

Linux Performance Tuning



Training Material

Linux Performance Tuning - 1
© Applegie Solutions 2004-2008. Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



License

 Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by-nc-sa/3.0/>

© Applepie Solutions 2004-2008, Some Rights Reserved
Except where otherwise noted, this work is licensed under Creative Commons Attribution Noncommercial Share Alike 3.0 Unported

You are free:

- **to Share** – to copy, distribute and transmit the work
- **to Remix** – to adapt the work

Under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Noncommercial.** You may not use this work for commercial purposes.
- **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to <http://creativecommons.org/licenses/by-nc-sa/3.0/>
- Any of the above conditions can be waived if you get permission from the copyright holder.
- Nothing in this license impairs or restricts the author's moral rights.

Linux Performance Tuning - 2

© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Contents

- Performance Requirements
- Measuring Performance
- Benchmarks
- Microbenchmarks
- Performance Tuning Exercise 1
- Tuning Guidelines
- Hardware Performance
 - General
 - Processor
 - Memory
 - I/O
 - Storage
 - Network
- OS Performance
 - Filesystem Tuning
 - Filesystems
 - Other Filesystems
- Performance Tuning Exercise 2
- OS Performance
 - General
 - Virtual Memory
 - Drive tuning
 - Network Tuning
 - Core Settings
 - TCP/IP Settings
 - CPU related tuning
 - 2.4 Kernel tunables
 - 2.6 Kernel tunables
- Performance Tuning Exercise 3
- Performance Monitoring
 - CPU Utilisation
 - Memory Utilisation
 - I/O Utilisation
 - sar
 - Network

Linux Performance Tuning - 3

© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Contents

- Performance Tuning
 - Web Servers
 - File & Print Server
 - Database Servers
 - Java App Server
- Tuning C
- Tuning Java
- Application Profiling
- Performance Tuning Exercise 4
- In closing ...

Linux Performance Tuning - 4

© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Performance Requirements

- Establish performance targets
- Map targets to business requirements
- Performance comes at a cost
 - More performance comes at a much higher cost
 - Identify *low hanging fruit*
 - How good is good enough?
- Understand your system and its bottlenecks
 - Only ever tune bottlenecks!
- Measure system performance
 - Repeatable benchmarks
 - Tuning without measurement is pointless
 - Record before and after

Linux Performance Tuning - 5
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Establish Targets

The first step in performance tuning any system, application or service is to establish clear measurable targets. Setting out to “improve the performance” or “make it go faster” will rarely produce useful results. If you are tuning a web-based service, establish goals such as how many concurrent users you want to serve, what the average response time to a request should be, what the minimum and maximum response times should be. You may want to identify different tasks that your system performs that take more time and should thus have different targets.

Map your targets to business requirements

Ultimately, your performance criteria should map back to the business requirements for the system. There is little point in spending a lot of effort tuning a part of your system that exceeds the business requirements, except as an academic exercise in performance tuning. Business requirements for the system need to be clearly measurable use-cases such as “The system will deliver a document in 10 seconds from initial request” rather than “The system will be fast” or “The system will be responsive”.

The cost of performance

Performance comes at a price – be it in the time spent monitoring and investigating performance problems, the time spent tuning the system and measuring the performance improvements provided by these changes and the time spent monitoring the system for performance degradation. Additional hardware and software to address performance problems also cost money in terms of both the capital investment and time needed to deploy the systems. It is important to recognise “low-hanging fruit” when it comes to performance tuning and realise that improving the performance of the system by 10% may take minimal effort while a further 10% improvement may take 10 times as much effort. Going back to your requirements, identify “how good is good enough?” with regard to performance.

Understand your system and its bottlenecks

Before starting to tune your system you must develop a good understanding of the entire system. The only part of a system that should ever be tuned is the bottleneck. Improving the performance of any other component other than the bottleneck will have result in no visible performance gains to the end user. This applies whether you are tuning the hardware of the system, the operating system or the application code itself.

Performance measurement

Performance tuning without measurement is pointless. You should identify one or more benchmarks that mirror the actual usage of your system and run these after each performance related change you make to the system. Any performance improvement made without the benefit of “before and after” measurements is pure guesswork and will not result in a more performant system (unless you're really lucky).

Measuring Performance

- Baseline your system
 - Identify important metrics
 - Measurements should reflect performance requirements
- Monitor performance after tuning
- Monitor performance over time
 - Identify performance problems which only occur over time
 - Flag environment changes which cause problems
- Measurement tools
 - Depends on the metrics you want to measure
 - Standard Linux commands
 - Roll your own
 - Off the shelf benchmarks

Linux Performance Tuning - 6
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Baseline your system

Before you begin tuning the system, decide on one or more performance benchmarks for your system and establish what the initial baseline performance of the system is. The benchmarks and metrics you use will depend entirely on the system you are tuning and the degree to which you want to tune it. Common metrics include,

- How many concurrent users a system can handle performing some operation
- How many requests or transactions a system can perform in a specific unit of time
- How long it takes to perform a particular operation (display a page or serve a document)
- How long it takes to complete a batch run.

More detailed benchmarks and metrics will combine a number of these figures. The main priority is that the metrics reflect the actual usage and performance requirements of the system.

Monitor performance after tuning

Each time you make a tuning adjustment, re-run the benchmarks and see if the tuning has been successful (some of your changes will degrade system performance for reasons that may only become apparent after the event).

Monitor performance over time

Run your benchmarks periodically afterwards to ensure that the performance isn't degrading over time for reasons beyond your control. Perhaps your application only suffers performance problems when it has been running for a long time? Perhaps something happens on the system or network periodically which introduces performance problems? Historical data will let you correlate performance degradation with known changes to the environment.

Measurement Tools

The tools to measure your system performance are as varied as the metrics you want to measure. Simply benchmarks can be performed using the standard commands provided with the operating system if all you want to do is measure how long an application takes to run, how much processor time it uses, how much memory it uses or what kind of I/O it performs. If you want to perform more complex measurements you can either create your own benchmarking tool or use an off the shelf benchmark that has some correlation with your system.

Benchmarks

- Key characteristics
 - Repeatable
 - Consistent
- Allows you to identify,
 - System statistics that are constant from run to run
 - System statistics that change slightly from run to run
 - System statistics that change dramatically from run to run
- Component or Microbenchmarks
 - Measure standard OS characteristics
 - Useful for comparing systems and tuning OS components
- Application or Enterprise benchmarks
 - Specific benchmark for your system
 - Expensive to develop and maintain

Linux Performance Tuning - 7
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Benchmarks

A benchmark is a measurement or set of measurements which can be used to compare system performance. You can use a benchmark to compare a reference system to other systems or indeed to compare the performance of your system at a given time with a certain configuration to the performance of your system at another time with a different configuration. The key characteristics of benchmarks is that they be repeatable and consistent (so it is possible to run the same measurement multiple times and always get the same result for the same system configuration).

Why benchmark?

Benchmarks allow you to identify the system statistics that are strongly affected by configuration changes to your system and those statistics that are not affected by configuration changes. For example, you can compare the performance of your file system running on a SATA disk, a SCSI disk and a RAID storage array if you use an appropriate file system benchmark. You would not normally expect your network throughput to be significantly affected by a change to storage hardware (although it is best to verify this with an appropriate benchmark, there are edge cases where it could).

Microbenchmarks

There are a wide range of standard benchmarks available which measure standard system characteristics such as the performance of your file system, the performance of your network device, the throughput of web server software and so on. These benchmarks focus on a specific set of measurements relating to a particular operating system characteristic. They are known as *component benchmarks* or *microbenchmarks* and are a good way of comparing the performance of different hardware or operating system configurations. They don't normally tell you a lot about the performance of your application.

Application benchmarks

Ideally, you would create a benchmark specifically for your application which allows you to clearly see how the performance of your application changes with changes to the underlying system and changes to your application code. In practice, this will normally involve identifying some key metrics for your application and then simulating a normal user workload on the application. This can be expensive to initially create and maintain as your application evolves. If performance is a high priority you may be able to justify the expense or create a minimal, but maintainable benchmark which exercises key aspects of your application.

Microbenchmarks

- OS
 - Lmbench
 - Re-AIM 7
 - SPEC SDET
- Disk
 - Bonnie/Bonnie++
 - IOzone
 - Iometer
- Network Benchmarks
 - Netperf
 - iperf
 - SPEC SFS
- Java Application Benchmarks
 - Volanomark
 - SPECjbb
 - SPECjvm
- Database Benchmarks
 - TPC
 - OSDL tests
- Webserver Benchmarks
 - SPECweb
 - TPC-w
 - SPECjAppServer
 - ECPerf

Linux Performance Tuning - 8
© Applepie Solutions 2004-2008. Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Microbenchmark types

There are microbenchmarks available for all sorts of operating systems and standard pieces of software. Some of these are developed by dedicated organisations such as the Standard Performance Evaluation Corporation (SPEC)¹ and you must purchase a license to run their tests. The Transaction Processing Performance Council (TPC)² are a non-profit organisation dedicated to developing standard benchmarks for comparing database software and related systems. The Open Source Development Lab (OSDL)³ is another non-profit organisation dedicated to enhancing and extending Linux – they have developed some standard benchmarks for comparing Linux system performance.

Operating System Benchmarks

Lmbench (<http://www.bitmover.com/lmbench>) - an established set of free operating system benchmarks which measure the performance of basic operating system primitives and calls. Very useful for comparing the performance of your system to others and measuring the effects of tuning changes at the operating system level.

Re-AIM 7 (<http://sourceforge.net/projects/re-aim-7>) - a rework of the AIM benchmark suite for the Linux community. The original AIM benchmark was developed to measure operating systems that feature multi-user environments.

SPEC SDET (<http://www.spec.org/sdm91/#sdet>) - this is a controlled commercial benchmark based on a script of typical programs run by an imaginary software developer. It has been widely used for about 20 years to compare the performance of UNIX-based systems.

Networking Benchmarks

Netperf (<http://www.netperf.org/netperf/NetperfPage.html>) - a benchmark for measuring network throughput supporting TCP, UDP and Unix sockets. It also handles Ipv4 and Ipv6.

Iperf (<http://dast.nlanr.net/Projects/Iperf/>) - another tool for measuring network throughput. Note that both Iperf and Netperf require a client and a server on the network to perform testing.

Java Application Benchmarks – there are a number of microbenchmarks designed to measure the performance of a Java application (allowing different JVMs, JVM settings and underlying systems to be compared).

Database Benchmarks – benchmarks for comparing the performance of RDBMS software.

Webserver Benchmarks – benchmark for comparing webserver software throughput and so on.

1. <http://www.spec.org>
2. <http://www.tpc.org>
3. <http://www.osdl.org>

Performance Tuning Exercise 1

- 1) Download the **lmbench** benchmark and run it on your system.
- 2) Pair up with someone else and download and run the **Iperf** benchmark between your systems.

Linux Performance Tuning - 9
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



lmbench

The **lmbench** tool expects the *bk* command to be installed but it is not necessary to run the benchmarks. Make the following changes after downloading (applies to *lmbench2*, other versions may require other changes).

- Add a dummy *bk* command to your path e.g.

```
#!/bin/sh  
echo "dummy bk command called"
```

Edit the Makefile in *src* and replace the *bk.ver* target with the following

```
bk.ver:  
    echo "ignoring bk.ver target"
```

in order to compile and run *lmbench* without errors – read the README for full instructions on running *lmbench*. Results can be viewed by doing the following

```
cd results  
make
```

Iperf

Download the compile the *Iperf* command. One machine will need to run as server and another as client. You may need to co-ordinate with other users in the room. Investigate the effects of multiple people running the benchmark simultaneously.

Tuning Guidelines

- Dos:
 - Change one parameter at a time
 - Run your benchmarks after each change
 - Keep records of benchmark runs for comparison
 - Tune bottlenecks
 - Make the least expensive changes first
 - Try to understand your system and how it should respond to changes
- Don'ts:
 - Don't tune without benchmarks
 - Don't tune on systems that are in use
 - Don't assume a change makes things better without measurements

Linux Performance Tuning - 10
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



1. **Do change one parameter at a time** – it is impossible to quantify the impact of changing a parameter if you only measure performance improvements after making a number of changes. You may expect that each change will improve things but in practice, systems are complex enough that you may be degrading the performance with some of your changes or introducing unexpected behaviour.
 2. **Do run your benchmarks after each change** – there is no other way to determine if the change improves things. If running an entire benchmark is too time consuming, consider running an indicative subset of the measurements after *every change* and verify the overall system performance periodically with a full benchmark.
 3. **Do keep records of benchmark runs for comparison** – it is invaluable to be able to compare the performance of the system with the same system at different times (or indeed running on different types of hardware). Ideally you should keep as much raw data as is practical but at a minimum store the summary key metrics from the system.
 4. **Do tune bottlenecks** – tuning anything else is pointless. If your system is I/O bound, adding faster processors will deliver no improvement whatsoever to your system.
 5. **Do make the least expensive changes first** – identify the changes that will deliver the most results for the least effort in time and money. In a lot of cases, it may make more sense to move the system to faster hardware rather than trying to optimise the application code (it can be notoriously difficult to identify performance problems in code, furthermore it is very difficult to estimate the time required upfront to resolve a performance problem in code).
 0. **Do try to understand your system and how it should respond to changes** – You need to start your tuning somewhere. Consider how your application operates and use that understanding as a starting point for your tuning activity. If your system does a lot of reading and writing data then a good starting point for tuning is the storage hardware, the I/O tunables of the operating system, the file system and the database. This gives you a starting point, you should still benchmark to verify any assumptions.
-
1. **Don't tune without benchmarks** – modern systems are a complex combination of hardware, operating system code and application code. It is very difficult to fully anticipate all the interactions in such systems and small changes to some parts of the system may have unexpected cascading effects on other parts of the system – use benchmarks to identify and understand these effects.
 2. **Don't tune on systems that are in use** – the system needs to be in a known state before and during your tests. It is pointless trying to perform any tests on a system that is being actively used. If your system involves a network, make sure there are no other activities on the network that may be affecting your results (or if these activities are typical, ensure that your benchmark always simulates them).
 3. **Don't assume a changes makes things better without measurements** – assumptions are bad

Hardware Performance - General

- Scale up - buy more hardware
 - add more memory
 - add more/faster processors
 - add more storage
- Consider scaling out
 - increases system capacity rather than speed
 - only suitable to some applications
- Do you need a faster car or a bigger truck?
 - speed or capacity?
- Performance trade-offs
 - removing a bottleneck may cause another one

Linux Performance Tuning - 11
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



The easiest way to improve overall system performance is usually to buy new hardware.

Overall, modern hardware doubles in performance every few years. Replacing hardware is generally much cheaper than either extensive performance tuning exercises or rewriting parts of your application for better performance. Buying new hardware or *scaling up* can involve any of the following,

- Moving the system to an entirely new hardware platform
- Adding more memory to your existing hardware platform
- Adding more or faster processors to your existing hardware platform

Another approach to improving the performance of your system is to *scale out* rather than *scale up* that is, consider whether your system can benefit from running as a distributed system where the system uses multiple hardware platforms simultaneously. This is normally only possible if the application has been designed to operate in a distributed fashion – a lot of modern web applications are particularly suited to this. Scaling out allows you to grow your system as your performance demands increase although it does introduce more management complexity.

Understand the performance characteristics of the hardware you are buying – some hardware will give you *more capacity* while other hardware will give you *more throughput* while yet other hardware will give you both. Scaling out in particular will normally increase the number of simultaneous transactions your system can handle but it may not allow the system to process any single transaction any faster.

Be aware as you tune components of the system that the performance bottleneck may move to other components. For instance, moving to a faster network technology such as Gigabit Ethernet may improve your network throughput – but the additional network traffic will result in your processor(s) having to do more work. This may or may not be a problem depending on how processor-intensive your system is.

Hardware Performance – Processor

- Speed
 - Processors double in speed every 18-24 months
 - Faster processors are better (if you use them)
- SMT
- SMP
 - multiple processors or processor cores
 - ensure your kernel is SMP-enabled
- Powersaving
- 64-bit
 - address more memory
 - more registers
 - enhanced media processing
 - ensure you are using 64-bit kernel

Linux Performance Tuning - 12
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Speed

Processors are still following *Moore's Law* (http://en.wikipedia.org/wiki/Moore's_law) which predicts that transistor density on processors doubles roughly every 18 months. This means that processors roughly double in performance every 18 to 24 months. Faster processors are an easy way to improve your performance, if your system is processor-bound. You may find that your system is only using a fraction of your current processing power, in which case a processor upgrade will yield no improvements (because it's not the bottleneck!).

SMT

Symmetric Multi-Threading is a technology whereby multiple threads can run on a processor core at the same time (by using different parts of the cores execution pipeline). It is not to be confused with a system that has multiple cores – in an SMT system there is only one core. Intel implemented a version of SMT called *Hyper-threading* in some of their P4 processors. This has been abandoned in newer processors. The performance gains from SMT are unclear, on the P4 in particular, some types of jobs were seen to be slower when run with Hyper-threading enabled. We have also seen some stability problems with Linux and Hyper-threading.

SMP

Symmetric Multi-Processing refers to the technology where a system provides multiple processor cores in a single system either as physically separate processors or multiple cores on a single processor. Newer processor designs include multiple processor cores on the same chip – processors from Intel labelled as *Core Duo* and processors from AMD labelled as *X2* are dual-core processors. SMP capable systems allow multiple threads and processes to be run simultaneously providing clear performance improvements for loads that are multi-threaded or involve multiple processes. As with SMT, you must be running an SMP-enabled kernel to utilise multiple processors in a system. You may also want to consider tuning the kernel scheduler to better load multiple processors.

Power-saving

Linux supports the various power saving schemes available in modern processors. It is advisable to disable these features if performance is a priority to eliminate the delays introduced by the kernel having to spin up processors which have had their speed reduced while idle.

32-bit versus 64-bit

Both AMD (Athlon64 and Opteron) and Intel (EM64T) have introduced 64-bit processors in recent times. 64-bit processors bring a number of advantages including 64-bit memory addressing (allowing up to 1TB of memory to be addressed), larger registers and more registers (improving compilation and other tasks) and enhanced media instructions.

Hardware Performance - Memory

- More is generally better (but only if it's a bottleneck).
- 32-bit addressing limits
- BIOS settings
- Kernel settings
- Memory speed
- System bus speed
- ECC memory



As a general rule, more memory is better – if your application does not directly use the memory, the operating system can use the memory for caching. If your operating system is swapping a lot (using virtual memory) then you should definitely increase the amount of physical memory. But again, if memory isn't the bottleneck or the cause of a bottleneck, adding more memory won't help the performance of the system.

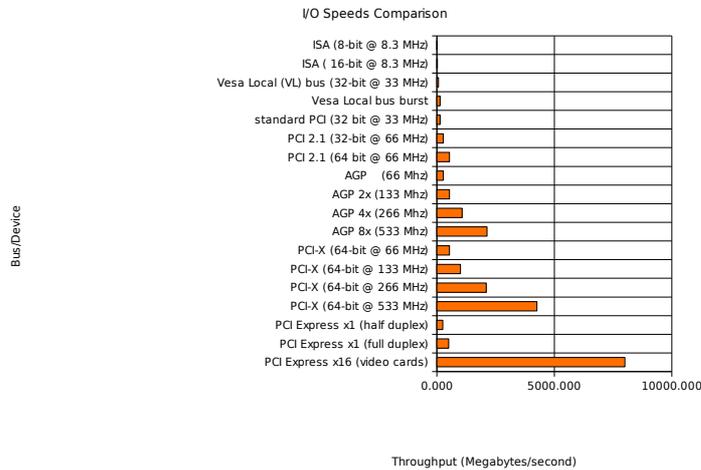
Modern 32-bit systems can handle up to 4GB of physical memory (although you may need to tweak the BIOS and operating system to access more than 3GB of memory). 64-bit systems do not have these limitations (although you may still need to tweak the BIOS to make the memory visible to the operating system).

Memory is available in different speeds but you are generally constrained to a specific type depending on the motherboard of the system. You should normally opt for the fastest memory that the motherboard can accommodate.

Server systems often use ECC or *parity* RAM. These are more expensive memory chips which include a parity bit which allows single-bit errors to be corrected without causing a system crash. There is a trade-off between the increased reliability and increased cost (in some cases it may make more sense to invest in larger amounts of non-parity RAM).

Memory can only be accessed as fast as the system bus can delivered data to and from it, so if memory access is critical, you should opt for the fastest possible system bus (again, you may have limited options depending on the motherboard or hardware you are choosing but note that server platforms are often more expensive because they provide hardware with better I/O).

Hardware Performance – I/O (1/4)



Linux Performance Tuning - 14

© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License

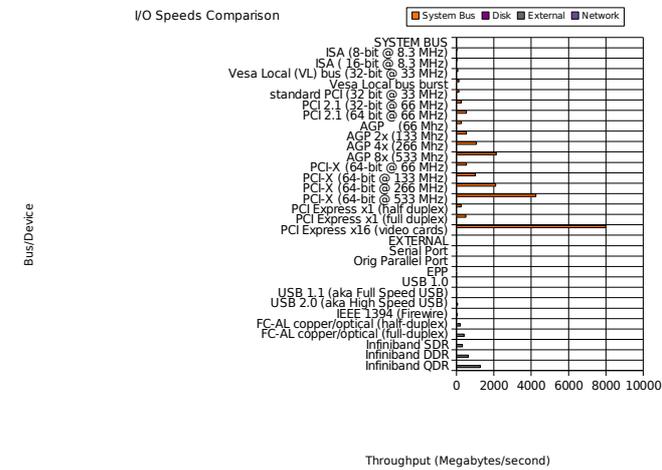


The I/O subsystem has 2 important characteristics, throughput and number of devices supported. Server class boards typically have better support for both. Each generation of system bus brings significant increases in bandwidth available.

It is important to recognise that the system bus bandwidth puts an absolute limit on how much data can pass through your system at any time – it is pointless putting a high throughput device onto a system bus that can only feed it a percentage of it's capacity.

Modern systems are using PCI-X or PCI Express (PCIe).

Hardware Performance – I/O (3/4)

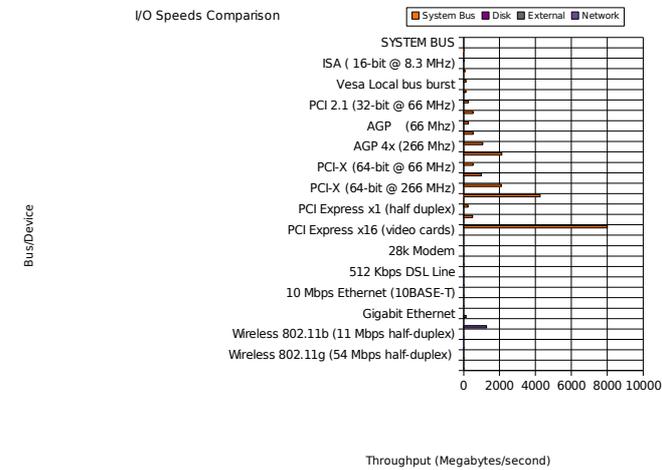


Linux Performance Tuning - 16
 © ApplePie Solutions 2004-2008, Some Rights Reserved
 Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



It is common to connect systems to external storage using either Firewire or FibreChannel for SANs. Note again that older system buses do not have the throughput to drive external buses to maximum capacity.

Hardware Performance – I/O (4/4)



Linux Performance Tuning - 17

© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



A consideration when moving to faster networking technologies is whether the system bus has the throughput to drive 1 (or possibly more) such devices. Certainly only very new systems will have the throughput to drive 10Gb Ethernet.

Hardware Performance - Storage

- Hard drive performance characteristics
 - Storage capacity
 - Transfer rate
 - Average latency
 - Spindle speed
- Drive types
 - PATA / IDE
 - SATA
 - SCSI
- RAID
 - RAID 0
 - RAID 1
 - RAID 5
 - JBOD
 - RAID 0+1 or 1+0



Hard drives

Drives can be compared using various performance characteristics including storage capacity (how much data the drive can store), transfer rate (the throughput of the drive under ideal conditions), average latency (the average time it takes to rotate the drive to a particular byte of data), spindle speed (how fast the drive spins).

Drive types

Traditionally, Parallel ATA (IDE) drives have been used for consumer PCs and business desktops. They were quite cheap to manufacture and offered reasonable performance and reliability. Businesses and performance users have traditionally used SCSI disks which offered higher performance and reliability at a higher cost (due to more complex drive electronics and more testing). Serial ATA (SATA) drives were introduced in 2003 as a successor to PATA drives. The performance of SATA drives approaches that of SCSI while offering a lower price. SATA drives are starting to be used in storage applications that would traditionally have been the domain of SCSI due to superior price/performance although high-end SCSI drives are still believed to be more reliable.

RAID

Redundant Array of Inexpensive Disks is a system which uses multiple disks together to provide either increased performance, data redundancy, capacity or a combination of all 3. RAID can be delivered using either hardware solutions, which require the use of a controller card to which the individual drives are connected, or software RAID which is performed at the operating system level. There are a number of different standard RAID combinations numbered 0-6, the following are the most common,

- RAID0 – striped set (spreads data evenly across 2 or more disks to increase the performance of reads and writes by fully utilising each disks throughput, no performance gains).
- RAID1 – mirrored set (creates an exact copy or mirror of a set of data to 2 or more disks, no performance gains but high reliability).
- RAID5 – striped set with parity (uses a parity block written to each disk to allow any 1 failed disk to be reconstructed with the parity information from the other disks, some redundancy gains but poor write performance).
- JBOD – just a bunch of disks (presenting a bunch of disks to the system as one logical drive, no reliability or performance gains).

RAID0 and RAID1 are often combined to give a high performance, high redundancy storage configuration known as either RAID 0+1 (striped mirrors) or RAID 1+0 (mirrored stripes).

Hardware Performance – Network (1/2)

- Ethernet
 - 10
 - 100 Fast
 - Gigabit
 - 10-Gigabit
- Processing Limits of Gigabit Ethernet
- High Performance Interconnects
 - Quadrics
 - Myrinet-2G
 - Myrinet-10G
 - Infiniband

Linux Performance Tuning - 19
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Ethernet

Ethernet is a set of standard computer networking technologies used in local area networks (LANs). The following are the most common implementations found in organisations today,

- Original Ethernet (10 Megabits/sec)
- Fast Ethernet (100 Megabits/sec)
- Gigabit Ethernet (1024 Megabits/sec)
- 10-gigabit ethernet (10240 Megabits/sec)

Most organisations are using a combination of Fast and Gigabit ethernet. Note that 10-gigabit ethernet is still being standardised and is mainly used at the moment in dedicated high performance computing environments.

Processing Limits

Note from the I/O slides that a system needs to be fast enough to drive gigabit ethernet to avail of the performance that gigabit offers – anecdotal evidence suggests that for every gigabit of network traffic a system processes, approximately 1GHz of CPU processing power is needed to perform work, so putting Gigabit interfaces in 1GHz systems is pointless.

High Performance Interconnects

If Gigabit Ethernet does not offer sufficient performance, there are a number of dedicated high performance interconnects available which offer performance in excess of 1Gbps at very low latencies. These interconnects are usually proprietary and require expensive switching hardware and dedicated cabling. For reference, Gigabit ethernet has a latency of 29-120ms and bandwidth of about 125MBps.

- Quadrics (1.29ms latency, 875-910 MBps bandwidth)
- Myrinet-2G (2.7-7 ms latency, 493 MBps bandwidth)
- Myrinet-10G (2.0 ms latency, 1200 MBps bandwidth)
- Infiniband (next generation standard supported by many companies including Compaq, IBM, HP, Dell, Intel and Sun – basic level provides 2.5Gbps in both directions and latencies of 2-4 ms).

Hardware Performance – Network (2/2)

- Jumbo Frames
 - larger MTU
 - less processing required
 - must be supported by all hardware
- Network Switch Backplanes
 - non-blocking
 - $2 \times n \times 1$ Gigabit
- Managed Switches
 - Monitoring
 - Bandwidth management
 - VLANs

Linux Performance Tuning - 20
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Jumbo Frames

The standard packet size or MTU used with ethernet is 1500 bytes. This has worked well up to now, but each packet received by a system has to be processed by the system, incurring a processor load. If you can increase the amount of data in a packet, the processor will have to do less work to process the same amount of data. Gigabit ethernet introduced the notion of *Jumbo Frames* which have a size of 9000 bytes. Only some Gigabit hardware supports Jumbo Frames, to use them from end to end requires that both Gigabit cards and any intermediate switches on the network support them.

Network Switch Backplanes

All network switches are not created equal! All the ports in a network switch are connected to some sort of backplane or internal bus. Ideally, the bandwidth of this backplane is $2 \times n \times 1$ Gigabit where n is the number of ports that the switch provides (we multiply by 2 because ideally, the switch should allow 1 Gigabit simultaneously in each direction). So a 10-port Gigabit Ethernet switch should have a bus backplane of 20 Gigabits/sec to be “non-blocking”.

Managed Switches

Basic network switches are described as unmanaged and don't provide any management or control access. At the mid to higher end of the network switch market, managed switches allow administrators to connect to a user interface on the switch and monitor/configure various aspects of the switches behaviour including,

- Monitor the bandwidth used by each port.
- Change the network settings (speed, data rate) of each port.
- Create Virtual LANs (VLANs) to separate traffic on different ports into distinct networks.
- Set bandwidth limits per port.
- Define Quality of Service priority for traffic.

OS Performance – Filesystem Tuning

- Match filesystem to workload
- Filesystem blocksize
 - 1024, 2048, 4096
 - number of inodes
- Separate different workloads
- Mount options
 - `noatime`
 - `nodiratime`
- Journalling parameters
 - logging mode
 - `barrier`
 - `ext3`
 - `enable barrier=1`
 - `disable barrier=0`
 - `ReiserFS`
 - `enable barrier=flush`
 - `disable barrier=none`

Linux Performance Tuning - 21
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Introduction

There are a number of ways of tuning the performance of a Linux file system,

- Choose a file system suited to your workload.
- For fixed block size filesystems, choose a size that suits the typical workload of the filesystem
- Use separate file systems for different types of workload.
- For journalling file system, tune the journalling parameters (logging modes, logging device)
- Tune file system mount options (`barrier`, `noatime`, `nodiratime`)

Filesystem blocksize

Filesystems such as `ext2` allow you to specify the block size to be used, one of 1024, 2048 or 4096 bytes per block. For a system that mostly creates files under 1k, it is more efficient to use 1024 byte blocks, while for a system that only ever stores large files, it is more efficient to use a larger block size. You can also tune the number of inodes that are created for a particular file system – a smaller number of inodes will result in a faster file system with the caveat that you cannot subsequently change the number of inodes for a file system and if you subsequently create a lot of small files on that file system, you run the risk of running out of inodes.

Separate file systems for different workloads

Consider using separate file systems for the following loads,

- Read-only or read-intensive file access.
- Write intensive file access.
- Streaming reads and writes.
- Random I/O

Journalling Parameters

`Ext3` and `ReiserFS` allow different logging modes to be used (see using mount option `data=<mode>` where *mode* is one of,

- `ordered` – default mode, all data is forced directly out to the main file system prior to its metadata being committed to the journal.
- `writeback` – data ordering is not preserved, data may be written into the main file system after its metadata has been committed to the journal. This gives high throughput at the risk of file system corruption in the event of a crash.
- `journal` – all data is committed into the journal prior to being written into the main file system. Generally slow, but may improve mail server workloads.

The `barrier` mount option controls whether or not the filesystem can re-order journal writes (re-ordering is good for performance but not so good for data integrity in the event of a power failure) – see `Documentation/block/barrier.txt` for more details.

OS Performance – Filesystems

- ext2
 - standard linux fs
- ext3
 - journaling ext2
 - stable
- JFS
 - good support for large files, large directories
 - relatively new
- ReiserFS
 - particularly good for large numbers of small files
 - generally fast
- Reiser4
 - successor to ReiserFS
 - experimental
- XFS
 - low latency
 - suited to very large machines
 - slow on metadata creation and deletes

Linux Performance Tuning - 22
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



ext2

Standard non-journaling Linux file system. Big disadvantage is the time it takes to perform a file system check (fsck) in the event of a file system failure or system crash. This can significantly affect the system's overall availability (system will be unavailable until the check is completed).

ext3

Journaling version of ext2 file system. Suitable for direct upgrades from ext2 (can be performed on the fly). Suitable for machines with more than 4 CPUs (ReiserFS does not scale well). Synchronous I/O applications where ReiserFS is not an option. If using large numbers of files in directories, enable hashed b-tree indexing for faster lookups

```
# mke2fs -O dir_index
```

JFS

Good support for large files and large numbers of files. Parallelises well, good features, low CPU usage. JFS seems particularly fast for read-intensive workloads. Relatively new, may not be as stable.

ReiserFS

Applications that use many small files (mail servers, NFS servers, database servers) and other applications that use synchronous I/O.

Reiser4

Experimental successor to ReiserFS, not yet integrated into the mainline kernel.

XFS

Low latency file system best suited for very large machines (>8 CPUs), very large file systems (>1 TB). Streaming applications may benefit from the low latency. Handles large files and large numbers of files well. High CPU usage. XFS seems particularly fast for write-intensive workloads. Slow on metadata creation and deletes.

OS Performance – Other Filesystems

- Cluster filesystems
 - concurrent access shared storage
 - GFS
 - GFS2
 - OCFS
 - OCFS2
- Distributed filesystems
 - fault tolerant shared storage
 - client server model
 - NFS
 - Samba (CIFS)
 - Lustre

Linux Performance Tuning - 23
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Introduction

As well as standard operating system file systems, there are a number of alternative file system approaches which may provide better performance or improved availability. These are used either over a network or over a dedicated storage interconnect of some sort.

Cluster file systems

Cluster file systems are typically used by clusters of computers performing related processing. Each node in the cluster has direct concurrent access to the storage and some distributed locking mechanism is usually supported. Cluster file systems are typically shared using some sort of dedicated interconnect (a shared SCSI bus or fibre) and allow multiple nodes to work on related data at the same time. Cluster file systems do not normally support disconnected modes where nodes periodically synchronise with a storage server, nor do they tolerate interconnect failures well. Typical cluster file systems found on Linux include GFS, GFS2, OCFS and OCFS2.

GFS

The Global File System (GFS) is a commercial product which was bought by Red Hat and released under an open source license.

GFS2

GFS2 is an improved version of GFS which is intended to be merged into the mainline kernel. It is not expected to be backwards compatible with GFS.

OCFS

Oracle Cluster File System is a non-POSIX compliant cluster file system developed by Oracle and released as open source. It is mainly intended to be use with clustered Oracle database servers and may give unexpected results if used as a general file system

OCFS2

A POSIX compliant successor to OCFS which has been integrated into the mainline kernel as of 2.6.16.

Distributed file systems

Distributed file systems are file systems that allow the sharing of files and resources over a network in the form of persistent storage. Normally implemented as a client-server model, distributed file systems are typically more fault tolerant than standard cluster file systems while bringing most of the advantages of a CFS. Typical distributed file systems include NFS and Samba (CIFS). Lustre is a highly scalable, high performance distributed file system intended to be used on clusters of 100s and 1000s of systems.

Performance Tuning Exercise 2

Suggest suitable filesystems for the following scenarios indicating what characteristics of that filesystem make it suitable,

- 1) A Linux desktop used for word processing, web browsing and email.
- 2) A Samba fileserver serving a small workgroup.
- 3) A streaming media server.
- 4) An NNTP news server.
- 5) A mailserver for a medium sized company.
- 6) A 100-node HPC cluster.
- 7) An enterprise fileserver.

Linux Performance Tuning - 24
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



lmbench

The **lmbench** tool expects the *bk* command to be installed but it is not necessary to run the benchmarks. Make the following changes after downloading (applies to lmbench2, other versions may require other changes).

- Add a dummy *bk* command to your path e.g.

```
#!/bin/sh  
echo "dummy bk command called"
```

Edit the Makefile in `src` and replace the `bk.ver` target with the following

```
bk.ver:  
    echo "ignoring bk.ver target"
```

in order to compile and run lmbench without errors – read the README for full instructions on running lmbench. Results can be viewed by doing the following

```
cd results  
make
```

Iperf

Download the compile the Iperf command. One machine will need to run as server and another as client. You may need to co-ordinate with other users in the room. Investigate the effects of multiple people running the benchmark simultaneously.

OS Performance – General

- Verify basic configuration
- Disable unnecessary services
 - `/etc/rcN.d`
- Remove unnecessary jobs
 - `cron`
- Restrict access to the system
 - development
 - test
 - production

Linux Performance Tuning - 25
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Verify basic configuration

Verify that the basic operating system is configured and operating correctly. Things to verify are,

- Do all file systems have enough free space?
- Is there enough swap allocated (`free`)?
- Are you running the latest stable kernel?
- Are you running the correct kernel for your architecture?
- Are there any os problems which may be affecting performance (`dmesg`)?
- Is the underlying storage operating without errors (performance can be expected to degrade considerably while a RAID array is rebuilding failed storage)?

Remove unnecessary services

Generally, you should eliminate any unnecessary work that the system is doing in order to maximise the resources available for the application. Identify any unnecessary services running and remove them from this runlevel (see `/etc/rcN.d` for a list of services started at this run-level) or change to a runlevel that is more suited to your needs (in particular, if you don't require X, consider changing to a non-graphical run-level).

Eliminate unnecessary jobs

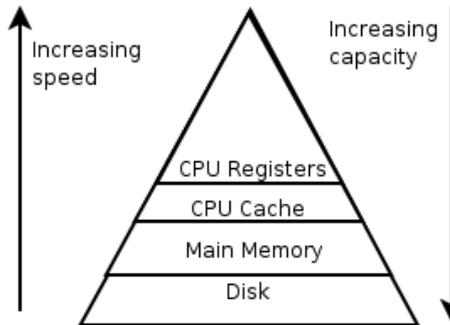
It is common to schedule system jobs to run periodically performing various system maintenance tasks – it may be preferable to remove these jobs and perform them manually during planned maintenance outages rather than have these jobs interfere with the performance of your application. Review the contents of the `/etc/cron.*` directories and each users `crontab`.

Restrict access to the system

Only users who need to login to the system to operate the application or performance maintenance should be using the system. Even common operating system tasks such as running the `find` command can significantly impact performance (it is normal to perform development and testing on separate systems).

OS Performance – Virtual Memory (1/2)

- Sizing
 - memory required for application
- Monitoring
 - swap usage
 - % cpu spent paging
- Tuning
 - rate at which swap is used
 - cache sizes



Linux Performance Tuning - 26
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Overview

Modern systems consist of a hierarchy of storage, with small amounts of very fast storage at the top of this hierarchy and large amounts of relatively slow storage at the bottom. Modern memory management systems manage this memory hierarchy by ensuring the fastest storage is allocated to active processes and moving inactive process memory down to the slower memory.

To optimise performance, your processes should spend most of their time at the top of this hierarchy – systems with too little main memory can expend significant processor time moving processes to and from disk storage (a process known as *paging*). The CPU registers and CPU caches are processor dependent (note that server class processors such as Intel Xeons and AMD Opterons tend to have larger amounts of cache than their desktop equivalents).

Monitoring

Monitor your system performance to ensure that,

- You are only using swap in exceptional circumstances
- Your system is not spending a lot of processor time paging processes
- Your system has adequate main memory for the expected workload

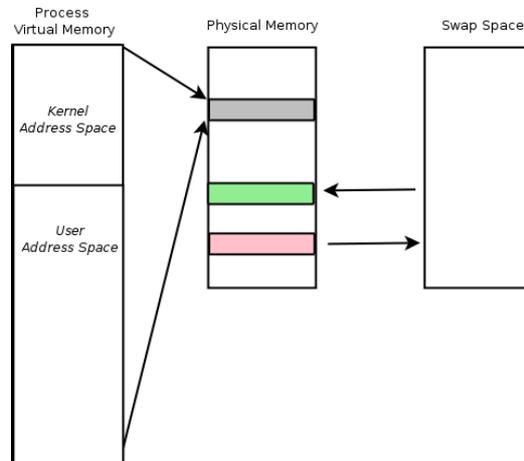
Tuning

Various aspects of Linux virtual memory can be tuned including,

- The rate at which processes are moved out of main memory
- The percentage of main memory used for caching

OS Performance – Virtual Memory (2/2)

- Virtual Address space
 - 4 GB on 32-bit systems
 - 16 EB on 64-bit systems
 - Kernel space
 - User space
- Paging
- Performance
 - minimise paging



Linux Performance Tuning - 27
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Virtual Memory Overview

Each process running on a Linux system sees the entire memory space that is addressable by the processor. This allows programs written for a Linux system to ignore details such as how much physical memory is in a particular system and how much of that memory is used by other processes.

Address space

The total address space depends on the processor architecture – 32-bit systems can address up to 2^{32} bytes (4 GB) while 64-bit systems can address up to 2^{64} bytes (17,179,869,184 GB). In practice, operating systems kernels do not use address spaces this large. Address space is usually split into an area managed by the kernel (*kernel address space*) and an area used by the application (*user address space*).

Paging/Swapping

When a process is created, it is allocated an address space. Only some of this space initially resides in physical memory – as the memory requirements of the process grow, more space is allocated to the process by the memory management system. As the physical memory is used up, the kernel will move pages of memory that were not recently used out onto the area of disk known as *swap space* – this process is known as *paging* or *swapping*.

Performance

Ideally, you would not need to use any swap space since it is expensive to move pages to and from disk storage devices. In practice, we try to minimise the amount of paging that occurs and ensure that our application normally runs entirely within main memory.

OS Performance – Drive tuning

- Tools
 - hdparm (IDE/PATA)
 - sdparm (SCSI)
- 2.6 I/O schedulers
 - cfq
 - deadline
 - as
 - noop
- 2.4 I/O scheduler
 - elvtune

Linux Performance Tuning - 28
© Applepie Solutions 2004-2008. Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



There are a number of things you can look at to tune the underlying drive hardware. Most modern Linux kernels automatically activate the most important drive settings, but there are some tools which can be used to manually tweak drive settings.

Tools

hdparm

The `hdparm` tool can be used to change DMA settings and other options such as write-caching mode. `hdparm` is intended to be used with IDE or PATA drives. SATA drives are mostly automatically configured by the SATA drives (`libata`). Some IDE drives support switching the drive between quiet mode and high performance mode (so called *acoustic management*), `hdparm` also allows you to switch between these modes on supported drives.

sdparm

This command is the equivalent of `hdparm` for SCSI drives. It allows you to change various SCSI settings including write-back and read cache settings.

2.6 I/O Schedulers

The Linux kernel uses a mechanism called an *I/O scheduler* to manage reads and writes to and from I/O devices. The 2.6 Linux kernel series allows different I/O schedulers to be plugged in at start-up time to change the I/O subsystem behaviour (boot parameter is `elevator=X` where *X* is one of `noop`, `deadline`, `as` or `cfq`). Different schedulers have different performance characteristics which make them more suitable for different types of workload, **cfq** – the Completely Fair Queueing scheduler is the default in RHEL4 and SLES9. It shares the I/O bandwidth equally among all I/O requests and is a good compromise between throughput and latency.

deadline – the deadline scheduler minimises the maximum latency per request providing good disk throughput. It is best for disk-intensive database applications.

as – the anticipatory scheduler is the default in mainline Linux kernels. It attempts to reduce disk seek operations by queueing I/O requests before dispatching them to the disk which may increase I/O latency. Best suited to file servers and desktops with single IDE or SATA disks.

noop – simple FIFO queue which is best used with storage with its own caching and storage mechanisms.

2.4 I/O Scheduler

The 2.4 Linux kernel does not support plugging in different I/O schedulers but it does allow some I/O scheduler parameters to be tuned using the `elvtune` command. You can increase the max latency on items in the I/O queue – this reduces the performance of interactive system but can increase throughput on server type systems.

OS Performance – Network Tuning – Core Settings

- Kernel Auto Tuning
- Socket Buffers
 - `net.core.wmem_default`
 - `net.core.rmem_default`
 - `net.core.rmem_max`
 - `net.core.wmem_max`
 - `net.core.netdev_max_backlog`
 - `net.core.somaxconn`
 - `optmem_max`



Kernel Auto Tuning

Current 2.4 and 2.6 Linux kernels auto-tune the some important network related kernel parameters at boot time (on the basis of total available memory and so on) and during operation. In most cases, this auto-tuning will provide a near-optimal configuration. Where it does not, it is possible to alter some of the tunables directly. The following are some of the most useful network related tunables.

Socket Buffers

```
net.core.wmem_default  
net.core.rmem_default
```

The global default size of the read and write socket buffers. These are auto-tuned to different values at boot-time depending on total system memory. Systems experiencing heavy network loads may benefit from increasing these.

```
net.core.rmem_max  
net.core.wmem_max
```

These values are the maximum size that the read and write socket buffers can be set to. These determine the maximum acceptable values for `SO_SNDBUF` and `SO_RCVBUF` (arguments to `setsockopt()` system call). The kernel sets the actual memory limit to twice the requested value (effectively doubling `rmem_max` and `wmem_max`) to provide for sufficient memory overhead.

```
net.core.netdev_max_backlog
```

Specifies the maximum number of incoming packets that will be queued for delivery to the device queue. Increasing this value allows a greater number of packets to be queued and reduces the number of packets dropped during high network load.

```
net.core.somaxconn
```

This specifies the maximum number of pending connection requests. When the number of queued incoming connection requests reaches this value, further connection requests are dropped.

```
optmem_max
```

The maximum initialisation size of socket buffers, expressed in bytes.

OS Performance – Network Tuning – TCP/IP Settings 1

- TCP buffer settings
 - `net.ipv4.tcp_rmem[]`
 - `net.ipv4.tcp_wmem[]`
 - `net.ipv4.tcp_mem[]`
- TCP Options
 - `net.ipv4.tcp_window_scaling`
 - `net.ipv4.tcp_sack`
- TCP connection Management
 - `net.ipv4.tcp_synack_retries`
 - `net.ipv4.tcp_retries2`

Linux Performance Tuning - 30
© Applepie Solutions 2004-2008. Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



TCP buffer settings

`net.ipv4.tcp_rmem[]`

An array of 3 integers specifying the minimum size of the read buffer ([0]), the default size of the read buffer ([1]) and the maximum size of the read buffer ([2]). When a TCP protocol socket is created (AF_INET, SOCK_STREAM) the size is set to the default. The minimum and maximum limits are used by the kernel when dynamically tuning the size.

`net.ipv4.tcp_wmem[]`

As with `tcp_rmem`, an array of 3 integers (minimum, default and maximum) specifying the size of the write buffer. Specifying a larger write buffer allows an application to transfer a large amount of data to the buffer without blocking.

`net.ipv4.tcp_mem[]`

This parameter specifies how the system balances the total network memory usage against other memory usage, such as disk buffers. It is initialized at boot time to appropriate fractions of total system memory. It is not recommended to alter this setting to its potential impact on other kernel subsystems.

TCP Options

`net.ipv4.tcp_window_scaling`

Enables TCP Window Scaling as described in RFC1379. Window scaling allows receive buffers larger than 64KB to be advertised, thus allowing a sender to fill network pipes whose bandwidth latency is larger than 64KB. This can be useful on high latency network links such as satellite or long range wireless.

`net.ipv4.tcp_sack`

Enables TCP Selective Acknowledgement (SACK) feature which allows the receiving side to give the sender more detail of lost packets, reducing the amount of data retransmitted. This is useful on high latency network links but throughput may be improved on high-speed local networks by disabling this.

TCP connection Management

`net.ipv4.tcp_synack_retries`

Controls the number of times the kernel tries to resend a response to an incoming SYN/ACK segment. Reduce this to speed of detection of failed connection attempt from remote host.

`net.ipv4.tcp_retries2`

Controls the number of times the kernel resends data to a remote host with which a connection has been established. Reducing this allows busy servers to free up resources quicker.

OS Performance – Network Tuning – TCP/IP Settings 2

- TCP Keep-Alive Management
 - `net.ipv4.tcp_keepalive_time`
 - `net.ipv4.tcp_keepalive_intvl`
 - `net.ipv4.tcp_keepalive_probes`
- IP Ports
 - `net.ipv4.ip_local_port_range`

Linux Performance Tuning - 31
© ApplePie Solutions 2004-2008. Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



TCP Keep-Alive Management

By default, a TCP connection can remain idle permanently. The Keep-Alive mechanism allows a host to monitor a TCP connection and learn of a failure in a reasonable time. Applications need to enable the Keep-Alive option when using the `setsockopt()` in order to avail of the following kernel settings.

`net.ipv4.tcp_keepalive_time`

If a connection is idle for the number of seconds specified by this parameter, the kernel initiates a probing of the connection to the remote host.

`net.ipv4.tcp_keepalive_intvl`

Specifies the time interval in seconds between the keepalive probes sent by the kernel to the remote host.

`net.ipv4.tcp_keepalive_probes`

Specifies the maximum number of probes the kernel sends to the remote host to detect if it is still alive.

The default kernel keepalive settings result in connections getting dropped after over 2 hours of idle time. This is a good thing for servers talking to clients over the internet which may have intermittent connections or long round-trip times but it can result in excessive usage of resources on busy servers. Reducing these settings can result in resources getting freed up faster.

IP Ports

`net.ipv4.ip_local_port_range`

Specifies the range of ports available for use by the IP protocol. The default is 28,232 ports on systems with more than 128KB of memory. Increasing the range can make more ports available.

OS Performance – CPU related tuning

- Linux 2.4 CPU Scheduler Characteristics
 - $O(n)$
 - Ok for servers
- Linux 2.6 CPU Scheduler Characteristics
 - $O(1)$
 - better SMP support
 - Good for both servers and desktops
- CPU Affinity
 - `taskset` command
- SMP IRQ Affinity
 - `/proc/irq/N/smp_affinity`

Linux Performance Tuning - 32
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Linux 2.4 CPU Scheduler Characteristics

- $O(n)$ characteristics – runtime increases linearly with inputs
- Robust scheduler that works across a wide variety of platforms
- Relatively simple design
- Relatively large average time slice (210ms)
- Non Pre-emptible kernel (doesn't allow any RT support)

Linux 2.6 CPU Scheduler Characteristics

- $O(1)$ characteristics – runtime is constant
- A runqueue for each processor
- dynamically tuned time slices
- prioritises I/O-bound processes over CPU-bound processes (lets I/O-bound processes finish faster and improves perceived performance)
- Soft real-time support (no guaranteed response times)
- Relatively small average time slice (100ms)
- Pre-emptible kernel (allows some RT support)
- Processor affinity

SuSE runs a patched kernel which allows min and max time slices to be adjusted – for other distributions need to edit `linux/sched.c` (see `MIN_TIMESLICE` and `MAX_TIMESLICE` macros in the code) and recompile kernel. Bigger time slice lengths will give high efficiency but reduces responsiveness (so good for servers but bad for desktops).

CPU Affinity

Specify that certain processes should run on certain CPUs (scheduler will automatically do some of this). Supported on 2.6 kernels only.

```
taskset -p <list of cpus> <pid of task>
```

SMP IRQ Affinity

System devices signal to the CPU that they have data to be processed by sending an interrupt to CPU. On systems with multiple processors (SMP), Linux allows you to bind certain devices to certain Interrupt Request (IRQ) numbers. Useful if you have, for example, 2 busy network devices and 2 or more CPUs in the system. You can bind the first network card to the first processor and the second network card to the second processor, balancing the load from the network cards. This is supported on both 2.4 and 2.6 kernels.

```
echo <bitmask of allowed CPUs> > /proc/irq/IRQ#/smp_affinity
```

OS Performance – 2.4 Kernel tunables

- `/proc/sys/vm/bdflush`
 - kernel caches writes to disk in memory buffers
 - `bdflush` daemon periodically flushes these to disk
 - tune to suit characteristics of underlying disk
- `/proc/sys/vm/kswapd`
 - kernel pages/swaps data from memory to disk
 - `kswapd` manages this
 - tune to suit workload
- `/proc/sys/vm/max-readahead`
- `/proc/sys/vm/min-readahead`
 - control VFS prefetching
 - larger numbers increase memory consumption
 - may increase performance for linear file reads

Linux Performance Tuning - 33
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Settings are from 2.4.28 mainline kernel, settings may change between versions and distributions.

`/proc/sys/vm/bdflush`

Takes 9 arguments as follows,

- Maximum numbers of dirty buffers in the buffer cache. Higher values cause kernel to delay disk writes for a longer time but results in a lot of I/O all at once when memory runs short. Lower values result in shorter, more frequent I/O operations. Default is 30%.
- The maximum number of dirty buffers that `bdflush` can write to the disk in one time. High values give delayed, bursty I/O while small values may lead to memory shortages when `bdflush` isn't woken often enough. Default is 500 dirty buffers.
- 3rd parameter is unused.
- 4th parameter is unused.
- The minimum rate at which `kupdate` will wake and flush. Value is specified in *jiffies*, on PCs, the number of *jiffies* per second is normally 100. Default is 5 seconds. Longer times reduce I/Os but may result in memory shortages or long I/Os.
- Maximum time Linux waits before writing out a dirty buffer to disk. Value is in *jiffies* again. Default value is 30 seconds.
- Specifies the percentage of buffer cache that is dirty before `bdflush` starts operating synchronously (before `bdflush` forces buffers to disk). Default is 60.
- The percentage of buffer cache that is dirty which will stop `bdflush`. Default is 20%.
- 9th parameter is unused.

```
echo 30 500 0 0 500 3000 60 20 0 > /proc/sys/vm/bdflush
```

`/proc/sys/vm/kswapd`

- Maximum number of pages `kswapd` tries to free in one round (usually divided by 4 or 8). Increase this to increase bandwidth to/from swap. Default is 512.
- Minimum number of times `kswapd` tries to free a page each time it is called. Default is 32.
- Number of pages written by `kswapd` in one turn. Keep large to reduce I/Os. Default is 8.

`/proc/sys/vm/max-readahead` (default 31)

`/proc/sys/vm/min-readahead` (default 3)

Specifies the maximum and minimum number of blocks the VFS will read ahead when accessing files. Increasing the max increases the number of blocks prefetched at the cost of more memory. May or may not increase performance. Increasing the min increases the readahead unconditionally. May improve performance if accessing files in a linear fashion.

OS Performance – 2.6 Kernel tunables (1/2)

- `/proc/sys/fs/file-max`
- `/proc/sys/fs/inode-max`
- `/proc/sys/vm/overcommit_memory`
- `/proc/sys/vm/overcommit_ratio`
- `/proc/sys/vm/dirty_ratio`
- `/proc/sys/vm/dirty_background_ratio`
- `/proc/sys/vm/dirty_expire_centisecs`
- `/proc/sys/vm/dirty_writeback_centisecs`



Settings are from 2.6.16 mainline kernel, settings may change between versions and distributions.

`/proc/sys/fs/file-max`

Specifies the maximum number of file-handles that the Linux kernel will allocate. If you get error messages about running out of file handles, you might want to increase this limit.

`/proc/sys/fs/inode-max`

Specifies the maximum number of inode handles. This value should be 3-4 times larger than the value in `file-max`, since `stdin`, `stdout` and network sockets also need an inode struct to handle them. Increase this value if you run out of inode handles.

`/proc/sys/vm/overcommit_memory`

Enables the memory overcommitment kernel feature. Use one of the following values,

- 0 – kernel attempts to estimate amount of free memory left when userspace requests memory.
- 1 – kernel pretends there is always enough memory until it runs out.
- 2 – kernel uses a *never overcommit* policy that attempts to prevent any overcommit of memory.

Lots of programs `malloc()` large amounts of memory that they never use so overcommitting may make sense for your workload (but may cause problems if your applications are well-behaved but memory hungry).

`/proc/sys/vm/overcommit_ratio`

When `overcommit_memory` is set to 2, the committed memory is not allowed to exceed swap plus this percentage of physical memory.

`/proc/sys/vm/dirty_ratio`

Contains, as a percentage of total system memory, the number of pages at which a process which is generating disk writes will itself start writing out dirty data.

`/proc/sys/vm/dirty_background_ratio`

Contains, as a percentage of total system memory, the number of pages at which the `bdflush` daemon will start writing out dirty data.

`/proc/sys/vm/dirty_expire_centisecs`

Specifies (in 100ths of a second) how long dirty pages remain dirty before being flushed.

`/proc/sys/vm/dirty_writeback_centisecs`

Specifies (in 100ths of a second) interval between runs of `bdflush`.

OS Performance – 2.6 Kernel tunables (2/2)

- `/proc/sys/vm/swapiness`



Settings are from 2.6.16 mainline kernel, settings may change between versions and distributions.

`/proc/sys/vm/swapiness`

When the system runs out of memory, it has 2 options to free up some memory – reduce the disk cache in memory by removing the oldest cached data or swap some the pages of inactive programs out to disk. `swapiness` gives you some control over this behaviour. The default is 60. Higher values favour swapping over reducing disk cache. Lower values favour reducing disk cache over swapping. To tune this optimally, determine if your application benefits more from large disk caches or swapping out of inactive processes.

Performance Tuning Exercise 3

Suggest some tuning that could be performed for the following scenarios,

- 1) A database server.
- 2) A Java application server.
- 3) A desktop system.
- 4) A batch processing system.

Linux Performance Tuning - 36
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Imbench

The **imbench** tool expects the *bk* command to be installed but it is not necessary to run the benchmarks. Make the following changes after downloading (applies to *imbench2*, other versions may require other changes).

- Add a dummy *bk* command to your path e.g.

```
#!/bin/sh  
echo "dummy bk command called"
```

Edit the Makefile in *src* and replace the *bk.ver* target with the following

```
bk.ver:  
    echo "ignoring bk.ver target"
```

in order to compile and run *imbench* without errors – read the README for full instructions on running *imbench*. Results can be viewed by doing the following

```
cd results  
make
```

Iperf

Download and compile the *Iperf* command. One machine will need to run as server and another as client. You may need to co-ordinate with other users in the room. Investigate the effects of multiple people running the benchmark simultaneously.

Performance Monitoring – CPU Utilisation

- uptime
- /proc/cpuinfo
- vmstat
- top

Linux Performance Tuning - 37
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



uptime

The `uptime` command gives a summary of system information including how long the system has been running, how many users are logged in and the system load averages for the last 1, 5 and 15 minutes. System load average is a good high-level measure of how busy the system is – it indicates the number of processes that are waiting to run at any time. On a CPU system, anything over 1 indicates that the system has more processes trying to run than it can finish (while on a 4 CPU system it indicates that the system is idle 75% of the time).

```
# uptime
09:37:05 up 12:54,  2 users,  load average: 0.00, 0.05, 0.07
```

/proc/cpuinfo

Contains details of how many processors the kernel can see and some of their properties – useful for understanding the load averages.

vmstat

The `vmstat` command is a real-time performance monitoring tool which gives performance statistics for memory, swap, I/O, the system and the CPU. We will focus on the CPU related information in this slide and look at the others later on.

```
# vmstat 1 10
procs -----memory----- --swap--  -----io----- -system--  ----cpu----
 r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa
 4  0    0 305528 251088 438840  0  0   13   6 255  81  0  0 99  0
 3  0    0 303564 251104 439464  0  0  628  32 328 1594 92  1  0  7
 0  2    0 296752 251140 445340  0  0 5848  0 603 2041 80  3  0 17
 0  2    0 284976 251232 456664  0  0 11384  0 565 2124 36  3  0 61
```

The `procs` section shows the number of running (`r`) and blocked (`b`) processes. Blocked processes are processes which are waiting for disk or network I/O. A large number of blocked processes suggests that there may be an I/O bottleneck on your system. The `system` section shows the number of interrupts (`in`) and context switches (`cs`) per second. If these are really high, your system may be too busy servicing requests and context switching to actually get any useful work done. The `cpu` section shows the percentage of total CPU time spent in *user space* (`us`), *kernel space* (`sy`), *idle* (`id`) and waiting for I/O (`wa`). Processes normally run in user space unless they make a system call in which case the process will run in kernel space. Things to watch for here are whether the system is very busy (`us+sy`) or whether lots of processes are waiting on I/O.

top

Shows the most active processes on the system.

Performance Monitoring – Memory Utilisation

- free
- /proc/meminfo
- /proc/slabinfo
- ps
- vmstat

Linux Performance Tuning - 38
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



free

The `free` command displays the amount of free and used memory in the system and can be useful to give an overview of how physical memory, swap and buffers are used.

```
# free
Mem:              total      used      free      shared    buffers    cached
584476
-/+ buffers/cache: 479788      1076300
Swap:            979924          0      979924
```

The *Mem* line describes physical memory usage, the *-/+ buffers/cache* line describes the amount of memory currently used by your applications (ignoring buffers and cache usage) and the *Swap* line describes the utilisation of system swap space.

/proc/meminfo

The `free` command is actually displaying information from the `proc` file system. You can directly view these files for more detailed information if required.

```
# cat /proc/meminfo
MemTotal:        1556088 kB           total physical memory in the system
MemFree:         177584 kB           total amount of unused memory
Buffers:         274768 kB           buffer cache for I/O operations
Cached:          614280 kB           memory for reading files from disk
Slab:            216588 kB           memory used by kernel data structures
```

/proc/slabinfo

Gives a breakdown of the memory used by kernel data structures.

ps

The `ps` command can be used to give a breakdown of system memory usage by processor.

```
# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1960    660 ?        S      Aug10   0:00 init [2]
root     4708  0.0  0.0   2588    928 ?        Ss     Aug10   0:00
/sbin/syslogd
smulcahy 7451  1.3  4.8 152156 75408 ?        Sl     09:18   2:03 /bin/firefox
```

Including the total percentage of system memory usage (*%MEM*), the virtual memory footprint (*VSZ*) and the amount of physical memory currently used (*RSS*).

vmstat

See previous slide for introduction – the *memory* section reports the amount of memory swapped (*swpd*), free memory (*free*), buffer cache for I/O (*buff*) and cached read (*cache*).

Performance Monitoring – I/O Utilisation

- `vmstat`
- `iostat`



vmstat

The `vmstat` command also reports back on I/O statistics. The *io* section includes the number of blocks read in (*si*) from I/O devices and the number of blocks written out (*bo*) in kilobytes per second. It provides a quick way of getting an overview of I/O throughput. Processes blocked (*wa*) and interrupts (*in*) may also give indicators of I/O performance.

iostat

The `iostat` command provides summary and realtime I/O performance statistics and can be useful for monitoring overall I/O and identifying opportunities for load balancing across I/O devices.

```
# iostat 1 10
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           8.91    0.00   1.98    0.00    0.00   89.11

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
hda                  11.88         0.00         126.73         0         128
hdb                   0.00         0.00          0.00         0          0
```

As with `vmstat`, `iostat` can provide either a summary of the I/O activity on the system since boot by calling the command without arguments or over a timed interval by passing timing arguments. The *avg-cpu* section gives some general CPU utilisation numbers including *%iowait* which gives the percentage of idle time spent by the CPU waiting for an I/O request to complete. A high *%iowait* may indicate an I/O bottleneck and should be investigated further.

The *device* section includes transfers per second (*tps*), blocks read per second (*Blk_read/s*), blocks written per second (*Blk_wrtn/s*), total number of blocks read (*Blk_read*) and total numbers of blocks written (*Blk_wrtn*). What you should be seeing in the *Device* section is I/O balanced across all available devices and possibly dedicated devices for reading and writing if the traffic can cleanly separate.

When used with the `-x` argument, `iostat` reports extended I/O statistics. These may not be available in some 2.4 kernels.

Performance Monitoring – sar

- Collect
 - I/O and transfer rate statistics
 - Paging statistics
 - Process creation counts
 - Block device activity
 - Network statistics
 - Processor activity
 - Queue lengths
 - Memory and space space utilisation
- Reports
- Trending

Linux Performance Tuning - 40
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Introduction

The `sar` command collects statistics on various operating system parameters by interrogating counters in the kernel. It provides some of the functionality of the commands covered already and is particularly useful for gathering historical information and reporting trends in operating system behaviour.

Any or all of the following information can be collected,

- I/O and transfer rate statistics
- Paging statistics
- Process creation counts
- Block device activity
- Network statistics
- Processor activity
- Queue lengths
- Memory and space space utilisation

Collection

To report performance trends on the system, a background `sar` process must run periodically and collect performance statistics. These are stored in a binary file which can subsequently be read by `sar`. `sar` has been developed to have a low system impact so it should not introduce any additional performance problems on your system.

Reporting

The `sar` tool interprets the saved data and prints summaries of the statistics gathered. 3rd party tools can also be used to graph the data or otherwise interpret it.

Trends

`sar` is particularly useful if you run it constantly and gather performance figures that are typical for your system when it is normally operating. If and when performance problems start occurring, you should be able to identify the bottlenecks, but also when the problem started occurring which may allow you to identify changes to the environment which introduced the problem.

Performance Monitoring – network

- Managed switch
 - diagnostics
 - port mirroring
- Commands
 - ifconfig
 - netstat
 - tcpdump
 - wireshark
 - iptraf

Linux Performance Tuning - 41
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Introduction

Network monitoring and tuning is a field in itself, here we will review some operating system level tools which can provide you with enough information to decide whether the network is your performance bottleneck and possibly some data to supply to your network people to help them diagnose the problem further. Note that modern managed switches include diagnostics to allow you to monitor traffic rates on a per port basis and so on.

ifconfig

The ifconfig command is normally used to configure network devices, but it also reports out some summary statistics for each network device including the number of packets sent and received, and counts of any errors in transmission. It can give a very high level view of what kind of traffic is running through your devices.

netstat

The netstat can be used to view a wide range of information relating to networking. Invoked without any arguments, it displays a summary of the currently opened connections including the protocol, the local address and port, the remote address and port and the state of the connection.

With the -i option, netstat provides similar, but more detailed information on network devices than that provided by ifconfig.

With the -s option, netstat provides TCP/IP statistics including details of failed connections, retransmits and so on.

tcpdump

The tcpdump command can be used to view network traffic travelling to and from the machine. It is a command-line tool and the output can be difficult to view. In addition, on a busy system it is possible to overload tcpdump causing it to lose some information. It can be useful to identify what kind of traffic is causing a problem if a network bottleneck is identified.

wireshark

Wireshark (previously known as Ethereal) is a graphical tool for viewing networking traffic. It can interpret tcpdump output files or it can be used to perform its own realtime monitoring. It is particularly useful if used with a managed switch that allows port mirroring – you can replay all traffic to a particular server to a different port to which a diagnostic system is attached.

iptraf

iptraf is an ncurses-based IP LAN monitor that generates various network statistics including TCP info, UDP counts, ICMP and OSPF information, Ethernet load info, node stats, IP checksum errors, and others.

Performance Tuning – Web Servers

- Use Apache 2.x rather than Apache 1.x.
- Ensure plenty of RAM.
- Ensure webserver is never swapping.
- Tune application (MaxClients setting).
- Consider load balancing multiple servers.
- Split dynamic and static load to different servers.



Performance Tuning – File & Print Server

- Use fast storage hardware
- Use software RAID
- Use a fast filesystem (ReiserFS or XFS)
- Mount filesystems with *noatime* option
- Consider using the *cfq* or *deadline* I/O scheduler
- Keep reads separate from random writes
- Lots of RAM



Performance Tuning – Database Servers

- Lots of RAM
- 64-bit processor
- Consider using the *deadline* I/O scheduler
- Tune the page cache down (but not too much)
- Tune shared memory
- Increase network buffers
- Spread database load over multiple disks



Performance Tuning – Java App Server

- Tune JVM memory settings
- Ensure sufficient network bandwidth
- Buy fast CPUs
- Buy lots of RAM
- Benchmarks!



Tuning C (1/2)

- Problems
 - larger code
 - debugging difficulty
 - errors
 - compile time
- Optimisation levels
 - O0
 - O1
 - O2
 - O3
 - Os
- Profiling

Linux Performance Tuning - 46
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Optimising your C Code

C compilers such as gcc can be instructed to perform optimisations on code that they are compiling which should normally speed up the code or cause it to run faster on a particular type of architecture.

Potential problems

Optimising code is not without problems - since optimisation makes changes to your code, it may introduce unusual bugs, or in some cases, actually slow down the code in edge cases. It will also take longer to compile.

Optimisation Levels

With gcc, and other compilers, rather than turn on specific optimisation features, you can enable sets of features using the `-On` flag with numbers from 0 to 3,

- 0 - turns off all optimisation
- 1 - turns on some optimisation reducing code size and execution time without performing any optimisations that increase compile time significantly.
- 2 - turn on more optimisations except those that will result in increased code size. May be more difficult to debug due to disappearing user variables and function bodies.
- 3 - very aggressive optimisations that may or may not provide better code. Some semantics may be modified, particular relating to floating point. Significant changes to order of execution. Much more difficult to debug.

There is also an `-Os` option which optimises the code for size, applying the O2 options that do not increase code size and some additional size related optimisations. Reducing the code size can have the affect of improving the performance (reducing memory requirements and paging demands).

Profiling

Optimising with profiling is a powerful tool but can be quite time consuming. The basic cycle is to compile your code with profiling turned on (the compiler adds instrumentation to your code to determine where execution time is spent in your code). You then run your code with some "training data" which is reflective of typical program workloads. You then recompile your code using the profile data generated during the training runs - the compiler will use these to guide optimisation.

```
-fprofile-generate  
-fprofile-use
```

Tuning C (2/2)

- Vectorisation
- Architecture-specific code
 - `-march=cpu`
 - `-mtune=cpu`
- Other Compilers
 - Intel
 - C, C++
 - Fortran (F77, F95)
 - The Portland Group
 - C, C++
 - Fortran (F77, F95)

Linux Performance Tuning - 47
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Vectorisation

Modern processors can perform the same operation on multiple pieces of data in a single step.

Enabling this processing can increase performance for some types of workloads, although the performance may degrade in some cases due to the alterations vectorising requires to your code.

```
-ftree-vectorize  
-msse2
```

Architecture specific optimisation

By default, gcc generates generic code which will run on any member of the target processor's family. Significant performance boosts can be had from compiling code for a specific processor.

```
-march=cpu  
-mtune=cpu
```

The only caveat with this is that the code will only work on the targeted processor (or may run on other processors in the family, but significantly slower than normal).

Generally, recommend using O2 and optimising for a particular architecture if appropriate. Fine tuning with specific optimisations is very time consuming and some combinations of flags may actually interfere with each other.

Other compilers

The standard C compiler used on Linux is the GNU C Compiler (GCC), versions of which can be used to compile C, C++, Fortran and other languages. There are also a number of commercial alternatives for specific platforms. GCC's main strength is its portability to different platforms, it is not noted for performance.

Commercial alternatives include,

- *Intel* sell high performance C, C++ and Fortran compilers which give better performing code in at least some cases when compiling code for Intel processors.
- *The Portland Group* sell high performance C, C++ and Fortran compilers for 32-bit and 64-bit x86 platforms (both *Intel* and *AMD* processors). The Portland compilers have a good reputation in the scientific community for HPC applications.

Users of architectures other than x86 may have the option of using a compiler from their hardware vendor.

Tuning Java (1/2)

- General
 - Only performance tune bottleneck code
 - Use the latest JDK
 - Use `StringBuffer` for string concatenation
 - Use `log.isDebugEnabled()`
 - Profile, profile, profile
- JDBC
 - `PreparedStatement`
 - Placeholders
 - Tune the SQL
 - Connection pooling
 - Stored procedures

Linux Performance Tuning - 48
© Applepie Solutions 2004-2008. Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Some specific suggestions

- Don't optimise code that doesn't need to be optimised (this applies to all programming). In general, prefer good programming practices to performance programming practices.
- Use the latest Java release where possible (so at the moment, prefer 1.5 to 1.4). Newer releases contain bug fixes and performance improvements.
- Use the latest Java update – these are point releases for a Java release which contain bug fixes and performance improvements without changing any APIs. Particularly where you cannot move to a newer Java release due to code compatibility issues, ensure you are running the latest update.
- String concatenation is expensive, use `StringBuffer` rather than `String` when concatenating strings.
- When logging, use

```
if (log.isDebugEnabled) {
    log.debug(... )
}
```

to avoid any expensive operations that are only necessary when debug is enabled in the logging framework.
- Valuable resources (resources that are expensive to create such as database connections, open files and so on) should not generally be maintained in instance variables. Use `finally` blocks to ensure resources are released.
- Consider using a messaging framework to make your application more scalable.
- For I/O consider using buffers (such as `BufferedInputStream` or `BufferedReader`). If using your own buffering scheme, investigate the effects of using larger buffers (within the constraints of your memory)
- Use a profiler to identify performance problems – <http://www.yourkit.com>

JDBC Performance

- Use prepared statements.
- Use parametrised SQL (place holders).
- Tune the SQL to minimize the data returned – so select statements should only retrieve the columns you are interested in, not all columns.
- Use connection pooling – creating connections to the database is an expensive process.
- Consider using stored procedures for *hot* parts of your code – they will normally run faster and can avail of the caching facilities of the database software. They do increase the complexity of your system and you will incur maintenance costs if your code is split between stored procedures and application code.
- Cache data to avoid repeated queries.
- Look at tuning the fetch size (can make big difference to performance)
- Use JDBC thin drivers – they are more portable and they generally scale better on multiple processor systems.

Tuning Java (2/2)

- JVM Settings
 - many options
 - heap size
 - garbage collection algorithm
 - `-server`
 - `-client`
- Microbenchmarking gotchas

```
java -server -Xmx512m -Xms512m -XX:+UseConcMarkSweepGC
```

Linux Performance Tuning - 49
© ApplePie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



JVM Settings

The JVM has many tunables (<http://blogs.sun.com/roller/resources/watt/jvm-options-list.html>), we will look at some of the most important. Tuning JVM memory settings is a course in itself!

JVM Heap

The heap is the memory available to JVM for java processes. Note that the total memory consumption of the JVM will exceed the size of the heap as additional memory is allocated for threads and so on. The default JVM heap settings are often too small for an application – if you are experiencing `OutOfMemoryErrors` then your heap size may be too small (or you may have a memory leak!).

- Xms : the initial heap size
- Xmx : the maximum heap size

It is common to set the initial and maximum to the same size. The heap size should never approach the size of physical memory in the system (this will significantly impact the operating system memory and induce unnecessary paging which will degrade the performance of the entire system). For a system running multiple JVMs, the sum of the heaps should never approach the size of physical memory.

JVM Garbage Collector

The garbage collector (GC) is the part of the JVM that reclaims memory from objects which are no longer used making it available for use by other classes. The JVM garbage collector can use a number of different algorithms which suit different types of system,

- XX:+UseParallelGC - parallel (throughput) garbage collector
- XX:+UseConcMarkSweepGC concurrent (low pause time) garbage collector (CMS)
- XX:+UseSerialGC serial garbage collector (for smaller applications and systems)

JVM self tuning

The JVM performs a certain amount of self tuning depending on what kind of system it is running on. The auto-tuning can be over-ridden by specifying 1 of the following options directly to the JVM. The JVM automatically uses `-server` on systems with 2 or more CPUs and 2GB or more of physical memory,

- `-client` (fast start-up, relatively small footprint)
- `-server` (better overall performance)

For example, with the server setting `-Xms` is automatically set to 1/64th of physical mem and `-Xmx` is set to 1/4th of physical mem up to 1GB max.

Microbenchmarking

The JVM performs dynamic optimisation of code it executes – it takes some time after the JVM starts for this optimisation to be applied so you should allow for JVM start-up time in benchmark and run benchmarks for more than a few minutes to ensure valid results.

Application Profiling

- `time`
 - user time and kernel time breakdown
- `top`
 - basic realtime resource usage
- `strace`
 - system call trace
- `gprof`
 - user call graphs
- `valgrind`
 - call graphs
 - memory usage analysis
- `oprofile`
 - system-wide profiling
 - low overhead

Linux Performance Tuning - 50
© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



Introduction

An important step in performance tuning is profiling your application. Profiling involves investigating your application to identify where it is spending its time, and what is consuming resources. There are a wide range of tools for profiling applications at various levels.

`time`

The most basic profiling you can do is to run the `time` command against your application. This will tell you how long your program is running and how much kernel time versus user time it takes. Running the `time` command with the `-v` also gives additional information including statistics on context switches (ensure you are running `/usr/bin/time` rather than a shell built-in if you want to use `-v`).

`top`

The `top` command can give you some high level information about the performance of your program, including how much CPU time it consumes while running and what memory requirements it has.

`strace`

The `strace` command shows what system calls a program makes and can be useful in diagnosing hung programs and to gain an understanding of what system calls the program runs and how frequently calls are made.

`gprof`

Produces a call graph showing the amount of time spent in each routine – useful for tracing performance problems. Programs have to be compiled with profiling instrumentation (use the `-pg` option to `gcc`). Running an instrumented program generates a file called `gmon.out` which contains the call graph data. The `gprof` command can parse and display the contents of this file.

`valgrind`

Valgrind is another profiler – it differs from `gprof` in that it does not require any instrumentation in the code and consists of a number of modules which perform different kinds of profiling. Valgrind is very strong on memory related profiling including tools to detect memory leaks and analyse program cache usage.

`oprofile`

Oprofile is a system-wide profiler that uses hardware counters on modern processors to collect timing information about running programs. It requires no instrumentation in programs, allows you to trace activity involving multiple programs and has a low overhead.

Performance Tuning Exercise 4

- 1) Write a program in Java or C to write 1 million X's to a file. Using some of the lessons learned in the course, test the program and tune it for performance. The following is a sample starting point in Java,

```
import java.io.FileOutputStream;
public class SpeedTest
{
    public static void main (String[]args) throws Exception
    {
        String filename = args[0];
        FileOutputStream fout = new FileOutputStream (filename);
        for (int i = 0; i < 1000000; i++)
        {
            fout.write ('X');
        }
        fout.close ();
    }
}
```

Linux Performance Tuning - 51

© Applepie Solutions 2004-2008. Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License



In closing ...

- Summary
- Next steps
- Questionnaire

Thank you and well done!



Linux Performance Tuning - 53

© Applepie Solutions 2004-2008, Some Rights Reserved
Licensed under a Creative Commons Attribution-Non-Commercial-Share Alike 3.0 Unported License

