Abstract:

The purpose of this document is to describe how to monitor Linux operating systems for performance. This paper examines how to interpret common Linux performance tool output. After collecting this output, the paper describes how to make conclusions about performance bottlenecks. This paper does not cover how to performance tune the kernel. Such topics will be covered in part II of this series.

Topic Outline:

1. Tuning Introduction
2. CPU Terminology
3. CPU Monitoring
4. Kernel CPU Thread Scheduling

1.0 Tuning Introduction

Performance tuning is the process of finding bottlenecks in a system and tuning the operating system to eliminate these bottlenecks. Many administrators believe that performance tuning can be a "cook book" approach, which is to say that setting some parameters in the kernel will simply solve a problem. This is not the case. Performance tuning is about achieving balance between the different sub-systems of an OS. These sub-systems include:

- CPU
- Memory
- IO
- Network

These sub-systems are all highly dependent on each other. Any one of them with high utilization can easily cause problems in the other. For example:

- large amounts of page-in IO requests can fill the memory queues
- full gigabit throughput on an Ethernet controller may consume a CPU
- a CPU may be consumed attempting to maintain free memory queues
- a large number of disk write requests from memory may consume a CPU and IO channels

In order to apply changes to tune a system, the location of the bottleneck must be located. Although one sub-system appears to be causing the problems, it may be as a result of overload on another sub-system.

1.1 Determining Application Type

In order to understand where to start looking for tuning bottlenecks, it is first important to understand the behavior of the system under analysis. The application stack of any system is often broken down into two types:

- **IO Bound** – An IO bound application requires heavy use of memory and the underlying storage system. This is due to the fact that an IO bound application is processing (in memory) large amounts of data. An IO bound application does not require much of the CPU or network (unless the storage system is on a network). IO bound applications use CPU resources to make IO requests and then often go into a sleep state. Database applications are often considered IO bound applications.
- **CPU Bound** – A CPU bound application requires heavy use of the CPU. CPU bound applications require the CPU for batch processing and/or mathematical calculations. High volume web servers, mail servers, and any kind of rendering server are often considered CPU bound applications.

1.2 Determining Baseline Statistics

System utilization is contingent on administrator expectations and system specifications. The only way to understand if a system is having performance issues is to understand what is expected of the system. What kind of performance should be expected and what do those numbers look like? The only way to establish this is to create a baseline. Statistics must be available for a system under acceptable performance so it can be compared later against unacceptable performance.

In the following example, a baseline snapshot of system performance is compared against a snapshot of the system under heavy utilization.

```
alpha -> vmstat 1
procs                      memory      swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy wa id
 1  0 138592  17932 126272 214244    0    0     1    18  109    19  2  1  1 96
 0  0 138592  17932 126272 214244    0    0     0     0  105    46  0  1  0 99
 0  0 138592  17932 126272 214244    0    0     0     0  198    62 40 14  0 45
 0  0 138592  17932 126272 214244    0    0     0     0  117    49  0  0  0 100
 0  0 138592  17924 126272 214244    0    0     0   176  220   938  3  4 13 80
 0  0 138592  17924 126272 214244    0    0     0     0  358  1522  8 17  0 75
 1  0 138592  17924 126272 214244    0    0     0     0  368  1447  4 24  0 72
 0  0 138592  17924 126272 214244    0    0     0     0  352  1277  9 12  0 79

alpha -> vmstat 1
procs                      memory      swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy wa id
 2  0 145940  17752 118600 215592    0    1     1    18  109    19  2  1  1 96
 2  0 145940  15856 118604 215652    0    0     0   468  789   108 86 14  0  0
 3  0 146208  13884 118600 214640    0  360     0   360  498    71 91  9  0  0
 2  0 146388  13764 118600 213788    0  340     0   340  672    41 87 13  0  0
 2  0 147092  13788 118600 212452    0  740     0  1324  620    61 92  8  0  0
 2  0 147360  13848 118600 211580    0  720     0   720  690    41 96  4  0  0
 2  0 147912  13744 118192 210592    0  720     0   720  605    44 95  5  0  0
 2  0 148452  13900 118192 209260    0  372     0   372  639    45 81 19  0  0
 2  0 149132  13692 117824 208412    0  372     0   372  457    47 90 10  0  0
```

Just by looking at the numbers in the last column (`id`) which represent idle time, we can see that under baseline conditions, the CPU is idle for 79% - 100% of the time. In the second output, we can see that the system is 100% utilized and not idle. What needs to be determined is whether or not the system at CPU utilization is managing.

2.0 CPU Terminology

The utilization of a CPU is largely dependent on what resource is attempting to access it. The kernel has a scheduler that is responsible for scheduling two kinds of resources: threads (single or multi) and interrupts. The scheduler gives different priorities to the different resources. The following list outlines the priorities from highest to lowest:

- **Hardware Interrupts -** These are requests made by hardware on the system to process data. For example, a disk may signal an interrupt when it has completed and IO transaction or a NIC may signal that a packet has been received.
- **Soft Interrupts -** These are kernel software interrupts that have to do with maintenance of the kernel. For example, the kernel clock tick thread is a soft interrupt. It checks to make sure a process has not passed its allotted time on a processor.
- **Real Time Threads -** Real time threads have more priority than the kernel itself. A real time process may come on the CPU and preempt (or "kick off) the kernel. The Linux 2.4

kernel is NOT a fully preemptable kernel, making it not ideal for real time application programming.

- **Kernel Threads -** All kernel processing is handled at this level of priority.
- **User Threads -** This space is often referred to as "userland". All software applications run in the user space. This space has the lowest priority in the kernel scheduling mechanism.

In order to understand how the kernel manages these different resources, a few key concepts need to be introduced. The following sections introduce context switches, run queues, and utilization.

2.1 Context Switches

Most modern processors can only run one process (single threaded) or thread at time. The *n*-way Hyper threaded processors have the ability to run *n* threads at a time. Still, the Linux kernel views each processor core on a dual core chip as an independent processor. For example, a system with one dual core processor is reported as two individual processors by the Linux kernel.

A standard Linux kernel can run anywhere from 50 to 50,000 process threads at once. With only one CPU, the kernel has to schedule and balance these process threads. Each thread has an allotted time quantum to spend on the processor. Once a thread has either passed the time quantum or has been preempted by something with a higher priority (a hardware interrupt, for example), that thread is place back into a queue while the higher priority thread is placed on the processor. This switching of threads is referred to as a **context switch**.

Every time the kernel conducts a context switch, resources are devoted to moving that thread off of the CPU registers and into a queue. The higher the volume of context switches on a system, the more work the kernel has to do in order to manage the scheduling of processes.

2.2 The Run Queue

Each CPU maintains a run queue of threads. Ideally, the scheduler should be constantly running and executing threads. Process threads are either in a sleep state (blocked and waiting on IO) or they are runnable. If the CPU sub-system is heavily utilized, then it is possible that the kernel scheduler can't keep up with the demand of the system. As a result, runnable processes start to fill up a run queue. The larger the run queue, the longer it will take for process threads to execute.

A very popular term called "load" is often used to describe the state of the run queue. The system load is a combination of the amount of process threads currently executing along with the amount of threads in the CPU run queue. If two threads were executing on a dual core system and 4 were in the run queue, then the load would be 6. Utilities such as `top` report load averages over the course of 1, 5, and 15 minutes.

2.3 CPU Utilization

CPU utilization is defined as the percentage of usage of a CPU. How a CPU is utilized is an important metric for measuring system. Most performance monitoring tools categorize CPU utilization into the following categories:

- User Time – The percentage of time a CPU spends executing process threads in the user space.
- System Time – The percentage of time the CPU spends executing kernel threads and interrupts.
- Wait IO – The percentage of time a CPU spends idle because ALL process threads are blocked waiting for IO requests to complete.

- Idle – The percentage of time a processor spends in a completely idle state.

2.4 Time Slicing

The timeslice2 is the numeric value that represents how long a task can run until it is preempted. The scheduler policy must dictate a default timeslice, which is not simple. A timeslice that is too long will cause the system to have poor interactive performance; the system will no longer feel as if applications are being concurrently executed. A timeslice that is too short will cause significant amounts of processor time to be wasted on the overhead of switching processes, as a significant percentage of the system's time will be spent switching from one process with a short timeslice to the next. Furthermore, the conflicting goals of I/O-bound versus processor-bound processes again arise; I/O-bound processes do not need longer timeslices, whereas processor-bound processes crave long timeslices (to keep their caches hot, for example).

2.5 Priorities

A common type of scheduling algorithm is priority-based scheduling. The idea is to rank processes based on their worth and need for processor time. Processes with a higher priority will run before those with a lower priority, while processes with the same priority are scheduled round-robin (one after the next, repeating). On some systems, Linux included, processes with a higher priority also receive a longer timeslice. The runnable process with timeslice remaining and the highest priority always runs. Both the user and the system may set a processes priority to influence the scheduling behavior of the system.

3.0 CPU Performance Monitoring

Understanding how well a CPU is performing is a matter of interpreting run queue, utilization, and context switching performance. As mentioned earlier, performance is all relative to baseline statistics. There are, however, some general performance expectations on any system. These expectations include:

- **Run Queues** – A run queue should have no more than 1-3 threads queued per processor. For example, a dual processor system should not have more than 6 threads in the run queue.
- **CPU Utilization** – If a CPU is fully utilized, then the following balance of utilization should be achieved.
    - 65% – 70% User Time
    - 30% - 35% System Time
    - 0% - 5% Idle Time
- **Context Switches** – The amount of context switches is directly relevant to CPU utilization. A high amount of context switching is acceptable if CPU utilization stays within the previously mentioned balance

There are many tools that are available for Linux that measure these statistics. The first two tools examined will be `vmstat` and `top`.

3.1 Using the vmstat Utility

The `vmstat` utility provides a good low-overhead view of system performance. Because `vmstat` is such a low-overhead tool, it is practical to keep it running on a console even under a very heavily loaded server were you need to monitor the health of a system at a glance. The utility runs in two modes: average and sample mode. The sample mode will measure statistics over a specified interval. This mode is the most useful when understanding performance under a sustained load. The following example demonstrates vmstat running at 1 second intervals.

```
alpha-> vmstat
procs                      memory      swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy wa id
 0  0 200560  88796  88612 179036    0    1     1    20  112    20  3  1  1 96
```

The relevant fields in the output are as follows

Field	Description
R	The amount of threads in the run queue. These are threads that are runnable, but the CPU is not available to execute them.
B	This is the number of processes blocked and waiting on IO requests to finish.
In	This is the number of interrupts being processed.
Cs	This is the number of context switches currently happening on the system.
Us	This is the percentage of user CPU utilization.
Sys	This is the percentage of kernel and interrupts utilization.
Wa	This is the percentage of idle processor time due to the fact that ALL runnable threads are blocked waiting on IO.
Id	This is the percentage of time that the CPU is completely idle.

3.1.1 Case Study: Application Spike

In the following example, a system is experiencing CPU performance spikes, going from completely idle to completely utilized.

```
alpha-> vmstat 1
procs                      memory      swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy wa id
 4  0 200560  91656  88596 176092    0    0     0     0  103    12  0  0  0 100
 0  0 200560  91660  88600 176092    0    0     0     0  104    12  0  0  0 100
 0  0 200560  91660  88600 176092    0    0     0     0  103    16  1  0  0 99
 0  0 200560  91660  88600 176092    0    0     0     0  103    12  0  0  0 100
 0  0 200560  91660  88604 176092    0    0     0    80  108    28  0  0  6 94
 0  0 200560  91660  88604 176092    0    0     0     0  103    12  0  0  0 100
 1  0 200560  91660  88604 176092    0    0     0     0  103    12  0  0  0 100
 1  0 200560  91652  88604 176092    0    0     0     0  113    27 14  3  0 83
 1  0 200560  84176  88604 176092    0    0     0     0  104    14 95  5  0  0
 2  0 200560  87216  88604 176092    0    0     0   324  137    96 86  9  1  4
 2  0 200560  78592  88604 176092    0    0     0     0  104    23 97  3  0  0
 2  0 200560  90940  88604 176092    0    0     0     0  149    63 92  8  0  0
 2  0 200560  83036  88604 176092    0    0     0     0  104    32 97  3  0  0
 2  0 200560  74916  88604 176092    0    0     0     0  103    22 93  7  0  0
 2  0 200560  80188  88608 176092    0    0     0   376  130   104 70 30  0  0
 3  0 200560  74028  88608 176092    0    0     0     0  103    69 70 30  0  0
 2  0 200560  81560  88608 176092    0    0     0     0  219   213 38 62  0  0
 1  0 200560  90200  88608 176100    0    0     8     0  153   118 56 31  0 13
 0  0 200560  88692  88612 179036    0    0  2940     0  249   249 44  4 24 28
 2  0 200560  88708  88612 179036    0    0     0   484  254    94 39 22  1 38
 0  0 200560  88708  88612 179036    0    0     0     0  121    22  0  0  0 100
 0  0 200560  88708  88612 179036    0    0     0     0  103    12  0  0  0 100
```

The following observations are made from the output:

- The run queue during the spike goes as high as 3, almost passing the threshold.
- The percentage of CPU time in the user space goes to almost 90%, but then levels off.
- During this time, the amount of context switches does not increase significantly, this qould suggest that a single threaded application used a large amount of processor for a short period of time.
- It appears that the application batches all disk writes in one action. For one second, the CPU experiences a disk usage spike (wa = 24%)

3.1.2 Case Study: Sustained CPU Utilization

In the next example, the system is completely utilized.

```
# vmstat 1
procs                      memory      swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy wa id
 3  0 206564  15092  80336 176080    0    0     0     0  718    26 81 19  0  0
 2  0 206564  14772  80336 176120    0    0     0     0  758    23 96  4  0  0
 1  0 206564  14208  80336 176136    0    0     0     0  820    20 96  4  0  0
 1  0 206956  13884  79180 175964    0  412     0  2680 1008    80 93  7  0  0
 2  0 207348  14448  78800 175576    0  412     0   412  763    70 84 16  0  0
 2  0 207348  15756  78800 175424    0    0     0     0  874    25 89 11  0  0
 1  0 207348  16368  78800 175596    0    0     0     0  940    24 86 14  0  0
 1  0 207348  16600  78800 175604    0    0     0     0  929    27 95  3  0  2
 3  0 207348  16976  78548 175876    0    0     0  2508  969    35 93  7  0  0
 4  0 207348  16216  78548 175704    0    0     0     0  874    36 93  6  0  1
 4  0 207348  16424  78548 175776    0    0     0     0  850    26 77 23  0  0
 2  0 207348  17496  78556 175840    0    0     0     0  736    23 83 17  0  0
 0  0 207348  17680  78556 175868    0    0     0     0  861    21 91  8  0  1
```

The following observations are made from the output:

- There are a high amount of interrupts and a low amount of context switches. It appears that a single process is making requests to hardware devices.
- To further prove the presence of a single application, the us time is constantly at 85% and above. Along with the low amount of context switches, the process comes on the processor and stays on the processor.
- The run queue is just about at the limits of acceptable performance. On a couple occasions, it goes beyond acceptable limits.

3.1.3 Case Study: Overloaded Scheduler

In the following example, the kernel scheduler is saturated with context switches.

```
alpha-> vmstat 1
procs                      memory      swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy wa id
 2  1 207740  98476  81344 180972    0    0  2496     0  900  2883  4 12 57 27
 0  1 207740  96448  83304 180984    0    0  1968   328  810  2559  8  9 83  0
 0  1 207740  94404  85348 180984    0    0  2044     0  829  2879  9  6 78  7
 0  1 207740  92576  87176 180984    0    0  1828     0  689  2088  3  9 78 10
 2  0 207740  91300  88452 180984    0    0  1276     0  565  2182  7  6 83  4
 3  1 207740  90124  89628 180984    0    0  1176     0  551  2219  2  7 91  0
 4  2 207740  89240  90512 180984    0    0   880   520  443   907 22 10 67  0
 5  3 207740  88056  91680 180984    0    0  1168     0  628  1248 12 11 77  0
 4  2 207740  86852  92880 180984    0    0  1200     0  654  1505  6  7 87  0
 6  1 207740  85736  93996 180984    0    0  1116     0  526  1512  5 10 85  0
 0  1 207740  84844  94888 180984    0    0   892     0  438  1556  6  4 90  0
```

The following conclusions can be drawn from the output:

- The amount of context switches is higher than interrupts, suggesting that the kernel is having to spend a considerable amount of time context switching threads.
- The high volume of context switches is causing an unhealthy balance of CPU utilization. This is evident by the fact that the wait on IO percentage is extremely high and the user percentage is extremely low.
- Because the CPU is block waiting for IO, the run queue starts to fill and the amount of threads blocked waiting on IO also fills.

3.2 Using the mpstat Utility

If your system has multiple processor cores, you can use the `mpstat` command to monitor each individual core. The Linux kernel treats a dual core processor as 2 CPU's. So, a dual processor system with dual cores will report 4 CPUs available. The `mpstat` command provides the same CPU utilization statistics as `vmstat`, but `mpstat` breaks the statistics out on a per processor basis.

```
# mpstat –P ALL 1
Linux 2.4.21-20.ELsmp (localhost.localdomain)   05/23/2006

05:17:31 PM  CPU   %user   %nice %system   %idle    intr/s
05:17:32 PM  all    0.00    0.00    3.19   96.53     13.27
05:17:32 PM    0    0.00    0.00    0.00  100.00      0.00
05:17:32 PM    1    1.12    0.00   12.73   86.15     13.27
05:17:32 PM    2    0.00    0.00    0.00  100.00      0.00
05:17:32 PM    3    0.00    0.00    0.00  100.00      0.00
```

3.2.1 Case Study: Underutilized Process Load

In the following case study, a 4 CPU cores are available. There are two CPU intensive processes running that fully utilize 2 of the cores (CPU 0 and 1). The third core is processing all kernel and other system functions (CPU 3). The fourth core is sitting idle (CPU 2).

```
# mpstat –P ALL 1
Linux 2.4.21-20.ELsmp (localhost.localdomain)   05/23/2006

05:17:31 PM  CPU   %user   %nice %system   %idle    intr/s
05:17:32 PM  all   81.52    0.00   18.48   21.17    130.58
05:17:32 PM    0   83.67    0.00   17.35    0.00    115.31
05:17:32 PM    1   80.61    0.00   19.39    0.00     13.27
05:17:32 PM    2    0.00    0.00   16.33   84.66      2.01
05:17:32 PM    3   79.59    0.00   21.43    0.00      0.00

05:17:32 PM  CPU   %user   %nice %system   %idle    intr/s
05:17:33 PM  all   85.86    0.00   14.14   25.00    116.49
05:17:33 PM    0   88.66    0.00   12.37    0.00    116.49
05:17:33 PM    1   80.41    0.00   19.59    0.00      0.00
05:17:33 PM    2    0.00    0.00    0.00  100.00      0.00
05:17:33 PM    3   83.51    0.00   16.49    0.00      0.00

05:17:33 PM  CPU   %user   %nice %system   %idle    intr/s
05:17:34 PM  all   82.74    0.00   17.26   25.00    115.31
05:17:34 PM    0   85.71    0.00   13.27    0.00    115.31
05:17:34 PM    1   78.57    0.00   21.43    0.00      0.00
05:17:34 PM    2    0.00    0.00    0.00  100.00      0.00
05:17:34 PM    3   92.86    0.00    9.18    0.00      0.00

05:17:34 PM  CPU   %user   %nice %system   %idle    intr/s
05:17:35 PM  all   87.50    0.00   12.50   25.00    115.31
05:17:35 PM    0   91.84    0.00    8.16    0.00    114.29
05:17:35 PM    1   90.82    0.00   10.20    0.00      1.02
05:17:35 PM    2    0.00    0.00    0.00  100.00      0.00
05:17:35 PM    3   81.63    0.00   15.31    0.00      0.00
```

3.3 CPU Performance Tuning

The Linux 2.6 kernel does not provide any tunable parameters for the CPU subsystem. The best way to tune the CPU subsystem is to familiarize yourself with acceptable CPU usage percentages. If a CPU is overutilized, use commands like top and ps to identify the offending application process. That process will need to be moved to another system or more hardware resources must be dedicated to running the application.

The following top output displays CPU utilization information for a system running StrongMail MTA servers. This system is at the acceptable CPU usage limit. Any more StrongMail MTA processes would require additional CPUs.

```
# top

Tasks: 102 total,   5 running,
  97 sleeping,   0 stopped,m 406m  69m R 89.6 10.0  26:15.64 strongmail-sm
   0 zombie
Cpu(s): 51.8% us, 47.8% sy,
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND

2717 strongma  18 0  799m 703m 68m R 12.6 17.4  29:43.39 strongmail-sm
tp                                                     :03.85 strongmail-smtp
2663 strongma  25   0  457m 450m 68m R 36.3 11.1  30:26.71 strongmail-sm
tp
2719 strongma  20   0  408m 406m  69m R 63.6 10.0  26:17.55 strongmail-sm
--More--(74%)4k free,  1543392k m 2428 S  9.1  0.3   2:08.91 strongmail-psto
cached
 2702 strongma  15   0 13688  10m 1788 S  0.0  0.2   0:00.97 strongmail-logp
```

4.0 Virtual Memory Terminology

Virtual memory uses a disk as an extension of RAM so that the effective size of usable memory grows correspondingly. The kernel will write the contents of a currently unused block of memory to the hard disk so that the memory can be used for another purpose. When the original contents are needed again, they are read back into memory. This is all made completely transparent to the user; programs running under Linux only see the larger amount of memory available and don't notice that parts of them reside on the disk from time to time. Of course, reading and writing the hard disk is slower (on the order of a thousand times slower) than using real memory, so the programs don't run as fast. The part of the hard disk that is used as virtual memory is called the swap space.

4.1 Virtual Memory Pages

Virtual memory is divided into pages. Each virtual memory page on the X86 architecture is 4KB. When the kernel writes memory to and from disk, it writes memory in pages. The kernel writes memory pages to both the swap device and the file system

4.2 Virtual Size (VSZ) and Resident Set Size (RSS)

When an application starts, it requests virtual memory (VSZ). The kernel either grants or denies the virtual memory request. As the application uses the requested memory, that memory is mapped into physical memory. The RSS is the amount of virtual memory that is physically mapped into memory. In most cases, an application uses less resident memory (RSS) than it requested (VSZ).

The following output from the ps command displays the VSZ and RSS values. In all cases, VSZ is greater than RSS. This means that although an application requested virtual memory, not all of it is allocated in RAM (RSS).

```
# ps –aux
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND

<snip>

daemon    2177  0.0  0.2  3352  648 ?        Ss   23:03   0:00 /usr/sbin/atd
dbus      2196  0.0  0.5 13180 1320 ?        Ssl  23:03   0:00 dbus-daemon-1 --sys
root      2210  0.0  0.4  2740 1044 ?        Ss   23:03   0:00 cups-config-daemon
root      2221  0.3  1.5  6108 4036 ?        Ss   23:03   0:02 hald
root      2231  0.0  0.1  2464  408 tty1     Ss+  23:03   0:00 /sbin/mingetty tty1
root      2280  0.0  0.1  3232  404 tty2     Ss+  23:03   0:00 /sbin/mingetty tty2
root      2343  0.0  0.1  1692  408 tty3     Ss+  23:03   0:00 /sbin/mingetty tty3
root      2344  0.0  0.1  2116  404 tty4     Ss+  23:03   0:00 /sbin/mingetty tty4
root      2416  0.0  0.1  1476  408 tty5     Ss+  23:03   0:00 /sbin/mingetty tty5
root      2485  0.0  0.1  1976  408 tty6     Ss+  23:03   0:00 /sbin/mingetty tty6
root      2545  0.0  0.9 10920 2336 ?        Ss   23:03   0:00 /usr/bin/gdm-binary
```

4.3 Paging and Swapping

Paging and swapping are two different actions taken by the kernel depending on system load. System paging is a normal activity. Memory pages are read and written to both the swap device and the file system. If the system is low on RAM, the kernel will first attempt to write pages to the swap device to free RAM. If the kernel can't free enough memory in time, it will start to swap whole processes. Whereas paging takes single memory pages, swapping takes entire memory regions associated with certain processes and writes them to the swap device.

4.4 Kernel Paging with pdflush and kswapd

There are two daemons that are responsible for synchronizing memory. When pages in memory are modified by running processes, they become "dirty". These dirty pages must be written back to either the disk or the swap device.

4.4.1 pdflush

The `pdflush` daemon is responsible for synchronizing any pages associated with a file on a filesystem back to disk. In other words, when a file is modified in memory, the `pdflush` daemon writes it back to disk.

```
# ps -ef | grep pdflush
root        28     3  0 23:01 ?        00:00:00 [pdflush]
root        29     3  0 23:01 ?        00:00:00 [pdflush]
```

The `pdflush` daemon starts synchronizing dirty pages back to the filesystem when 10% of the pages in memory are dirty. This is due to a kernel tuning parameter called `vm.dirty_background_ratio.`

```
# sysctl -n vm.dirty_background_ratio
10
```

4.4.2 kswapd

The kswapd daemon is responsible for freeing memory in the event of a memory shortage. If available system memory pages fall below a minimum free threshold, then the kswapd daemon starts scanning memory pages. It performs the following actions:

- If the page is unmodified, it places the page on the free list.
- If the page is modified and backed by a filesystem, it writes the contents of the page to disk.
- If the page is modified and not backed up by any filesystem, it writes the contents of the page to the swap device.

```
# ps -ef | grep kswapd
root        30     1  0 23:01 ?        00:00:00 [kswapd0]
```

4.5 Case Study: Large Inbound I/O

The vmstat utility reports on virtual memory usage in addition to CPU usage. The following fields in the vmstat output are relevant to virtual memory:

Field	Description
Swapd	The amount of virtual memory in KB currently in use. As free memory reaches low thresholds, more data is paged to the swap device.
Free	The amount of physical RAM in kilobytes currently available to running applications.
Buff	The amount of physical memory in kilobytes in the buffer cache as a result of read() and write() operations.
Cache	The amount of physical memory in kilobytes mapped into process address space.
so	The amount of data in kilobytes written to the swap disk.
si	The amount of data in kilobytes written from the swap disk back into RAM.
Bo	The amount of disk blocks paged out from the RAM to the filesystem or swap device.
Bi	The amount of disk blocks paged into RAM from the filesystem or swap device.

The following vmstat output demonstrates heavy utilization of virtual memory during an I/O application spike.

```
# vmstat 3
 procs           memory                    swap          io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 3  2 809192 261556  79760  886880  416    0  8244   751  426   863 17  3  6 75
 0  3 809188 194916  79820  952900  307    0 21745  1005 1189  2590 34  6 12 48
 0  3 809188 162212  79840  988920   95    0 12107     0 1801  2633  2  2  3 94
 1  3 809268  88756  79924 1061424  260   28 18377   113 1142  1694  3  5  3 88
 1  2 826284  17608  71240 1144180  100 6140 25839 16380 1528  1179 19  9 12 61
 2  1 854780  17688  34140 1208980    1 9535 25557 30967 1764  2238 43 13 16 28
 0  8 867528  17588  32332 1226392   31 4384 16524 27808 1490  1634 41 10  7 43
 4  2 877372  17596  32372 1227532  213 3281 10912  3337  678   932 33  7  3 57
 1  2 885980  17800  32408 1239160  204 2892 12347 12681 1033   982 40 12  2 46
 5  2 900472  17980  32440 1253884   24 4851 17521  4856  934  1730 48 12 13 26
 1  1 904404  17620  32492 1258928   15 1316  7647 15804  919   978 49  9 17 25
 4  1 911192  17944  32540 1266724   37 2263 12907  3547  834  1421 47 14 20 20
 1  1 919292  17876  31824 1275832    1 2745 16327  2747  617  1421 52 11 23 14
 5  0 925216  17812  25008 1289320   12 1975 12760  3181  772  1254 50 10 21 19
 0  5 932860  17736  21760 1300280    8 2556 15469  3873  825  1258 49 13 24 15
```


The following observations are made from this output:

- A large amount of disk blocks are paged in (bi) from the filesystem. This is evident in the fact that the cache of data in process address spaces (cache) grows.
- During this period, the amount of free memory (free) remains steady at 17MB even though data is paging in from the disk to consume free RAM.
- To maintain the free list, kswapd steals memory from the read/write buffers (buff) and assigns it to the free list. This is evident in the gradual decrease of the buffer cache (buff).
- The kswapd process then writes dirty pages to the swap device (so). This is evident in the fact that the amount of virtual memory utilized gradually increases (swpd).

5.0 Linux Virtual Memory Kernel Tuning

The Linux kernel contains a series of tunable parameters for the virtual memory subsystem. These parameters are accessible via the /proc interface. Linux provides the sysctl command as an administrator interface to the /proc filesystem and the ability to tune the VM subsystem. Some of these parameters are tunable while others are read only.

```
# sysctl –a | grep vm
vm.legacy_va_layout = 0
vm.vfs_cache_pressure = 100
vm.block_dump = 0
vm.laptop_mode = 0
vm.max_map_count = 65536
vm.min_free_kbytes = 512
vm.lower_zone_protection = 0
vm.hugetlb_shm_group = 0
vm.nr_hugepages = 0
vm.swappiness = 60
vm.nr_pdflush_threads = 2
vm.dirty_expire_centisecs = 3000
vm.dirty_writeback_centisecs = 500
vm.dirty_ratio = 40
vm.dirty_background_ratio = 10
vm.page-cluster = 3
vm.overcommit_ratio = 50
vm.overcommit_memory = 0
```

The following tunable parameters will be discussed as they are the ones that have maximum impact on the system.

5.1 laptop mode

Laptop Mode is an umbrella setting designed to increase battery life in lap-tops. By enabling laptop mode the VM makes decisions regarding the write-out of pages in such a way as to attempt to minimize high power operations. Specifically, enabling laptop mode does the following:

- Modifies the behavior of kswapd to allow more pages to dirty before swapping
- Modifies the behavior of pdflush to allow more buffers to be dirty before writing them back to disk

- Coordinates the activities of kswapd and pdflush such that they write to disk when the disk is active to avoid unneeded disk spin up activity, which wastes battery power.

5.2 overcommit memory

Overcommit memory is a value which sets the general kernel policy toward granting memory allocations. If the value in this file is 0, then the kernel will check to see if there is enough memory free to grant a memory request to a malloc call from an application. If there is enough memory then the request is granted. Otherwise it is denied and an error code is returned to the application. If the setting in this file is 1, the kernel will allow all memory allocations, regardless of the current memory allocation state. If the value is set to 2, then the kernel will grant allocations above the amount of physical ram and swap in the system, as defined by the overcommit ratio value (defined below). Enabling this feature can be somewhat helpful in

environments which allocate large amounts of memory expecting worst case scenarios, but do not use it all.

You can check to see how much memory you are using and how much you have free by using the free command. Run the free command when your system is running at the best performance. This will ensure that all applications have already taken their memory.

In the following output, the system only uses 110 MB of 256 MB of total swap.

```
# free
         total       used       free     shared    buffers     cached
Mem:     256044     110984     145060          0       4212      33820
-/+ buffers/cache:      72952     183092
Swap:       524280      17736     506544
```

You can check to see per process if your applications are using all of their virtual memory with the ps command. The following output displays how much RAM (RSS) sendmail is actually using.

```
# ps -aux | egrep 'RSS| sendmail'
USER       PID %CPU %MEM   VSZ  RSS TTY   STAT START   TIME COMMAND
smmsp     2108  0.0  0.9  6892 2436 ?     Ss   18:12   0:00 sendmail:
root      2100  0.0  1.0  7688 2668 ?     Ss   18:12   0:00 sendmail:
accepting connections
```

5.3 overcommit ratio

This tunable defines the amount by which the kernel will overextend its memory resources, in the event that overcommit memory is set to the value 2. The value in this file represents a percentage which will be added to the amount of actual RAM in a system when considering whether to grant a particular memory request. For instance, if this value was set to 50, then the kernel would treat a system with 1GB of RAM and 1GB of swap as a system with 2.5GB of allocatable memory when considering weather to grant a malloc request from an application.

5.4 dirty expire centisecs

This tunable, expressed in 100thsof a second, defines how long a disk buffer can remain in RAM in a dirty state. If a buffer is dirty, and has been in RAM longer than this amount of time, it will be written back to disk when the `pdflush` daemon runs. Applications not reliant on I/O can benefit from tuning this parameter up and thus decreasing the amount of interrupts generated by disk synchronization I/O requests from `pdflush`.

5.5 dirty writeback centisecs

This tunable, also expressed in 100thsof a second, defines the poll interval between iterations of any one of the `pdflush` daemons. Lowering this value causes a pdflush task to wake up more often, decreasing the latency between the time a buffer is dirtied, and the time it is written back to disk, while lowering it increases the poll interval and the sync-to-disk latency. Systems not generating I/O can benefit by tuning this up and decreasing the frequency of when `pdflush` runs.

5.6 dirty ratio

This value, expressed as a percentage of total system memory, defines the limit at which processes which are generating dirty buffers will begin to synchronously write out data to disk, rather than relying on the pdflush daemons to do it.

Increasing this value tends to make disk write access and response times faster for a for I/O intensive processes ONLY if enough I/O bandwidth is available. If this parameter is tuned up too high, it may cause an I/O bottleneck by sending too many requests at once.

5.7 page-cluster

This tunable defines how many pages of data are read into memory on a page fault. In an effort to decrease disk I/O, the Linux VM reads pages beyond the page faulted on into memory, on the assumption that the pages of data beyond the page being accessed will soon be accessed by the same task.

If the system is a sequential I/O system like a large scale database, then tuning up the page cluster size will reduce the amount of disk seeks and rotational operations needed to page data into the disk.

5.8 Swappiness

Swappiness lets an admin decide how quickly they want the VM to reclaim mapped pages, rather than just try to flush out dirty page cache data. The algorithm for deciding whether to reclaim mapped pages is based on a combination of the percentage of the inactive list scanned in an effort to reclaim pages, the amount of total system memory mapped, and the swappiness value.

By tuning swappiness up, the kernel will dedicate more resources to try to free existing memory pages in RAM, generating less I/O, but also increasing system CPU time. If your system is running at acceptable levels and you have 20% to 30% idle time, you may tune this parameter higher to dedicate more CPU time to freeing memory.

By tuning swappiness down, the kernel will spend less system CPU time freeing memory and generate more I/O. If your system is CPU intensive with relatively idle I/O, then tuning this parameter down will decrease CPU cycles and leverage the idle I/O channels. I/O is not CPU intensive or expensive.

References

Understanding Virtual Memory in RedHat 4, Neil Horman, 12/05
http://people.redhat.com/nhorman/papers/rhel4_vm.pdf