUBSTRING(DATE, 1, 7) = '2017-11';
```

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

Flume

1. flume 有哪些组件，flume 的 source、channel、sink 具体是做什么的

source 组件是专门用来收集数据的，可以处理各种类型、各种格式的日志数据，包括 avro、thrift、exec、jms、spooling directory、netcat、sequence generator、syslog、http、legacy

source 组件把数据收集来以后，临时存放在 channel 中，即 channel 组件在 agent 中是专门用来存放临时数据的——对采集到的数据进行简单的缓存，可以存放在 memory、jdbc、file 等等。

sink 组件是用于把数据发送到目的地的组件，目的地包括 hdfs、logger、avro、thrift、ipc、file、null、Hbase、solr、自定义。

2. 你是如何实现 flume 数据传输的监控的

使用第三方框架 Ganglia 实时监控 flume。

3. flume 的 source,sink,channel 的作用？你们 source 是什么类型？

source 组件是专门用来收集数据的，可以处理各种类型、各种格式的日志数据，包括 avro、thrift、exec、jms、spooling directory、netcat、sequence generator、syslog、http、legacy

source 组件把数据收集来以后，临时存放在 channel 中，即 channel 组件在 agent 中是专门用来存放临时数据的——对采集到的数据进行简单的缓存，可以存放在 memory、jdbc、file 等等。

sink 组件是用于把数据发送到目的地的组件，目的地包括 hdfs、logger、avro、thrift、ipc、file、null、Hbase、solr、自定义。

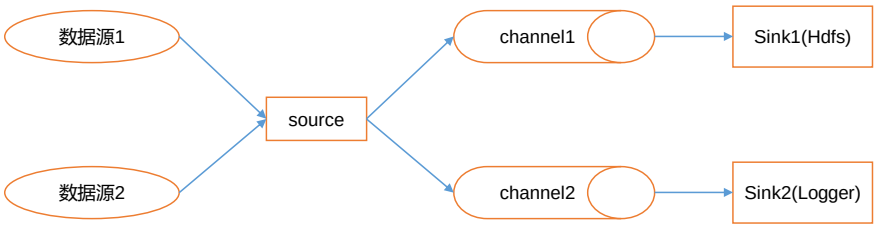
监控后台日志：exec

监控后台产生日志的端口：netcat

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

4. Flume 的 Channel Selectors

Flume Channel Selectors



Channel Selectors，可以让不同的项目日志通过不同的Channel到不同的Sink中去。官方文档上Channel Selectors 有两种类型:Replicating Channel Selector (default)和Multiplexing Channel Selector

这两种Selector的区别是:Replicating 会将source过来的events发往所有channel,而Multiplexing可以选择该发往哪些Channel。

让天下没有难学的技术

5. Flume 参数调优

1. Source

增加 Source 个数（使用 Tair Dir Source 时可增加 FileGroups 个数）可以增大 Source 的读取数据的能力。例如：当某一个目录产生的文件过多时需要将这个文件目录拆分成多个文件目录，同时配置好多个 Source 以保证 Source 有足够的能力获取到新产生的数据。

batchSize 参数决定 Source 一次批量运输到 Channel 的 event 条数，适当调大这个参数可以提高 Source 搬运 Event 到 Channel 时的性能。

2. Channel

type 选择 memory 时 Channel 的性能最好，但是如果 Flume 进程意外挂掉可能会丢失数据。type 选择 file 时 Channel 的容错性更好，但是性能上会比 memory channel 差。

使用 file Channel 时 dataDirs 配置多个不同盘下的目录可以提高性能。

Capacity 参数决定 Channel 可容纳最大的 event 条数。transactionCapacity 参数决定每次 Source 往 channel 里面写的最大 event 条数和每次 Sink 从 channel 里面读的最大 event 条数。transactionCapacity 需要大于 Source 和 Sink 的 batchSize 参数。

3. Sink

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

增加 Sink 的个数可以增加 Sink 消费 event 的能力。Sink 也不是越多越好够用就行，过多的 Sink 会占用系统资源，造成系统资源不必要的浪费。

batchSize 参数决定 Sink 一次批量从 Channel 读取的 event 条数，适当调大这个参数可以提高 Sink 从 Channel 搬出 event 的性能。

6. Flume 的事务机制

Flume 的事务机制（类似数据库的事务机制）：Flume 使用两个独立的事务分别负责从 Soucrce 到 Channel，以及从 Channel 到 Sink 的事件传递。比如 spooling directory source 为文件的每一行创建一个事件，一旦事务中所有的事件全部传递到 Channel 且提交成功，那么 Soucrce 就将该文件标记为完成。同理，事务以类似的方式处理从 Channel 到 Sink 的传递过程，如果因为某种原因使得事件无法记录，那么事务将会回滚。且所有的事件都会保持到 Channel 中，等待重新传递。

7. Flume 采集数据会丢失吗?

不会，Channel 存储可以存储在 File 中，数据传输自身有事务。

【更多 Java、HTML5、Android、python、大数据 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】