Linux Benchmarking HOWTO

Table of Contents

Linux Benchmarking HOWTO

by André D. Balsa, andrewbalsa@usa.net

v0.12, 15 August 1997

The Linux Benchmarking HOWTO discusses some issues associated with the benchmarking of Linux systems and presents a basic benchmarking toolkit, as well as an associated form, which enable one to produce significant benchmarking information in a couple of hours. Perhaps it will also help diminish the amount of useless articles in comp.os.linux.hardware...

1.Introduction

2.Benchmarking procedures and interpretation of results

3.The Linux Benchmarking Toolkit (LBT)

4.Example run and results

5.Pitfalls and caveats of benchmarking

6.FAQ

7.Copyright, acknowledgments and miscellaneous

1.Introduction

"What we cannot speak about we must pass over in silence."

> *Ludwig Wittgenstein (1889−1951), Austrian philosopher*

Benchmarking means **measuring** the speed with which a computer system will execute a computing task, in a way that will allow comparison between different hard/software combinations. It **does not** involve user−friendliness, aesthetic or ergonomic considerations or any other subjective judgment.

Benchmarking is a tedious, repetitive task, and takes attention to details. Very often the results are not what one would expect, and subject to interpretation (which actually may be the most important part of a benchmarking procedure).

Finally, benchmarking deals with facts and figures, not opinion or approximation.

1.1 Why is benchmarking so important ?

Apart from the reasons pointed out in the BogoMips Mini−HOWTO (section 7, paragraph 2), one occasionally is confronted with a limited budget and/or minimum performance requirements while putting together a Linux box. In other words, when confronted with the following questions:

- How do I maximize performance within a given budget ?
- How do I minimize costs for a required minimum performance level ?
- How do I obtain the best performance/cost ratio (within a given budget or given performance requirements)?

one will have to examine, compare and/or produce benchmarks. Minimizing costs with no performance requirements usually involves putting together a machine with leftover parts (that old 386SX−16 box lying around in the garage will do fine) and does not require benchmarks, and maximizing performance with no cost ceiling is not a realistic situation (unless one is willing to put a Cray box in his/her living room − the leather−covered power supplies around it look nice, don't they ?).

Benchmarking per se is senseless, a waste of time and money; it is only meaningful as part of a decision process, i.e. if one has to make a choice between two or more alternatives.

Usually another parameter in the decision process is **cost**, but it could be availability, service, reliability, strategic considerations or any other rational, measurable characteristic of a computer system. When comparing the performance of different Linux kernel versions, for example, **stability** is almost always more important than speed.

1.2 Invalid benchmarking considerations

Very often read in newsgroups and mailing lists, unfortunately:

1. Reputation of manufacturer (unmeasurable and meaningless).
2. Market share of manufacturer (meaningless and irrelevant).
3. Irrational parameters (for example, superstition or prejudice: would you buy a processor labeled 131313ZAP and painted pink ?)
4. Perceived value (meaningless, unmeasurable and irrational).
5. Amount of marketing hype: this one is the worst, I guess. I personally am fed up with the "XXX inside" or "kkkkkws compatible" logos (now the "aaaaaPowered" has joined the band − what next ?). IMHO, the billions of dollars spent on such campaigns would be better used by research teams on the design of new, faster, (cheaper :−) bug−free processors. No amount of marketing hype will remove a floating−point bug in the FPU of the brand−new processor you just plugged in your motherboard, but an exchange against a redesigned processor will.
6. "You get what you pay for" opinions are just that: opinions. Give me the facts, please.

2.Benchmarking procedures and interpretation of results

A few semi−obvious recommendations:

1. First and foremost, **identify your benchmarking goals**. What is it you are exactly trying to benchmark ? In what way will the benchmarking process help later in your decision making ? How much time and resources are you willing to put into your benchmarking effort ?
2. **Use standard tools**. Use a current, stable kernel version, standard, current gcc and libc and a standard benchmark. In short, use the LBT (see below).
3. Give a **complete description** of your setup (see the LBT report form below).
4. Try to **isolate a single variable**. Comparative benchmarking is more informative than "absolute" benchmarking. **I cannot stress this enough.**
5. **Verify your results**. Run your benchmarks a few times and verify the variations in your results, if any. Unexplained variations will invalidate your results.
6. If you think your benchmarking effort produced meaningful information, **share it** with the Linux community in a **precise** and **concise** way.
7. Please **forget about BogoMips**. I promise myself I shall someday implement a very fast ASIC with the BogoMips loop wired in. Then we shall see what we shall see !

2.1 Understanding benchmarking choices

Synthetic vs. applications benchmarks

Before spending any amount of time on benchmarking chores, a basic choice must be made between "synthetic" benchmarks and "applications" benchmarks.

Synthetic benchmarks are specifically designed to measure the performance of individual components of a computer system, usually by exercising the chosen component to its maximum capacity. An example of a well−known synthetic benchmark is the **Whetstone** suite, originally programmed in 1972 by Harold Curnow in FORTRAN (or was that ALGOL ?) and still in widespread use nowadays. The Whestone suite will measure the floating−point performance of a CPU.

The main critic that can be made to synthetic benchmarks is that they do not represent a computer system's performance in real−life situations. Take for example the Whetstone suite: the main loop is very short and will easily fit in the primary cache of a CPU, keeping the FPU pipeline constantly filled and so exercising the FPU to its maximum speed. We cannot really criticize the Whetstone suite if we remember it was programmed 25 years ago (its design dates even earlier than that !), but we must make sure we interpret its results with care, when it comes to benchmarking modern microprocessors.

Another very important point to note about synthetic benchmarks is that, ideally, they should tell us something about a **specific** aspect of the system being tested, independently of all other aspects: a synthetic benchmark for Ethernet card I/O throughput should result in the same or similar figures whether it is run on a 386SX−16 with 4 MBytes of RAM or a Pentium 200 MMX with 64 MBytes of RAM. Otherwise, the test will be measuring the overall performance of the CPU/Motherboard/Bus/Ethernet card/Memory subsystem/DMA combination: not very useful since the variation in CPU will cause a greater impact than the change in Ethernet network card (this of course assumes we are using the same kernel/driver combination, which could cause an even greater variation)!

Finally, a very common mistake is to average various synthetic benchmarks and claim that such an average is a good representation of real−life performance for any given system.

Here is a comment on FPU benchmarks quoted with permission from the Cyrix Corp. Web site:

> *"A Floating Point Unit (FPU) accelerates software designed to use floating point mathematics : typically CAD programs, spreadsheets, 3D games and design applications. However, today's most popular PC applications make use of both floating point and integer instructions. As a result, Cyrix chose to emphasize "parallelism" in the design of the 6x86 processor to speed up software that intermixes these two instruction types.*
>
> *The x86 floating point exception model allows integer instructions to issue and complete while a floating point instruction is executing. In contrast, a second floating point instruction cannot begin execution while a previous floating point instruction is executing. To remove the performance limitation created by the floating point exception model, the 6x86 can speculatively issue up to four floating point instructions to the on−chip FPU while continuing to issue and execute integer instructions. As an example, in a code sequence of two floating point instructions (FLTs) followed by six integer instructions (INTs) followed by two FLTs, the 6x86 processor can issue all ten instructions to the appropriate execution units prior to completion of the first FLT. If none of the instructions fault (the typical case), execution continues with both the integer and floating point units completing instructions in parallel. If one of the FLTs faults (the atypical case), the speculative execution capability of the 6x86 allows the processor state to be restored in such a way that it is compatible with the x86 floating point exception model.*
>
> *Examination of benchmark tests reveals that synthetic floating point benchmarks use a pure floating point−only code stream not found in real−world applications. This type of benchmark does not take advantage of the speculative execution capability of the 6x86 processor. Cyrix believes that non−synthetic benchmarks based on real−world applications better reflect the actual performance users will achieve. Real−world applications contain intermixed integer and floating point instructions and therefore benefit from the 6x86 speculative execution capability."*

So, the recent trend in benchmarking is to choose common applications and use them to test the performance of complete computer systems. For example, **SPEC**, the non−profit corporation that designed the well−known SPECINT and SPECFP synthetic benchmark suites, has launched a project for a new applications benchmark suite. But then again, it is very unlikely that such commercial benchmarks will ever include any Linux code.

Summarizing, synthetic benchmarks are valid as long as you understand their purposes and limitations. Applications benchmarks will better reflect a computer system's performance, but none are available for Linux.

High−level vs. low−level benchmarks

Low−level benchmarks will directly measure the performance of the hardware: CPU clock, DRAM and cache SRAM cycle times, hard disk average access time, latency, track−to−track stepping time, etc... This can be useful in case you bought a system and are wondering what components it was built with, but a better way to check these figures would be to open the case, list whatever part numbers you can find and somehow obtain the data sheet for each part (usually on the Web).

Another use for low−level benchmarks is to check that a kernel driver was correctly configured for a specific piece of hardware: if you have the data sheet for the component, you can compare the results of the low−level benchmarks to the theoretical, printed specs.

High−level benchmarks are more concerned with the performance of the hardware/driver/OS combination for a specific aspect of a microcomputer system, for example file I/O performance, or even for a specific hardware/driver/OS/application performance, e.g. an Apache benchmark on different microcomputer systems.

Of course, all low−level benchmarks are synthetic. High−level benchmarks may be synthetic or applications benchmarks.

2.2 Standard benchmarks available for Linux

IMHO a simple test that anyone can do while upgrading any component in his/her Linux box is to launch a kernel compile before and after the hard/software upgrade and compare compilation times. If all other conditions are kept equal then the test is valid as a measure of compilation performance and one can be confident to say that:

> "Changing A to B led to an improvement of x % in the compile time of the Linux kernel under such and such conditions".

No more, no less !

Since kernel compilation is a very usual task under Linux, and since it exercises most functions that get exercised by normal benchmarks (except floating−point performance), it constitutes a rather good **individual** test. In most cases, however, results from such a test cannot be reproduced by other Linux users because of variations in hard/software configurations and so this kind of test cannot be used as a "yardstick" to compare dissimilar systems (unless we all agree on a standard kernel to compile − see below).

Unfortunately, there are no Linux−specific benchmarking tools, except perhaps the Byte Linux Benchmarks which are a slightly modified version of the Byte Unix Benchmarks dating back from May 1991 (Linux mods by Jon Tombs, original authors Ben Smith, Rick Grehan and Tom Yager).

There is a central Web site for the Byte Linux Benchmarks.

An improved, updated version of the Byte Unix Benchmarks was put together by David C. Niemi. It is called UnixBench 4.01 to avoid confusion with earlier versions. Here is what David wrote about his mods:

> *"The original and slightly modified BYTE Unix benchmarks are broken in quite a number of ways which make them an unusually unreliable indicator of system performance. I intentionally made my "index" values look a lot different to avoid confusion with the old benchmarks."*

David has setup a majordomo mailing list for discussion of benchmarking on Linux and competing OSs. Join with "subscribe bench" sent in the body of a message to majordomo@wauug.erols.com. The Washington Area Unix User Group is also in the process of setting up a Web site for Linux benchmarks.

Also recently, Uwe F. Mayer, mayer@math.vanderbilt.edu ported the BYTE Bytemark suite to Linux. This is a modern suite carefully put together by Rick Grehan at BYTE Magazine to test the CPU, FPU and memory system performance of modern microcomputer systems (these are strictly processor−performance oriented benchmarks, no I/O or system performance is taken into account).

Uwe has also put together a Web site with a database of test results for his version of the Linux BYTEmark benchmarks.

While searching for synthetic benchmarks for Linux, you will notice that sunsite.unc.edu carries few benchmarking tools. To test the relative speed of X servers and graphics cards, the xbench−0.2 suite by Claus Gittinger is available from sunsite.unc.edu, ftp.x.org and other sites. Xfree86.org refuses (wisely) to carry or recommend any benchmarks.

The XFree86−benchmarks Survey is a Web site with a database of x−bench results.

For pure disk I/O throughput, the hdparm program (included with most distributions, otherwise available from sunsite.unc.edu) will measure transfer rates if called with the −t and −T switches.

There are many other tools freely available on the Internet to test various performance aspects of your Linux box.

2.3 Links and references

The comp.benchmarks.faq by Dave Sill is the standard reference for benchmarking. It is not Linux specific, but recommended reading for anybody serious about benchmarking. It is available from a number of FTP and web sites and lists **56 different benchmarks**, with links to FTP or Web sites that carry them. Some of the benchmarks listed are commercial (SPEC for example), though.

I will not go through each one of the benchmarks mentionned in the comp.benchmarks.faq, but there is at least one low−level suite which I would like to comment on: the lmbench suite, by Larry McVoy. Quoting David C. Niemi:

> *"Linus and David Miller use this a lot because it does some useful low−level measurements and can also measure network throughput and latency if you have 2 boxes to test with. But it does not attempt to come up with anything like an overall "figure of merit"..."*

A rather complete FTP site for **freely** available benchmarks was put together by Alfred Aburto. The Whetstone suite used in the LBT can be found at this site.

There is a **multipart FAQ by Eugene Miya** that gets posted regularly to comp.benchmarks; it is an excellent reference.

3.The Linux Benchmarking Toolkit (LBT)

I will propose a basic benchmarking toolkit for Linux. This is a preliminary version of a comprehensive Linux Benchmarking Toolkit, to be expanded and improved. Take it for what it's worth, i.e. as a proposal. If you don't think it is a valid test suite, feel free to email me your critics and I will be glad to make the changes and improve it if I can. Before getting into an argument, however, read this HOWTO and the mentionned references: informed criticism is welcomed, empty criticism is not.

3.1 Rationale

This is just common sense:

1. It should not take a whole day to run. When it comes to comparative benchmarking (various runs), nobody wants to spend days trying to figure out the fastest setup for a given system. Ideally, the entire benchmark set should take about 15 minutes to complete on an average machine.
2. All source code for the software used must be freely available on the Net, for obvious reasons.
3. Benchmarks should provide simple figures reflecting the measured performance.
4. There should be a mix of synthetic benchmarks and application benchmarks (with separate results, of course).
5. Each **synthetic** benchmarks should exercise a particular subsystem to its maximum capacity.
6. Results of **synthetic** benchmarks should **not** be averaged into a single figure of merit (that defeats the whole idea behind synthetic benchmarks, with considerable loss of information).
7. Applications benchmarks should consist of commonly executed tasks on Linux systems.

3.2 Benchmark selection

I have selected five different benchmark suites, trying as much as possible to avoid overlap in the tests:

1. Kernel 2.0.0 (default configuration) compilation using gcc.
2. Whetstone version 10/03/97 (latest version by Roy Longbottom).
3. xbench−0.2 (with fast execution parameters).
4. UnixBench benchmarks version 4.01 (partial results).
5. BYTE Magazine's BYTEmark benchmarks beta release 2 (partial results).

For tests 4 and 5, "(partial results)" means that not all results produced by these benchmarks are considered.

3.3 Test duration

1. Kernel 2.0.0 compilation: 5 − 30 minutes, depending on the **real** performance of your system.
2. Whetstone: 100 seconds.
3. Xbench−0.2: < 1 hour.
4. UnixBench benchmarks version 4.01: approx. 15 minutes.
5. BYTE Magazine's BYTEmark benchmarks: approx. 10 minutes.

3.4 Comments

Kernel 2.0.0 compilation:

- **What:** it is the only application benchmark in the LBT.
- The code is widely available (i.e. I finally found some use for my old Linux CD−ROMs).
- Most linuxers recompile the kernel quite often, so it is a significant measure of overall performance.
- The kernel is large and gcc uses a large chunk of memory: attenuates L2 cache size bias with small tests.
- It does frequent I/O to disk.
- Test procedure: get a pristine 2.0.0 source, compile with default options (make config, press Enter repeatedly). The reported time should be the time spent on compilation i.e. after you type make zImage, **not** including make dep, make clean. Note that the default target architecture for the kernel is the i386, so if compiled on another architecture, gcc too should be set to cross−compile, with i386 as the target architecture.
- **Results:** compilation time in minutes and seconds (please don't report fractions of seconds).

Whetstone:

- **What:** measures pure floating point performance with a short, tight loop. The source (in C) is quite readable and it is very easy to see which floating−point operations are involved.
- Shortest test in the LBT :−).
- It's an "Old Classic" test: comparable figures are available, its flaws and shortcomings are well known.
- Test procedure: the newest C source should be obtained from Aburto's site. Compile and run in double precision mode. Specify gcc and −O2 as precompiler and precompiler options, and define POSIX 1 to specify machine type.
- **Results:** a floating−point performance figure in MWIPS.

Xbench−0.2:

- **What:** measures X server performance.
- The xStones measure provided by xbench is a weighted average of several tests indexed to an old Sun station with a single−bit−depth display. Hmmm... it is questionable as a test of modern X

servers, but it's still the best tool I have found.

- Test procedure: compile with −O2. We specify a few options for a shorter run: `./xbench -timegoal 3 > results/name_of_your_linux_box.out`. To get the xStones rating, we must run an awk script; the simplest way is to type `make summary.ms`. Check the summary.ms file: the xStone rating for your system is in the last column of the line with your machine name specified during the test.
- **Results:** an X performance figure in xStones.
- Note: this test, as it stands, is outdated. It should be re−coded.

UnixBench version 4.01:

- **What:** measures overall Unix performance. This test will exercice the file I/O and kernel multitasking performance.
- I have discarded all arithmetic test results, keeping only the system−related test results.
- Test procedure: make with −O2. Execute with `./Run -1` (run each test once). You will find the results in the ./results/report file. Calculate the geometric mean of the EXECL THROUGHPUT, FILECOPY 1, 2, 3, PIPE THROUGHPUT, PIPE−BASED CONTEXT SWITCHING, PROCESS CREATION, SHELL SCRIPTS and SYSTEM CALL OVERHEAD indexes.
- **Results:** a system index.

BYTE Magazine's BYTEmark benchmarks:

- **What:** provides a good measure of CPU performance. Here is an excerpt from the documentation: *"These benchmarks are meant to expose the theoretical upper limit of the CPU, FPU, and memory architecture of a system. They cannot measure video, disk, or network throughput (those are the domains of a different set of benchmarks). You should, therefore, use the results of these tests as part, not all, of any evaluation of a system."*
- I have discarded the FPU test results since the Whetstone test is just as representative of FPU performance.
- I have split the integer tests in two groups: those more representative of memory−cache−CPU performance and the CPU integer tests.
- Test procedure: make with −O2. Run the test with `./nbench > myresults.dat` or similar. Then, from myresults.dat, calculate geometric mean of STRING SORT, ASSIGNMENT and BITFIELD test indexes; this is the **memory index**; calculate the geometric mean of NUMERIC SORT, IDEA, HUFFMAN and FP EMULATION test indexes; this is the **integer index**.
- **Results:** a memory index and aninteger index calculated as explained above.

3.5 Possible improvements

The ideal benchmark suite would run in a few minutes, with synthetic benchmarks testing every subsystem separately and applications benchmarks providing results for different applications. It would also automatically generate a complete report and eventually email the report to a central database on the Web.

We are not really interested in portability here, but it should at least run on all recent (> 2.0.0) versions and flavours (i386, Alpha, Sparc...) of Linux.

If anybody has any idea about benchmarking network performance in a simple, easy and reliable way, with a short (less than 30 minutes to setup and run) test, please contact me.

3.6 LBT Report Form

Besides the tests, the benchmarking procedure would not be complete without a form describing the setup, so here it is (following the guidelines from comp.benchmarks.faq):

```
LINUX BENCHMARKING TOOLKIT REPORT FORM
```

```
CPU
==
Vendor:
Model:
Core clock:
Motherboard vendor:
Mbd. model:
Mbd. chipset:
Bus type:
Bus clock:
Cache total:
Cache type/speed:
SMP (number of processors):
```

```
RAM
====
Total:
Type:
Speed:
```

```
Disk
====
Vendor:
Model:
Size:
Interface:
Driver/Settings:
```

```
Video board
===========
Vendor:
Model:
Bus:
Video RAM type:
Video RAM total:
X server vendor:
X server version:
X server chipset choice:
Resolution/vert. refresh rate:
Color depth:
```

```
Kernel
=====
Version:
Swap size:
```

```
gcc
===
Version:
Options:
libc version:
```

```
Test notes
==========
```

```
RESULTS
========
Linux kernel 2.0.0 Compilation Time: (minutes and seconds)
Whetstones: results are in MWIPS.
Xbench: results are in xstones.
Unixbench Benchmarks 4.01 system INDEX:
BYTEmark integer INDEX:
BYTEmark memory INDEX:
```

```
Comments*
=========
* This field is included for possible interpretations of the results, and as
such, it is optional. It could be the most significant part of your report,
though, specially if you are doing comparative benchmarking.
```

3.7 Network performance tests

Testing network performance is a challenging task since it involves at least two machines, a server and a client machine, hence twice the time to setup and many more variables to control, etc... On an ethernet network, I guess your best bet would be the ttcp package. (to be expanded)

3.8 SMP tests

SMP tests are another challenge, and any benchmark specifically designed for SMP testing will have a hard time proving itself valid in real−life settings, since algorithms that can take advantage of SMP are hard to come by. It seems later versions of the Linux kernel (> 2.1.30 or around that) will do "fine−grained" multiprocessing, but I have no more information than that for the moment.

According to David Niemi, *" ... shell8* [part of the Unixbench 4.01 benchmaks]*does a good job at comparing similar hardware/OS in SMP and UP modes."*

4.Example run and results

The LBT was run on my home machine, a Pentium−class Linux box that I put together myself and that I used to write this HOWTO. Here is the LBT Report Form for this system:

```
LINUX BENCHMARKING TOOLKIT REPORT FORM

CPU

==

Vendor: Cyrix/IBM

Model: 6x86L P166+

Core clock: 133 MHz

Motherboard vendor: Elite Computer Systems (ECS)

Mbd. model: P5VX−Be

Mbd. chipset: Intel VX

Bus type: PCI

Bus clock: 33 MHz

Cache total: 256 KB

Cache type/speed: Pipeline burst 6 ns

SMP (number of processors): 1

RAM

====

Total: 32 MB

Type: EDO SIMMs

Speed: 60 ns

Disk

====

Vendor: IBM

Model: IBM−DAQA−33240

Size: 3.2 GB

Interface: EIDE
```

```
Driver/Settings: Bus Master DMA mode 2

Video board

===========

Vendor: Generic S3

Model: Trio64-V2

Bus: PCI

Video RAM type: EDO DRAM

Video RAM total: 2 MB

X server vendor: XFree86

X server version: 3.3

X server chipset choice: S3 accelerated

Resolution/vert. refresh rate: 1152x864 @ 70 Hz

Color depth: 16 bits

Kernel

=====

Version: 2.0.29

Swap size: 64 MB

gcc

===

Version: 2.7.2.1

Options: -O2

libc version: 5.4.23

Test notes

==========

Very light load. The above tests were run with some of the special
Cyrix/IBM 6x86 features enabled with the setx86 program: fast ADS,
fast IORT, Enable DTE, fast LOOP, fast Lin. VidMem.

RESULTS

========

Linux kernel 2.0.0 Compilation Time: 7m12s

Whetstones: 38.169 MWIPS.

Xbench: 97243 xStones.
```

```
BYTE Unix Benchmarks 4.01 system INDEX: 58.43

BYTEmark integer INDEX: 1.50

BYTEmark memory INDEX: 2.50

Comments

=========

This is a very stable system with homogeneous performance, ideal
for home use and/or Linux development. I will report results
with a 6x86MX processor as soon as I can get my hands on one!
```

5.Pitfalls and caveats of benchmarking

After putting together this HOWTO I began to understand why the words "pitfalls" and "caveats" are so often associated with benchmarking...

5.1 Comparing apples and oranges

Or should I say Apples and PCs ? This is so obvious and such an old dispute that I won't go into any details. I doubt the time it takes to load Word on a Mac compared to an average Pentium is a real measure of anything. Likewise booting Linux and Windows NT, etc... Try as much as possible to compare identical machines with a single modification.

5.2 Incomplete information

A single example will illustrate this very common mistake. One often reads in comp.os.linux.hardware the following or similar statement: "I just plugged in processor XYZ running at nnn MHz and now compiling the linux kernel only takes i minutes" (adjust XYZ, nnn and i as required). This is irritating, because no other information is given, i.e. we don't even know the amount of RAM, size of swap, other tasks running simultaneously, kernel version, modules selected, hard disk type, gcc version, etc... I recommend you use the LBT Report Form, which at least provides a standard information framework.

5.3 Proprietary hardware/software

A well−known processor manufacturer once published results of benchmarks produced by a special, customized version of gcc. Ethical considerations apart, those results were meaningless, since 100% of the Linux community would go on using the standard version of gcc. The same goes for proprietary hardware. Benchmarking is much more useful when it deals with off−the−shelf hardware and free (in the GNU/GPL sense) software.

5.4 Relevance

We are talking Linux, right ? So we should forget about benchmarks produced on other operating systems (this is a special case of the "Comparing apples and oranges" pitfall above). Also, if one is going to benchmark Web server performance, **do not** quote FPU performance and other irrelevant information. In such cases, less is more. Also, you do **not** need to mention the age of your cat, your mood while benchmarking, etc..

6.FAQ

Q1.

Is there any single figure of merit for Linux systems ?

A:

No, thankfully nobody has yet come up with a Lhinuxstone (tm) measurement. And if there was one, it would not make much sense: Linux systems are used for many different tasks, from heavily loaded Web servers to graphics workstations for individual use. No single figure of merit can describe the performance of a Linux system under such different situations.

Q2.

Then, how about a dozen figures summarizing the performance of diverse Linux systems ?

A:

That would be the ideal situation. I would like to see that come true. Anybody volunteers for a **Linux Benchmarking Project** ? With a Web site and an on−line, complete, well−designed reports database ?

Q3.

... BogoMips ... ?

A:

BogoMips has nothing to do with the performance of your system. Check the BogoMips Mini−HOWTO.

Q4.

What is the "best" benchmark for Linux ?

A:

It all depends on which performance aspect of a Linux system one wants to measure. There are different benchmarks to measure the network (Ethernet sustained transfer rates), file server (NFS), disk I/O, FPU, integer, graphics, 3D, processor−memory bandwidth, CAD performance, transaction time, SQL performance, Web server performance, real−time performance, CD−ROM performance, Quake performance (!), etc ... AFAIK no bechmark suite exists for Linux that supports all these tests.

Q5.

What is the fastest processor under Linux ?

A:

Fastest at what task ? If one is heavily number−crunching oriented, a very high clock rate Alpha (600 MHz and going) should be faster than anything else, since Alphas have been designed for that kind of performance. If, on the other hand, one wants to put together a very fast news server, it is probable that the choice of a fast hard disk subsystem and lots of RAM will result in higher performance improvements than a change of processor, for the same amount of $.

Q6.

Let me rephrase the last question, then: is there a processor that is fastest for general purpose applications ?

A:

This is a tricky question but it takes a very simple answer: **NO**. One can always design a faster system even for general purpose applications, independent of the processor. Usually, all other things being equal, higher clock rates will result in higher performance systems (and more headaches too). Taking out an old 100 MHz Pentium from an (usually not) upgradable motherboard, and plugging in the 200 MHz version, one should feel the extra "hummph". Of course, with only 16 MBytes of RAM, the same investment would have been more wisely spent on extra SIMMs...

Q7.

So clock rates influence the performance of a system ?

A:

For most tasks except for NOP empty loops (BTW these get removed by modern optimizing compilers), an increase in clock rate will not give you a linear increase in performance. Very small processor intensive programs that will fit entirely in the primary cache inside the processor (the L1 cache, usually 8 or 16 K) will have a performance increase equivalent to the clock rate increase, but most "true" programs are much larger than that, have loops that do not fit in the L1 cache, share the L2 (external) cache with other processes, depend on external components and will give much smaller performance increases. This is because the L1 cache runs at the same clock rate as the processor, whereas most L2 caches and all other subsystems (DRAM, for example) will run asynchronously at lower clock rates.

Q8.

OK, then, one last question on that matter: which is the processor with the best price/performance ratio for general purpose Linux use ?

A:

Defining "general purpose Linux use" in not an easy thing ! For any particular application, there is always a processor with THE BEST price/performance ratio at any given time, but it changes rather frequently as manufacturers release new processors, so answering Processor XYZ running at n MHz would be a snapshot answer. However, the price of the processor is insignificant when compared to the price of the whole system one will be putting together. So, really, the question should be how can one maximize the price/performance ratio for a given system ? And the answer to that question depends heavily on the minimum performance requirements and/or maximum cost established for the configuration being considered. Sometimes, off−the−shelf hardware will not meet minimum performance requirements and expensive RISC systems will be the only alternative. For home use, I recommend a balanced, homogeneous system for overall performance (now go figure what I mean by balanced and homogeneous :−); the choice of a processor is an important decision , but no more than choosing hard disk type and capacity, amount of RAM, video card, etc...

Q9.

What is a "significant" increase in performance ?

A:

I would say that anything under 1% is not significant (could be described as "marginal"). We, humans, will hardly perceive the difference between two systems with a 5 % difference in response time. Of course some hard−core benchmarkers are not humans and will tell you that, when comparing systems with 65.9 and 66.5 performance indexes, the later is "definitely faster".

Q10.

How do I obtain "significant" increases in performance at the lowest cost ?

A:

Since most source code is available for Linux, careful examination and algorithmic redesign of key subroutines could yield order−of−magnitude increases in performance in some cases. If one is dealing with a commercial project and does not wish to delve deeply in C source code a **Linux consultant should be called in**. See the Consultants−HOWTO.

7.Copyright, acknowledgments and miscellaneous

7.1 How this document was produced

The first step was reading section 4 "Writing and submitting a HOWTO" of the HOWTO Index by Tim Bynum.

I knew absolutely nothing about SGML or LaTeX, but was tempted to use an automated documentation generation package after reading the various comments about SGML−Tools. However, inserting tags manually in a document reminds me of the days I hand−assembled a 512 byte monitor program for a now defunct 8−bit microprocessor, so I got hold of the LyX sources, compiled it, and used its LinuxDoc mode. Highly recommended combination: **LyX and SGML−Tools**.

7.2 Copyright

7.3 New versions of this document

New versions of the Linux Benchmarking−HOWTO will be placed on sunsite.unc.edu and mirror sites. There are other formats, such as a Postscript and dvi version in the other−formats directory. The Linux Benchmarking−HOWTO is also available for WWW clients such as Grail, a Web browser written in Python. It will also be posted regularly to comp.os.linux.answers.

7.4 Feedback

Suggestions, corrections, additions wanted. Contributors wanted and acknowledged. Flames not wanted.

I can always be reached at andrewbalsa@usa.net.

7.5 Acknowledgments

David Niemi, the author of the Unixbench suite, has proved to be an endless source of information and (valid) criticism.

I also want to thank Greg Hankins one of the main contributors to the SGML−tools package, Linus Torvalds and the entire Linux community. This HOWTO is my way of giving back.

7.6 Disclaimer

Your mileage may, and will, vary. Be aware that benchmarking is a touchy subject and a great time−and−energy consuming activity.

7.7 Trademarks

Pentium and Windows NT are trademarks of Intel and Microsoft Corporations respectively.

BYTE and BYTEmark are trademarks of McGraw−Hill, Inc.

Cyrix and 6x86 are trademarks of Cyrix Corporation.

Linux is not a trademark, hopefully never will be.
