

Tuning the Linux Kernel

Long Yi, James Connan

Department of Computer Science

University of the Western Cape, Bellville, South Africa, 7535

Tel:+(27) (0)21 959-3010 Fax: +(27) (0)21 959-3006

Email: 2475600@uwc.ac.za, jconnan@uwc.ac.za

Abstract—Linux continues to attract telecommunication industry attention. Optimising the Linux kernel can improve system performance and reduce system total costs. The Linux kernel has a large number of parameters to fine-tune system performance available to the system administrator. We focus on the kernel parameters that are most relevant to adjust for improving performance for specific system classes. We optimise the Linux kernel for two important applications: Web servers and Database servers. The results show that our optimised GNU/Linux system running the Kernel 2.4.29 achieves effective performance improvements.

Index Terms—Linux kernel, Tuning, operating system, optimisation, performance, benchmark, workload, open source, system profiler.

I. INTRODUCTION

LINUX continues to attract telecommunication industry attention. Tradition telecommunications were built on proprietary systems that had to meet specific requirements such as availability, reliability and performance. Those systems were composed of specific vendor hardware, operating system and middleware. Such systems were limited to design flexibility and performance, also produced systems that were expensive and difficult to expend. As a result, the telecommunication industry is moving toward open source systems. Open Source Development Labs (OSDL) has published its Carrier Grade Linux (CGL) definition and is dedicated to accelerate Linux deployment in the telecommunication industry. Linux kernel is the core of the linux operating system, optimising the kernel can improve system performance and reduce system total costs.

Linux kernel optimisation is the manipulation of kernel parameters in order to achieve maximal possible performance, or reach a target workload for an acceptable cost within given constraints. To increase flexibility and improve performance, system administrators tune the kernel using some so-called “rules of thumb”, adjusting parameters such as network cache’s size or the kernel’s behaviour for a particular system classes. Our optimisation strategy is dependent on the system class and the functions performed by the server. Therefore, it is important to understand the intended use of the system and the performance constraints of the particular system. We choose two system classes: Web server and Database server to test in this paper. The system classes are described below:

- *Web servers*

The role of web servers are to host web pages and run web applications. Because of high hit ratios and the

transfer of large objects the network is very important sites [13].

If web site content is static, the subsystem that has the most impact on the web servers’ performance is the network subsystem. If the web site is dynamic, the subsystem that has the most impact on web servers’ performance is the memory subsystem [4].

- *Database servers* The role of the database server is to store, search, retrieve and update data from disk.

The subsystem that has the most impact on database servers performance is the memory subsystem [4].

II. GNU/LINUX AND ITS KERNEL

GNU/Linux is a free, popular UNIX clone operating system. Linux is the name given to the Linux kernel by its creator, Linus Torvalds and it was initially created in 1991. In common usage, Linux refers to the entire operating system which is based on the Linux kernel and GNU software. Linux is the *kernel*, an essential part of an operating system, which refers to the low-level system software that provides an abstraction layer between hardware and software. It enables the operating system to manage all the hardware and resources such as: hardware devices, memory assignment, data store, processes, workload balance, networking, running applications and security. A complete Linux operating system is composed of GNU software and the Linux kernel. The Free Software Foundation (FSF) ask that the Linux system to be referred as “GNU/Linux” [13].

The kernel is the core part of a GNU/Linux operating system. The other parts of the system cannot run on a machine if the kernel does not participate as part of the entire system [8]. As such, it makes sense to discuss the kernel in the context of an entire system. Figure 1 shows the structured overview of a GNU/Linux operating system.

The GNU/Linux operating system is composed of four major subsystems:

1. *Applications*. The set of user software in use on a GNU/Linux system. Applications allow users to perform one or more specific tasks. Typical applications include word processors, spreadsheets, business programs, databases and games. The applications installed on the GNU/Linux depend on what the machine is used for [13].

2. *Shell*. It is typically considered part of a GNU/Linux system and it provides an operating interface through which users can interact with the system and issue commands. It

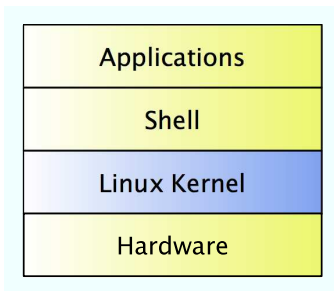


Fig. 1. The structure of a GNU/Linux operating system

is the utility that processes user requests [13]. Some certain application can interface with the kernel directly without shell.

3. *Kernel*. A single executable program, loaded into memory at boot time and remaining there until the system is powered down. The kernel abstracts hardware devices and acts as an intermediary between the hardware and shell and provides hardware devices, memory and processor management functions, e.g. handling interrupt requests from hardware devices, sharing memory and the processor(s) among processes, etc. User applications can interface with the kernel through system calls [13].

4. *Hardware*. All the possible physical devices in a computer, for example, processor, memory, hard disk, network cards, etc [13].

Each of the layers in Figure 1 can only communicate with adjacent layers. In addition, the dependencies between layers are from top to bottom: higher layers depend on lower layers. Since the primary interest of this research is the kernel, we will ignore the application, shell and hardware layers and focus on the kernel layer only.

The kernel is the heart of a GNU/Linux operating system. It has two main tasks:

1. To serve low level hardware requirements;
2. To provide an environment for running programs.

The function of the kernel is to present a virtual machine for user processes and hide the underlying hardware [2]. The kernel (1) *controls and mediates access to hardware*, (2) *implements and supports fundamental abstractions*—processes, files, devices etc., (3) *allocates and releases system resources*—memory, CPU, disk, descriptors, etc., (4) *enforces security and protection*, and (5) *responds to user requests for service*—system calls. In addition, the kernel (6) *supports a multi-tasking computing environment* that is transparent to user processes. Each process acts as if it is the only process running on the computer, with exclusive use of memory and other hardware resources. The kernel actually runs multiple processes concurrently and rapidly switches back and forth among the processes. Figure 2 shows the divided components of the kernel.

Ivan Bowman [8] divides the Linux kernel into five logically divided components. We regard *device drivers* as an essential component of the kernel and regard its components to be:

1. *Process scheduler*. It controls the way processes access the processor. The scheduler applies a policy to ensure that each process on the computer gets its fair share of the

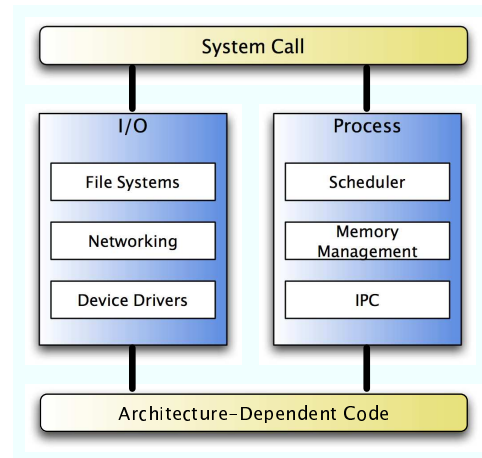


Fig. 2. The components of the kernel

processor[3], [8].

2. *Memory manager*. It keeps track of the whole memory map, allocates and de-allocates memory to processes, and manages swapping between main memory and disk. It permits multiple processes to share the main memory simultaneously [3], [8].

3. *Interprocess communication (IPC)*. It allows one process to communicate with another process. It is required in all modern operating systems [3], [8].

4. *File System*. Its job is to keep track of the whole disk map, to allocate and de-allocate sectors to files, and to manage files and directories. In addition, it abstracts the various hardware devices' details by presenting a common file interface [3], [8].

5. *Device Drivers*. They control different hardware devices. They issue commands to the devices, interact with them and provide interfaces to the devices that are simple and easy to access.

6. *Networking*. It provides access to network function using several networking protocols [3], [8].

We will discuss the optimisation parameters of the various kernel components in Section IV-C.

III. EXPLORING /PROC

The /proc file-system is a real-time reflection of the running system kept in memory and represented in a hierarchical manner. The job of the /proc file-system is to track the information of the system and running processes and to provide an easy way to view such information. The /proc is a pseudo file-system residing in the virtual memory and it does not exist on any physical media. The premise behind the /proc file-system is to view the state of the system and information of currently running processes by easily reading from related files in the /proc instead of having difficult to understand system calls. For security reasons, the content of the /proc can only be read or written to by users with the appropriate authority. The /proc file-system can provide information such as:

- Viewing hardware information
- Viewing kernel runtime status and memory/disk/network statistical information

And can modify information such as:

- Modifying kernel runtime parameters
- Modifying memory, disk, network parameters

The following list [9], [10] briefly describes a few files in the `/proc` file-system.

- `loadavg`

Example of `cat loadavg`

```
0.00 0.00 0.00 3/57 28964
```

This file contains system load information about the kernel. The first three entries represent the average number of active tasks on the system that were actually running over the last 1, 5, and 15 minutes. The next entry is the number of currently runnable processes that are scheduled to run and the total number of processes on the system. The final entry is the process ID of the process that ran most recently.

- `uptime`

Example of `cat /proc/uptime`

```
1588073.71 1009804.77
```

This file contains the time in seconds since the system was booted and the total time used by processes. Both of these values are given as floating point values, in seconds.

IV. KERNEL OPTIMISATION

Another important part of the `/proc` file-system is the directory `/proc/sys`. This directory not only provides information of the running kernel, it also allows administrators to modify the value of the parameters that the kernel uses to determine behaviour. The files in `/proc/sys` can be used to monitor and to optimise general and miscellaneous operation of the Linux kernel. Unlike most other directories in the `/proc` file-system, `/proc/sys` variables are typically writable, and are used to adjust the running kernel rather than simply monitoring currently running processes and system information. The value of a parameter can be changed by simply applying the new value to the related file using the `echo` or the `sysctl` command. We will focus on `/proc/sys/kernel`, `/proc/sys/fs`, `/proc/sys/vm`, and `/proc/sys/net`, which are used to tune the kernel, file-system, virtual memory and disk buffers, and networking respectively. The 2.4.29 kernel has many parameters that can be used for improving performance [1], [11]. In the remainder of this section, we present several areas of `sysctl` that can result in large performance improvements.

A. Overview of the Kernel Parameters

The definitions of the kernel parameters can be found in the Linux kernel 2.4.29 documents which can be found in the kernel source package [9]. The list below shows the kernel parameters that are most relevant to performance [4], [5], [12], [7], [11].

- `/proc/sys/net/ipv4/inet_peer_gc_maxtime`
- `/proc/sys/net/ipv4/inet_peer_gc_mintime`
- `/proc/sys/net/ipv4/inet_peer_maxttl`
- `/proc/sys/net/ipv4/inet_peer_minttl`
- `/proc/sys/net/ipv4/inet_peer_threshold`

- `/proc/sys/vm/hugetlb_pool`
- `/proc/sys/vm/inactive_clean_percent`
- `/proc/sys/vm/pagetable_cache`
- `/proc/sys/fs/file-nr`
- `/proc/sys/fs/file-max`
- `/proc/sys/vm/bdflush`
- `/proc/sys/vm/kswapd`
- `/proc/sys/net/ipv4/tcp_max_syn_backlog`
- `/proc/sys/net/ipv4/ip_local_port_range`
- `/proc/sys/net/ipv4/tcp_wmem`
- `/proc/sys/net/ipv4/tcp_rmem`
- `/proc/sys/net/ipv4/tcp_keepalive_time`
- `/proc/sys/net/core/wmem_max`
- `/proc/sys/net/core/rmem_max`

The following lists the kernel parameters that are relevant but not typically used in performance tuning [4], [5].

- `/proc/sys/kernel/panic`
- `/proc/sys/kernel/pid_max`
- `/proc/sys/net/ipv4/tcp_tw_recycle`
- `/proc/sys/vm/overcommit_ratio`

B. Using `sysctl`

`sysctl` is an interface that exports the ability to tune kernel parameters in a running Linux system. It is easy to make changes to the kernel parameters by issuing the `sysctl` command. For example, to modify the `file-max` kernel parameter, `root` can alter the value of the parameter with two different commands:

- using `echo` command


```
cat /proc/sys/fs/file-max
26188
echo 30000 > /proc/sys/fs/file-max
cat /proc/sys/fs/file-max
30000
```
- using the `sysctl` command


```
sysctl fs.file-max
fs.file-max = 26188
sysctl -w fs.file-max=30000
fs.file-max = 30000
sysctl fs.file-max
fs.file-max = 30000
```

Notice that using the `echo` command can easily introduce errors, so we prefer using `sysctl` because it checks the consistency of the parameter before it makes change. The system call `sysctl` is available to programmers. It is an alternative way to change parameters rather than modifying parameters by using read/write system calls. The advantage of `sysctl` is that it is faster, as no fork is executed nor any directory look-up. To use `sysctl` in a C program, the header file `linux/sysctl.h` must be included. The declaration is like this:

```
int sysctl (int *name, int nlen, void
*oldval, size_t *oldlenp, void *newval,
size_t newlen);
```

C. Suggested Values for Optimisation

There are no set rules for adjusting these parameters as they are very dependent on the system class. Our approach of finding the ‘right’ value of the parameters is by experimentation. We wrote a script to adjust the value periodically and measured its effect until we found the ‘right’ value for the target system class. These values are examined on our test machine. The IV and V (See Appendix) describe the parameters that can achieve the most improvement in performance and their best possible values for two system classes: web server class and database server class respectively.

V. EXPERIMENTAL RESULTS

Our experimental result shows that the optimisations achieved substantial performance improvements for our test workloads.

A. Experimental Details

Our experimental system used the Linux kernel 2.4.29. We ran our experiments on two Intel x86-based PCs. One machine represented a server from one of our chosen system classes (Web server and Database server). The other machine was a client—a workload generator to simulate a real computing environment. The two systems both include a 512 KB second-level cache and 256 MB of main memory and they have exactly the same hardware configuration. The server was initially installed by a Gentoo Linux boot CD 2004.2. Gentoo is a source-based Linux distribution and the system has been updated to its latest version. The client is based on SUSE Linux 10.0. We used three workloads: a web server workload and a database workload for the three system classes in this paper. The chosen web server and database server version are: Apache 2.0.55 and MySQL 4.1.16.

Table I gives a description of each workload generator. Table II gives summary statistics for the workloads. All experiments for this paper were run in single-system-class mode. Two factors limited our choice of workloads. First, the system workload must be easily simulated by existing tools. Second, a benchmark tool for the workload must be readily available. As a result we were unable to simulate workloads for many popular system classes (DNS server, File server, Print server, etc).

B. Optimisation Results

We optimised the system, and then we benchmarked the optimised system. Table III shows optimisation results of the two system classes. Each optimisation improvement given in Table III are computed from the average of ten tests in Table VI and VII. Table III shows that in each case the optimised system performs better than the original system. The optimisation results show that the benefits of the parameter-driven optimisation are significant.

TABLE I
WORKLOAD SIMULATION

System Class	Workload Generator
web server	http_load runs multiple http fetches in parallel, to simulate a web server workload. It can also be considered a web server benchmark tool. Example: ./http_load -rate 10 -seconds 30 urls
database server	The mysql test suit is a database benchmark tool. It includes test_insert, test_select, test_connect and test_create tools. They can be found in the mysql package. They can also be used as sql simulation generators to simulate database workloads. Example: ./test_insert;./test_select

TABLE II
WORKLOAD STATISTIC

Workload Generator	Number of Users	Total Time
http_load	1	1000
mysql test	1	1000

VI. SUMMARY

This paper discusses Linux kernel optimisation for the x86 (32bit) platform. The Linux kernel offers a large number of parameters to fine-tune performance. The actual choice of parameters depends on the type of application currently running on the machine. These parameters are accessible through an extensive kernel interface represented by files in the /proc filesystem. The parameters can be adjusted by changing the related files using the *echo* or the *sysctl* command. We evaluated two important applications: Web server and Database server with their suggested choice of parameters. Our results show that the optimisations are effective. The same test script should be able to find appropriate Linux parameters for telecommunication applications, such as VOIP server, PBX server, etc. Unfortunately, there are no benchmark and simulation tools available for those applications. We hope to apply our test and script to telecommunication applications in the future.

REFERENCES

- [1] Alessandro Rubini, *The sysctl Interface*. 1997. Available at <http://www.linuxjournal.com/article/2365>
- [2] Andrew Tanenbaum, *Modern Operating Systems 2ed*. Prentice Hall, 2001
- [3] David A Rusling, *The Linux Kernel*. 1999. Available at <http://ldp.org/LDP/tlk/tlk.html>
- [4] David Watts and Martha Centeno and Raymond Phillips and Luciano Magalhes Tome, *Tuning Red Hat Enterprise Linux on IBM Eserver xSeries Servers*. IBM Corp., 2004. Available at <http://www.redbooks.ibm.com/redpapers/pdfs/redp3861.pdf>

TABLE III
PERFORMANCE RESULTS

System Class	Test Tool	Improvement
Web server	time, httpperf	6.16%
Database server	time, test-insert	8.04%

- [5] Dustin Puryear, *Linux Kernel Tuning Using System Control*. 2003. Available at <http://www.samag.com/documents/s=8920/sam0311a/0311a.htm>
- [6] FSF, *GNU Public License*. Available at <http://www.gnu.org/licenses/gpl.html>
- [7] Gian-Paolo D. Musumeci, Mike Loukides, *System Performance Tuning, 2nd Edition*. O'Reilly Media, 2002.
- [8] Ivan Bowman, *Concrete Architecture of the Linux Kernel*. 1998. Available at <http://plg.uwaterloo.ca/~itbowman/CS746G/a2>
- [9] Linus Torvalds with the assistance of developers around the world, *linux-2.4.29 source*. Available at <http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.29.tar.bz2>
- [10] Olaf Kirch, Terry Dawson, *Linux Network Administrator's Guide, Second Edition*. O'Reilly, 2000.
- [11] Phillip Ezolt, *Optimizing Linux Performance*. O'Reilly, 2005.
- [12] Sandra K. Johnson, Gerrit Huizenga, Badari Pulavarty, *Performance Tuning for Linux(R) Servers*. IBM Press, 2005.
- [13] Wikipedia, *Wikipedia*. 2006. Available at <http://en.wikipedia.org/wiki/>.

Long Yi received the B.S. degree in computer science from Wuhan University of Science and Technology, China, in 2002, and the M.S. degree (cum laude) from University of the Western Cape, South Africa, in 2007, respectively. From 2002 to 2004, he served as a software engineer at a leading software company (Neusoft) in China. He is currently working on the Broadband Applications and Networks Group (BANG) project in UWC.

APPENDIX

TABLE IV
TUNED VALUES FOR WEB SERVERS

Parameter	Suggested Value
net.ipv4.inet_peer_gc_maxtime	240
net.ipv4.inet_peer_gc_mintime	80
net.ipv4.inet_peer_maxttl	500
net.ipv4.inet_peer_minttl	80
net.ipv4.inet_peer_threshold	65644
vm.hugetlb_pool	4608
vm.inactive_clean_percent	30
vm.pagecache	50 100
vm.bdflush	30 500 0 0 500 3000 80 20 0
vm.kswapd	1024 32 64
net.ipv4.tcp_max_syn_backlog	8192
net.ipv4.ip_local_port_range	16384 65536
net.ipv4.tcp_wmem	4096 131072 262144
net.ipv4.tcp_rmem	4096 87380 174760
net.ipv4.tcp_keepalive_time	1800
net.core.wmem_max	262144
net.core.rmem_max	103424
net.ipv4.ip_forward	0
net.ipv4.conf.all.rp_filter	1
net.ipv4.conf.lo.rp_filter	1
net.ipv4.conf.eth0.rp_filter	1
net.ipv4.conf.default.rp_filter	1
net.ipv4.conf.all.accept_redirects	0
net.ipv4.conf.lo.accept_redirects	0
net.ipv4.conf.eth0.accept_redirects	0
net.ipv4.conf.default.accept_redirects	0
net.ipv4.conf.all.accept_redirects	0
net.ipv4.conf.lo.accept_redirects	0
net.ipv4.conf.eth0.accept_redirects	0
net.ipv4.conf.default.accept_redirects	0
net.ipv4.tcp_fin_timeout	15
net.ipv4.tcp_window_scaling	0
net.ipv4.tcp_sack	0
net.ipv4.tcp_timestamps	0
net.ipv4.icmp_echo_ignore_broadcasts	1
net.ipv4.icmp_ignore_bogus_error_responses	1
net.ipv4.conf.all.log_martians	1
net.ipv4.tcp_max_tw_buckets	1440000
net.ipv4.tcp_sack	0
net.ipv4.tcp_timestamps	0

TABLE V
TUNED VALUES FOR DATABASE SERVERS

Parameter	Suggested Value
net.ipv4.inet_peer_gc_maxtime	240
net.ipv4.inet_peer_gc_mintime	80
net.ipv4.inet_peer_maxttl	500
net.ipv4.inet_peer_minttl	80
net.ipv4.inet_peer_threshold	65644
vm.hugetlb_pool	4608
vm.inactive_clean_percent	30
vm.pagecache	50 100
vm.bdflush	30 500 0 0 500 3000 60 20 0
vm.kswapd	1024 32 64
net.ipv4.tcp_max_syn_backlog	1024
net.ipv4.ip_local_port_range	1024 4999
net.ipv4.tcp_wmem	4096 87380 8388608
net.ipv4.tcp_rmem	4096 87380 8388608
net.ipv4.tcp_keepalive_time	7200
net.core.wmem_max	8388608
net.core.rmem_max	8388608
kernel.shmmax	268435456
kernel.msgmni	1024
fs.file-max	8192
kernel.sem	250 32000 32 1024
kernel.sem	500 512000 64 2048
kernel.msgmni	2048
kernel.msgmax	64000
net.ipv4.tcp_tw_reuse	1
net.ipv4.tcp_tw_recycle	1

TABLE VII
DATABASE SERVER RESULTS

Test	Tool	Default	Optimised	Improvement
1	time, test-insert	36.263	33.656	7.2%
2	time, test-insert	36.871	33.491	9.2%
3	time, test-insert	36.123	33.131	8.3%
4	time, test-insert	36.443	33.154	9.0%
5	time, test-insert	36.754	33.212	9.6%
6	time, test-insert	36.278	33.512	7.6%
7	time, test-insert	36.218	33.625	7.2%
8	time, test-insert	36.389	33.643	7.5%
9	time, test-insert	36.261	33.712	7.0%
10	time, test-insert	36.411	33.589	7.8%

TABLE VI
WEB SERVER RESULTS

Test	Tool	Default	Optimised	Improvement
1	time, httpperf	50.956	47.283	7.2%
2	time, httpperf	50.993	47.043	7.7%
3	time, httpperf	50.453	47.843	5.2%
4	time, httpperf	50.879	47.409	6.0%
5	time, httpperf	51.334	48.943	4.7%
6	time, httpperf	50.339	47.463	5.7%
7	time, httpperf	50.421	47.579	5.6%
8	time, httpperf	50.842	47.085	7.4%
9	time, httpperf	50.563	47.467	6.1%
10	time, httpperf	50.675	47.636	6.0%