

1. 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

2. 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数据到一定的大小时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。

3. 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。

4. 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。

5. 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。

6. 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mapTask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

mapred.tasktracker.map.tasks.maximum 2 mapred.tasktracker.reduce.tasks.maximum 2 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 handler 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： mapred.compress.map.output true mapred.map.output.compression.codec org.apache.hadoop.io.compress.SnappyCodec

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Spark-streaming 中进行过滤，把符合要求的数据保存到Redis中

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 $\text{hash}(\text{query}) \% 10$ 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）。找一台内存存在 2G 左右的机器，依次对用 $\text{hash_map}(\text{query}, \text{query_count})$ 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_count 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。

9. 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 $2^{32} * 2 \text{ bit} = 1 \text{ GB}$ 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。扫描完后，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。

10. 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 $O(\log n)$ ，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。

11. 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）。

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。

13. 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 $O(n \cdot l_e)$ (l_e 表示单词的平均长度)。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 $O(n \cdot \lg 10)$ 。所以总的时间复杂度，是 $O(n \cdot l_e)$ 与 $O(n \cdot \lg 10)$ 中较大的哪一个。

14. 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 $O(100w \cdot \lg 100)$ 。
- 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 $O(100w \cdot 100)$ 。
- 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 $O(100w \cdot 100)$ 。

15. 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 $n \lg n$ 。
- 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 $O(n)$ 。

注：资料来源于网络。