A Survey of Hardware Performance Analysis Tools

Scott Helvick, shelvick@wustl.edu (A project report written under the guidance of Prof. Raj Jain)

Download

Abstract

Among computer enthusiasts and professionals alike, few performance measures are as interesting as those of a system's hardware. This paper will list and discuss the pros, cons, and intended usage of several hardware performance analysis tools: nmon, iostat, collectl, bonnie++, dbench, nbench, and hardinfo. It will also emphasize the difference between quick-hit, synthetic, and application benchmarks, while discussing the quick-hit and synthetic benchmarks and their uses in measuring hardware performance.

Keywords: hardware, performance, benchmarks, measurement, quick-hit, synthetic, application, Linux

Table of Contents

1 Introduction

Among computer enthusiasts and professionals alike, few performance measures are as interesting as those of a system's hardware. Regardless of its intended use, the first thing a power user will do when he is done building a system is to test its hardware performance. The expansive variety of hardware performance analysis tools created by the open source community is proof of this.

This paper will list and discuss the pros, cons, and intended usage of several such tools. It is important to remember that performance tools are run on an operating system (GNU/Linux, in the case of those described in this paper) and may be affected by other processes running on a given system. Thus, there will always be a margin of error in any measurement. The tools in this paper have been chosen with the goal of minimizing this overhead, so it is hoped that their measurements will maintain a high a degree of accuracy.

Not all system components are created equally, and every component has a different impact on the system as a whole, an impact which changes with every workload. For example, a system used only for word processing and web browsing may benefit most from a simple upgrade to system memory. On the other hand, the bottleneck in a high-powered gaming PC is usually the graphics card [Pegoraro04]. Surprisingly enough, not all metrics for *measuring* system performance are created equally, either. The Linux kernel displays, on boot-up, a metric called BogoMips. BogoMips is a measurement of how fast a certain type of busy loop, calibrated to a machine's processor speed, runs on that machine. Quite literally, it measures "the number of million times per second a processor can do absolutely nothing." Incidentally, "Bogo" comes from the word "bogus," a way of mocking how the calculation is unscientific [Dorst06].

---

2 Tool Overview

Benchmarking software tools may be classified under three categories -- **Quick-hit**, **Synthetic**, and **Application** benchmarks. **Quick-hit** benchmarks are simple tests to take a particular measurement or get a reading of a specific aspect of performance. They are not meant to give a holistic perspective of system performance, but may be useful in the cases where only one component needs to be analyzed. In some cases, quick-hit benchmarks can also be useful for identifying damaged hardware. **Synthetic** benchmarks are usually more extensive tests meant to put a system or a single performance aspect under heavy load. Synthetic benchmarks are useful for measuring the maximum capacity or throughput for a given component. However, they do not represent a "real-world" workload -- that's why **Application** benchmarks exist. Application benchmarks are intended to test systems with loads similar to what they would experience in a "production" environment [Wright02]. Because application benchmarks attempt to simulate a real-world workload, their performance is often influenced more by the operating system than the performance of synthetic or quick-hit benchmarks. For this reason, the author has determined that they are not as relevant to hardware performance, thus no application benchmarks are discussed in this paper.

2.1 Quick-Hit Benchmarks

2.1.1 nmon [nmon]

Nmon, short for Nigel's Monitor, is a multi-faceted monitoring tool. Hosted by IBM, nmon was written for AIX, but the author had no trouble running it on GNU/Linux. However, the tool is provided only as a binary file and has not been open sourced; thus, anyone wishing to compile it for themselves or run it on an incompatible operating system may be out of luck. Nmon captures a wide variety of performance data -- network I/O rates, disk I/O rates, memory usage, and others. One of nmon's defining features is its support for exporting, analysing, and graphing its data output. Running nmon with the -f or -F switch will save its output as a .csv file. IBM provides other nmon tools, such as an Excel Analyser, which will make use of this file [Griffiths08]. Figure 2-1 depicts nmon reporting CPU, memory, network, and disk measurements. Because the system is mostly idle, this particular screenshot is only an example of nmon output and does not provide useful data. Nmon may also display information about system build and processors, the system kernel, filesystems, processes, and Network File System shares.

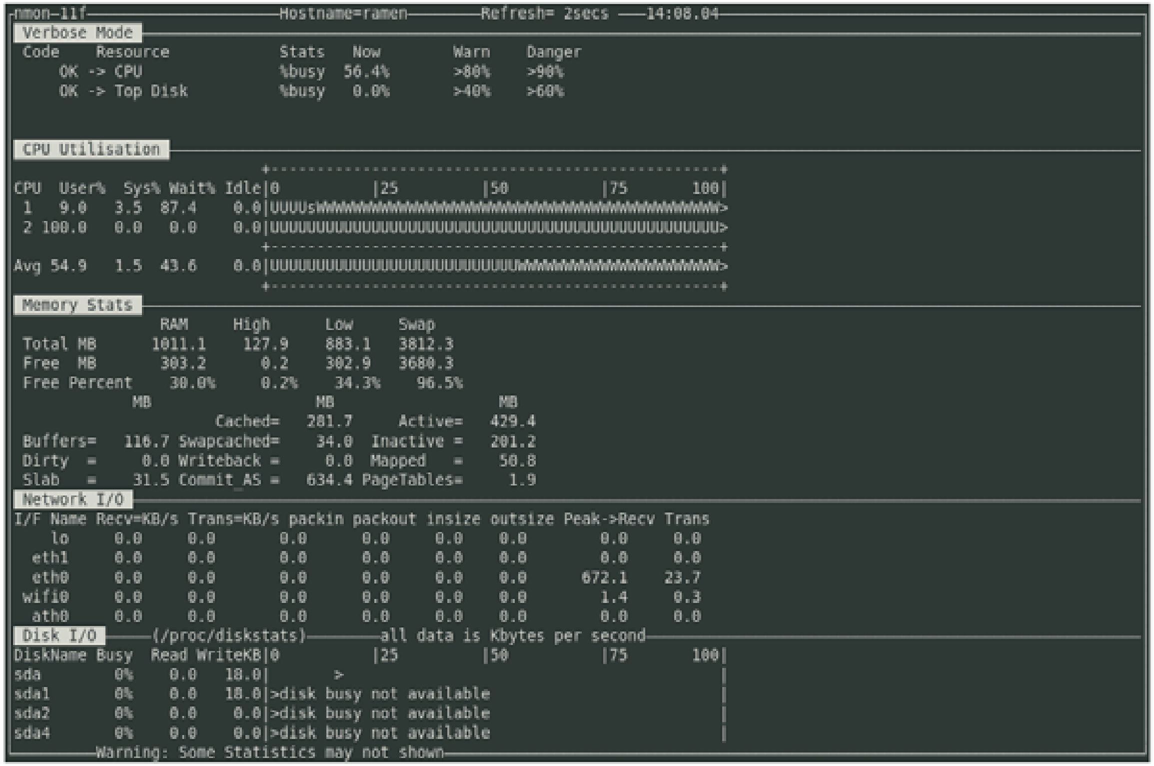

Figure 2-1: nmon reporting several measurements of CPU, memory, network, and disk performance

2.1.2 iostat [iostat]

Iostat is a tool used to monitor system I/O by reading files in the /proc filesystem and comparing the time the devices are active to their average transfer rates. It is available as part of the sysstat package, which also includes sar and mpstat. Iostat may generate reports detailing statistics about CPU utilization, device utilization, and/or network filesystems. One of iostat's differentiating features is that it measures both instantaneous, one-time performance as well as performance over time. Figure 2-2 depicts the CPU and device utilization reports. In this example, the CPU is mostly idle with no outstanding disk I/O requests. Also listed in the CPU report are percentages for system- and user-level executions (with and without nice prioritization), I/O waits and waits due to the hypervisor servicing other virtual processors. The device utilization report lists the following fields (in this order): read requests merged per second, write requests merged per second, actual reads per second, actual writes per second, megabytes read per second, megabytes written per second, average request size (in sectors), average queue length, average wait time per request, average service time per request, and percentage of CPU time utilized by I/O [Godard08]. In summary, the system analyzed in Figure 2-2 has a history of long idle times followed by large numbers of write requests (as shown by the high number of queued writes/second compared to queued reads/second).

```
scott@ramen:~$ iostat -m -x
Linux 2.6.24-21-generic (ramen)         11/09/2008

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           6.27   10.89    1.14    0.08    0.00   81.62

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.58    19.61    0.56    0.75     0.03     0.08   175.44     0.14  106.80   2.20   0.29

scott@ramen:~$
```

Figure 2-2: iostat displays extended statistics for the CPU and disk device sda

2.1.3 collectl [collectl]

Collectl is a versatile "do-it-all" performance monitoring tool. It includes options to run interactively or as a daemon, options to format its output in various ways, and options which ensure the user sees only the data he wants to see, at the rate at which he wants to see it. Figure 2-3 shows a default run of collectl; with no options specified, it displays terse statistics about CPU, disk, and network performance. (Collectl's complete domain includes CPU interrupts, NFS shares, inodes, the Lustre file system, memory, sockets, TCP, and Infiniband statistics.) Each line in the example represents one second of sampling. The CPU measurements are, in order: CPU utilization, time executing in system mode, interrupts per second, and context switches per second. The disk and network sections display kilobytes read and written (and total reads/writes), and kilobytes in/out (and packets in/out), respectively. The example system is mostly idle, with the exception of a single-threaded process, in user space, using one CPU. (This particular system has two processor cores, so it may have allowed collectl alone to monopolize an entire core.)

```
scott@ramen:~/Desktop/collectl-3.1.1$ collectl
waiting for 1 second sample...
#<----CPU[HYPER]-----><----------Disks-----------><----------Network---------->
#cpu sys inter  ctxsw KBRead  Reads KBWrit Writes   KBIn  PktIn  KBOut  PktOut
 100   1   370    212      0      0      0      0      0      0      0       0
 100   1   392    323      0      0      0      0      0      0      0       0
 100   0   379    197      0      0      0      0      0      0      0       0
 100   1   412    321      0      0      0      0      0      0      0       0
 100   0   393    192      0      0      0      0      0      0      0       0
 100   0   417    423      0      0      0      0      0      0      0       0
 100   1   443    788      0      0      0      0      0      1      0       0
  99   3   447    756     40      1    212     25      0      1      0       2
 100   1   378    136      0      0      0      0      0      0      0       0
 100   2   472    428      0      0      0      0      0      0      0       0
 100   0   494    457      0      0      0      0      0      0      0       0
 100   3   488    659      0      0      0      0      0      0      0       0
 100   0   471    463      0      0      0      0      0      0      0       0
 100  11   482    489      0      0    120      6      0      0      0       0
 100   6   428   1316      0      0      0      0      0      0      0       0
 100   4   491    849      0      0      0      0      0      0      0       0
 100   4   406    479      0      0      0      0      0      0      0       0
 100   2   473    563      0      0      0      0      0      0      0       0
  99   1   451    523      0      0      0      0      0      0      0       0
```

Figure 2-3: collectl sampling measurements of CPU, disk, and network performance

2.2 Synthetic Benchmarks

2.2.1 bonnie++ [bonnie++]

Bonnie++ is a benchmark written in C++ for the purpose of testing hard drive and filesystem performance. Its predecessor, bonnie, was written in C and included a series of I/O tests meant to simulate various types of database applications. Bonnie++ tests in two sequences. The first is bonnie's original series of database I/O operations, while the second sequence tests the reading and writing of many small files. Twelve tests are performed in total, including three types of sequential output, two types of sequential input, and random seeks. Sequential access is simply reading/writing disk blocks in sequential order. In practice, most disk accesses are *not* sequential, the exceptions being large files or formatting operations. However, testing sequential access can be a great synthetic benchmark, because the disk head moves very little, resulting in high transfer speeds. Random access, of course, involves reading/writing in random locations on the disk. This is slower than sequential access, since the disk head is required to move rapidly, although it is closer to a real-world simulation [LinuxInsight07]. The results of running bonnie++ with no parameters are shown in Figure 2-4. Two gigabytes were written (in three different ways) and the speeds and CPU utilizations were measured. Then they were read, sequentially and randomly, again measuring the speed (in kilobytes and random seeks per second) and CPU utilization. Finally, 16*1024 files were created and deleted, randomly and sequentially. The +'s signify a test that could not be accurately measured because it ran in less than 500 ms [Coker01].

```
scott@ramen:~$ bonnie++
Writing with putc()...done
Writing intelligently...done
Rewriting...done
Reading with getc()...done
Reading intelligently...done
start 'em...done...done...done...
Create files in sequential order...done.
Stat files in sequential order...done.
Delete files in sequential order...done.
Create files in random order...done.
Stat files in random order...done.
Delete files in random order...done.
Version 1.03b       ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine        Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
ramen            2G 35765  98 58607  30 30991  10 43916  95 73800   8 235.1   1
                    ------Sequential Create------ --------Random Create--------
                    -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
              files  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
                 16 25139  85 +++++ +++ +++++ +++ 25661  80 +++++ +++ 27251  73
ramen,2G,35765,98,58607,30,30991,10,43916,95,73800,8,235.1,1,16,25139,85,+++++,+++,+++++,+++,25661,80,+++++,+++,27251,73
scott@ramen:~$
```

Figure 2-4: bonnie++ displaying results of its sequential and random reads, writes, and file creation benchmarks

2.2.2 dbench [dbench]

Dbench is a synthetic benchmark which attempts to measure disk throughput by simulating a run of Netbench, the industry-standard benchmark for Windows file servers. To do this, dbench parses a text file containing a network sniffer dump of an actual Netbench run. In this way, dbench "fakes" a Netbench session and produces a load of about 90,000 operations. Figure 2-5 shows the tail end of a ten-minute dbench run. In this particular run, four client processes were simulated; the total mean throughput was 230.814 MB/sec. The figure does not show a complete run, but dbench actually goes through three phases -- warmup (a lighter load which allows disk throughput to slowly increase), execute (the most strenuous part of the benchmark), and cleanup (when any created files are deleted). Though a comprehensive benchmark, dbench is limited in its versatility -- only seven options may be passed to it via the command line, and two of those are specific to tbench, which is a client-server version packaged with dbench [Tridgell02].

```
4    7290088    230.56 MB/sec  execute 566 sec
4    7295476    230.34 MB/sec  execute 567 sec
4    7306741    230.37 MB/sec  execute 568 sec
4    7317914    230.39 MB/sec  execute 569 sec
4    7329096    230.41 MB/sec  execute 570 sec
4    7338694    230.40 MB/sec  execute 571 sec
4    7348681    230.41 MB/sec  execute 572 sec
4    7359578    230.41 MB/sec  execute 573 sec
4    7370322    230.42 MB/sec  execute 574 sec
4    7380976    230.44 MB/sec  execute 575 sec
4    7391212    230.42 MB/sec  execute 576 sec
4    7401828    230.42 MB/sec  execute 577 sec
4    7413175    230.45 MB/sec  execute 578 sec
4    7424510    230.46 MB/sec  execute 579 sec
4    7435766    230.48 MB/sec  execute 580 sec
4    7446162    230.49 MB/sec  execute 581 sec
4    7457466    230.50 MB/sec  execute 582 sec
4    7468841    230.53 MB/sec  execute 583 sec
4    7480027    230.55 MB/sec  execute 584 sec
4    7491245    230.56 MB/sec  execute 585 sec
4    7501825    230.56 MB/sec  execute 586 sec
4    7513102    230.58 MB/sec  execute 587 sec
4    7524193    230.59 MB/sec  execute 588 sec
4    7535521    230.61 MB/sec  execute 589 sec
4    7546654    230.63 MB/sec  execute 590 sec
4    7557656    230.64 MB/sec  execute 591 sec
4    7568941    230.67 MB/sec  execute 592 sec
4    7580313    230.69 MB/sec  execute 593 sec
4    7591499    230.71 MB/sec  execute 594 sec
4    7602705    230.73 MB/sec  execute 595 sec
4    7613587    230.74 MB/sec  execute 596 sec
4    7624872    230.77 MB/sec  execute 597 sec
4    7636090    230.78 MB/sec  execute 598 sec
4    7647497    230.79 MB/sec  execute 599 sec
4    7658615    230.81 MB/sec  cleanup 600 sec
4    7658615    230.78 MB/sec  cleanup 600 sec

Throughput 230.814 MB/sec 4 procs
scott@ramen:~$
```

Figure 2-5: Tail end of dbench output, showing the cleanup phase and mean throughput

2.2.3 nbench [nbench]

Nbench is based off of BYTE Magazine's BYTEmark benchmark program; the original BYTE benchmarks were modified to work better on 64-bit machines. Nbench is a synthetic benchmark intended to test a system's CPU, FPU, and memory system. Nbench runs ten single-threaded tests, including integer and string sorting, Fourier coefficients, and Huffman compression. A number of options are available, but their accessibility is limted by the requirement of a command file, decreasing the tool's usability. Something especially interesting, and perhaps unique, about nbench is that it statistically analyzes its own results for confidence and increases the number of runs if necessary. Practically, this means that the benchmarks may be run even on a heavily-loaded system (whether or not that's a good idea) and still produce accurate results -- the greater variance just means it will take longer to get there. Figure 2-6 illustrates a default run of nbench; the unit of measure is iterations/second, so these metrics are HB. The measurements of the system under test are compared to those of a Pentium 90 and an AMD K6/233 [Mayer03]. The example system, a dual-core Pentium 4 (2.8 GHz), trounces the baseline systems except, strangely, in the Assignment and Neural Net benchmarks. (Further investigation is beyond the scope of this paper, but this author speculates that the Pentium 4 may have a design flaw which inhibits its performance on specific tasks.) The index scores at the end denote, on average, how many times faster the target system ran the benchmarks compared to the baseline systems. In this example, the P4 particularly excelled at the floating-point benchmarks -- Fourier, Neural Net, and LU Decomposition.

```
scott@ramen:~/Desktop/nbench-byte-2.2.3$ ./nbench

BYTEmark* Native Mode Benchmark ver. 2 (10/95)
Index-split by Andrew D. Balsa (11/97)
Linux/Unix* port by Uwe F. Mayer (12/96,11/97)

TEST                : Iterations/sec.  : Old Index   : New Index
                    :                  : Pentium 90* : AMD K6/233*
--------------------:------------------:-------------:------------
NUMERIC SORT        :          422.76  :      10.84  :       3.56
STRING SORT         :          45.048  :      20.13  :       3.12
BITFIELD            :      2.1451e+08  :      36.80  :       7.69
FP EMULATION        :          98.441  :      47.24  :      10.90
FOURIER             :           13407  :      15.25  :       8.56
ASSIGNMENT          :          15.846  :      60.30  :      15.64
IDEA                :          1792.1  :      27.41  :       8.14
HUFFMAN             :            1090  :      30.23  :       9.65
NEURAL NET          :          13.767  :      22.12  :       9.30
LU DECOMPOSITION    :          632.04  :      32.74  :      23.64
==========================ORIGINAL BYTEMARK RESULTS==========================
INTEGER INDEX       : 29.392
FLOATING-POINT INDEX: 22.267
Baseline (MSDOS*)   : Pentium* 90, 256 KB L2-cache, Watcom* compiler 10.0
==============================LINUX DATA BELOW===============================
CPU                 : Dual GenuineIntel Intel(R) Pentium(R) 4 CPU 2.80GHz 2800MHz
L2 Cache            : 512 KB
OS                  : Linux 2.6.24-21-generic
C compiler          : gcc version 4.2.4 (Ubuntu 4.2.4-1ubuntu3)
libc                : libc-2.7.so
MEMORY INDEX        : 7.208
INTEGER INDEX       : 7.431
FLOATING-POINT INDEX: 12.350
Baseline (LINUX)    : AMD K6/233*, 512 KB L2-cache, gcc 2.7.2.3, libc-5.4.38
* Trademarks are property of their respective holder.
scott@ramen:~/Desktop/nbench-byte-2.2.3$
```

Figure 2-6: Summarized nbench output and scores compared to baseline

2.2.4 hardinfo [hardinfo]

Hardinfo, a rare GUI-only performance analysis tool, is both a quick-hit and synthetic benchmark. Figure 2-7 shows a report generated by hardinfo after running its six benchmark routines; the Zlib, MD5 and SHA1 CPU tests are HB metrics, while the CPU's Fibonacci and Blowfish computations as well as the FPU Raytracing measurement are LB metrics. With the exception of the SHA1 benchmark, the example machine's performance is on par with that of the Celeron processor. The hardinfo GUI itself also displays a host of information about a system's hardware specifications; it does this by parsing several files in the /proc directory [Pereira03]. Hardinfo is packaged with the Ubuntu Linux distribution and commonly included with the GNOME desktop. Unfortunately, hardinfo *only* contains a GUI interface and its output may not be directed to the command line.

CPU ZLib

CPU ZLib	
This Machine	8323.279
PowerPC 740/750 (280.00MHz)	2150.597408
Intel(R) Celeron(R) M processor 1.50GHz	8761.604561

CPU Fibonacci

CPU Fibonacci	
This Machine	7.436
Intel(R) Celeron(R) M processor 1.50GHz	8.1375674
PowerPC 740/750 (280.00MHz)	58.07682

CPU MD5

CPU MD5	
This Machine	37.570
PowerPC 740/750 (280.00MHz)	7.115258
Intel(R) Celeron(R) M processor 1.50GHz	38.6607998

CPU SHA1

CPU SHA1	
This Machine	33.163
PowerPC 740/750 (280.00MHz)	6.761451
Intel(R) Celeron(R) M processor 1.50GHz	49.6752776

CPU Blowfish

CPU Blowfish	
This Machine	26.090
Intel(R) Celeron(R) M processor 1.50GHz	26.1876862
PowerPC 740/750 (280.00MHz)	172.816713

FPU Raytracing

FPU Raytracing	
This Machine	32.236
Intel(R) Celeron(R) M processor 1.50GHz	40.8816714
PowerPC 740/750 (280.00MHz)	161.312647

Figure 2-7: A report generated after running the hardinfo benchmarks

3 Summary

With as quickly as the technology sector is growing, performance analysis -- particularly of hardware -- is sure to continue to be popular among computer enthusiasts and professionals alike. It is this author's hope that some of the open source tools developed in the late 1990's will be updated and enhanced to perform well with upcoming system architectures.

This paper has listed and discussed several hardware performance analysis tools; in particular, synthetic and quick-hit benchmarks created by the open source community. Figure 3-1 summarizes the discussion of these utilities: nmon, iostat, collectl, bonnie++, dbench, nbench, and hardinfo. While no tool running in software can perfectly measure the performance of hardware, the tools in this paper have been chosen to minimize this problem. In conclusion, this survey of hardware performance analysis tools is significant, but far from comprehensive.

Tool Summary				
Name	**Type**	**Uses**	**Pros**	**Cons**
nmon	Quick-hit	Monitor CPU/memory/disk/network interactively in real-time	Interactive, easy-to-use, versatile	Closed-source, binaries only
iostat	Quick-hit	Monitor instantaneous system I/O, compare to historical I/O	Powerful and versatile, monitors historical and real-time performance	Generally only available as part of the sysstat package
collectl	Quick-hit	Collect large numbers of system performance stats for processing by another application	Offers an extremely wide variety of measurements	Large number of options may scare away novice users
bonnie++	Synthetic	Test hard drive and filesystem performance with simulated real-life benchmarks	Runs a wide array of tests, fairly realistic	Not all tests are useful on an extremely fast machine or with very limited disk space
dbench	Synthetic	Measure disk throughput, simulate Netbench on Linux	Accurate and powerful, free version of a well-known benchmark	Options are limited
nbench	Synthetic	Measure CPU/FPU/memory performance via several methods	Wide array of well-known benchmarks, highly robust and accurate	Difficult to use, most options require a command file

hardinfo	Synthetic	Quickly gather information about a system and its performance	Easy to use, quick one-click benchmarks	GUI only

Figure 3-1: A summary of the tools discussed, including classification, uses, and pros/cons

4 References

4.1 Articles

1. [Pegoraro04] Rob Pegoraro. "A Processor's Clock Speed Is Just One Measure of Performance". Washington Post. 13 Jun 2004. http://www.washingtonpost.com/wp-dyn/articles/A36196-2004Jun12.html.
2. [Dorst06] Wim van Dorst. "BogoMips mini-Howto". 2 March 2006. http://www.clifton.nl/bogomips.html.
3. [LinuxInsight07] "admin". "How fast is your disk?". LinuxInsight. 16 Jan 2007. http://www.linuxinsight.com/how_fast_is_your_disk.html.
4. [Wright02] John Wright. "Linux Benchmark Suite Homepage". Sourceforge. 15 May 2002. http://lbs.sourceforge.net.

4.2 Tool Documentation

1. [Coker01] Russell Coker. "Bonnie++". 2001. http://www.coker.com.au/bonnie++/readme.html.
2. [Tridgell02] Andrew Tridgell. "Emulating Netbench". Samba. 29 Dec 2002. http://samba.org/ftp/tridge/dbench/README.
3. [Mayer03] Uwe Mayer. "README". nbench README file. 18 Feb 2003. http://www.tux.org/~mayer/linux/nbench-byte-2.2.3.tar.gz.
4. [Griffiths08] Nigel Griffiths. "nmon for AIX & Linux Performance Monitoring". IBM. 24 Oct 2008. http://www.ibm.com/developerworks/wikis/display/WikiPtype/nmon.
5. [Pereira03] Leandro Pereira. "hardinfo(1)". Hardinfo man page. 15 June 2003. http://prdownload.berlios.de/hardinfo/hardinfo-0.4.2.3.tar.bz2.
6. [Godard08] Sebastien Godard. "iostat manual page". 13 Nov 2008. http://pagesperso-orange.fr/sebastien.godard/man_iostat.html.
7. [Seger08] Mark Seger. "collectl - Documentation". Sourceforge. 7 Nov 2008. http://collectl.sourceforge.net/Documentation.html.

4.3 Tool Downloads

1. [bonnie++] Russell Coker. "Bonnie++". 13 Jan 2003. http://www.coker.com.au/bonnie++.
2. [dbench] Andrew Tridgell. "dbench". Samba. 29 Dec 2002. http://samba.org/ftp/tridge/dbench.
3. [nbench] Uwe Mayer. "Linux/Unix nbench". Tux. 12 May 2008. http://www.tux.org/~mayer/linux/bmark.html.
4. [nmon] Nigel Griffiths. "nmon for AIX & Linux Performance Monitoring". IBM. 24 Oct 2008. http://www.ibm.com/developerworks/wikis/display/WikiPtype/nmon.
5. [hardinfo] Leandro Pereira. "Hardinfo: Download". 4 Nov 2007. http://wiki.hardinfo.org/Downloads.
6. [iostat] Sebastien Godard. "SYSSTAT". 13 Nov 2008. http://pagesperso-orange.fr/sebastien.godard/download.html.
7. [collectl] Mark Seger. "Collectl". Sourceforge. 7 Nov 2008. http://collectl.sourceforge.net.

5 List of Acronyms

Acronym	Meaning
CPU	Central Processing Unit
FPU	Floating Point Unit
GUI	Graphical User Interface
HB	Higher is Better
I/O	Input/Output

LB	Lower is Better
NFS	Network File System

Last modified on November 24, 2008
This and other papers on latest advances in performance analysis are available on line at http://www.cse.wustl.edu/~jain/cse567-08/index.html
SHARE Back to Raj Jain's Home Page