再谈性能调试

性能调试无疑是系统管理工作中最具挑战性的工作，许多系统管理员热中于从性能调试中获取乐趣。不过要提醒的是，系统管理员的 90%以上，甚至更高比例的任务是性能调试以外的工作，所以建议大家还是把注意力更多的放在了解操作系统的基本原理上。

下面的内容摘译了一些讨论性能调试的文章，并结合自己的经验和理解，希望对大家有所帮助。注意：这些内容无法覆盖性能调试的全部，如果想系统地获得性能调试的知识，除了参看网站其他的相关文章，还可以看 IBM 随机文档光盘上的性能调试手册，再就是参加 IBM 昂贵的培训了。

性能调试的顺序

一般按以下步骤来优化系统：

1. 应用系统本身的设计
2. 数据库访问
3. 内存管理
4. 磁盘I/O调试
5. 调整CPU的使用情况
6. 避免数据库资源的竞争

其中(1)、(2)、(6)涉及到具体使用到的数据库和应用系统，其他的部分则与AIX的内存、磁盘I/O和CPU管理有关，对性能调试具有通用的意义。

内存和调页管理

当很多进程都要求更多的内存时，彼此之间就会发生竞争，一旦出现这种情况，系统会自动把暂时不用的内存页面调换到磁盘上，即发生调页。

控制缓冲区页面的调度策略

很多数据库系统使用文件系统来存放数据，在AIX中，往往采用大量的缓冲区保存最近读写的文件块来提高文件访问的效率，，而数据库系统本身也会把大量的数据缓冲区驻留在内存中以提供数据库系统的工作效率，这就造成了内存使用上的冗余，在内存不足时，由此产生的频繁调页行为将会严重地影响系统的性能。另外存放调页空间的磁盘也必将成为I/O访问的瓶颈，最终整个系统吞吐能力会降低到极其可怜的低点。

AIX中对缓冲区页面的调度策略是可以通过参数来控制的，需要强调的是，对操作系统的工作方式的任何修改必须加倍小心，否则

很有可能导致系统运行环境更趋于恶劣，甚至崩溃！

有四个与缓冲区页面的调度策略相关的AIX系统参数：

- minfree：最小的空闲页链表尺寸，一旦低于该值，系统开始换出一部分最近不访问的的内存页（称之为偷页），以填充空闲页链表。
- maxfree：最大的空闲页链表尺寸，一旦高于该值，系统停止偷页。
- minperm：用于文件I/O访问的最少缓冲区页数。
- maxperm：用于文件I/O访问的最多缓冲区页数。

修改这些参数，必须以*root*用户身份执行命令vmtune。注意：不同操作系统版本下的vmtune命令有可能不同，在使用前最好仔细阅读该命令的帮助，否则可能导致整个操作系统的失败。

minfree值的设置可依据有响应时间要求的程序代码段尺寸，至少空闲页链表中有足够的空闲内存页，以免装入该程序代码到内存时，不需要先偷页腾出空间。**maxfree**值必须大于max(8, **maxpgahead**)（意思是取maxpgahead和8之间的大者）。例如：你的**minfree**值设置为128，而**maxpgahead**值为16，那么你可以执行命令：

```
/usr/lpp/bos/samples/vmtune -f 128 -F 144
```

把**minfree**设为128，而**maxfree**设为144。

调整 AIX 文件缓冲区

AIX缓冲区的作用在于减少使用文件时的磁盘访问。如果设置得太小，会导致磁盘过于繁忙，甚至有个别磁盘达到饱和的状态；如果设置得太大，则会浪费宝贵的内存资源。

缓冲区的大小可以通过设置**minperm**和**maxperm**参数值来调整。一般来说，如果缓冲区的命中率太低（低于90%，可用sar –b命令查看），可以增加**minperm**值；如果缓冲区的命中率对系统并不是很关键，那么可以通过减少**minperm**值来增加可用的物理内存。

AIX提供一种松散的方式来控制内存中不同类型页面的比率。主要有两种内存页：存放文件数据的内存页（通常称之为文件型的内存页）和存放程序代码及所使用内存空间的内存页（通常称之为计算型的内存页），这两种内存页的比率控制也是通过设置**minperm**和**maxperm**参数值来实现，具体说明如下：

- 如果文件型内存页在实存中所占的比例低于**minperm**值时，调页算法在进行偷页时，将不考虑（某类型内存页）重新调入的频率，偷页的类型包括文件型和计算型。
- 如果文件型内存页在实存中所占的比例高于**maxperm**值时，调页算法在进行偷页时，只选择文件型的内存页。
- 如果文件型内存页在实存中所占的比例介于**minperm**和**maxperm**两值之间时，通常内存管理系统偷页时只选择文

件型的内存页，但如果文件型内存页的重新调入（即文件中的内容再次被访问）频率高于计算型时，偷页也会包括计算型的内存页。

minperm和maxperm的缺省值分别为20和80，表示20%和80%的比例值，用命令：

vmtune –p 5 –P 20

把**minperm**值改为5% ，**maxperm**值改为20%

如果数据库文件使用裸设备，由于AIX的文件缓冲区仅仅对文件系统有用，因此可以把**minperm**和**maxperm**两个参数设得很低（如分别为5%和20%），这样数据库系统可以分配到更多的内存空间。

分配足够的调页空间

一旦调页空间不足，往往导致系统反应令人无法忍受的慢以至于完全挂起。 虽然AIX可以动态地增加调页空间，但合理的设置调页空间的大小仍然十分重要，这要根据物理内存大小和应用系统需求来综合来考虑。lsps命令可以用来查看调页空间的使用情况，vmstat则可以用来监测系统调页的活动情况。

大多数情况下，调页空间的大小设为物理内存的2~3倍比较合适，对于内存大于1GB的高配置系统，调页空间设为物理内存的1.5

倍就基本满足需求了。

控制调页活动

持续过度的调页表明内存负担过重，这时你应该：

- 尽量避免。除非系统所配置的存储系统非常快，以至于系统调页的速度大大超过数据库系统正常的数据读写（如果数据库系统内存缓冲区较小，能够放在缓冲区的数据随之减少，所以数据库系统会按需读写数据；如果数据库系统缓冲区设得较大，导致物理内存不足时，操作系统将进行调页以满足内存需求；这样，如果调页速度快，可以扩大数据库缓冲区的尺寸，减少数据库系统自行的数据读写）

- 把有限的内存资源用到刀刃上。
- 内存确实不足，按下面的优先级顺序分配内存：
  - 操作系统和数据库系统核心；
  - 用户应用程序；
  - 数据库日志缓冲区；
  - 数据库代码缓冲池；
  - 数据库锁；
  - 数据库数据缓冲区。

附：下面的AIX系统命令可以查看调页的状态和统计信息：

```
$ vmstat -s
$ vmstat interval [repeats] and
$ sar -r interval [repeats]
```

磁盘 I/O 管理

导致磁盘I/O资源争夺现象的原因，可能是不正确的内存管理（引起相应的调页操作）；另外还有可能是由于不正确的数据库文件分布，引起I/O访问无法平均地分布到所有磁盘上。下面的介绍主要涉及在规划数据库时如何合理地安排，尽量平均地分散磁盘I/O。

索引与数据表分开存放

如果表和它的索引在同一块硬盘上，那么在执行基于索引的检索操作时，所有 I/O 都集中在同一块盘，为了避免这种情况发生，表和索引应该放在不同的硬盘上，用以分散硬盘的 I/O 负担。

数据库日志放在专门的硬盘上

如果数据库中频繁地进行数据插入和修改操作，那么把数据库日志放在专门的硬盘上能够很大程度上提高数据库系统的性能。如果把数据库日志放在裸设备上，可以进一步提高性能，这是因为：

- 日志文件总是按顺序进行读写，最适合裸设备的特性（裸设备有时相对于块设备也叫字符设备）。

- 日志文件的尺寸总是固定，这样能最小化裸设备的管理开销。

热点文件移到别的硬盘上

把经常被访问的“热点”文件移到相对负担较轻的硬盘上，以平衡 I/O 访问。通常的做法包括，整个文件移动到别的硬盘上，或者把该热点文件按条块化分散到多块硬盘上（RAID-0）。

减少对热点文件的 I/O 访问

如果硬盘上只有一个热点文件，针对该文件有大量 I/O 请求，

那么不管把该文件移动到哪块硬盘，都会使该块硬盘成为系统的瓶颈，这时解决的方法就只有从逻辑角度，即从数据表、索引角度来进行调整。例如：把存放在热点数据文件中（ORACLE 中叫 TABLESPACE，INFORMIX 中叫 DBSPACE）的多个表移到别的数据文件中。如果热点文件中只有一个热表，那么只有通过把数据条块化才能解决了。

AIX 逻辑卷管理

AIX 的逻辑卷管理（LVM）提供数据条块化功能，可以把数据均匀地分布在多块硬盘上，减少磁盘资源的竞争，分散硬盘 I/O 访

问，从而提高系统的整体性能。

- 设计条块化的逻辑卷

  定义一个条块化的逻辑卷前，需要确定有关的属性：

  1. 参与的硬盘：至少两块硬盘，硬盘上的 I/O 访问要尽可能的少，条件许可的话，最好由不同的适配卡驱动。
  2. 条块单元大小：可以设成从 2KB 到 128KB 之间的值（2 的 n 次方），大多数应用环境下设为 32KB 或 64KB 比较合适。
  3. 逻辑卷大小：物理分区（PP）数必须是参与硬盘的整数倍

- 建议的条块化逻辑卷参数

  下面的建议参数通常能最大化串行读写的性能

  1. 条块单元大小：64KB
  2. **minpgahead**值：2（最少的预读页数）
  3. **maxpgahead**值：参与硬盘个数的16倍（最多的预读页数），这样最多的预读数据值就等于条块单元大小（64KB）和硬盘个数的乘积（这是因为AIX中存储管理操作的页大小为4KB），结果就是在理想的情况下，预读将涉及每块硬盘上的一个条块单元。
  4. **maxfree**值：根据**maxpgahead**值变化（参见前面的内容）。

- 其他的考虑事项

虽然调整LVM所能起的性能提升程度与应用系统的类型有关，一般来，对于决策支持系统（DSS）效果明显，对于联机事务处理（OLTP）系统或混合型往往也能有很大的提高。

- 避免把不同的数据文件放在同一块硬盘上

如果硬盘上的分区有可能同时被访问，那么应该避免把不同的数据文件放在同一块硬盘。例如：由于数据文件和日志文件总是同时存取，所以做条块化时它们应该放在不同的硬盘组中，否则会导致磁盘磁头频繁移动，如果同时访问到的物理分区相距很远，情况将更加恶劣，系统的性能会受到严重影响。

异步I/O

异步I/O（简称AIO）能把多个I/O操作集中起来以提高I/O子系统的吞吐量。不过AIO能够高效率工作的前提是数据很合理地分布在不同的硬盘上，AIX的逻辑卷管理（LVM）及条块化支持就能满足AIO的效率要求，通过数据的条块化，可以避免应用程序对磁盘资源的竞争，从而很大地提高数据库系统的性能。

AIX版本4下，支持各种类型的AIO，不论数据库文件是基于文件系统还是基于裸设备。一旦发出AIO调用，应用程序会继续执行，对于基于文件系统、VSD（虚拟共享盘的英文缩写，SP上的一项技

术）或HSD（HASH共享盘的缩写，类似VSD）的AIO请求，此时称为*server*的核心进程(*kproc*)负责从队列中接收请求并进行处理，只到完成该I/O请求，*server* 的个数限制了能够并发处理的AIO请求个数。AIX版本4下，基于裸设备（非VSD和HSD）的AIO请求，不经由*server*核心进程处理，而是操作系统核心直接处理。

*server*的个数可以通过SMIT菜单进行设置，具体的SMIT菜单路径如下：

smit -> Devices -> Asynchronous I/O

-> Change/Show Characteristics of Asynchronous I/O

-> {MINIMUM|MAXIMUM}

或者直接：smit aio

其中最小值（minimum）是系统启动时的server个数，最大值（maximum ）是允许用来响应并发请求的最大server个数。缺省值**minservers**为1，而**maxservers**为10，对数据库系统来说这样的设置通常太小，建议值为：

- **maxservers=10*（AIO需并发处理的硬盘个数）**
- **minservers= maxservers/2**

RAID 技术应用

RAID5 技术可以提高串行读的性能，但降低了整体的写性能（因为多一个写校验盘的过程），所以该技术通常用于写数据较少的应用系统，否则性能会比非 RAID 的环境还差。

RAID 0/1 又称为条块化/镜像技术，总是能带来性能的提高，硬件上的 RAID0/1 比 AIX 或数据库级别的条块化/镜像技术效率更高（这是肯定的，因为减少了额外的 CPU、内存开销）。

RAID 7通过内嵌的硬件操作系统提供异步I/O、多通道等功能，因此相较其他的RAID技术，它的性能最好。

滞后写

为了减少内存脏页（被修改过与磁盘上不一致的页面）的数目，同时最小化磁盘碎片的产生，从而提高写性能（脏页过多，需要频繁地写回磁盘，自然影响性能；磁盘碎片对性能的影响应该不用多说了），文件系统把每个文件分成 16KB 大小的块。只有当进程开始写下一 16K 块中的第一个字节时，才把本块写回到磁盘上，这个过程称为滞后写。滞后写的块大小是可以通过 vmtune 命令定义的，执行命令：

```
vmtune -c 8
```

把滞后写的缓冲设为8*16KB；

执行命令：

```
vmtune -c 0
```

关闭滞后写功能，由于数据库系统本身对数据写盘做了控制，

所以有时关闭滞后写功能，能够提高一点系统性能。

串行预读

虚存管理系统（VMM）能够猜测目前的进程是否正在访问一个串行文件（这样的文件总是一个字节接一个字节的访问）。这是通过观察文件访问的模式实现的，当进程同时访问两个连续的页面，VMM就假设该进程还将继续串行地访问后续的页面，于是把后续的页面预先读入，这样当进程真的需要它们时，就能最快速地得到它们。VMM 预读功能可以通过下面两个阀值进行控制：

- **minpgahead**值：VMM认为进入串行读模式最少的连续访问页面数；
- **maxpgahead**值：VMM最多预读的页面数；

这两个值的缺省值分别为2和8，对于条块化逻辑卷，**maxpgahead**值应该设得大些，以满足串行读的需求。你可以用vmtune命令修改这两个参数的设置。执行命令：

```
vmtune -r 512 -R 1024
```

把**minpgahead**设为512页，**maxpgahead**设为1024页（注意：这两个值应该设成2的次方）。

I/O Pacing

I/O pacing是AIX提供的一项技术，用以限制对一个文件同时执

行的I/O请求的个数，这可以避免大规模的I/O请求耗尽CPU时间，使得交互式或CPU需求大的应用的响应时间有所保障。I/O pacing是通过**high-water mark**和**low-water mark**两个参数来实现控制的，但对一个文件执行的I/O请求达到high-water mark时，所有对该文件写操作的进程将被挂起，只到执行的I/O请求回落至low-water mark时，才重新被唤醒执行。

high-water和low-water marks两个参数可通过smit命令来修改，遵循下面的SMIT路径，可以进入修改界面：

smit

->System Environment

-> Change/Show Characteristics of Operating System

具体值的设置可以边试边改，一般high-water设为33，low-water设为24作为开始比较合适。如果想关闭I/O pacing功能，把这两项参数都设为0。

磁盘内部分布位置的考虑

AIX 上建逻辑卷时可以选择存放的位置，对几个经常访问的热点逻辑卷在设定位置尽可能相连，以减少磁盘的磁头移动和寻道时间，从而提高性能。

CPU 调度和进程优先级设定

CPU 是系统另一个存在潜在竞争的资源，虽然 AIX 核心在大多数时间内可以高效地分配 CPU，但多个进程争夺 CPU 时间片是客观存在的（任何一个多用户、多进程的操作系统都存在这种竞争）。

修改进程运行的时间片大小

CPU 时间片的缺省大小是 10ms，它可以通过执行 schedtune 命令来修改。对一个需要长时间运行的程序（不会中途因 I/O 操作挂起），长时间片意味着较少的时间片切换，当时间片切换时，要涉及相关上下文数据的卸载和装入，这都需要消耗 CPU 时间，所以长时间片能够通过减少不必要的 CPU 开销而提高性能；但长时间片也带来副作用，特别是在单 CPU 的系统中，很多进程不得不等待当前进程释放 CPU 资源，所以系统的响应时间可能变得很长。

缺省 10ms 的的时间片设置适合于大多数的应用系统，如果系统的运行队列中等待的任务很多，而且大多数的程序运行时间较长，那么可以考虑用下面命令增加时间片的大小：

```
# /usr/samples/kernel/schedtune -t n
```

n可以设为0、1、2、3等等（相应于10 ms、20 ms、30 ms、40 ms…）。

在 SMP 的机器上使用 CPU 绑定技术

在 SMP 的机器上，把进程绑定在一个 CPU 上，可以使高优先级别的应用分配到相对多的 CPU 时间，进程的上下文关联内容也能保存较长的时间，同时能够减少 Cache 缓存失效造成的损失（如果进程被调度到另一个 CPU 上运行，该 CPU 上的 Cache 缓存肯定不会有该进程的最近访问数据），从而提高性能。

AIX 版本 4 后，操作系统提供对 CPU 绑定技术的充分支持。你必须显式地执行 bindprocessor 命令把进程绑定到指定的 CPU 上，该进程的所有子进程自动继承绑定。要提醒的是，把一个进程绑定在某个 CPU 上执行，并不意味着其他的进程不能在该 CPU 上执行。另外一旦进程绑定后，它将不会在其他的 CPU 上运行，哪怕其他的 CPU 已处于空闲状态，这也 CPU 绑定技术潜在的副作用。

上面的内容从操作系统的角度论述了有关性能调试的一些尝试既注意事项，对具体的数据库软件和应用系统的优化，需参照有关软件附带的技术文档。