MapReduce初步1

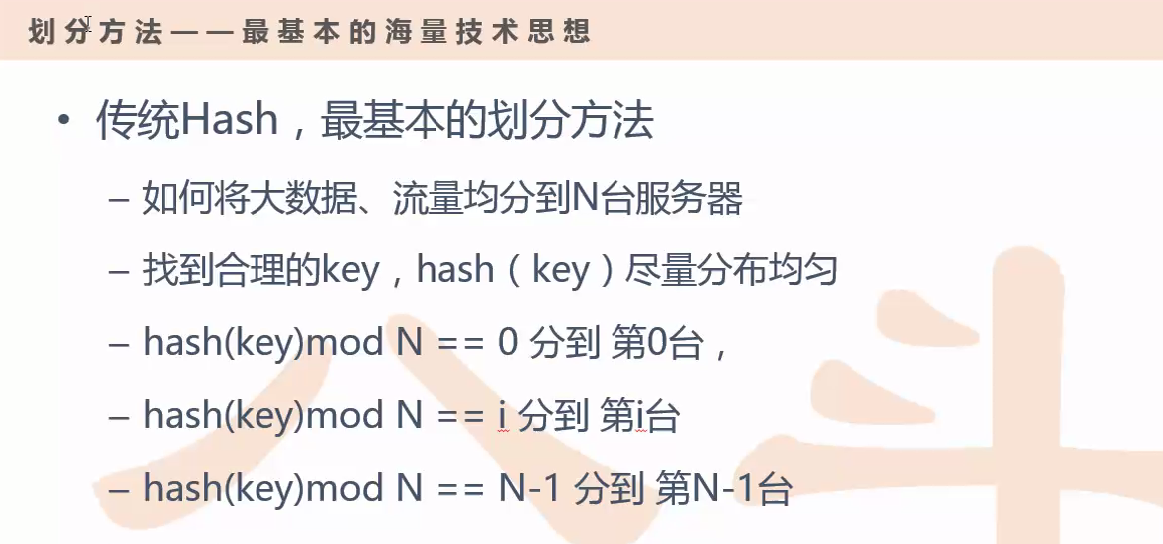


1. 从事大数据行业，必须要知道MapReduce，就算你以后到了公司不做MapReduce开发，那么也是要学会MapReduce的一个思想，如果是不知道这个MapReduce这个思想的话，再去学其他的大数据相关的东西比如数据流转等会有一定的困难，所以MapReduce是一个基础也是大数据之中的核心的核心，起码我是这么认为的（假装我是林中天/孙老师）



2.今天带来的内容有两方面，主要有两个内容，一个就是海量数据分流处理技术，那么MapReduce它是属于并发计算，那么可以认为MapReduce是一个海量数据分流处理技术，因为它自身是基于hadoop平台的，在hadoop生态里面它承担着一个集成框架这么核心的角色，那么它自身是可以处理大数据的。MapReduce是海量数据分流处理技术，但是我们反过来思考呢，其实并不是所有的海量的数据或者是大数据处理都是要利用MapReduce来解决的，所以这一点是不一样的，就是以后要碰到大数据的场景不一定要用到MapReduce

，但是MapReduce是处理大数据的，那是为啥呢？因为毕竟这个hadoop的这个生态虽然最近几年大家用的已经很普及了。

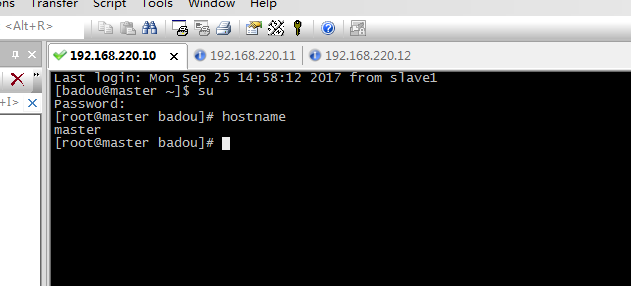


1. 传统海量数据处理：

其实最简单最容易想到的这种海量数据处理的方法就是Hash方法，那么什么是Hash方法呢？那么我们来举个例子：打开一台master终端，进行操作（如下图所示）

命令行输入：su 切换管理员权限

命令行输入：hostname 查看该台终端的名字



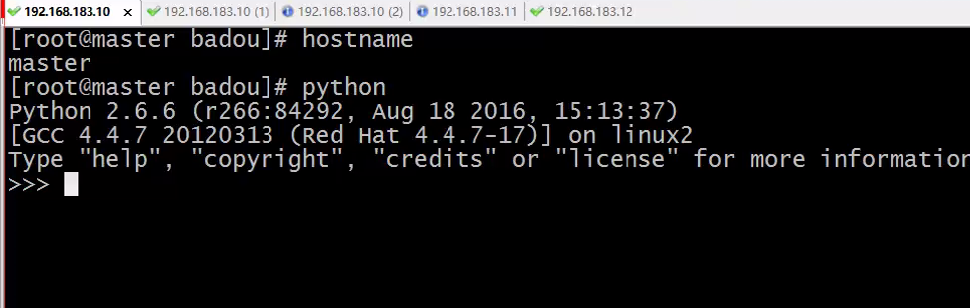
4.因为系统安装完了以后会自动帮你安装一个python环境，那么只需要在终端命令行输入python环境即可

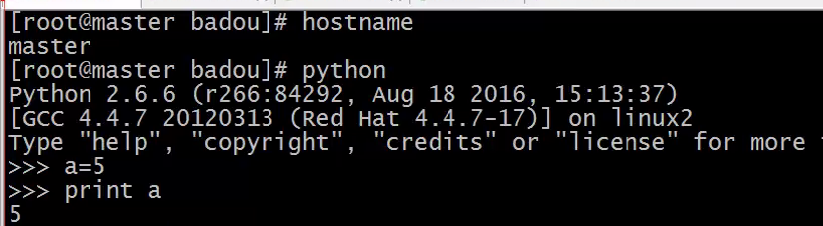
命令行输入：python

然后就可以在python环境中进行语法的演练

python环境命令输入：a=5

python环境命令输入：print a



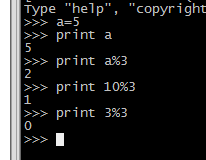


刚刚说了hash方法，那么什么是hash方法呢？比方说我们要a模3，那么a模3是怎么计算的呢？(如下图所示)

python环境命令输入：print a%3 得于2

python环境命令输入：print 10%3 得于1

python环境命令输入：print 3%3 得于0



5. 不管是什么样的整数数字，最后它的取值0，1，2而不会超过3，那如果后面的那个模变成了5的话呢，那他最后的输出肯定是0到4之间，就是求一个余数，那么这种就是相当于一种hash方法的简单的理解。也就是说你当来到了一个数字的话呢，那我面前就会有几个桶，那这几个桶呢会通过一个余数的方式进行一个表示，然后通过这样的一个计算呢，通过这样一个简单的hash公式就可以得到我相应的这个数字我应该是命中哪个桶，返回哪个桶号。那通过这样一个简单的一个计算，其实这种思想大量的运用，特别是（如下图所示）



那么这张图是什么意思呢？假设说我们访问一个新浪的首页，因为这个全国各地很多的人在去同时请求新浪的首页，那大家呢都是通过流量请求，瞬间或者是同一秒钟那有大量的并发进来，如果你要用同一台机器去扛这个高并发的话是很难的，很容易把这个机器给扛爆了，通常在互联网行业里面提供服务的一台机器肯定是不行的，需要多台机器进行相互合作，那么这Hash（）%max下面有六条线，代表的是六台服务器集群，然后每一个数字代表一个节点，每一个节点代表一个具体的一个服务器，那这个时候如果要是前面有那么大的流量请求的话，只用一台机器是扛不住的，怎么办呢？那么需要把这个流量请求进行一个让别人来帮我共同分担，那这个时候相当于是整体的集群它承担的压力总数不变，但是每台机器上扛得压力是被均分了，使得每一个服务器它的一个压力就减少很多，这就是一个非常简单的一个负载均衡的一个原理。

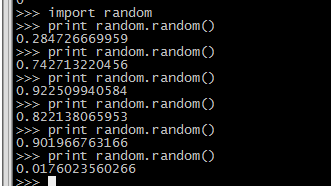
来了一个流量请求，然后我通过一定的算法，通过一定的hash方法把这个请求直接打到了相应的服务器上去，那具体我来了一个流量请求，我这个流量请求应该打到哪台服务器上去呢？就通过这么一个hash方法

那当前来了一个流量请求，怎么知道把这个流量请求打到哪台服务器上去呢？那如果这个流量请求里面带了一个固定的数字的话那也是不行的，所以呢得对每一个流量请求会有一个随机数，通常这个随机值根据你的时间作为一个种子然后random一下，保证他的这种随机性非常均匀的一个状态，那么我们在master终端模拟一下随机数（如下图所示）

python里面有一个随机数的random的一个库，这个库就是提供一个随机数，那么我们通过这样的一个方式把random加载进来

Python环境命令输入：import random

Python环境命令输入：python random.random()



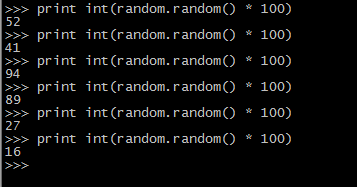
每次执行完，随机数的数值都不同

6.假设说我们在随机数上面乘一个100，然后我们把它转成，因为随机数乘完100以后呢，它还是一个浮点型（就是带小数点的值），那么这个时候强制的把浮点型转换为int（整数型）

Python环境命令输入：print int(random.random() \* 100)

随机生成一个100以内的整数型的随机数（如下图所示）

相当于我来了一个流量请求就得到一个随机的数字，而且这个随机得来的数字是非常均匀的



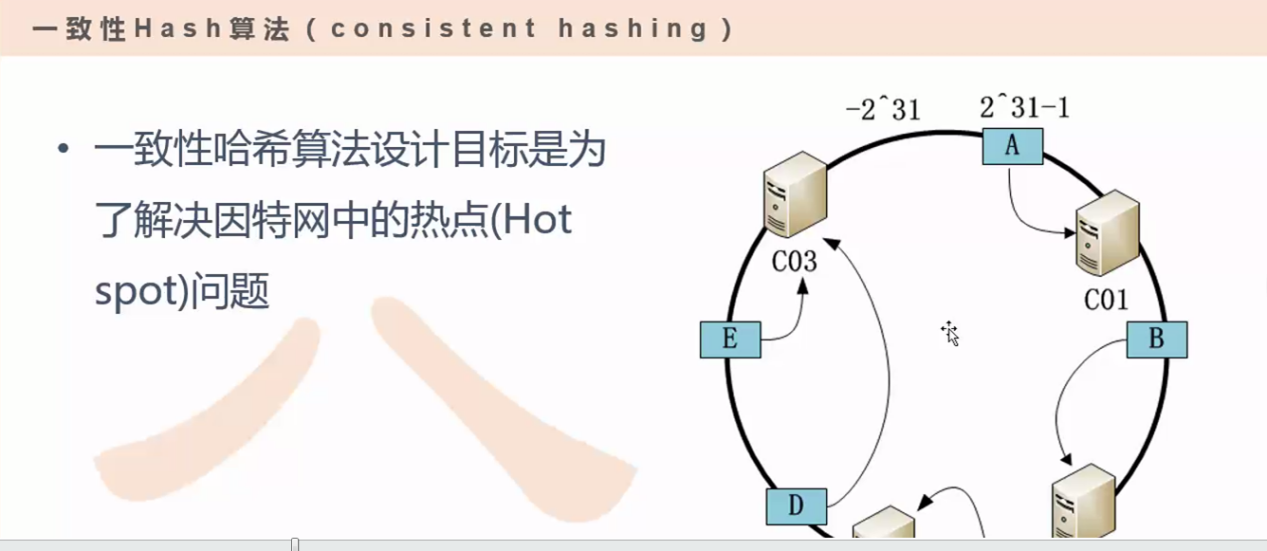
7.如果要是说像刚刚那个流量分发的图一样，有6个桶的话，那具体这个请求真正的应该被哪个桶所对应的服务器进行服务呢？刚刚上面写的余数0到5之间，那么1是代表第二台机器。那么就是比如在随机数上得到的第一个数字给第一个流量请求进行服务器的服务。

通过这样的一个模拟，也就是说通过一个很简单的hash方法把大量的这种流量请求进行分流，其实大量的请求也就是大数据的一个场景。

其实这是一个非常简单的一个思想。这种情况下只是对这个流量请求的流量做一个hash方法，其实这种思想呢还是适合于很多的场景，比如说存储的时候，如果你有大量的数据摆在你的面前，如果你把这个大量的数据直接存到一个mysql的话呢，那势必给这个mysql建立一个非常大的一个表，而且这个表，它是每一条数据都是代表一行记录，这个时候也可以通过这种分区的方式呢可以建立多个表，比如说可以建32个分区或者是建10个分区，然后你可以建10个表来承担整个的一个数据存储，所以这也是利用了一个hash的方法，那这种方法看了下非常简单并且这种方法还很直接，那其实我们实际遇到的情况要远远比这个复杂的多，那什么情况下这种hash算法是不太适合的？

假设说来了一个流量请求，刚刚好把这个流量请求分配到第三号机器上，那就在这个时候，这个机器突然挂掉了，那既然把这个数据发送过来给第三号机器出现问题了已经死掉了，那这第三号机器不能给你正常的提供服务了，那么相当于本次请求是一个失败的请求，那么很容易引起就是很多的请求都不能服务的到，那么对于网站来说就是一个损失，那有没有解决这样的一个问题的方法呢？答案是肯定是有的，还有一种更高级的hash方法就是：一致性hash方法。

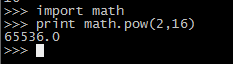
7.一致性hash哈希方法就是：动态增长更高级的一个划分方法，一致性hash哈希算法设计目标是为了解决因特网中的热点（Hotspot）问题（如下图所示）



通过这么一个页面，你可能感受不到什么是一致性哈希方法，你看到的是一个环，那对于这种比如说一个uint16，uint16的最大值是多少？我们来看一下（如下操作和下图所示）

Python环境命令输入：import math //这是一个数学算法的一个模块

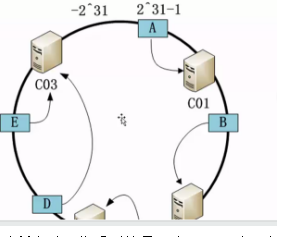
Python环境命令再输入：print math.pow(2,16) //2的16次方



最后得到的值是：65536，那么也就是说uint16就是一个0到65536的这么一个范围，那如果是说当前正好声明这个变量刚刚是uint16并且当前它这个赋的值就赋了一个最大值65536，那如果是再往上加一个1，那么其实它就超出了这个一个范围了，加完1之后相当于最大值变成一个最小值，这个概念在我们大家学编程的时候应该知道把？如果uint16，那你从一个最大值又加了一个1，它就会变成最小值，变成个0。所以我们发现如果我们一直对这个数字一直加的话呢相当于一个环（就如下环图所示），

那么在这下面环图里面列的是int32类型的，int32类型负的2的31次幂到2的31次幂减一这么一个范围

那如果是2的31次幂加1的话呢相当于是又变成它的最小值，所以你可以想象把这个数字的最大值和这个最小值首尾相接变成了这样的一个环形（如下图所示），相当于哈希方法在这么一个环形的哈希空间上然后呢同样借助刚才那种传统的哈希思想然后把相应的key然后往这个环型上进行一个映射



8.再举个例子，如下图所示



那么这图的意思是什么呢？

假设说有三台机器，这三台机器可以认为是新浪的三台不同服务器，每台机器都可以向外提供服务，然后每台机器提供服务都是一模一样的，无论你这个请求达到一个任何服务器的其中一个，得到的服务都是一样的。然后呢这三台机器对应着三台不同的ip地址(如上图所示)

Ip1到ip3，那么我们用一个非常直观的方法来对应这三台机器，比如用一个三角形来描述这个A这台机器，四方形描述这个B这台机器，圆形描述C这台机器，代表着这三台不同的机器，但是由于这三台机器可能它部署的机房的原因或者是这三台机器自身的硬件有一定的差别，所以导致这三台机器它能够承担压力是不一样的，比如说这个三角形A这台机器它可能是32核的机器并且内存超大，那么它就可以扛得住非常大的并发请求，那么圆形C这个机器可能比较弱一点，它可能就是一个32位的一个处理器，内存小一点，它承担的这种压力是非常小的就比如我们的笔记本性能的差异，每台笔记本都有性能不同的机子。

所以既然出现这么一个彼此的自身的一个差异的存在，所以我们假设ABC三台机器他们的处理能力不同，那么我们就假定说A 可以承担百分之50，B可以承担百分之30，C可以承担百分之20，就是说A这台机器是好的呢，那么我们尽量让A这台机器多干点活也就是说把更多的流量请求发到A这台机器上进行处理。

那我们怎么能够去实现这么一个方案呢？那么这时候我们需要做一些个虚拟机器的创建，但是这个虚拟机器不是一个真实的存在，它只是个虚拟的一个节点，这个虚拟节点怎么虚拟呢？我们有三台机器如A,B,C然后A这台机器不是可以承担百分之50吗？那我就按照比例，5比3比2的方式假设说我由A衍生出五个不同的机器（A1到A5，如下图所示）

分别是5个虚拟出来的机器，但是你去访问任何一个机器都是A来提供服务



然后B呢衍生出3个机器（B1到B3，如下图所示）



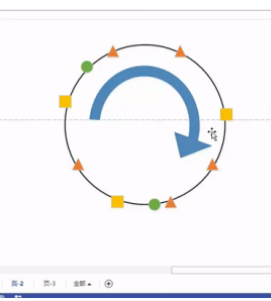
C呢衍生出2个机器（C1和C2，如下图所示）



那说虚拟出一个三台机器，到底是通过怎么样的一个形式去描述呢？那相当于最简单的方式就是如上图所示有IP地址，可以在这IP地址可以认为是一个字符串，你可以在这字符串的节点上就加一个1，A2就是192.168.1.100\_2（如下图所示）



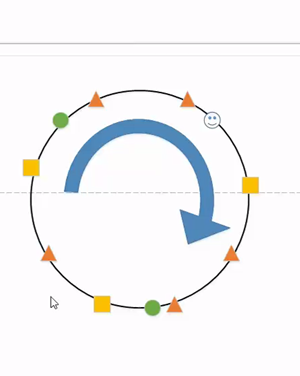
所以相当于你可以通过这样的一个虚拟的一个映射瞬间能达到10台不同的ip地址，然后这个时候呢你就得到了不同的10个字符串，你拥有这10个不同的字符串通过一个哈希方法或者通过一个加密方法就可以映射出一个数字出来，然后这个数字就可以在这个环上（如下图所示）找到相应的位置，那既然A这台机器拥有5台机器，那肯定是我在如下图的环上可以找到5个不同的三角形点，然后B呢就是环形图中3个四方形的点，而C呢就是环形图中2个圆形的点（如下图所示）



从这三台不同的机器，然后一直到往这个环上进行映射整个的流程应该都明白了把？

那这个时候呢再做一个假设，那这个数字是由着最大值变成一个最小值，那相当于在这个环上可以认为一个顺时针也好逆时针也好，只要这个顺序不变就可以，那就按照顺时针的方式去遍历，那这个时候呢假设说我有一个请求比如有一个用户叫张三进来了，他通过页面访问新浪，按了下回车，然后呢它会得到一个随机数，那这个随机数就映射到环上的随意的一个地方（如下图所示）下面三角形旁边有一个笑脸的就代表着张三，那么这ABC三台机器应该由哪台机器给他进行服务呢？顺时针嘛，从笑脸那个点开始走找到离笑脸最近的是四方形B机器，那么就由B机器来给张三提供服务。

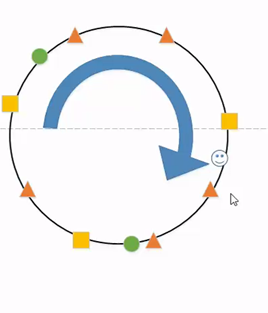
这就是非常简单的一个一致性哈希的一个思想



相当于是我是用一台实际的一个ip地址，虚拟出了多个相互之间不会重复的一个字符串，然后来表示那些虚拟的机器。不是说这个机器A处理能力强吗？那我在环形上多建立几个A的节点，通过这样的一个方式在这个环上A的节点是不是最多啊？

随着这种请求的流量的增大，会发现A的命中肯定是百分之50，如果要是在用哈希方法在这个环上哈希的够均匀的话，其实A承受的这种压力也就是大概是5比3比2这样的比例。

那如果是张三或者是李四请求落到了（如下图所示）的点上

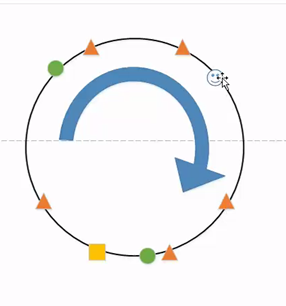


那么你会发现C不是只有两个节点吗？所以很难去命中C的节点

那么这就是很简单的一个一致性哈希方法。

那这个一致性哈希方法刚才说过了，它可以拟补传统的哈希方法的不足。

那传统哈希方法有什么不足呢？我们提到了假设说有一台机器突然挂掉了，没有办法提供服务，假设说这个B这台机器可以给张三提供服务，但是这台B机器刚刚好出现故障了，那么这个时候B这台机器在这个环上删除，删除之后呢，那接下来的B它不能提供服务了，那我该由谁来帮B服务呢？起码要有人接班嘛（如下图所示）



那么顺时针走，发现A离自己最近，那么自然而然的就到A这台机器进行处理请求服务了

那么这时候突然B机器修好了，那么这B机器映射好的字符串是不变的，那它又在环上做了个还原，那接下来呢如果有同样的这样流量请求达到这个区间的话呢，那么这个区间和流量仍然由这个B机器进行处理。

所以它是一个动态的一个调节的一个方法

9.再举个例子：

比如当前来了100个请求，那么这个时候ABC分别承担，如果是均匀的话呢A是承担50个请求，B是30个请求，C是20个请求，那如果要是说这个B这台机器都挂了，本来这个B能可以承担30个请求的，但是B死掉了，那么这B的30个请求必须由A和C这两台机器来共同承担，那具体这30个请求，A和C怎么来去瓜分这样的一个形式呢？仍然按照这种5比2的形式，也就是说30个请求乘于7分之5的流量请求是给A这台机器，然后剩下的7分之2的流量请求给了C这台机器，这30个流量要统一的分配到A和C这两台机器，因为A和C不可能是均匀分配，毕竟A和C两台机器他们的这个性能不一样所以区分了。

整体的一致性哈希的思想就是这样。

讲了两种哈希方法还做了总结，那这两种哈希方法都是早期的数据处理最简单的数据处理思想，那就是面对海量的问题时候，一般把这个问题会分成三类：一个是大数据量，一个是大流量，一个是面对大计算，那什么是大数据量呢？大数据量刚才我们已经说过了，就是相当于刚才说的那个对于mysql去分表这个一个思想，那一个表呢存不下那么多数据，那这时候我们需要用创建多个小表来共同承担，所以大数据量呢就可以通过刚才这种哈希的方式可以来解决这个方案。

大流量其实也是哈希方式，这个大流量相当于刚才我们举得那个新浪的，用户去请求它的首页一样，它是按照流量去划分，对流量进行一个哈希分流也可以用刚才说的一致性哈希。

大计算是一个最后的一个情况，而这个情况跟刚才我们说的那两种很简单的哈希方法是没有太大的关系的，那面对这种情况下，大数据怎么搞呢？大计算就需要用到MapReduce来去解决，那由此呢我们就引出了一个MapReduce这么一个概念。

那MapReduce就是一个集成框架，它这种集成框架就是来帮助我们解决大计算的问题，那通过今天的课程，大家会起码知道什么是大计算和MapReduce怎么去解决这种大计算的一个流程，那通常说做Hadoop相当于就是给服务给大数据，那如果提到大数据的话呢通常也会提到云计算，那对于云计算来说通常也就是一些个分布式的那些机器，分布式的那些系统来去提供服务，那也就是说在对于这种处理云计算的一些情况下呢那肯定是没有这种单节点去全部承担所有的数据压力和存储，那需要把这个问题通过分布式的方法来解决，就好比一个人能力总会有限的，需要一个团队来共同合作去完成某个事情某个任务。

那对于一个分布式集群来说要考虑一些问题，特别是这个稳定性和这个容错能力，对于单台机器来说这个稳定性是没有什么问题的，因为只要硬件不坏程序设计的合理那就没有其他问题，但是这种分布式的场景，因为你是需要考虑到多个不同机器互相协作，那这时候考虑到的问题就会很多，所以这是一个难点。

还有一个就是数据一致性怎么去保证，比如说这有一个画为蓝色的这台机器（如下图所示），这台机器里面有个变量就是A=5，然后这里面有两个用户，其中一个用户的目的是想把这个A=5这个变量做一个加加（++），那怎么做加加（++）呢？首先需要把A=5这个变量读取到一个用户中，然后在用户把这个A=5做一个加法，A就变成了一个A=6，读完之后，本地做完计算，把这个数据进行写回去，那么这个时候蓝色里面的A=5就变成了A=6（如下图二所示）

图1：

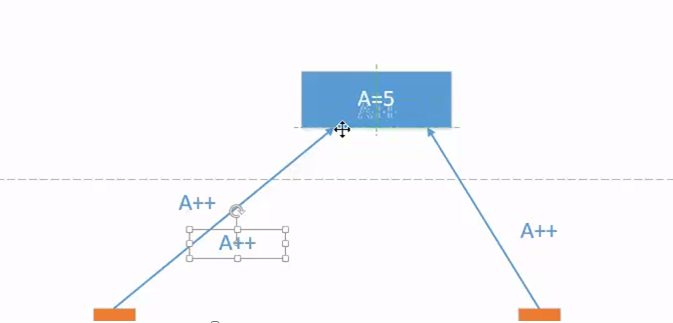
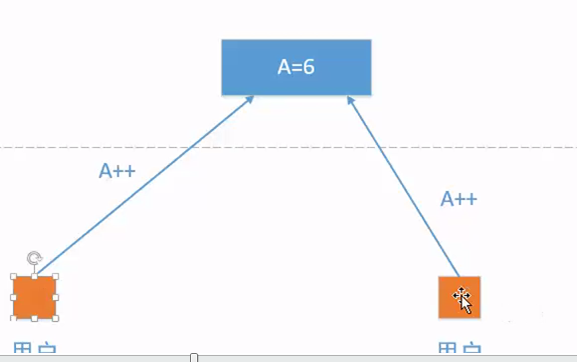
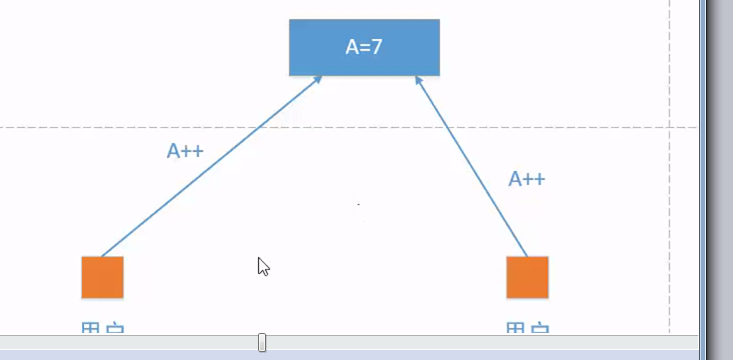


图2：

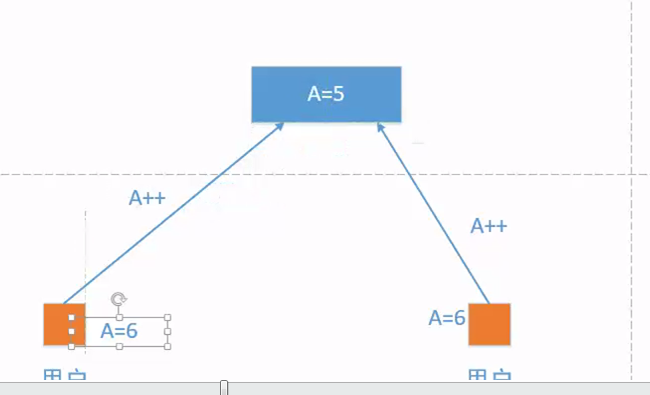


假如图二的图，左边的用户是A用户，右边的用户是B用户，那A用户已经做完了加法了，那B用户也需要做加法，那么这个时候相当于通常的流程，把这个蓝色颜色里面的A=6这个变量读取到B用户当中，然后B用户进行加法操作变成A=7，然后再把这个A=7返回回去给蓝色小图那里，然后蓝色小图就相当于是A=7了（如下图所示）

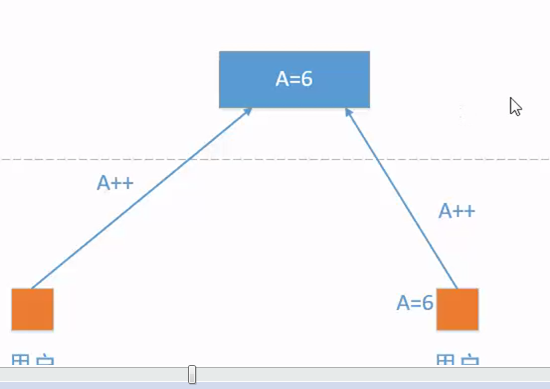


这感觉没有什么问题，但是有一种场景是当这个用户A进行了读取并进行本地加法操作然后保留在自己本地而并没有没有返回加法操作以后的数据给蓝色小图那里使得蓝色小图里面的值没有改动（如下图一所示），那么这个时候B用户发送了请求然后把蓝色小图的变量A=5读取下来了，然后在用户B当中进行本地加法操作变成A=6，然后用户A把做完加法操作的数据（A=6）进行返回回去然后蓝色小图（如图二所示），那么这个时候B用户也进行了在本地加法操作了，也把A=6这个变量返回给蓝色小图（如图二所示），会发现这个蓝色小图里面的A=6仍然等于A=6而并没有改动，那为啥出现这种情况，就是因为A和B这两个用户他们是没有遵守一个顺利一致性这么一个情况。

图一：



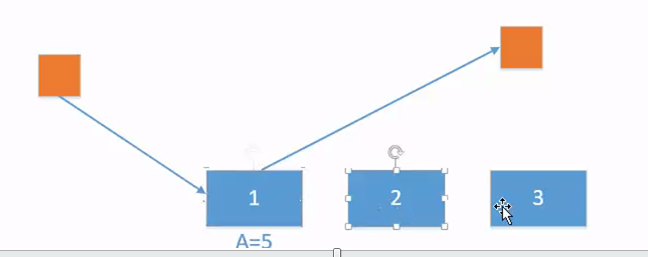
图二：



所以在这个集群里面经常会遇到这种如果是数据不一致的问题会导致一些数据的读取错误。比方说B用户这时候读的数据是脏数据，那既然B用户读的数据是脏数据怎么做计算，最后也是得到的是一个计算错误的数据。数据一致性分了两种：一个是弱一致性，一个是强一致性。那什么是弱一致性呢？

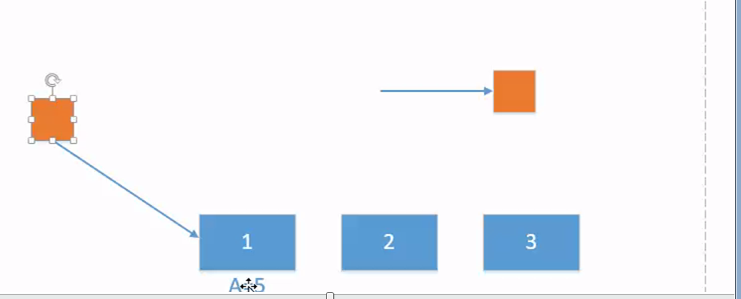
比如：因为在整个的一个集群环境里面有很多的机器，然后呢比如说有这么一批机器每一个机器它提供的服务都是一样的，也就是说每台机器它上面存的数据应该都是保证一致性的，那么这时候比如里面有三台机器，三个蓝色小框代表三台机器（如下图三所示）

图三：



然后红色左边部分是A用户，右边红色部分是B用户，那么A用户往这下面三个蓝色小框的集群上开始写（如下图四所示）

图四：



首先呢这个A用户先A=5这个数据写入1号蓝色机器上去，然后A=5又写到第2号蓝色机器上去，然后又把A=5写到第3号蓝色机器上去(如下图图5所示)。

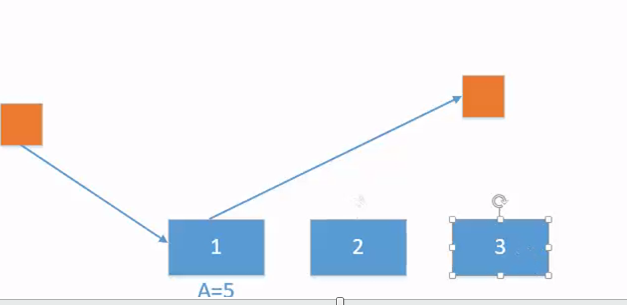
图5：



那么这时候这三台机器它在同一时间同一时刻状态都是一致的，那么这时候右边B红色用户无论从哪台机器上进行去读取数据，读取的结果都是一样的，那么这种情况是一个：强一致性。

那也就是说必须把所有的节点，每个节点上的数据都要保持一致性之后整体才能提供服务，否则一旦有哪台机器的数据还没有就绪状态，那么即使其他两台机器或者其他已经就绪的状态的机器或者是数据已经是最新的了那么也是不允许提供服务的。

那弱一致性是什么意思呢？比如：当我有一台1号机器的数据已经是最新的了，这么这台1号机器可以对外提供服务了（如下图6所示）



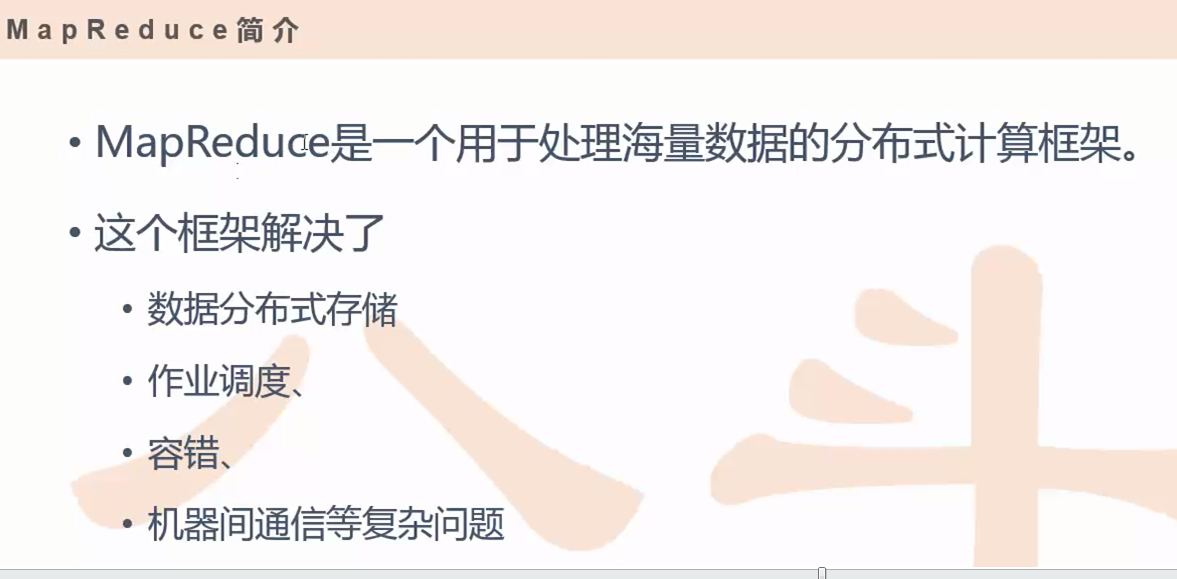
那么2号和3号机器还没有获取到最新的数据，就不提供服务。

当整个集群里面只有一台机器或者是有部分机器它的数据更新了，那么就可以提供服务了，那么这个时候有一个好处就是跟刚才强一致性比它是不是提供服务的速度要快很多？从你写入到读出，中间这个时间差要缩短很多，所以这种弱一致性它可以快速的提供服务，但是有一个问题，如果你从全局上来看，它的数据是不一致的，虽然你提供服务给人的感觉是很一致的，其实整体来看其数据还是不一致的，所以这是一个弱一致性的理解。

那对于大数据这种或者是分布式集群里面对这种数据一致性要求是每一个集群或者是每一个场景它要求是不一样的，另外对于分布式集群系统它有什么难点呢？难点就是说在这个集群里面你任何消息都存在丢失的可能性，然后任何机器都有可能存在故障的可能，也就是说当有消息丢失了，怎么去拟补这么一个损失，如果有一台机器出现了故障硬盘坏了CPU烧掉了，那怎么去拟补这么一个问题，起码这台机器坏了不应该影响整个的集群服务，对外让用户来看它是透明的。

所以接下来我们前面做了很多的铺垫，接下来我们就正式的去揭晓这个MapReduce这个集成框架的一个神秘面纱。

10.MapReduce其实就是一个集成框架，这个集成框架是处理海量数据的并且是一个分布式的，就是有多个机器共同组成了一个集群来提供服务



这个框架解决什么问题呢？

一个是数据分布式存储，对这个之前复习过MapReduce的你们来说，对这个hadoop其实它自身是不存储数据的，那么数据其实都是存储到HDFS上，hadoop生态的底层是HDFS然后再往上走就是一个MapReduce，其实这个MapReduce是不存数据的，它只负责计算，其实它的数据都存储到HDFS上，MapReduce它的计算目标数据就是来自HDFS上，所以MapReduce仅仅是做计算的，存储是由HDFS接管。

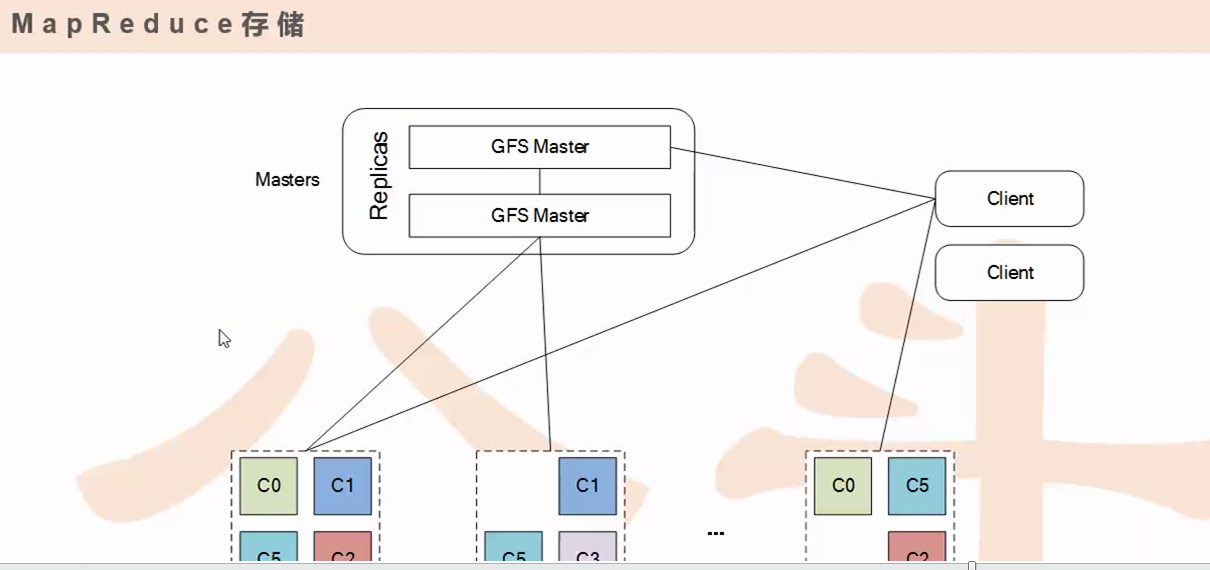
作业调度:就是说因为整个MapReduce集群你不可能比如在一个公司里面搭了一个hadoop集群，这hadoop集群可以跑很多的MapReduce，那不可能说这个MapReduce这个资源都被某一个人占了也就是说你可以提交一个任务那其他人也可以同时提交任务，就是这个资源大家共享，就像这个游泳池一样，那整个的游泳池不可能都让你一个人占了，其他人也想游泳，那么怎办？那么就大家一块都下到游泳池里面，然后一块去游泳，所以在游泳池里面可能会划分出每个人的任务都不同，可能你划分出这一区域大小也有可能不一样，其实呢你整个的游泳池它也是有一个大小的概念，不可能容纳所有的人都进去，那一旦我这个游泳的人数的密度一旦达到了一定程度了那如果再往里容纳更多的人肯定会影响当前的用户体验，那这个时候后面的人后面的任务必须要排队，那这游泳池里面哪个任务执行成功之后那他自然的会退出，那么这个时候会把他的资源空闲出来，那么后面的人再进去相当于是先后先出这么一个队列，大概是这么一个思想。

容错：也很好理解；比如你的服务器出现故障了怎办？如果你的任务出现故障了怎办？只要不是你的代码导致的，那其他这种非个人因素所导致的问题，那么这个时候呢就需要有一个系统说如果由于非我个人原因导致的一些问题比如说网络之间可能突然阻塞了，那不需要影响到我当前的这个任务怎办呢？如果当前这个网络不够通畅，那你应该这个集群帮我能够自动的识别出网络不通畅，应该是不是帮我再切换到其他的节点上去这么个情况，相当于在背后可以帮我隐藏出一些底层实现的细节，它既然可以帮我屏蔽一些细节，那就可以达到了一定的容错的目的。

机器间通信等复杂问题：就是说对于用户来说我们不需要关心哪台机器具体做什么，这个系统又是透明的，我们只需要知道我们当前的这个任务用了多少资源就可以了，当你这个机器通信一旦出问题了那就不能够影响我当前这个任务，这其实跟刚刚的容错差不多的问题。

接着刚才说MapReduce它是不存储数据的，那MapReduce它的这个数据存储在HDFS上。我们先大概的了解一下HDFS的情况，那其实这是一个整个HDFS存储的图（如下图1所示）

图1：

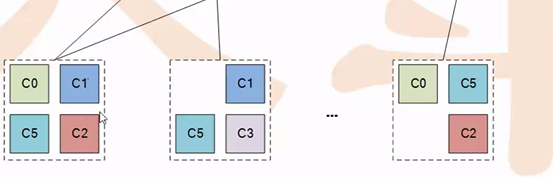


在这个图1里面其实你可以看的出来也是一个集群， Masters代表着一个机器或者多个机器，那么下面那些正方形里面有很多的小正方形代表的是从节点，所以如果对于集群没有这种概念的人来讲通常面对一个集群来说都需要有两种角色，一种角色是Masters,另外一种角色是slave，一个主一个从，那么主是干嘛的呢，就是公司的领导，那公司的领导一般通常是不干活的，那他只是做一个任务的下发，就是说突然这个公司我来了一笔买卖，那我这个买卖应该怎么做，具体的事情领导不出面去做，他应该把这个工作下发给下面的人去做，所以Masters就是一个调度工作的，它是分配任务的工作，那么具体干活的是slave从节点也就是底层员工，这就是这么一个概念。

Masters有了，slave有了，这里面还有一个角色那就是Client，可以认为Client就是个刚刚不是有一个领导吗？那么这个公司领导是乙方，这Client是甲方，这Client就相当于一个用户，这个用户既然我可以付现金，而你应该帮我提供一些服务，因为用户是上帝，所以用户的位置是摆的非常高的，所以尽可能要满足当前用户的需求，所以这Client就是上帝，那么上面那张图就是有三个角色，这很明确了：Masters,slave和Client,一个集群里面通常有这么三个角色，那么Client是分配任务的，首先这个任务分配给Masters说应该要做什么事情，你看看能不能帮我找个人或者帮我定位一个资源在哪，比如说Client要读一个数据，但是有很多的机器，用户不知道这数据在哪台机器里面，于是这时候Masters知道Client的请求了，Masters在这个机器里面就会有一个名单，通过这个名单查询出用户所需要的数据文件应该在哪台机器上，这时候Masters会告诉用户说你想要的数据是比如说在A这台机器上，那么这时Client就已经从Masters拿到了他想要读取数据的真正存在的位置，这个时候这个用户就开始请求真正的这个数据，用户就可以直接的跟sleep进行通信联系了，然后拿到用户所需要的数据，大概是这么一个过程，但是这里面还有很多很多的细节。

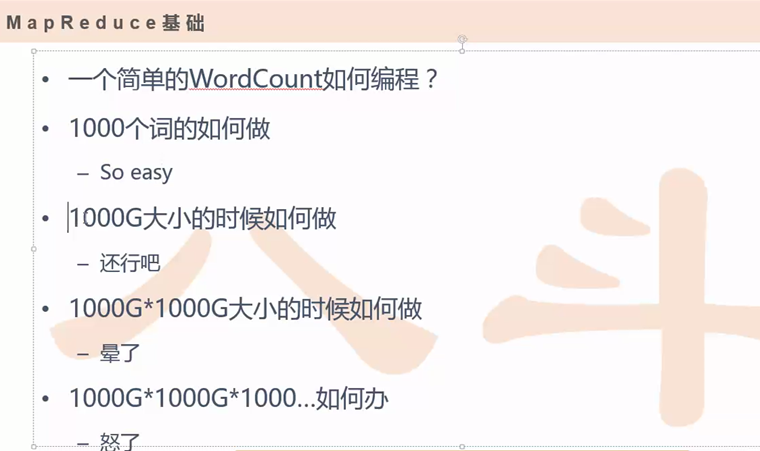
比如说：会发现这下图2有不同的颜色的四方块是代表着一个数据块一个block，在hadoop里面的最基本的数据单位是block。在block你会发现C1在第一台机器上有，在第二台机器上也有，这什么意思呢？也就是说有一个数据它可以帮你备份出多个数据，比如来了一个数据之后，我要往这个数据里面进行存储，那为了要保证这数据的有效性，比如这个数据放到这个机器了，万一 这个机器坏了怎办？这个时候就应该把这个数据拷贝成多个不同的一模一样的数据，并且这个数据都是分配到了不同的其他机器上去，当有一台机器出现故障了，没关系，其他机器还有这数据的副本，所以就可以保证了这数据的可用性。这就是一个副本的概念，还有很多关于这样的细节

图二：



1. 我们怎么去理解这个MapReduce呢？

举个例子：做一个简单的WordCount，什么是WordCount呢？比如说一本书，这本书里面有很多的词语，把这本书从前到后全部看一遍然后列出来每一个词语在这个书里面一共出现了多少次？那么这个问题就是WordCount的问题（如下图所示）



假设说这本书很短，这本书一共就1000个字，还没去做去重，那么这个任务交给计算机来进行处理就很简单了。

那假设说1000G大小的数据量交给计算机处理也还OK，也就是计算机的硬盘搞大一点，所以存这么大的数据不是问题。

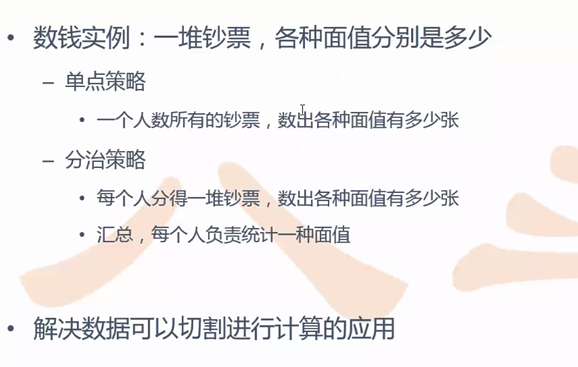
那假设说1000G\*1000G大小的数据量，那么这时候光靠一个硬盘肯定是不够的，所以很难解决。

再假设说1000G\*1000G\*1000。。。的数据量，那么就会崩了 ，那么怎么办呢？

这时候我们就需要MapReduce来搞定这些大数据量。

再举个例子：MapReduce分而治之思想

这是一个很经典的例子来提醒MapReduce的核心思想。（如下图所示）



有一张桌子，桌子上面洒满了很多的钞票，而且这个钞票很多，另外在这堆钞票里面面值很丰富，有100元有50元的有10元的面值的，这个时候感觉很杂乱，那么希望快速知道这个桌子上面这些钱的一个总的额度应该怎么办呢？那么光靠一个人数钱是很慢的，那么就邀请好几个人来一起帮忙数钱，比方说找103个人，其中有三个人保持不动，剩下100个人围住桌子然后每一个人去自己边上这个钱归拢到自己的面前，这时候每个人都不需要去数钱的，仅仅这100个人是归总的作用并不是去做计算，这个时候不仅要归总还要对这100个人提出要求把自己面前的这堆钱按照面值去分文别类整理整齐，就比如100面值的一堆，50元面值的一堆，10元面值的一堆，那么这100个人把这三种不同面值的钱摆放整齐，左边是100元面值的，中间是50元面值的，右边是10元面值的，然后剩下3个人过来，这时候该交给100个人一个任务，就是统一把这个100元的面值这一堆钱发给3个人中的=的A这个人，然后用50元面值的钱发给3个人中的B这个人，然后再把10元面值的钱发给3个人中的C这个人，然后这三个人就各自有了不同面值的钱，这个时候100个人的工作就完成了结束了，剩下3个人开始工作，这三个人不需要去做加法就比如（100+100），只需要去数这个钱的张数，最后就出三个数字，然后把这三个数字加起来就是这一整个桌子的全额度总数。

整个例子里面会发现它前后经过了两个过程，第一个过程是100个人，第二个过程是3个人，100人相当于就是做分解，把一整张桌子的钱去分解成100份，剩下这三个人就是做合并的作用。

相当于这个思想就是MapReduce的分而治之的思想，这个分而治之的思想的分治思想是理解MapReduce的核心。