# 1

Zookeeper：

1. [zookeeper是什么（7点）](#_1._ZooKeeper_是什么？（7）)
2. [zookeeper提供了什么](#_2._ZooKeeper_提供了什么？)
3. [服务器角色](#_3._服务器角色)
4. [Zookeeper 下server工作状态](#_4._Zookeeper_下)
5. [Zk节点宕机如何处理](#_5._zk_节点宕机如何处理？)
6. [集群规则](#_6._集群最少要几台机器，集群规则是怎样的？集群中有_3)
7. [功能](#_7._Zookeeper_都有哪些功能？)

Hadoop：

1. mr的执行流程（8个步骤）
2. hdfs使用格式，hive文件使用格式，orcrc文件的差别（）
3. Hadoop shuffer原理
4. Hdfs文件上传流程（3点）
5. 数据倾斜问题解决方案（map（2）和key（2）上进行解决）
6. Mr调优方法（4点）
7. Hadoop进程
8. Hadoop与spark区别

Hive：

1. hive的动态分区
2. 项目里hive具体应用
3. Hive如何集成hbase
4. Hive查询on和where的区别
5. Hive中leftjoin
6. Hive内部表、外部表、分区表、分桶表
7. Hive日常工作
8. Hive与mysql的区别
9. Hive调优
10. sql优化
11. Hive数据倾斜原因
12. 内部表、外部表区别
13. Sort by 和order by区别
14. 数据倾斜解决方案
15. 分区、分桶；内部表、外部表区别
16. 项目hive问题
17. Join种类

Hbase

1. Hbase与hive区别
2. Hbase中rowkey设计原则
3. Scan和get功能以及实现
4. Compact用途
5. Hbase优化
6. HRegionServer宕机
7. Hbase读写流程
8. 如何提高读写性能

Spark

1. Driver功能
2. Spark部署模式
3. 为什么比mapreduce块
4. Hadoop和spark shuffle
5. Rdd宽窄依赖
6. Cache和persisit区别
7. 常规的容错方式
8. Rdd缺陷
9. Spark中数据的位置是被谁管理
10. Spark数据本地行
11. Rdd集中操作类型
12. Spark程序执行流程、为什么会产生task、如何删减task个数
13. 没有获得足够资源就执行job程序，会导致什么问题
14. Join操作优化
15. Cogroup rdd实现原理

# Zookeeper

## 1. ZooKeeper 是什么？（7）

ZooKeeper 是一个开源的分布式协调服务。它是一个为分布式应用提供一致性服务的软件，分布式应用程序可以基于 Zookeeper 实现诸如数据发布/订阅、分布式协调/通知、分布式锁和分布式队列、负载均衡、命名服务、集群管理、Master 选举等功能。

ZooKeeper 的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

Zookeeper 保证了如下分布式一致性特性：

（1）顺序一致性

（2）原子性

（3）单一视图

（4）可靠性

（5）实时性（最终一致性）

客户端的读请求可以被集群中的任意一台机器处理，如果读请求在节点上注册了监听器，这个监听器也是由所连接的 zookeeper 机器来处理。对于写请求，这些请求会同时发给其他 zookeeper 机器并且达成一致后，请求才会返回成功。因此，随着 zookeeper 的集群机器增多，读请求的吞吐会提高但是写请求的吞吐会下降。

有序性是 zookeeper 中非常重要的一个特性，所有的更新都是全局有序的，每个更新都有一个唯一的时间戳，这个时间戳称为 zxid（Zookeeper Transaction Id）。而读请求只会相对于更新有序，也就是读请求的返回结果中会带有这个zookeeper 最新的 zxid。

## 2. ZooKeeper 提供了什么？

文件系统

通知机制

## 3. 服务器角色

Leader

（1）事务请求的唯一调度和处理者，保证集群事务处理的顺序性

（2）集群内部各服务的调度者

Follower

（1）处理客户端的非事务请求，转发事务请求给 Leader 服务器

（2）参与事务请求 Proposal 的投票

（3）参与 Leader 选举投票

Observer

（1）3.0 版本以后引入的一个服务器角色，在不影响集群事务处理能力的基础上提升集群的非事务处理能力

（2）处理客户端的非事务请求，转发事务请求给 Leader 服务器

（3）不参与任何形式的投票

## 4. Zookeeper 下 Server 工作状态

服务器具有四种状态，分别是 LOOKING、FOLLOWING、LEADING、OBSERVING。

（1）LOOKING：寻找Leader状态。当服务器处于该状态时，它会认为当前集群中没有Leader，因此需要进入Leader 选举状态。

（2）FOLLOWING：跟随者状态。表明当前服务器角色是Follower。

（3）LEADING：领导者状态。表明当前服务器角色是Leader。

（4）OBSERVING：观察者状态。表明当前服务器角色是Observer。

## 5. zk 节点宕机如何处理？

Zookeeper 本身也是集群，推荐配置不少于 3 个服务器。Zookeeper 自身也要保证当一个节点宕机时，其他节点会继续提供服务。

如果是一个 Follower 宕机，还有 2 台服务器提供访问，因为 Zookeeper 上的数据是有多个副本的，数据并不会丢失；

如果是一个 Leader 宕机，Zookeeper 会选举出新的 Leader。

ZK 集群的机制是只要超过半数的节点正常，集群就能正常提供服务。只有在 ZK节点挂得太多，只剩一半或不到一半节点能工作，集群才失效。

所以

3 个节点的 cluster 可以挂掉 1 个节点(leader 可以得到 2 票>1.5)

2 个节点的 cluster 就不能挂掉任何 1 个节点了(leader 可以得到 1 票<=1)

## 6. 集群最少要几台机器，集群规则是怎样的？集群中有 3 台服务器，其中一个节点宕机，这个时候 Zookeeper 还可以使用吗？

集群规则为 2N+1 台，N>0，即 3 台。可以继续使用，单数服务器只要没超过一半的服务器宕机就可以继续使用。

## 7. Zookeeper 都有哪些功能？

集群管理：监控节点存活状态、运行请求等；

主节点选举：主节点挂掉了之后可以从备用的节点开始新一轮选主，主节点选举说的就是这个选举的过程，使用Zookeeper 可以协助完成这个过程；

分布式锁：Zookeeper 提供两种锁：独占锁、共享锁。独占锁即一次只能有一个线程使用资源，共享锁是读锁共享，读写互斥，即可以有多线线程同时读同一个资源，如果要使用写锁也只能有一个线程使用。Zookeeper可以对分布式锁进行控制。

命名服务：在分布式系统中，通过使用命名服务，客户端应用能够根据指定名字来获取资源或服务的地址，提供者等信息。

# Hadoop

## MR的执行流程。

一个文件分成多个split数据片。

每个split由多一个map进行处理。

Map处理完一个数据就把处理结果放到一个环形缓冲区内存中。

环形缓冲区满后里面的数据会被溢写到一个个小文件中。

小文件会被合并成一个大文件，大文件会按照partition进行排序。

reduce节点将所有属于自己的数据从partition中拷贝到自己的缓冲区中，并进行合并。

最后合并后的数据交给reduce处理程序进行处理。

处理后的结果存放到HDFS上。

MR运行在集群上：YARN（Yet Another Resource Negotiator）

## 你们hdfs用的格式是什么？hive用的文件格式？orc和rc有什么差别？

Csv orc

ORC File，它的全名是Optimized Row Columnar (ORC) file，其实就是对RCFile做了一些优化。据官方文档介绍，这种文件格式可以提供一种高效的方法来存储Hive数据。它的设计目标是来克服Hive其他格式的缺陷。运用ORC File可以提高Hive的读、写以及处理数据的性能。

和RCFile格式相比，ORC File格式有以下优点：

(1)、每个task只输出单个文件，这样可以减少NameNode的负载；

(2)、支持各种复杂的数据类型，比如：datetime, decimal, 以及一些复杂类型(struct, list, map, and union)；

(3)、在文件中存储了一些轻量级的索引数据；

(4)、基于数据类型的块模式压缩：a、integer类型的列用行程长度编码(run-length encoding);b、String类型的列用字典编码(dictionary encoding)；

(5)、用多个互相独立的RecordReaders并行读相同的文件；

(6)、无需扫描markers就可以分割文件；

(7)、绑定读写所需要的内存；

(8)、metadata的存储是用 Protocol Buffers的，所以它支持添加和删除一些列。

## Hadoop Shuffer原理(越详细越好)?

一、Map端的shuffle

Map端会处理输入数据并产生中间结果，这个中间结果会写到本地磁盘，而不是HDFS。每个Map的输出会先写到内存缓冲区中，当写入的数据达到设定的阈值时，系统将会启动一个线程将缓冲区的数据写到磁盘，这个过程叫做spill。

在spill写入之前，会先进行二次排序，首先根据数据所属的partition进行排序，然后每个partition中的数据再按key来排序。partition的目是将记录划分到不同的Reducer上去，以期望能够达到负载均衡，以后的Reducer就会根据partition来读取自己对应的数据。接着运行combiner(如果设置了的话)，combiner的本质也是一个Reducer，其目的是对将要写入到磁盘上的文件先进行一次处理，这样，写入到磁盘的数据量就会减少。最后将数据写到本地磁盘产生spill文件(spill文件保存在{mapred.local.dir}指定的目录中，Map任务结束后就会被删除)。

最后，每个Map任务可能产生多个spill文件，在每个Map任务完成前，会通过多路归并算法将这些spill文件归并成一个文件。至此，Map的shuffle过程就结束了。

二、Reduce端的shuffle

Reduce端的shuffle主要包括三个阶段，copy、sort(merge)和reduce。

首先要将Map端产生的输出文件拷贝到Reduce端，但每个Reducer如何知道自己应该处理哪些数据呢？因为Map端进行partition的时候，实际上就相当于指定了每个Reducer要处理的数据(partition就对应了Reducer)，所以Reducer在拷贝数据的时候只需拷贝与自己对应的partition中的数据即可。每个Reducer会处理一个或者多个partition，但需要先将自己对应的partition中的数据从每个Map的输出结果中拷贝过来。

接下来就是sort阶段，也成为merge阶段，因为这个阶段的主要工作是执行了归并排序。从Map端拷贝到Reduce端的数据都是有序的，所以很适合归并排序。最终在Reduce端生成一个较大的文件作为Reduce的输入。

## 简述一下hdfs文件上传流程

首先用户即客户端想要上传文件，就先要给namenode发个请求，告诉它说我要上传文件了（即写数据），然后namenode会返回一个响应，这个响应是namenode根据自身情况，比如会先查一下namenode里面还能存数据吗？能存多少？有几个datanode能存？（心跳信息）然后返回一个可以存储的节点列表，意思是这些里面能存，但不是能存的datanode全部返回，它会根据上传文件的大小来适当的返回，客户端收到这个列表后，就向根据返回的列表里面的datanode中写数据，文件会分为多个块（block），写的过程是以流的方式写入的，一个块存入一个DataNode，存完后DataNode就会备份，我们默认的备份数是3，存完后，会告诉namenode说我存完了。这样就结束了hdfs文件上传的流程。

## MapReduce中数据倾斜问题的解决方案?

set hive.map.aggr=true；

set hive.groupby.skewindata=true;

hive.map.aggr=true：在map中会做部分聚集操作，效率更高但需要更多的内存。

hive.groupby.skewindata=true：数据倾斜时负载均衡，当选项设定为true，生成的查询计划会有两个MRJob。第一个MRJob 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GroupByKey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MRJob再根据预处理的数据结果按照GroupByKey分布到Reduce中（这个过程可以保证相同的GroupByKey被分布到同一个Reduce中），最后完成最终的聚合操作。

由上面可以看出起到至关重要的作用的其实是第二个参数的设置，它使计算变成了两个mapreduce，先在第一个中在 shuffle 过程 partition时随机给 key 打标记，使每个key 随机均匀分布到各个 reduce 上计算，但是这样只能完成部分计算，因为相同key没有分配到相同reduce上，所以需要第二次的mapreduce,这次就回归正常 shuffle,但是数据分布不均匀的问题在第一次mapreduce已经有了很大的改善，因此基本解决数据倾斜。

2、在 key 上面做文章，在 map 阶段将造成倾斜的key 先分成多组，例如aa这个key,map 时随机在 aaa 后面加上 1,2,3,4 这四个数字之一，把 key 先分成四组，先进行一次运算，之后再恢复 key 进行最终运算。

3、能先进行 group 操作的时候先进行 group 操作，把 key 先进行一次 reduce,之后再进行count或者distinct count操作。

4、join操作中，使用map join在map端就先进行join，免得到reduce时卡住。

以上4中方式，都是根据数据倾斜形成的原因进行的一些变化。要么将 reduce 端的隐患在 map 端就解决，要么就是对 key 的操作，以减缓reduce 的压力。

## 简列几条mapreduce的调优方法

1. 设置combiner，在Map端提前进行一次reduce处理。可减少Map Task中间输出的结果，从而减少各个Reduce Task的远程拷贝数据量，最终表现为Map Task和Reduce Task执行时间缩短。
2. 设置reduce数量。默认情况下，一个block对应一个map，对应一个reduce。
3. Map任务的输出压缩。
4. Shuffle参数设置。设置排序因子、内存缓冲区大小(默认100M)，设置内存缓冲使用比，设置每个map任务和reduce任务运行时使用的虚拟cpu数量。

## Hadoo中有哪几个进程，各自的作用是什么？

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.

## 简单说明下hdfs中, namenode, datanode的作用

namenode:

Namenode是中心服务器，单一节点（简化系统的设计和实现），负责管理文件系统的名称空间（namespace）以及客户端对文件的访问。

文件操作，Namenode负责文件元数据的操作，DataNode负责处理文件内容的读写请求，跟文件内容相关的数据流不会经过Namenode，只会询问它跟那个DataNode联系，否则Namenode会成为系统的瓶颈。

副本存放在哪些DataNode上由Namenode来控制，根据全局情况作出块放置决定，读取文件时Namenode尽量让用户先读取最近的副本，降低带块消耗和读取延时。

Namenode全权管理数据块的复制，它周期性地从集群中的每个Datanode接收心跳信号和块状态报告。接收到心跳信号意味着该Datanode节点工作正常。块状态报告包含了一个该Datanode上所有数据块的列表。

Namenode是主节点，存储文件的元数据如文件名，文件目录结构，文件属性（生成时间，副本数，文件权限）以及每个文件的块列表，以及块所在的DataNode等等。

datanode:

一个数据块在Datanode以文件存储在磁盘上，包括两个文件，一个是数据本身，一个是元数据包括数据块的长度，块数据的校验和，以及时间戳。

Datanode启动后想namenode注册，通过后，周期性（1小时）的向namenode上报所有的块信息。

心跳是每3秒一次，心跳返回结果带有namenode给该Datanode的命令如复制块数据到另一台机器，或删除某个数据块。如果超过10分钟没有收到某个Datanode的心跳，则认为该节点不可用。

集群运行中可用安全加入和退出一些机器。

文件切分成块，（默认大小128M），以块为单位，每个块有多个副本存储在不同的机器上，副本数可在文件生成时指定（默认3）

Datanode再本地文件系统存储文件块数据，以及块数据的校验和。

可以创建，删除，移动和重命名文件，当文件创建，写入和关闭之后不能修改文件的内容。

当Datanode读取block的时候，它会计算checksum，如果计算后的checksum，与block创建时值不一样，说明该block已经损坏。

如果块已损坏，Client会读取其它Datanode上的block.

namenode标记该块已经损坏，然后复制block达到预期设置的文件备份数。

Datanode在其文件创建后三周验证其checksum

## hadoop和 spark都是并行计算,那么他们有什么相同和区别?

两者都是用mr模型来进行并行计算，hadoop的一个作业称为job，job里面分为map task和reduce task，每个task都是在自己的进程中运行的，当task结束时，进程也会结束

spark用户提交的任务成为application，一个application对应一个sparkcontext，app中存在多个job，每触发一次action操作就会产生一个job

这些job可以并行或串行执行，每个job中有多个stage，stage是shuffle过程中DAG Schaduler 通过RDD之间的依赖关系划分job而来的，每个stage里面有多个task，组成taskset有 TaskSchaduler 分发到各个executor中执行，executor的生命周期是和app一样的，即使没有job运行也是存在的，所以task可以快速启动读取内存进行计算

hadoop的job只有map和reduce操作，表达能力比较欠缺而且在mr过程中会重复的读写hdfs，造成大量的io操作，多个job需要自己管理关系。

spark的迭代计算都是在内存中进行的，API中提供了大量的RDD操作如join，groupby等，而且通过DAG图可以实现良好的容错。

# Hive

## 描述一下hive动态分区和分桶使用场景和使用方法。

1、hive的动态分区

hive的静态分区需要用户在插入数据的时候必须**手动指定hive的分区字段值**，但是这样的话会导致用户的操作复杂度提高，而且在使用的时候会导致数据只能插入到某一个指定分区，无法让数据散列分布，因此更好的方式是当数据在进行插入的时候，根据数据的某一个字段或某几个字段值动态的将数据插入到不同的目录中，此时，引入动态分区。

--hive设置hive动态分区开启

set hive.exec.dynamic.partition=true;

默认：true

--hive的动态分区模式

set hive.exec.dynamic.partition.mode=nostrict;

默认：strict（至少有一个分区列是静态分区）

2、分桶

1、Hive分桶表是对列值取hash值得方式，将不同数据放到不同文件中存储

​2、对于hive中每一个表、分区都可以进一步进行分桶

​3、由列的hash值除以桶的个数来决定每条数据划分在哪个桶中

--设置hive支持分桶

set hive.enforce.bucketing=true;

## 项目里具体hive做的哪一块？

在项目里我们可以利用hive做数据清洗、数据分析、生成特征值列。

## HIVE是怎么集成HBase的？具体讲讲。

Hive是建立在Hadoop之上的数据仓库基础构架、是为了减少MapReduce编写工作的批处理系统，**Hive本身不存储和计算数据，它完全依赖于HDFS和MapReduce。**

Hive可以理解为一个客户端工具，将我们的sql操作转换为相应的MapReduce jobs，然后在Hadoop上面运行。

HBase全称为Hadoop Database，即HBase是Hadoop的数据库，是一个分布式的存储系统。HBase利用Hadoop的HDFS作为其文件存储系统，利用Hadoop的MapReduce来处理HBase中的海量数据。利用Zookeeper作为其协调工具。

HBase数据库的缺点在于——**语法格式异类，没有类sql的查询方式**，因此在实际的业务当中操作和计算数据非常不方便，但是Hive就不一样了，**Hive支持标准的sql语法**，于是我们就希望通过Hive这个客户端工具对HBase中的数据进行操作与查询，进行相应的数据挖掘，这就是所谓Hive与hBase整合的含义。

根据CDH版本决定，有些版本无需拷贝jar包，因为hbase的lib文件夹下没有hive-hbase-handler.jar包，这个包的存放路径在hive的lib文件夹下面，所以需要拷贝，在hbase的master节点中

通过Hive -> 配置 -> 搜索栏中搜索hive.server2.enable.doAs ，默认为勾选，取消勾选即可，即能修改配置为 false。

重启 hive、hbase、hue、还有oozie

hbase.zookeeper.quorum：

指定HBase使用的zookeeper集群，默认端口是2181，可以不指定，如果指定，格式为zkNode1:2222,zkNode2:2222,zkNode3:2222

zookeeper.znode.parent

指定HBase在zookeeper中使用的根目录

使用hive-hbase-handler.jar连接hbase

使用外部表映射到HBase中的表，这样，在Hive中删除表，并不会删除HBase中的表，否则，就会删除。

另外，除了rowkey，其他字段使用Map结构来保存HBase中的每一个列族

## Hive查询的时候on和where有什么区别？为什么？

on 和 where 在筛选条件的时候，on 会显示所有满足或不满足条件的数据而 where 只显示满足条件的数据。

## Hive里面的left join是怎么回事？他是怎么执行的？

LEFT JOIN一般用于A表有而B表没有的记录进行关联，然后用on或where过滤掉B表中有NULL的记录行

1.不考虑where条件下,left join 会把左表所有数据查询出来，on及其后面的条件仅仅会影响右表的数据(符合就显示,不符合全部为null)

2.在匹配阶段，where子句的条件都不会被使用，仅在匹配阶段完成以后，where子句条件才会被使用，它将从匹配阶段产生的数据中检索过滤

3.所以左连接关注的是左边的主表数据，不应该把on后面的从表中的条件加到where后，这样会影响原有主表中的数据

4.where后面：是先连接然生成临时查询结果，然后再筛选

on后面：先根据条件过滤筛选，再连接生成临时查询结果

5.对于条件在on加个and还是用子查询，查询结果是一模一样的，至于如何使用这个需要分情况，用子查询的话会多一个maptask，但是如果利用这个子查询能过滤到很多数据的话，用子查询还是比较建议的，因为不会加载太多的数据到内存中，如果过滤数据不多的情况下，建议用on后面加and条件

## 说一下hive内部表、外部表、分区表。

**内部表（managed table）**

默认创建的是内部表（managed table），存储位置在hive.metastore.warehouse.dir设置，默认位置是/user/hive/warehouse。

导入数据的时候是将文件剪切（移动）到指定位置，即原有路径下文件不再存在

删除表的时候，数据和元数据都将被删除

默认创建的就是内部表create table xxx (xx xxx)

**外部表（external table）**

外部表文件可以在外部系统上，只要有访问权限就可以

外部表导入文件时不移动文件，仅仅是添加一个metadata

删除外部表时原数据不会被删除

分辨外部表内部表可以使用DESCRIBE FORMATTED table\_name 命令查看

创建外部表命令添加一个external即可，即create external table xxx (xxx)

外部表指向的数据发生变化的时候会自动更新，不用特殊处理

**分区表（Partitioned table）**

有些时候数据是有组织的，比方按日期/类型等分类，而查询数据的时候也经常只关心部分数据。

## 谈谈对hive的理解，日常用hive做哪些工作。

1、Hive解决了用户从一个现有的数据基础架构转移到Hadoop上，并且是基于传统关系型数据库和结构化查询语句，方面sql用户的使用。利用HQL来查询存储在Hadoop集群中的数据。

Hive不支持记录级别的更新、插入或者删除操作。但是用户可以通过查询生成新表或者将查询结果导入到文件中。

Hive查询延时严重，因为Hadoop是一个面向批处理的系统，而MapReduce任务启动过程需要消耗较长的时间，HQL在底层转化为MapReduce任务执行。

Hive不支持事务

2、Hive最终会被转化成**MapReduce的代码去执行**，所以数据库的优化原则基本上都不适用于 Hive。也正因如此，**Hive实际上是用来做计算的**，而不像数据库是用作存储的。

## hive和mysql有什么区别，大数据为什么不用mysql做存储和数据处理。

1.查询语言不同：hive是hql语言，mysql是sql语句；

2.数据存储位置不同：hive是把数据存储在hdfs上，而mysql数据是存储在自己的系统中；

3.数据格式：hive数据格式可以用户自定义，mysql有自己的系统定义格式；

4.数据更新：hive不支持数据更新，只可以读，不可以写，而sql支持数据更新；

5.索引：hive没有索引，因此查询数据的时候是通过mapreduce很暴力的把数据都查询一遍，也造成了hive查询数据速度很慢的原因，而mysql有索引；

6.延迟性：hive延迟性高，原因就是上边一点所说的，而mysql延迟性低；

7.数据规模：hive存储的数据量超级大，而mysql只是存储一些少量的业务数据；

8.底层执行原理：hive底层是用的mapreduce，而mysql是excutor执行器；

## hive如何调优？

hive调优可以分为几个模块进行考虑，数据的压缩与存储，sql的优化，hive参数的优化，解决数据的倾斜等。

1）数据的压缩与存储

选择合适的存储格式与压缩方式能提高hive的分析效率，常用的压缩算法有GZIP、BZIP2、LZO、Snappy。常见的存储格式有TextFile（默认）、Sequence Files、RCFile、ORCFile、Parquet，也可以自定义文件格式。

2）创建分区表，桶表，拆分表

3）优化sql

设置条件，减少job数量。如：where条件优化、union优化、count distinct优化、用in 来代替join、消灭子查询内的 group by、 COUNT(DISTINCT)，MAX，MIN、join 优化

4）hive参数优化

5）数据倾斜

6）合并小文件

7）查看sql的执行计划

8）在脚本中并行

## 工作中SQL的优化

设置条件，减少job数量。如：where条件优化，优化后where条件在map端执行而不是在reduce端执行；union优化，尽量不要使用union而是使用 union all 然后在用group by 去重；count distinct优化，不要使用count (distinct cloumn) ,使用子查询；如果需要根据一个表的字段来约束另为一个表，尽量用in 来代替join；消灭子查询内的 group by、 COUNT(DISTINCT)，MAX，MIN，可以减少job的数量；join 优化。

## hive数据倾斜原因和处理。

**原因：**某个reduce的数据输入量远远大于其他reduce数据的输入量

1) key分布不均匀

2) 业务数据本身的特性

3) 建表时考虑不周

4) 某些SQL语句本身就有数据倾斜

**处理：**

1）参数调节

set hive.map.aggr=true

set hive.groupby.skewindata=true

2) 熟悉数据的分布，优化sql的逻辑，找出数据倾斜的原因。

## 内部表外部表区别

未被external修饰的是内部表（managed table），被external修饰的为外部表（external table）；

**区别：**

1）内部表数据由Hive自身管理，外部表数据由HDFS管理；

内部表数据存储的位置是hive.metastore.warehouse.dir（默认：/user/hive/warehouse），外部表数据的存储位置由自己制定（如果没有LOCATION，Hive将在HDFS上的/user/hive/warehouse文件夹下以外部表的表名创建一个文件夹，并将属于这个表的数据存放在这里）；

2）删除内部表会直接删除元数据（metadata）及存储数据；删除外部表仅仅会删除元数据，HDFS上的文件并不会被删除；

3）对内部表的修改会将修改直接同步给元数据，而对外部表的表结构和分区进行修改，则需要修复

## sort by 和order By区别

order by：

order by 会对数据进行全局排序,和oracle和mysql等数据库中的order by 效果一样，它只在一个reduce中进行所以数据量特别大的时候效率非常低。

而且当设置：set hive.mapred.mode=strict的时候不指定limit，执行select会报错，如下：LIMIT must also be specified。

sort by：

sort by 是单独在各自的reduce中进行排序，所以并不能保证全局有序，一般和distribute by 一起执行，而且distribute by 要写在sort by前面。

如果mapred.reduce.tasks=1和order by效果一样，如果大于1会分成几个文件输出每个文件会按照指定的字段排序，而不保证全局有序。

sort by 不受 hive.mapred.mode 是否为strict ,nostrict 的影响。

## 列举Hive几种数据倾斜的场景以及解决方案?

1)、key分布不均匀

2)、业务数据本身的特性

3)、建表时考虑不周

4)、某些SQL语句本身就有数据倾斜

解决方法：

1参数调节：

hive.map.aggr = true

Map 端部分聚合，相当于Combiner

hive.groupby.skewindata=true

有数据倾斜的时候进行负载均衡，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。

2 SQL语句调节：

如何Join：

关于驱动表的选取，选用join key分布最均匀的表作为驱动表

做好列裁剪和filter操作，以达到两表做join的时候，数据量相对变小的效果。

大小表Join：

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.

大表Join大表：

把空值的key变成一个字符串加上随机数，把倾斜的数据分到不同的reduce上，由于null值关联不上，处理后并不影响最终结果。

count distinct大量相同特殊值

count distinct时，将值为空的情况单独处理，如果是计算count distinct，可以不用处理，直接过滤，在最后结果中加1。如果还有其他计算，需要进行group by，可以先将值为空的记录单独处理，再和其他计算结果进行union。

group by维度过小：

采用sum() group by的方式来替换count(distinct)完成计算。

特殊情况特殊处理：

在业务逻辑优化效果的不大情况下，有些时候是可以将倾斜的数据单独拿出来处理。最后union回去。

## Hive中内部表与外部表的区别,分区与桶的区别

内部表与外部表的区别：

1.未被external修饰的是内部表，被external修饰的为外部表（external table）

2.内部表数据由Hive自身管理，外部表数据由HDFS管理

3.内部表数据存储的位置是hive.metastore.warehouse.dir，外部表数据的存储位置由自己制定（如果没有LOCATION，Hive将在HDFS上的/user/hive/warehouse文件夹下以外部表的表名创建一个文件夹，并将属于这个表的数据存放在这里）；

4.删除内部表会直接删除元数据（metadata）及存储数据；删除外部表仅仅会删除元数据，HDFS上的文件并不会被删除

5.对内部表的修改会将修改直接同步给元数据，而对外部表的表结构和分区进行修改，则需要修复（MSCK REPAIR TABLE table\_name;）

分区与分桶：

分区：

Hive的分区使用HDFS的子目录功能实现。每一个子目录包含了分区对应的列名和每一列的值

分桶：

分桶表是在表或者分区表的基础上，进一步对表进行hash，并用hash结果除以桶的个数做取余运算的方式来分桶，保证了每个桶中都有数据，但每个桶中的数据条数不一定相等。

区别：

分桶随机分割数据库，分区是非随机分割数据库。因为分桶是按照列的哈希函数进行分割的，相对比较平均；而分区是按照列的值来进行分割的，容易造成数据倾斜。

分桶是对应不同的文件（细粒度），分区是对应不同的文件夹（粗粒度）。桶是更为细粒度的数据范围划分，分桶的比分区获得更高的查询处理效率，使取样更高效。

## hive项目中遇到的问题?怎么解决

数据倾斜 （参考前面的回答）/ 小文件过多（进行小文件合并，调优参数如下）

set hive.merge.mapfiles = true ##在 map only 的任务结束时合并小文件

set hive.merge.mapredfiles = false ## true 时在 MapReduce 的任务结束时合并小文件

set hive.merge.size.per.task = 25610001000 ##合并文件的大小

set mapred.max.split.size=256000000; ##每个 Map 最大分割大小

set mapred.min.split.size.per.node=1; ##一个节点上 split 的最少值

set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat; ##执行 Map 前进行小文件合并。

## hive的几种join

**common join** 普通连接，在SQL中不特殊指定连接方式使用的都是这种普通连接。

**map join map** 端连接，与普通连接的区别是这个连接中不会有reduce阶段存在，连接在map端完成

适用场景：大表与小表连接，小表数据量应该能够完全加载到内存，否则不适用

**bucket map join**

分桶连接：hive 建表的时候支持hash 分区通过指定clustered by (col\_name,xxx ) into number\_buckets buckets 关键字.当连接的两个表的join key 就是bucket column 的时候，就可以通过 hive.optimize.bucketmapjoin= true 来执行该优化策略。

**skew join**

倾斜连接，主要针对数据倾斜的情况优化。

# Hbase

## hbase和hive的区别



## 描述 HBase 的 rowKey 的设计原则？

① Rowkey 长度原则

Rowkey 是一个二进制码流，Rowkey 的长度被很多开发者建议说设计在 10~100 个字节，不过建议是越短越好，不要超过 16 个字节。

原因如下：

（1）数据的持久化文件 HFile 中是按照 KeyValue 存储的，如果 Rowkey 过长比如 100个字节，1000 万列数据光 Rowkey 就要占用 100\*1000 万=10 亿个字节，将近 1G 数据，这会极大影响 HFile 的存储效率；

（2）MemStore 将缓存部分数据到内存，如果 Rowkey 字段过长内存的有效利用率会降低，系统将无法缓存更多的数据，这会降低检索效率。因此 Rowkey 的字节长度越短越好。

（3）目前操作系统是都是 64 位系统，内存 8 字节对齐。控制在 16 个字节，8 字节

的整数倍利用操作系统的最佳特性。

② Rowkey 散列原则

如果Rowkey 是按时间戳的方式递增，不要将时间放在二进制码的前面，建议将Rowkey的高位作为散列字段，由程序循环生成，低位放时间字段，这样将提高数据均衡分布在每个Regionserver 实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息将产生所有新数据都在一个 RegionServer 上堆积的热点现象，这样在做数据检索的时候负载将会集中在个别 RegionServer，降低查询效率。

③ Rowkey 唯一原则

必须在设计上保证其唯一性。

## 描述 HBase 中 scan 和 get 的功能以及实现的异同？

 HBase 的查询实现只提供两种方式：

1）按指定 RowKey 获取唯一一条记录，get 方法（org.apache.hadoop.hbase.client.Get）Get 的方法处理分两种 : 设置了 ClosestRowBefore 和没有设置 ClosestRowBefore 的rowlock。主要是用来保证行的事务性，即每个 get 是以一个 row 来标记的。一个 row 中可以有很多 family 和 column。

2）按指定的条件获取一批记录，scan 方法(org.apache.Hadoop.hbase.client.Scan）实现条件查询功能使用的就是 scan 方式。

## 简述 HBase 中 compact 用途是什么，什么时候触发，分为哪两种，有什么区别，有哪些相关配置参数？

在 hbase 中每当有 memstore 数据 flush 到磁盘之后，就形成一个 storefile，当 storeFile的数量达到一定程度后，就需要将 storefile 文件来进行 compaction 操作。

Compact 的作用：

① 合并文件

② 清除过期，多余版本的数据

③ 提高读写数据的效率

HBase 中实现了两种 compaction 的方式：minor and major. 这两种 compaction 方式的

区别是：

1、Minor 操作只用来做部分文件的合并操作以及包括 minVersion=0 并且设置 ttl 的过期版本清理，不做任何删除数据、多版本数据的清理工作。

2、Major 操作是对 Region 下的 HStore 下的所有 StoreFile 执行合并操作，最终的结果是整理合并出一个文件。

## HBase 优化？

（1）高可用

在 HBase 中 Hmaster 负责监控 RegionServer 的生命周期，均衡 RegionServer 的负载，如果 Hmaster 挂掉了，那么整个 HBase 集群将陷入不健康的状态，并且此时的工作状态并不会维持太久。所以 HBase 支持对 Hmaster 的高可用配置。

（2）预分区

每一个 region 维护着 startRow 与 endRowKey，如果加入的数据符合某个 region 维护的rowKey 范围，则该数据交给这个 region 维护。那么依照这个原则，我们可以将数据所要投放的分区提前大致的规划好，以提高 HBase 性能 。

（3）RowKey 设计

一条数据的唯一标识就是 rowkey，那么这条数据存储于哪个分区，取决于 rowkey 处于哪个一个预分区的区间内，设计 rowkey 的主要目的 ，就是让数据均匀的分布于所有的 region中，在一定程度上防止数据倾斜。接下来我们就谈一谈 rowkey 常用的设计方案

（4）7.4 内存优化

HBase 操作过程中需要大量的内存开销，毕竟 Table 是可以缓存在内存中的，一般会分配整个可用内存的 70%给 HBase 的 Java 堆。但是不建议分配非常大的堆内存，因为 GC 过程持续太久会导致 RegionServer 处于长期不可用状态，一般 16~48G 内存就可以了，如果因为框架占用内存过高导致系统内存不足，框架一样会被系统服务拖死。

（5）基础优化

## HRegionServer 宕机如何处理？

1）ZooKeeper 会监控 HRegionServer 的上下线情况，当 ZK 发现某个 HRegionServer 宕机之后会通知 HMaster 进行失效备援；

2）该 HRegionServer 会停止对外提供服务，就是它所负责的 region 暂时停止对外提供服务；

3）HMaster 会将该 HRegionServer 所负责的 region 转移到其他 HRegionServer 上，并且会对 HRegionServer 上存在 memstore 中还未持久化到磁盘中的数据进行恢复；

4）这个恢复的工作是由 WAL 重播来完成，这个过程如下：

· wal 实际上就是一个文件，存在/hbase/WAL/对应 RegionServer 路径下。

· 宕机发生时，读取该 RegionServer 所对应的路径下的 wal 文件，然后根据不同的region 切分成不同的临时文件 recover.edits。

· 当 region 被分配到新的 RegionServer 中，RegionServer 读取 region 时会进行是否存在 recover.edits，如果有则进行恢复。

## HBase 读写流程？（☆☆☆☆☆）

读：

① HRegionServer 保存着 meta 表以及表数据，要访问表数据，首先 Client 先去访问zookeeper，从 zookeeper 里面获取 meta 表所在的位置信息，即找到这个 meta 表在哪个HRegionServer 上保存着。

② 接着 Client 通过刚才获取到的 HRegionServer 的 IP 来访问 Meta 表所在的HRegionServer，从而读取到 Meta，进而获取到 Meta 表中存放的元数据。

③ Client 通过元数据中存储的信息，访问对应的 HRegionServer，然后扫描所在HRegionServer 的 Memstore 和 Storefile 来查询数据。

④ 最后 HRegionServer 把查询到的数据响应给 Client。

写：

① Client 先访问 zookeeper，找到 Meta 表，并获取 Meta 表元数据。

② 确定当前将要写入的数据所对应的 HRegion 和 HRegionServer 服务器。

③ Client 向该 HRegionServer 服务器发起写入数据请求，然后 HRegionServer 收到请求并响应。

④ Client 先把数据写入到 HLog，以防止数据丢失。

⑤ 然后将数据写入到 Memstore。

⑥ 如果 HLog 和 Memstore 均写入成功，则这条数据写入成功

⑦ 如果 Memstore 达到阈值，会把 Memstore 中的数据 flush 到 Storefile中。

⑧ 当 Storefile 越来越多，会触发 Compact 合并操作，把过多的 Storefile合并成一个大的 Storefile。

⑨ 当 Storefile 越来越大，Region 也会越来越大，达到阈值后，会触发 Split 操作，将Region 一分为二。

## 如何提高 HBase 客户端的读写性能？请举例说明（☆☆☆☆☆）

1 开启 bloomfilter 过滤器，开启 bloomfilter 比没开启要快 3、4 倍

2 Hbase 对于内存有特别的需求，在硬件允许的情况下配足够多的内存给它

3 通过修改 hbase-env.sh 中的

export HBASE\_HEAPSIZE=3000 #这里默认为 1000m

4 增大 RPC 数量

通过修改 hbase-site.xml 中的 hbase.regionserver.handler.count 属性，可以适当的放大RPC 数量，默认值为 10 有点小。

## 直接将时间戳作为行健，在写入单个 region 时候会发生热点问题，为什么呢？（☆☆☆☆☆）

region 中的 rowkey 是有序存储，若时间比较集中。就会存储到一个 region 中，这样一个 region 的数据变多，其它的 region 数据很少，加载数据就会很慢，直到 region 分裂，此问题才会得到缓解。

# Spark

# spark精华面试题

## 1、driver的功能是什么？

1）一个Spark作业运行时包括一个Driver进程，也是作业的主进程，具有main函数，并且有SparkContext的实例，是程序的人口点；

2）功能：负责向集群申请资源，向master注册信息，负责了作业的调度，负责作业的解析、生成Stage并调度Task到Executor上。包括DAGScheduler，TaskScheduler。

## 2、spark的有几种部署模式，每种模式特点？

1）本地模式：适用于测试  
　　2） standalone 模式：使用spark自带的资源调度框架  
　　3） spark on yarn 模式：最流行的方式，使用yarn集群调度资源  
　　4） mesos模式：国外使用多

## 3、Spark为什么比mapreduce快？

1）基于内存计算，减少低效的磁盘交互；

2）高效的调度算法，基于DAG；

3）容错机制Linage，精华部分就是DAG和Lingae

## 4、hadoop和spark的shuffle相同和差异？

1）从 high-level 的角度来看，两者并没有大的差别。 都是将 mapper（Spark 里是 ShuffleMapTask）的输出进行 partition，不同的 partition 送到不同的 reducer（Spark 里 reducer 可能是下一个 stage 里的 ShuffleMapTask，也可能是 ResultTask）。Reducer 以内存作缓冲区，边 shuffle 边 aggregate 数据，等到数据 aggregate 好以后进行 reduce() （Spark 里可能是后续的一系列操作）。

2）从 low-level 的角度来看，两者差别不小。 Hadoop MapReduce 是 sort-based，进入 combine() 和 reduce() 的 records 必须先 sort。这样的好处在于 combine/reduce() 可以处理大规模的数据，因为其输入数据可以通过外排得到（mapper 对每段数据先做排序，reducer 的 shuffle 对排好序的每段数据做归并）。目前的 Spark 默认选择的是 hash-based，通常使用 HashMap 来对 shuffle 来的数据进行 aggregate，不会对数据进行提前排序。如果用户需要经过排序的数据，那么需要自己调用类似 sortByKey() 的操作；如果你是Spark 1.1的用户，可以将spark.shuffle.manager设置为sort，则会对数据进行排序。在Spark 1.2中，sort将作为默认的Shuffle实现。

3）从实现角度来看，两者也有不少差别。 Hadoop MapReduce 将处理流程划分出明显的几个阶段：map(), spill, merge, shuffle, sort, reduce() 等。每个阶段各司其职，可以按照过程式的编程思想来逐一实现每个阶段的功能。在 Spark 中，没有这样功能明确的阶段，只有不同的 stage 和一系列的 transformation()，所以 spill, merge, aggregate 等操作需要蕴含在 transformation() 中。如果我们将 map 端划分数据、持久化数据的过程称为 shuffle write，而将 reducer 读入数据、aggregate 数据的过程称为 shuffle read。那么在 Spark 中，问题就变为怎么在 job 的逻辑或者物理执行图中加入 shuffle write 和 shuffle read 的处理逻辑？以及两个处理逻辑应该怎么高效实现？ Shuffle write由于不要求数据有序，shuffle write 的任务很简单：将数据 partition 好，并持久化。之所以要持久化，一方面是要减少内存存储空间压力，另一方面也是为了 fault-tolerance。

## 5、RDD宽依赖和窄依赖？

RDD和它依赖的parent RDD(s)的关系有两种不同的类型，即窄依赖（narrow dependency）和宽依赖（wide dependency）。

1）窄依赖指的是每一个parent RDD的Partition最多被子RDD的一个Partition使用

2）宽依赖指的是多个子RDD的Partition会依赖同一个parent RDD的Partition

## 6、cache和pesist的区别

1）cache和persist都是用于将一个RDD进行缓存的，这样在之后使用的过程中就不需要重新计算了，可以大大节省程序运行时间；

2） cache只有一个默认的缓存级别MEMORY\_ONLY ，cache调用了persist，而persist可以根据情况设置其它的缓存级别；

3）executor执行的时候，默认60%做cache，40%做task操作，persist最根本的函数，最底层的函数

## 7、常规的容错方式有哪几种类型？RDD通过Linage（记录数据更新）的方式为何很高效？

1）.数据检查点,会发生拷贝，浪费资源

2）.记录数据的更新，每次更新都会记录下来，比较复杂且比较消耗性能

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

1） ​lazy记录了数据的来源，RDD是不可变的，且是lazy级别的，且rDD之间构成了链条，lazy是弹性的基石。由于RDD不可变，所以每次操作就产生新的rdd，不存在全局修改的问题，控制难度下降，所有有计算链条将复杂计算链条存储下来，计算的时候从后往前回溯900步是上一个stage的结束，要么就checkpoint

2） ​记录原数据，是每次修改都记录，代价很大如果修改一个集合，代价就很小，官方说rdd是粗粒度的操作，是为了效率，为了简化，每次都是操作数据集合，写或者修改操作，都是基于集合的rdd的写操作是粗粒度的，rdd的读操作既可以是粗粒度的也可以是细粒度，读可以读其中的一条条的记录。

3） ​简化复杂度，是高效率的一方面，写的粗粒度限制了使用场景如网络爬虫，现实世界中，大多数写是粗粒度的场景

## 8、RDD有哪些缺陷？

1）不支持细粒度的写和更新操作（如网络爬虫），spark写数据是粗粒度的所谓粗粒度，就是批量写入数据，为了提高效率。但是读数据是细粒度的也就是说可以一条条的读  
　　2）不支持增量迭代计算，Flink支持

## 9、Spark中数据的位置是被谁管理的？

每个数据分片都对应具体物理位置，数据的位置是被blockManager，无论数据是在磁盘，内存还是tacyan，都是由blockManager管理

## 10、Spark的数据本地性有哪几种？

答：Spark中的数据本地性有三种：a.PROCESS\_LOCAL是指读取缓存在本地节点的数据b.NODE\_LOCAL是指读取本地节点硬盘数据c.ANY是指读取非本地节点数据通常读取数据PROCESS\_LOCAL>NODE\_LOCAL>ANY，尽量使数据以PROCESS\_LOCAL或NODE\_LOCAL方式读取。其中PROCESS\_LOCAL还和cache有关，如果RDD经常用的话将该RDD cache到内存中，注意，由于cache是lazy的，所以必须通过一个action的触发，才能真正的将该RDD cache到内存中

## 11、rdd有几种操作类型？

1）transformation，rdd由一种转为另一种rdd

2）action，

3）cronroller，crontroller是控制算子,cache,persist，对性能和效率的有很好的支持三种类型，不要回答只有2中操作

## 12、Spark程序执行，有时候默认为什么会产生很多task，怎么修改默认task执行个数？

1）因为输入数据有很多task，尤其是有很多小文件的时候，有多少个输入block就会有多少个task启动；

2）spark中有partition的概念，每个partition都会对应一个task，task越多，在处理大规模数据的时候，就会越有效率。不过task并不是越多越好，如果平时测试，或者数据量没有那么大，则没有必要task数量太多。

3）参数可以通过spark\_home/conf/spark-default.conf配置文件设置:spark.sql.shuffle.partitions 50 spark.default.parallelism 10第一个是针对spark sql的task数量第二个是非spark sql程序设置生效

## 13、为什么Spark Application在没有获得足够的资源，job就开始执行了，可能会导致什么什么问题发生?

答：会导致执行该job时候集群资源不足，导致执行job结束也没有分配足够的资源，分配了部分Executor，该job就开始执行task，应该是task的调度线程和Executor资源申请是异步的；如果想等待申请完所有的资源再执行job的：需要将spark.scheduler.maxRegisteredResourcesWaitingTime设置的很大；spark.scheduler.minRegisteredResourcesRatio 设置为1，但是应该结合实际考虑否则很容易出现长时间分配不到资源，job一直不能运行的情况。

## 14、join操作优化经验？

join其实常见的就分为两类： map-side join 和 reduce-side join。当大表和小表join时，用map-side join能显著提高效率。将多份数据进行关联是数据处理过程中非常普遍的用法，不过在分布式计算系统中，这个问题往往会变的非常麻烦，因为框架提供的 join 操作一般会将所有数据根据 key 发送到所有的 reduce 分区中去，也就是 shuffle 的过程。造成大量的网络以及磁盘IO消耗，运行效率极其低下，这个过程一般被称为 reduce-side-join。如果其中有张表较小的话，我们则可以自己实现在 map 端实现数据关联，跳过大量数据进行 shuffle 的过程，运行时间得到大量缩短，根据不同数据可能会有几倍到数十倍的性能提升。

## 15、介绍一下cogroup rdd实现原理，你在什么场景下用过这个rdd？

答：cogroup的函数实现:这个实现根据两个要进行合并的两个RDD操作,生成一个CoGroupedRDD的实例,这个RDD的返回结果是把相同的key中两个RDD分别进行合并操作,最后返回的RDD的value是一个Pair的实例,这个实例包含两个Iterable的值,第一个值表示的是RDD1中相同KEY的值,第二个值表示的是RDD2中相同key的值.由于做cogroup的操作,需要通过partitioner进行重新分区的操作,因此,执行这个流程时,需要执行一次shuffle的操作(如果要进行合并的两个RDD的都已经是shuffle后的rdd,同时他们对应的partitioner相同时,就不需要执行shuffle