1、kafka 的 message 包括哪些信息

一个 Kafka 的 Message 由一个固定长度的 header 和一个变长的消息体 body 组成

header 部分由一个字节的 magic(文件格式)和四个字节的 CRC32(用于判断 body 消息体是否正常)构成。当 magic 的值为 1 的时候，会在 magic 和 crc32 之间多一个字节的数据：attributes(保存一些相关属性，比如是否压缩、压缩格式等等)；如果 magic 的值为 0，那么不存在 attributes 属性

body 是由 N 个字节构成的一个消息体，包含了具体的 key/value 消息

2、怎么查看 kafka 的 offset

0.9 版本以上，可以用最新的 Consumer client 客户端，有 consumer.seekToEnd() / consumer.position() 可以用于得到当前最新的 offset：

3、hadoop 的 shuffle 过程

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。

最后就是 Reduce 过程了，在这个过程中产生了最终的输出结果，并将其写到 HDFS 上。

4、spark 集群运算的模式

Spark 有很多种模式，最简单就是单机本地模式，还有单机伪分布式模式，复杂的则运行在集群中，目前能很好的运行在 Yarn 和 Mesos 中，当然 Spark 还有自带的 Standalo

ne 模式，对于大多数情况 Standalone 模式就足够了，如果企业已经有 Yarn 或者 Mesos 环境，也是很方便部署的。

standalone(集群模式) :典型的 Mater/slave 模式 ,不过也能看出 Master 是有单点故障的；Spark 支持 ZooKeeper 来实现 HA

on yarn(集群模式)： 运行在 yarn 资源管理器框架之上，由 yarn 负责资源管理，Spark 负责任务调度和计算

on mesos(集群模式)： 运行在 mesos 资源管理器框架之上，由 mesos 负责资源管理，Spark 负责任务调度和计算

on cloud(集群模式)：比如 AWS 的 EC2，使用这个模式能很方便的访问 Amazon 的 S3;Spark 支持多种分布式存储系统：HDFS 和 S3

5、HDFS 读写数据的过程

读：

1、跟 namenode 通信查询元数据，找到文件块所在的 datanode 服务器

2、挑选一台 datanode（就近原则，然后随机）服务器，请求建立 socket 流

3、datanode 开始发送数据（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，现在本地缓存，然后写入目标文件

写：

1、根 namenode 通信请求上传文件，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、client 请求第一个 block 该传输到哪些 datanode 服务器上

4、namenode 返回 3 个 datanode 服务器 ABC

5、client 请求 3 台 dn 中的一台 A 上传数据（本质上是一个 RPC 调用，建立 pipeline），A 收到请求会继续调用 B，然后 B 调用 C，将真个 pipeline 建立完成，逐级返回客户端

6、client 开始往 A 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，A 收到一个 packet 就会传给 B，B 传给 C；A 每传一个 packet 会放入一个应答队列等待应答

7、当一个 block 传输完成之后，client 再次请求 namenode 上传第二个 block 的服务器。

6、RDD 中 reduceBykey 与 groupByKey 哪个性能好，为什么

reduceByKey：reduceByKey 会在结果发送至 reducer 之前会对每个 mapper 在本地进行 merge，有点类似于在 MapReduce 中的 combiner。这样做的好处在于，在 map 端进行一次 reduce 之后，数据量会大幅度减小，从而减小传输，保证 reduce 端能够更快的进行结果计算。

groupByKey：groupByKey 会对每一个 RDD 中的 value 值进行聚合形成一个序列(Iterator)，此操作发生在 reduce 端，所以势必会将所有的数据通过网络进行传输，造成不必要的浪费。同时如果数据量十分大，可能还会造成 OutOfMemoryError。

通过以上对比可以发现在进行大量数据的 reduce 操作时候建议使用 reduceByKey。不仅可以提高速度，还是可以防止使用 groupByKey 造成的内存溢出问题。

7、spark2.0 的了解

更简单：ANSI SQL 与更合理的 API

速度更快：用 Spark 作为编译器

更智能：Structured Streaming

8、 rdd 怎么分区宽依赖和窄依赖

宽依赖：父 RDD 的分区被子 RDD 的多个分区使用　例如 groupByKey、reduceByKey、sortByKey 等操作会产生宽依赖，会产生 shuffle

窄依赖：父 RDD 的每个分区都只被子 RDD 的一个分区使用　例如 map、filter、union 等操作会产生窄依赖

9、spark streaming 读取 kafka 数据的两种方式

这两种方式分别是：

Receiver-base

使用 Kafka 的高层次 Consumer API 来实现。receiver 从 Kafka 中获取的数据都存储在 Spark Executor 的内存中，然后 Spark Streaming 启动的 job 会去处理那些数据。然而，在默认的配置下，这种方式可能会因为底层的失败而丢失数据。如果要启用高可靠机制，让数据零丢失，就必须启用 Spark Streaming 的预写日志机制（Write Ahead Log，WAL）。该机制会同步地将接收到的 Kafka 数据写入分布式文件系统(比如 HDFS)上的预写日志中。所以，即使底层节点出现了失败，也可以使用预写日志中的数据进行恢复。

Direct

Spark1.3 中引入 Direct 方式，用来替代掉使用 Receiver 接收数据，这种方式会周期性地查询 Kafka，获得每个 topic+partition 的最新的 offset，从而定义每个 batch 的 offset

的范围。当处理数据的 job 启动时，就会使用 Kafka 的简单 consumer api 来获取 Kafka 指定 offset 范围的数据。

10、kafka 的数据存在内存还是磁盘

Kafka 最核心的思想是使用磁盘，而不是使用内存，可能所有人都会认为，内存的速度一定比磁盘快，我也不例外。在看了 Kafka 的设计思想，查阅了相应资料再加上自己的测试后，发现磁盘的顺序读写速度和内存持平。

而且 Linux 对于磁盘的读写优化也比较多，包括 read-ahead 和 write-behind，磁盘缓存等。如果在内存做这些操作的时候，一个是 JAVA 对象的内存开销很大，另一个是随着堆内存数据的增多，JAVA 的 GC 时间会变得很长，使用磁盘操作有以下几个好处：

磁盘缓存由 Linux 系统维护，减少了程序员的不少工作。

磁盘顺序读写速度超过内存随机读写。

JVM 的 GC 效率低，内存占用大。使用磁盘可以避免这一问题。

系统冷启动后，磁盘缓存依然可用。

11、怎么解决 kafka 的数据丢失

producer 端：

宏观上看保证数据的可靠安全性，肯定是依据分区数做好数据备份，设立副本数。

broker 端：

topic 设置多分区，分区自适应所在机器，为了让各分区均匀分布在所在的 broker 中，分区数要大于 broker 数。

分区是 kafka 进行并行读写的单位，是提升 kafka 速度的关键。

Consumer 端

consumer 端丢失消息的情形比较简单：如果在消息处理完成前就提交了 offset，那么就有

可能造成数据的丢失。由于 Kafka consumer 默认是自动提交位移的，所以在后台提交位移前一定要保证消息被正常处理了，因此不建议采用很重的处理逻辑，如果处理耗时很长，则建议把逻辑放到另一个线程中去做。为了避免数据丢失，现给出两点建议：

enable.auto.commit=false 关闭自动提交位移

在消息被完整处理之后再手动提交位移

12、fsimage 和 edit 的区别？

大家都知道 namenode 与 secondary namenode 的关系，当他们要进行数据同步时叫做 checkpoint 时就用到了 fsimage 与 edit，fsimage 是保存最新的元数据的信息，当 fsimage 数据到一定的大小事会去生成一个新的文件来保存元数据的信息 ,这个新的文件就是 edit，edit 会回滚最新的数据。

13、列举几个配置文件优化？

1）Core-site.xml 文件的优化

a、fs.trash.interval，默认值： 0；说明： 这个是开启 hdfs 文件删除自动转移到垃圾箱的选项，值为垃圾箱文件清除时间。一般开启这个会比较好，以防错误删除重要文件。单位是分钟。

b、dfs.namenode.handler.count，默认值：10；说明：hadoop 系统里启动的任务线程数，这里改为 40，同样可以尝试该值大小对效率的影响变化进行最合适的值的设定。

c、mapreduce.tasktracker.http.threads，默认值：40；说明：map 和 reduce 是通过 http 进行数据传输的，这个是设置传输的并行线程数。

14、datanode 首次加入 cluster 的时候，如果 log 报告不兼容文件版本，那需要 namenode 执行格式化操作，这样处理的原因是？

1）这样处理是不合理的，因为那么 namenode 格式化操作，是对文件系统进行格式化，namenode 格式化时清空 dfs/name 下空两个目录下的所有文件，之后，会在目录 dfs.name.dir 下创建文件。

2）文本不兼容，有可能时 namenode 与 datanode 的 数据里的 namespaceID、clusterID 不一致，找到两个 ID 位置，修改为一样即可解决。

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

1）一个 MapReduce 作业由 Map 阶段和 Reduce 阶段两部分组成，这两阶段会对数据排序，从这个意义上说，MapReduce 框架本质就是一个 Distributed Sort。

2）在 Map 阶段，Map Task 会在本地磁盘输出一个按照 key 排序（采用的是快速排序）的文件（中间可能产生多个文件，但最终会合并成一个），在 Reduce 阶段，每个 Reduce Task 会对收到的数据排序，这样，数据便按照 Key 分成了若干组，之后以组为单位交给 reduce（）处理。

3）很多人的误解在 Map 阶段，如果不使用 Combiner 便不会排序，这是错误的，不管你用不用 Combiner，Map Task 均会对产生的数据排序（如果没有 Reduce Task，则不会排序，实际上 Map 阶段的排序就是为了减轻 Reduce 端排序负载）。

4）由于这些排序是 MapReduce 自动完成的，用户无法控制，因此，在 hadoop 1.x 中无法避免，也不可以关闭，但 hadoop2.x 是可以关闭的。

16、hadoop 的优化？

1）优化的思路可以从配置文件和系统以及代码的设计思路来优化

2）配置文件的优化：调节适当的参数，在调参数时要进行测试

3）代码的优化：combiner 的个数尽量与 reduce 的个数相同，数据的类型保持一致，可以减少拆包与封包的进度

4）系统的优化：可以设置 linux 系统打开最大的文件数预计网络的带宽 MTU 的配置

5）为 job 添加一个 Combiner，可以大大的减少 shuffer 阶段的 maoTask 拷贝过来给远程的 reduce task 的数据量，一般而言 combiner 与 reduce 相同。

6）在开发中尽量使用 stringBuffer 而不是 string，string 的模式是 read-only 的，如果对它进行修改，会产生临时的对象，二 stringBuffer 是可修改的，不会产生临时对象。

7）修改一下配置：以下是修改 mapred-site.xml 文件

a、修改最大槽位数：槽位数是在各个 tasktracker 上的 mapred-site.xml 上设置的，默认都是 2

```
<property>
<name>mapred.tasktracker.map.tasks.maximum</name>
<value>2</value>
</property>
<property>
<name>mapred.tasktracker.reduce.tasks.maximum</name>
<value>2</value>
</property>
```

b、调整心跳间隔：集群规模小于 300 时，心跳间隔为 300 毫秒

mapreduce.jobtracker.heartbeat.interval.min 心跳时间

mapred.heartbeats.in.second 集群每增加多少节点，时间增加下面的值

mapreduce.jobtracker.heartbeat.scaling.factor 集群每增加上面的个数，心跳增多少

c、启动带外心跳

mapreduce.tasktracker.outofband.heartbeat 默认是 false

d、配置多块磁盘

mapreduce.local.dir

e、配置 RPC hander 数目

mapred.job.tracker.handler.count 默认是 10，可以改成 50，根据机器的能力

f、配置 HTTP 线程数目

tasktracker.http.threads 默认是 40，可以改成 100 根据机器的能力

g、选择合适的压缩方式，以 snappy 为例：

<property>

<name>mapred.compress.map.output</name>

<value>true</value>

</property>

<property>

<name>mapred.map.output.compression.codec</name>

<value>org.apache.hadoop.io.compress.SnappyCodec</value>

</property>

17、设计题

1）采集 nginx 产生的日志，日志的格式为 user ip time url htmlId 每天产生的文件的数据量上亿条，请设计方案把数据保存到 HDFS 上，并提供一下实时查询的功能（响应时间小于 3s）

A、某个用户某天访问某个 URL 的次数

B、某个 URL 某天被访问的总次数

实时思路是：使用 Logstash + Kafka + Spark-streaming + Redis + 报表展示平台

离线的思路是：Logstash + Kafka + Elasticsearch + Spark-streaming + 关系型数据库

A、B、数据在进入到 Spark-streaming 中进行过滤，把符合要求的数据保存到 Redis 中

18、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。 还是典型的 TOP K 算法，

解决方案如下：

1）方案 1：

顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。 找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为）。对这 10 个文件进行归并排序（内排序与外排序相结合）。

2）方案 2：

一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，

一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

3）方案 3：

与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

19、在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

1）方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 2^32 * 2 bit=1 GB 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

2）方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

20、腾讯面试题：给 40 亿个不重复的 unsigned int 的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 40 亿个数当中？

1）方案 1：oo，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入 40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不存在。

2）方案 2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下： 又因为 2^32 为 40 亿多，所以给定一个数可能在，也可能不在其中； 这里我们

把 40 亿个数中的每一个用 32 位的二进制来表示 ,假设这 40 亿个数开始放在一个文件中。 然后将这 40 亿个数分成两类:

1.最高位为 0

2.最高位为 1

并将这两类分别写入到两个文件中 ,其中一个文件中数的个数<=20 亿 ,而另一个>=20 亿（这相当于折半了）； 与要查找的数的最高位比较并接着进入相应的文件再查找 再然后把这个文件为又分成两类:

1.次最高位为 0

2.次最高位为 1

并将这两类分别写入到两个文件中 ,其中一个文件中数的个数<=10 亿 ,而另一个>=10 亿（这相当于折半了）； 与要查找的数的次最高位比较并接着进入相应的文件再查找。

.....

以此类推，就可以找到了,而且时间复杂度为 O(logn)，方案 2 完。

3)附：这里，再简单介绍下，位图方法： 使用位图法判断整形数组是否存在重复 ,判断集合中存在重复是常见编程任务之一 ,当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 max 创建一个长度为 max+1 的新数组 ,然后再次扫描原数组 ,遇到几就给新数组的第几位置上 1 ,如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这 种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。它的运算次数最坏的情况为 2N。如果已知数组的最大值即能事先给新数组定长的话效 率还能提高一倍。

21、怎么在海量数据中找出重复次数最多的一个？

1）方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

22、上千万或上亿数据（有重复），统计其中出现次数最多的钱 N 个数据。

1）方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

23、一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，给出思想，给出时间复杂度分析。

1）方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度）。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 O(n*lg10)。所以总的时间复杂度，是 O(n*le)与 O(n*lg10)中较大的哪一 个。

24、100w 个数中找出最大的 100 个数。

1）方案 1：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

2）方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的

一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

3）方案 3：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的 要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，直到扫描了所有的元素。复杂度为 O(100w*100)。

25、有一千万条短信，有重复，以文本文件的形式保存，一行一条，有重复。 请用 5 分钟时间，找出重复出现最多的前 10 条。

1）分析： 常规方法是先排序，在遍历一次，找出重复最多的前 10 条。但是排序的算法复杂度最低为 nlgn。

2）可以设计一个 hash_table, hash_map<string, int>，依次读取一千万条短信，加载到 hash_table 表中，并且统计重复的次数，与此同时维护一张最多 10 条的短信表。 这样遍历一次就能找出最多的前 10 条，算法复杂度为 O(n)。