**海量数据处理面试题**

海量数据处理，就是在海量数据上的存储、处理、操作。海量的意思就是数据量太大，所以导致要么是无法在较短时间内迅速解决，要么是数据太大，导致无法一次性装入内存。

解决办法：针对时间，我们可以采用巧妙的算法搭配合适的数据结构，如Bloom filter/Hash/bit-map/堆/数据库或倒排索引/trie树；针对空间，可以大而化小，分而治之（hash映射），规模太大的就把规模大化为规模小的，各个击破不就完了嘛。

单机及集群问题，单机就是处理装载数据的机器有限(只要考虑cpu，内存，硬盘的数据交互)，而集群，机器有多辆，适合分布式处理，并行计算(更多考虑节点和节点间的数据交互)。

处理海量数据问题的办法就是分而治之/hash映射 + hash统计 + 堆/快速/归并排序；双层桶划分；Bloom filter/Bitmap；Trie树/数据库/倒排索引；外排序；分布式处理之Hadoop/Mapreduce。

STL容器分两种，序列式容器(vector/list/deque/stack/queue/heap)和关联式容器。关联式容器又分为set(集合)和map(映射表)两大类，以及这两大类的衍生体multiset(多键集合)和multimap(多键映射表)，这些容器均以RB-tree完成。此外，还有第3类关联式容器，如hashtable(散列表)，以及以hashtable为底层机制完成的hash\_set(散列集合)/hash\_map(散列映射表)/hash\_multiset(散列多键集合)/hash\_multimap(散列多键映射表)。也就是说，set/map/multiset/multimap都内含一个RB-tree，而hash\_set/hash\_map/hash\_multiset/hash\_multimap都内含一个hashtable。

什么样的结构决定其什么样的性质，因为set/map/multiset/multimap都是基于RB-tree之上，所以有自动排序功能，而hash\_set/hash\_map/hash\_multiset/hash\_multimap都是基于hashtable之上，所以不含有自动排序功能，至于加个前缀multi\_无非就是允许键值重复而已。

**处理海量数据问题之六把密匙**

**密匙一：分而治之/Hash映射 + Hash\_map统计 + 堆/快速/归并排序**

1、海量日志数据，提取出某日访问百度次数最多的那个IP。海量数据处理，说白了，就是先映射，而后统计，最后排序：

分而治之/hash映射：针对数据太大，内存受限，只能是：把大文件化成(取模映射)小文件，即16字方针：大而化小，各个击破，缩小规模，逐个解决

hash\_map统计：当大文件转化了小文件，那么我们便可以采用常规的hash\_map(ip，value)来进行频率统计。

堆/快速排序：统计完了之后，便进行排序(可采取堆排序)，得到次数最多的IP。

具体而论，则是： “首先是这一天，并且是访问百度的日志中的IP取出来，逐个写入到一个大文件中。注意到IP是32位的，最多有个2^32个IP。同样可以采用映射的方法，比如%1000，把整个大文件映射为1000个小文件，再找出每个小文中出现频率最大的IP（可以采用hash\_map对那1000个文件中的所有IP进行频率统计，然后依次找出各个文件中频率最大的那个IP）及相应的频率。然后再在这1000个最大的IP中，找出那个频率最大的IP，即为所求。”关于本题，还有几个问题，如下：

1、Hash取模是一种等价映射，不会存在同一个元素分散到不同小文件中的情况，即这里采用的是mod1000算法，那么相同的IP在hash取模后，只可能落在同一个文件中，不可能被分散的。因为如果两个IP相等，那么经过Hash(IP)之后的哈希值是相同的，将此哈希值取模（如模1000），必定仍然相等。

2、那到底什么是hash映射呢？简单来说，就是为了便于计算机在有限的内存中处理big数据，从而通过一种映射散列的方式让数据均匀分布在对应的内存位置(如 大数据 通过取余的方式映射成小树存放在内存中，或大文件映射成多个小文件)，而这个映射散列方式便是我们通常所说的hash函数，设计的好的hash函数能让数据均匀分布而减少冲突。尽管数据映射到了另外一些不同的位置，但数据还是原来的数据，只是代替和表示这些原始数据的形式发生了变化而已。

2、寻找热门查询，300万个查询字符串中统计最热门的10个查询

原题： 搜索引擎 会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为1-255字节。假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是1千万，但如果除去重复后，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门），请你统计最热门的10个查询串，要求使用的内存不能超过1G。

解答：由上面第1题，我们知道，数据大则划为小的，如如一亿个Ip求Top 10，可先%1000将ip分到1000个小文件中去，并保证一种ip只出现在一个文件中，再对每个小文件中的ip进行hashmap计数统计并按数量排序，最后归并或者最小堆依次处理每个小文件的top10以得到最后的结。

但如果数据规模比较小，能一次性装入内存呢?比如这第2题，虽然有一千万个Query，但是由于重复度比较高，因此事实上只有300万的Query，每个Query255Byte，因此我们可以考虑把他们都放进内存中去（300万个字符串假设没有重复，都是最大长度，那么最多占用内存3M\*1K/4=0.75G。所以可以将所有字符串都存放在内存中进行处理），而现在只是需要一个合适的数据结构，在这里，HashTable绝对是我们优先的选择。

所以我们放弃分而治之/hash映射的步骤，直接上hash统计，然后排序。So，针对此类典型的TOP K问题，采取的对策往往是：hashmap + 堆。如下所示：

hash\_map统计：先对这批海量数据预处理。具体方法是：维护一个Key为Query字串，Value为该Query出现次数的HashTable，即hash\_map(Query，Value)，每次读取一个Query，如果该字串不在Table中，那么加入该字串，并且将Value值设为1；如果该字串在Table中，那么将该字串的计数加一即可。最终我们在O(N)的时间复杂度内用Hash表完成了统计；

堆排序：第二步、借助堆这个数据结构，找出Top K，时间复杂度为N‘logK。即借助堆结构，我们可以在log量级的时间内查找和调整/移动。因此，维护一个K(该题目中是10)大小的小根堆，然后遍历300万的Query，分别和根元素进行对比。所以，我们最终的时间复杂度是：O（N） + N’ \* O（logK），（N为1000万，N’为300万）。

当然，你也可以采用trie树，关键字域存该查询串出现的次数，没有出现为0。最后用10个元素的最小推来对出现频率进行排序。

3、有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M。返回频数最高的100个词。

由上面那两个例题，分而治之 + hash统计 + 堆/快速排序这个套路，我们已经开始有了屡试不爽的感觉。下面，再拿几道再多多验证下。请看此第3题：又是文件很大，又是内存受限，咋办?还能怎么办呢?无非还是：

分而治之/hash映射：顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件（记为x0,x1,…x4999）中。这样每个文件大概是200k左右。如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M。

hash\_map统计：对每个小文件，采用trie树/hash\_map等统计每个文件中出现的词以及相应的频率。

堆/归并排序：取出出现频率最大的100个词（可以用含100个结点的最小堆）后，再把100个词及相应的频率存入文件，这样又得到了5000个文件。最后就是把这5000个文件进行归并（类似于归并排序）的过程了。

**密匙二：多层划分**

多层划分—-本质还是分而治之的思想，重在“分”的技巧上！

适用范围：第k大，中位数，不重复或重复的数字

基本原理及要点：因为元素范围很大，不能利用直接寻址表，所以通过多次划分，逐步确定范围，然后最后在一个可以接受的范围内进行。

问题实例：

4、2.5亿个整数中找出不重复的整数的个数，内存空间不足以容纳这2.5亿个整数。

有点像鸽巢原理，整数个数为2^32,也就是，我们可以将这2^32个数，划分为2^8个区域(比如用单个文件代表一个区域)，然后将数据分离到不同的区域，然后不同的区域在利用bitmap就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。

5、5亿个int找它们的中位数。

这个例子比上面那个更明显。首先我们将int划分为2^16个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

实际上，如果不是int是int64，我们可以经过3次这样的划分即可降低到可以接受的程度。即可以先将int64分成2^24个区域，然后确定区域的第几大数，在将该区域分成2^20个子区域，然后确定是子区域的第几大数，然后子区域里的数的个数只有2^20，就可以直接利用direct addr table进行统计了。

**密匙三：Bloom filter/Bitmap**

Bloom filter

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集

基本原理及要点：

对于原理来说很简单，位数组+k个独立hash函数。将hash函数对应的值的位数组置1，查找时如果发现所有hash函数对应位都是1说明存在，很明显这个过程并不保证查找的结果是100%正确的。同时也不支持删除一个已经插入的关键字，因为该关键字对应的位会牵动到其他的关键字。所以一个简单的改进就是 counting Bloom filter，用一个counter数组代替位数组，就可以支持删除了。

如何根据输入元素个数n，确定位数组m的大小及hash函数个数也是一个很重要的问题。当hash函数个数k=(ln2)\*(m/n)时错误率最小。在错误率不大于E的情况下，m至少要等于n\*lg(1/E)才表示任意n个元素的集合。但m还应该更大些，因为还要保证bit数组里至少一半为0，则m应该>=nlg(1/E)\*lge 大概就是nlg(1/E)1.44倍(lg表示以2为底的对数)。

举个例子我们假设错误率为0.01，则此时m应大概是n的13倍。这样k大概是8个。

注意这里m与n的单位不同，m是bit为单位，而n则是以元素个数为单位(准确的说是不同元素的个数)。通常单个元素的长度都是有很多bit的。所以使用bloom filter内存上通常都是节省的。

密匙四：Trie树/数据库/倒排索引

Trie树

适用范围：数据量大，重复多，但是数据种类小可以放入内存

基本原理及要点：实现方式，节点孩子的表示方式

扩展：压缩实现。

数据库索引

适用范围：大数据量的增删改查

基本原理及要点：利用数据的设计实现方法，对海量数据的增删改查进行处理。

倒排索引(Inverted index)

适用范围：搜索引擎，关键字查询

基本原理及要点：为何叫倒排索引？一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。

以英文为例，下面是要被索引的文本：

T0 = “it is what it is”

T1 = “what is it”

T2 = “it is a banana”

我们就能得到下面的反向文件索引：

“a”: {2}

“banana”: {2}

“is”: {0, 1, 2}

“it”: {0, 1, 2}

“what”: {0, 1}

检索的条件”what”,”is”和”it”将对应集合的交集。

正向索引开发出来用来存储每个文档的单词的列表。正向索引的查询往往满足每个文档有序频繁的全文查询和每个单词在校验文档中的验证这样的查询。在正向索引中，文档占据了中心的位置，每个文档指向了一个它所包含的索引项的序列。也就是说文档指向了它包含的那些单词，而反向索引则是单词指向了包含它的文档，很容易看到这个反向的关系。

**密匙五、外排序**

适用范围：大数据的排序，去重

基本原理及要点：外排序的归并方法，置换选择败者树原理，最优归并树

1).有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16个字节，内存限制大小是1M。返回频数最高的100个词。

这个数据具有很明显的特点，词的大小为16个字节，但是内存只有1M做hash明显不够，所以可以用来排序。内存可以当输入缓冲区使用。

**密匙六：分布式处理之Mapreduce**

MapReduce是一种计算模型，简单的说就是将大批量的工作（数据）分解（MAP）执行，然后再将结果合并成最终结果（REDUCE）。这样做的好处是可以在任务被分解后，可以通过大量机器进行并行计算，减少整个操作的时间。但如果你要我再通俗点介绍，那么，说白了，Mapreduce的原理就是一个归并排序。

适用范围：数据量大，但是数据种类小可以放入内存

基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

The canonical example application of MapReduce is a process to count the appearances of each different word in a set of documents:

海量数据分布在100台电脑中，想个办法高效统计出这批数据的TOP10。

一共有N个机器，每个机器上有N个数。每个机器最多存O(N)个数并对它们操作。如何找到N^2个数的中数(median)？

本文题目为秒杀99%的海量数据处理面试题，而不是100%呢。OK，给读者看最后一道题，如下：

非常大的文件，装不进内存。每行一个int类型数据，现在要你随机取100个数。

我们发现上述这道题，无论是以上任何一种模式/方法都不好做，那有什么好的别的方法呢？我们可以看看：操作系统内存分页系统设计(说白了，就是映射+建索引)。

Windows 2000使用基于分页机制的虚拟内存。每个进程有4GB的虚拟地址空间。基于分页机制，这4GB地址空间的一些部分被映射了物理内存，一些部分映射硬盘上的交换文 件，一些部分什么也没有映射。程序中使用的都是4GB地址空间中的虚拟地址。而访问物理内存，需要使用物理地址。 关于什么是物理地址和虚拟地址，请看：

物理地址 (physical address): 放在寻址总线上的地址。放在寻址总线上，如果是读，电路根据这个地址每位的值就将相应地址的物理内存中的数据放到数据总线中传输。如果是写，电路根据这个 地址每位的值就将相应地址的物理内存中放入数据总线上的内容。物理内存是以字节(8位)为单位编址的。

虚拟地址 (virtual address): 4G虚拟地址空间中的地址，程序中使用的都是虚拟地址。 使用了分页机制之后，4G的地址空间被分成了固定大小的页，每一页或者被映射到物理内存，或者被映射到硬盘上的交换文件中，或者没有映射任何东西。对于一 般程序来说，4G的地址空间，只有一小部分映射了物理内存，大片大片的部分是没有映射任何东西。物理内存也被分页，来映射地址空间。对于32bit的 Win2k，页的大小是4K字节。CPU用来把虚拟地址转换成物理地址的信息存放在叫做页目录和页表的结构里。

物理内存分页，一个物理页的大小为4K字节，第0个物理页从物理地址 0x00000000 处开始。由于页的大小为4KB，就是0x1000字节，所以第1页从物理地址 0x00001000 处开始。第2页从物理地址 0x00002000 处开始。可以看到由于页的大小是4KB，所以只需要32bit的地址中高20bit来寻址物理页。

返回上面我们的题目：非常大的文件，装不进内存。每行一个int类型数据，现在要你随机取100个数。针对此题，我们可以借鉴上述操作系统中内存分页的设计方法，做出如下解决方案：

操作系统 中的方法，先生成4G的地址表，在把这个表划分为小的4M的小文件做个索引，二级索引。30位前十位表示第几个4M文件，后20位表示在这个4M文件的第几个，等等，基于key value来设计存储，用key来建索引。

但如果现在只有10000个数，然后怎么去随机从这一万个数里面随机取100个数？请读者思考。