一线大厂大数据面试真题解析（30道）

简

1. 怎么查看Hive有什么自带函数?怎么查看函数的详细信息? (腾讯)

(1) 问题分析

考官主要考核你对Hive命令的掌握程度,因此需要说出查看Hive自带函数的命令和查看函数详细信息的命令。

(2) 核心问题回答

```
-- 查看系统自带函数
show functions;
-- 显示自带函数的用法
desc function upper;
或
desc function extended upper;
```

2. 写出你常用的HDFS命令? (京东)

(1) 问题分析

考官主要考核你对HDFS命令的梳理程度,因此需要讲出常用的HDFS命令,至少10个。

(2) 核心问题回答

```
-mkdir:在HDFS上创建目录;
-moveFromLocal:从本地剪切粘贴到HDFS
-appendToFile:追加一个文件到已经存在的文件末尾
-cat:显示文件内容
-copyFromLocal:从本地文件系统中拷贝文件到HDFS路径
-copyToLocal:从HDFS拷贝到本地
-cp:从HDFS的一个路径拷贝到HDFS的另一个路径
-mv:在HDFS目录中移动文件
-tail:显示一个文件的末尾
-rm:删除文件或文件夹
-rmdir:删除空目录
-du:统计文件夹的大小信息
-setrep:设置HDFS中文件的副本数量
-lsr:递归查看根目录下所有文件和文件夹
-df:统计文件系统的可用空间信息
-touchz:在Hadoop指定目录下新建一个空文件
```

3. Redis分布式锁怎么实现? (京东金融)

(1) 问题分析

考官主要考核你对Redis分布式锁的理解,因此需要讲出Redis分布式锁的概念以及具体实现即可。

(2) 核心问题回答

分布式锁是控制分布式系统之间同步访问共享资源的一种方式,其具体实现是使用set命令获取分布式锁，使用Redis+lua脚本释放锁。

4. HDFS文件系统中,Fsimage和Edit的区别? (水滴互助)

(1) 问题分析

考官主要考核你对Fsimage和Edit的理解,因此需要讲出Fsimage和Edit的概念和区别即可。

(2) 核心问题回答

Fsimage镜像文件:是元数据的一个持久化的检查点,包含Hadoop文件系统中的所有目录和文件元数据信息,但不包含文件块位置的信息。文件块位置信息只存储在内存中,是在DataNode加入集群的时候,NameNode询问DataNode得到的,并且间断的更新。

•Edits编辑日志:存放的是Hadoop文件系统的所有更改操作(文件创建,删除或修改),文件系统客户端执行的更改操作首先会被记录到Edits文件中。

•相同点:
• Fsimage和Edits文件都是经过序列化的,在NameNode启动时,它会将Fsimage文件中的内容加载到内存中,之后再执行Edits文件中的各项操作,使得内存中的元数据和实际的同步，存在内存中的元数据支持客户端的读操作，也是完整的元数据。

•不同点:
• 当客户端对HDFS中的文件进行新增或者修改操作,操作记录首先被记入Edits日志文件中,当客户端操作成功后,相应的元数据会更新到内存元数据中.因为Fsimage文件一般都很大(GB级别的很常见),如果所有的更新操作都往Fsimage文件中添加,这样会导致系统运行的十分缓慢。

• HDFS这种设计实现：一是内存中数据更新、查询快,极大缩短了操作响应时间;二是内存中元数据丢失风险颇高(断电等),因此辅佐元数据镜像文件(Fsimage)+编辑日志文件(Edits)的备份机制进行确保元数据的安全。

5. Flume的拦截器做了什么? (泰康保险)

(1) 问题分析

考官主要考核你对Flume拦截器的理解,因此需要讲出Flume拦截器的作用即可。

(2) 核心问题回答

拦截器的位置是在Source和Channel之间,当我们为Source指定拦截器后,在拦截器中会得到Event,从而根据需求对Event进行保留或舍弃操作,舍弃的数据不会进入Channel中。

6. HDFS如何保证数据安全性？ (中关村在线)

(1) 问题分析

HDFS作为Hadoop中很重要的组件，需要理解它的原理，这部分面试官想考察的是你对基础的理解。

(2) 核心问题回答

a）HDFS是典型的Master/Slave架构，它往往是一个NameNode加多个DataNode组成,NameNode是集群的；且HDFS中是分块存储的，为了容错文件的每个Block都会有副本。
b）HBase只支持一次写入多次读出的场景且不支持文件的随机修改。
c）第一个副本一般放置在与Client（客户端）所在的同一节点上（若客户端无DataNode，则随机放），第二个副本放置到与第一个副本同一机架的不同节点，第三个副本放到不同机架的DataNode节点，当取用时遵循就近原则；
d）DataNode已Block为单位，每3s报告心跳状态，做10min内不报告心跳状态则NameNode认为Block已死掉，NameNode会把其上面的数据备份到其他一个DataNode节点上，保证数据的副本数量；
e）Datanode会默认每小时把自己节点上的所有块状态信息报告给NameNode；
f）采用Safemode模式：DataNode会周期性的报告Block信息且HDFS元数据采用SecondaryName备份或者HA备份；

(3) 问题扩展

Hadoop三大组件MapReduce，HDFS，Yarn都要做到了解原理。HDFS读写流程，MapReduce的原理图也要做到理解。

(4) 结合项目使用

清楚哪些组件和HDFS结合时候用的，像Hive，HBase都有结合HDFS使用。

7. Hive跟HBase的区别是什么？（e代驾）

(1) 问题分析

考察对Hive,HBase大数据相关组件的了解程度，考察基础，可以从Hive，HBase概念延伸到区别然后结合实际做一个回答。

(2) 核心问题回答

Hive：是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张表，并提供类似于SQL查询用于解决海量结构化日志的数据统计(海量的结构化数据的运算分析)，本质为将HiveSQL转化成MapReduce程序或者Spark程序(分布式运算框架)
Hive的元数据默认存储在自带的derby数据库中，推荐使用MySQL,而原始数据一般储存在HDFS中.

- HBase：HBase是一个高可靠性、高性能(快)、面向列、可伸缩的分布式数据库系统，HBase支持单行事务,数据存储在HDFS中。适用于单表数据超千万,并发量高,数据分析需求弱,无需过于灵活和实时，与Hadoop一样，HBase目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加计算和存储能力。

- 二者区别:
- Hive和HBase是两种基于Hadoop的不同技术：Hive是一种类SQL的引擎，并且运行MapReduce任务，HBase是一种在Hadoop之上的NoSQL的Key/Value数据库，只支持简单的行列操作。当然，这两种工具是可以同时使用的。Hive可以用来进行统计查询，HBase可以用来进行快速的实时查询,数据也可以从Hive写到HBase,设置再从HBase写回Hive。

(3) 问题扩展

类似的数据库有很多，要清楚他们之间的区别，比如MySQL和HBase，Redis和HBase等等，要清楚他们的核心功能以及特性异同。

(4) 综合项目使用

知道在哪种场景下，使用哪种技术。当所存储数据需要快速插入查询时，使用HBase。
当数据需要大量聚合运算，计算分析结果时，使用hive存储，所以Hive是数仓，etl的常用工具。

8. 请说明Hive中SORT BY,ORDER BY,CLUSTER BY,DISTRIBUTE BY各代表什么意思？ (01财经)

(1) 问题分析

主要考察HiveSQL基础。在回答这几种是什么的情况下，再把各自的特点结合讲一下。

(2) 核心问题回答

1）SORT BY：会在每个reduce中排序,全局无序.但是分区内有序；
2）ORDER BY：对输入做全局排序,因此只有一个Reduce,当输入规模较大时,会消耗很大的时间
3）DISTRIBUTE BY：根据字段进行分区，类似MR中Partition，进行分区，结合sort by使用。
4）CLUSTER BY：当DISTRIBUTE BY和SORT 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。但是排序只能是升序排序，不能指定排序规则为ASC或者DESC。

(3) 问题扩展

Hive中常用分析函数：
Row_Number():从1开始，按照顺序，生成分组内记录的序列,Row_Number()的值不会存在重复,当排序的值相同时,按照表中记录的顺序进行排列;通常用于获取分组内排序第一的记录;获取一个Session中的第一条refer等。
Rank()：生成数据项在分组中的排名，排名相等会在名次中留下空位。
Dense_Rank():生成数据项在分组中的排名，排名相等会在名次中不会留下空位。

(4) 综合项目使用

HiveSQL在实际生产中，数仓这部分非常常用的一个工具，所以要重视起来，项目中常用的一些分析函数一定要记住3、5个。

中

1. Kafka零拷贝的原理?(京东)

(1) 问题分析

考官主要考核你对Kafka零拷贝的理解,因此需要讲出零拷贝的原理即可.

(2) 核心问题回答

零拷贝并不是不需要拷贝,而是减少不必要的拷贝次数,通常是说在IO读写过程中.
场景:读取文件,Socket发送
传统方式实现:先读取、再发送,实际经过1~4四次Copy.
a.第一次:将磁盘文件,读取到操作系统内核缓冲区;
b.第二次:将内核缓冲区的数据,Copy到Application应用程序的Buffer;
c.第三步:将Application应用程序Buffer中的数据,Copy到Socket网络发送缓冲区(属于操作系统内核的缓冲区);
d.第四次:将Socket Suffer的数据,Copy到网络协议栈,由网卡进行网络传输.

- 零拷贝实现:
- 零拷贝指的是Kafka将磁盘数据通过DMA(直接存储器访问)拷贝到内核态Buffer,直接通过DMA拷贝到NIC Buffer(Socket Buffer),无需CPU拷贝。
- 具体实现:数据传输通过TransportLayer来完成,其子类PlaintextTransportLayer通过Java NIO的FileChannel的TransferTo和TransferFrom方法实现零拷贝。

2. HiveSQL语句中SELECT FROM WHERE GROUP BY LIMIT ORDER BY的执行顺序 （京东）

(1) 问题分析

掌握sql的执行原理并进行优化是一项非常重要的技能，结合Hive SQL如何写能更加优化来进行回答。

(2) 核心问题讲解

```
SELECT (查询)                          --最后一步执行
FROM（进行表的查找和加载）       --第一步执行
WHERE(过滤)                            --第二步执行
GROUP BY(执行分组后的相关计算)     --第三步执行
ORDER BY(对于结果集进行排序)       --第四步执行
LIMIT(排序)                            --第五步执行
```

•这里再结合Hive中写sql时的一些提高效率节省资源的部分:

- a) 比如使用一个表的字段来约束另一个表时,用IN来代替JOIN，IN比JOIN快.
- b) 使用GROUP BY去重COUNT(1)统计来 代替 COUNT DISTINCT去重统计.
- c) 应尽量避免在 WHERE 子句中对字段进行 NULL 值判断;避免WHERE子句中使用!= <>操作符;
- d) 避免WHERE子句中使用OR连接条件,会导致引擎放弃索引而进行全表扫描.
- e) 用EXISTS代替 IN.
- f) 尽量不要使用UNION去掉重复的记录,而是使用UNION ALL再用GROUP BY去重.

(3) 问题扩展

知道了Hive的执行顺序后，我们来看几条可以提升Hive速度的方式。

- 原则一：尽量加上分区；
- 原则二：连接表时使用相同的关键词，这样只会产生一个Job；
- 原则三：减少每个阶段的数据量，有时候为了减少代码量，包含了多余的字段，导致速度会变慢一些，所以我们只选出需要的，在JOIN表前就进行过滤；
- 原则四：Map端聚合。hive.map.aggr=true; // 用于设定是否在 Map 端进行聚合，默认值为真；
- hive.groupby.mapaggr.checkinterval=100000; // 用于设定 Map 端进行聚合操作的条目数；

(4) 综合项目使用

在工作中经常使用Hive的sql来对数据进行一个处理，随着数据量的增大，掌握sql的执行原理并进行优化在实际生产环境中是非常必要的，可以有一个更高的视角来看自己写的。

3. Hive为什么要做分区,Hive的元数据存在哪?（新浪）

(1) 问题分析

考官主要考核你对Hive开发中分区和元数据存储的理解,因此需要讲出自己对他们的理解即可。

(2) 核心问题回答

分区：Hive在执行查询时,一般会扫描整个表的数据,由于表的数据量大,全表扫描消耗的时间长、效率低。而有时,查询只需要扫描表中的一部分数据即可,Hive引入了分区表的概念,将表的数据存储在不同的子目录中,每一个子目录对应一个分区,只查询部分分区数据时,可避免全表扫描,从而提高查询效率。

- 元数据存储：Hive将元数据存储在关系型数据库中（如MySQL、derby）,Hive的元数据包括数据库名、表名及类型、字段名称及数据类型、数据所存储的位置等。
- Hive的元数据默认存储在自带的derby数据库中,但是derby数据库是由java语言开发的,占用资源少、单进程、单用户,而我们一般均使用MySQL存储Hive元数据。

(3) 问题扩展

除了分区表和元数据的存储外,Hive还有分桶表以及内部表、外部表,面试官有可能会从分区的话题上引入到分桶、内部表、外部表的概念、以及分区表与分桶表、内部表与外部表的区别。

•分桶：当单个分区或者表的数据量过大时,分区不能更细粒度的划分数据,就需要使用分桶技术将数据划分成更细的粒度,将数据按照指定的字段进行分成多个桶,即将数据按照字段进行划分,数据按照字段划分到多个文件当中.
•分区和分桶的区别：
- a. 分桶是按照列的哈希值进行划分的,相对比较平均;而分区是按照列的值来进行划分的,容易造成数据倾斜.
- b. 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）.分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.
- c. 普通表（外部表、内部表）、分区表这三个都是对应HDFS上的目录,桶表对应是目录里的文件.

•Hive创建表时,可指定表的类型,表的类型有两种,分别为内部表和外部表,两者有如下区别：
- 创建表：默认情况下,我们创建的是内部表,若是想要创建外部表,则需要使用关键字EXTERNAL;
- 加载数据：加载数据到内部表,数据则会移动到自己的数据仓库目录下,而加载数据到外部表,数据并没有移动到自己的数据仓库目录下(如果指定了Location),也就是说外部表中的数据并不是由它自己来管理的;
- 删除表：删除内部表时,表的定义（元数据）和数据会被同时删除,而删除外部表时,仅仅删除表的定义,数据则被保留。

(4) 结合项目中使用

若是我们要收集某个大型网站的日志数据,一个网站每天的日志数据存储在同一张表上,并且每天会生成大量的日志,导致数据表的内容巨大,在查询时进行全表扫描耗费的资源非常多.那其实这个情况下,我们可以按照日期对表进行分区,不同日期的数据存放在不同的分区,在查询时只要指定分区字段的值就可以直接从该分区查找.

4. Yarn调度策略（顺丰）

(1) 问题分析

考官主要考核你对Yarn调度策略的理解,因此需要讲出Yarn调度策略的类型、概念及优缺点即可.

(2) 核心问题回答

Yarn调度策略主要有三种,分别为FIFO(先进先出调度器)、Capacity Scheduler(容量调度器)和Fair Scheduler(公平调度器).

•FIFO:按照任务到达的时间排序,先到先服务.
- 优点：运行一些较大的任务可以得到更多的资源;
- 缺点：由于任务的优先级只简单按照时间来排序,也就是说,如果这个较大的任务,需要执行2小时,后面的小任务只需执行1分钟就搞定,但是由于上一个任务未执行完,而第二个任务必须先等待2小时.

- Capacity Scheduler:允许多个组织共享整个集群,每个组织可以获得集群的一部分计算能力.通过为每个组织分配专门的队列,然后再为每个队列分配一定的集群资源,这样整个集群就可以通过设置多个队列的方式给多个组织提供服务了.除此之外,队列内部又可以垂直划分,这样一个组织内部的多个成员就可以共享这个队列资源了,在一个队列内部,资源的调度是采用FIFO策略。
- 优点:支持多队列共享集群资源.
- 缺点:若某队列将其他队列中资源给占用了,那么其他资源队列的资源将减少,此时其他队列的任务较多,就会找占用资源的队列归还资源,如果占用资源的队列未执行完,则会一直占用着,因此会存在阻塞等待的现象.

- Apache Hadoop默认使用的调度策略是Capacity Scheduler.
- Fair Scheduler:设计目标是为所有的作业分配公平的资源,也可以在多个队列间工作.
- 优点:支持多队列多用户,不需要预留一定量的资源,该调度器会在所有作业之间动态平衡资源,可以保证小作业能及时完成.
- 缺点:由于公平调度器支持抢占功能,而抢占功能本身虽然会减少作业执行的延迟时间，但会降低整个集群的效率，因为被抢占的作业会终止运行，最终会重新开始执行.
- CDH版本的Hadoop默认使用的调度策略是Fair Scheduler.

5. 描述MR中Shuffle的过程以及作用.(水滴互助)

(1) 问题分析

考官主要考核你对Shuffle机制的理解，因此只需要讲出Shuffle机制即可。

(2) 核心问题回答

Shuffle过程指的是MapTask的map方法之后，ReduceTask的Reduce方法之前的数据处理过程，Shuffle过程是MR中最关键的一个流程。

Shuffle过程包括Collect阶段、Spill阶段、两次Merge阶段、Copy阶段、Merge阶段以及Sort阶段，详解如下：

a. Collect阶段：将MapTask的结果输出到默认大小为100M的环形缓冲区，保存的是key/value，Partition分区信息等。

b. Spill阶段：当内存中的数据量达到一定的阀值的时候,就会将数据写入本地磁盘,在将数据写入磁盘之前需要对数据进行一次排序的操作,如果配置了Combiner,还会将有相同分区号和key的数据进行排序。

c. Merge阶段：把所有溢出的临时文件进行一次合并操作,以确保一个MapTask最终只产生一个中间数据文件。

d. Copy阶段：ReduceTask启动Fetcher线程到已完成MapTask的节点上复制一份属于自己的数据,并将数据写入本地磁盘,在将数据写入磁盘之前需要对数据进行一次排序的操作,如果配置了Combiner,还会将有相同分区号和key的数据进行排序。这些数据默认会保存在内存的缓冲区中，当内存的缓冲区达到一定的阀值的时候,就会将数据写到磁盘之上。

e. Merge阶段：在ReduceTask远程复制数据的同时，会在后台开启两个线程对内存到本地的数据文件进行合并操作。

f. Sort阶段：在对数据进行合并的同时，会进行排序操作，由于MapTask阶段已经对数据进行了局部的排序，ReduceTask只需保证Copy的数据的最终整体有效性即可。

注意：Shuffle中的缓冲区大小会影响到MR程序的执行效率，原则上说，缓冲区越大，磁盘IO的次数越少，执行速度就越快。缓冲区的大小可以通过参数调整，如参数：mapreduce.task.io.sort.mb 默认100M

6. HBase如何读取数据?(中国人寿)

(1) 问题分析

考官主要考核你对HBase读数据流程的理解，因此需要讲出HBase的读数据流程。

(2) 核心问题回答

由于我们通过Zookeeper部署了一个高可用的HBase集群，HRegionServer保存着Meta表以及表数据。

a. 若是想要读取数据，则首先通过Client从Zookeeper里找到Meta表所在的位置信息，即找到这个Meta表在哪个HRegionServer上保存着；

b. 然后通过获取到的HRegionServer IP地址来访问Meta表所在的HRegionServer，从而读取到Meta表，进而获取到Meta表中存放的元数据；

c. 接着通过元数据中存储的信息，访问对应的HRegionServer，并扫描所在HRegionServer的MetaStore和StoreFile来查询数据；

d. 最后HRegionServer将查询到的数据响应给Client。

(3) 问题扩展

除了读取数据外，面试官有可能会从读取数据的话题上引入到HBase写数据的流程。

a. Client先访问Zookeeper，找到Meta表，并获取Meta表元数据；

b. 确定当前将要写入的数据所对应的HRegion和HRegionServer服务器；

c. 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；

d. Client先把数据写入到HLog和MemStore各一份,以防止数据丢失；

e. 当MemStore达到阈值后，把数据flush到磁盘，生成StoreFile文件，当StoreFile越来越多，会触发Compact合并操作，减少StoreFile的数量，提高查询速度。

7. Kafka的偏移量Offset存放在哪儿,为什么? (瓜子二手车)

(1) 问题分析

考官主要考核你是否真的使用过Kafka，因此需要讲出Offset存放的位置以及为何选用该位置进行存储Offset。

(2) 核心问题回答

由于Zookeeper不适合大批量的频繁写入操作，所以Kafka 1.0.2将Consumer的Offset(位移信息)保存在Kafka内部的Topic中，即_consumer_offsets主题，并默认提供Kafka_consumer_groups.sh脚本供用户查看Consumer信息。

8. 简单描述HBase的Rowkey设计原则?(东方国信)

(1) 问题分析

考官主要考核你对Rowkey设计原则的了解，因此需要讲出Rowkey设计的原则即可。

(2) 核心问题回答

Rowkey设计需要遵循三个原则，即长度原则、散列原则、唯一原则。

a. 长度原则

由于Rowkey是一个二进制码流，可以是任意字符串，最大长度64kb，实际应用中一般为10-100bytes，以byte形式保存，一般设计成定长。 建议越短越好，不要超过16个字节，设计过长会降低MemStore内存的利用率和HFile存储数据的效率。

b. 散列原则

建议将Rowkey的高位作为散列字段，这样将提高数据均衡分布在每个RegionServer，以实现负载均衡。

c. 唯一原则

必须在设计上保证其唯一性。

(3) 问题扩展

除了Rowkey设计原则外，面试官有可能会从Rowkey设计原则上引入到HBase的热点问题，因此还需要回答出现热点的原因以及解决热点的方案。

"热点"现象指的是检索HBase的记录，首先要通过Rowkey来定位数据行，当大量的Client访问HBase集群的一个或少数几个节点，造成少数RegionServer的读/写请求过多、负载过大，而其他RegionServer负载却很小。

•解决方案：

- 预分区、加盐、哈希、反转。
- 预分区:预分区的目的让表的数据可以均衡的分散在集群中，而不是默认只有一个Region分布在集群的一个节点上。
- 加盐:这里所说的加盐不是密码学中的加盐，而是在Rowkey的前面增加随机数，具体就是给Rowkey分配一个随机前缀，从而使得它和之前的Rowkey的开头不同。
- 哈希:哈希会使同一行永远用一个前缀加盐，也可以使负载分散到整个集群，但是读却是可以预测的，使用确定的哈希可以让客户端重构完整的Rowkey，可以使用get操作准确获取某一个行数据。
- 反转:反转固定长度或者数字格式的Rowkey，这样可以使得Rowkey中经常改变的部分(最没有意义的部分)放在前面，这样可以有效的随机Rowkey，但是牺牲了Rowkey的有序性。

9. Redis缓存穿透、缓存雪崩、缓存击穿?(VIVO)

(1) 问题分析

考官主要考核你对Redis数据库的了解。

(2) 核心问题回答

a.缓存穿透是指查询一个一定不存在的数据。由于缓存命不中时会去查询数据库，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿透。

解决方案：

①是将空对象也缓存起来，并给它设置一个很短的过期时间，最长不超过5分钟

② 采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的BitMap中，一个一定不存在的数据会被这个BitMap拦截掉，从而避免了对底层存储系统的查询压力

b.如果缓存集中在一段时间内失效，发生大量的缓存穿透，所有的查询都落在数据库上，就会造成缓存雪崩。

解决方案：

尽量让失效的时间点不分布在同一个时间点

c.缓存击穿，是指一个Key非常热点，在不停的扛着大并发，当这个Key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。

解决方案：

可以设置Key永不过期

10. Zookeeper的选举机制,以及我们还可以用Zookeeper做些什么?(小米)

(1) 问题分析

考官主要考核你对Zookeeper选举机制的了解和Zookeeper的使用场景，因此需要阐述Zookeeper的选举机制和Zookeeper的使用场景。

(2) 核心问题回答

选举机制：

- a. 半数机制：集群中半数以上服务器存活，集群可用。一般推荐Zookeeper安装(2n+1)奇数台服务器。
- b. Zookeeper虽然在配置文件中没有指定Master和Slave，但是，Zookeeper工作时，是有一个节点为Leader，其它都为Follower，而Leader是通过内部的选举机制产生的。

- 假设，目前有一个Zookeeper集群，该集群是由五台服务器组成的，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，在存放数据量方面都是一样的。
- 依次启动这五台服务器,情况为:
- a. 服务器1启动,此时只有它一台服务?启动了,它发出的报?无任何响应,所以它的选举状态一直是LOOKING状态;
- b. 服务器2启动,它与最开始启动的服务器1进?通信,互相交换?己的选举结果,由于两者都无历史数据,所以ID值较大的服务器2胜出,但由于未达到超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5/2为3),所以服务器1、2还是继续保持LOOKING状态;
- c. 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,所以它成为这次选举出的Leader;
- d. 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?的,但是由于前面已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令,即Follower;
- e. 服务器5启动,同服务器4一样,被称为Follower.

- 集群非首次启动，每个节点在选举时都会参考自身节点的zxid值(事务ID),优先选择zxid值大的节点称为Leader.

•Zookeeper使用场景:

- 由于ZooKeeper是一个典型的发布/订阅模式的分布式数据管理与协调框架,我们可使?它来进行分布式数据的发布与订阅;
- 另一方面,通过对ZooKeeper中丰富的数据节点类型进行交叉使用,再配合Watcher事件通知机制,可以非常?便地构建一系列分布式应用中都会涉及的核心功能,例如,数据发布/订阅、命名服务、集群管理、Master选举、分布式锁和分布式队列等。

- 具体实践：监听服务器动态上下线和实现分布式锁(保证线程安全).

11. Flume丢不丢数据？（快手）

(1) 问题分析

考察Flume在实际生产环境的使用，尽量结合实际表述Flume是如何保证不丢数据的。

(2) 核心问题回答

Flume是一个数据采集工具；可以从各种各样的数据源（服务器）上采集数据传输（汇聚）到大数据生态的各种存储系统中（HDFS、HBase、Hive、Kafka）；
Flume不会丢数据，因为Channel存储可以存储在File中，数据传输自身有事务。
Flume组成：
Taildir Source：断点续传，多目录。
File Channel：数据存储在磁盘，宕机数据可以保存。但传输速率慢。适合对数据传输可靠性要求高的场景。
Memory Channel：数据存储在内存中，宕机数据丢失。传输速率快。适合对数据传输可靠性不高的场景。
Kafka Source：减少了Flume的Sink阶段，提高了传输效率。

(3) 问题扩展

要清楚Source，Channel，Sink这三个组件分别是负责哪个部分。
Source：数据源组件，用于跟数据源对接，以获取数据，他有各种各样的内置实现。
Sink:下沉组件（输出），用于往下一级Agent传递数据或向最终存储系统传递数据。
Channel：缓存通道组件，用于从Source将数据传递到Sink。

(4) 结合项目中使用

清楚Flume在实际生产中如何配置能够保持数据的安全性。

12. 列式存储和行级存储的区别？（言之有物）

(1) 问题分析

主要想考察面试者对数据库的理解。可以从几个方面做答：行列存储都有哪些数据库，概念以及优缺点。

(2) 核心问题回答

1） 传统的关系型数据库，如 Oracle、DB2、MySQL、SQL SERVER 等采用行式存储法(Row-based)，在基于行式存储的数据库中，数据是按照行数据为基础逻辑存储单元进行存储的，一行中的数据在存储介质中以连续存储形式存在。
2） 列式存储(Column-based)是相对于行式存储来说的，新兴的 HBase、HP Vertica、EMC Greenplum 等分布式数据库均采用列式存储。在基于列式存储的数据库中， 数据是按照列为基础逻辑存储单元进行存储的，一列中的数据在存储介质中以连续存储形式存在。
3） 行式存储的适用场景：
a．适合随机的增删改查操作；
b．需要在行中选取所有属性的查询操作；
c．需要频繁插入或更新的操作，其操作与索引和行的大小更为相关。
4） 列式存储引擎的适用场景：
a．查询过程中，可针对各列的运算并发执行(SMP)，最后在内存中聚合完整记录集，最大可能降低查询响应时间；
b．可在数据列中高效查找数据，无需维护索引(任何列都能作为索引)，查询过程中能够尽量减少无关IO，避免全表扫描；
c． 因为各列独立存储，且数据类型已知，可以针对该列的数据类型、数据量大小等因素动态选择压缩算法，以提高物理存储利用率;如果某一行的某一列没有数据，那在列存储时，就可以不存储该列的值，这将比行式存储更节省空间。

(3) 问题扩展

OLTP传统关系型数据库和OLAP分布式数据库的不同应用。
a．OLTP传统关系型数据库主要应用，用来执行一些基本的，日常的事务处理。可查询，插入，更新，删除。
b．OLAP分布式数据库它对实时性要求不高，但处理的数据量通常应用于复杂的动态报表系统上,查询为主。

(4) 结合项目使用

了解行列存储的优劣异同有利于我们在项目中根据实际的合适的场景去选择数据库。

难

1. 说一下Hive怎么优化？ （腾讯）

(1) 问题分析

这个考察对Hive的深入的理解，直接说明Hive如何优化就可以。

(2) 核心问题回答

a）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。

b）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。

c）采用分桶技术
d）采用分区技术
e）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；

f）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。

g）合理设置Reduce数

- 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；

h）常用参数

```
// 输出合并小文件
SET hive.merge.mapfiles = true; -- 默认true，在map-only任务结束时合并小文件
SET hive.merge.mapredfiles = true; -- 默认false，在map-reduce任务结束时合并小文件
SET hive.merge.size.per.task = 268435456; -- 默认256M
SET hive.merge.smallfiles.avgsize = 16777216; -- 当输出文件的平均大小小于该值时，启动一个独立的Map-Reduce任务进行文件Merge
```

(3) 问题扩展

Hive相关很重要，这块可以结合Hive sql如何写能使资源利用最大化去学习，比如Hive的函数如何合理利用。

(4) 结合项目使用

HiveSQL相关函数，如何调优在实际生产中很重要，要灵活使用。

2. Redis的热键问题？Redis的数据类型 （京东）

(1) 问题分析

本题面试官考察的是基础知识是否牢靠，这些小的知识点也不能忽视哦。

(2) 核心问题回答

热键问题：

1) 利用二级缓存，将热k放到JVM的缓存当中，将请求分散到多台机器上；

2) 备份热k,将热点k备份到多台Redis中，当热点k请求时，可以随机的从备份的选一台，进行访问取值，返回数据。

Redis中存储数据是通过key-value存储的，对于value的数据类型有五种：

字符串类型:Map<String,String>

Hash类型：Map<String,Map<String,String>>

List: Map<String,List> 有顺序，可重复。

Set: Map<String,HashSet> 无顺序，不能重复。

SortedSet(zset): Map<String,TreeSet> 有顺序，不能重复。

(3) 问题扩展

可以结合适用场景去对这五种数据类型给面试官进行拓展回答：
字符串类型适用场景：自增主键，商品编号，订单编号采用string的递增数字特性生成。
Hash类型适用场景：存储商品信息，商品字段，定义商品信息的key。
List类型适用场景：商品评论列表，在Redis中创建商品评论列表
SortedSet(zset)类型适用场景：例如根据商品销售量对商品进行排行显示。

(4) 结合项目中使用

要清楚Redis在什么场景下适用，和其他几种数据库相比较有哪些异同优劣。

3. Flume事务实现? (今日头条)

(1) 问题分析

考官主要考核你对Flume事务的理解,因此需要讲出Flume事务以及具体的实现.

(2) 核心问题回答

一提到事务,首先就想到的是关系型数据库中的事务,事务一个典型的特征就是将一批操作做成原子性的,要么都成功,要么都失败。在Flume中一共有两个事务,分别是Put事务和Take事务，其中Put事务处于Source到Channel之间,而Take事务处于Channel到Sink之间。Put事务时,数据在Flume中会被封装成Event对象,也就是一批Event,把这批Event放到一个事务中， 然后再把这个事务(一批Event)一次性的放入Channel中。同理,Take事务时,也是把这一批Event组成的事务统一拿出来Sink到HDFS上.

Put事务:

- 事务开始时,调用doPut方法,doPut方法将一批数据放在putList中;
- 数据顺利的放到putList后,接着调用docommit方法,检查channel内存队列是否有足够空间,若有,则将putList中所有的Event放到 Channel中,成功放完之后就清空putList;若空间不够,则调用doRollback方法回滚数据.

Take事务:

- 事务开始时,调用doTake方法,将Channel数据写入takeList和HDFS缓冲区(若Sink到HDFS,则写入HDFS);
- 若是数据全部发送成功,则清空takeList;若数据发送过程中失败,则清空临时缓冲区,将数据还给Channel.

4. 数据同样存在HDFS，为什么HBase支持在线查询？ (头条)

(1) 问题分析

考察对HBase的深入理解；概念和特性可做扩展回答，考察基础；

(2) 核心答案讲解

先了解一下，在线查询，即反应根据当前时间的数据，可以认为这些数据始终是在内存的，保证了数据的实时响应。可以认为是从内存中查询，一般响应时间在1秒内。所以可以从HBase的存储机制和底层架构和读取方式这三个方面来分析。

1）HBase的存储机制：

- 首先，HBase的机制是数据先写入到内存中，当数据量达到一定的量， 再写入磁盘中， 在内存中，是不进行数据的更新或合并操作的，只增加数据，这使得用户的写操作只要进入内存中就可以立即返回，保证了HBase I/O的高性能。
- 其次，在内存中的数据是有序的，如果内存空间满了，会刷写到HFile中，而在HFile中保存的内容也是有序的。HFile文件为磁盘顺序读取做了优化，按页存储。是顺序写入而不是随机写入，所以速度很稳定，这样保持稳定的同时，加快了速度。

2）HBase底层架构：

- HBase底层是LSM-Tree+ HTable(Region分区) + Cache--客户端可以直接定位到要查数据所在的HRegion Server服务器，然后直接在服务器的一个Region上查找要匹配的数据，并且这些数据部分是经过Cache缓存的。

3）HBase的读取

- 读取速度快是因为它使用了LSM树型结构。磁盘的顺序读取速度很快。HBase的存储结构导致它需要磁盘寻道时间在可预测范围内，而关系型数据库，即使有索引，也无法确定磁盘寻道次数。而且，HBase读取首先会在缓存中查找，它采用了LRU（最近最少使用算法），如果缓存中没找到，会从内存中的MemStore中查找，只有这两个地方都找不到时，才会加载HFile中的内容，而我们也提到读取HFile速度会很快，因为节省了寻道开销。

(3) 问题扩展

HBase的概念和特点可做扩展回答，再根据特点去回答优劣和核心功能，再延伸到适用场景。

1）HBase概念：

HBase是建立在HDFS之上,提供高可靠性的列存储，实时读写的数据库系统。它介于Nosql和关系型数据库之间，仅通过主键和主键的Range来检索数据，仅支持单行事务。主要用来存储非结构化和半结构化的松散数据。

2）HBase的优缺点：

- 优点：
- 高容错性，高扩展性。
- key/value存储方式面对海量数据也不会导致查询性能下降。相对于传统行式数据库，在单张表字段很多的时候，可以将相同的列存到不同的服务实例上，分散负载压力。
- 缺点：
- 架构设计复杂，且使用HDFS作为分布式存储，所以在存储少量数据时，它也不会很快。HBase不支持表关联操作，数据分析是HBase的弱项。HBase只部分支持ACID，只支持单行单次操作的事务。

(4) 综合项目中使用

清楚HBase和其他数据库的区别，清楚HBase优势劣势，在哪种场景下，使用哪种技术，当所存储数据需要快速插入查询时，使用HBase，所以Storm或Sparksteaming常常存储入HBase。当数据需要大量聚合运算，计算分析结果时，使用Hive存储，所以Hive是数仓，etl的常用工具。

5. Flume HDFS Sink小文件处理？（顺丰）

(1) 问题分析

这道题考察的是大数据相关组件的理解程度,可以结合HDFS Sink存入大量小文件会有什么影响去切入小文件如何处理这个点,由点及面的回答.

(2) 核心问题回答

1）HDFS存入大量小文件，有什么影响？

- 元数据层面：每个小文件都有一份元数据，其中包括文件路径，文件名，所有者，所属组，权限，创建时间等，这些信息都保存在Namenode内存中。所以小文件过多，会占用Namenode服务器大量内存，影响Namenode性能和使用寿命
- 计算层面：默认情况下MR会对每个小文件启用一个Map任务计算，非常影响计算性能。同时也影响磁盘寻址时间。

2）HDFS小文件处理

- 官方默认的这三个参数配置写入HDFS后会产生小文件，hdfs.rollInterval、hdfs.rollSize、hdfs.rollCount基于以上hdfs.rollInterval=3600，hdfs.rollSize=134217728，hdfs.rollCount =0，hdfs.roundValue=3600，hdfs.roundUnit= second几个参数综合作用，效果如下：
- a.tmp文件在达到128M时会滚动生成正式文件
- b.tmp文件创建超3600秒时会滚动生成正式文件
- 举例：在2018-01-01 05:23的时侯sink接收到数据，那会产生如下文件：/atguigu/20180101/atguigu.201801010520.tmp
- 即使文件内容没有达到128M，也会在06:23时滚动生成正式文件

(3) 问题扩展

Flume作为开发中的一个重要组件，我们要对它做一个自顶向下的了解，从他的底层架构，到细节部分Flume的优化，还有一些Flume相关，比如Flume的拦截器，Flume内存等等，这些面试都常涉及。

(4) 综合项目使用

Flume HDFS Sink 小文件过多会有很大影响，会非常影响性能，所以在实际生产中为了合理利用资源，我们需要对相关问题有一个清楚地了解。

6. Hive自定义哪些UDF函数?(瓜子二手车)

(1) 问题分析

考官主要考核你工作中是否使用过自定义的UDF函数,因此需要准备2个及以上的自定义UDF函数.

(2) 核心问题回答

例如:自定义转换小写函数、自定义解析公共字段函数、自定义测试函数是否创建成功的函数以及自定义将json变成List集合的函数。

```
/**
- 自定义转换小写函数
 */
    public class MyLowerUDF extends UDF {
        public Text evaluate(Text str){
            if(null == str.toString()){
                return null;
        }
        return new Text(str.toString().toLowerCase());
            }
    }
```

- /**

- 自定义udf函数用于解析公共字段

```
 */
public class BaseFieldUDF extends UDF {
    public String evaluate(String line, String jsonKeysString) {
        StringBuilder result = new StringBuilder();
        if(StringUtils.isBlank(line)) {
            return null;
        }
        String[] lineSplit = line.split("\\|");
        if(lineSplit.length != 2 || StringUtils.isBlank(lineSplit[1])) {
            return null;
        } else {
            String[] jsonKeys = jsonKeysString.split(",");
            try {
                JSONObject baseContent = new JSONObject(lineSplit[1]);
                JSONObject jsonObject = baseContent.getJSONObject("cm");
                for(String jsonKey : jsonKeys) {
                    if(jsonObject.has(jsonKey.trim())) {

result.append(jsonObject.getString(jsonKey.trim())).append("\t");
                    } else {
                        result.append("\t");
                    }
                }
                result.append(baseContent.getString("et")).append("\t");
                result.append(lineSplit[0]).append("\t");
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return result.toString();
    }
}

/**
- 自定义测试函数是否创建成功的函数
  */
  public class MyHiveUDF extends UDF{
  public Text evaluate(Text content) {
    return evaluate(content, new IntWritable(0));
  }
  public Text evaluate(Text content, IntWritable flag) {
    if (content==null)
        return null;

    if(flag.get()==1) {
        return new Text(content.toString().toUpperCase());
    }else {
        return new Text(content.toString().toLowerCase());
    }
  }
  }

/**

- 将json变成list集合,方便在hive中使用explode函数
  */
  public class JsonArray extends UDF{
    /**
```

```
      - 由于hive中的null和java中的null不一样,因此这里将参数类型设置为Object
      - hive中的null非String类型,若传入的参数为null,则直接报错,所以设置参数类型为Object
      - @param jsonStr  可以接收的参数格式  [{},{}]或{"key1":[{},{}],"key2":[{},
{}]}
      - @return    json数组转换后的list集合
        */
          public ArrayList<String> evaluate(Object jsonStr){
        //非String类型,返回null
        if(!(jsonStr instanceof String)) {
            return null;
        }
        String jsonString = (String) jsonStr;
        if(jsonString.startsWith("[")){
            return getJsonList(jsonString);
        }else if(jsonString.startsWith("{")){
            return getAllJsonList(jsonString);
        }else{
            return null;
        }
          }

        // 将一个json数组进行转换操作后,返回一个list集合
        // 例如:由"[{},{}]" ---> "list集合"
        // list集合在hive中相当于是数组,可以直接使用explode函数
        public ArrayList<String> getJsonList(String jsonArrayString){
            if(jsonArrayString.length() == 0 || jsonArrayString == null) return
null;
            JSONArray jsonArrayObj = null;
            try{
                jsonArrayObj = JSON.parseArray(jsonArrayString);
            }catch (Exception e){
                return null;
            }
            ArrayList<String> result = new ArrayList<String>();
            for(int i = 0 ; i<jsonArrayObj.size() ; i++){
                result.add(jsonArrayObj.get(i).toString());
            }
            return result;
        }

        //将一个json数组{"key":[{},{}],"key":[{},{}]}进行转换操作，返回list集合 [{},
{},{}]
        public ArrayList<String> getAllJsonList(String jsonString){
            JSONObject jsonObject = null;
            try{
                jsonObject = JSON.parseObject(jsonString);
            }catch (Exception e){
                return null;
            }
            //获取json对象中所有的key
            Set<String> keys = jsonObject.keySet();
            ArrayList<Object> jsonObjList = new ArrayList<Object>();
            //获取所有的json对象(格式上是一个json数组)
            for (String key : keys) {
                jsonObjList.add(jsonObject.get(key));
            }
            ArrayList<String> result = new ArrayList<String>();
```

```
            //将所有的对象合并到一个list集合中返回
            for (Object jsonObj : jsonObjList) {
                ArrayList<String> jsonList = getJsonList(jsonObj.toString());
                result.addAll(jsonList);
            }
            return result;
        }
    }
```

7. Kafka的分区分配策略（中信银行）

(1) 问题分析

主要考察大数据相关组件的了解程度，是否比较深入的了解Kafka的运行机制及原理，这里可以结合一个小例子去回答。

(2) 核心问题回答

在Kafka内部存在两种默认的分区分配策略：Range和RoundRobin。

Range是默认策略。

Range是对每个Topic而言的（即一个Topic一个Topic分），首先对同一个Topic里面的分区按照序号进行排序，并对消费者按照字母顺序进行排序。然后用Partitions分区的个数除以消费者线程的总数来决定每个消费者线程消费几个分区。如果除不尽，那么前面几个消费者线程将会多消费一个分区。

例如：我们有10个分区，两个消费者（C1，C2），3个消费者线程，10 / 3 = 3而且除不尽。

C1-0 将消费 0, 1, 2, 3 分区

C2-0 将消费 4, 5, 6 分区

C2-1 将消费 7, 8, 9 分区

RoundRobin策略:前提：同一个Consumer Group里面的所有消费者的num.streams（消费者消费线程数）必须相等；每个消费者订阅的主题必须相同。

首先将所有主题分区组成TopicAndPartition列表，然后对TopicAndPartition列表按照hashCode进行排序，最后按照轮询的方式发给每一个消费线程。

(3) 问题扩展

Kafka相关知识点比较多,这也是一个比较重要的组件,尽量从多个方面去学习,比如Kafka压测，Kafka监控，Kafka副本数设定，它丢不丢数据消息积压消费能力不足怎么办等等。

(4) 综合项目使用

Kafka可以将主题划分为多个分区，会根据分区规则选择把消息存储到哪个分区中，如果分区规则设置的合理，那么所有的消息将会被均匀的分布到不同的分区中，这样就实现了。

8. 谈谈数据倾斜,如何发生的,并给出优化方案？ （e代驾）

(1) 问题分析

考察实操，在数据倾斜是什么如何产生的基础上再去分析如何优化。

(2) 核心问题回答

数据倾斜产生的原因：

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

数据倾斜的解决：

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？
不是的。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，会造成很大的资源浪费。而且同时可执行的Map数是有限的。
（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

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.sql.io.CombineHiveInputFormat
```

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。

4. 合理设置Reduce数:(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。

5. 并行执行
Hive会将一个查询转化成一个或者多个阶段,而默认情况下，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,所以如果有更多的阶段可以并行执行，那么job可能就越快完成。

6. 开启严格模式(开启后会禁止三种类型的查询)
Hive提供了一个严格模式，可以防止用户执行那些可能意想不到的不好的影响的查询,通过设置属性hive.mapred.mode值默认是非严格模式Nonstrict ，开启严格模式需要修改hive.mapred.mode值为strict
（1）对于分区表，除非WHERE语句中含有分区字段的过滤条件来限制范围，否则不允许执行也就是避免扫描所有分区,因为每个分区都可能拥有非常大数据集。
（2）对于使用了ORDER BY语句的查询，要求必须使用Limit语句
因为一旦使用ORDER BY ,所有的结果数据会分发到同一个Reducer中进行处理,从而会使Reducer额外执行一段很长的时间。
（3）限制笛卡尔积的查询

7. JVM重用

由于Hadoopde默认配置通常是使用派生JVM来执行Map和Reduce任务,因此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,而N的值需要根据具体业务场景测试得出。JVM重用的缺点，开启JVM重用将一直占用使用到的task插槽，以便进行重用，直到任务完成后才能释放,一旦由于某个job执行时间过长的话,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。

8. 推测执行
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。

(3) 问题扩展

Hive Mr，Spark都会产生数据倾斜，在了解数据倾斜是什么的基础上去对具体的数据倾斜比如Spark的基础上去学习（目前我们还没有学习到Spark，后边的有Spark的课程，这里先做一个简单的了解。）

(4) 结合项目使用

数据倾斜是实际生产中比较常见的一个部分，所以掌握如何避免如何处理数据倾斜很重要。

9. Hadoop优化时经常修改的配置文件和其中配置项？ （搜狐）

(1) 问题分析

这个主要看你对Hadoop框架的把控，在此基础上去分析Hadoop常见的几个可以优化的配置文件。

(2) 核心问题回答

1）在hdfs-site.xml文件中配置多目录，最好提前配置好，否则更改目录需要重新启动集群
2）NameNode有一个工作线程池，用来处理不同DataNode的并发心跳以及客户端并发的元数据操作。
dfs.namenode.handler.count=20 * log2(Cluster Size)，比如集群规模为10台时，此参数设置为60
3）编辑日志存储路径dfs.namenode.edits.dir设置与镜像文件存储路径dfs.namenode.name.dir尽量分开，达到最低写入延迟
4）服务器节点上YARN可使用的物理内存总量，默认是8192（MB），注意，如果你的节点内存资源不够8GB，则需要调减小这个值，而YARN不会智能的探测节点的物理内存总量。
yarn.nodemanager.resource.memory-mb
5）单个任务可申请的最多物理内存量，默认是8192（MB）。yarn.scheduler.maximum-allocation-mb

(3) 问题扩展

Hadoop作为大数据很重要的一部分，自然也是面试中比较常见的问题，他的三大组件要做到了然于胸，还有一些小的知识点，比如Hadoop宕机怎么处理，Hadoop常用端口号等等这些都要在平时多做积累。

(4) 综合项目使用

当发现作业运行效率不理想时，需要对作业执行进行性能监测，以及对作业本身、集群平台进行优化。优化后的集群可能最大化利用硬件资源，从而提高作业的执行效率。这些在Hadoop集群平台搭建以及作业运行过程中一些常用优化手段需要多做了解。

10. MySQL的索引如何理解？常用引擎是什么？有什么区别？比较Redis和MySQL的区别？说一下各自的持久化机制 （搜狗金融）

(1) 问题分析

考察对数据库的深入理解，把问题的每个部分拆解开回答即可。

(2) 核心问题回答

1) MySQL索引的理解：

索引(Index)是帮助MySQL高效获取数据的数据结构。我们可以简单理解为：快速查找排好序的一种数据结构。Mysql索引主要有两种结构：B+Tree索引和Hash索引。

索引虽然能非常高效的提高查询速度，同时却会降低更新表的速度。实际上索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引列也是要占用空间的。所以要看实际情况是否建立索引。

2) 常用引擎是什么：

a.常用引擎有三种：InnoBD，Myisam，Memory

b.区别：

InnoDB：支持事务，支持外键，支持行锁，写入数据时操作快。

Myisam:不支持事务。不支持外键，支持表锁，支持全文索引，读取数据快。

Memory：所有的数据都保留在内存中，不需要进行磁盘的IO所以读取的速度很快，一旦关机的话表的结构会保留，但是数据会丢失，表支持Hash索引，因此查找速度很快。

3）Redis和MySQL的区别：

a. MySQL和Redis的数据库类型

MySQL是关系型数据库，主要用于存放持久化数据，将数据存储在硬盘中，读取速度较慢。

Redis是NOSQL，即非关系型数据库，也是缓存数据库，即将数据存储在缓存中，缓存的读取速度快，能够大大的提高运行效率，但是保存时间有限。

b. MySQL的运行机制

MySQL作为持久化存储的关系型数据库，相对薄弱的地方在于每次请求访问数据库时，都存在着I/O操作，如果反复频繁的访问数据库。第一：会在反复链接数据库上花费大量时间，从而导致运行效率过慢；第二：反复的访问数据库也会导致数据库的负载过高，那么此时缓存的概念就衍生了出来。

c. 缓存

缓存就是数据交换的缓冲区（Cache），当浏览器执行请求时，首先会对在缓存中进行查找，如果存在，就获取；否则就访问数据库。缓存的好处就是读取速度快。

d. Redis数据库

Redis数据库就是一款缓存数据库，用于存储使用频繁的数据，这样减少访问数据库的次数，提高运行效率。

e. Redis和MySQL的区别总结

（1）类型上

从类型上来说，MySQL是关系型数据库，Redis是缓存数据库

（2）作用上

MySQL用于持久化的存储数据到硬盘，功能强大，但是速度较慢

Redis用于存储使用较为频繁的数据到缓存中，读取速度快

（3）需求上

MySQL和Redis因为需求的不同，一般都是配合使用。

（4）Redis和MySQL各自的持久化机制：

- a）Redis的持久化机制：Redis提供两种持久化方式，RDB和AOF；
- AOF可以完整的记录整个数据库,在AOF模式下，Redis会把执行过的每一条更新命令记录录下来，保存到AOF文件中；而RDB记录的只是数据库某一时刻的快照。
- 两种机制的区别：是一个是持续的用日志记录写操作，Crash后利用日志恢复；一个是平时写操作的时候不触发写，只有手动提交save命令，或关闭命令的时候，才触发备份操作。

- b）MySQL的持久化机制：InnoDB引擎会引入redo日志作为中间层来保证MySQL的持久化，当有一条记录更新的时候，InnoDB引擎就会先把记录写到redolog中，并更新内存。同时InnoDB引擎会在适当的时候，将这个操作记录更新到磁盘里面。

(3) 问题扩展

MySQL索引存在的好处，以及和Redis各自适用的场景。

(4) 综合项目中使用

> 几种数据库的特性要理解清楚，这样对项目选型会有更深刻的了解。