最近几年一直参与大数据产品的研发，同时大数据产品在海量数据场景下其处理性能又是其主要的买点和突破，所以个人在这几年经常忙于如何对大数据产品进行性能上面的优化，而优化一般分为两部分，其一是，硬件和系统层面的观测，来发现瓶颈，并进行硬件或者系统级的调整，；其二是主要通过对软件使用方法的调整来实现优化。

## 硬件方面的监测

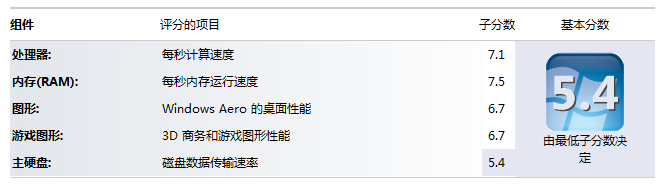


图1. Win7性能指数

关于硬件性能本身，个人觉得最好对性能的诠释就像图1大家比较熟悉的Win7性能指数所展示的一样，性能本身并在于其所长，而是在于其所短，就像图1里面那个5.4分主硬盘托了整体的后腿一样，**只要有短板存在，其他地方再强也收效甚微**，所以需要硬件的性能检测就是找出短板在那里，并且尽可能地找到应对的方法。

在硬件观测角度方面，主要通过以下四个维度来判断到底哪里是瓶颈，它们分别是CPU、内存、硬盘还有网络。

#### CPU利用率

首先，在讲检测CPU性能之前，我们可以通过这个“cat /proc/cpuinfo |grep "processor"|wc -l”命令来获取本机的核数（如果开了超线程，一个核可以被看作两个核），这样可以知道CPU利用率的上限是多少。

最常用CPU监测工具是TOP，当然TOP输出是一个瞬间值，如果想获取精确的数据，需要持续关注一段时间。

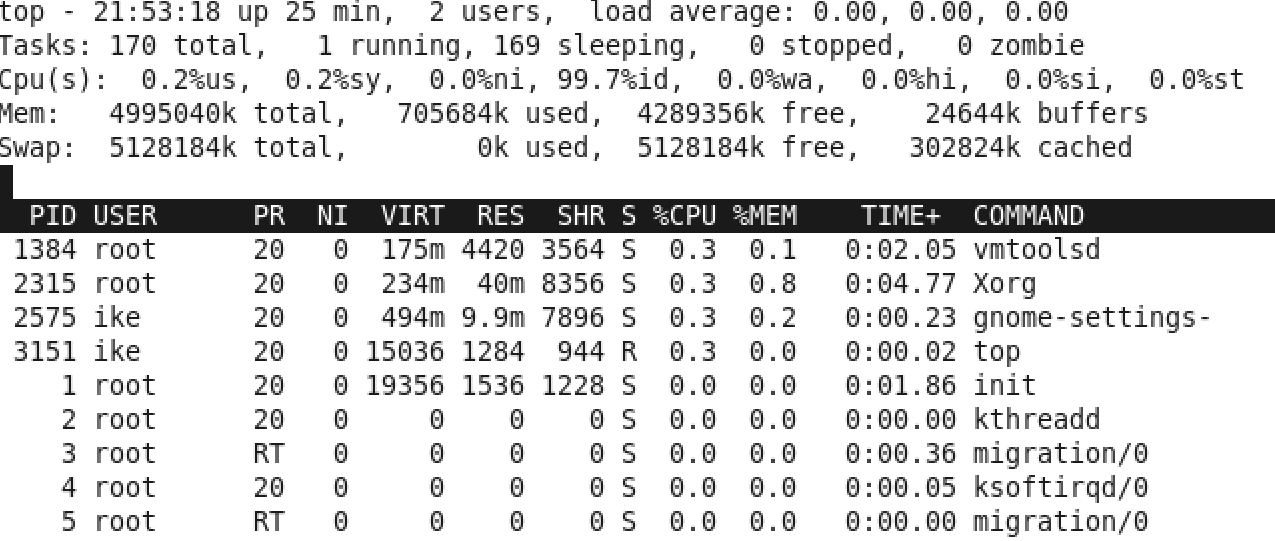


图2 TOP示例

TOP的使用主要看两个值，其一是总体使用值，其最大值是100%，就是图2第三行Cpu（s）,前面两个0.2%分别是用户态和内核态的利用率，而99.7%是CPU空闲率，从这个可以看出，本机的CPU部分基本是空闲的；其二可以看相关进程，看它的“%CPU”使用率，比如，Xorg这个GUI进程的占用率是0.3%，但是这里面的100%不是本机所有CPU的100%，而是单个核的100%。所以它的上限会是本机核数\*100%。

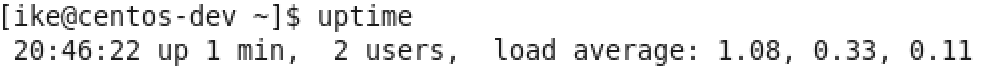


图3 uptime示例

因为TOP主要关注的是瞬时的值，如果要看一段时间的均值，这个时候可以用uptime这个命令，见图3，它除了可以显示当前总运行时，当前在线玩家，更重要的是可以显示1分钟、5分钟、15分钟的整机CPU的平均负载情况。

假设在平时监测的时候，如果经常碰到用满80%以上CPU资源的话，可以理解为CPU利用率高，在这种场景下大多数只能靠优化执行逻辑，才能提升效率。

#### 内存的监测

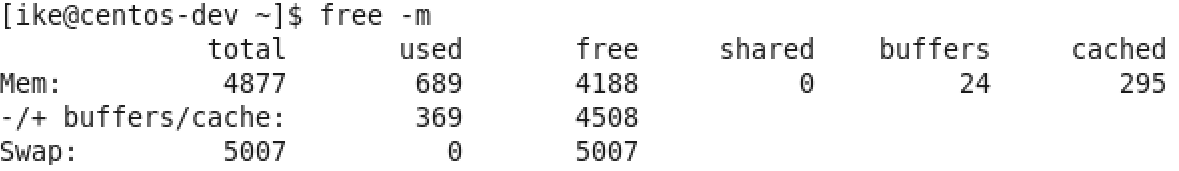


图4 free -m的示例

关于内存的监测，常用的命令是free -m，通过这个命令可以查看系统内存的使用情况。其中total，used和free都很好理解，通过这三列可以看出此时系统总内存，已经使用内存和没有被使用的内存，而cached这列则表示有多少内存已经被Page Cache占用，但当系统内存吃紧的时候，Page Cache会立即被回收并分配给请求内存的应用程序，所以Page Cache也可以被视为free。

还有下面的Swap分区，如果used数值比较高，说明内存非常紧张，系统已经动用交换区，同时IO开销也会增长非常明显。

当发现内存不够用的情况，可以考虑重启或者关闭那些占用很多内存的进程。

在这里稍微扩展一下Page Cache这个内存机制，因为这个机制对大数据挺重要的。一般在Linux系统上，利用默认系统I/O接口写入的文件块，会先在Page Cache上面有一个缓存，之后再写入到I/O设备上面，那么假设系统内存没有被占有满的话，在这种情况下，这个缓存会长时间保留，并不会被洗出内存，这样等下次程序访问到这些文件块的时候，肯定会访问Page Cache上面的那个版本，也就是直接访问内存，所以性能方面是内存级别的。

#### I/O性能的监测

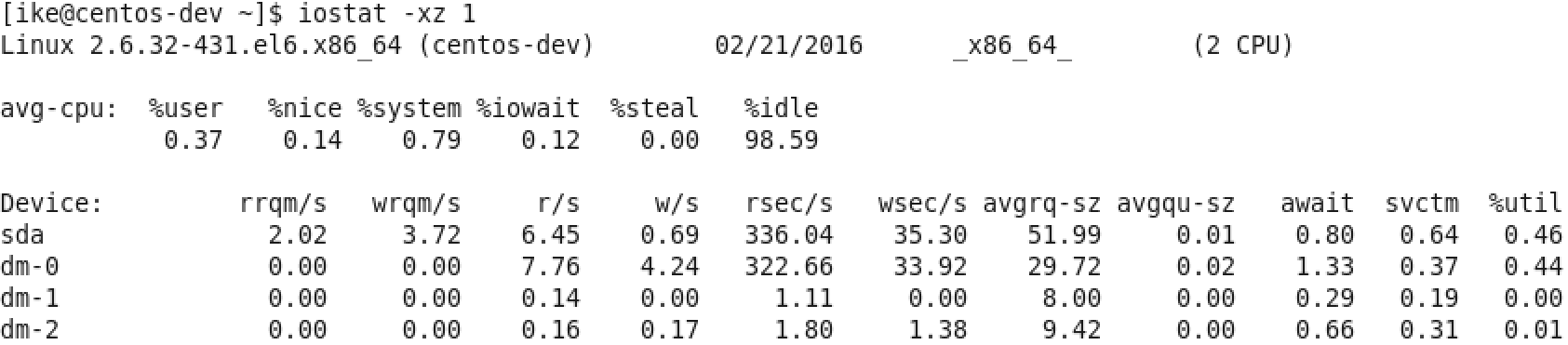


图5 iostat –xz 1示例

关于I/O性能，可以通过iostat这个命令来观察I/O的性能，具体见图5（sda是主硬盘），虽然参数比较多，但可以主要关注这两个参数：其一是await，它代表了IO操作的平均等待时间，单位是毫秒，这也是应用和磁盘之间操作所要消耗的时间，包括等待和实际的操作，如果这个数值大，说明I/O资源非常忙或者有故障；其二是%util，也就是设备利用率，数值如果超过60，所以利用率很高，并会影响I/O平均等待时间，如果到100，那就说明设备已饱和了，只能添加更多I/O资源。

#### 网络方面的监测

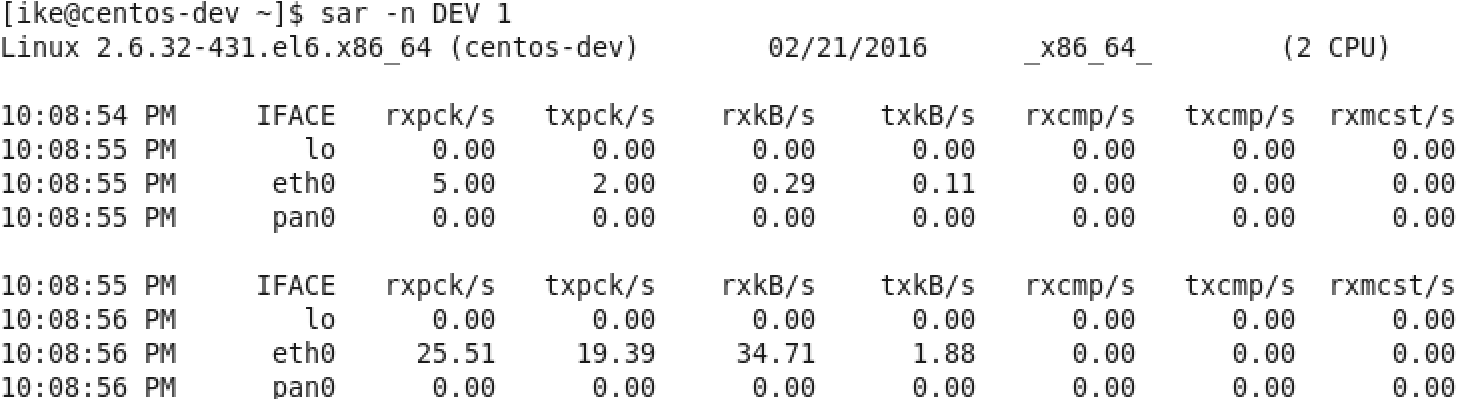


图6 sar –n DEV 1示例

在网络方面，使用的比较多的sar（System Activity Reporter）命令，如图6。这个命令可以查看网络设备的吞吐率,并在这个基础上，将吞吐量和硬件上限做对比，来判断网络设备是否已经饱和，假设以单张千兆网卡为例，如果“rxkB/s”和“txkB/s”两种相加超过100MB的话，说明网络已经接近饱和了。还有除了这个通过命令行来获取网络数据之外，还可以通过开源的nload的工具来进行监测，具体见下图:

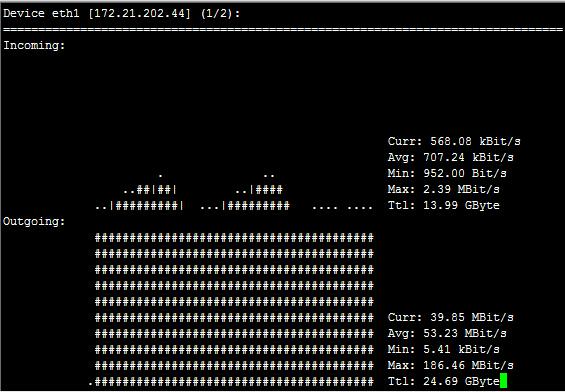


图7：nload示例

#### VMSTAT

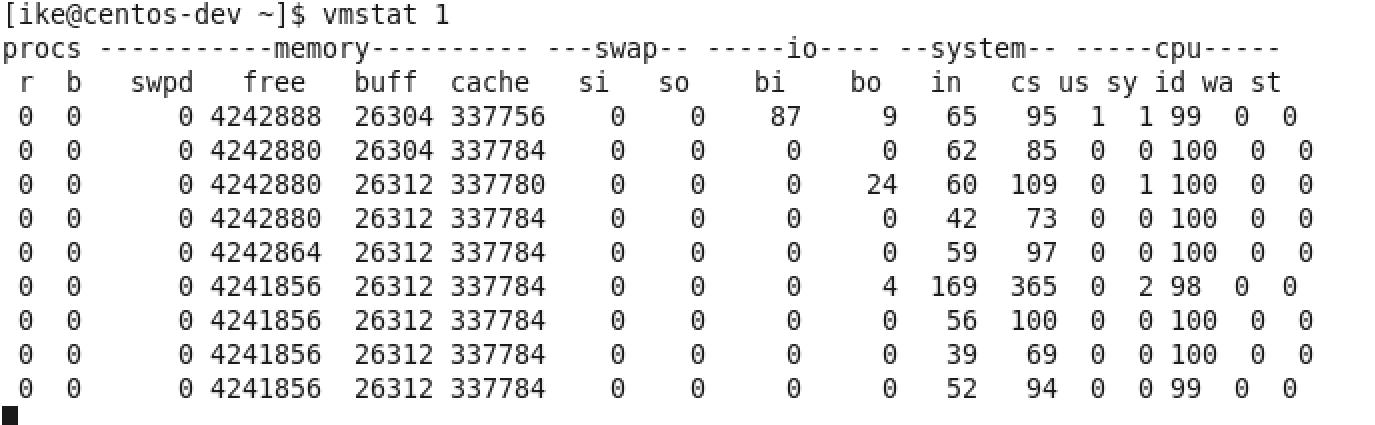


图8 vmstat 1示例

其实除了上面这些工具外，还有一个vmstat这个全能的命令，能监控硬件的方方面面，比如，如图8所示，Procs的“r”列，这个列显示正在等待CPU资源的进程数，这个数据比之前看的top和uptime更加能够体现CPU负载情况，并且这个数据不包含等待IO的进程。如果这个数值大于机器CPU核数，那么机器的CPU资源已经饱和。­

Memory部分的“free”，“buff”和“cache”列的作用和上面free作用类似，而“si”和“so”说明使用Swap的次数，如果这个数据不为0，说明Swap交换区已经在使用，也意味着物理内存已经不足。

Cpu部分也大体和TOP上面显示类似，但可以关注“wa”这列，其代表的是IO等待时间，如果数值大于0的话，可以判断I/O资源有争抢。

如果通过上面硬件方面的监测，发现了瓶颈，或者发现了有很多余量，可以通过下半部分的软件方面的优化来进行调整，如果软件方面也无能为力的话，那么只能通过购买和安装更多的硬件。

## 软件方面的优化

这个方面因为各个大数据产品的实现方式不同，并且需要优化点也不同，操作方式更是不同，所以在这里，主要提供一些方针供大家参考。

#### 写入优化

因为常见大数据产品的写入和传统关系型数据库是不同，传统关系数据库的写入是一行一行的写入，而常见大数据产品的写入是批量的写入，并且每次批量写入之后，都会生成新的数据文件，并且这个数据文件是不会被修改的。所以导入数据粒度小的话会导致很多细小文件产生，这样会导致更多的I/O操作，所以在使用大数据产品的时候，导入数据规模是越大越好，常见的规模在100MB以上为佳。

### 尽可能地并行

假设通过前面的硬件方面的测试方面，发现无论是CPU，内存，I/O还是网络，都没有遇到瓶颈，并且至少有20%潜力可挖，这个时候可以考虑尽可能地通过并行来提升性能，主要有两个方式：其一是每台机器上面部署更多的进程来压榨硬件资源；其二是提升单个进程的多线程数，这种方式比第一种更简单，风险也更低。总体而言，尽量使每台机器所使用到的线程数可以达到系统自身线程数的80%。

### 尽可能使用压缩和列存

对于一些新入门的工程师，也包括那些有很多传统关系数据库使用经验的专业DBA而言，大家都对列存比较一知半解，从而不敢使用。列存和传统行存相比，主要有两个比较大的区别：其一是数据不是按照行来存储，而且是将很多行的数据按列归属在一起，并存储 ，具体可以看图9；其二是一般行存的写入是一行行，而且列存是比较批量的，所以写入的数据库块会比较大，一般大于行存常见的8KB。基于我这几年的经验，列存在极大多数分析场景下，都能提升3倍以上的性能，除了那些需要遍历一半以上列的场景，因为通过列存不仅能够通过避免那些不要列的导入，这样能减少硬盘的I/O总量，并且由于列存本身数据是一个大块一个大块的存在，所以是硬盘I/O读取操作的次数也会减伤，这个对于硬盘I/O非常有利，因为本身硬盘I/O单次随机读取操作的成本非常高，和SSD相比，但是批量连续成本却非常优秀，当然如果使用SSD的话，性能会更优，在这个基础上，由于连续数据都归属于一列都比较类似，比如，性别，所以对其压缩的效果非常不错，一般在1比5左右，并且通过压缩节省的I/O远大于压缩和解压缩所带来CPU的损耗。这也导致就算所有数据全都在硬盘上，其性能的损失和所有数据在内存上面缓存相比，一般慢4到5倍左右，其他也不会特别亏，

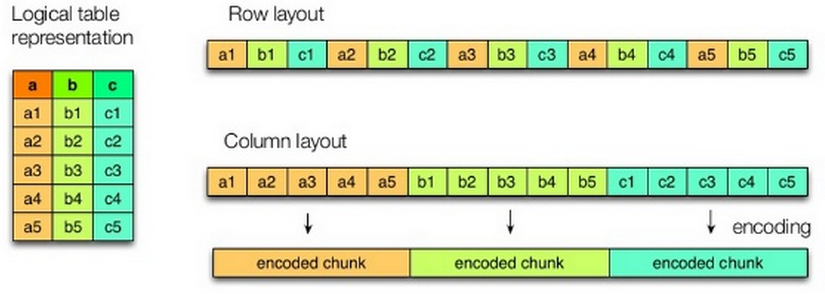


图9 列存

#### 善加利用Page Cache

在上半部分已经提到了， 利用好Page Cache可以达到最基础级别的内存计算的效果，当然和真正意义上的内存计算还是很大的距离。在性能测试的时候，这个优化是比较常见的。一般作法是，先通过命令“sync; echo 3 > /proc/sys/vm/drop\_caches”来清空page cache，之后跑一下比较简单，但又能加载所有相关数据的语句，比如，对每一列进行求总，这种做法的坏处是没有机会应对真实可能存在性能瓶颈，这对今后的实际运行会产生很多不可控的因素，因为真实业务场景肯定会比所预想到的场景更复杂。

#### 利用好分区特性

众所周知，最快SQL就是什么都不做的SQL，比如，“select 1”；当然在实际的操作过程中，肯定不会有类似“select 1”这样没有意义的操作。所以对于传统关系数据库而言，为了减少读取不必要的数据，一般会使用索引。但是对于大数据这样分析操作而言，索引这种机制太昂贵，而且收效甚微。

分析大数据应用常用的过滤数据的方式是分区，特别是按照时间来分区，因为一般时间是最合适分割大数据的维度，比如，数据按照月分区，这样如果查询只需要涉及到某月数据，那么其余十一个月数据可以立刻忽略，当然如果按日来分区的，效果可能会更好，但尽量避免因为粒度太小，导致写入文件过于碎片化的情况。

#### Join的优化

对于大数据的分析应用而言，Join操作是非常常见的，并且Join操作本身对硬件的短板也更敏感，特别是网络，因为大多数的分布式操作，每个数据节点可以独立地完成，但 Join经常需要来自其他节点数据才能完成本节点的执行，并且这个量可能很大，有的时候，一个节点执行所需要的数据远超本节点自带的数据，类似场景还有unique这样的去重操作，所以在调优方面消耗的功夫也最多。常见Join方式，主要有三种：其一是Broadcast广播，常用于大小表之间的Join，Join发起方会将小表的相关数据完整地分发到每个数据节点，之后当每个数据节点收到小表之后，会找其本地的大表数据来完成Join的，如图10，pages是小表，visits是大表，发起方将Pages这张小表分发到每个数据节点；其二是对小表Local化，这个机制本质上非常类似Broadcast，只是分发小表这个操作是做导入数据的时候自动完成，性能肯定比Broadcast更好，因为减少传输小表的网络消耗和等待时间，但是需要在创建表的时候，做一些额外的设置，这个机制在MPP数据是非常常见的，但是在Hadoop平台上面还是比较少见，因为其底层的HDFS比较强调硬件无关，地址透明，这个和Local化的思路是违背的；其三是Shuffle或者Partitioned Join机制，其常用于两张大表之间的Join。因为将大表都分发给每个节点肯定成本太高了，而且数据节点的内存不一定能放的下这么多数据，所以通过Shuffle洗牌机制，也就是将所有参与的Join表的相关部分按照某种机制均匀分发到各个节点，并且每个节点数据都是独立的，如图11所示，pages和visits都是大表，它们按照Join列Hash的值来进行再次分布，节点1有Join列为A-E的数据，之后依次类推，虽然成本很高，但是对于大表之间的Join是最合理和最可行的方法。

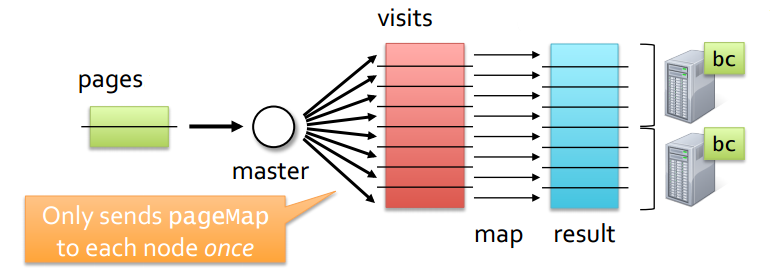


图10 Broadcast Join

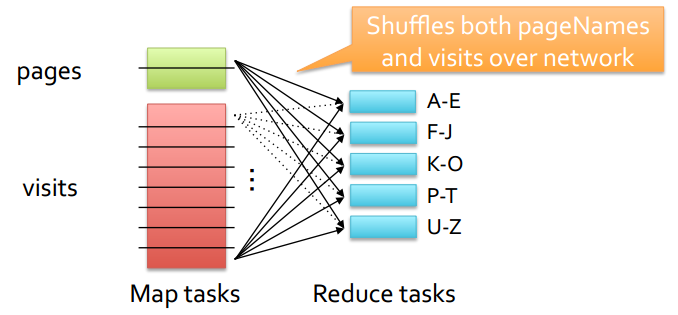


图11 Shuffle Join

介绍完Join机制之后，再深入一下Join的优化，也主要有三个方面：其一是在大表和小表摆放顺序要符合技术规范，这样能避免优化器将大表作为Broadcast表来进行分发；其二是开启或者执行预统计，也就是在查询之前，开启表的预统计，虽然预统计会耗费一点时间，但这样能够让优化器知道表的具体情况，从而做出合理的方案，即使之前表的顺序写错了，还有由于预统计会遍历数据，这样可以将数据预先加载到Page Cache上面；其三是选择合理的Join机制，也就是做好Broadcast和Shuffle之间的抉择，两个大表之间选择Shuffle，如果不是选择Broadcast，当然假如优化器能判断出是更好不过了，但当优化器出现问题的时候，可以通过人工输入一些提示符来帮助优化器来判断；

#### 多看Profile

介绍很多优化技术，但是这样技术都比较笼统，为了更好做优化，做某个产品优化，还是最好能多看看每次执行后的Profile，这样能对产品更深的理解。

因为大数据产品和技术比较多，并且每个产品和特色和设计都不同，所以在细节方面没有特别深入，但是的确有非常多的共性，所以通过硬件的监测，以及软件方面的优化，应该能把常见的大数据产品发挥到八成的功力。

参考资料：

1. 用十条命令在一分钟内检查Linux服务器性能http://www.infoq.com/cn/news/2015/12/linux-performance
2. 在 Linux/UNIX 终端下使用 nload 实时监控网络流量和带宽使用http://linux.cn/article-2871-1.html