

# Appendix E Linux System and Performance Utilities

This appendix summarizes common system and performance utilities available on a Linux machine. Linux professionals use these utilities to check their Linux system configurations and monitor/diagnose performance issues on their Linux systems. We already introduced them in the main text, and summarizing them here is just for more convenience.

## E.1 LINUX SYSTEM UTILITIES

Table E.1 summarizes common Linux system utilities. You can use this list of utilities to get a good understanding of the *raw* performance from your Linux system.

**Table E.1 Common Linux system utilities**

| Utility                         | Description   |
|---------------------------------|---|
| <code>cat /etc/*release*</code> | Check vendor release version  |
| <code>uname -r</code>           | Check Linux kernel version  |
| <code>nproc</code>              | Check # of CPUs   |
| <code>less /proc/cpuinfo</code> | CPU specs   |
| <code>lscpu</code>              | Less verbose CPU specs  |
| <code>cat /proc/meminfo</code>  | Detailed memory usage   |
| <code>free -m</code>            | Check memory utilizations   |
| <code>df -m</code>              | Check disk utilizations   |
| <code>ifconfig</code>           | Basic statistics about network interfaces   |
| <code>netstat</code>            | Check port status. Add <code>-ano   grep "&lt;port&gt;"</code> to check a particular port |
| <code>ulimit -a</code>          | Check kernel settings   |

## E.2 Linux Performance Diagnosing/Monitoring Utilities

Table E.2 summarizes the usage of the Linux utilities for diagnosing/monitoring the performance of a particular Linux system. The utilities are mentioned for how to identify:

- Whether the system is lightly or heavily loaded overall
- Whether the system is bottlenecked on CPU or IO
- Which processes are particularly *hot*

With the `vmstat` utility, note the following:

- The first output always displays average values since the last reboot.
- You can add pipe “ `| awk '{now=strftime("%Y-%m-%d %T "); print now $0}'` ” to precede each output line with a timestamp to be used with a graphing tool.

**Table E.2 Common Linux performance utilities**

| Category              | Utilities  |
|-----------------------|--|
| Overall               | Use <code>top</code> to get a quick assessment of whether the system is lightly or heavily loaded. The important columns include <i>pid</i> , <i>RES</i> , <i>R/S</i> , <i>%CPU</i> , <i>%MEM</i> , <i>TIME+</i> , and so on. Keep in mind that by default, <code>top</code> sorts by <i>%CPU</i> , but you can toggle sorting with Shift+M for <i>%MEM</i> , Shift+T for <i>TIME+</i> , and Shift+P for <i>%CPU</i> . Shift+I also allows you to turn Irix mode off so that the <i>%CPU</i> column would display total average rather than cumulative CPU utilizations.   |
| CPU or IO bottlenecks | Run the <code>vmstat -n &lt;interval&gt; &lt;count&gt; -S M</code> command and: <ul style="list-style-type: none"> <li>■ If the <b>r</b> column is high, it means CPU is the bottleneck, as <b>r</b> means the # of processes in the run queue, waiting for a free CPU slot.</li> <li>■ If the <b>b</b> column is high, it means IO (disk or network) is the bottleneck, as <b>b</b> means the # of processes waiting for a resource other than a CPU.</li> </ul>  |
| Hot processes         | Run <code>top -c</code> to identify hot processes that have high <i>%CPU</i> , high <i>%MEM</i> , and high <i>TIME+</i> . Use <i>sort-by</i> as described above to toggle the top processes. To learn how a (hot) process is launched, run the <code>ps -fwwp &lt;pid&gt;</code> command, where <i>&lt;pid&gt;</i> can be found with the <code>top -c</code> command. You can also run the <code>top -d &lt;interval&gt; -b -n &lt;count&gt; -p &lt;pid1&gt;,&lt;pid2&gt;,...</code> command to obtain samples for the identified hot processes. Use the <code>pstree</code> command to find out the ancestor processes of a hot process, all the way to <code>init</code> or <code>systemd</code> . |

## E.3 THE SAR UTILITY

Linux has a very versatile utility called `sar`. This utility requires the `sysstat` package, which may not be installed by default. However, it's easy to install it. For example, on openSUSE, use the below procedure to get `sar` working:

- 1 Run `zypper` in `sysstat` to install `sysstat`.

2 Run `/etc/init.d/boot.sysstat start` to start the `sadc` daemon to enable collecting data automatically. This will add a link to `/etc/cron.d/` that calls `sadc` with the following default configurations:

- All available data will be collected in `/var/log/sa/saDD`, where `DD` stands for the current day. If a file already exists, it will be archived.
- The summary report is written to `/var/log/sa/sarDD`.
- Data is collected every 10 minutes, and a summary report is generated every 6 hours. Of course, these settings are customizable.

The `sar` utility can be run on the fly with the following form:

```
sar <option> <interval> <count>
```

, where `<option>` can be:

- **-u**: CPU
- **-r**: RAM (memory)
- **-B**: paging, with high `majflt/s` (major faults per second) indicating insufficient main memory
- **-d**: disk, with the following specially interesting columns:
  - `avgque-sz`: average queue length
  - `await`: service time + latency in milliseconds
  - `svctm`: service time in milliseconds
  - `%util`: percent utilization
- **-n ALL**: network
- **-q**: run-queue (# of tasks waiting), `plist` (# of tasks in the task list), and load average
- **-w**: `proc/s` and `cswh/s`

In addition, you can add `-s hh:mm:ss` to query past data since the specified start time. At last, don't forget adding the `-p` option for pretty print, for example:

```
sar -d -p 5 10
```

Keep using these utilities all the time!



# Appendix F The Harp Utility for Optimizing UI Performance

This appendix introduces the Harp utility I developed for facilitating optimizing UI performance. This has turned out to be an extremely powerful tool for facilitating optimizing UI performance. I have had many very successful use cases with using this tool for my projects, which motivates me to share this tool with you.

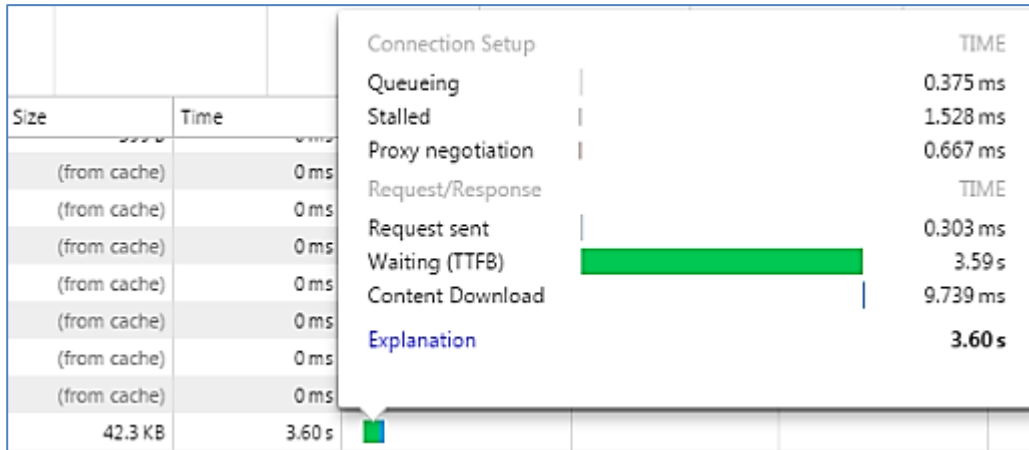
My Harp tool is a Java-based tool developed for offline post-analyzing HTTP traffic captured with Chrome Dev Tool (CDT) or FireBug with FireFox. Based on the HAR files saved with the CDT or FireBug, it parses the given input file in Har format, calculates some important metrics such as the server time, client time, network latency, as well as some other metrics such as the # of HTTP requests, page weight, maximum # of connections issued *concurrently* from the browser, and so on. At the end, it generates an HTML5-based file that can be opened with any browser for a timeline chart that illustrates the HTTP requests issued from the client to the server with all timings displayed on the chart. It's a very useful tool for accurately assessing where majority of the time spent (whether on client or server side) and which parts contribute most to the end-to-end response time of a page, etc.

The article at <https://developer.chrome.com/devtools/docs/network#resource-network-timing> explains more about CDT and the Har format. As an example, an HTTP response may return a timings element as follows:

```
"timings": {
  "blocked": 1.52799999341369,
  "dns": -1,
  "connect": -1,
  "send": 0.3029999788850499,
  "wait": 3586.530999979001,
  "receive": 9.739000117409432,
  "ssl": -1
},
"connection": "353",
"pageref": "page_2"
},
```

On the HTTP traffic chart, the above timings element would manifest as shown in Figure F.1, which shows a waiting of 3.59s (TTFB - time-to-first-byte). If you click on *Explanation* visible on the mouse-over popup shown in Figure F.1, you would get an explanation as shown in Figure F.2.

The CDT/Harp combination is a useful tool, as each large wait time displayed on the HTTP traffic timeline chart, whether online through CDT or offline through Harp, is a potential UI response time optimization opportunity. By the way, the Harp tool is preferred as you can save each Harp profile for comparisons over time.



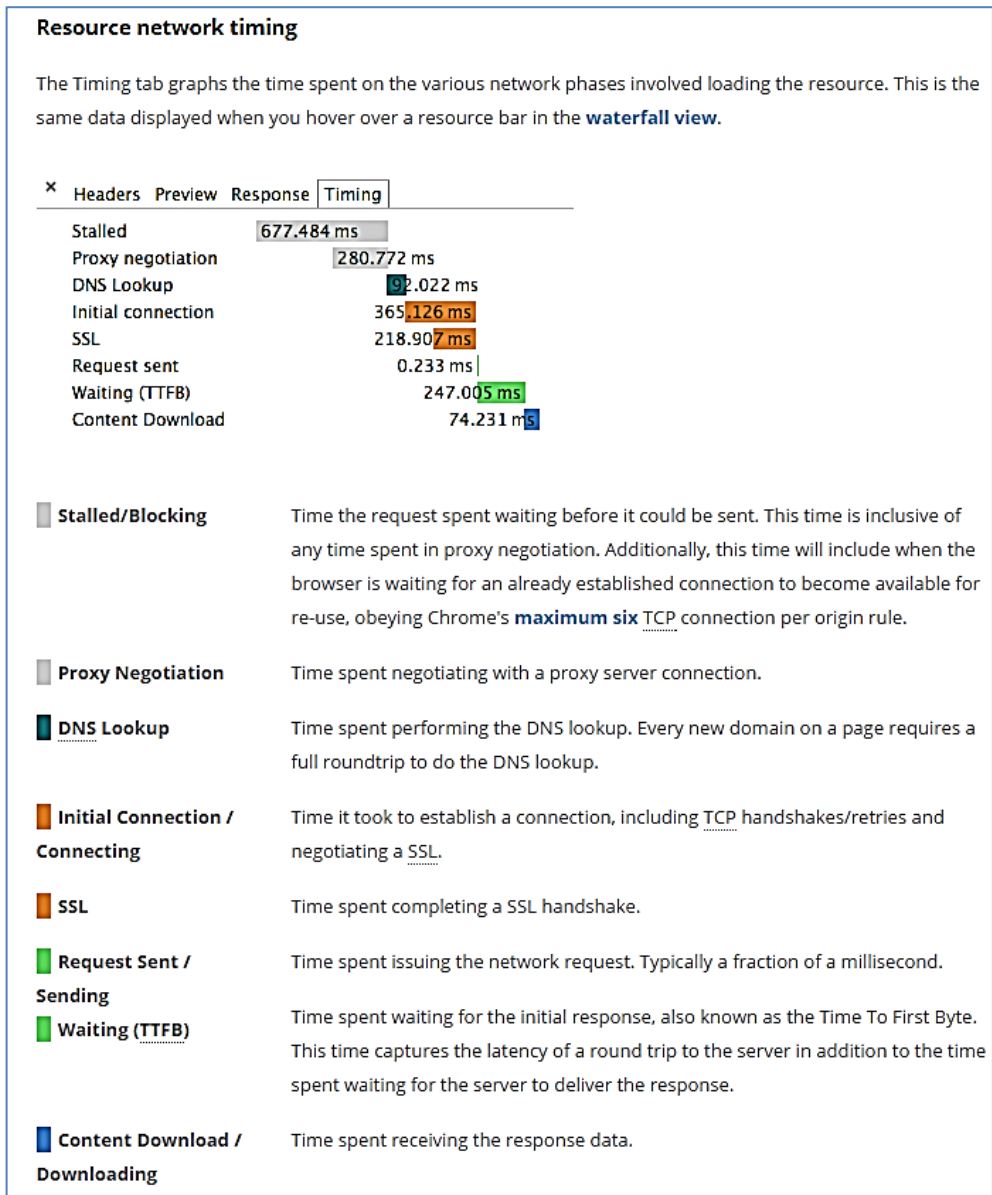
**Figure F.1** An HTTP request/response displayed on CDT

## F.1 SETTING UP A JDK FOR RUNNING THE HARP TOOL

Steps to get started with using Harp:

- 1 Since Harp processes Har files, it's very helpful to spend some time understanding the Har (HTTP archive) format. Although it's not required to fully understand the Har format, it's highly recommended to have a cursory look at the introduction to HAR posted online at <https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html> so that you would know immediately what you are dealing with.
- 2 The next step is to download the Harp tool. You can download it from this book's website at <http://www.perfmath.com>.
- 3 Make sure you have a JDK 7 installed on the machine you intend to run the tool. It may not work with Java 6 or earlier versions of JDKs. If you encounter any issues, please send me an email using my email address provided in this book.

Next, we describe how to save a Har file with CDT.

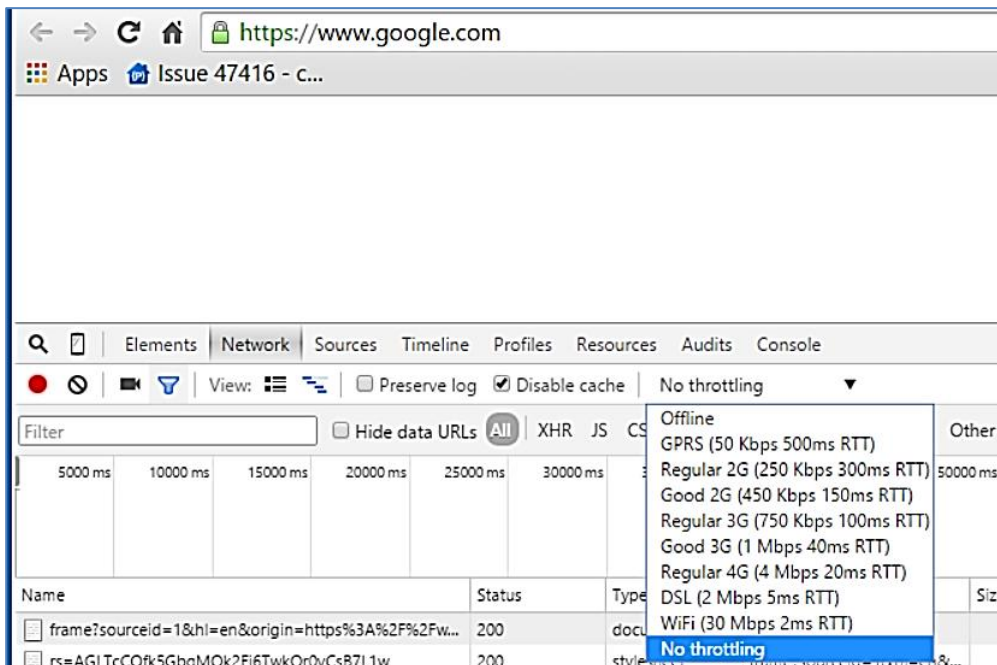


**Figure F.2** Explanations on resource network timing

## F.2 HOW TO SAVE NETWORK TRAFFIC CAPTURED WITH CDT IN HAR FORMAT

Of course, you are interested in this tool because you want to use it to help you analyze your own Har profiles. It's very important that you create your own Har files properly by following the below procedure:

- 1 After opening up your Chrome browser, press Ctrl+Shift+I to start up the CDT. Figure F.3 shows how CDT looks like with the Google home page opened with Chrome.
- 2 Verify that the *Network* tab is selected.
- 3 The left most solid dot icon controls starting/stopping recording. When it's red, it means "recording is active." Otherwise, it means "recording is inactive."
- 4 Make sure *Disable cache* is checked. I recommend disabling cache for repeatable results unless you are investigating effects of caching the page you are testing at the browser level.
- 5 You can simulate different network bandwidth by choosing a desired throttling setting as shown in Figure F.4. In general, I choose *No throttling* with my projects.
- 6 When you initiate an action on a UI, for example, entering the URL of a website or clicking a tab on a page, etc., you'll start seeing network traffic recorded in the content pane of CDT. After a page is completed, right click anywhere in the content pane and select "Save as HAR with content" as shown in Figure F.4. This is how a Har file is saved, and the Harp tool processes such Har files as described next.



**Figure F.3** CDT with Chrome



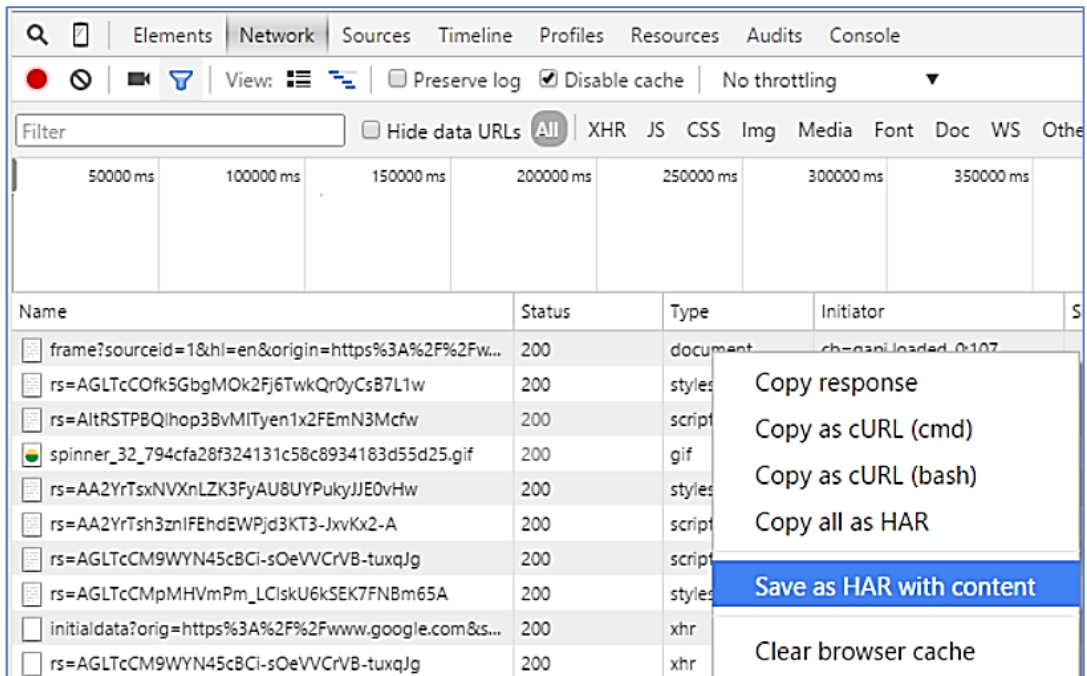


Figure F.4 Save As HAR with content option with CDT

### F.3 HOW TO RUN THE HARP TOOL

After downloading the tool, setting up your JDK, and saving a Har file, use the following procedure to run my Harp tool against your Har file:

- 1 Change to the folder where your Har file resides.
- 2 Put the following content in a .bat script in you are on Windows, for example, a file named run-harp.bat:

```
java -jar Harp2-1.0-SNAPSHOT-jar-with-dependencies.jar %1 %2 %3
```

- 3 Then issue the command “run-harp <your-har-file>” to process the designated Har file. The remaining two parameters are for `clickInterval` and `requestLimit`, which are for special cases. For example, if the total # of HTTP requests were 50, and the last request was not a part of the UI rendering (e.g., a log request), you can run the Harp tool with “run-harp <your-har-file> 10000 49” to prevent the last request from being counted into the total end-to-end time. Here, 10000 means 10000 milliseconds for distinguishing multiple clicks, but this is still an experimental feature and not used very often even by myself.

After conducting the above run, you should see an HTML file in the folder that is named the same as your Har file without the .har extension. Open up that HTML file in a browser, and you should see a chart similar to what was displayed on CDT, except that you will also see timing and data breakdowns,

etc. There are also some auxiliary files in that folder that can be used to get details about the HTTP requests/responses processed by the Harp tool.

In case you encounter any errors, please contact me, as the Harp tool may not deal with some Har files out of ordinary properly.

## **F.4 FEEDBACK**

Your feedback is very important for me to improve this tool over time. Please feel free to send me emails. Whatever feedback you give, it will be taken seriously.

## **F.5 BEST PRACTICES WITH USING THE HARP TOOL**

This section summarizes some known best practices with using this tool:

- 1 Always save a Har file per user transaction, which simply is a single click. The tool is developed for analyzing each individual user action, so recording multiple user transactions into one Har file may complicate the analysis of those user transactions.
- 2 As soon as you see a user transaction is completed, for example, you notice that the spinner has just stopped spinning, put down the # of requests displayed at the lower left corner of CDT and then save as fast as you can, as it's very likely that the browser may continue to record additional HTTP traffic that is not supposed to be part of the traffic initiated by a user action. You may want to stop recording before saving the network traffic into a Har file to avoid including un-related HTTP requests, such as some spurious alerts or logging requests.
- 3 If you noticed that some extra, spurious HTTP requests had been recorded at the end of a user action, you can instruct the program to exclude such requests, as described in §F.3 (step 3), by giving one extra command-line argument to specify exactly how many requests should be processed. The Harp tool actually ignores some known types of spurious HTTP traffic.
- 4 If you did not stop recording after saving your Har file, click the "Clear" icon (second from the left) before starting a new recording. This will give you a fresh new start for the next user action. This action is not required if you stopped and restarted recording, which automatically clears the previously recorded HTTP traffic.
- 5 If you see gaps between adjacent requests, most likely, caching is not implemented properly on the server side. With one of my projects, large gaps were observed with some user actions, and those large gaps disappeared after extension IDs of plug-ins on the server side were cached properly. You should conduct such gap analysis constantly with the UI pages you optimize.
- 6 If you observe that the same CSS files are downloaded multiple times from the server and each download gets longer and longer, it might be due to some gadgets having no sharing implemented, which can be re-considered to enhance sharing. With one of my projects, a UI page's end-to-end response time was reduced from 12 seconds to about 3 seconds after an issue like this was identified with Harp and cured with the above resolution.

Please share your experience in using Harp with me so that it can be improved over time.

# Index

32-bit computing, 28-29, 123  
64-bit computing 32, 106-107, 204

## A

AccessController, 134-136, 140-143  
Active cores, 38, 265  
Adaptive optimization, 65-66  
Address space 29, 31, 106  
Addressing modes, 29, 46  
AGP, 32, 36  
Algorithm  
    dynamic, 192-195, 197-198  
    recursive 97, 192-195, 197-199  
    static, 192-195, 199  
AMD64 106-108, 110, 123  
Analysis  
    database activity, 162, 166  
    JVM activity, 162, 164  
    thread pool, 162, 165-166  
Application architecture, 126  
Application profiling, 80, 130  
Architecture  
    accumulator, 51, 106  
    instruction set 29, 31-32, 41  
    load-store, 49  
    microarchitecture 29-33, 35-37, 41  
    processor hardware, 29, 31  
    register-memory, 49, 51  
    register-oriented, 49-50  
    register-register, 49, 51  
    stack-oriented, 49  
Array processing 10, 196-198, 200

Array size, 77, 158  
Arrival rate, 5, 8-9  
Artifact, 3, 45  
Authentication Server 10-12, 155, 161-162  
AVX, 32  
AWR 156, 161-162, 166  
AWT, 65, 184

## B

B language, 201  
Baseline 92, 137, 144  
Batch jobs, 16, 154  
Bell Labs, 201  
Bottleneck analysis 8, 195, 199-200  
BSD, 201-202  
Bus interface unit, 35  
Byte index, 50  
Bytecode, 43-51, 65-66

## C

C language  
C/C++ 47, 52, 66  
Cache 23, 25, 30-36  
    L1 30-32, 38, 110  
    L2 30-33, 36, 38  
    L3 32-33, 38, 221  
Caching, 25, 38  
Cat 15, 97-99, 210  
CentOS 95, 206-210, 232  
CFS, 217-218  
Circuit 3-5, 25-26, 256

- Class 44, 46-48, 50
- Class libraries, 44
- Class sharing, 88
- ClassLoader, 46, 165
- Closed model, 9
- CMS 66, 70-73, 76
- Code
  - native 43, 66, 109
- Code prefetch unit, 35
- Cold start-up, 66
- Command line options
  - developer, 74
  - HotSpot JVM, 74, 77
  - standard, 74
- Compatible 32, 106-107, 203
- Completely fair scheduling. See CFS
- Concurrency 11, 48, 65
- Concurrent 4-6, 11-12, 16-19
- Concurrent users 4-6, 11-12, 16-19
- Constant pool, 48
- Context switching, 66
- Core i7 12-13, 32-33, 35-40
- CPU benchmarking 28, 39, 41
- CPU usage 1, 13-14, 18-19
- CPU-bound, 153
- CPU-intensive 12, 126-127, 196
- CPUCheck 40-41, 78-79, 82-83
  - E5320 versus Core 2 Duo E8500, 40-41
  - NUM\_OF\_LEVELS, 92
  - runClient, 92, 268
  - runServer, 92, 268
  - service directory tree structure 92, 194, 196
  - services.properties 92, 97-98, 190
  - SUBSERVICES\_PER\_NODE, 92
- Cpuinfo, 219, 264
- Cyclic reference, 53-55
- D
  - Data type 29, 45, 47
  - Denominator, 27-28
  - Df, 225
  - Disk 1, 20, 23
  - DiskCheck, 126
  - DMI, 36, 224
  - Dmidecode, 222, 224
  - Drop\_caches, 25
- E
  - E5320 12-13, 40-41, 222
  - E5620 12-13, 16, 22
  - E7330, 12, 222
  - Echo, 25, 236
  - EM64T, 106-108, 123
  - EnableAudit, 127
  - Equilibrium condition 5, 9, 155
  - Exception handling, 29
  - Exec, 217
  - Execution unit, 34-35
- F
  - Feedback, 8-9, 217
  - File handle, 226-228
  - File-max, 227-229
  - Finalizing, 47
  - Fork, 217
  - Fragmentation 59-60, 71, 73
  - Frame 48, 60, 79
  - Free 23, 53, 59-61
  - FSB, 36-37, 258-259
- G
  - G1 66, 73, 162
  - Garbage collection
    - CMS 70, 76, 100-101
    - full 61-62, 89, 101
    - live object copying, 89
    - MarkSweep, 79
    - minor 62, 89, 145
    - minor GC, 89
    - monitoring 65, 77, 100
    - Old Generation 62, 75, 79
    - ParallelGC, 70, 79
    - patterns, 89
    - pause 53, 69-70, 75

- premature promotion, 89
  - promotion failure, 89
  - Survivor spaces overflowing, 89
  - throughput 68-70, 75-76, 79
  - tuning 44, 74-77, 100
  - Young Generation 62, 68, 70
- Garbage collection algorithm 51-54, 63, 65
- reference counting, 52-54
  - tracing, 54
  - vectorial, 54
- Garbage collector
- CMS 66, 73, 76
  - compacting 59, 78, 100
  - concurrent 66, 73, 76
  - copying 59, 68-69, 78
  - footprint, 43-44, 69
  - frequency, 69
  - G1 66, 73, 175-176
  - metrics, 69
  - non-compacting, 59
  - parallel 59, 66, 69
  - parallelOld, 100
  - pause 66, 69, 73
  - promptness, 69
  - serial 59, 69, 78
  - throughput 68-69, 73, 75-76
- Garbage collector. See GC
- Garbage object 52-53, 55, 57-59
- GC strategy
- compacting, 59-61
  - concurrent, 61-62
  - copying, 59-62, 69
  - generational, 62-63, 69
  - incremental, 62-63
  - non-compacting, 59-61
  - parallel 59, 61, 69
  - serial 59, 61, 69
  - stop-and-copy, 61-62
- Gcutil, 96, 99
- Generation
- Eden 68-69, 79, 89
  - old 62, 75, 79-81
  - permanent, 68, 109
  - survivor 68, 75, 79
  - tenured 68-73, 75, 78
  - young 62, 68-70, 72-73
- GNOME, 203
- ## H
- Hard 14-15, 33, 55
- Hardware specs, 126-127
- Heap 46-48, 52-53, 55
- Code Cache 79, 113-114, 116-117
  - Compile Time 89, 145, 164
  - current utilization, 89
  - currentCapacity, 89, 145
  - Eden 68-69, 76, 79
  - Eden space 89, 98, 111
  - GC Time 89, 95-96, 145
  - maxCapacity, 89, 145
  - memory usage, 79, 111
  - Old 75, 79, 89
  - Old Gen 75, 79, 89
  - old generation 75, 79, 89
  - Perform GC, 79, 83
  - Perm 68-69, 74, 76
  - Perm Gen 79, 89, 145
  - PermGen 76, 83, 109
  - S0 89, 96, 145
  - S1 69, 89, 96
  - sizing, 74-76, 101
  - survivor 68, 75-76, 79
  - tenured 68-69, 73, 75-76
  - tenured space 73, 111, 113-114
  - tuning 73-77, 131, 145
  - used space, 111, 114
  - young generation 68, 75, 78-79
- Hosking, 64, 101
- HotSpot JVM 2, 51, 63
- HTML, 162
- Hunt 60, 64, 89
- Hyper-thread, 32, 219
- Hyper-threaded 12, 14, 18
- Hyper-threading 14, 25-26, 32
- ## I

- I7 12-14, 22, 32-33
- I7-2600 12-13, 39-40, 78
- IA64, 107-108
- Iadd, 49-51
- IBM 43, 45, 205
- Idiv, 49
- Ifconfig, 225
- Iload, 49-51
- Imul, 49
- Inclusive shared L3 cache, 38
- Input/Output Hub. See IOH
- Instruction decoder unit, 35
- Instruction set 29, 31-32, 41
  - 0-operand, 51
  - 1-operand, 51
  - 2-operand, 51
  - n-operand, 50
- Instruction set architecture 29, 32, 41
- Instructions 29, 31-32, 35
- Integer 30, 49-51, 71
- Intel processors
  - 4004, 28
- Intel Processors
  - history, 28
- Intel processors
  - multicore, 28
  - Netburst, 32
  - x86, 32
  - x86-64, 32
- Interface 20, 29, 33-36
- Interrupt 23, 29, 202
- IO-bound, 23
- IP port range, 226-228
- IPL32
- Istore, 49-51
- Isub, 49
- Itanium, 106-108, 205

## J

### Java

- ergonomics 70, 74, 101
- language specification, 46
- reference type, 47, 109

- Java APIs, 44
- Java Enterprise Application Server. See JEAS
- Java performance and scalability on Linux 103, 201, 203
- Java quick starter, 66-68
- Java Virtual Machine Profiler Interface. See JVM PI
- Java Web Start, 66
- Java2D, 66
- JConsole 43, 65, 77-82
- JDBC tuning
  - max-pool-size, 158
  - min-pool-size, 157-158
- JDBC tunings, 153, 157-158
- JDK 43, 45, 65
- JEE Application Server 10-13, 16, 18-19
- JIT, 43-44, 65-66
- JNDIBasedSecurityManagement, 127
- Jones, 63-64, 101
- JProfiler, 132
- JRE 43-44, 46, 77
- JRockit, 45, 66
- JSP, 76
- JVM 2, 12, 44-49
  - 64-bit 12, 46, 48-49
    - activity analysis, 162, 164
    - exit, 47, 97
    - implementation 45-46, 48, 63
    - initializing, 46
    - linking, 46, 48
    - loading, 46, 164
    - specification 45-47, 51, 64
    - start-up 46, 51, 65-66
    - statistics 97, 148, 153
- JVM options
  - ParallelGCThreads, 77
  - UseCompressedOops, 77
- JVM Tuning 143, 148, 151

## K

- KDE, 203, 207
- Kernel 23, 134, 202-206
- Kernel parameters 226, 228, 252

## L

Latency 36, 71, 80  
Limits.conf, 218, 229  
Linear addresses, 35  
Linear scalability, 18  
Linux 2-3, 11, 15

- Debian, 205, 210
- Fedora, 205-206, 210
- file system caching, 25
- OpenSUSE 205-206, 210, 219
- RHEL, 205-206, 210
- Ubuntu, 205, 210

Linux versus Windows, 248  
List

- singly-linked, 53

LKM, 203  
Load average 231, 235, 240-241  
Load testing, 127, 154  
Locality, 60  
Lock contention 5, 85, 166  
Logical addresses, 35  
Long 3, 23, 25  
LP6, 107-108  
Lscpu, 219-221, 264

M

Marking 55-58, 70-71, 96  
MBean

- dynamic, 134
- standard, 134

Meminfo, 222-223  
Memory 1, 20, 23

- resident 25, 231, 235
- virtual 20, 23, 25

Memory bloat 105, 107-108, 123  
Memory leak 43, 76, 245  
Memory management 33, 43-44, 48  
Memory Management Unit. See MMU  
Memory structure, 29  
Memory-bound, 23, 153  
Method area, 48

## Microarchitecture

Core 31-33, 35-37, 41  
Nehalem 31-32, 36-37, 138  
Netburst, 30-32  
Sandy Bridge, 31-32  
Microkernel, 202-203, 206  
Minix, 202  
MMU, 34, 213  
Moss, 64, 101  
MQMS, 7-8  
Multicore 26, 28, 31  
Mutator, 57

## N

Nehalem 31-32, 36-37, 138  
Netstat, 225-226  
NIO, 65-67  
Node 9-10, 53-55, 82  
Nofile 228-230, 239, 248  
Notation

- infix, 49
- postfix, 49
- prefix, 49

Nproc 219, 228-230, 239  
NUMA 36, 215-216, 219-222  
NumberOfCores, 14  
Numerator, 27-28

## O

Object 1, 8, 20

- allocation 53, 59, 61
- de-allocation, 53, 109
- dead, 52
- garbage 43-44, 47-48, 52-55
- live 52, 57-58, 60-62
- reachable 54-55, 57, 59-62
- root 54-55, 57-58, 70

OLTP 6, 70, 114  
OOME, 46  
Oops 109-119, 122-123, 148

- compressed 109, 113-114, 116-117
- uncompressed 113, 116, 118-119

Opcode, 32, 47-51  
Open model, 9  
OpenJDK 45, 98, 209  
Operator, 109  
Opteron, 110  
OutOfMemoryErrors 76, 101, 118

## P

Page directory, 35  
Page frames, 213, 215  
Page reclaiming, 215  
Page table entries, 35  
Page tables 214, 243, 253  
Pages 76, 213, 215  
Paging unit, 35  
Parallelism, 31, 73  
Passing  
    by reference, 47  
    by value, 47  
PassMark 39-41, 138-139, 200  
Pattern  
    CPU 131, 145, 157  
    heap GC, 163-164  
Pause 53, 66, 69-70  
PCI, 32  
PCI-X, 32, 36  
PCIe, 32, 36-37  
Pcpu, 25, 235-236  
Perfmon 20-21, 162, 236  
Performance and scalability  
    anti-pattern, 198-199  
Performance and scalability issues 2, 5, 11  
Performance and scalability metrics, 3  
Performance counters 1, 3, 19-21  
Performance law 6-8, 10, 27  
Performance monitor snap-in, 20-21  
PGD, 214-215  
PMD, 214-215  
Pointer 43, 47-49, 52  
    managed, 48, 109  
    unmanaged, 109  
Pointers 43, 47, 49  
Power consumption, 37

Preemptive multitasking, 217  
Processes 23, 35, 78  
Profile  
    execution 66, 92, 135  
Profiling  
    self time, 86, 93  
Ps 23, 79, 235-238  
PSI-Probe 159-162, 165-166, 178-180  
PTE, 214-215

## Q

QPI 37, 139, 259  
Quantifying performance and scalability issues,  
    126  
Queuing node, 9  
Queuing theory 2, 5-6, 8  
Queuing workload model, 127, 155  
QuickPath Interconnect 32, 37, 139  
QuickPath Interconnect. See QPI

## R

RAID 0, 110  
Reachability, 57  
Reference 26, 39, 41-42  
    cyclic, 53-55  
Register 29, 31-32, 34  
Registers  
    AX, 106  
    BP, 106  
    BX, 106  
    control, 35, 105  
    CX, 106  
    DI, 106  
    DX, 29, 106  
    eax, 51, 107  
    ebp, 107  
    ebx, 107  
    ecx, 51, 107  
    edi, 107  
    edx, 51, 107  
    esi, 107  
    esp, 107



- 
- floating point, 105
  - integer 49, 51, 105-107
  - r10, 107
  - r11, 107
  - r12, 107
  - r13, 107
  - r14, 107
  - r15, 107
  - r8, 107
  - r9, 107
  - rax, 107
  - rbp, 107
  - rbx, 107
  - rcx, 107
  - rdi, 107
  - rdx, 107
  - rsi, 107
  - SI, 106
  - SP, 106
  - Residence time, 9-10
  - Resolution
    - late, 46
    - lazy, 46
    - static, 46
  - Resource utilization 5-6, 9-10, 19
  - Response time 4-7, 9-10, 12-13
  - Response time performance laws, 6-7
  - RMI 54, 182, 197
  - Root 54-58, 70, 125
  - Root set, 57-58
  - Rss 25, 235-236, 245-246
  - Runtime 41, 43-44, 46-48
  - Runtime data area, 47-48, 51
  - S
  - Sar 23, 234-235, 242-243
  - Scalability 28, 30, 32
  - Scalability analysis, 246
  - Scalability testing 3, 18, 26
  - Scaling-out, 7
  - Scaling-up, 7-8
  - See loadable kernel module. See LKM
  - Segment descriptor cache, 35
  - Segmentation unit, 35
  - Semaphore, 226-227
  - Semmi, 227-228
  - Semms, 227-228
  - Semmsl, 227-228
  - Semopm, 227-228
  - Service demand, 9-10, 25
  - Service time 6-7, 9-10, 25
  - ServiceDirectory 78, 82-83, 93
  - Shared memory 37, 215, 226-228
  - Shell limits, 228-229, 252
  - Shmall, 227-228
  - Shmmax, 227-228
  - Shmmni, 227-228
  - Silk Performer 11, 127, 133-134
  - Single core, 28, 37
  - Sipush, 50
  - Sizing guidelines, 126, 226
  - Sizing thread pools, 153, 159
  - Soft, 229-230, 239
  - Solaris 68, 101, 108
  - SQMS, 7-8
  - SQSS, 7-8
  - Stack 44, 47-51, 63
  - Stack trace, 100
  - StackOverflowError, 48
  - Standard C library, 210
  - StandardMBean, 134-135, 140-141
  - Start
    - cold, 66-68
    - warm, 66-68
  - Stop-the-world 53, 61-63, 70-71
  - Swapping, 215, 243
  - Sweeping 55-56, 71-72, 96
  - Swing 65, 183, 185
  - Symbol table, 48
  - Sync, 25
  - System
    - physical 12-14, 18, 20
    - virtual 2, 14, 20
  - System call, 211, 253
  - T

Task Manager 14-15, 21, 25  
TASK\_SIZE, 212  
Test environment 10-11, 126-127, 130  
Threads 32, 37-38, 48  
Throughput 4-6, 8-9, 16  
Throughput performance law, 8  
Time range, 21  
TLB, 215  
Top 25, 51, 57  
Translation lookaside buffer, 34-35, 215  
Tuning  
    AggressiveOpts 74, 162, 170-171  
    DoEscapeAnalysis 162, 170-171, 173  
    JDBC connection pool 157-159, 161-162, 167  
Turbo Boost Technology 38, 255, 257  
Turbo clock rate, 37-38  
Type  
    array 47, 197, 224  
    class 46-48, 52, 86  
    interface 20, 29, 46-48  
    numerical, 47  
    primitive, 47  
    reference 26, 41, 46-48  
    returnAddress, 47

## U

Ulimit, 229-230  
UMA versus NUMA, 215  
Uname, 218-219, 236  
UNIX 44-45, 48, 201-203  
Unloading, 47  
Use cases, 126, 153  
UseCompressedOops 77-78, 98, 105

## V

Variable table, 50  
Virtual memory 23, 25, 68  
Visual VM  
    snapshot, 82  
VisualGC 89, 92, 100-101  
    Graphs, 89

    Histogram, 89  
    Spaces 89, 145, 149  
VisualVM 43, 65, 77  
    Back Traces, 135, 143  
    Call Tree, 93-94  
    Classes 80, 83, 86  
    Combined, 93  
    CPU 80, 83, 85-87  
    CPU profiling 86-87, 92, 94  
    CPU sampling 86, 92-93, 132  
    Heap 77, 83, 89  
    heap dump, 83  
    Hot Spots, 93-94  
    instrumenting, 86  
    invocation, 86  
    jconsole 43, 65, 77  
    jinfo, 82  
    jstack, 82  
    jstat, 82, 209  
    Memory profiling 86, 88, 92  
    memory sampling, 86-87, 92  
    monitor 77, 82-84, 90  
    offline analysis 77, 93, 162  
    overhead 88, 90, 92-93  
    overview 80, 82-84, 93  
    Pause live results, 86  
    perform GC, 83  
    plugins, 89  
    Profiler 83, 86, 88  
    Sampler 83, 85-87, 92-93  
    sampling 86-87, 92-93, 100  
    self time 86, 93, 135  
    settings 86, 92-93, 131  
    snapshot 82, 86, 88  
    Threads 80, 83-85, 93  
    Time 43, 65, 77  
    Update live results, 86  
    VisualGC 89, 92, 100  
VM slice, 12  
Vmstat 22-25, 162, 233-234  
Vsz 25, 235-236, 245-246

## W

Web Server 8, 10, 16  
Windows 2-3, 11, 14  
Wmic, 14  
Word, 47  
Working set 20, 57-58, 69  
Workload 12, 73, 76

## X

X86 29, 31-32, 51  
Xeon  
    5550, 240  
    E5620 12-13, 16, 39-40



Other texts by the same author, available on Amazon:

